<table>
<thead>
<tr>
<th></th>
<th>Interfaces</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>51</td>
<td>interfaces.afni</td>
<td>407</td>
</tr>
<tr>
<td>52</td>
<td>interfaces.ants</td>
<td>547</td>
</tr>
<tr>
<td>53</td>
<td>interfaces.base</td>
<td>603</td>
</tr>
<tr>
<td>54</td>
<td>interfaces.brainsuite</td>
<td>609</td>
</tr>
<tr>
<td>55</td>
<td>interfaces.camino</td>
<td>639</td>
</tr>
<tr>
<td>56</td>
<td>interfaces.camino2trackvis</td>
<td>701</td>
</tr>
<tr>
<td>57</td>
<td>interfaces.cmtk</td>
<td>705</td>
</tr>
<tr>
<td>58</td>
<td>interfaces.diffusion_toolkit</td>
<td>719</td>
</tr>
<tr>
<td>59</td>
<td>interfaces.dipy</td>
<td>731</td>
</tr>
<tr>
<td>60</td>
<td>interfaces.dt itk</td>
<td>743</td>
</tr>
<tr>
<td>61</td>
<td>interfaces.elastix</td>
<td>771</td>
</tr>
<tr>
<td>62</td>
<td>interfaces.freesurfer</td>
<td>777</td>
</tr>
<tr>
<td>63</td>
<td>interfaces.fsl</td>
<td>911</td>
</tr>
<tr>
<td>64</td>
<td>interfaces.minc</td>
<td>1065</td>
</tr>
<tr>
<td>65</td>
<td>interfaces.mipav</td>
<td>1113</td>
</tr>
<tr>
<td>66</td>
<td>interfaces.mixins</td>
<td>1141</td>
</tr>
<tr>
<td>67</td>
<td>interfaces.mne</td>
<td>1143</td>
</tr>
<tr>
<td>68</td>
<td>interfaces.mrtrix</td>
<td>1145</td>
</tr>
<tr>
<td>69</td>
<td>interfaces.mrtrix3</td>
<td>1179</td>
</tr>
<tr>
<td>70</td>
<td>interfaces.niftyfit</td>
<td>1211</td>
</tr>
<tr>
<td>71</td>
<td>interfaces.niftyreg</td>
<td>1225</td>
</tr>
<tr>
<td>72</td>
<td>interfaces.niftyseg</td>
<td>1241</td>
</tr>
<tr>
<td>73</td>
<td>interfaces.nipy</td>
<td>1263</td>
</tr>
<tr>
<td>74</td>
<td>interfaces.nitime</td>
<td>1269</td>
</tr>
<tr>
<td>75</td>
<td>interfaces.semtools</td>
<td>1271</td>
</tr>
<tr>
<td>76</td>
<td>interfaces.slicer</td>
<td>1401</td>
</tr>
<tr>
<td>77</td>
<td>interfaces.spm</td>
<td>1487</td>
</tr>
<tr>
<td>78</td>
<td>interfaces.utility</td>
<td>1531</td>
</tr>
<tr>
<td>79</td>
<td>interfaces.vista</td>
<td>1537</td>
</tr>
<tr>
<td>80</td>
<td>interfaces.workbench</td>
<td>1539</td>
</tr>
</tbody>
</table>
Previous versions: 1.2.1 1.2.0

Michael Notter’s Nipype guide

Be sure to read Michael’s excellent tutorials.

Interfaces, Workflows and Examples

• Workflows
CHAPTER 1

workflows.data

1.1 get_flirt_schedule()

Link to code
2.1 workflows.dmri.camino.connectivity_mapping

2.1.1 create_connectivity_pipeline()

Link to code

Creates a pipeline that does the same connectivity processing as in the dMRI: Connectivity - Camino, CMTK, FreeSurfer example script. Given a subject id (and completed Freesurfer reconstruction) diffusion-weighted image, b-values, and b-vectors, the workflow will return the subject’s connectome as a Connectome File Format (CFF) file for use in Connectome Viewer (http://www.cmtk.org).

Example

```python
>>> from nipype.workflows.dmri.camino.connectivity_mapping import create_connectivity_pipeline
>>> conmapper = create_connectivity_pipeline("nipype_conmap")
>>> conmapper.inputs.inputnode.subjects_dir = '.'
>>> conmapper.inputs.inputnode.subject_id = 'sub1'
>>> conmapper.inputs.inputnode.dwi = 'data.nii.gz'
>>> conmapper.inputs.inputnode.bvecs = 'bvecs'
>>> conmapper.inputs.inputnode.bvals = 'bvals'
>>> conmapper.run() # doctest: +SKIP
```

Inputs:

- inputnode.subject_id
- inputnode.subjects_dir
- inputnode.dwi
- inputnode.bvecs
- inputnode.bvals
- inputnode.resolution_network_file

Outputs:

- outputnode.connectome
- outputnode.cmatrix
- outputnode.gpickle_network

(continues on next page)
Graph

2.2 workflows.dmri.camino.diffusion

2.2.1 create_camino_dti_pipeline()

Link to code

Creates a pipeline that does the same diffusion processing as in the ../../users/examples/dmri_camino_dti example script. Given a diffusion-weighted image, b-values, and b-vectors, the workflow will return the tractography computed from diffusion tensors and from PICo probabilistic tractography.

Example

```python
>>> import os

>>> nipype_camino_dti = create_camino_dti_pipeline("nipype_camino_dti")
>>> nipype_camino_dti.inputs.inputnode.dwi = os.path.abspath('dwi.nii')
>>> nipype_camino_dti.inputs.inputnode.bvecs = os.path.abspath('bvecs')
>>> nipype_camino_dti.inputs.inputnode.bvals = os.path.abspath('bvals')
>>> nipype_camino_dti.run()                 # doctest: +SKIP
```

Inputs:

- inputnode.dwi
- inputnode.bvecs
- inputnode.bvals

Outputs:

- outputnode.fa
- outputnode.trace
- outputnode.tracts_pico
- outputnode.tracts_dt
- outputnode.tensors
2.3 workflows.dmri.camino.group_connectivity

2.3.1 create_group_connectivity_pipeline()

Link to code

Creates a pipeline that performs basic Camino structural connectivity processing on groups of subjects. Given a diffusion-weighted image, and text files containing the associated b-values and b-vectors, the workflow will return each subjects’ connectomes in a Connectome File Format (CFF) file, for use in Connectome Viewer (http://www.cmtk.org).

Example

```python
>>> import nipype.interfaces.freesurfer as fs
>>> import nipype.workflows.dmri.camino.group_connectivity as groupwork
>>> subjects_dir = '.
>>> data_dir = '.
>>> output_dir = '.
>>> fs.FSCommand.set_default_subjects_dir(subjects_dir)
>>> group_list = {}
>>> group_list['group1'] = [subject1, subject2]
>>> group_list['group2'] = [subject3, subject4]
>>> template_args = dict(dwi=[[subject_id, 'dwi']], bvecs=[[subject_id, 'bvecs ->']], bvals=[[subject_id, 'bvals']])
>>> group_id = 'group1'
>>> l1pipeline = groupwork.create_group_connectivity_pipeline(group_list, group_id, data_dir, subjects_dir, output_dir, template_args)
>>> l1pipeline.run() # doctest: +SKIP
```

Inputs:
group_list: Dictionary of subject lists, keyed by group name
group_id: String containing the group name
data_dir: Path to the data directory
subjects_dir: Path to the Freesurfer 'subjects' directory
output_dir: Path for the output files
template_args_dict: Dictionary of template arguments for the connectivity pipeline datasource

```python
e.g. info = dict(dwi=[['subject_id', 'dwi']],
bvecs=[['subject_id', 'bvecs']],
bvals=[['subject_id', 'bvals']])
```

### 2.4 workflows.dmri.connectivity.group_connectivity

#### 2.4.1 `concatcsv()`

Link to code

This function will concatenate two “comma-separated value” text files, but remove the first row (usually column headers) from all but the first file.

#### 2.4.2 `create_average_networks_by_group_workflow()`

Link to code

Creates a fourth-level pipeline to average the networks for two groups and merge them into a single CFF file. This pipeline will also output the average networks in .gexf format, for visualization in other graph viewers, such as Gephi.

Example

```python
>>> import nipype.workflows.dmri.connectivity.group_connectivity as groupwork
>>> from nipype.testing import example_data
>>> subjects_dir = '.
>>> data_dir = '.
>>> output_dir = '.
>>> group_list = {}
>>> group_list['group1'] = ['subj1', 'subj2']
>>> group_list['group2'] = ['subj3', 'subj4']
>>> l4pipeline = groupwork.create_average_networks_by_group_workflow(group_list,
... data_dir, subjects_dir, output_dir)
>>> l4pipeline.run() # doctest: +SKIP
```

Inputs:

- `group_list`: Dictionary of subject lists, keyed by group name
- `data_dir`: Path to the data directory
- `subjects_dir`: Path to the Freesurfer 'subjects' directory
- `output_dir`: Path for the output files
- `title`: String to use as a title for the output merged CFF file (default 'group')

#### 2.4.3 `create_merge_group_network_results_workflow()`

Link to code

Creates a third-level pipeline to merge the Connectome File Format (CFF) outputs from each group and combines them into a single CFF file for each group. This version of the third-level pipeline also concatenates the comma-separated value files for the NetworkX metrics and the connectivity matrices into single files.
Example

```python
>>> import nipype.workflows.dmri.connectivity.group_connectivity as groupwork
>>> from nipype.testing import example_data
>>> subjects_dir = '.
>>> data_dir = '.'
>>> output_dir = '.'
>>> group_list = {}
>>> group_list['group1'] = ['subj1', 'subj2']
>>> group_list['group2'] = ['subj3', 'subj4']
>>> l3pipeline = groupwork.create_merge_group_networks_workflow(group_list, data_dir, subjects_dir, output_dir)
>>> l3pipeline.run()  # doctest: +SKIP
```

Inputs:

- `group_list`: Dictionary of subject lists, keyed by group name
- `data_dir`: Path to the data directory
- `subjects_dir`: Path to the Freesurfer 'subjects' directory
- `output_dir`: Path for the output files
- `title`: String to use as a title for the output merged CFF file (default 'group')

### 2.4.4 `create_merge_group_networks_workflow()`

**Link to code**

Creates a third-level pipeline to merge the Connectome File Format (CFF) outputs from each group and combines them into a single CFF file for each group.

Example

```python
>>> import nipype.workflows.dmri.connectivity.group_connectivity as groupwork
>>> from nipype.testing import example_data
>>> subjects_dir = '.
>>> data_dir = '.'
>>> output_dir = '.'
>>> group_list = {}
>>> group_list['group1'] = ['subj1', 'subj2']
>>> group_list['group2'] = ['subj3', 'subj4']
>>> l3pipeline = groupwork.create_merge_group_networks_workflow(group_list, data_dir, subjects_dir, output_dir)
>>> l3pipeline.run()  # doctest: +SKIP
```

Inputs:

- `group_list`: Dictionary of subject lists, keyed by group name
- `data_dir`: Path to the data directory
- `subjects_dir`: Path to the Freesurfer 'subjects' directory
- `output_dir`: Path for the output files
- `title`: String to use as a title for the output merged CFF file (default 'group')

### 2.4.5 `create_merge_network_results_by_group_workflow()`

**Link to code**

Creates a second-level pipeline to merge the Connectome File Format (CFF) outputs from the group-level MR-trix structural connectivity processing pipeline into a single CFF file for each group.
Example

```python
>>> import nipype.workflows.dmri.connectivity.group_connectivity as groupwork
>>> from nipype.testing import example_data
>>> subjects_dir = '.
>>> data_dir = '.
>>> output_dir = '.
>>> group_list = {}
>>> group_list['group1'] = ['subj1', 'subj2']
>>> group_list['group2'] = ['subj3', 'subj4']
>>> group_id = 'group1'
>>> l2pipeline = groupwork.create_merge_network_results_by_group_workflow(group_list, group_id, data_dir, subjects_dir, output_dir)
>>> l2pipeline.run() # doctest: +SKIP
```

Inputs:

- `group_list`: Dictionary of subject lists, keyed by group name
- `group_id`: String containing the group name
- `data_dir`: Path to the data directory
- `subjects_dir`: Path to the Freesurfer 'subjects' directory
- `output_dir`: Path for the output files

### 2.4.6 create_merge_networks_by_group_workflow()

**Link to code**

Creates a second-level pipeline to merge the Connectome File Format (CFF) outputs from the group-level MR-trix structural connectivity processing pipeline into a single CFF file for each group.

**Example**

```python
>>> import nipype.workflows.dmri.connectivity.group_connectivity as groupwork
>>> from nipype.testing import example_data
>>> subjects_dir = '.
>>> data_dir = '.
>>> output_dir = '.
>>> group_list = {}
>>> group_list['group1'] = ['subj1', 'subj2']
>>> group_list['group2'] = ['subj3', 'subj4']
>>> group_id = 'group1'
>>> l2pipeline = groupwork.create_merge_networks_by_group_workflow(group_list, group_id, data_dir, subjects_dir, output_dir)
>>> l2pipeline.run() # doctest: +SKIP
```

**Inputs:**

- `group_list`: Dictionary of subject lists, keyed by group name
- `group_id`: String containing the group name
- `data_dir`: Path to the data directory
- `subjects_dir`: Path to the Freesurfer 'subjects' directory
- `output_dir`: Path for the output files

### 2.4.7 pullnodeIDs()

**Link to code**

This function will return the values contained, for each node in a network, given an input key. By default it will return the node names.
2.5 workflows.dmri.connectivity.nx

2.5.1 create_cmats_to_csv_pipeline()

Link to code
Creates a workflow to convert the outputs from CreateMatrix into a single comma-separated value text file. An extra column / field is also added to the text file. Typically, the user would connect the subject name to this field.

Example

```python
>>> from nipype.workflows.dmri.connectivity.nx import create_cmats_to_csv_pipeline
>>> csv = create_cmats_to_csv_pipeline("cmats_to_csv", "subject_id")
>>> csv.inputs.inputnode.extra_field = 'subj1'
>>> csv.inputs.inputnode.matlab_matrix_files = ['subj1_cmatrix.mat', 'subj1_mean_fiber_length.mat', 'subj1_median_fiber_length.mat', 'subj1_fiber_length_std.mat']
>>> csv.run()  # doctest: +SKIP
```

Inputs:

- inputnode.extra_field
- inputnode.matlab_matrix_files

Outputs:

- outputnode.csv_file
2.5.2 create_networkx_pipeline()

Link to code
Creates a workflow to calculate various graph measures (via NetworkX) on an input network. The output measures are then converted to comma-separated value text files, and an extra column / field is also added. Typically, the user would connect the subject name to this field.

Example

```python
>>> from nipype.workflows.dmri.connectivity.nx import create_networkx_pipeline
>>> nx = create_networkx_pipeline("networkx", "subject_id")
>>> nx.inputs.inputnode.extra_field = 'subj1'
>>> nx.inputs.inputnode.network_file = 'subj1.pck'
>>> nx.run()   # doctest: +SKIP
```

Inputs:

- inputnode.extra_field
- inputnode.network_file

Outputs:
2.5.3 add_global_to_filename()
Link to code

2.5.4 add_nodal_to_filename()
Link to code

2.6 workflows.dmri.dipy.denoise

2.6.1 nlmeans_pipeline()
Link to code
Workflow that performs nlmeans denoising

Example

```python
>>> from nipype.workflows.dmri.dipy.denoise import nlmeans_pipeline
>>> denoise = nlmeans_pipeline()
>>> denoise.inputs.inputnode.in_file = 'diffusion.nii'
>>> denoise.inputs.inputnode.in_mask = 'mask.nii'
>>> denoise.run() # doctest: +SKIP
```
**Graph**

```
inputnode (utility)

NoiseMsk (utility)

NLMeans (dipy)

outputnode (utility)
```

Denoise

---

### 2.6.2 bg_mask()

[Link to code](#)

Rough mask of background from brain masks

### 2.6.3 csf_mask()

[Link to code](#)

Artesanal mask of csf in T2w-like images

---

### 2.7 workflows.dmri.dtitk.tensor_registration

#### 2.7.1 affine_tensor_pipeline()

[Link to code](#)

Workflow that performs a linear registration (Rigid followed by Affine)

---

**Example**
>>> from nipype.workflows.dmri.dtitk.tensor_registration import affine_tensor_pipeline
>>> affine = affine_tensor_pipeline()
>>> affine.inputs.inputnode.fixed_file = 'im1.nii'
>>> affine.inputs.inputnode.moving_file = 'im2.nii'
>>> affine.run() # doctest: +SKIP

Graph

```
inputnode (utility)

rigid_node (dtitk)

affine_node (dtitk)

outputnode (utility)

AffTen
```

### 2.7.2 diffeomorphic_tensor_pipeline()

**Link to code**

Workflow that performs a diffeomorphic registration (Rigid and Affine followed by Diffeomorphic) Note: the requirements for a diffeomorphic registration specify that the dimension 0 is a power of 2 so images are resliced prior to registration. Remember to move origin and reslice prior to applying xfm to another file!

**Example**

```python
>>> from nipype.workflows.dmri.dtitk.tensor_registration import diffeomorphic_tensor_pipeline
>>> diffeo = diffeomorphic_tensor_pipeline()
>>> diffeo.inputs.inputnode.fixed_file = 'im1.nii'
```
>>> diffeo.inputs.inputnode.moving_file = 'im2.nii'
>>> diffeo.run()  # doctest: +SKIP

Graph

2.8 workflows.dmri.fsl.artifacts

2.8.1 all_fmb_pipeline()

Link to code
Builds a pipeline including three artifact corrections: head-motion correction (HMC), susceptibility-derived distortion correction (SDC), and Eddy currents-derived distortion correction (ECC).

The displacement fields from each kind of distortions are combined. Thus, only one interpolation occurs between input data and result.

**Warning:** this workflow rotates the gradients table ($b$-vectors) [Leemans09].

**Examples**

```python
>>> from nipype.workflows.dmri.fsl.artifacts import all_fmb_pipeline
>>> allcorr = all_fmb_pipeline()
>>> allcorr.inputs.inputnode.in_file = 'epi.nii'
>>> allcorr.inputs.inputnode.in_bval = 'diffusion.bval'
>>> allcorr.inputs.inputnode.in_bvec = 'diffusion.bvec'
>>> allcorr.inputs.inputnode.bmap_mag = 'magnitude.nii'
>>> allcorr.inputs.inputnode.bmap_pha = 'phase.nii'
>>> allcorr.inputs.inputnode.epi_param = 'epi_param.txt'
>>> allcorr.run() # doctest: +SKIP
```
2.8.2 all_fsl_pipeline()

Link to code
Workflow that integrates FSL topup and eddy.

**Warning:** this workflow rotates the gradients table ($b$-vectors) [Leemans09].

**Warning:** this workflow does not perform jacobian modulation of each $DWI$ [Jones10].

Examples

```python
>>> from nipype.workflows.dmri.fsl.artifacts import all_fsl_pipeline
>>> allcorr = all_fsl_pipeline()
>>> allcorr.inputs.inputnode.in_file = 'epi.nii'
>>> allcorr.inputs.inputnode.alt_file = 'epi_rev.nii'
>>> allcorr.inputs.inputnode.in_bval = 'diffusion.bval'
>>> allcorr.inputs.inputnode.in_bvec = 'diffusion.bvec'
>>> allcorr.run() # doctest: +SKIP
```
2.8.3 all_peb_pipeline()

**Link to code**
Builds a pipeline including three artifact corrections: head-motion correction (HMC), susceptibility-derived distortion correction (SDC), and Eddy currents-derived distortion correction (ECC).

**Warning:** this workflow rotates the gradients table ($b$-vectors) [Leemans09].

**Examples**

```python
>>> from nipype.workflows.dmri.fsl.artifacts import all_peb_pipeline
>>> allcorr = all_peb_pipeline()
>>> allcorr.inputs.inputnode.in_file = 'epi.nii'
>>> allcorr.inputs.inputnode.alt_file = 'epi_rev.nii'
```
>>> allcorr.inputs.inputnode.in_bval = 'diffusion.bval'
>>> allcorr.inputs.inputnode.in_bvec = 'diffusion.bvec'
>>> allcorr.run()  # doctest: +SKIP
2.8.4 ecc_pipeline()

Link to code
ECC stands for Eddy currents correction.
Creates a pipeline that corrects for artifacts induced by Eddy currents in dMRI sequences. It takes a series of diffusion weighted images and linearly co-registers them to one reference image (the average of all b0s in the dataset).
DWIs are also modulated by the determinant of the Jacobian as indicated by [Jones10] and [Rohde04].
A list of rigid transformation matrices can be provided, sourcing from a hmc_pipeline() workflow, to initialize registrations in a motion free framework.
A list of affine transformation matrices is available as output, so that transforms can be chained (discussion here).

References

Example

```python
>>> from nipype.workflows.dmri.fsl.artifacts import ecc_pipeline
>>> ecc = ecc_pipeline()
>>> ecc.inputs.inputnode.in_file = 'diffusion.nii'
>>> ecc.inputs.inputnode.in_bval = 'diffusion.bval'
>>> ecc.inputs.inputnode.in_mask = 'mask.nii'
>>> ecc.run() # doctest: +SKIP
```

Inputs:
- inputnode.in_file - input dwi file
- inputnode.in_mask - weights mask of reference image (a file with data range sin →[0.0, 1.0], indicating the weight of each voxel when computing the metric.
- inputnode.in_bval - b-values table
- inputnode.in_xfms - list of matrices to initialize registration (from head-motion →correction)

Outputs:
- outputnode.out_file - corrected dwi file
- outputnode.out_xfms - list of transformation matrices
Graph
2.8.5 hmc_pipeline()

Link to code

HMC stands for head-motion correction. Creates a pipeline that corrects for head motion artifacts in dMRI sequences. It takes a series of diffusion weighted images and rigidly co-registers them to one reference image. Finally, the $b$-matrix is rotated accordingly [Leemans09] making use of the rotation matrix obtained by FLIRT. Search angles have been limited to 4 degrees, based on results in [Yendiki13]. A list of rigid transformation matrices is provided, so that transforms can be chained. This is useful to correct for artifacts with only one interpolation process (as previously discussed here), and also to compute nuisance regressors as proposed by [Yendiki13].

**Warning:** This workflow rotates the $b$-vectors, so please be advised that not all the dicom converters ensure the consistency between the resulting nifti orientation and the gradients table (e.g. dcm2nii checks it).

References

Example

```python
>>> from nipype.workflows.dmri.fsl.artifacts import hmc_pipeline
>>> hmc = hmc_pipeline()
>>> hmc.inputs.inputnode.in_file = 'diffusion.nii'
>>> hmc.inputs.inputnode.in_bvec = 'diffusion.bvec'
>>> hmc.inputs.inputnode.in_bval = 'diffusion.bval'
>>> hmc.inputs.inputnode.in_mask = 'mask.nii'
>>> hmc.run()  # doctest: +SKIP
```

Inputs:

- **inputnode.in_file** - input dwi file
- **inputnode.in_mask** - weights mask of reference image (a file with data range in \([0. \rightarrow 0, 1.0]\), indicating the weight of each voxel when computing the metric.
- **inputnode.in_bval** - b-values file
- **inputnode.in_bvec** - gradients file (b-vectors)
- **inputnode.ref_num** (optional, default=0) index of the b0 volume that should be taken as reference

Outputs:

- **outputnode.out_file** - corrected dwi file
- **outputnode.out_bvec** - rotated gradient vectors table
- **outputnode.out_xfms** - list of transformation matrices
2.8.6 **remove_bias()**

Link to code

This workflow estimates a single multiplicative bias field from the averaged $b0$ image, as suggested in
Example

```python
>>> from nipype.workflows.dmri.fsl.artifacts import remove_bias
>>> bias = remove_bias()
>>> bias.inputs.inputnode.in_file = 'epi.nii'
>>> bias.inputs.inputnode.in_bval = 'diffusion.bval'
>>> bias.inputs.inputnode.in_mask = 'mask.nii'
>>> bias.run()  # doctest: +SKIP
```
Graph

2.8.7 sdc_fmb()

Link to code
SDC stands for susceptibility distortion correction. FMB stands for fieldmap-based. The fieldmap based (FMB) method implements SDC by using a mapping of the B0 field as proposed by [Jezzard95]. This workflow uses the implementation of FSL (FUGUE). Phase unwrapping is performed using PRELUDE [Jenkinson03]. Preparation of the fieldmap is performed reproducing the script in FSL fsl_prepare_fieldmap.

Example

```python
>>> from nipype.workflows.dmri.fsl.artifacts import sdc_fmb
>>> fmb = sdc_fmb()
>>> fmb.inputs.inputnode.in_file = 'diffusion.nii'
>>> fmb.inputs.inputnode.in_ref = list(range(0, 30, 6))
>>> fmb.inputs.inputnode.in_mask = 'mask.nii'
>>> fmb.inputs.inputnode.bmap_mag = 'magnitude.nii'
>>> fmb.inputs.inputnode.bmap_pha = 'phase.nii'
>>> fmb.inputs.inputnode.settings = 'epi_param.txt'
>>> fmb.run()  # doctest: +SKIP
```

**Warning:** Only SIEMENS format fieldmaps are supported.

References
Graph
2.8.8 sdc_peb()

Link to code

SDC stands for susceptibility distortion correction. PEB stands for phase-encoding-based. The phase-encoding-based (PEB) method implements SDC by acquiring diffusion images with two different encoding directions [Andersson2003]. The most typical case is acquiring with opposed phase-gradient blips (e.g. A>>>P and P>>>A, or equivalently, \(-y\) and \(y\)) as in [Chiou2000], but it is also possible to use orthogonal configurations [Cordes2000] (e.g. \(A>>>P\) and \(L>>>R\), or equivalently \(-y\) and \(x\)). This workflow uses the implementation of FSL (TOPUP).

Example

```python
>>> from nipype.workflows.dmri.fsl.artifacts import sdc_peb
>>> peb = sdc_peb()
>>> peb.inputs.inputnode.in_file = 'epi.nii'
>>> peb.inputs.inputnode.alt_file = 'epi_rev.nii'
>>> peb.inputs.inputnode.in_bval = 'diffusion.bval'
>>> peb.inputs.inputnode.in_mask = 'mask.nii'
>>> peb.run()  # doctest: +SKIP
```

References
Graph

inputnode (utility)

b0_ref (fsl)  b0_alt (fsl)

b0_list (utility)

b0_merged (fsl)

topup (fsl)

Shiftmap2Warping

inputnode (utility)

unwarp (fsl)

Fix_hdr (utility)

ScaleField (fsl)

vsm2dfm (fsl)
2.9 workflows.dmri.fsl.dti

2.9.1 bedpostx_parallel()

Link to code

Does the same as create_bedpostx_pipeline() by splitting the input dMRI in small ROIs that are better suited for parallel processing.

Example

```python
>>> from nipype.workflows.dmri.fsl.dti import bedpostx_parallel
>>> params = dict(n_fibres = 2, fudge = 1, burn_in = 1000,
... n_jumps = 1250, sample_every = 25)
>>> bpwf = bedpostx_parallel('nipype_bedpostx_parallel', params=params)
>>> bpwf.inputs.inputnode.dwi = 'diffusion.nii'
>>> bpwf.inputs.inputnode.mask = 'mask.nii'
>>> bpwf.inputs.inputnode.bvecs = 'bvecs'
>>> bpwf.inputs.inputnode.bvals = 'bvals'
>>> bpwf.run(plugin='CondorDAGMan') # doctest: +SKIP

Inputs:

inputnode.dwi
inputnode.mask
inputnode.bvecs
inputnode.bvals

Outputs:

outputnode wraps all XFibres outputs
2.9.2 `create_bedpostx_pipeline()`

Link to code

Creates a pipeline that does the same as bedpostx script from FSL - calculates diffusion model parameters (distributions not MLE) voxelwise for the whole volume (by splitting it slicewise).

Example

```python
>>> from nipype.workflows.dmri.fsl.dti import create_bedpostx_pipeline
>>> params = dict(n_fibres = 2, fudge = 1, burn_in = 1000,
...    n_jumps = 1250, sample_every = 25)
>>> bpwf = create_bedpostx_pipeline('nipype_bedpostx', params)
>>> bpwf.inputs.inputnode.dwi = 'diffusion.nii'
>>> bpwf.inputs.inputnode.mask = 'mask.nii'
>>> bpwf.inputs.inputnode.bvecs = 'bvecs'
>>> bpwf.inputs.inputnode.bvals = 'bvals'
>>> bpwf.run() # doctest: +SKIP
```

Inputs:

- `inputnode.dwi`
- `inputnode.mask`
- `inputnode.bvecs`
- `inputnode.bvals`

Outputs:

- `outputnode` wraps all XFibres outputs
2.9.3 merge_and_mean()

Link to code
Graph

inputnode (utility)

Merge (fsl)

Mean (fsl)

outputnode (utility)

mm

2.9.4 merge_and_mean_parallel()

Link to code
Graph

2.9.5 transpose()

Link to code

2.10 workflows.dmri.fsl.epi

2.10.1 create_dmri_preprocessing()

Link to code
Creates a workflow that chains the necessary pipelines to correct for motion, eddy currents, and, if selected, susceptibility artifacts in EPI dMRI sequences.

Deprecated since version 0.9.3: Use nipype.workflows.dmri.preprocess.epi.all_fmb_pipeline() or nipype.workflows.dmri.preprocess.epi.all_peb_pipeline() instead.

Warning: This workflow rotates the b-vectors, so please be advised that not all the dicom converters ensure the consistency between the resulting nifti orientation and the b matrix table (e.g. dcm2nii checks it).
Example

>>> nipype_dmri_preprocess = create_dmri_preprocessing('nipype_dmri_prep')
>>> nipype_dmri_preprocess.inputs.inputnode.in_file = 'diffusion.nii'
>>> nipype_dmri_preprocess.inputs.inputnode.in_bvec = 'diffusion.bvec'
>>> nipype_dmri_preprocess.inputs.inputnode.ref_num = 0
>>> nipype_dmri_preprocess.inputs.inputnode.fieldmap_mag = 'magnitude.nii'
>>> nipype_dmri_preprocess.inputs.inputnode.fieldmap_pha = 'phase.nii'
>>> nipype_dmri_preprocess.inputs.inputnode.te_diff = 2.46
>>> nipype_dmri_preprocess.inputs.inputnode.epi_echospacing = 0.77
>>> nipype_dmri_preprocess.inputs.inputnode.epi_rev_encoding = False
>>> nipype_dmri_preprocess.inputs.inputnode.pi_accel_factor = True
>>> nipype_dmri_preprocess.run() # doctest: +SKIP

Inputs:

<table>
<thead>
<tr>
<th>inputnode.in_file</th>
<th>The diffusion data</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputnode.in_bvec</td>
<td>The b-matrix file, in FSL format and consistent with the in_file orientation</td>
</tr>
<tr>
<td>inputnode.ref_num</td>
<td>The reference volume (a b=0 volume in dMRI)</td>
</tr>
<tr>
<td>inputnode.fieldmap_mag</td>
<td>The magnitude of the fieldmap</td>
</tr>
<tr>
<td>inputnode.fieldmap_pha</td>
<td>The phase difference of the fieldmap</td>
</tr>
<tr>
<td>inputnode.te_diff</td>
<td>TE increment used (in msec.) on the fieldmap acquisition (generally 2.46ms for 3T scanners)</td>
</tr>
<tr>
<td>inputnode.epi_echospacing</td>
<td>The EPI EchoSpacing parameter (in msec.)</td>
</tr>
<tr>
<td>inputnode.epi_rev_encoding</td>
<td>True if reverse encoding was used (generally False)</td>
</tr>
<tr>
<td>inputnode.pi_accel_factor</td>
<td>Parallel imaging factor (aka GRAPPA acceleration)</td>
</tr>
<tr>
<td>inputnode.vsm_sigma</td>
<td>Sigma (in mm.) of the gaussian kernel used for in-slice smoothing of the deformation field (voxel shift map, vsm)</td>
</tr>
</tbody>
</table>

Outputs:

<table>
<thead>
<tr>
<th>outputnode.dmri_corrected</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>outputnode.bvec_rotated</td>
<td></td>
</tr>
</tbody>
</table>

Optional arguments:

| use_fieldmap      | True if there are fieldmap files that should be used (default True) |
| fieldmap_registration | True if registration to fieldmap should be performed (default False) |
Graph

2.10.2 create_eddy_correct_pipeline()

Link to code
Deprecated since version 0.9.3: Use \texttt{nipype.workflows.dmri.preprocess.epi.ecc_pipeline()} instead.

Creates a pipeline that replaces eddy\_correct script in FSL. It takes a series of diffusion weighted images and linearly co-registers them to one reference image. No rotation of the B-matrix is performed, so this pipeline should be executed after the motion correction pipeline.

### Example

```python
>>> nipype_eddycorrect = create_eddy_correct_pipeline('nipype_eddycorrect')
>>> nipype_eddycorrect.inputs.inputnode.in_file = 'diffusion.nii'
>>> nipype_eddycorrect.inputs.inputnode.ref_num = 0
>>> nipype_eddycorrect.run() # doctest: +SKIP
```

**Inputs:**

- inputnode.in_file
- inputnode.ref_num

**Outputs:**

- outputnode.eddy_corrected
2.10.3 `create_epidewarp_pipeline()`

Link to code
Replaces the epidewarp.fsl script (http://www.nmr.mgh.harvard.edu/~greve/fbirn/b0/epidewarp.fsl) for susceptibility distortion correction of dMRI & fMRI acquired with EPI sequences and the fieldmap information (Jezzard et al., 1995) using FSL's FUGUE. The registration to the (warped) fieldmap (strictly following the original script) is available using fieldmap_registration=True.
Warning: This workflow makes use of epidewarp.fsl a script of FSL deprecated long time ago. The use of this workflow is not recommended, use nipype.workflows.dmri.preprocess.epi.sdc_fmb() instead.

Example

```python
>>> nipype_epicorrect = create_epidewarp_pipeline('nipype_epidewarp', fieldmap_registration=False)
>>> nipype_epicorrect.inputs.inputnode.in_file = 'diffusion.nii'
>>> nipype_epicorrect.inputs.inputnode.fieldmap_mag = 'magnitude.nii'
>>> nipype_epicorrect.inputs.inputnode.fieldmap_pha = 'phase.nii'
>>> nipype_epicorrect.inputs.inputnode.te_diff = 2.46
>>> nipype_epicorrect.inputs.inputnode.epi_echosSpacing = 0.77
>>> nipype_epicorrect.inputs.inputnode.epi_rev_encoding = False
>>> nipype_epicorrect.inputs.inputnode.pi_accel_factor = 1.0
>>> nipype_epicorrect.run()  # doctest: +SKIP
```

Inputs:

- `inputnode.in_file` - The volume acquired with EPI sequence
- `inputnode.fieldmap_mag` - The magnitude of the fieldmap
- `inputnode.fieldmap_pha` - The phase difference of the fieldmap
- `inputnode.te_diff` - Time difference between TE in ms.
- `inputnode.epi_echosspacing` - The echo spacing (aka dwell time) in the EPI sequence
- `inputnode.epi_ph_encoding_dir` - The phase encoding direction in EPI acquisition (default y)
- `inputnode.epi_rev_encoding` - True if it is acquired with reverse encoding
- `inputnode.pi_accel_factor` - Acceleration factor used for EPI parallel imaging (GRAPPA)
- `inputnode.vsm_sigma` - Sigma value of the gaussian smoothing filter applied to the vsm (voxel shift map)
- `inputnode.ref_num` - The reference volume (B=0 in dMRI or a central frame in fMRI)

Outputs:

- `outputnode.epi_corrected`

Optional arguments:

- `fieldmap_registration` - True if registration to fieldmap should be done (default False)
2.10.4 create_motion_correct_pipeline()

Link to code
Creates a pipeline that corrects for motion artifact in dMRI sequences. It takes a series of diffusion weighted images and rigidly co-registers them to one reference image. Finally, the b-matrix is rotated accordingly (Lee-mans et al. 2009 - http://www.ncbi.nlm.nih.gov/pubmed/19319973), making use of the rotation matrix obtained by FLIRT.

Deprecated since version 0.9.3: Use nipype.workflows.dmri.preprocess.epi.hmc_pipeline() instead.
**Warning:** This workflow rotates the b-vectors, so please be advised that not all the dicom converters ensure the consistency between the resulting nifti orientation and the b matrix table (e.g. dcm2nii checks it).

### Example

```python
>>> nipype_motioncorrect = create_motion_correct_pipeline('nipype_motioncorrect')
>>> nipype_motioncorrect.inputs.inputnode.in_file = 'diffusion.nii'
>>> nipype_motioncorrect.inputs.inputnode.in_bvec = 'diffusion.bvec'
>>> nipype_motioncorrect.inputs.inputnode.ref_num = 0
>>> nipype_motioncorrect.run() # doctest: +SKIP
```

**Inputs:**

- `inputnode.in_file`
- `inputnode.ref_num`
- `inputnode.in_bvec`

**Outputs:**

- `outputnode.motion_corrected`
- `outputnode.out_bvec`
2.10.5 `fieldmap_correction()`

Link to code

Deprecated since version 0.9.3: Use `nipype.workflows.dmri.preprocess.epi.sdc_fmb()` instead.

Fieldmap-based retrospective correction of EPI images for the susceptibility distortion artifact (Jezzard et al., 1995). Fieldmap images are assumed to be already registered to EPI data, and a brain mask is required. Replaces the former workflow, still available as `create_epidewarp_pipeline()`. The difference with respect the epidewarp pipeline is that now the workflow uses the new `fsl_prepare_fieldmap` available as of FSL 5.0.
Example

```python
>>> nipype_epicorrect = fieldmap_correction('nipype_epidewarp')
>>> nipype_epicorrect.inputs.inputnode.in_file = 'diffusion.nii'
>>> nipype_epicorrect.inputs.inputnode.in_mask = 'brainmask.nii'
>>> nipype_epicorrect.inputs.inputnode.fieldmap_pha = 'phase.nii'
>>> nipype_epicorrect.inputs.inputnode.fieldmap_mag = 'magnitude.nii'
>>> nipype_epicorrect.inputs.inputnode.te_diff = 2.46
>>> nipype_epicorrect.inputs.inputnode.epi_echospacing = 0.77
>>> nipype_epicorrect.inputs.inputnode.encoding_direction = 'y'
>>> nipype_epicorrect.run()  # doctest: +SKIP
```

Inputs:

- `in_file`: The volume acquired with EPI sequence
- `in_mask`: A brain mask
- `fieldmap_pha`: The phase difference map from the fieldmapping, registered to `in_file`
- `fieldmap_mag`: The magnitude maps (usually 4D, one magnitude per GRE scan) from the fieldmapping, registered to `in_file`
- `te_diff`: Time difference in msec. between TE in ms of the fieldmapping (usually a GRE sequence).
- `epi_echospacing`: The effective echo spacing (aka dwell time) in msec. of the EPI sequence. If EPI was acquired with parallel imaging, then the effective echo spacing is `eff_es = es / acc_factor`.
- `encoding_direction`: The phase encoding direction in EPI acquisition (default y)
- `vsm_sigma`: Sigma value of the gaussian smoothing filter applied to the vsm (voxel shift map)

Outputs:

- `epi_corrected`
- `out_vsm`
2.10.6 topup_correction()

Link to code

Deprecated since version 0.9.3: Use `nipype.workflows.dmri.preprocess.epi.sdc_peb()` instead.

Corrects for susceptibility distortion of EPI images when one reverse encoding dataset has been acquired

Example

```python
>>> nppype_epicorrect = topup_correction('nipype_topup')
>>> nppype_epicorrect.inputs.inputnode.in_file_dir = 'epi.nii'
>>> nppype_epicorrect.inputs.inputnode.in_file_rev = 'epi_rev.nii'
>>> nppype_epicorrect.inputs.inputnode.encoding_direction = ['y', 'y-']
>>> nppype_epicorrect.inputs.inputnode.ref_num = 0
>>> nppype_epicorrect.run() # doctest: +SKIP
```

Inputs:
**Graph**

```python
inputnode.in_file_dir - EPI volume acquired in 'forward' phase encoding
inputnode.in_file_rev - EPI volume acquired in 'reversed' phase encoding
inputnode.encoding_direction - Direction encoding of in_file_dir
inputnode.ref_num - Identifier of the reference volumes (usually B0 volume)

Outputs:
outputnode.epi_corrected
```
2.11 workflows.dmri.fsl.tbss

2.11.1 create_tbss_1_preproc()

Link to code
Preprocess FA data for TBSS: erodes a little and zero end slicers and creates masks (for use in FLIRT & FNIRT from FSL). A pipeline that does the same as tbss_1_preproc script in FSL.

Example

```python
>>> from nipype.workflows.dmri.fsl import tbss
>>> tbss1 = tbss.create_tbss_1_preproc()
>>> tbss1.inputs.inputnode.fa_list = ['s1_FA.nii', 's2_FA.nii', 's3_FA.nii']
```

Inputs:

<table>
<thead>
<tr>
<th>inputnode.fa_list</th>
</tr>
</thead>
</table>

Outputs:

<table>
<thead>
<tr>
<th>outputnode.fa_list</th>
<th>outputnode.mask_list</th>
<th>outputnode.slices</th>
</tr>
</thead>
</table>
Graph

2.11.2 create_tbss_2_reg()

Link to code

TBSS nonlinear registration: A pipeline that does the same as ‘tbss_2_reg -t’ script in FSL. ‘-n’ option is not supported at the moment.

Example

```python
>>> from nipype.workflows.dmri.fsl import tbss
>>> tbss2 = create_tbss_2_reg(name="tbss2")
>>> tbss2.inputs.inputnode.target = fsl.Info.standard_image("FMRIB58_FA_1mm.nii.gz")  # doctest: +SKIP
>>> tbss2.inputs.inputnode.fa_list = ["s1_FA.nii", "s2_FA.nii", "s3_FA.nii"]
>>> tbss2.inputs.inputnode.mask_list = ["s1_mask.nii", "s2_mask.nii", "s3_mask.nii"]
```
Inputs:

<table>
<thead>
<tr>
<th>inputnode.fa_list</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputnode.mask_list</td>
</tr>
<tr>
<td>inputnode.target</td>
</tr>
</tbody>
</table>

Outputs:

| outputnode.field_list |

Graph

2.11.3 create_tbss_3_postreg()

Link to code

Post-registration processing: derive mean_FA and mean_FA_skeleton from mean of all subjects in study. Target is assumed to be FMRIB58_FA_1mm. A pipeline that does the same as ‘tbss_3_postreg -S’ script from FSL Setting ‘estimate_skeleton to False will use precomputed FMRIB58_FA-skeleton_1mm skeleton (same as ‘tbss_3_postreg -T’).

Example
>>> from nipype.workflows.dmri.fsl import tbss
>>> tbss3 = tbss.create_tbss_3_postreg()
>>> tbss3.inputs.inputnode.fa_list = ['s1_wrapped_FA.nii', 's2_wrapped_FA.nii',
                                      's3_wrapped_FA.nii']

Inputs:

```
inputnode.field_list
inputnode.fa_list
```

Outputs:

```
outputnode.groupmask
outputnode.skeleton_file
outputnode.meanfa_file
outputnode.mergefa_file
```
Graph

```
inputnode (utility)

applywarp (fsl)

mergefa (fsl)

groupmask (fsl)

maskgroup (fsl)

meanfa (fsl)

makeskeleton (fsl)

outputnode (utility)

tbss_3_postreg
```
2.11.4 create_tbss_4_prestats()

Link to code

Post-registration processing: Creating skeleton mask using a threshold projecting all FA data onto skeleton.

A pipeline that does the same as tbss_4_prestats script from FSL

Example

```python
>>> from nipype.workflows.dmri.fsl import tbss
>>> tbss4 = tbss.create_tbss_4_prestats(name='tbss4')
>>> tbss4.inputs.inputnode.skeleton_thresh = 0.2
```

Inputs:

<table>
<thead>
<tr>
<th>inputnode.skeleton_thresh</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputnode.groupmask</td>
</tr>
<tr>
<td>inputnode.skeleton_file</td>
</tr>
<tr>
<td>inputnode.meanfa_file</td>
</tr>
<tr>
<td>inputnode.mergefa_file</td>
</tr>
</tbody>
</table>

Outputs:

<table>
<thead>
<tr>
<th>outputnode.all_FA_skeletonised</th>
</tr>
</thead>
<tbody>
<tr>
<td>outputnode.mean_FA_skeleton_mask</td>
</tr>
<tr>
<td>outputnode.distance_map</td>
</tr>
<tr>
<td>outputnode.skeleton_file</td>
</tr>
</tbody>
</table>
Graph

2.11.5 create_tbss_all()

Link to code
Create a pipeline that combines create_tbss_* pipelines

Example
```python
>>> from nipype.workflows.dmri.fsl import tbss
>>> tbss_wf = tbss.create_tbss_all('tbss', estimate_skeleton=True)
>>> tbss_wf.inputs.inputnode.skeleton_thresh = 0.2
>>> tbss_wf.inputs.inputnode.fa_list = ['s1_wrapped_FA.nii', 's2_wrapped_FA.nii', 's3_wrapped_FA.nii']

>>> tbss_wf = tbss.create_tbss_all('tbss', estimate_skeleton=False)
>>> tbss_wf.inputs.inputnode.skeleton_thresh = 0.2
>>> tbss_wf.inputs.inputnode.fa_list = ['s1_wrapped_FA.nii', 's2_wrapped_FA.nii', 's3_wrapped_FA.nii']
```

**Inputs:**

- `inputnode.fa_list`
- `inputnode.skeleton_thresh`

**Outputs:**

- `outputnode.meanfa_file`
- `outputnode.projectedfa_file`
- `outputnode.skeleton_file`
- `outputnode.skeleton_mask`
2.11.6 create_tbss_non_FA()

A pipeline that implement tbss_non_FA in FSL

Example

```python
>>> from nipype.workflows.dmri.fsl import tbss
>>> tbss_MD = tbss.create_tbss_non_FA()
>>> tbss_MD.inputs.inputnode.file_list = []
>>> tbss_MD.inputs.inputnode.field_list = []
>>> tbss_MD.inputs.inputnode.skeleton_thresh = 0.2
>>> tbss_MD.inputs.inputnode.groupmask = './xxx'
>>> tbss_MD.inputs.inputnode.meanfa_file = './xxx'
>>> tbss_MD.inputs.inputnode.distance_map = []
>>> tbss_MD.inputs.inputnode.all_FA_file = './xxx'
```

Inputs:

- `inputnode.file_list`
- `inputnode.field_list`
- `inputnode.skeleton_thresh`
- `inputnode.groupmask`
- `inputnode.meanfa_file`
- `inputnode.distance_map`
- `inputnode.all_FA_file`

Outputs:

- `outputnode.projected_nonFA_file`
Graph

2.11.7 \texttt{tbssl\_op\_string()}
Link to code

2.11.8 \texttt{tbss4\_op\_string()}
Link to code
2.12 workflows.dmri.fsl.utils

2.12.1 apply_all_corrections()

Link to code

Combines two lists of linear transforms with the deformation field map obtained typically after the SDC process. Additionally, computes the corresponding bspline coefficients and the map of determinants of the jacobian.
Graph

inputnode (utility)

ConvertWarp (fsl)

SplitDWIs (fsl)

Reference (utility)

CoeffComp (fsl)

UnwarpDWIs (fsl)

JacobianComp (fsl)

ModulateDWIs (fsl)

RemoveNegative (fsl)

MergeDWIs (fsl)

outputnode (utility)

UnwarpArtifacts
2.12.2 cleanup_edge_pipeline()

Link to code
Perform some de-spiking filtering to clean up the edge of the fieldmap (copied from fsl_prepare_fieldmap)

Graph
2.12.3 dwi_flirt()

Link to code
Generates a workflow for linear registration of dwi volumes

Graph

```
inputnode (utility)
  InitXforms (utility)
  MskDilate (fsl)
  SplitDWIs (fsl)
  Bias (ants)
  DWEqualize (utility)
  B0Equalize (utility)
  CoRegistration (fsl)
  ApplyXFM (fsl)
  RemoveNegative (fsl)
  MergeDWIs (fsl)

outputnode (utility)
```

2.12.4 vsm2warp()

Link to code
Converts a voxel shift map (vsm) to a displacements field (warp).
Graph

2.12.5 add_empty_vol()

Link to code
Adds an empty vol to the phase difference image

2.12.6 b0_average()

Link to code
A function that averages the b0 volumes from a DWI dataset. As current dMRI data are being acquired with all b-values > 0.0, the lowb volumes are selected by specifying the parameter max_b.

Warning: b0 should be already registered (head motion artifact should be corrected).
2.12.7 b0_indices()
Link to code
Extract the indices of slices in a b-values file with a low b value

2.12.8 compute_readout()
Link to code
Computes readout time from epi params (see eddy documentation).

**Warning:** params['echospacing'] should be in sec units.

2.12.9 copy_hdr()
Link to code

2.12.10 demean_image()
Link to code
Demean image data inside mask

2.12.11 eddy_rotate_bvecs()
Link to code
Rotates the input bvec file accordingly with a list of parameters sourced from eddy, as explained here.

2.12.12 enhance()
Link to code

2.12.13 extract_bval()
Link to code
Writes an image containing only the volumes with b-value specified at input

2.12.14 hmc_split()
Link to code
Selects the reference and moving volumes from a dwi dataset for the purpose of HMC.

2.12.15 insert_mat()
Link to code

2.12.16 rads2radsec()
Link to code
Converts input phase difference map to rads

2.12.17 recompose_dwi()
Link to code
Recompose back the dMRI data accordingly the b-values table after EC correction
2.12.18 recompose_xfm()

Link to code
Insert identity transformation matrices in b0 volumes to build up a list

2.12.19 remove_comp()

Link to code
Removes the volume volid from the 4D nifti file

2.12.20 reorient_bvecs()

Link to code
Checks reorientations of in_dwi w.r.t. old_dwi and reorients the in_bvec table accordingly.

2.12.21 rotate_bvecs()

Link to code
Rotates the input bvec file accordingly with a list of matrices.

Note: the input affine matrix transforms points in the destination image to their corresponding coordinates in the original image. Therefore, this matrix should be inverted first, as we want to know the target position of \( \vec{r} \).

2.12.22 siemens2rads()

Link to code
Converts input phase difference map to rads

2.12.23 time_avg()

Link to code
Average the input time-series, selecting the indices given in index

Warning: time steps should be already registered (corrected for head motion artifacts).

2.13 workflows.dmri.mrtrix.connectivity_mapping

2.13.1 create_connectivity_pipeline()

Link to code
Creates a pipeline that does the same connectivity processing as in the dMRI: Connectivity - MRtrix, CMTK, FreeSurfer example script. Given a subject id (and completed Freesurfer reconstruction) diffusion-weighted image, b-values, and b-vectors, the workflow will return the subject’s connectome as a Connectome File Format (CFF) file for use in Connectome Viewer (http://www.cmtk.org).

Example

```python
>>> from nipype.workflows.dmri.mrtrix.connectivity_mapping import create_connectivity_pipeline
>>> conmapper = create_connectivity_pipeline("nipype_conmap")
>>> conmapper.inputs.inputnode.subjects_dir = '.'
>>> conmapper.inputs.inputnode.subject_id = 'subj1'
```
>>> conmapper.inputs.inputnode.dwi = 'data.nii.gz'
>>> conmapper.inputs.inputnode.bvecs = 'bvecs'
>>> conmapper.inputs.inputnode.bvals = 'bvals'
>>> conmapper.run()  # doctest: +SKIP

Inputs:

inputnode.subject_id
inputnode.subjects_dir
inputnode.dwi
inputnode.bvecs
inputnode.bvals
inputnode.resolution_network_file

Outputs:

outputnode.connectome
outputnode.cmatrix
outputnode.networks
outputnode.fa
outputnode.struct
outputnode.tracts
outputnode.rois
outputnode.odfs
outputnode.filtered_tractography
outputnode.tdi
outputnode.nxstatscff
outputnode.nxcsv
outputnode.cmatrices_csv
outputnode.mean_fiber_length
outputnode.median_fiber_length
outputnode.fiber_length_std

Graph

2.14 workflows.dmri.mrtrix.diffusion

2.14.1 create_mrtrix_dti_pipeline()

Link to code

Creates a pipeline that does the same diffusion processing as in the ../users/examples/dmri_mrtrix_dti example script. Given a diffusion-weighted image, b-values, and b-vectors, the workflow will return the tractography
computed from spherical deconvolution and probabilistic streamline tractography

Example

```python
>>> dti = create_mrtrix_dti_pipeline("mrtrix_dti")
>>> dti.inputs.inputnode.dwi = 'data.nii'
>>> dti.inputs.inputnode.bvals = 'bvals'
>>> dti.inputs.inputnode.bvecs = 'bvecs'
>>> dti.run()  # doctest: +SKIP
```

Inputs:

- `inputnode.dwi`
- `inputnode.bvecs`
- `inputnode.bvals`

Outputs:

- `outputnode.fa`
- `outputnode.tdi`
- `outputnode.tracts_tck`
- `outputnode.tracts_trk`
- `outputnode.csdeconv`

Graph
2.15 workflows.dmri.mrtrix.group_connectivity

2.15.1 create_group_connectivity_pipeline()

Link to code

Creates a pipeline that performs MRtrix structural connectivity processing on groups of subjects. Given a diffusion-weighted image, and text files containing the associated b-values and b-vectors, the workflow will return each subjects’ connectomes in a Connectome File Format (CFF) file, for use in Connectome Viewer (http://www.cmtk.org).

Example

```python
>>> import nipype.interfaces.freesurfer as fs
>>> import nipype.workflows.dmri.mrtrix.group_connectivity as groupwork
>>> import cmp
# doctest: +SKIP
>>> from nipype.testing import example_data

>>> subjects_dir = '.'
>>> data_dir = '.'
>>> output_dir = '.'
>>> fs.FSCommand.set_default_subjects_dir(subjects_dir)
>>> group_list = {}
>>> group_list['group1'] = ['subj1', 'subj2']
>>> group_list['group2'] = ['subj3', 'subj4']
>>> template_args = dict(dwi=[['subject_id', 'dwi']], bvecs=[['subject_id', 'bvecs']], bvals=[['subject_id', 'bvals']])
>>> group_id = 'group1'

>>> l1pipeline = groupwork.create_group_connectivity_pipeline(group_list, group_id, data_dir, subjects_dir, output_dir, template_args)
>>> parcellation_name = 'scale500'
>>> l1pipeline.inputs.connectivity.mapping.Parcellate.parcellation_name = parcellation_name
>>> cmp_config = cmp.configuration.PipelineConfiguration()  # doctest: +SKIP
>>> cmp_config.parcellation_scheme = "Lausanne2008"  # doctest: +SKIP
>>> l1pipeline.inputs.connectivity.mapping.inputnode_within.resolution_network_file = cmp_config._get_lausanne_parcellation('Lausanne2008')['parcellation_name']["node_information_graphml"]  # doctest: +SKIP
>>> l1pipeline.run()  # doctest: +SKIP
```

Inputs:

- **group_list**: Dictionary of subject lists, keyed by group name
- **group_id**: String containing the group name
- **data_dir**: Path to the data directory
- **subjects_dir**: Path to the Freesurfer 'subjects' directory
- **output_dir**: Path for the output files
- **template_args_dict**: Dictionary of template arguments for the connectivity pipeline datasource

  e.g.  info = dict(dwi=[['subject_id', 'dwi']], bvecs=[['subject_id', 'bvecs']], bvals=[['subject_id', 'bvals']])
3.1 workflows.fmri.fsl.estimate

3.1.1 create_fixed_effects_flow()

Link to code
Create a fixed-effects workflow
This workflow is used to combine registered copes and varcopes across runs for an individual subject

Example

```python
g fixedfx = create_fixed_effects_flow()
g fixedfx.base_dir = '.'
g fixedfx.inputs.inputspec.copes = [['cope1run1.nii.gz', 'cope1run2.nii.gz'], ['cope2run1.nii.gz', 'cope2run2.nii.gz']] # per contrast
g fixedfx.inputs.inputspec.varcopes = [['varcope1run1.nii.gz', 'varcope1run2.nii.gz'], ['varcope2run1.nii.gz', 'varcope2run2.nii.gz']] # per contrast
g fixedfx.inputs.inputspec.dof_files = ['dofrun1', 'dofrun2'] # per run
g fixedfx.run() # doctest: +SKIP```

Inputs:
- `inputspec.copes`: list of list of cope files (one list per contrast)
- `inputspec.varcopes`: list of list of varcope files (one list per contrast)
- `inputspec.dof_files`: degrees of freedom files for each run

Outputs:
- `outputspec.res4d`: 4d residual time series
- `outputspec.copes`: contrast parameter estimates
- `outputspec.varcopes`: variance of contrast parameter estimates
- `outputspec.zstats`: z statistics of contrasts
- `outputspec.tstats`: t statistics of contrasts
3.1.2 create_modelfit_workflow()

Link to code
Create an FSL individual modelfitting workflow

Example

```python
>>> modelfit = create_modelfit_workflow()
>>> modelfit.base_dir = '.'
>>> info = dict()
>>> modelfit.inputs.inputspec.session_info = info
>>> modelfit.inputs.inputspec.interscan_interval = 3.
>>> modelfit.inputs.inputspec.film_threshold = 1000
>>> modelfit.run()  #doctest: +SKIP
```

Inputs:

- `inputspec.session_info`: info generated by modelgen.SpecifyModel
- `inputspec.interscan_interval`: interscan interval
- `inputspec.contrasts`: list of contrasts

(continues on next page)
inputspec.film_threshold : image threshold for FILM estimation
inputspec.model_serial_correlations
inputspec.bases

Outputs:

outputspec.copes
outputspec.varcopes
outputspec.dof_file
outputspec.pfiles
outputspec.zfiles
outputspec.parameter_estimates
3.1.3 create_overlay_workflow()

Link to code

Setup overlay workflow

Graph

![Graph of overlaystats (fsl) and slicestats (fsl) connected to overlay]

3.2 workflows.fmri.fsl.preprocess

3.2.1 create_featreg_preproc()

Link to code

Create a FEAT preprocessing workflow with registration to one volume of the first run

Parameters

- **name**: name of workflow (default: featpreproc)
- **highpass**: boolean (default: True)
- **whichvol**: which volume of the first run to register to ('first', 'middle', 'last')
- **whichrun**: which run to draw reference volume from (integer index or 'first', 'middle', 'last')

Inputs:

- **inputspec.func**: functional runs (filename or list of filenames)
- **inputspec.fwhm**: fwhm for smoothing with SUSAN
- **inputspec.highpass**: HWHM in TRs (if created with highpass=True)

Outputs:

- **outputspec.reference**: volume to which runs are realigned
- **outputspec.motion_parameters**: motion correction parameters
- **outputspec.realigned_files**: motion corrected files
- **outputspec.motion_plots**: plots of motion correction parameters
- **outputspec.mask**: mask file used to mask the brain
- **outputspec.smoothed_files**: smoothed functional data

(continues on next page)
Example

```python
>>> preproc = create_featreg_preproc()
>>> preproc.inputs.inputspec.func = ['f3.nii', 'f5.nii']
>>> preproc.inputs.inputspec.fwhm = 5
>>> preproc.inputs.inputspec.highpass = 128./(2*2.5)
>>> preproc.base_dir = '/tmp'
>>> preproc.run() # doctest: +SKIP

>>> preproc = create_featreg_preproc(highpass=False, whichvol='mean')
>>> preproc.inputs.inputspec.func = 'f3.nii'
>>> preproc.inputs.inputspec.fwhm = 5
>>> preproc.base_dir = '/tmp'
>>> preproc.run() # doctest: +SKIP
```
Graph
3.2.2 create_fsl_fs_preproc()

Link to code
Create a FEAT preprocessing workflow together with freesurfer

Parameters

name : name of workflow (default: preproc)
highpass : boolean (default: True)
whichvol : which volume of the first run to register to ('first', 'middle', 'mean →')

Inputs:

inputspec.func : functional runs (filename or list of filenames)
inputspec.fwhm : fwhm for smoothing with SUSAN
inputspec.highpass : HWHM in TRs (if created with highpass=True)
inputspec.subject_id : freesurfer subject id
inputspec.subjects_dir : freesurfer subjects dir

Outputs:

outputs.spec.reference : volume to which runs are realigned
outputs.spec.motion_parameters : motion correction parameters
outputs.spec.realigned_files : motion corrected files
outputs.spec.motion_plots : plots of motion correction parameters
outputs.spec.mask_file : mask file used to mask the brain
outputs.spec.smoothed_files : smoothed functional data
outputs.spec.highpassed_files : highpassed functional data (if highpass=True)
outputs.spec.reg_file : bbregister registration files
outputs.spec.reg_cost : bbregister registration cost files

Example

```python
>>> preproc = create_fsl_fs_preproc(whichvol='first')
>>> preproc.inputs.inputspec.highpass = 128./(2*2.5)
>>> preproc.inputs.inputspec.func = ['f3.nii', 'f5.nii']
>>> preproc.inputs.inputspec.subjects_dir = '.
>>> preproc.inputs.inputspec.subject_id = 's1'
>>> preproc.inputs.inputspec.fwhm = 6
>>> preproc.run() # doctest: +SKIP
```
3.2.3 create_parallelfeat_preproc()

Link to code
Preprocess each run with FSL independently of the others

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name of workflow (default: featpreproc)</td>
</tr>
<tr>
<td>highpass</td>
<td>boolean (default: True)</td>
</tr>
</tbody>
</table>

Inputs:
inputspec.func : functional runs (filename or list of filenames)
inputspec.fwhm : fwhm for smoothing with SUSAN
inputspec.highpass : HWHM in TRs (if created with highpass=True)

Outputs:

outputs.spec.reference : volume to which runs are realigned
outputs.spec.motion_parameters : motion correction parameters
outputs.spec.realigned_files : motion corrected files
outputs.spec.motion_plots : plots of motion correction parameters
outputs.spec.mask : mask file used to mask the brain
outputs.spec.smoothed_files : smoothed functional data
outputs.spec.highpassed_files : highpassed functional data (if highpass=True)
outputs.spec.mean : mean file

Example

```python
>>> preproc = create_parallelfeat_preproc()
>>> preproc.inputs.inputspec.func = ['f3.nii', 'f5.nii']
>>> preproc.inputs.inputspec.fwhm = 5
>>> preproc.inputs.inputspec.highpass = 128./(2*2.5)
>>> preproc.base_dir = '/tmp'
>>> preproc.run() # doctest: +SKIP

>>> preproc = create_parallelfeat_preproc(highpass=False)
>>> preproc.inputs.inputspec.func = 'f3.nii'
>>> preproc.inputs.inputspec.fwhm = 5
>>> preproc.base_dir = '/tmp'
>>> preproc.run() # doctest: +SKIP
```
3.2.4 create_susan_smooth()

Create a SUSAN smoothing workflow

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name of workflow (default: susan_smooth)</td>
</tr>
<tr>
<td>separate_masks</td>
<td>separate masks for each run</td>
</tr>
</tbody>
</table>

**Inputs:**

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputnode.in_files</td>
<td>functional runs (filename or list of filenames)</td>
</tr>
<tr>
<td>inputnode.fwhm</td>
<td>fwhm for smoothing with SUSAN (float or list of floats)</td>
</tr>
<tr>
<td>inputnode.mask_file</td>
<td>mask used for estimating SUSAN thresholds (but not for smoothing)</td>
</tr>
</tbody>
</table>

**Outputs:**

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outputnode.smoothed_files</td>
<td>functional runs (filename or list of filenames)</td>
</tr>
</tbody>
</table>

**Example**

```python
gf
>>> smooth = create_susan_smooth()
>>> smooth.inputs.inputnode.in_files = 'f3.nii'
>>> smooth.inputs.inputnode.fwhm = 5
>>> smooth.inputs.inputnode.mask_file = 'mask.nii'
>>> smooth.run() # doctest: +SKIP
```
Graph

3.2.5 chooseindex()

Link to code
3.2.6 create_reg_workflow()

Create a FEAT preprocessing workflow

Parameters

```
name : name of workflow (default: 'registration')
```

Inputs:

```
inputspec.source_files : files (filename or list of filenames to register)
inputspec.mean_image : reference image to use
inputspec.anatomical_image : anatomical image to coregister to
inputspec.target_image : registration target
```

Outputs:

```
outputspec.func2anat_transform : FLIRT transform
outputspec.anat2target_transform : FLIRT+FNIRT transform
outputspec.transformed_files : transformed files in target space
outputspec.transformed_mean : mean image in target space
```

Example

3.2.7 getbtthresh()

3.2.8 getmeanscale()

3.2.9 getthreshop()

3.2.10 getusans()

3.2.11 pickfirst()

3.2.12 pickmiddle()

3.2.13 pickrun()

3.2.14 pickvol()
3.3 workflows.fmri.spm.preprocess

3.3.1 create_DARTEL_template()

Link to code
Create a vbm workflow that generates DARTEL-based template

Example

```python
>>> preproc = create_DARTEL_template()
>>> preproc.inputs.inputspec.structural_files = [os.path.abspath('s1.nii'), os.path.abspath('s3.nii')]
>>> preproc.inputs.inputspec.template_prefix = 'Template'
>>> preproc.run()  # doctest: +SKIP
```

Inputs:

- `inputspec.structural_files`: structural data to be used to create templates
- `inputspec.template_prefix`: prefix for dartel template

Outputs:

- `outputspec.template_file`: DARTEL template
- `outputspec.flow_fields`: warps from input struct files to the template
3.3.2 create_spm_preproc()

Create an spm preprocessing workflow with freesurfer registration and artifact detection. The workflow realigns and smooths and registers the functional images with the subject’s freesurfer space.

Example

```plaintext
>>> preproc = create_spm_preproc()
>>> preproc.base_dir = '.
>>> preproc.inputs.inputspec.fwhm = 6
>>> preproc.inputs.inputspec.subject_id = 's1'
>>> preproc.inputs.inputspec.subjects_dir = '.
>>> preproc.inputs.inputspec.functionals = ['f3.nii', 'f5.nii']
>>> preproc.inputs.inputspec.norm_threshold = 1
>>> preproc.inputs.inputspec.zintensity_threshold = 3
```

Inputs:

- `inputspec.functionals`: functional runs use 4d nifti
- `inputspec.subject_id`: freesurfer subject id
- `inputspec.subjects_dir`: freesurfer subjects dir

(continues on next page)
| inputspec.fwhm : smoothing fwhm |
| inputspec.norm_threshold : norm threshold for outliers |
| inputspec.zintensity_threshold : intensity threshold in z-score |

**Outputs:**

| outputspec.realignement_parameters : realignment parameter files |
| outputspec.smoothed_files : smoothed functional files |
| outputspec.outlier_files : list of outliers |
| outputspec.outlier_stats : statistics of outliers |
| outputspec.outlier_plots : images of outliers |
| outputspec.mask_file : binary mask file in reference image space |
| outputspec.reg_file : registration file that maps reference image to freesurfer space |
| outputspec.reg_cost : cost of registration (useful for detecting misalignment) |
3.3.3 create_vbm_preproc()

Link to code
Create a vbm workflow that generates DARTEL-based warps to MNI space
Based on: http://www.fil.ion.ucl.ac.uk/~john/misc/VBMclass10.pdf
Example

```python
>>> preproc = create_vbm_preproc()
>>> preproc.inputs.inputspec.fwhm = 8
>>> preproc.inputs.inputspec.structural_files = [
...    os.path.abspath('s1.nii'), os.path.abspath('s3.nii')]
>>> preproc.inputs.inputspec.template_prefix = 'Template'
>>> preproc.run()  # doctest: +SKIP
```

**Inputs:**

- **inputspec.structural_files**: structural data to be used to create templates
- **inputspec.fwhm**: single of triplet for smoothing when normalizing to MNI space
- **inputspec.template_prefix**: prefix for dartel template

**Outputs:**

- **outputspec.normalized_files**: normalized gray matter files
- **outputspec.template_file**: DARTEL template
- **outputspec.icv**: intracranial volume (cc - assuming dimensions in mm)
Graph

- inputspec (utility)
- dartel_template
- inputspec (utility)
- segment (spm)
- dartel (spm)
- calc_icv (utility)
- outputspec (utility)
- norm2mni (spm)
- outputspec (utility)
- vbmPreproc
4.1 workflows.misc.utils

4.1.1 get_affine()
Link to code

4.1.2 get_data_dims()
Link to code

4.1.3 get_vox_dims()
Link to code

4.1.4 id_list_from_lookup_table()
Link to code

4.1.5 region_list_from_volume()
Link to code

4.1.6 select_aparc()
Link to code

4.1.7 select_aparc_annot()
Link to code
5.1 workflows.rsfmri.fsl.resting

5.1.1 create_realign_flow()

Link to code
Realign a time series to the middle volume using spline interpolation
Uses MCFLIRT to realign the time series and ApplyWarp to apply the rigid body transformations using spline interpolation (unknown order).

Example

```python
>>> wf = create_realign_flow()
>>> wf.inputs.inputspec.func = 'f3.nii'
>>> wf.run() # doctest: +SKIP
```
5.1.2 create_resting_preproc()

Link to code
Create a “resting” time series preprocessing workflow
The noise removal is based on Behzadi et al. (2007)

Parameters

name : name of workflow (default: restpreproc)

Inputs:
**inputspec.func** : functional run (filename or list of filenames)

**Outputs:**

- **outputspec.noise_mask_file** : voxels used for PCA to derive noise components
- **outputspec.filtered_file** : bandpass filtered and noise-reduced time series

**Example**

```python
>>> TR = 3.0
>>> wf = create_resting_preproc()
>>> wf.inputs.inputspec.func = 'f3.nii'
>>> wf.inputs.inputspec.num_noise_components = 6
>>> wf.inputs.inputspec.highpass_sigma = 100/(2*TR)
>>> wf.inputs.inputspec.lowpass_sigma = 12.5/(2*TR)
>>> wf.run() # doctest: +SKIP
```
5.1.3 select_volume()

Link to code
Return the middle index of a file
6.1 workflows.smri.ants.ANTSBuildTemplate

6.1.1 ANTS_TemplateBuildSingleIterationWF()

Link to code

Inputs:

```python
inputs.spec.images :
inputs.spec.fixed_image :
inputs.spec.ListOfPassiveImagesDictionaries :
```

Outputs:

```python
outputs.spec.template :
outputs.spec.transforms_list :
outputs.spec.passive_deformed_templates :
```
6.1.2 FlattenTransformAndImagesList()
Link to code

6.1.3 GetFirstListElement()
Link to code

6.1.4 MakeListsOfTransformLists()
Link to code

6.1.5 MakeTransformListWithGradientWarps()
Link to code

6.1.6 RenestDeformedPassiveImages()
Link to code
6.2 workflows.smri.ants.antsRegistrationBuildTemplate

6.2.1 antsRegistrationTemplateBuildSingleIterationWF()

Link to code

Inputs:

inputspec.images : 
inputspec.fixed_image : 
inputspec.ListOfPassiveImagesDictionaries : 
inputspec.interpolationMapping :

Outputs:

outputspec.template : 
outputspec.transforms_list : 
outputspec.passive_deformed_templates :

Graph

6.2.2 FlattenTransformAndImagesList()

Link to code

6.2.3 GetFirstListElement()

Link to code
6.2.4 GetMovingImages()
Link to code
This currently ONLY works when registrationImageTypes has length of exactly 1. When the new multi-variate registration is introduced, it will be expanded.

6.2.5 GetPassiveImages()
Link to code

6.2.6 MakeTransformListWithGradientWarps()
Link to code

6.2.7 RenestDeformedPassiveImages()
Link to code

6.2.8 SplitAffineAndWarpComponents()
Link to code

6.2.9 makeListOfOneElement()
Link to code

6.3 workflows.smri.freesurfer.autorecon1

6.3.1 checkT1s()
Link to code
Verifying size of inputs and setting workflow parameters

6.3.2 create_AutoRecon1()
Link to code
Creates the AutoRecon1 workflow in nipype.
Inputs: inputspec.T1_files : T1 files (mandatory) inputspec.T2_file : T2 file (optional) inputspec.FLAIR_file : FLAIR file (optional) inputspec.cw256 : Conform inputs to 256 FOV (optional) inputspec.num_threads: Number of threads to use with EM Register (default=1)
Outputs:

6.4 workflows.smri.freesurfer.autorecon2

6.4.1 copy_ltas()
Link to code

6.5 workflows.smri.freesurfer.bem

6.5.1 create_bem_flow()
Link to code
Uses MNE’s Watershed algorithm to create Boundary Element Meshes (BEM) for a subject’s brain, inner/outer skull, and skin. The surfaces are returned in the desired (by default, stereolithic .stl) format.

Example

```python
>>> from nipype.workflows.smri.freesurfer import create_bem_flow
>>> bemflow = create_bem_flow()
>>> bemflow.inputs.inputspec.subject_id = 'subj1'
>>> bemflow.inputs.inputspec.subjects_dir = '.'
>>> bemflow.run()  # doctest: +SKIP
```

Inputs:
- `inputspec.subject_id`: freesurfer subject id
- `inputspec.subjects_dir`: freesurfer subjects directory

Outputs:
- `outputspec.meshes`: output boundary element meshes in (by default) stereolithographic (.stl) format

Graph

```
inputspec (utility) -> WatershedBEM (mne) -> surfconvert (freesurfer) -> outputspec (utility)
```

bem
6.6 workflows.smri.freesurfer.recon

6.6.1 create_reconall_workflow()

Link to code

Creates the ReconAll workflow in Nipype. This workflow is designed to run the same commands as FreeSurfer's reconall script but with the added features that a Nipype workflow provides. Before running this workflow, it is necessary to have the FREESURFER_HOME environmental variable set to the directory containing the version of FreeSurfer to be used in this workflow.

Example

```python
>>> from nipype.workflows.smri.freesurfer import create_reconall_workflow
>>> recon_all = create_reconall.workflow()
>>> recon_all.inputs.inputspec.subject_id = 'subj1'
>>> recon_all.inputs.inputspec.subjects_dir = '.
>>> recon_all.inputs.inputspec.T1_files = 'T1.nii.gz'
>>> recon_all.run()
# doctest: +SKIP
```

**Inputs:**
- inputspec.subjects_dir : subjects directory (mandatory)
- inputspec.subject_id : name of subject (mandatory)
- inputspec.T1_files : T1 files (mandatory)
- inputspec.T2_file : T2 file (optional)
- inputspec.FLAIR_file : FLAIR file (optional)
- inputspec.cw256 : Conform inputs to 256 FOV (optional)
- inputspec.num_threads: Number of threads on nodes that utilize OpenMP (default=1)
- plugin_args : Dictionary of plugin args to set to nodes that utilize OpenMP (optional)

**Outputs:**
- postdatasink_outputspec.subject_id : name of the datasinked output folder in the subjects directory

Note: The input subject_id is not passed to the commands in the workflow. Commands that require subject_id are reading implicit inputs from `{SUBJECTS_DIR}/{subject_id}`. For those commands the subject_id is set to the default value and SUBJECTS_DIR is set to the node directory. The implicit inputs are then copied to the node directory in order to mimic a SUBJECTS_DIR structure. For example, if the command implicitly reads in brainmask.mgz, the interface would copy that input file to `{node_dir}/{subject_id}/mri/brainmask.mgz` and set SUBJECTS_DIR to node_dir. The workflow only uses the input subject_id to datasink the outputs to `{subjects_dir}/{subject_id}`.
6.6.2 create_skullstripped_recon_flow()

Link to code
Performs recon-all on volumes that are already skull stripped. FreeSurfer fails to perform skullstripping on some volumes (especially MP2RAGE). This can be avoided by doing skullstripping before running recon-all (using for example SPECTRE algorithm).

Example

```python
>>> from nipype.workflows.smri.freesurfer import create_skullstripped_recon_flow
>>> recon_flow = create_skullstripped_recon_flow()
>>> recon_flow.inputs.inputspec.subject_id = 'subj1'
>>> recon_flow.inputs.inputspec.T1_files = 'T1.nii.gz'
>>> recon_flow.run() # doctest: +SKIP
```

**Inputs:**
- inputspec.T1_files: skullstripped T1_files (mandatory)
- inputspec.subject_id: freesurfer subject id (optional)
- inputspec.subjects_dir: freesurfer subjects directory (optional)

**Outputs:**
- outputspec.subject_id: freesurfer subject id
- outputspec.subjects_dir: freesurfer subjects directory
6.7 workflows.smri.freesurfer.utils

6.7.1 create_get_stats_flow()

Link to code
Retrieves stats from labels

Parameters

name [string] name of workflow
withreg [boolean] indicates whether to register source to label

Example

Inputs:
inputs.spec.source_file: reference image for mask generation
inputs.spec.label_file: label file from which to get ROIs

(optionally with registration)
inputs.spec.reg_file: bbreg file (assumes reg from source to label
inputs.spec.inverse: boolean whether to invert the registration
inputs.spec.subjects_dir: freesurfer subjects directory

Outputs:

outputs.spec.stats_file: stats file

Graph

6.7.2 create_getmask_flow()

Link to code
Registers a source file to freesurfer space and create a brain mask in source space
Requires fsl tools for initializing registration

Parameters

name [string] name of workflow
dilate_mask [boolean] indicates whether to dilate mask or not

Example

```python
>>> getmask = create_getmask_flow()
>>> getmask.inputs.inputspec.source_file = 'mean.nii'
```
>>> getmask.inputs.inputspec.subject_id = 's1'
>>> getmask.inputs.inputspec.subjects_dir = '.'
>>> getmask.inputs.inputspec.contrast_type = 't2'

Inputs:

| inputspec.source_file : reference image for mask generation |
| inputspec.subject_id : freesurfer subject id |
| inputspec.subjects_dir : freesurfer subjects directory |
| inputspec.contrast_type : MR contrast of reference image |

Outputs:

| outputspec.mask_file : binary mask file in reference image space |
| outputspec.reg_file : registration file that maps reference image to freesurfer space |
| outputspec.reg_cost : cost of registration (useful for detecting misalignment) |
6.7.3 create_tessellation_flow()

Link to code
Tessellates the input subject’s aseg.mgz volume and returns the surfaces for each region in stereolithic (.stl) format
### Example

```python
>>> from nipype.workflows.smri.freesurfer import create_tessellation_flow
>>> tessflow = create_tessellation_flow()
>>> tessflow.inputs.inputspec.subject_id = 'subj1'
>>> tessflow.inputs.inputspec.subjects_dir = '.'
>>> tessflow.inputs.inputspec.lookup_file = 'FreeSurferColorLUT.txt'  # doctest:+SKIP
>>> tessflow.run()  # doctest: +SKIP
```

**Inputs:**

- `inputspec.subject_id`: freesurfer subject id
- `inputspec.subjects_dir`: freesurfer subjects directory
- `inputspec.lookup_file`: lookup file from freesurfer directory

**Outputs:**

- `outputspec.meshes`: output region meshes in (by default) stereolithographic (.stl) format
Graph

inputs (utility)

fssource (io)

volconvert (freesurfer)

region_list_from_volume_node (utility)

id_list_from_lookup_table_node (utility)

tessellate (freesurfer)

surfconvert (freesurfer)

smoother (meshfix)

outputs (utility)

tessellate
6.7.4 copy_file()

Link to code
Create a function to copy a file that can be modified by a following node without changing the original file.

6.7.5 copy_files()

Link to code
Create a function to copy a file that can be modified by a following node without changing the original file.

6.7.6 get_aparc_aseg()

Link to code
Return the aparc+aseg.mgz file.

6.7.7 getdefaultconfig()

Link to code

6.7.8 mkdir_p()

Link to code

6.8 workflows.smri.niftyreg.groupwise

6.8.1 create_groupwise_average()

Link to code
Create the overall workflow that embeds all the rigid, affine and non-linear components.

Inputs:

| inputspec.in_files | The input files to be registered |
| inputspec.ref_file | The initial reference image that the input files are registered to |
| inputspec.rmask_file | Mask of the reference image |
| inputspec.in_trans_files | Initial transformation files (affine or cpps) |

Outputs:

| outputspec.average_image | The average image |
| outputspec.cpp_files | The bspline transformation files |

Example

```python
>>> from nipype.workflows.smri.niftyreg import create_groupwise_average
>>> node = create_groupwise_average('groupwise_av') # doctest: +SKIP
>>> node.inputs.inputspec.in_files = ...
... 'file1.nii.gz', 'file2.nii.gz'] # doctest: +SKIP
>>> node.inputs.inputspec.ref_file = ['ref.nii.gz'] # doctest: +SKIP
>>> node.inputs.inputspec.rmask_file = ['mask.nii.gz'] # doctest: +SKIP
>>> node.run() # doctest: +SKIP
```
Graph
6.8.2 create_linear_gw_step()

Link to code

Creates a workflow that performs linear co-registration of a set of images using RegAladin, producing an average image and a set of affine transformation matrices linking each of the floating images to the average.

Inputs:

<table>
<thead>
<tr>
<th>inputspec.in_files</th>
<th>The input files to be registered</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputspec.ref_file</td>
<td>The initial reference image that the input files are registered to</td>
</tr>
<tr>
<td>inputspec.rmask_file</td>
<td>Mask of the reference image</td>
</tr>
<tr>
<td>inputspec.in_aff_files</td>
<td>Initial affine transformation files</td>
</tr>
</tbody>
</table>

Outputs:

<table>
<thead>
<tr>
<th>outputspec.average_image</th>
<th>The average image</th>
</tr>
</thead>
<tbody>
<tr>
<td>outputspec.aff_files</td>
<td>The affine transformation files</td>
</tr>
</tbody>
</table>

Optional arguments:

<table>
<thead>
<tr>
<th>linear_options_hash</th>
<th>An options dictionary containing a list of parameters for RegAladin that take the same form as given in the interface (default None)</th>
</tr>
</thead>
<tbody>
<tr>
<td>demean</td>
<td>Selects whether to demean the transformation matrices when performing the averaging (default True)</td>
</tr>
<tr>
<td>initial_affines</td>
<td>Selects whether to iterate over initial affine images, which we generally won't have (default False)</td>
</tr>
</tbody>
</table>

Example

```python
>>> from nipype.workflows.smri.niftyreg import create_linear_gw_step
>>> lgw = create_linear_gw_step('my_linear_coreg')  # doctest: +SKIP
>>> lgw.inputs.inputspec.in_files = ['file1.nii.gz', 'file2.nii.gz']  # doctest: +SKIP
>>> lgw.inputs.inputspec.ref_file = ['ref.nii.gz']  # doctest: +SKIP
>>> lgw.run()  # doctest: +SKIP
```
6.8.3 create_nonlinear_gw_step()

Link to code
Creates a workflow that perform non-linear co-registrations of a set of images using RegF3d, producing an non-linear average image and a set of cpp transformation linking each of the floating images to the average.

Inputs:

| inputspec.in_files - The input files to be registered |
| inputspec.ref_file - The initial reference image that the input files are registered to |
| inputspec.rmask_file - Mask of the reference image |
| inputspec.in_trans_files - Initial transformation files (affine or ccpps) |

Outputs:

| outputspec.average_image - The average image |
| outputspec.cpp_files - The bspline transformation files |

Optional arguments:

| nonlinear_options_hash - An options dictionary containing a list of parameters for RegAladin that take the |

(continues on next page)
Example

```python
>>> from nipype.workflows.smri.niftyreg import create_nonlinear_gw_step
>>> nlc = create_nonlinear_gw_step('nonlinear_coreg')  # doctest: +SKIP
>>> nlc.inputs.inputspec.in_files = [
    ...    'file1.nii.gz', 'file2.nii.gz']  # doctest: +SKIP
>>> nlc.inputs.inputspec.ref_file = ['ref.nii.gz']  # doctest: +SKIP
>>> nlc.run()  # doctest: +SKIP
```

Graph

- Examples
CHAPTER 7

dMRI: Camino, DTI

7.1 Introduction

This script, camino_dti_tutorial.py, demonstrates the ability to perform basic diffusion analysis in a Nipype pipeline:

```python
dmri_camino_dti.py
```

We perform this analysis using the FSL course data, which can be acquired from here: http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz

Import necessary modules from nipype.

```python
import os  # system functions
import nipype.interfaces.io as nio  # Data i/o
import nipype.interfaces.utility as util  # utility
import nipype.pipeline.engine as pe  # pipeline engine
import nipype.interfaces.camino as camino
import nipype.interfaces.fsl as fsl
import nipype.interfaces.camino2trackvis as cam2trk
import nipype.algorithms.misc as misc
```

We use the following functions to scrape the voxel and data dimensions of the input images. This allows the pipeline to be flexible enough to accept and process images of varying size. The SPM Face tutorial (fmri_spm_face.py) also implements this inferral of voxel size from the data.

```python
def get_vox_dims(volume):
    import nibabel as nb
    from nipype.utils import NUMPY_MMAP
    if isinstance(volume, list):
        volume = volume[0]
    nii = nb.load(volume, mmap=NUMPY_MMAP)
    hdr = nii.header
    voxdims = hdr.get_zooms()
    return [float(voxdims[0]), float(voxdims[1]), float(voxdims[2])]

def get_data_dims(volume):
    import nibabel as nb
```

(continues on next page)
from nipype.utils import NUMPY_MMAP

if isinstance(volume, list):
    volume = volume[0]
nii = nb.load(volume, mmap=NUMPY_MMAP)
hdr = nii.header
datadims = hdr.get_data_shape()
return [int(datadims[0]), int(datadims[1]), int(datadims[2])]


def get_affine(volume):
    import nibabel as nb
    from nipype.utils import NUMPY_MMAP
    nii = nb.load(volume, mmap=NUMPY_MMAP)
    return nii.affine

subject_list = ['subj1']
fsl.FSLCommand.set_default_output_type('NIFTI')

Map field names to individual subject runs

info = dict(
    dwi=[['subject_id', 'data']],
    bvecs=[['subject_id', 'bvecs']],
    bvals=[['subject_id', 'bvals']])

infosource = pe.Node(interface=util.IdentityInterface(fields=['subject_id']), name="infosource")

Here we set up iteration over all the subjects. The following line is a particular example of the flexibility of the system. The datasource attribute iterables tells the pipeline engine that it should repeat the analysis on each of the items in the subject_list. In the current example, the entire first level preprocessing and estimation will be repeated for each subject contained in subject_list.

infosource.iterables = ('subject_id', subject_list)

Now we create a nipype.interfaces.io.DataGrabber object and fill in the information from above about the layout of our data. The nipype.pipeline.engine.Node module wraps the interface object and provides additional housekeeping and pipeline specific functionality.

datasource = pe.Node(
    interface=nio.DataGrabber(
        infields=['subject_id'], outfields=list(info.keys())),
    name='datasource')

 datasource.inputs.template = "%s/\$s"

# This needs to point to the fdt folder you can find after extracting
# http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz
datasource.inputs.base_directory = os.path.abspath('fsl_course_data/fdt/')

datasource.inputs.field_template = dict(dwi='\$s/\$s.nii.gz')

datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True

An inputnode is used to pass the data obtained by the data grabber to the actual processing functions

inputnode = pe.Node(
interface=util.IdentityInterface(fields=["dwi", "bvecs", "bvals"], name="inputnode")

7.1.1 Setup for Diffusion Tensor Computation

In this section we create the nodes necessary for diffusion analysis. First, the diffusion image is converted to voxel order.

image2voxel = pe.Node(interface=camino.Image2Voxel(), name="image2voxel")
fsl2scheme = pe.Node(interface=camino.FSL2Scheme(), name="fsl2scheme")
fsl2scheme.inputs.usegradmod = True

Second, diffusion tensors are fit to the voxel-order data.

dtifit = pe.Node(interface=camino.DTIFit(), name='dtifit')

Next, a lookup table is generated from the schemefile and the signal-to-noise ratio (SNR) of the unweighted (q=0) data.

dtlutgen = pe.Node(interface=camino.DTLUTGen(), name="dtlutgen")
dtlutgen.inputs.snr = 16.0
dtlutgen.inputs.inversion = 1

In this tutorial we implement probabilistic tractography using the PICo algorithm. PICo tractography requires an estimate of the fibre direction and a model of its uncertainty in each voxel; this is produced using the following node.

picopdfs = pe.Node(interface=camino.PicoPDFs(), name="picopdfs")
picopdfs.inputs.inputmodel = 'dt'

An FSL BET node creates a brain mask is generated from the diffusion image for seeding the PICo tractography.

bet = pe.Node(interface=fsl.BET(), name="bet")
bet.inputs.mask = True

Finally, tractography is performed. First DT streamline tractography.

trackdt = pe.Node(interface=camino.TrackDT(), name="trackdt")

Now camino’s Probabilistic Index of connectivity algorithm. In this tutorial, we will use only 1 iteration for time-saving purposes.

trackpico = pe.Node(interface=camino.TrackPICo(), name="trackpico")
trackpico.inputs.iterations = 1

Currently, the best program for visualizing tracts is TrackVis. For this reason, a node is included to convert the raw tract data to .trk format. Solely for testing purposes, another node is added to perform the reverse.

cam2trk_dt = pe.Node(interface=cam2trk.Camino2Trackvis(), name="cam2trk_dt")
cam2trk_dt.inputs.min_length = 30
cam2trk_dt.inputs.voxel_order = 'LAS'

cam2trk_pico = pe.Node(interface=cam2trk.Camino2Trackvis(), name="cam2trk_pico")
cam2trk_pico.inputs.min_length = 30
cam2trk_pico.inputs.voxel_order = 'LAS'

Tracts can also be converted to VTK and OOGL formats, for use in programs such as GeomView and Paraview, using the following two nodes. For VTK use VtkStreamlines.
procstreamlines = pe.Node(
    interface=camino.ProcStreamlines(), name="procstreamlines")
procstreamlines.inputs.outputtracts = 'oogl'

We can also produce a variety of scalar values from our fitted tensors. The following nodes generate the fractional anisotropy and diffusivity trace maps and their associated headers.

fa = pe.Node(interface=camino.ComputeFractionalAnisotropy(), name='fa')
trace = pe.Node(interface=camino.ComputeTensorTrace(), name='trace')
dteig = pe.Node(interface=camino.ComputeEigensystem(), name='dteig')

analyzeheader_fa = pe.Node(
    interface=camino.AnalyzeHeader(), name="analyzeheader_fa")
analyzeheader_fa.inputs.datatype = "double"
analyzeheader_trace = analyzeheader_fa.clone('analyzeheader_trace')
fa2nii = pe.Node(interface=misc.CreateNifti(), name='fa2nii')
trace2nii = fa2nii.clone("trace2nii")

Since we have now created all our nodes, we can now define our workflow and start making connections.

tractography = pe.Workflow(name='tractography')
tractography.connect([[(inputnode, bet, [("dwi", "in_file")])]])

File format conversion

tractography.connect([[(inputnode, image2voxel, [("dwi", "in_file")]),
    (inputnode, fsl2scheme, [("bvecs", "bvec_file"),
    ("bvals", "bval_file")])]])

Tensor fitting

tractography.connect([[(image2voxel, dtifit, [('voxel_order', 'in_file')]),
    (fsl2scheme, dtifit, [('scheme', 'scheme_file')])]])

Workflow for applying DT streamline tractography

tractography.connect([[(bet, trackdt, [('mask_file', 'seed_file')])]])
tractography.connect([[(dtifit, trackdt, [('tensor_fitted', 'in_file')])]])

Workflow for applying PICo

tractography.connect([[(bet, trackpico, [('mask_file', 'seed_file')])]])
tractography.connect([[(fsl2scheme, dtlutgen, [('scheme', 'scheme_file')])]])
tractography.connect([[(dtifit, picopdfs, [('tensor_fitted', 'in_file')])]])
tractography.connect([[(picopdfs, trackpico, [('pdfs', 'in_file')])]])

# ProcStreamlines might throw memory errors - comment this line out in such case
tractography.connect([[(trackdt, procstreamlines, [('tracked', 'in_file')])]])

Connecting the Fractional Anisotropy and Trace nodes is simple, as they obtain their input from the This is also where our voxel- and data-grabbing functions come in. We pass these functions, along with the original DWI image from the input node, to the header-generating nodes. This ensures that the files
Finally, we create another higher-level workflow to connect our tractography workflow with the info and data-grabbing nodes declared at the beginning. Our tutorial can is now extensible to any arbitrary number of subjects by simply adding their names to the subject list and their data to the proper folders.

```python
workflow = pe.Workflow(name="workflow")
workflow.base_dir = os.path.abspath('camino_dti_tutorial')
workflow.connect([[infosource, datasource, [('subject_id', 'subject_id')]],
                  (datasource, tractography,
                   [('dwi', 'inputnode.dwi'), ('bvals', 'inputnode.bvals'),
                    ('bvecs', 'inputnode.bvecs')]])
```

The following functions run the whole workflow and produce a .dot and .png graph of the processing pipeline.

```python
if __name__ == '__main__':
    workflow.run()
    workflow.write_graph()
```

You can choose the format of the exported graph with the format option. For example `workflow.write_graph(format='eps')`

**Example source code**

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the examples directory.
8.1 Introduction

This script, connectivity_tutorial.py, demonstrates the ability to perform connectivity mapping using Nipype for pipelining, Freesurfer for Reconstruction / Parcellation, Camino for tensor-fitting and tractography, and the Connectome Mapping Toolkit (CMTK) for connectivity analysis:

```
python connectivity_tutorial.py
```

We perform this analysis using the FSL course data, which can be acquired from here:

- http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz

This pipeline also requires the Freesurfer directory for ‘subj1’ from the FSL course data. To save time, this data can be downloaded from here:

- http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz

A data package containing the outputs of this pipeline can be obtained from here:

- http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz

Along with Camino, Camino-Trackvis, FSL, and Freesurfer, you must also have the Connectome File Format library installed as well as the Connectome Mapper. These are written by Stephan Gerhard and can be obtained from:

- http://www.cmtk.org/
- http://www.cmtk.org/

Or on github at:

- CFFLib: https://github.com/LTS5/cfflib
- CMP: https://github.com/LTS5/cmp

Output data can be visualized in the ConnectomeViewer

ConnectomeViewer: https://github.com/LTS5/connectomeviewer

First, we import the necessary modules from nipype.

```
import inspect
import os.path as op  # system functions
import cmp  # connectome mapper
import nipype.interfaces.io as nio  # Data i/o
import nipype.interfaces.utility as util  # utility
import nipype.pipeline.engine as pe  # pypeline engine
import nipype.interfaces.camino as camino
import nipype.interfaces.fsl as fsl
import nipype.interfaces.camino2trackvis as cam2trk
```

(continues on next page)
We define the following functions to scrape the voxel and data dimensions of the input images. This allows the pipeline to be flexible enough to accept and process images of varying size. The SPM Face tutorial (fmri_spm_face.py) also implements this inferal of voxel size from the data. We also define functions to select the proper parcellation/segregation file from Freesurfer’s output for each subject. For the mapping in this tutorial, we use the aparc+seg.mgz file. While it is possible to change this to use the regions defined in aparc.a2009s+aseg.mgz, one would also have to write/obtain a network resolution map defining the nodes based on those.

```python
def get_vox_dims(volume):
    import nibabel as nb
    from nipype.utils import NUMPY_MMAP
    if isinstance(volume, list):
        volume = volume[0]
    nii = nb.load(volume, mmap=NUMPY_MMAP)
    hdr = nii.header
    vxdims = hdr.get_zooms()
    return [float(vxdims[0]), float(vxdims[1]), float(vxdims[2])]

def get_data_dims(volume):
    import nibabel as nb
    from nipype.utils import NUMPY_MMAP
    if isinstance(volume, list):
        volume = volume[0]
    nii = nb.load(volume, mmap=NUMPY_MMAP)
    hdr = nii.header
    datadims = hdr.get_data_shape()
    return [int(datadims[0]), int(datadims[1]), int(datadims[2])]

def get_affine(volume):
    import nibabel as nb
    from nipype.utils import NUMPY_MMAP
    nii = nb.load(volume, mmap=NUMPY_MMAP)
    return nii.affine

def select_aparc(list_of_files):
    for in_file in list_of_files:
        if 'aparc+aseg.mgz' in in_file:
            idx = list_of_files.index(in_file)
    return list_of_files[idx]

def select_aparc_annot(list_of_files):
    for in_file in list_of_files:
        if '.aparc.annot' in in_file:
            idx = list_of_files.index(in_file)
    return list_of_files[idx]
```

These need to point to the main Freesurfer directory as well as the freesurfer subjects directory. No assumptions are made about where the directory of subjects is placed. Recon-all must have been run on subj1 from the FSL course data.
This needs to point to the fdt folder you can find after extracting http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz.

An infosource node is used to loop through the subject list and define the input files. For our purposes, these are the diffusion-weighted MR image, b vectors, and b values. The info dictionary is used to provide a template of the naming of these files. For instance, the 4D nifti diffusion image is stored in the FSL course data as data.nii.gz.

A datasource node is used to perform the actual data grabbing. Templates for the associated images are used to obtain the correct images. The data are assumed to lie in data_dir/subject_id/.

FreeSurferSource nodes are used to retrieve a number of image files that were automatically generated by the recon-all process. Here we use three of these nodes, two of which are defined to return files for solely the left and right hemispheres.

Since the b values and b vectors come from the FSL course, we must convert it to a scheme file for use in Camino.
FSL's Brain Extraction tool is used to create a mask from the b0 image

FSL's FLIRT function is used to coregister the b0 mask and the structural image. A convert_xfm node is then used to obtain the inverse of the transformation matrix. FLIRT is used once again to apply the inverse transformation to the parcellated brain image.

A number of conversion operations are required to obtain NIFTI files from the FreesurferSource for each subject. Nodes are used to convert the following:
- Original structural image to NIFTI
- Parcellated white matter image to NIFTI
- Parcellated whole-brain image to NIFTI
- Pial, white, inflated, and spherical surfaces for both the left and right hemispheres are converted to GIFTI for visualization in ConnectomeViewer
- Parcellated annotation files for the left and right hemispheres are also converted to GIFTI

An inputnode is used to pass the data obtained by the data grabber to the actual processing functions.
image2voxel = pe.Node(interface=camino.Image2Voxel(), name="image2voxel")

Second, diffusion tensors are fit to the voxel-order data. If desired, these tensors can be converted to a Nifti tensor image using the DT2NIITI interface.

dtifit = pe.Node(interface=camino.DTIFit(), name='dtifit')

Next, a lookup table is generated from the schemefile and the signal-to-noise ratio (SNR) of the unweighted (q=0) data.

dtlutgen = pe.Node(interface=camino.DTLUTGen(), name="dtlutgen")
dtlutgen.inputs.snr = 16.0
dtlutgen.inputs.inversion = 1

In this tutorial we implement probabilistic tractography using the PICo algorithm. PICo tractography requires an estimate of the fibre direction and a model of its uncertainty in each voxel; this probability distribution map is produced using the following node.

picopdfs = pe.Node(interface=camino.PicoPDFs(), name="picopdfs")
picopdfs.inputs.inputmodel = 'dt'

Finally, tractography is performed. In this tutorial, we will use only one iteration for time-saving purposes. It is important to note that we use the TrackPICo interface here. This interface now expects the files required for PICo tracking (i.e. the output from picopdfs). Similar interfaces exist for alternative types of tracking, such as Bayesian tracking with Dirac priors (TrackBayesDirac).

track = pe.Node(interface=camino.TrackPICo(), name="track")
track.inputs.iterations = 1

Currently, the best program for visualizing tracts is TrackVis. For this reason, a node is included to convert the raw tract data to .trk format. Solely for testing purposes, another node is added to perform the reverse.

camino2trackvis = pe.Node(
    interface=cam2trk.Camino2Trackvis(), name="camino2trk")
camino2trackvis.inputs.min_length = 30
camino2trackvis.inputs.voxel_order = 'LAS'
trk2camino = pe.Node(interface=cam2trk.Trackvis2Camino(), name="trk2camino")

Tracts can also be converted to VTK and OOGL formats, for use in programs such as GeomView and Paraview, using the following two nodes.

vtkstreamlines = pe.Node(
    interface=camino.VtkStreamlines(), name="vtkstreamlines")
procstreamlines = pe.Node(
    interface=camino.ProcStreamlines(), name="procstreamlines")
procstreamlines.inputs.outputtracts = 'oogl'

We can easily produce a variety of scalar values from our fitted tensors. The following nodes generate the fractional anisotropy and diffusivity trace maps and their associated headers, and then merge them back into a single .nii file.

fa = pe.Node(interface=camino.ComputeFractionalAnisotropy(), name='fa')
trace = pe.Node(interface=camino.ComputeTensorTrace(), name='trace')
dteig = pe.Node(interface=camino.ComputeEigensystem(), name='dteig')

analyzeheader_fa = pe.Node(
    interface=camino.AnalyzeHeader(), name='analyzeheader_fa'
)analyzeheader_fa.inputs.datatype = 'double'
analyzeheader_trace = pe.Node(
    interface=camino.AnalyzeHeader(), name='analyzeheader_trace'
)

(continues on next page)
analyzeheader_trace.inputs.datatype = 'double'

fa2nii = pe.Node(interface=misc.CreateNifti(), name='fa2nii')
trace2nii = fa2nii.clone("trace2nii")

This section adds the Connectome Mapping Toolkit (CMTK) nodes. These interfaces are fairly experimental and may not function properly. In order to perform connectivity mapping using CMTK, the parcellated structural data is rewritten using the indices and parcellation scheme from the connectome mapper (CMP). This process has been written into the ROIGen interface, which will output a remapped aparc+aseg image as well as a dictionary of label information (i.e. name, display colours) pertaining to the original and remapped regions. These label values are input from a user-input lookup table, if specified, and otherwise the default Freesurfer LUT (/freesurfer/FreeSurferColorLUT.txt).

roigen = pe.Node(interface=cmtk.ROIGen(), name="ROIGen")
cmp_config = cmp.configuration.PipelineConfiguration(
    parcellation_scheme="NativeFreesurfer")
cmp_config.parcellation_scheme = "NativeFreesurfer"
roigen.inputs.LUT_file = cmp_config.get_freeview_lut("NativeFreesurfer")['freesurferaparc']
roigen_structspace = roigen.clone('ROIGen_structspace')

The CreateMatrix interface takes in the remapped aparc+aseg image as well as the label dictionary and fiber tracts and outputs a number of different files. The most important of which is the connectivity network itself, which is stored as a ‘gpickle’ and can be loaded using Python’s NetworkX package (see CreateMatrix docstring). Also outputted are various NumPy arrays containing detailed tract information, such as the start and endpoint regions, and statistics on the mean and standard deviation for the fiber length of each connection. These matrices can be used in the ConnectomeViewer to plot the specific tracts that connect between user-selected regions.

creatematrix = pe.Node(interface=cmtk.CreateMatrix(), name="CreateMatrix")
creatematrix.inputs.count_region_intersections = True
createnodes = pe.Node(interface=cmtk.CreateNodes(), name="CreateNodes")
createnodes.inputs.resolution_network_file = cmp_config.parcellation['freesurferaparc']['node_information_graphml']

Here we define the endpoint of this tutorial, which is the CFFConverter node, as well as a few nodes which use the Nipype Merge utility. These are useful for passing lists of the files we want packaged in our CFF file.

CFFConverter = pe.Node(interface=cmtk.CFFConverter(), name="CFFConverter")
giftiSurfaces = pe.Node(interface=util.Merge(8), name="GiftiSurfaces")
giftiLabels = pe.Node(interface=util.Merge(2), name="GiftiLabels")
niftiVolumes = pe.Node(interface=util.Merge(3), name="NiftiVolumes")
fiberDataArrays = pe.Node(interface=util.Merge(4), name="FiberDataArrays")
gpickledNetworks = pe.Node(interface=util.Merge(1), name="NetworkFiles")

Since we have now created all our nodes, we can define our workflow and start making connections.

mapping = pe.Workflow(name='mapping')

First, we connect the input node to the early conversion functions. FreeSurfer input nodes:

mapping.connect([(inputnode, FreeSurferSource, ["subject_id"], "subject_id")])
mapping.connect([(inputnode, FreeSurferSourceLH, ["subject_id"], "subject_id")])
mapping.connect([(inputnode, FreeSurferSourceRH, ["subject_id"], "subject_id")])

Required conversions for processing in Camino:
Nifti conversions for the parcellated white matter image (used in Camino’s conmap), and the subject’s stripped brain image from Freesurfer:

```python
mapping.connect([(FreeSurferSource, mri_convert_WMParc, [('wmparc', 'in_file')])])
mapping.connect([(FreeSurferSource, mri_convert_Brain, [('brain', 'in_file')])])
```

Surface conversions to GIFTI (pial, white, inflated, and sphere for both hemispheres)

```python
mapping.connect([(FreeSurferSourceLH, mris_convertLH, [('pial', 'in_file')])])
mapping.connect([(FreeSurferSourceRH, mris_convertRH, [('pial', 'in_file')])])
mapping.connect([(FreeSurferSourceLH, mris_convertLHwhite, [('white', 'in_file')])])
mapping.connect([(FreeSurferSourceRH, mris_convertRHwhite, [('white', 'in_file')])])
mapping.connect([(FreeSurferSourceLH, mris_convertLHinflated, [('inflated', 'in_file')])])
mapping.connect([(FreeSurferSourceRH, mris_convertRHinflated, [('inflated', 'in_file')])])
mapping.connect([(FreeSurferSourceLH, mris_convertLHsphere, [('sphere', 'in_file')])])
mapping.connect([(FreeSurferSourceRH, mris_convertRHsphere, [('sphere', 'in_file')])])
```

The annotation files are converted using the pial surface as a map via the MRIsConvert interface. One of the functions defined earlier is used to select the lh.aparc.annot and rh.aparc.annot files specifically (rather than i.e. rh.aparc.a2009s.annot) from the output list given by the FreeSurferSource.

```python
mapping.connect([(FreeSurferSourceLH, mris_convertLHlabels, [('pial', 'in_file')])])
mapping.connect([(FreeSurferSourceRH, mris_convertRHlabels, [('pial', 'in_file')])])
mapping.connect([(FreeSurferSourceLH, mris_convertLHlabels, [('annot', select_aparc_annot), 'annot_file'])])
mapping.connect([(FreeSurferSourceRH, mris_convertRHlabels, [('annot', select_aparc_annot), 'annot_file'])])
```

This section coregisters the diffusion-weighted and parcellated white-matter / whole brain images. At present the conmap node connection is left commented, as there have been recent changes in Camino code that have presented some users with errors.

```python
mapping.connect([(inputnode, b0Strip, ['dwi', 'in_file'])])
mapping.connect([(b0Strip, coregister, ['out_file', 'in_file'])])
mapping.connect([(mri_convert_Brain, coregister, ['out_file', 'reference'])])
mapping.connect([(coregister, convertxfm, ['out_matrix_file', 'in_file'])])
mapping.connect([(b0Strip, inverse, ['out_file', 'reference'])])
mapping.connect([(convertxfm, inverse, ['out_file', 'in_matrix_file'])])
mapping.connect([(mri_convert_WMParc, inverse, ['out_file', 'in_file'])])
```

The tractography pipeline consists of the following nodes. Further information about the tractography can be found in nipype/examples/dmri_camino_dti.py.
Connecting the Fractional Anisotropy and Trace nodes is simple, as they obtain their input from the tensor fitting. This is also where our voxel- and data-grabbing functions come in. We pass these functions, along with the original DWI image from the input node, to the header-generating nodes. This ensures that the files will be correct and readable.

The output tracts are converted to Trackvis format (and back). Here we also use the voxel- and data-grabbing functions defined at the beginning of the pipeline.

Here the CMTK connectivity mapping nodes are connected. The original aparc+aseg image is converted to NIFTI, then registered to the diffusion image and delivered to the ROIgen node. The remapped parcellation, original tracts, and label file are then given to CreateMatrix.
mapping.connect([(roigen_structspace, createnodes, ["roi_file", "roi_file")])
mapping.connect([(inverse_AparcAseg, roigen, ["out_file", "aparc_aseg_file")])
mapping.connect([(roigen, creatematrix, ["roi_file", "roi_file")])
mapping.connect([(camino2trackvis, creatematrix, ["track_file", "tract_file")])
mapping.connect([(inputnode, creatematrix, ["subject_id", "out_matrix_file")])
mapping.connect([(inputnode, creatematrix, ["subject_id", "out_matrix_mat_file")])

The merge nodes defined earlier are used here to create lists of the files which are destined for the CFFConverter.

mapping.connect([(creatematrix, gpickledNetworks, ["matrix_files", "in1")])
mapping.connect([(mris_convertLH, giftiSurfaces, ["converted", "in1")])
mapping.connect([(mris_convertRH, giftiSurfaces, ["converted", "in2")])
mapping.connect([(mris_convertLHwhite, giftiSurfaces, ["converted", "in3")])
mapping.connect([(mris_convertRHwhite, giftiSurfaces, ["converted", "in4")])
mapping.connect([(mris_convertLHinflated, giftiSurfaces, ["converted", "in5")])
mapping.connect([(mris_convertRHinflated, giftiSurfaces, ["converted", "in6")])
mapping.connect([(mris_convertLHsphere, giftiSurfaces, ["converted", "in7")])
mapping.connect([(mris_convertRHsphere, giftiSurfaces, ["converted", "in8")])

mapping.connect([(creatematrix, fiberDataArrays, ["endpoint_file", "in1")])
mapping.connect([(creatematrix, fiberDataArrays, ["endpoint_file_mm", "in2")])
mapping.connect([(creatematrix, fiberDataArrays, ["fiber_length_file", "in3")])
mapping.connect([(creatematrix, fiberDataArrays, ["fiber_label_file", "in4")])

This block actually connects the merged lists to the CFF converter. We pass the surfaces and volumes that are to be included, as well as the tracts and the network itself. The currently running pipeline (dmri_connectivity.py) is also scraped and included in the CFF file. This makes it easy for the user to examine the entire processing pathway used to generate the end product.

CFFConverter.inputs.script_files = op.abspath(
    inspect.getfile(inspect.currentframe()))
mapping.connect([(giftiSurfaces, CFFConverter, ["out", "gifti_surfaces")])
mapping.connect([(giftiLabels, CFFConverter, ["out", "gifti_labels")])
mapping.connect([(gpickledNetworks, CFFConverter, ["out", "gpickled_networks")])
mapping.connect([(niftiVolumes, CFFConverter, ["out", "nifti_volumes")])

8.1. Introduction
Finally, we create another higher-level workflow to connect our mapping workflow with the info and datagrabbing nodes declared at the beginning. Our tutorial can is now extensible to any arbitrary number of subjects by simply adding their names to the subject list and their data to the proper folders.

```python
connectivity = pe.Workflow(name="connectivity")
connectivity.base_dir = op.abspath('dmri_connectivity')
connectivity.connect([(infosource, datasource, [('subject_id', 'subject_id')]),
                      (datasource, mapping, [('dwi', 'inputnode.dwi'), ('bvals', 'inputnode.bvals'), ('bvecs', 'inputnode.bvecs')]),
                      (infosource, mapping, [('subject_id', 'inputnode.subject_id')])])
```

The following functions run the whole workflow and produce graphs describing the processing pipeline. By default, write_graph outputs a .dot file and a .png image, but here we set it to output the image as a vector graphic, by passing the format='eps' argument.

```python
if __name__ == '__main__':
    connectivity.run()
    connectivity.write_graph(format='eps')
```

The output CFF file of this pipeline can be loaded in the Connectome Viewer. After loading the network into memory it can be examined in 3D or as a connectivity matrix using the default scripts produced by the Code Oracle. To compare networks, one must use the MergeCNetworks interface to merge two networks into a single CFF file. Statistics can then be run using the Network Brain Statistics (NBS) plugin. Surfaces can also be loaded along with their labels from the aparc+aseg file. The tractography is included in the file so that region-to-region fibers can be individually plotted using the Code Oracle.

**Example source code**
You can download the full source code of this example. This same script is also included in the Nipype source distribution under the examples directory.
9.1 Introduction

This script, connectivity_tutorial_advanced.py, demonstrates the ability to perform connectivity mapping using Nipype for pipelining, Freesurfer for Reconstruction / Segmentation, MRtrix for spherical deconvolution and tractography, and the Connectome Mapping Toolkit (CMTK) for further parcellation and connectivity analysis:

```
python connectivity_tutorial_advanced.py
```

We perform this analysis using the FSL course data, which can be acquired from here:
- [http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz](http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz)

This pipeline also requires the Freesurfer directory for ‘subj1’ from the FSL course data. To save time, this data can be downloaded from here:

The result of this processing will be the connectome for subj1 as a Connectome File Format (CFF) File, using the Lausanne2008 parcellation scheme. A data package containing the outputs of this pipeline can be obtained from here:
- [http://db.tt/909Q3AC1](http://db.tt/909Q3AC1)

See also:
- [connectivity_tutorial.py](http://www.cmtk.org)  Original tutorial using Camino and the NativeFreesurfer Parcellation Scheme
- [www.cmtk.org](http://www.cmtk.org)  For more info about the parcellation scheme

**Warning:** The ConnectomeMapper ([https://github.com/LTS5/cmp](https://github.com/LTS5/cmp) or [www.cmtk.org](http://www.cmtk.org)) must be installed for this tutorial to function!

9.2 Packages and Data Setup

Import necessary modules from nipype.

```
import nipype.interfaces.io as nio  # Data i/o
import nipype.interfaces.utility as util  # utility
import nipype.pipeline.engine as pe  # pipeline engine
import nipype.interfaces.fsl as fsl
import nipype.interfaces.freesurfer as fs
import nipype.interfaces.mrtrix as mrtrix
```

(continues on next page)
import nipype.algorithms.misc as misc
import nipype.interfaces.cmtk as cmtk
import nipype.interfaces.dipy as dipy
import inspect
import os
import os.path as op
# system functions
from nipype.workflows.dmri.fsl.dti import create_eddy_correct_pipeline
from nipype.workflows.dmri.camino.connectivity_mapping import select_aparc_annot
from nipype.utils.misc import package_check
import warnings
from nipype.workflows.dmri.connectivity.nx import create_networkx_pipeline,
    create_cmats_to_csv_pipeline
from nipipe.workflows.smri.freesurfer import create_tessellation_flow

try:
    package_check('cmp')
except Exception as e:
    warnings.warn('cmp not installed')
else:
    import cmp

This needs to point to the freesurfer subjects directory (Recon-all must have been run on subj1 from the FSL
course data) Alternatively, the reconstructed subject data can be downloaded from:


subjects_dir = op.abspath(op.join(op.curdir, './subjects'))
fs.FSCommand.set_default_subjects_dir(subjects_dir)
fs.FSLCommand.set_default_output_type('NIFTI')

fs_dir = os.environ['FREESURFER_HOME']
lookup_file = op.join(fs_dir, 'FreeSurferColorLUT.txt')

This needs to point to the fdt folder you can find after extracting

- http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz

data_dir = op.abspath(op.join(op.curdir, 'exdata/'))
subject_list = ['subj1']

Use infosource node to loop through the subject list and define the input files. For our purposes, these are the
diffusion-weighted MR image, b vectors, and b values.

infosource = pe.Node(
    interface=util.IdentityInterface(fields=['subject_id']), name="infosource")
infosource.iterables = ('subject_id', subject_list)

info = dict(
    dwi=[['subject_id', 'data']],
    bvecs=[['subject_id', 'bvecs']],
    bvals=[['subject_id', 'bvals']])

Use datasource node to perform the actual data grabbing. Templates for the associated images are used to obtain
the correct images.

datasource = pe.Node(
    interface=nio.DataGrabber(
        infields=['subject_id'], outfields=list(info.keys())),
    name='datasource')
datasource.inputs.template = "%s/%s"
datasource.inputs.base_directory = data_dir
datasource.inputs.field_template = dict(dwi='%(dwi)s/%s.nii.gz')
datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True

The input node and Freesurfer sources declared here will be the main conduits for the raw data to the rest of the processing pipeline.

inputnode = pe.Node(
    interface=util.IdentityInterface(
        fields=['subject_id', 'dwi', 'bvecs', 'bvals', 'subjects_dir'],
        name='inputnode')
inputnode.inputs.subjects_dir = subjects_dir

FreeSurferSource = pe.Node(interface=nio.FreeSurferSource(), name='fssource')
FreeSurferSourceLH = FreeSurferSource.clone('fssourceLH')
FreeSurferSourceLH.inputs.hemi = 'lh'
FreeSurferSourceRH = FreeSurferSource.clone('fssourceRH')
FreeSurferSourceRH.inputs.hemi = 'rh'

9.3 Creating the workflow’s nodes

9.3.1 Conversion nodes

A number of conversion operations are required to obtain NIFTI files from the FreesurferSource for each subject. Nodes are used to convert the following:

• Original structural image to NIFTI
• Pial, white, inflated, and spherical surfaces for both the left and right hemispheres are converted to GIFTI for visualization in ConnectomeViewer
• Parcellated annotation files for the left and right hemispheres are also converted to GIFTI

mri_convert_Brain = pe.Node(
    interface=fs.MRIConvert(), name='mri_convert_Brain')
mri_convert_Brain.inputs.out_type = 'nii'
mri_convert_ROI_scale500 = mri_convert_Brain.clone('mri_convert_ROI_scale500')
mris_convertLH = pe.Node(interface=fs.MRIsConvert(), name='mris_convertLH')
mris_convertLH.inputs.out_datatype = 'gii'
mris_convertRH = mris_convertLH.clone('mris_convertRH')
mris_convertRWhite = mris_convertLH.clone('mris_convertRWhite')
mris_convertRWhite = mris_convertLH.clone('mris_convertRWhite')
mris_convertRHinflated = mris_convertLH.clone('mris_convertRHinflated')
mris_convertRHsphere = mris_convertLH.clone('mris_convertRHsphere')
mris_convertRlabels = mris_convertLH.clone('mris_convertRlabels')
mris_convertRlabels = mris_convertLH.clone('mris_convertRlabels')

9.3.2 Diffusion processing nodes

See also:
dmri_mrtrix_dti.py Tutorial that focuses solely on the MRtrix diffusion processing
b-values and b-vectors stored in FSL’s format are converted into a single encoding file for MRTrix.
Distortions induced by eddy currents are corrected prior to fitting the tensors. The first image is used as a reference for which to warp the others.

Tensors are fitted to each voxel in the diffusion-weighted image and from these three maps are created:
- Major eigenvector in each voxel
- Apparent diffusion coefficient
- Fractional anisotropy

These nodes are used to create a rough brain mask from the b0 image. The b0 image is extracted from the original diffusion-weighted image, put through a simple thresholding routine, and smoothed using a 3x3 median filter.

The brain mask is also used to help identify single-fiber voxels. This is done by passing the brain mask through two erosion steps, multiplying the remaining mask with the fractional anisotropy map, and thresholding the result to obtain some highly anisotropic within-brain voxels.

For whole-brain tracking we also require a broad white-matter seed mask. This is created by generating a white matter mask, given a brainmask, and thresholding it at a reasonably high level.

The spherical deconvolution step depends on the estimate of the response function in the highly anisotropic voxels we obtained above.
Warning: For damaged or pathological brains one should take care to lower the maximum harmonic order of these steps.

```python
estimateresponse = pe.Node(
    interface=mrtrix.EstimateResponseForSH(), name='estimateresponse')
estimateresponse.inputs.maximum_harmonic_order = 6
csdeconv = pe.Node(
    interface=mrtrix.ConstrainedSphericalDeconvolution(), name='csdeconv')
csdeconv.inputs.maximum_harmonic_order = 6
```

Finally, we track probabilistically using the orientation distribution functions obtained earlier. The tracts are then used to generate a tract-density image, and they are also converted to TrackVis format.

```python
probCSDstreamtrack = pe.Node(
    interface=mrtrix.ProbabilisticSphericallyDeconvolutedStreamlineTrack(),
    name='probCSDstreamtrack')
probCSDstreamtrack.inputs.inputmodel = 'SD_PROB'
probCSDstreamtrack.inputs.desired_number_of_tracks = 150000
tracks2prob = pe.Node(interface=mrtrix.Tracks2Prob(), name='tracks2prob')
tracks2prob.inputs.colour = True
MRconvert_tracks2prob = MRconvert_fa.clone(name='MRconvert_tracks2prob')
tck2trk = pe.Node(interface=dipy.MTrix2TrackVis(), name='tck2trk')
trk2tdi = pe.Node(interface=dipy.TrackDensityMap(), name='trk2tdi')
```

### 9.3.3 Structural segmentation nodes

The following node identifies the transformation between the diffusion-weighted image and the structural image. This transformation is then applied to the tracts so that they are in the same space as the regions of interest.

```python
coregister = pe.Node(interface=fsl.FLIRT(dof=6), name='coregister')
coregister.inputs.cost = ('normmi')
```

Parcellation is performed given the aparc+aseg image from Freesurfer. The CMTK Parcellation step subdivides these regions to return a higher-resolution parcellation scheme. The parcellation used here is entitled “scale500” and returns 1015 regions.

```python
parcellation_name = 'scale500'
parcellate = pe.Node(interface=cmtk.Parcellate(), name="Parcellate")
parcellate.inputs.parcellation_name = parcellation_name
```

The CreateMatrix interface takes in the remapped aparc+aseg image as well as the label dictionary and fiber tracts and outputs a number of different files. The most important of which is the connectivity network itself, which is stored as a ‘gpickle’ and can be loaded using Python’s NetworkX package (see CreateMatrix docstring). Also outputted are various NumPy arrays containing detailed tract information, such as the start and endpoint regions, and statistics on the mean and standard deviation for the fiber length of each connection. These matrices can be used in the ConnectomeViewer to plot the specific tracts that connect between user-selected regions. Here we choose the Lausanne2008 parcellation scheme, since we are incorporating the CMTK parcellation step.

```python
parcellation_name = 'scale500'
 cmp_config = cmp.configuration.PipelineConfiguration()
 cmp_config.parcellation_scheme = "Lausanne2008"
 createnodes = pe.Node(interface=cmtk.CreateNodes(), name="CreateNodes")
 createnodes.inputs.resolution_network_file = cmp_config._get_lausanne_parcellation('Lausanne2008')[parcellation_name]['node_information_graphml']
```

(continues on next page)
Next we define the endpoint of this tutorial, which is the CFFConverter node, as well as a few nodes which use the Nipype Merge utility. These are useful for passing lists of the files we want packaged in our CFF file. The inspect.getfile command is used to package this script into the resulting CFF file, so that it is easy to look back at the processing parameters that were used.

```python
creatematrix = pe.Node(interface=cmtk.CreateMatrix(), name="CreateMatrix")
creatematrix.inputs.count_region_intersections = True

CFFConverter = pe.Node(interface=cmtk.CFFConverter(), name="CFFConverter")
CFFConverter.inputs.script_files = op.abspath(
    inspect.getfile(inspect.currentframe()))
giftiSurfaces = pe.Node(interface=util.Merge(9), name="GiftiSurfaces")
giftiLabels = pe.Node(interface=util.Merge(2), name="GiftiLabels")
niftiVolumes = pe.Node(interface=util.Merge(3), name="NiftiVolumes")
fiberDataArrays = pe.Node(interface=util.Merge(4), name="FiberDataArrays")
gpickledNetworks = pe.Node(interface=util.Merge(2), name="NetworkFiles")
```

We also create a workflow to calculate several network metrics on our resulting file, and another CFF converter which will be used to package these networks into a single file.

```python
networkx = create_networkx_pipeline(name='networkx')
cmats_to_csv = create_cmats_to_csv_pipeline(name='cmats_to_csv')
NxStatsCFFConverter = pe.Node(
    interface=cmtk.CFFConverter(), name="NxStatsCFFConverter")
NxStatsCFFConverter.inputs.script_files = op.abspath(
    inspect.getfile(inspect.currentframe()))
tessflow = create_tessellation_flow(name='tessflow', out_format='gii')
tessflow.inputs.inputspec.lookup_file = lookup_file
```

### 9.4 Connecting the workflow

Here we connect our processing pipeline.

#### 9.4.1 Connecting the inputs, FreeSurfer nodes, and conversions

```python
mapping = pe.Workflow(name='mapping')
```

First, we connect the input node to the FreeSurfer input nodes.

```python
mapping.connect([(inputnode, FreeSurferSource, ["subjects_dir", "subjects_dir"])])
mapping.connect([(inputnode, FreeSurferSource, ["subject_id", "subject_id"])])
mapping.connect([(inputnode, FreeSurferSourceLH, ["subjects_dir", "subjects_dir"])])
mapping.connect([(inputnode, FreeSurferSourceLH, ["subject_id", "subject_id"])])
mapping.connect([(inputnode, FreeSurferSourceRH, ["subjects_dir", "subjects_dir"])])
mapping.connect([(inputnode, FreeSurferSourceRH, ["subject_id", "subject_id"])])
```
mapping.connect([(inputnode, tessflow, [('subjects_dir', 'inputspec.subjects_dir')])])
mapping.connect([(inputnode, tessflow, [('subject_id', 'inputspec.subject_id')])])

Nifti conversion for subject’s stripped brain image from Freesurfer:

mapping.connect([(FreeSurferSource, mri_convert_Brain, [('brain', 'in_file')])])

Surface conversions to GIFTI (pial, white, inflated, and sphere for both hemispheres)

mapping.connect([(FreeSurferSourceLH, mris_convertLH, [('pial', 'in_file')])])
mapping.connect([(FreeSurferSourceRH, mris_convertRH, [('pial', 'in_file')])])
mapping.connect([(FreeSurferSourceLH, mris_convertLHwhite, [('white', 'in_file')])])
mapping.connect([(FreeSurferSourceRH, mris_convertRHwhite, [('white', 'in_file')])])
mapping.connect([(FreeSurferSourceLH, mris_convertLHinflated, [('inflated', 'in_file')])])
mapping.connect([(FreeSurferSourceRH, mris_convertRHinflated, [('inflated', 'in_file')])])
mapping.connect([(FreeSurferSourceLH, mris_convertLHsphere, [('sphere', 'in_file')])])
mapping.connect([(FreeSurferSourceRH, mris_convertRHsphere, [('sphere', 'in_file')])])

The annotation files are converted using the pial surface as a map via the MRIsConvert interface. One of the functions defined earlier is used to select the lh.aparc.annot and rh.aparc.annot files specifically (rather than e.g. rh.aparc.a2009s.annot) from the output list given by the FreeSurferSource.

mapping.connect([(FreeSurferSourceLH, mris_convertLHlabels, [('pial', 'in_file')])])
mapping.connect([(FreeSurferSourceRH, mris_convertRHlabels, [('pial', 'in_file')])])
mapping.connect([(FreeSurferSourceLH, mris_convertLHlabels, [('annot', select_aparc_annot), 'annot_file'])])
mapping.connect([(FreeSurferSourceRH, mris_convertRHlabels, [('annot', select_aparc_annot), 'annot_file'])])

**9.4.2 Diffusion Processing**

Now we connect the tensor computations:

mapping.connect([(inputnode, fsl2mrtrix, [('bvecs', 'bvec_file'), ('bvals', 'bval_file')])])
mapping.connect([(inputnode, eddycorrect, [('dwi', 'inputnode.in_file')])])
mapping.connect([(eddycorrect, dwi2tensor, [('outputnode.eddy_corrected', 'in_file')])])
mapping.connect([(fsl2mrtrix, dwi2tensor, [('encoding_file', 'encoding_file')])])
This block creates the rough brain mask to be multiplied, multiplies it with the fractional anisotropy image, and thresholds it to get the single-fiber voxels.

Here the thresholded white matter mask is created for seeding the tractography.

Next we estimate the fiber response distribution.

Run constrained spherical deconvolution.

Connect the tractography and compute the tract density image.
9.4.3 Structural Processing

First, we coregister the diffusion image to the structural image

```python
mapping.connect([(eddy_correct, coregister, [('outputnode.eddy_corrected', 'in_file')])])
mapping.connect([(mri_convert_Brain, coregister, ['out_file', 'reference'])])
```

The MRtrix-tracked fibers are converted to TrackVis format (with voxel and data dimensions grabbed from the DWI). The connectivity matrix is created with the transformed .trk fibers and the parcellation file.

```python
mapping.connect([(eddy_correct, tck2trk, ['outputnode.eddy_corrected', 'image_file'])])
mapping.connect([(mri_convert_Brain, tck2trk, ['out_file', 'registration_image_file'])])
mapping.connect([(coregister, tck2trk, ['out_file', 'matrix_file'])])
mapping.connect([(probCSDstreamtrack, tck2trk, ['tracked', 'in_file'])])
mapping.connect([(tck2trk, creatematrix, ['subject_id', 'out_matrix_file'])])
mapping.connect([(tck2trk, creatematrix, ['subject_id', 'out_matrix_mat_file'])])
mapping.connect([(parcellate, creatematrix, ['roi_file', 'roi_file'])])
mapping.connect([(parcellate, createnodes, ['roi_file', 'roi_file'])])
mapping.connect([(createnodes, creatematrix, ['node_network', 'resolution_network_file'])])
```

The merge nodes defined earlier are used here to create lists of the files which are destined for the CFFConverter.

```python
mapping.connect([(mris_convertLH, giftiSurfaces, ['converted', 'in1'])])
mapping.connect([(mris_convertRH, giftiSurfaces, ['converted', 'in2'])])
mapping.connect([(mris_convertLHwhite, giftiSurfaces, ['converted', 'in3'])])
mapping.connect([(mris_convertRHwhite, giftiSurfaces, ['converted', 'in4'])])
mapping.connect([(mris_convertLHinflated, giftiSurfaces, ['converted', 'in5'])])
mapping.connect([(mris_convertRHinflated, giftiSurfaces, ['converted', 'in6'])])
mapping.connect([(mris_convertLHsphere, giftiSurfaces, ['converted', 'in7'])])
mapping.connect([(mris_convertRHsphere, giftiSurfaces, ['converted', 'in8'])])
mapping.connect([(tessflow, giftiSurfaces, ['outputspec.meshes', 'in9'])])
mapping.connect([(mris_convertLHlabels, giftiLabels, ['converted', 'in1'])])
mapping.connect([(mris_convertRHlabels, giftiLabels, ['converted', 'in2'])])
mapping.connect([(parcellate, niftiVolumes, ['roi_file', 'in1'])])
mapping.connect([(eddycorrect, niftiVolumes, ['outputnode.eddy_corrected', 'in2'])])
mapping.connect([(mri_convert_Brain, niftiVolumes, ['out_file', 'in3'])])
```

(continues on next page)
mapping.connect([(creatematrix, fiberDataArrays, ["endpoint_file", "in1"])]
mapping.connect([(creatematrix, fiberDataArrays, ["endpoint_file_mm", 
"in2"])]
mapping.connect([(creatematrix, fiberDataArrays, ["fiber_length_file", 
"in3"])]
mapping.connect([(creatematrix, fiberDataArrays, ["fiber_label_file", 
"in4"])]
This block actually connects the merged lists to the CFF converter. We pass the surfaces and volumes 
that are to be included, as well as the tracts and the network itself. The currently running pipeline 
(dmri_connectivity_advanced.py) is also scraped and included in the CFF file. This makes it easy for the user 
to examine the entire processing pathway used to generate the end product.

mapping.connect([(giftiSurfaces, CFFConverter, ["out", "gifti_surfaces"])]
mapping.connect([(giftiLabels, CFFConverter, ["out", "gifti_labels"])]
mapping.connect([(creatematrix, CFFConverter, ["matrix_files", 
"gpickled_networks"])]
mapping.connect([(niftiVolumes, CFFConverter, ["out", "nifti_volumes"])]
mapping.connect([(fiberDataArrays, CFFConverter, ["filtered_tractographies", 
"tract_files"])]
mapping.connect([(inputnode, CFFConverter, ["subject_id", "title"])]
The graph theoretical metrics are computed using the networkx workflow and placed in another CFF file

mapping.connect([(inputnode, networkx, ["subject_id", 
"inputnode.extra_field"])]
mapping.connect([(creatematrix, networkx, ["intersection_matrix_file", 
"inputnode.network_file"])]
mapping.connect([(networkx, NxStatsCFFConverter, ["outputnode.network_files", 
"gpickled_networks"])]
mapping.connect([(giftiSurfaces, NxStatsCFFConverter, ["out", 
"gifti_surfaces"])]
mapping.connect([(giftiLabels, NxStatsCFFConverter, ["out", 
"gifti_labels"])]
mapping.connect([(niftiVolumes, NxStatsCFFConverter, ["out", 
"nifti_volumes"])]
mapping.connect([(fiberDataArrays, NxStatsCFFConverter, ["out", 
"data_files"])]
mapping.connect([(inputnode, NxStatsCFFConverter, ["subject_id", "title"])]
mapping.connect([(inputnode, cmats_to_csv, ["subject_id", 
"inputnode.extra_field"])]

9.4.4 Create a higher-level workflow
Finally, we create another higher-level workflow to connect our mapping workflow with the info and datagrab-
bng nodes declared at the beginning. Our tutorial is now extensible to any arbitrary number of subjects by 
simply adding their names to the subject list and their data to the proper folders.

connectivity = pe.Workflow(name="connectivity")
connectivity.base_dir = op.abspath('dmri_connectivity_advanced')
connectivity.connect([{'subject_id', 'subject_id'}]),
  (datasource, mapping,
   [{'dwi', 'inputnode.dwi'}, {'bvals', 'inputnode.bvals'},
     ('bvecs', 'inputnode.bvecs')],
  (infosource, mapping, [{'subject_id',
     'inputnode.subject_id'}]))

The following functions run the whole workflow and produce a .dot and .png graph of the processing pipeline.

```python
if __name__ == '__main__':
    connectivity.run()
    connectivity.write_graph()
```

Example source code
You can download the full source code of this example. This same script is also included in the Nipype source distribution under the examples directory.
A pipeline example that uses several interfaces to perform analysis on diffusion weighted images using Diffusion Toolkit tools.

This tutorial is based on the 2010 FSL course and uses data freely available at the FSL website at: http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz

More details can be found at http://www.fmrib.ox.ac.uk/fslcourse/lectures/practicals/fdt/index.htm

In order to run this tutorial you need to have Diffusion Toolkit and FSL tools installed and accessible from matlab/command line. Check by calling fslinfo and dtk from the command line.

Tell python where to find the appropriate functions.

```python
import nipype.interfaces.io as nio  # Data i/o
import nipype.interfaces.fsl as fsl  # fsl
import nipype.interfaces.diffusion_toolkit as dtk
import nipype.interfaces.utility as util  # utility
import nipype.pipeline.engine as pe  # pipeline engine
import os  # system functions
from nipype.workflows.dmri.fsl.dti import create_eddy_correct_pipeline
```

Confirm package dependencies are installed. (This is only for the tutorial, rarely would you put this in your own code.)

```python
from nipype.utils.misc import package_check

package_check('numpy', '1.3', 'tutorial1')
package_check('scipy', '0.7', 'tutorial1')
package_check('IPython', '0.10', 'tutorial1')
```

## 10.1 Setting up workflows

This is a generic workflow for DTI data analysis using the FSL

## 10.2 Data specific components

The nipype tutorial contains data for two subjects. Subject data is in two subdirectories, dwis1 and dwis2. Each subject directory contains each of the following files: bvec, bval, diffusion weighted data, a set of target masks, a seed file, and a transformation matrix.
Below we set some variables to inform the `datasource` about the layout of our data. We specify the location of the data, the subject sub-directories and a dictionary that maps each run to a mnemonic (or field) for the run type (`dwi` or `bvals`). These fields become the output fields of the `datasource` node in the pipeline.

Specifying the subject directories

```python
subject_list = ['subj1']
```

Map field names to individual subject runs

```python
info = dict(
    dwi=[['subject_id', 'data']],
    bvecs=[['subject_id', 'bvecs']],
    bvals=[['subject_id', 'bvals']]
)
```

```python
infosource = pe.Node(
    interface=util.IdentityInterface(fields=['subject_id']),
    name="infosource")
```

Here we set up iteration over all the subjects. The following line is a particular example of the flexibility of the system. The `datasource` attribute `iterables` tells the pipeline engine that it should repeat the analysis on each of the items in the `subject_list`. In the current example, the entire first level preprocessing and estimation will be repeated for each subject contained in `subject_list`.

```python
infosource.iterables = ('subject_id', subject_list)
```

Now we create a `nipype.interfaces.io.DataGrabber` object and fill in the information from above about the layout of our data. The `nipype.pipeline.engine.Node` module wraps the interface object and provides additional housekeeping and pipeline specific functionality.

```python
datasource = pe.Node(
    interface=nio.DataGrabber(
        infields=['subject_id'],
        outfields=list(info.keys()),
        name='datasource')
)
```

```python
datasource.inputs.template = "%s/%s"
# This needs to point to the fdt folder you can find after extracting
# http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz
datasource.inputs.base_directory = os.path.abspath('fsl_course_data/fdt/')
datasource.inputs.field_template = dict(dwi='%s/%s.nii.gz')
datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True
```

### 10.3 Setup for Diffusion Tensor Computation

Here we will create a generic workflow for DTI computation.

```python
computeTensor = pe.Workflow(name='computeTensor')
```

extract the volume with b=0 (nodif_brain)

```python
fslroi = pe.Node(interface=fsl.ExtractROI(), name='fslroi')
fslroi.inputs.t_min = 0
fslroi.inputs.t_size = 1
```

create a brain mask from the nodif_brain
nipype Documentation, Release 1.2.1

```python
bet = pe.Node(interface=fsl.BET(), name='bet')
bet.inputs.mask = True
bet.inputs.frac = 0.34
```
correct the diffusion weighted images for eddy_currents

```python
eddycorrect = create_eddy_correct_pipeline('eddycorrect')
eddycorrect.inputs.inputnode.ref_num = 0
```
compute the diffusion tensor in each voxel

```python
dtifit = pe.Node(interface=dtk.DTIRecon(), name='dtifit')
```
connect all the nodes for this workflow

```python
computeTensor.connect([(fslroi, bet, [('roi_file', 'in_file')]),
                       (eddycorrect, dtifit, [('outputnode.eddy_corrected', 'DWI')])])
```

### 10.4 Setup for Tracktography

Here we will create a workflow to enable deterministic tractography

```python
tractography = pe.Workflow(name='tractography')
dtk_tracker = pe.Node(interface=dtk.DTITracker(), name="dtk_tracker")
dtk_tracker.inputs.invert_x = True
smooth_trk = pe.Node(interface=dtk.SplineFilter(), name="smooth_trk")
smooth_trk.inputs.step_length = 0.5
```
connect all the nodes for this workflow

```python
tractography.connect([(dtk_tracker, smooth_trk, [('track_file', 'track_file')])])
```

Setup data storage area

```python
datasink = pe.Node(interface=nio.DataSink(), name='datasink')
datasink.inputs.base_directory = os.path.abspath('dtiresults')
def getstripdir(subject_id):
    return os.path.join(
        os.path.abspath('data/workingdir/dwiproc'),
        '_subject_id_%s' % subject_id)
```

### 10.5 Setup the pipeline that combines the 2 workflows: tractography & computeTensor

```python
dwiproc = pe.Workflow(name="dwiproc")
dwiproc.base_dir = os.path.abspath('dtk_dti_tutorial')
dwiproc.connect([(infosource, datasource, [('subject_id', 'subject_id')]),
                 (datasource, computeTensor,
                  [('dwi', 'fslroi.in_file'), ('bvals', 'dtifit.bvals'),
                   ('subject_id', 'subject_id')])])
```

(continues on next page)
Example source code
You can download the full source code of this example. This same script is also included in the Nipype source distribution under the examples directory.
A pipeline example that uses several interfaces to perform analysis on diffusion weighted images using Diffusion Toolkit tools. This tutorial is based on the 2010 FSL course and uses data freely available at the FSL website at: http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz

More details can be found at http://www.fmrib.ox.ac.uk/fslcourse/lectures/practicals/fdt/index.htm

In order to run this tutorial you need to have Diffusion Toolkit and FSL tools installed and accessible from matlab/command line. Check by calling fslinfo and dtk from the command line.

Tell python where to find the appropriate functions.

```python
import nipype.interfaces.io as nio  # Data i/o
import nipype.interfaces.fsl as fsl  # fsl
import nipype.interfaces.diffusion_toolkit as dtk
import nipype.interfaces.utility as util  # utility
import nipype.pipeline.engine as pe  # pipeline engine
import os  # system functions
from nipype.workflows.dmri.fsl.dti import create_eddy_correct_pipeline
```

Confirm package dependencies are installed. (This is only for the tutorial, rarely would you put this in your own code.)

```python
from nipype.utils.misc import package_check
package_check('numpy', '1.3', 'tutorial1')
package_check('scipy', '0.7', 'tutorial1')
package_check('IPython', '0.10', 'tutorial1')
```

### 11.1 Setting up workflows

This is a generic workflow for DTI data analysis using the FSL

### 11.2 Data specific components

The nipype tutorial contains data for two subjects. Subject data is in two subdirectories, `dwis1` and `dwis2`. Each subject directory contains each of the following files: bvec, bval, diffusion weighted data, a set of target masks, a seed file, and a transformation matrix.
Below we set some variables to inform the datasource about the layout of our data. We specify the location of the data, the subject sub-directories and a dictionary that maps each run to a mnemonic (or field) for the run type (dwi or bvals). These fields become the output fields of the datasource node in the pipeline.

Specify the subject directories

```python
subject_list = ['siemens_hardi_test']
```

Map field names to individual subject runs

```python
info = dict(
    dwi=[['subject_id', 'siemens_hardi_test_data']],
    bvecs=[['subject_id', 'siemens_hardi_test_data.bvec']],
    bvals=[['subject_id', 'siemens_hardi_test_data.bval']])
```

```python
infosource = pe.Node(
    interface=util.IdentityInterface(fields=['subject_id']), name="infosource")
```

Here we set up iteration over all the subjects. The following line is a particular example of the flexibility of the system. The datasource attribute iterables tells the pipeline engine that it should repeat the analysis on each of the items in the subject_list. In the current example, the entire first level preprocessing and estimation will be repeated for each subject contained in subject_list.

```python
infosource.iterables = ('subject_id', subject_list)
```

Now we create a nipype.interfaces.io.DataGrabber object and fill in the information from above about the layout of our data. The nipype.pipeline.engine.Node module wraps the interface object and provides additional housekeeping and pipeline specific functionality.

```python
datasource = pe.Node(
    interface=nio.DataGrabber(
        infields=['subject_id'], outfields=list(info.keys())),
    name='datasource')
```

```python
datasource.inputs.template = "%s\/%s"
```

# This needs to point to the fdt folder you can find after extracting
# http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz

```python
datasource.inputs.base_directory = os.path.abspath('data')
```

```python
datasource.inputs.field_template = dict(dwi='%s/\%s.nii')
```

```python
datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True
```

### 11.3 Setup for ODF Computation

Here we will create a generic workflow for ODF computation

```python
compute_ODF = pe.Workflow(name='compute_ODF')
```

```python
extract the volume with b=0 (nodif_brain)
```

```python
fslroi = pe.Node(interface=fsl.ExtractROI(), name='fslroi')
fslroi.inputs.t_min = 0
fslroi.inputs.t_size = 1
```

create a brain mask from the nodif_brain
correct the diffusion weighted images for eddy_currents

connect all the nodes for this workflow

11.4 Setup for Tracktography

Here we will create a workflow to enable deterministic tractography

connect all the nodes for this workflow

11.5 Setup the pipeline that combines the 2 workflows: tractography and compute_ODF

(continues on next page)
```python
dwiproc.inputs.compute_ODF.odf_recon.n_directions = 31
dwiproc.inputs.compute_ODF.odf_recon.n_b0 = 5
dwiproc.inputs.compute_ODF.odf_recon.n_output_directions = 181

if __name__ == '__main__':
    dwiproc.run()
    dwiproc.write_graph()
```

**Example source code**

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the examples directory.
A pipeline example that uses several interfaces to perform analysis on diffusion weighted images using FSL FDT tools.
This tutorial is based on the 2010 FSL course and uses data freely available at the FSL website at: http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz
More details can be found at http://www.fmrib.ox.ac.uk/fslcourse/lectures/practicals/fdt/index.htm
In order to run this tutorial you need to have fsl tools installed and accessible from matlab/command line. Check by calling fslinfo from the command line.
Tell python where to find the appropriate functions.

```python
import nipype.interfaces.io as nio   # Data i/o
import nipype.interfaces.fsl as fsl  # fsl
import nipype.interfaces.utility as util  # utility
import nipype.pipeline.engine as pe  # pipeline engine
import os  # system functions
from nipype.workflows.dmri.fsl.dti import create_eddy_correct_pipeline,
s create_bedpostx_pipeline
```

Confirm package dependencies are installed. (This is only for the tutorial, rarely would you put this in your own code.)

```python
from nipype.utils.misc import package_check
package_check('numpy', '1.3', 'tutorial1')
package_check('scipy', '0.7', 'tutorial1')
package_check('IPython', '0.10', 'tutorial1')
```

### 12.1 Setting up workflows

This is a generic workflow for DTI data analysis using the FSL

### 12.2 Data specific components

The nipype tutorial contains data for two subjects. Subject data is in two subdirectories, `dwis1` and `dwis2`. Each subject directory contains each of the following files: bvec, bval, diffusion weighted data, a set of target masks, a seed file, and a transformation matrix.
Below we set some variables to inform the `datasource` about the layout of our data. We specify the location of the data, the subject sub-directories and a dictionary that maps each run to a mnemonic (or field) for the run type (dwi or bvals). These fields become the output fields of the `datasource` node in the pipeline. Specify the subject directories

```python
subject_list = ['subj1']
```

Map field names to individual subject runs

```python
info = dict(
    dwi=[['subject_id', 'data']],
    bvecs=[['subject_id', 'bvecs']],
    bvals=[['subject_id', 'bvals']],
    seed_file=[['subject_id', 'MASK_average_thal_right'] ],
    target_masks=[
        ['subject_id', 'MASK_average_M1_right',
         'MASK_average_S1_right',
         'MASK_average_occipital_right',
         'MASK_average_pfc_right',
         'MASK_average_pmc_right',
         'MASK_average_ppc_right',
         'MASK_average_temporal_right'] ])
```

```python
infosource = pe.Node(
    interface=util.IdentityInterface(fields=['subject_id']),
    name="infosource")
```

Here we set up iteration over all the subjects. The following line is a particular example of the flexibility of the system. The `datasource` attribute `iterables` tells the pipeline engine that it should repeat the analysis on each of the items in the `subject_list`. In the current example, the entire first level preprocessing and estimation will be repeated for each subject contained in `subject_list`.

```python
infosource.iterables = ('subject_id', subject_list)
```

Now we create a `nipype.interfaces.io.DataGrabber` object and fill in the information from above about the layout of our data. The `nipype.pipeline.engine.Node` module wraps the interface object and provides additional housekeeping and pipeline specific functionality.

```python
datasource = pe.Node(
    interface=nio.DataGrabber( 
        infields=['subject_id'], outfields=list(info.keys())),
    name='datasource')
```

```python
datasource.inputs.template = "%s/%s"
# This needs to point to the fdt folder you can find after extracting
# http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz
datasource.inputs.base_directory = os.path.abspath('fsl_course_data/fdt/')
```

```python
datasource.inputs.field_template = dict( 
    dwi='"%s/nii.gz"',
    seed_file='"%s.bedpostX/%s.nii.gz"',
    target_masks='"%s.bedpostX/%s.nii.gz"')
datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True
```

## 12.3 Setup for Diffusion Tensor Computation

Here we will create a generic workflow for DTI computation
computeTensor = pe.Workflow(name='computeTensor')

extract the volume with b=0 (nodif_brain)

fsroi = pe.Node(interface=fsl.ExtractROI(), name='fsroi')
fsroi.inputs.t_min = 0
fsroi.inputs.t_size = 1

create a brain mask from the nodif_brain

bet = pe.Node(interface=fsl.BET(), name='bet')
bet.inputs.mask = True
bet.inputs.frac = 0.34

correct the diffusion weighted images for eddy_currents

eddycorrect = create_eddy_correct_pipeline('eddycorrect')
eddycorrect.inputs.inputnode.ref_num = 0

compute the diffusion tensor in each voxel

dtifit = pe.Node(interface=fsl.DTIFit(), name='dtifit')

connect all the nodes for this workflow

computeTensor.connect(
    [(fsroi, bet, [('roi_file', 'in_file')]),
     (eddycorrect, dtifit, [('outputnode.eddy_corrected', 'dwi')]),
     (infosource, dtifit, [['subject_id', 'base_name']]),
     (bet, dtifit, [('mask_file', 'mask')])])

12.4 Setup for Tracktography

Here we will create a workflow to enable probabilistic tracktography and hard segmentation of the seed region

tractography = pe.Workflow(name='tractography')
tractography.base_dir = os.path.abspath('fsl_dti_tutorial')

estimate the diffusion parameters: phi, theta, and so on

bedpostx = create_bedpostx_pipeline()
bedpostx.get_node("xfibres").iterables = ("n_fibres", [1, 2])

flirt = pe.Node(interface=fsl.FLIRT(), name='flirt')
flirt.inputs.in_file = fsl.Info.standard_image('MNI152_T1_2mm_brain.nii.gz')
flirt.inputs.dof = 12

perform probabilistic tracktography

probtrackx = pe.Node(interface=fsl.ProbTrackX(), name='probtrackx')
probtrackx.inputs.mode = 'seedmask'
probtrackx.inputs.c_thresh = 0.2
probtrackx.inputs.n_steps = 2000
probtrackx.inputs.step_length = 0.5
probtrackx.inputs.n_samples = 5000
probtrackx.inputs.opd = True
probtrackx.inputs.os2t = True
probtrackx.inputs.loop_check = True

perform hard segmentation on the output of probtrackx
findthebiggest = pe.Node(interface=fsl.FindTheBiggest(), name='findthebiggest')

connect all the nodes for this workflow

tractography.add_nodes([bedpostx, flirt])
tractography.connect([[('outputnode.thsamples', 'thsamples'), ('outputnode.phsamples', 'phsamples'), ('outputnode.fsamples', 'fsamples')]],
                     [(bedpostx, probtrackx, [('targets', 'in_files')]),
                      (flirt, probtrackx, [('out_matrix_file', 'xfm')])])

Setup data storage area

datasink = pe.Node(interface=nio.DataSink(), name='datasink')
datasink.inputs.base_directory = os.path.abspath('dtiresults')

def getstripdir(subject_id):
    import os
    return os.path.join(
        os.path.abspath('data/workingdir/dwiproc'),
        '_subject_id_%s' % subject_id)

12.5 Setup the pipeline that combines the 2 workflows: tractography & computeTensor

dwiproc = pe.Workflow(name="dwiproc")
dwiproc.base_dir = os.path.abspath('fsl_dti_tutorial')
dwiproc.connect(["subject_id", 'subject_id'])),
                  [(infosource, datasource, [('subject_id', 'subject_id')]),
                   (datasource, computeTensor, [('dwi', 'fslroi.in_file'), ('bvals', 'dtifit.bvals'), ('bvecs', 'dtifit.bvecs'), ('dwi', 'eddycorrect.outputnode.in_file')]),
                   (datasource, tractography, [('bvals', 'bedpostx.inputnode.bvals'), ('bvecs', 'bedpostx.inputnode.bvecs'), ('seed_file', 'probtrackx.seed'), ('target_masks', 'probtrackx.target_masks')]),
                   (computeTensor, tractography, [('eddycorrect.outputnode.eddy_corrected', 'bedpostx.inputnode.dwi'), ('bet.mask_file', 'bedpostx.inputnode.mask')],
                   (fslroi.roi_file', 'flirt.reference')]),
                   (infosource, datasink, [('subject_id', 'container')]),
                   (tractography, datasink, [('findthebiggest.out_file', 'fbiggest.@biggestsegmentation')])
    )

if __name__ == '__main__':
    dwiproc.run()
    dwiproc.write_graph()

Example source code
You can download the full source code of this example. This same script is also included in
the Nipype source distribution under the examples directory.
13.1 Introduction

This script, dmri_group_connectivity_camino.py, runs group-based connectivity analysis using the dmri.camino.connectivity_mapping Nipype workflow. Further detail on the processing can be found in dMRI: Connectivity - Camino, CMTK, FreeSurfer. This tutorial can be run using:

```python
dmri_group_connectivity_camino.py
```

We perform this analysis using one healthy subject and two subjects who suffer from Parkinson’s disease. The whole package (960 mb as .tar.gz / 1.3 gb uncompressed) including the Freesurfer directories for these subjects, can be acquired from here:

- http://db.tt/b6F1t0QV
- http://db.tt/kNvAI751

A data package containing the outputs of this pipeline can be obtained from here:

- http://db.tt/b6F1t0QV

Along with Camino, Camino-Trackvis, FSL, and Freesurfer, you must also have the Connectome File Format library installed as well as the Connectome Mapper.

- Camino-Trackvis: http://www.nitrc.org/projects/camino-trackvis/
- FSL: http://www.fmrib.ox.ac.uk/fsl/
- Freesurfer: http://surfer.nmr.mgh.harvard.edu/
- CMTK: http://www.cmtk.org/
- CFF: sudo apt-get install python-cfflib

Or on github at:

- CFFlib: https://github.com/LTS5/cfflib
- CMP: https://github.com/LTS5/cmp

Output data can be visualized in ConnectomeViewer, TrackVis, and anything that can view Nifti files.

- ConnectomeViewer: https://github.com/LTS5/connectomeviewer
- TrackVis: http://trackvis.org/

The fiber data is available in Numpy arrays, and the connectivity matrix is also produced as a MATLAB matrix.

13.1.1 Import the workflows

First, we import the necessary modules from nipype.
import nipype.interfaces.fsl as fsl
import nipype.interfaces.freesurfer as fs  # freesurfer
import os.path as op  # system functions
import cmp
from nipype.workflows.dmri.camino.group_connectivity import create_group_connectivity_pipeline
from nipipe.workflows.dmri.connectivity.group_connectivity import (create_merge_networks_by_group_workflow, create_merge_group_networks_workflow, create_average_networks_by_group_workflow)

13.1.2 Set the proper directories

First, we import the necessary modules from nipype.

fs_dir = op.abspath('/usr/local/freesurfer')
subjects_dir = op.abspath('groupcondatapackage/subjects/')
data_dir = op.abspath('groupcondatapackage/data/')
fs.FSCommand.set_default_subjects_dir(subjects_dir)
fsl.FSLCommand.set_default_output_type('NIFTI')

13.1.3 Define the groups

Here we define the groups for this study. We would like to search for differences between the healthy subject and the two vegetative patients. The group list is defined as a Python dictionary (see http://docs.python.org/tutorial/datastructures.html), with group IDs (‘controls’, ‘parkinsons’) as keys, and subject/patient names as values. We set the main output directory as ‘groupcon’.

group_list = {}
group_list['controls'] = ['cont17']
group_list['parkinsons'] = ['pat10', 'pat20']

The output directory must be named as well.

global output_dir
output_dir = op.abspath('dmri_group_connectivity_camino')

13.2 Main processing loop

The title for the final grouped-network connectome file is dependent on the group names. The resulting file for this example is ‘parkinsons-controls.cff’. The following code implements the format a-b-c-…x.cff for an arbitrary number of groups.

Warning: The ‘info’ dictionary below is used to define the input files. In this case, the diffusion weighted image contains the string ‘dwi’. The same applies to the b-values and b-vector files, and this must be changed to fit your naming scheme.

This line creates the processing workflow given the information input about the groups and subjects.

See also:
• nipype/workflows/dmri/mrtrix/group_connectivity.py
• nipype/workflows/dmri/camino/connectivity_mapping.py
• dMRI: Connectivity - Camino, CMTK, FreeSurfer

The purpose of the second-level workflow is simple: It is used to merge each subject’s CFF file into one, so that there is a single file containing all of the networks for each group. This can be useful for performing Network
Brain Statistics using the NBS plugin in ConnectomeViewer.

See also:
http://www.connectomeviewer.org/documentation/users/tutorials/tut_nbs.html

```python
for idx, group_id in enumerate(group_list.keys()):
    title += group_id
    if not idx == len(list(group_list.keys())) - 1:
        title += '-'
    info = dict(
        dwi=[['subject_id', 'dti']],
        bvecs=[['subject_id', 'bvecs']],
        bvals=[['subject_id', 'bvals']])

l1pipeline = create_group_connectivity_pipeline(
    group_list, group_id, data_dir, subjects_dir, output_dir, info)

l1pipeline.run()
l1pipeline.write_graph(format='eps', graph2use='flat')

# Here we define the parcellation scheme and the number of tracks to produce
parcellation_scheme = 'NativeFreesurfer'
cmp_config = cmp.configuration.PipelineConfiguration()
cmp_config.parcellation_scheme = parcellation_scheme
l1pipeline.inputs.connectivity.inputnode.resolution_network_file = cmp_config._get_lausanne_parcellation('freesurferaparc')['node_information_graphml']

l1pipeline.run()
l1pipeline.write_graph(format='eps', graph2use='flat')

# The second-level pipeline is created here
l2pipeline = create_merge_networks_by_group_workflow(
    group_list, group_id, data_dir, subjects_dir, output_dir)
l2pipeline.run()
l2pipeline.write_graph(format='eps', graph2use='flat')

Now that the for loop is complete there are two grouped CFF files each containing the appropriate subjects. It is also convenient to have every subject in a single CFF file, so that is what the third-level pipeline does.

```python
l3pipeline = create_merge_group_networks_workflow(
    group_list, data_dir, subjects_dir, output_dir, title)
l3pipeline.run()
l3pipeline.write_graph(format='eps', graph2use='flat')
```

The fourth and final workflow averages the networks and saves them in another CFF file

```python
l4pipeline = create_average_networks_by_group_workflow(
    group_list, data_dir, subjects_dir, output_dir, title)
l4pipeline.run()
l4pipeline.write_graph(format='eps', graph2use='flat')
```

### Example source code
You can download the full source code of this example. This same script is also included in the Nipype source distribution under the examples directory.

13.2. Main processing loop

161
14.1 Introduction

This script, dmri_group_connectivity_mrtrix.py, runs group-based connectivity analysis using the dmri.mrtrix.connectivity_mapping Nipype workflow. Further detail on the processing can be found in dMRI: Connectivity - MRtrix, CMTK, FreeSurfer. This tutorial can be run using:

```python
dmri_group_connectivity_mrtrix.py
```

We perform this analysis using one healthy subject and two subjects who suffer from Parkinson’s disease. The whole package (960 mb as .tar.gz / 1.3 gb uncompressed) including the Freesurfer directories for these subjects, can be acquired from here:

- [http://db.tt/b6F1t0QV](http://db.tt/b6F1t0QV)

A data package containing the outputs of this pipeline can be obtained from here:

- [http://db.tt/elmMnIt1](http://db.tt/elmMnIt1)

Along with MRtrix, FSL, and Freesurfer, you must also have the Connectome File Format library installed as well as the Connectome Mapper (cmp).

- [MRtrix](http://www.brain.org.au/software/mrtrix/)
- [FSL](http://www.fmrib.ox.ac.uk/fsl/)
- [Freesurfer](http://surfer.nmr.mgh.harvard.edu/)
- [CTMK](http://www.cmtk.org/)
- CFF: `sudo apt-get install python-cfflib`
- Or on github at:
  - [CFFlib](https://github.com/LTS5/cfflib)
  - [CMP](https://github.com/LTS5/cmp)

Output data can be visualized in ConnectomeViewer, TrackVis, Gephi, the MRtrix Viewer (mrview), and anything that can view Nifti files.

- [ConnectomeViewer](https://github.com/LTS5/connectomeviewer)
- [TrackVis](http://trackvis.org/)
- [Gephi](http://gephi.org/)

The fiber data is available in Numpy arrays, and the connectivity matrix is also produced as a MATLAB matrix.

14.1.1 Import the workflows

First, we import the necessary modules from nipype.
14.1.2 Set the proper directories

First, we import the necessary modules from nipype.

```python
subjects_dir = op.abspath('groupcondatapackage/subjects/')
data_dir = op.abspath('groupcondatapackage/data/')
fs.FSCommand.set_default_subjects_dir(subjects_dir)
fsl.FSLCommand.set_default_output_type('NIFTI')
```

14.1.3 Define the groups

Here we define the groups for this study. We would like to search for differences between the healthy subject and the two vegetative patients. The group list is defined as a Python dictionary (see http://docs.python.org/tutorial/datastructures.html), with group IDs (‘controls’, ‘parkinsons’) as keys, and subject/patient names as values. We set the main output directory as ‘groupcon’.

```python
group_list = {}
group_list['controls'] = ['cont17']
group_list['parkinsons'] = ['pat10', 'pat20']
```

The output directory must be named as well.

```python
global output_dir
output_dir = op.abspath('dmri_group_connectivity_mrtrix')
```

14.2 Main processing loop

The title for the final grouped-network connectome file is dependent on the group names. The resulting file for this example is ‘parkinsons-controls.cff’. The following code implements the format a-b-c-...x.cff for an arbitrary number of groups.

**Warning:** The ‘info’ dictionary below is used to define the input files. In this case, the diffusion weighted image contains the string ‘dti’. The same applies to the b-values and b-vector files, and this must be changed to fit your naming scheme.

The workflow is created given the information input about the groups and subjects.

**See also:**
- nipype/workflows/dmri/mrtrix/group_connectivity.py
- nipype/workflows/dmri/mrtrix/connectivity_mapping.py
- dMRI: Connectivity - MRtrix, CMTK, FreeSurfer

We set values for absolute threshold used on the fractional anisotropy map. This is done in order to identify single-fiber voxels. In brains with more damage, however, it may be necessary to reduce the threshold, since their brains have lower average fractional anisotropy values.
We invert the b-vectors in the encoding file, and set the maximum harmonic order of the pre-tractography spherical deconvolution step. This is done to show how to set inputs that will affect both groups.

Next we create and run the second-level pipeline. The purpose of this workflow is simple: It is used to merge each subject’s CFF file into one, so that there is a single file containing all of the networks for each group. This can be useful for performing Network Brain Statistics using the NBS plugin in ConnectomeViewer.

See also:
http://www.connectomeviewer.org/documentation/users/tutorials/tut_nbs.html

```python
for idx, group_id in enumerate(group_list.keys()):
    title += group_id
    if not idx == len(list(group_list.keys())) - 1:
        title += '-'
    info = dict(
        dwi=[['subject_id', 'dti']],
        bvecs=[['subject_id', 'bvecs']],
        bvals=[['subject_id', 'bvals']])

l1pipeline = create_group_connectivity_pipeline(
    group_list, group_id, data_dir, subjects_dir, output_dir, info)

# Here with invert the b-vectors in the Y direction and set the maximum harmonic order of the spherical deconvolution step
l1pipeline.inputs.connectivity.mapping.fsl2mrtrix.invert_y = True
l1pipeline.inputs.connectivity.mapping.csdeconv.maximum_harmonic_order = 6

# Here we define the parcellation scheme and the number of tracks to produce
parcellation_name = 'scale500'
parcellation = cmp_config._get_lausanne_parcellation('Lausanne2008')[parcellation_name]['node_information_graphml']
l1pipeline.inputs.connectivity.mapping.probCSDstreamtrack.desired_number_of_tracks = 100000

l1pipeline.run()
l1pipeline.write_graph(format='eps', graph2use='flat')

# The second-level pipeline is created here
l2pipeline = create_merge_network_results_by_group_workflow(
    group_list, group_id, data_dir, subjects_dir, output_dir)

l2pipeline.inputs.l2inputnode.network_file = cmp_config._get_lausanne_parcellation('Lausanne2008')[parcellation_name]['node_information_graphml']
l2pipeline.run()
l2pipeline.write_graph(format='eps', graph2use='flat')
```

Now that the for loop is complete there are two grouped CFF files each containing the appropriate subjects. It is also convenient to have every subject in a single CFF file, so that is what the third-level pipeline does.

```python
l3pipeline = create_merge_group_network_results_workflow(
    group_list, data_dir, subjects_dir, output_dir, title)
l3pipeline.run()
```

(continues on next page)
l3pipeline.write_graph(format='eps', graph2use='flat')

The fourth and final workflow averages the networks and saves them in another CFF file

l4pipeline = create_average_networks_by_group_workflow(
    group_list, data_dir, subjects_dir, output_dir, title
)
l4pipeline.run()
l4pipeline.write_graph(format='eps', graph2use='flat')

Example source code
You can download the full source code of this example. This same script is also included in the Nipype source distribution under the examples directory.
15.1 Introduction

This script, dmri_mrtrix_dti.py, demonstrates the ability to perform advanced diffusion analysis in a Nipype pipeline:

```python
dmri_mrtrix_dti.py
```

We perform this analysis using the FSL course data, which can be acquired from here:

- [http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz](http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz)

Import necessary modules from nipype:

```python
import nipype.interfaces.io as nio  # Data i/o
import nipype.interfaces.utility as util  # utility
import nipype.pipeline.engine as pe  # pipeline engine
import nipype.interfaces.mrtrix as mrtrix  # <---- The important new part!
import nipype.interfaces.fsl as fsl
import nipype.algorithms.misc as misc
import os
import os.path as op  # system functions
```

This needs to point to the fdt folder you can find after extracting:

- [http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz](http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz)

Use `infosource` node to loop through the subject list and define the input files. For our purposes, these are the diffusion-weighted MR image, b vectors, and b values.

```python
subject_list = ['subj1']
```

```python
subject_id = util.IdentityInterface(fields=['subject_id'], name='infosource')
subject_id.iterables = ('subject_id', subject_list)
```

```python
info = dict(
    dwi=[['subject_id', 'data']],
    bvecs=[['subject_id', 'bvecs']],
)```
Use datasource node to perform the actual data grabbing. Templates for the associated images are used to obtain the correct images.

```python
bvals=[['subject_id', 'bvals'])
```

An inputnode is used to pass the data obtained by the data grabber to the actual processing functions.

```python
datasource = pe.Node(
    interface=nio.DataGrabber(
        infields=['subject_id'], outfields=list(info.keys()),
        name='datasource'))

datasource.inputs.template = "%s"/

datasource.inputs.base_directory = data_dir

datasource.inputs.field_template = dict(dwi='%s.nii.gz')

datasource.inputs.template_args = info

datasource.inputs.sort_filelist = True

inputnode = pe.Node(
    interface=util.IdentityInterface(fields=['dwi', 'bvecs', 'bvals']),
    name="inputnode")
```

15.1.1 Diffusion processing nodes

See also:

dmri_connectivity_advanced.py Tutorial with further detail on using MRtrix tractography for connectivity analysis


b-values and b-vectors stored in FSL’s format are converted into a single encoding file for MRTrix.

```python
fsl2mrtrix = pe.Node(interface=mrtrix.FSL2MRTrix(), name='fsl2mrtrix')
```

Tensors are fitted to each voxel in the diffusion-weighted image and from these three maps are created:

- Major eigenvector in each voxel
- Apparent diffusion coefficient
- Fractional anisotropy

```python
gunzip = pe.Node(interface=misc.Gunzip(), name='gunzip')
dwi2tensor = pe.Node(interface=mrtrix.DWI2Tensor(), name='dwi2tensor')
tensor2vector = pe.Node(interface=mrtrix.Tensor2Vector(), name='tensor2vector')
tensor2adc = pe.Node(
    interface=mrtrix.Tensor2ApparentDiffusion(), name='tensor2adc')
tensor2fa = pe.Node(
    interface=mrtrix.Tensor2FractionalAnisotropy(), name='tensor2fa')
```

These nodes are used to create a rough brain mask from the b0 image. The b0 image is extracted from the original diffusion-weighted image, put through a simple thresholding routine, and smoothed using a 3x3 median filter.

```python
MRconvert = pe.Node(interface=mrtrix.MRConvert(), name='MRconvert')
MRconvert.inputs.extract_at_axis = 3
MRconvert.inputs.extract_at_coordinate = [0]
threshold_b0 = pe.Node(interface=mrtrix.Threshold(), name='threshold_b0')
median3d = pe.Node(interface=mrtrix.MedianFilter3D(), name='median3d')
```

The brain mask is also used to help identify single-fiber voxels. This is done by passing the brain mask through two erosion steps, multiplying the remaining mask with the fractional anisotropy map, and thresholding the result to obtain some highly anisotropic within-brain voxels.
For whole-brain tracking we also require a broad white-matter seed mask. This is created by generating a white matter mask, given a brainmask, and thresholding it at a reasonably high level.

The spherical deconvolution step depends on the estimate of the response function in the highly anisotropic voxels we obtained above.

**Warning:** For damaged or pathological brains one should take care to lower the maximum harmonic order of these steps.

Finally, we track probabilistically using the orientation distribution functions obtained earlier. The tracts are then used to generate a tract-density image, and they are also converted to TrackVis format.

### 15.1.2 Creating the workflow

In this section we connect the nodes for the diffusion processing.

(continues on next page)
This block creates the rough brain mask to be multiplied, multiplies it with the fractional anisotropy image, and thresholds it to get the single-fiber voxels.

Here the thresholded white matter mask is created for seeding the tractography.

Next we estimate the fiber response distribution.

Run constrained spherical deconvolution.

Connect the tractography and compute the tract density image.

(continues on next page)
Finally, we create another higher-level workflow to connect our tractography workflow with the info and data-grabbing nodes declared at the beginning. Our tutorial is now extensible to any arbitrary number of subjects by simply adding their names to the subject list and their data to the proper folders.

```
dwiproc = pe.Workflow(name="dwiproc")
dwiproc.base_dir = os.path.abspath('dmri_mrtrix_dti')
dwiproc.connect([(infosource, datasource, [('subject_id', 'subject_id')]),
                (datasource, tractography,
                 [('dwi', 'inputnode.dwi'), ('bvals', 'inputnode.bvals'),
                  ('bvecs', 'inputnode.bvecs')])])

if __name__ == '__main__':
    dwiproc.run()
    dwiproc.write_graph()
```

**Example source code**
You can download the full source code of this example. This same script is also included in the Nipype source distribution under the examples directory.
16.1 Introduction

This script, dmri_preprocessing.py, demonstrates how to prepare dMRI data for tractography and connectivity analysis with nipype.

We perform this analysis using the FSL course data, which can be acquired from here: http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz

Can be executed in command line using python dmri_preprocessing.py

Import necessary modules from nipype.

```python
import os  # system functions
import nipype.interfaces.io as nio  # Data i/o
import nipype.interfaces.utility as niu  # utility
import nipype.algorithms.misc as misc
import nipype.pipeline.engine as pe  # pipeline engine
from nipype.interfaces import fsl
from nipype.interfaces import ants

from nipype.workflows.dmri.fsl.artifacts import all_fsl_pipeline, remove_bias
```

Load specific nipype's workflows for preprocessing of dMRI data: nipype.workflows.dmri.preprocess.epi.all_peb_pipeline, as data include a \( b0 \) volume with reverse encoding direction (\( P>>>A \), or \( y \)), in contrast with the general acquisition encoding that is \( A>>>P \) or -\( y \) (in RAS systems).

```python
from nipype.workflows.dmri.fsl.artifacts import all_fsl_pipeline, remove_bias
```

Map field names into individual subject runs

```python
info = dict(
    dwi=[['subject_id', 'dwidata']],
    bvecs=[['subject_id', 'bvecs']],
    bvals=[['subject_id', 'bvals']],
    dwi_rev=[['subject_id', 'nodif_PA']])(continues on next page)
```

```python
infosource = pe.Node(
    interface=niu.IdentityInterface(fields=['subject_id']), name="infosource")
```

# Set the subject identifier in subject_list,
Here we set up iteration over all the subjects. The following line is a particular example of the flexibility of the system. The `datasource` attribute `iterables` tells the pipeline engine that it should repeat the analysis on each of the items in the `subject_list`. In the current example, the entire first level preprocessing and estimation will be repeated for each `subject_list`.

```python
infosource.iterables = ('subject_id', subject_list)
```

Now we create a `nipype.interfaces.io.DataGrabber` object and fill in the information from above about the layout of our data. The `Node` module wraps the interface object and provides additional housekeeping and pipeline specific functionality.

```python
datasource = pe.Node(
    nio.DataGrabber(infields=['subject_id'], outfields=list(info.keys())),
    name='datasource')
```

```python
# This needs to point to the fdt folder you can find after extracting
# http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz
datasource.inputs.base_directory = os.path.abspath('fdt1')
datasource.inputs.field_template = dict(
    dwi='\$s/\$s.nii.gz', dwi_rev='\$s/\$s.nii.gz')
datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True
```

An inputnode is used to pass the data obtained by the data grabber to the

```python
inputnode = pe.Node(
    niu.IdentityInterface(fields=['dwi', 'bvecs', 'bvals', 'dwi_rev']),
    name="inputnode")
```

### 16.2 Setup for dMRI preprocessing

In this section we initialize the appropriate workflow for preprocessing of diffusion images.

#### 16.2.1 Artifacts correction

We will use the combination of `topup` and `eddy` as suggested by FSL.

In order to configure the susceptibility distortion correction (SDC), we first write the specific parameters of our echo-planar imaging (EPI) images. Particularly, we look into the `acqparams.txt` file of the selected subject to gather the encoding direction, acceleration factor (in parallel sequences it is > 1), and readout time or echospacing.

```python
epi_AP = {'echospacing': 66.5e-3, 'enc_dir': 'y-'}
epi_PA = {'echospacing': 66.5e-3, 'enc_dir': 'y'}
prep = all_fsl_pipeline(epi_params=epi_AP, altepi_params=epi_PA)
```

#### 16.2.2 Bias field correction

Finally, we set up a node to correct for a single multiplicative bias field from computed on the $b0$ image, as suggested in [Jeurissen2014].

16.3 Connect nodes in workflow

We create a higher level workflow to connect the nodes. Please excuse the author for writing the arguments of the `connect` function in a not-standard style with readability aims.

```python
bias = remove_bias()

wf = pe.Workflow(name="dMRI_Preprocessing")
wf.base_dir = os.path.abspath('preprocessing_dmri_tutorial')
wf.connect([[infosource, datasource, [('subject_id', 'subject_id')]],
    {datasource, prep,
     [('dwi', 'inputnode.in_file'), ('dwi_rev', 'inputnode.alt_file'),
      ('bvals', 'inputnode.in_bval'), ('bvecs', 'inputnode.in_bvec')]),
     (prep, bias, [('outputnode.out_file', 'inputnode.in_file'),
      ('outputnode.out_mask', 'inputnode.in_mask')]),
     (datasource, bias, [('bvals', 'inputnode.in_bval')])])

Run the workflow as command line executable

```python
if __name__ == '__main__':
    wf.run()
    wf.write_graph()
```

**Example source code**

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the examples directory.
A pipeline to do a TBSS analysis on the NKI rockland sample data

```python
from nipype.workflows.dmri.fsl.dti import create_eddy_correct_pipeline
from nipype.workflows.dmri.fsl.tbss import create_tbss_non_FA, create_tbss_all
```

Tell python where to find the appropriate functions.

```python
import nipype.interfaces.io as nio  # Data i/o
import nipype.interfaces.fsl as fsl  # fsl
import nipype.interfaces.utility as util  # utility
import nipype.pipeline.engine as pe  # pipeline engine
import os  # system functions

fsl.FSLCommand.set_default_output_type('NIFTI')
```

You can get the data from:
http://fcon_1000.projects.nitrc.org/indi/pro/eNKI_RS_TRT/FrontPage.html

```python
dataDir = os.path.abspath('nki_rs_data')
workingdir = './tbss_example'
subjects_list = ['2475376', '3313349', '3808535', '3893245', '8735778', '9630905']

gen_fa = pe.Workflow(name="gen_fa")
gen_fa.base_dir = os.path.join(os.path.abspath(workingdir), 'l1')

subject_id_infosource = pe.Node(
    util.IdentityInterface(fields=['subject_id']),
    name='subject_id_infosource')

subject_id_infosource.iterables = ('subject_id', subjects_list)

datasource = pe.Node(
    interface=nio.DataGrabber(
        infields=['subject_id'], outfields=['dwi', 'bvec', 'bval']),
    name='datasource')

datasource.inputs.base_directory = os.path.abspath(dataDir)
datasource.inputs.template = '%s/session2/DTI_mx_137/dti.%s'
```

(continues on next page)
datasource.inputs.template_args = dict(
    dwi=[['subject_id', 'nii.gz']],
    bvec=[['subject_id', 'bvec']],
    bval=[['subject_id', 'bval']])

datasource.inputs.sort_filelist = True

gfa = pe.Workflow(name='gen_fa')
gfa.connect(subject_id_infosource, 'subject_id', datasource, 'subject_id')

eddy_correct = create_eddy_correct_pipeline()
eddy_correct.inputs.inputnode.ref_num = 0
gfa.connect(datasource, 'dwi', eddy_correct, 'inputnode.in_file')

bet = pe.Node(interface=fsl.BET(), name='bet')
bet.inputs.mask = True
bet.inputs.frac = 0.34
gfa.connect(eddy_correct, 'pick_ref.out', bet, 'in_file')

dtifit = pe.Node(interface=fsl.DTIFit(), name='dtifit')
gfa.connect(eddy_correct, 'outputnode.eddy_corrected', dtifit, 'dwi')
gfa.connect(subject_id_infosource, 'subject_id', dtifit, 'base_name')
gfa.connect(bet, 'mask_file', dtifit, 'mask')
gfa.connect(datasource, 'bvec', dtifit, 'bvecs')
gfa.connect(datasource, 'bval', dtifit, 'bvals')

data = pe.Workflow(name='gen_fa')
data.connect(subject_id_infosource, 'subject_id', dtifit, 'base_name')
data.connect(bet, 'mask_file', dtifit, 'mask')
data.connect(datasource, 'dwi', dtifit, 'dwi')

if __name__ == '__main__':
    gen_fa.run()
Example source code
You can download the full source code of this example. This same script is also included in the Nipype source distribution under the examples directory.
fMRI: OpenfMRI.org data, FSL, ANTS, c3daffine

A growing number of datasets are available on OpenfMRI. This script demonstrates how to use nipype to analyze a data set:

```python
python fmri_ants_openfmri.py --datasetdir ds107
```

This workflow also requires 2mm subcortical templates that are available from MindBoggle. Specifically the 2mm version of the MNI template.

Import necessary modules from nipype.

```python
from __future__ import division, unicode_literals
from builtins import open, range, str, bytes
from glob import glob
import os
from nipype import config
from nipype import LooseVersion
from nipype import Workflow, Node, MapNode
from nipype.utils.filemanip import filename_to_list
import nipype.pipeline.engine as pe
import nipype.algorithms.modelgen as model
import nipype.algorithms.rapidart as ra
from nipype.algorithms.misc import TSNR, CalculateMedian
from nipype.interfaces.c3 import C3dAffineTool
from nipype.interfaces import fsl, Function, ants, freesurfer as fs
import nipype.interfaces.io as nio
from nipype.interfaces.io import FreeSurferSource
import nipype.interfaces.utility as niu
from nipype.workflows.fmri.fsl import (create_featreg_preproc,
screate_modelfit_workflow,
ccreate_fixed_effects_flow)

from nipype.utils import NUMPY_MMAP

config.enable_provenance()
version = 0
if (fsl.Info.version()
and LooseVersion(fsl.Info.version()) > LooseVersion('5.0.6')):
    version = 507
fsl.FSLCommand.set_default_output_type('NIFTI_GZ')

imports = ['import os',
           'import nibabel as nb',
           'import numpy as np',
           'import scipy as sp',
           'import os',
           'from nipype.utils.filemanip import filename_to_list, list_to_filename,
            split_filename',
           'from scipy.special import legendre']


def create_reg_workflow(name='registration'):
    """Create a FEAT preprocessing workflow together with freesurfer
    Parameters
    ----------
    name : name of workflow (default: 'registration')
    Inputs:
    inputspec.source_files : files (filename or list of filenames to register)
    inputspec.mean_image : reference image to use
    inputspec.anatomical_image : anatomical image to coregister to
    inputspec.target_image : registration target
    Outputs:
    outputspec.func2anat_transform : FLIRT transform
    outputspec.anat2target_transform : FLIRT+FNIRT transform
    outputspec.transformed_files : transformed files in target space
    outputspec.transformed_mean : mean image in target space
    Example
    -------
    See code below
    """
    register = pe.Workflow(name=name)

    inputnode = pe.Node(
        interface=niu.IdentityInterface(fields=[
            'source_files', 'mean_image', 'anatomical_image', 'target_image',
            'target_image_brain', 'config_file',
        ]),
        name='inputspec')
    outputnode = pe.Node(
        interface=niu.IdentityInterface(fields=[
            'func2anat_transform', 'anat2target_transform',
            'transformed_files', 'transformed_mean', 'anat2target',
            'mean2anat_mask',
        ]),
        name='outputspec')
    
    register.connect([inputnode, outputnode])

    # Add nodes and connections
    pass

(continues on next page)
Estimate the tissue classes from the anatomical image. But use spm's segment as FSL appears to be breaking.

```python
stripper = pe.Node(fsl.BET(), name='stripper')
register.connect(inputnode, 'anatomical_image', stripper, 'in_file')
fast = pe.Node(fsl.FAST(), name='fast')
register.connect(stripper, 'out_file', fast, 'in_files')

Binarize the segmentation
```

```python
binarize = pe.Node(fsl.ImageMaths(op_string='-nan -thr 0.5 -bin'), name='binarize')
pickindex = lambda x, i: x[i]
register.connect(fast, {'partial_volume_files', pickindex, 2}, binarize, 'in_file')

Calculate rigid transform from mean image to anatomical image
```

```python
mean2anat = pe.Node(fsl.FLIRT(), name='mean2anat')
mean2anat.inputs.dof = 6
register.connect(inputnode, 'mean_image', mean2anat, 'in_file')
register.connect(stripper, 'out_file', mean2anat, 'reference')

Now use bbr cost function to improve the transform
```

```python
mean2anatbbr = pe.Node(fsl.FLIRT(), name='mean2anatbbr')
mean2anatbbr.inputs.dof = 6
mean2anatbbr.inputs.cost = 'bbr'
mean2anatbbr.inputs.schedule = os.path.join(os.getenv('FSLDIR'), 'etc/flirtsch/bbr.sch')
register.connect(inputnode, 'mean_image', mean2anatbbr, 'in_file')
register.connect(binarize, 'out_file', mean2anatbbr, 'wm_seg')
register.connect(inputnode, 'anatomical_image', mean2anatbbr, 'reference')
register.connect(mean2anat, 'out_matrix_file', mean2anatbbr, 'in_matrix_file')

Create a mask of the median image coregistered to the anatomical image
```

```python
mean2anat_mask = Node(fsl.BET(mask=True), name='mean2anat_mask')
register.connect(mean2anatbbr, 'out_file', mean2anat_mask, 'in_file')

Convert the BBRegister transformation to ANTS ITK format
```

```python
convert2itk = pe.Node(C3dAffineTool(), name='convert2itk')
convert2itk.inputs.fsl2ras = True
convert2itk.inputs.itk_transform = True
register.connect(mean2anatbbr, 'out_matrix_file', convert2itk, 'transform_file')
register.connect(inputnode, 'mean_image', convert2itk, 'source_file')
register.connect(stripper, 'out_file', convert2itk, 'reference_file')
```
Compute registration between the subject's structural and MNI template

* All parameters are set using the example from:
  #https://github.com/stnava/ANTs/blob/master/Scripts/newAntsExample.sh
* This is currently set to perform a very quick registration. However, the registration can be made significantly more accurate for cortical structures by increasing the number of iterations.

```python
reg = pe.Node(ants.Registration(), name='antsRegister')
reg.inputs.output_transform_prefix = "output_"
reg.inputs.transforms = ['Rigid', 'Affine', 'SyN']
reg.inputs.transform_parameters = [(0.1, ), (0.1, ), (0.2, 3.0, 0.0)]
reg.inputs.number_of_iterations = [[10000, 11110, 11110]] * 2 + [[100, 30, 20]]
reg.inputs.dimension = 3
reg.inputs.write_composite_transform = True
reg.inputs.collapse_output_transforms = True
reg.inputs.initial_moving_transform_com = True
reg.inputs.metric = ['Mattes'] * 2 + ['Mattes', 'CC']
reg.inputs.metric_weight = [1] * 2 + [0.5, 0.5]
reg.inputs.radius_or_number_of_bins = [32] * 2 + [32, 4]
reg.inputs.sampling_strategy = ['Regular'] * 2 + [None, None]
reg.inputs.sampling_percentage = [0.3] * 2 + [None, None]
reg.inputs.convergence_threshold = [1.e-8] * 2 + [-0.01]
reg.inputs.smoothing_sigmas = [[4, 2, 1]] * 2 + [[1, 0.5, 0]]
reg.inputs.sigma_units = ['vox'] * 3
reg.inputs.shrink_factors = [[3, 2, 1]] * 2 + [[4, 2, 1]]
reg.inputs.use_estimate_learning_rate_once = [True] * 3
reg.inputs.use_histogram_matching = [False] * 2 + [True]
reg.inputs.winsorize_lower_quantile = 0.005
reg.inputs.winsorize_upper_quantile = 0.995
reg.inputs.args = '--float'
reg.inputs.output_warped_image = 'output_warped_image.nii.gz'
reg.inputs.num_threads = 4
reg.plugin_args = {
    'qsub_args': '-pe orte 4',
    'sbatch_args': '--mem=6G -c 4'
}

register.connect(stripper, 'out_file', reg, 'moving_image')
register.connect(inputnode, 'target_image_brain', reg, 'fixed_image')
```

Concatenate the affine and ants transforms into a list

```python
merge = pe.Node(niu.Merge(2), iterfield=['in2'], name='mergexfm')
register.connect(convert2itk, 'itk_transform', merge, 'in2')
register.connect(reg, 'composite_transform', merge, 'in1')
```

Transform the mean image. First to anatomical and then to target

```python
warpmean = pe.Node(ants.ApplyTransforms(), name='warpmean')
warpmean.inputs.input_image_type = 0
warpmean.inputs.interpolation = 'Linear'
warpmean.inputs.invert_transform_flags = [False, False]
warpmean.terminal_output = 'file'
```

(continues on next page)
register.connect(inputnode, 'target_image_brain', warpmean, 'reference_image')
register.connect(inputnode, 'mean_image', warpmean, 'input_image')
register.connect(merge, 'out', warpmean, 'transforms')

"""
Transform the remaining images. First to anatomical and then to target
"""

warpall = pe.MapNode(
    ants.ApplyTransforms(), iterfield=['input_image'], name='warpall')
warpall.inputs.input_image_type = 0
warpall.inputs.interpolation = 'Linear'
warpall.inputs.invert_transform_flags = [False, False]
warpall.terminal_output = 'file'

register.connect(inputnode, 'target_image_brain', warpall, 'reference_image')
register.connect(inputnode, 'source_files', warpall, 'input_image')
register.connect(merge, 'out', warpall, 'transforms')

"""
Assign all the output files
"""

register.connect(reg, 'warped_image', outputnode, 'anat2target')
register.connect(warpmean, 'output_image', outputnode, 'transformed_mean')
register.connect(warpall, 'output_image', outputnode, 'transformed_files')
register.connect(mean2anatbbr, 'out_matrix_file', outputnode, 'func2anat_transform')
register.connect(mean2anat_mask, 'mask_file', outputnode, 'mean2anat_mask')
register.connect(reg, 'composite_transform', outputnode, 'anat2target_transform')

return register

def get_aparc_aseg(files):
    """Return the aparc+aseg.mgz file"""
    for name in files:
        if 'aparc+aseg.mgz' in name:
            return name
    raise ValueError('aparc+aseg.mgz not found')

def create_fs_reg_workflow(name='registration'):
    """Create a FEAT preprocessing workflow together with freesurfer"

    Parameters
    ----------

    name : name of workflow (default: 'registration')

    Inputs:

    inputspec.source_files : files (filename or list of filenames to register)
    inputspec.mean_image : reference image to use
    inputspec.target_image : registration target
Outputs:

- `outputspec.func2anat_transform`: FLIRT transform
- `outputspec.anat2target_transform`: FLIRT+FNIRT transform
- `outputspec.transformed_files`: transformed files in target space
- `outputspec.transformed_mean`: mean image in target space

Example
-------

See code below

```python
register = Workflow(name=name)

inputnode = Node(
    interface=IdentityInterface(fields=[
        'source_files', 'mean_image', 'subject_id', 'subjects_dir', 'target_image'
    ]),
    name='inputspec')

outputnode = Node(
    interface=IdentityInterface(fields=[
        'func2anat_transform', 'out_reg_file', 'anat2target_transform', 'transforms', 'transformed_mean', 'transformed_files', 'min_cost_file', 'anat2target', 'aparc', 'mean2anat_mask'
    ]),
    name='outputspec')

# Get the subject's FreeSurfer source directory
fssource = Node(FreeSurferSource(), name='fssource')
register.connect(inputnode, 'subject_id', fssource, 'subject_id')
register.connect(inputnode, 'subjects_dir', fssource, 'subjects_dir')

convert = Node(freesurfer.MRIConvert(out_type='nii'), name='convert')
register.connect(fssource, 'T1', convert, 'in_file')

# Coregister the median to the surface
bbregister = Node(freesurfer.BBRegister(registered_file=True), name='bbregister')
bbregister.inputs.init = 'fsl'
bbregister.inputs.contrast_type = 't2'
bbregister.inputs.out_fsl_file = True
register.connect(inputnode, 'subject_id', bbregister, 'subject_id')
register.connect(inputnode, 'mean_image', bbregister, 'source_file')
register.connect(inputnode, 'subjects_dir', bbregister, 'subjects_dir')

# Create a mask of the median coregistered to the anatomical image
mean2anat_mask = Node(fsl.BET(mask=True), name='mean2anat_mask')
register.connect(bbregister, 'registered_file', mean2anat_mask, 'in_file')

use aparc+aseg's brain mask
```

(continues on next page)
binarize = Node(
    fs.Binarize(min=0.5, out_type="nii.gz", dilate=1),
    name="binarize_aparc")
register.connect(fssource, ("aparc_aseg", get_aparc_aseg), binarize,
    "in_file")

stripper = Node(fsl.ApplyMask(), name='stripper')
register.connect(binarize, "binary_file", stripper, "mask_file")
register.connect(convert, 'out_file', stripper, 'in_file')

""" Apply inverse transform to aparc file """
aparcxfm = Node(
    freesurfer.ApplyVolTransform(inverse=True, interp='nearest'),
    name='aparc_inverse_transform')
register.connect(inputnode, 'subjects_dir', aparcxfm, 'subjects_dir')
register.connect(bbregister, 'out_reg_file', aparcxfm, 'reg_file')
register.connect(fssource, ('aparc_aseg', get_aparc_aseg), aparcxfm,
    'target_file')
register.connect(inputnode, 'mean_image', aparcxfm, 'source_file')

""" Convert the BBRegister transformation to ANTS ITK format """
convert2itk = Node(C3dAffineTool(), name='convert2itk')
convert2itk.inputs.fsl2ras = True
convert2itk.inputs.itk_transform = True
register.connect(bbregister, 'out_fsl_file', convert2itk, 'transform_file')
register.connect(inputnode, 'mean_image', convert2itk, 'source_file')
register.connect(stripper, 'out_file', convert2itk, 'reference_file')

""" Compute registration between the subject's structural and MNI template """
reg = Node(ants.Registration(), name='antsRegister')
reg.inputs.output_transform_prefix = "output_"
reg.inputs.transforms = ['Rigid', 'Affine', 'SyN']
reg.inputs.transform_parameters = [(0.1, ), (0.1, ), (0.2, 3.0, 0.0)]
reg.inputs.number_of_iterations = [[[10000, 11110, 11110]] * 2 + [[[100, 30, 20]]]]
reg.inputs.dimension = 3
reg.inputs.write_composite_transform = True
reg.inputs.collapse_output_transforms = True
reg.inputs.initial_moving_transform_com = True
reg.inputs.metric = ['Mattes'] * 2 + [['Mattes', 'CC']]
reg.inputs.metric_weight = [1] * 2 + [[0.5, 0.5]]
reg.inputs.radius_or_number_of_bins = [32] * 2 + [[32, 4]]
reg.inputs.sampling_strategy = ['Regular'] * 2 + [[None, None]]
reg.inputs.sampling_percentage = [0.3] * 2 + [[None, None]]

(continues on previous page)
Concatenate the affine and ants transforms into a list

```python
merge = Node(Merge(2), iterfield=['in2'], name='mergexfm')
register.connect(convert2itk, 'itk_transform', merge, 'in2')
register.connect(reg, 'composite_transform', merge, 'in1')
```

Transform the mean image. First to anatomical and then to target

```python
warpmean = Node(ants.ApplyTransforms(), name='warpmean')
warpmean.inputs.input_image_type = 0
warpmean.inputs.interpolation = 'Linear'
warpmean.inputs.invert_transform_flags = [False, False]
warpmean.terminal_output = 'file'
warpmean.inputs.args = '--float'
# warpmean.inputs.num_threads = 4
# warpmean.plugin_args = {'sbatch_args': '--mem=4G -c 4'}
```

Transform the remaining images. First to anatomical and then to target

```python
warpall = pe.MapNode(
    ants.ApplyTransforms(), iterfield=['input_image'], name='warpall')
warpall.inputs.input_image_type = 0
warpall.inputs.interpolation = 'Linear'
warpall.inputs.invert_transform_flags = [False, False]
warpall.terminal_output = 'file'
warpall.inputs.args = '--float'
warpall.inputs.num_threads = 2
warpall.plugin_args = {'sbatch_args': '--mem=6G -c 2'}
```
register.connect(inputnode, 'source_files', warpall, 'input_image')
register.connect(merge, 'out', warpall, 'transforms')

""
Assign all the output files
""

register.connect(reg, 'warped_image', outputnode, 'anat2target')
register.connect(aparcxfm, 'transformed_file', outputnode, 'aparc')
register.connect(bbregister, 'out_fsl_file', outputnode, 'func2anat_transform')
register.connect(bbregister, 'out_reg_file', outputnode, 'out_reg_file')
register.connect(bbregister, 'min_cost_file', outputnode, 'min_cost_file')
register.connect(mean2anat_mask, 'mask_file', outputnode, 'mean2anat_mask')
register.connect(reg, 'composite_transform', outputnode, 'anat2target_transform')
register.connect(merge, 'out', outputnode, 'transforms')
return register

Get info for a given subject

def get_subjectinfo(subject_id, base_dir, task_id, model_id):
    """Get info for a given subject
    Parameters
    ----------
    subject_id : string
        Subject identifier (e.g., sub001)
    base_dir : string
        Path to base directory of the dataset
    task_id : int
        Which task to process
    model_id : int
        Which model to process
    Returns
    -------
    run_ids : list of ints
        Run numbers
    conds : list of str
        Condition names
    TR : float
        Repetition time
    """

    from glob import glob
    import os
    import numpy as np
    condition_info = []
    cond_file = os.path.join(base_dir, 'models', 'model%03d' % model_id, 'condition_key.txt')
    with open(cond_file, 'rt') as fp:
        for line in fp:
            info = line.strip().split()
            condition_info.append([info[0], info[1], ' '.join(info[2:])])
    if len(condition_info) == 0:
        raise ValueError('No condition info found in %s % cond_file')
taskinfo = np.array(condition_info)
n_tasks = len(np.unique(taskinfo[:, 0]))
conds = []
run_ids = []
if task_id > n_tasks:
    raise ValueError('Task id %d does not exist' % task_id)
for idx in range(n_tasks):
    taskidx = np.where(taskinfo[:, 0] == 'task%03d' % (idx + 1))
    conds.append(
        condition.replace(' ', '_')
        for condition in taskinfo[taskidx[0], 2]
    )
    files = sorted(
        glob(
            os.path.join(base_dir, subject_id, 'BOLD',
            'task%03d_run*' % (idx + 1)))
        for val in files)
    runs = [int(val[-3:]) for val in files]
    run_ids.insert(idx, runs)
json_info = os.path.join(base_dir, subject_id, 'BOLD', 'task%03d_run%03d'
    % (task_id, run_ids[task_id - 1][0]), 'bold_scaninfo.json')
if os.path.exists(json_info):
    import json
    with open(json_info, 'rt') as fp:
        data = json.load(fp)
        TR = data['global']['const']['RepetitionTime'] / 1000.
else:
    task_scan_key = os.path.join(
        base_dir, subject_id, 'BOLD', 'task%03d_run%03d'
        % (task_id, run_ids[task_id - 1][0]), 'scan_key.txt')
    if os.path.exists(task_scan_key):
        TR = np.genfromtxt(task_scan_key)[1]
    else:
        TR = np.genfromtxt(os.path.join(base_dir, 'scan_key.txt'))[1]
return run_ids[task_id - 1], conds[task_id - 1], TR

**Analyze an open fmri dataset**

```python
def analyze_openfmri_dataset(data_dir, 
    subject=None, 
    model_id=None, 
    task_id=None, 
    output_dir=None, 
    subj_prefix='*', 
    hpcutoff=120., 
    use_derivatives=True, 
    fwhm=6.0, 
    subjects_dir=None, 
    target=None):

    """Analyze an open fmri dataset

    Parameters
    ----------

    data_dir : str
        Path to the base data directory

    (continues on next page)```
work_dir : str
    Nipype working directory (defaults to cwd)

    
    Load nipype workflows
    
    preproc = create_featreg_preproc(whichvol='first')
    modelfit = create_modelfit_workflow()
    fixed_fx = create_fixed_effects_flow()
    if subjects_dir:
        registration = create_fs_reg_workflow()
    else:
        registration = create_reg_workflow()

    Remove the plotting connection so that plot iterables don’t propagate
to the model stage

    preproc.disconnect(
        preproc.get_node('plot_motion'), 'out_file',
        preproc.get_node('outputspec'), 'motion_plots')

    Set up openfmri data specific components

    subjects = sorted([
        path.split(os.path.sep)[-1]
        for path in glob(os.path.join(data_dir, subj_prefix))
    ])

    infosource = pe.Node(
        niu.IdentityInterface(fields=["subject_id", 'model_id', 'task_id'])),
        name='infosource')
    if len(subject) == 0:
        infosource.iterables = ["subject_id", subjects],
            ('model_id', [model_id]), ('task_id', task_id]
    else:
        infosource.iterables = ["subject_id", [subjects[subjects.index(subj)] for subj in subject ],
            ('model_id', [model_id]), ('task_id', task_id])

    subjinfo = pe.Node(
        niu.Function(
            input_names=['subject_id', 'base_dir', 'task_id', 'model_id'],
            output_names=['run_id', 'conds', 'TR'],
            function=get_subjectinfo),
            name='subjectinfo')
    subjinfo.inputs.base_dir = data_dir

    Return data components as anat, bold and behav

    contrast_file = os.path.join(data_dir, 'models', 'model%03d' % model_id,
        'task_contrasts.txt')
    has_contrast = os.path.exists(contrast_file)
    if has_contrast:
datascource = pe.Node(
    nio.DataGrabber(
        infields=['subject_id', 'run_id', 'task_id', 'model_id'],
        outfields=['anat', 'bold', 'behav', 'contrasts'],
        name='datasource')
else:
    datasource = pe.Node(
        nio.DataGrabber(
            infields=['subject_id', 'run_id', 'task_id', 'model_id'],
            outfields=['anat', 'bold', 'behav'],
            name='datasource')
    datasource.inputs.base_directory = data_dir
    datasource.inputs.template = '*'

if has_contrast:
    datasource.inputs.field_template = {
        'anat': '$s/anatomy/T1_001.nii.gz',
        'bold': '$s/BOLD/task%03d_r*/bold.nii.gz',
        'behav': ('$s/model/model%03d/onsets/task%03d_'
                  'run%03d/cond*.*'),
        'contrasts': ('models/model%03d/'
                      'task_contrasts.txt')
    }
    datasource.inputs.template_args = {
        'anat': ['subject_id'],
        'bold': ['subject_id', 'task_id'],
        'behav': ['subject_id', 'model_id', 'task_id', 'run_id'],
        'contrasts': ['model_id']
    }
else:
    datasource.inputs.field_template = {
        'anat': '$s/anatomy/T1_001.nii.gz',
        'bold': '$s/BOLD/task%03d_r*/bold.nii.gz',
        'behav': ('$s/model/model%03d/onsets/task%03d_'
                  'run%03d/cond*.*')
    }
    datasource.inputs.template_args = {
        'anat': ['subject_id'],
        'bold': ['subject_id', 'task_id'],
        'behav': ['subject_id', 'model_id', 'task_id', 'run_id']
    }

data sourcedata.inputs.sort_filelist = True

Create meta workflow

"""
wf = pe.Workflow(name='openfmri')
wf.connect(infosource, 'subject_id', subjinfo, 'subject_id')
wf.connect(infosource, 'model_id', subjinfo, 'model_id')
wf.connect(infosource, 'task_id', subjinfo, 'task_id')
wf.connect(infosource, 'subject_id', datasource, 'subject_id')
wf.connect(infosource, 'model_id', datasource, 'model_id')
wf.connect(infosource, 'task_id', datasource, 'task_id')
wf.connect(subjinfo, 'run_id', datasource, 'run_id')
wf.connect((
    datasource, preproc, [('bold', 'inputspec.func')]),
(continues on next page)
def get_highpass(TR, hpcutoff):
    return hpcutoff / (2. * TR)

def get_contrasts(contrast_file, task_id, conds):
    import numpy as np
    import os
    contrast_def = []
    if os.path.exists(contrast_file):
        with open(contrast_file, 'rt') as fp:
            contrast_def.extend([np.array(row.split()) for row in fp.readlines() if row.strip()])
    contrasts = []
    for row in contrast_def:
        if row[0] != 'task%03d' % task_id:
            continue
        con = [row[1], 'T', ['cond%03d' % (i + 1) for i in range(len(conds))], row[2:].astype(float).tolist()]
        contrasts.append(con)

    # add auto contrasts for each column
    for i, cond in enumerate(conds):
        con = [cond, 'T', ['cond%03d' % (i + 1)], [1]]
        contrasts.append(con)

    return contrasts

def get_contrasts(contrast_file, task_id, conds):
    import numpy as np
    import os
    contrast_def = []
    if os.path.exists(contrast_file):
        with open(contrast_file, 'rt') as fp:
            contrast_def.extend([np.array(row.split()) for row in fp.readlines() if row.strip()])
    contrasts = []
    for row in contrast_def:
        if row[0] != 'task%03d' % task_id:
            continue
        con = [row[1], 'T', ['cond%03d' % (i + 1) for i in range(len(conds))], row[2:].astype(float).tolist()]
        contrasts.append(con)

    # add auto contrasts for each column
    for i, cond in enumerate(conds):
        con = [cond, 'T', ['cond%03d' % (i + 1)], [1]]
        contrasts.append(con)

    return contrasts

def get_contrasts(contrast_file, task_id, conds):
    import numpy as np
    import os
    contrast_def = []
    if os.path.exists(contrast_file):
        with open(contrast_file, 'rt') as fp:
            contrast_def.extend([np.array(row.split()) for row in fp.readlines() if row.strip()])
    contrasts = []
    for row in contrast_def:
        if row[0] != 'task%03d' % task_id:
            continue
        con = [row[1], 'T', ['cond%03d' % (i + 1) for i in range(len(conds))], row[2:].astype(float).tolist()]
        contrasts.append(con)

    # add auto contrasts for each column
    for i, cond in enumerate(conds):
        con = [cond, 'T', ['cond%03d' % (i + 1)], [1]]
        contrasts.append(con)

    return contrasts

def get_contrasts(contrast_file, task_id, conds):
    import numpy as np
    import os
    contrast_def = []
    if os.path.exists(contrast_file):
        with open(contrast_file, 'rt') as fp:
            contrast_def.extend([np.array(row.split()) for row in fp.readlines() if row.strip()])
    contrasts = []
    for row in contrast_def:
        if row[0] != 'task%03d' % task_id:
            continue
        con = [row[1], 'T', ['cond%03d' % (i + 1) for i in range(len(conds))], row[2:].astype(float).tolist()]
        contrasts.append(con)

    # add auto contrasts for each column
    for i, cond in enumerate(conds):
        con = [cond, 'T', ['cond%03d' % (i + 1)], [1]]
        contrasts.append(con)

    return contrasts

def get_contrasts(contrast_file, task_id, conds):
    import numpy as np
    import os
    contrast_def = []
    if os.path.exists(contrast_file):
        with open(contrast_file, 'rt') as fp:
            contrast_def.extend([np.array(row.split()) for row in fp.readlines() if row.strip()])
    contrasts = []
    for row in contrast_def:
        if row[0] != 'task%03d' % task_id:
            continue
        con = [row[1], 'T', ['cond%03d' % (i + 1) for i in range(len(conds))], row[2:].astype(float).tolist()]
        contrasts.append(con)

    # add auto contrasts for each column
    for i, cond in enumerate(conds):
        con = [cond, 'T', ['cond%03d' % (i + 1)], [1]]
        contrasts.append(con)

    return contrasts
iterfield=['realigned_files', 'realignment_parameters', 'mask_file'],
    name="art")
modelspec = pe.Node(interface=model.SpecifyModel(), name="modelspec")
modelspec.inputs.input_units = 'secs'

def check_behav_list(behav, run_id, conds):
    import numpy as np
    num_conds = len(conds)
    if isinstance(behav, (str, bytes)):
        behav = [behav]
    behav_array = np.array(behav).flatten()
    num_elements = behav_array.shape[0]
    return behav_array.reshape(int(num_elements / num_conds),
                                num_conds).tolist()

reshape_behav = pe.Node(  
    niu.Function(  
        input_names=['behav', 'run_id', 'conds'],
        output_names=['behav'],
        function=check_behav_list),  
    name='reshape_behav')

wf.connect(subjinfo, 'TR', modelspec, 'time_repetition')
wf.connect(datasource, 'behav', reshape_behav, 'behav')
wf.connect(subjinfo, 'run_id', reshape_behav, 'run_id')
wf.connect(subjinfo, 'conds', reshape_behav, 'conds')
wf.connect(reshape_behav, 'behav', modelspec, 'event_files')
wf.connect(subjinfo, 'TR', modelfit, 'inputspec.interscan_interval')
wf.connect(subjinfo, 'conds', contrastgen, 'conds')
if has_contrast:
    wf.connect(datasource, 'contrasts', contrastgen, 'contrast_file')
else:
    contrastgen.inputs.contrast_file = ''
wf.connect(infosource, 'task_id', contrastgen, 'task_id')
wf.connect(contrastgen, 'contrasts', modelfit, 'inputspec.contrasts')
wf.connect([(preproc, art,  
              [('outputspec.motion_parameters', 'realignment_parameters'),  
               ('outputspec.realigned_files',  
                 'realigned_files'), ('outputspec.mask', 'mask_file')]),  
              (preproc, modelspec,  
               [('outputspec.highpassed_files', 'functional_runs'),  
                ('outputspec.motion_parameters', 'realignment_parameters')]),  
              (art, modelspec,  
               [('outlier_files', 'outlier_files')]), (modelspec, modelfit, [  
                ('session_info', 'inputspec.session_info')  
              ]), (preproc, modelfit, [('outputspec.highpassed_files',  
               'inputspec.functional_data')])])

# Compute TSNR on realigned data regressing polynomials up to order 2
# Comute TSNR on realigned data regressing polynomials up to order 2
tsnr = MapNode(TSNR(regress_poly=2), iterfield=['in_file'], name='tsnr')
wf.connect(preproc, "outputspec.realigned_files", tsnr, "in_file")

# Compute the median image across runs
calc_median = Node(CalculateMedian(), name='median')
Reorder the copes so that now it combines across runs

```python
def sort_copes(copes, varcopes, contrasts):
    import numpy as np
    if not isinstance(copes, list):
        copes = [copes]
    varcopes = [varcopes]
    num_copes = len(contrasts)
    n_runs = len(copes)
    all_copes = np.array(copes).flatten()
    all_varcopes = np.array(varcopes).flatten()
    outcopes = all_copes.reshape(int(len(all_copes) / num_copes), num_copes).T.tolist()
    outvarcopes = all_varcopes.reshape(int(len(all_varcopes) / num_copes), num_copes).T.tolist()
    return outcopes, outvarcopes, n_runs

cope_sorter = pe.Node(
    niu.Function(
        input_names=['copes', 'varcopes', 'contrasts'],
        output_names=['copes', 'varcopes', 'n_runs'],
        function=sort_copes),
    name='cope_sorter')
```

```python
pickfirst = lambda x: x[0]
```

```python
wf.connect(contrastgen, 'contrasts', cope_sorter, 'contrasts')
```

```python
wf.connect((preproc, fixed_fx, [(('outputspec.mask', pickfirst), 'flameo.mask_file')], (modelfit, cope_sorter, [(('outputspec.copes', 'copes')], (cope_sorter, fixed_fx, [(('copes', 'inputspec.copes'), ('varcopes', 'inputspec.varcopes'), ('n_runs', 'l2model.num_copes')], (modelfit, fixed_fx, [(('outputspec.dof_file', 'inputspec.dof_files')])]))
```

```python
wf.connect(calc_median, 'median_file', registration, ('inputspec.mean_image')
```

```python
if subjects_dir:
    wf.connect(infosource, 'subject_id', registration, ('inputspec.subject_id'))
registration.inputs.inputspec.subjects_dir = subjects_dir
registration.inputs.inputspec.target_image = fsl.Info.standard_image('MN152_T1_2mm_brain.nii.gz')
else:
    registration.inputs.inputspec.target_image = target
```

(continues on next page)
registration.inputs.inputspec.target_image_brain = fsl.Info.standard_image('MNI152_T1_2mm_brain.nii.gz')
registration.inputs.inputspec.config_file = 'T1_2_MNI152_2mm'

```python
def merge_files(copes, varcopes, zstats):
    out_files = []
    splits = []
    out_files.extend(copes)
    splits.append(len(copes))
    out_files.extend(varcopes)
    splits.append(len(varcopes))
    out_files.extend(zstats)
    splits.append(len(zstats))
    return out_files, splits
```

```python
mergefunc = pe.Node(  
    niu.Function(  
        input_names=['copes', 'varcopes', 'zstats'],
        output_names=['out_files', 'splits'],
        function=merge_files),
    name='merge_files')
wf.connect((fixed_fx.get_node('outputspec'), mergefunc, [  
    ('copes', 'copes'),
    ('varcopes', 'varcopes'),
    ('zstats', 'zstats'),
  ]))
wf.connect(mergefunc, 'out_files', registration, 'inputspec.source_files')
```

```python
def split_files(in_files, splits):
    copes = in_files[:splits[0]]
    varcopes = in_files[splits[0]:(splits[0] + splits[1])]
    zstats = in_files[(splits[0] + splits[1]):]
    return copes, varcopes, zstats
```

```python
splitfunc = pe.Node(  
    niu.Function(  
        input_names=['in_files', 'splits'],
        output_names=['copes', 'varcopes', 'zstats'],
        function=split_files),
    name='split_files')
wf.connect(mergefunc, 'splits', splitfunc, 'splits')
wf.connect(registration, 'outputspec.transformed_files', splitfunc, 'in_files')
```

```python
if subjects_dir:
    get_roi_mean = pe.MapNode(  
        fs.SegStats(default_color_table=True),
        iterfield=['in_file'],
        name='get_aparc_means')
    get_roi_mean.inputs.avgwf_txt_file = True
    wf.connect(
        fixed_fx.get_node('outputspec'), 'copes', get_roi_mean, 'in_file')
    wf.connect(registration, 'outputspec.aparc', get_roi_mean, 'segmentation_file')
    get_roi_tsnr = pe.MapNode(  
        (continues on next page)
fs.SegStats(default_color_table=True),
iterfield=['in_file'],
name='get_aparc_tsnr')
get_roi_tsnr.inputs.avgwftxt_file = True
wf.connect(tsnr, 'tsnr_file', get_roi_tsnr, 'in_file')
wf.connect(registration, 'outputspec.aparc', get_roi_tsnr,
'segmentation_file')

"""
Connect to a datasink
"""
def get_subs(subject_id, conds, run_id, model_id, task_id):
    subs = [('_subject_id_' % subject_id, '')]
    subs.append(('model_id_%d' % model_id, 'model%d' % model_id))
    subs.append(('task_id_%d' % task_id, '/task%d' % task_id))
    subs.append(('bold_dtype_mcf_mask_smooth_mask_gms_tempfilt_mean_warp',
'mean'))
    subs.append(('bold_dtype_mcf_mask_smooth_mask_gms_tempfilt_mean_flirt',
'affine'))
    for i in range(len(conds)):
        subs.append(('flameo%d/cope1.' % i, 'cope%d.' % (i + 1)))
        subs.append(('flameo%d/varcope1.' % i, 'varcope%d.' % (i + 1)))
        subs.append(('flameo%d/zstat1.' % i, 'zstat%d.' % (i + 1)))
        subs.append(('warpall%d/cope1_warp.' % i, 'cope%d.' % (i + 1)))
        subs.append(('warpall%d/varcope1_warp.' % i, 'varcope%d.' % (i + 1)))
        subs.append(('warpall%d/zstat1_warp.' % (2 * len(conds) + i),
'zstat%d.' % (i + 1)))
        subs.append(('__get_aparc_means%d/cope1_' % (i + 1)))
    for i, run_num in enumerate(run_id):
        subs.append(('__get_aparc_tsnr%d/cope1' % i, '/run%d_' % (i + 1)))
        subs.append(('__art%d/cope1' % i, '/run%d_' % (i + 1)))
        subs.append(('__dilatemask%d/cope1' % i, '/run%d_' % (i + 1)))
        subs.append(('__realign%d/cope1' % i, '/run%d_' % (i + 1)))
        subs.append(('__modelgen%d/cope1' % i, '/run%d_' % (i + 1)))
        subs.append(('/model%d/task%d/' % (model_id, task_id), '/'))
        subs.append(('/model%d/task%d/' % (model_id, task_id), '/'))
        subs.append(('/median_flirt_brain_mask', 'median_brain_mask'))
        return subs
subsgen = pe.Node(niu.Function(
    input_names=['subject_id', 'conds', 'run_id', 'model_id', 'task_id'],
), (continues on previous page))
output_names=['substitutions'],
    function=get_subs),
    name='subsgen')
wf.connect(subjinfo, 'run_id', subsgen, 'run_id')

datasink = pe.Node(interface=nio.DataSink(), name="datasink")
wf.connect(infosource, 'subject_id', datasink, 'container')
wf.connect(infosource, 'subject_id', subsgen, 'subject_id')
wf.connect(infosource, 'model_id', subsgen, 'model_id')
wf.connect(infosource, 'task_id', subsgen, 'task_id')
wf.connect(contrastgen, 'contrasts', subsgen, 'conds')
wf.connect(subsgen, 'substitutions', datasink, 'substitutions')
wf.connect(
    [(fixed_fx.get_node('outputspec'), datasink,
      [('res4d', 'res4d'), ('copes', 'copes'), ('varcopes', 'varcopes'),
       ('zstats', 'zstats'), ('tstats', 'tstats')])],
    [(modelfit.get_node('modelgen'), datasink, 
      ['design_cov', 'qa.model'],
      ['design_image', 'qa.model.@matrix_image'],
      ['design_file', 'qa.model.@matrix'])])
wf.connect(
    [(preproc, datasink, [('outputspec.motion_parameters', 'qa.motion'),
      ('outputspec.motion_plots', 'qa.motion.plots'),
      ('outputspec.mask', 'qa.mask')]),
    [(registration, 'outputspec.mean2anat_mask', datasink, 'qa.mask.mean2anat'),
    (art, 'norm_files', datasink, 'qa.art.@norm')]
    [(art, 'intensity_files', datasink, 'qa.art.@intensity')]
    [(art, 'outlier_files', datasink, 'qa.art.@outlier_files')]
    [(registration, 'outputspec.anat2target', datasink, 'qa.anat2target')]
    [(tsnr, 'tsnr_file', datasink, 'qa.tsnr.@map')]
    if subjects_dir:
        [(registration, 'outputspec.min_cost_file', datasink, 'qa.mincost')]
        [(get_roi_tsnr, datasink, [('avgwf_txt_file', 'qa.tsnr'),
          ('summary_file', 'qa.tsnr.@summary')])]
        [(get_roi_mean, datasink, [('avgwf_txt_file', 'copes.roi'),
          ('summary_file', 'copes.roi.@summary')])]
        [(splitfunc, datasink, [('copes', 'copes.mni'),
          ('varcopes', 'varcopes.mni'),
          ('zstats', 'zstats.mni')])]
        [(calc_median, 'median_file', datasink, 'mean')]
        [(registration, 'outputspec.transformed_mean', datasink, 'mean.mni')]
        [(registration, 'outputspec.func2anat_transform', datasink, 'xfm.mean2anat')]
        [(registration, 'outputspec.anat2target_transform', datasink, 'xfm.anat2target')]
"
Set processing parameters
"
The following functions run the whole workflow.

```python
if __name__ == '__main__':
    import argparse
defstr = ' (default $%(default)s)'
parser = argparse.ArgumentParser(
    prog='fmri_openfmri.py', description=__doc__)
parser.add_argument('-d', '--datasetdir', required=True)
parser.add_argument(
    '-s',
    '--subject',
    default=[],
    nargs='+',
    type=str,
    help='Subject name (e.g. 'sub001')")
parser.add_argument(
    '-m', '--model', default=1, help='Model index' + defstr)
parser.add_argument(
    '-x',
    '--subjectprefix',
    default='sub*',
    help='Subject prefix' + defstr)
parser.add_argument(
    '-t',
    '--task',
    default=1,  # nargs='+',
    type=int,
    help='Task index' + defstr)
parser.add_argument(
    '--hpfilter',
    default=120.,
    type=float,
    help='High pass filter cutoff (in secs)' + defstr)
parser.add_argument(
    '--fwhm', default=6.,
    type=float,
    help='Spatial FWHM' + defstr)
parser.add_argument(
    '--derivatives', action='store_true',
    help='Use derivatives' + defstr)
parser.add_argument(
    '-o', '--output_dir', dest='outdir',
    help='Output directory base')
parser.add_argument(
    '-w', '--work_dir', dest='work_dir',
    help='Output directory base')
parser.add_argument(
    '-p',
    '--plugin',
    dest='plugin',
    default='Linear',
...
help="Plugin to use")
parser.add_argument("--plugin_args", dest="plugin_args", help="Plugin arguments")
parser.add_argument("--sd",
dest="subjects_dir",
help="FreeSurfer subjects directory (if available)")
parser.add_argument("--target",
dest="target_file",
help="(Target in MNI space. Best to use the MindBoggle "
"template - only used with FreeSurfer"
"OASIS-30_Atropos_template_in_MNI152_2mm.nii.gz")")
args = parser.parse_args()
outdir = args.outdir
work_dir = os.getcwd()
if args.work_dir:
    work_dir = os.path.abspath(args.work_dir)
if outdir:
    outdir = os.path.abspath(outdir)
else:
    outdir = os.path.join(work_dir, 'output')
outdir = os.path.join(outdir, 'model%02d' % int(args.model),
'task%03d' % int(args.task))
derivatives = args.derivatives
if derivatives is None:
    derivatives = False
wf = analyze_openfmri_dataset(
data_dir=os.path.abspath(args.datasetdir),
subject=args.subject,
model_id=int(args.model),
task_id=[int(args.task)],
subj_prefix=args.subjectprefix,
output_dir=outdir,
hpcutoff=args.hpfilter,
use_derivatives=derivatives,
fwhm=args.fwhm,
subjects_dir=args.subjects_dir,
target=args.target_file)
    # wf.config['execution']['remove_unnecessary_outputs'] = False
wf.base_dir = work_dir
if args.plugin_args:
    wf.run(args.plugin, plugin_args=eval(args.plugin_args))
else:
    wf.run(args.plugin)

Example source code
You can download the full source code of this example. This same script is also included in
the Nipype source distribution under the examples directory.
This tutorial illustrates how to perform surface-based smoothing of cortical data using FreeSurfer and then perform first-level model and contrast estimation using SPM. A surface-based second level glm illustrates the use of spherical registration and freesurfer’s glm functions.

### 19.1 Preparing environment

#### 19.1.1 Step 0

In order to run this tutorial you need to have SPM and FreeSurfer tools installed and accessible from matlab/command line. Check by calling mri_info from the command line.

#### 19.1.2 Step 1

Link the fsaverage directory for your freesurfer distribution. To do this type:

```bash
cd nipype-tutorial/fsdata
ln -s $FREESURFER_HOME/subjects/fsaverage
cd ..
```

### 19.2 Defining the workflow

```python
from __future__ import print_function
from builtins import str
from builtins import range
import os   # system functions
import nipype.algorithms.modelgen as model   # model generation
import nipype.algorithms.rapidart as ra      # artifact detection
import nipype.interfaces.freesurfer as fs     # freesurfer
import nipype.interfaces.io as nio            # i/o routines
import nipype.interfaces.matlab as mlab       # how to run matlab
import nipype.interfaces.spm as spm           # spm
```

(continues on next page)
import nipype.interfaces.utility as util # utility
import nipype.pipeline.engine as pe # pipeline engine

19.2.1 iminaries

Set any package specific configuration. Setting the subjects directory and the appropriate matlab command to use, if you want to use a different spm version/path, it should also be entered here. These are currently being set at the class level, so every node will inherit these settings. However, these can also be changed or set for an individual

```python
# Tell freesurfer what subjects directory to use
subjects_dir = os.path.abspath('fsdata')
sf.FSCommand.set_default_subjects_dir(subjects_dir)

# Set the way matlab should be called
mlab.MatlabCommand.set_default_matlab_cmd("matlab -nodesktop -nosplash")
# If SPM is not in your MATLAB path you should add it here
mlab.MatlabCommand.set_default_paths('/software/spm8')
```

19.2.2 eprocessing workflow

preproc = pe.Workflow(name='preproc')

Use nipype.interfaces.spm.Realign for motion correction and register all images to the mean image.

```python
realign = pe.Node(interface=spm.Realign(), name="realign")
realign.inputs.register_to_mean = True
```

Use nipype.algorithms.rapidart to determine which of the images in the functional series are outliers based on deviations in intensity or movement.

```python
art = pe.Node(interface=ra.ArtifactDetect(), name="art")
art.inputs.use_differences = [True, False]
art.inputs.use_norm = True
art.inputs.norm_threshold = 1
art.inputs.zintensity_threshold = 3
art.inputs.mask_type = 'file'
art.inputs.parameter_source = 'SPM'
```

Use nipype.interfaces.freesurfer.BBRegister to coregister the mean functional image generated by realign to the subjects’ surfaces.

```python
surfregister = pe.Node(interface=fs.BBRegister(), name='surfregister')
surfregister.inputs.init = 'fsl'
surfregister.inputs.contrast_type = 't2'
```

Use nipype.interfaces.io.FreeSurferSource to retrieve various image files that are automatically generated by the recon-all process.

```python
FreeSurferSource = pe.Node(interface=nio.FreeSurferSource(), name='fssource')
```

Use nipype.interfaces.freesurfer.ApplyVolTransform to convert the brainmask generated by freesurfer into the realigned functional space.

```python
ApplyVolTransform = pe.Node(interface=fs.ApplyVolTransform(), name='applyreg')
ApplyVolTransform.inputs.inverse = True
```

Use nipypeinterfaces.freesurfer.Binarizetoe xtract a binary brain mask.
Threshold = pe.Node(interface=fs.Binarize(), name='threshold')
Threshold.inputs.min = 10
Threshold.inputs.out_type = 'nii'

Two different types of functional data smoothing are performed in this workflow. The volume smoothing option performs a standard SPM smoothing using nipype.interfaces.spm.Smooth. In addition, we use a smoothing routine from freesurfer (nipype.interfaces.freesurfer.Binarize) to project the functional data from the volume to the subjects’ surface, smooth it on the surface and fit it back into the volume forming the cortical ribbon. The projection uses the average value along a “cortical column”. In addition to the surface smoothing, the rest of the volume is smoothed with a 3d gaussian kernel.

Note: It is very important to note that the projection to the surface takes a 3d manifold to a 2d manifold. Hence the reverse projection, simply fills the thickness of cortex with the smoothed data. The smoothing is not performed in a depth specific manner. The output of this branch should only be used for surface-based analysis and visualization.

volsmooth = pe.Node(interface=spm.Smooth(), name="volsmooth")
surfsmooth = pe.MapNode(
  interface=fs.Smooth(proj_frac_avg=(0, 1, 0.1)),
  name="surfsmooth",
  iterfield=['in_file'])

We connect up the different nodes to implement the preprocessing workflow.

preproc.connect(
  [(realign, surfregister, [('mean_image', 'source_file')]),
   (FreeSurferSource, ApplyVolTransform, [('brainmask', 'target_file')]),
   (surfregister, ApplyVolTransform, [('out_reg_file', 'reg_file')]),
   (realign, ApplyVolTransform, [('mean_image', 'source_file')]),
   (ApplyVolTransform, Threshold, [('transformed_file', 'in_file')]),
   (realign, art, [('realignment_parameters', 'realignment_parameters'),
                   ('realigned_files', 'realigned_files')]),
   (Threshold, art, [('binary_file', 'mask_file')]),
   (realign, volsmooth, [('realigned_files', 'in_files')]),
   (realign, surfsmooth, [('realigned_files', 'in_file')]),
   (surfregister, surfsmooth, [('out_reg_file', 'reg_file')])
)

19.2.3 Set up volume analysis workflow

volanalysis = pe.Workflow(name='volanalysis')

Generate SPM-specific design information using nipype.interfaces.spm.SpecifyModel.

modelspec = pe.Node(interface=model.SpecifySPMModel(), name="modelspec")
modelspec.inputs.concatenate_runs = True

Generate a first level SPM.mat file for analysis nipype.interfaces.spm.Level1Design.

level1design = pe.Node(interface=spm.Level1Design(), name="level1design")
level1design.inputs.bases = {'hrf': ['level1design']}

Use nipype.interfaces.spm.EstimateModel to determine the parameters of the model.

level1estimate = pe.Node(interface=spm.EstimateModel(), name="level1estimate")
level1estimate.inputs.estimation_method = ['level1estimate']
Use `nipype.interfaces.spm.EstimateContrast` to estimate the first level contrasts specified in a few steps above.

```python
contrastestimate = pe.Node(
    interface=spm.EstimateContrast(), name="contrastestimate")
```

```python
volanalysis.connect([  
    (modelspec, level1design, [('session_info', 'session_info')]),  
    (level1design, level1estimate, [('spm_mat_file', 'spm_mat_file')]),  
    (level1estimate, contrastestimate,  
        [('spm_mat_file', 'spm_mat_file'), ('beta_images', 'beta_images'),  
        ('residual_image', 'residual_image')])  
])
```

### 19.2.4 Set up surface analysis workflow

We simply clone the volume analysis workflow.

```python
surfanalysis = volanalysis.clone(name='surfanalysis')
```

### 19.2.5 Set up volume normalization workflow

The volume analysis is performed in individual space. Therefore, post analysis we normalize the contrast images to MNI space.

```python
volnorm = pe.Workflow(name='volnormconimages')
```

Use `nipype.interfaces.freesurfer.MRIConvert` to convert the brainmask, an mgz file and the contrast images (nifti-1 img hdr pairs), to single volume nifti images.

```python
convert = pe.Node(interface=fs.MRIConvert(out_type='nii'), name='convert2nii')
convert2 = pe.MapNode(
    interface=fs.MRIConvert(out_type='nii'),
    iterfield=['in_file'],
    name='convertimg2nii')
```

Use `nipype.interfaces.spm.Segment` to segment the structural image and generate the transformation file to MNI space.

**Note:** Segment takes longer than usual because the nose is wrapped behind the head in the structural image.

```python
segment = pe.Node(interface=spm.Segment(), name='segment')
```

Use `nipype.interfaces.freesurfer.ApplyVolTransform` to convert contrast images into freesurfer space.

```python
normwreg = pe.MapNode(
    interface=fs.ApplyVolTransform(),
    iterfield=['source_file'],
    name='applyreg2con')
```

Use `nipype.interfaces.spm.Normalize` to normalize the contrast images to MNI space

```python
normalize = pe.Node(interface=spm.Normalize(jobtype='write'), name='norm2mni')
```

Connect up the volume normalization components
19.2.6 Preproc + Analysis + VolumeNormalization workflow

Connect up the lower level workflows into an integrated analysis. In addition, we add an input node that specifies all the inputs needed for this workflow. Thus, one can import this workflow and connect it to their own data sources. An example with the nifti-tutorial data is provided below.

For this workflow the only necessary inputs are the functional images, a freesurfer subject id corresponding to recon-all processed data, the session information for the functional runs and the contrasts to be evaluated.

```python
inputnode = pe.Node(
    interface=util.IdentityInterface(
        fields=['func', 'subject_id', 'session_info', 'contrasts'],
        name='inputnode')
)
```

Connect the components into an integrated workflow.

```python
llpipeline = pe.Workflow(name='firstlevel')
llpipeline.connect([
    (inputnode, preproc, [
        ('func', 'realign.in_files'),
        ('subject_id', 'surfregister.subject_id'),
        ('subject_id', 'fssource.subject_id'),
    ]),
    (inputnode, volanalysis, [(['session_info', 'modelspec.subject_info'],
        ('contrasts', 'contrastestimate.contrasts'))]),
    (inputnode, surfanalysis, [(['session_info', 'modelspec.subject_info'],
        ('contrasts', 'contrastestimate.contrasts'))]),
])
```

# attach volume and surface model specification and estimation components

```python
llpipeline.connect(
    (preproc, volanalysis,
        [('realign.realignment_parameters', 'modelspec.realignment_parameters'),
        ('volsmooth.smoothed_files', 'modelspec.focus的なruns'),
        ('art.outlier_files',
            'modelspec.outlier_files'),
        ('threshold.binary_file',
            'level1design.mask_image'))),

    (preproc, surfanalysis,
        [('realign.realignment_parameters', 'modelspec.realignment_parameters'),
        ('surfsmoooth.smoothed_file', 'modelspec.focus的なruns'),
        ('art.outlier_files',
            'modelspec.outlier_files'),
        ('threshold.binary_file',
            'level1design.mask_image'))]
)
```

# attach volume contrast normalization components

```python
llpipeline.connect([(preproc, volnorm,
    [('fssource.orig', 'convert2nii.in_file'),
    ('surfregister.out_reg_file', 'applyreq2con.reg_file'),
    ('fssource.orig', 'applyreq2con.target_file')]),
    (volanalysis, volnorm, [
        ('contrastestimate.con_images',
```

(continues on next page)
The nipype tutorial contains data for two subjects. Subject data is in two subdirectories, s1 and s2. Each subject directory contains four functional volumes: f3.nii, f5.nii, f7.nii, f10.nii. And mical volume named struct.nii. Below we set some variables to inform the datasource about the layout of our data. We specify the location of the data, the subject sub-directories and a dictionary that maps each run to a mnemonic (or field) for the run type (struct or func). These fields become the output fields of the datasource node in the pipeline. In the example below, run ‘f3’ is of type ‘func’ and gets mapped to a nifti filename through a template ‘%s.nii’. So ‘f3’ would become

```python
# Specify the location of the data.
data_dir = os.path.abspath('data')
# Specify the subject directories
subject_list = ['s1', 's3']
# Map field names to individual subject runs.
info = dict(func=[['subject_id', ['f3', 'f5', 'f7', 'f10']]],
            struct=[['subject_id', 'struct']])
```

Here we set up iteration over all the subjects. The following line is a particular example of the flexibility of the system. The datasource attribute iterables tells the pipeline engine that it should repeat the analysis on each of the items in the subject_list. In the current example, the entire first level preprocessing and estimation will be repeated for each subject contained in subject_list.

```python
infosource.iterables = ('subject_id', subject_list)
```

Now we create a nipype.interfaces.io.DataGrabber object and fill in the information from above about the layout of our data. The nipype.pipeline.NodeWrapper module wraps the interface object and provides additional housekeeping and pipeline specific functionality.

```python
datasource = pe.Node(
    interface=nio.DataGrabber(
        infields=['subject_id'], outfields=['func', 'struct'],
        name='datasource'),
datasource.inputs.base_directory = data_dir
datasource.inputs.template = '%s/%s.nii'
datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True
```

19.2.7 Set preprocessing parameters

```python
l1pipeline.inputs.preproc.fssource.subjects_dir = subjects_dir
l1pipeline.inputs.preproc.volsmooth.fwhm = 4
l1pipeline.inputs.preproc.surfsmooth.surface_fwhm = 5
l1pipeline.inputs.preproc.surfsmooth.vol_fwhm = 4
```

19.2.8 Experimental paradigm specific components

Here we create a function that returns subject-specific information about the experimental paradigm. This is used by the nipype.interfaces.spm.SpecifyModel to create the information necessary to generate an SPM design matrix. In this tutorial, the same paradigm was used for every participant.
def subjectinfo(subject_id):
    from nipype.interfaces.base import Bunch
    from copy import deepcopy
    print("Subject ID: %s\n" % str(subject_id))
    output = []
    names = ['Task-Odd', 'Task-Even']
    for r in range(4):
        onsets = [list(range(15, 240, 60)), list(range(45, 240, 60))]
        output.insert(r, Bunch(
            conditions=names,
            onsets=deepcopy(onsets),
            durations=[[15 for s in names]],
        ))
    return output

Setup the contrast structure that needs to be evaluated. This is a list of lists. The inner list specifies the contrasts and has the following format - [Name,Stat,[list of condition names],[weights on those conditions]. The condition names must match the names listed in the subjectinfo function described above.

cont1 = ('Task>Baseline', 'T', ['Task-Odd', 'Task-Even'], [0.5, 0.5])
cont2 = ('Task-Odd>Task-Even', 'T', ['Task-Odd', 'Task-Even'], [1, -1])
contrasts = [cont1, cont2]

19.2.9 Set up node specific inputs

We replicate the modelspec parameters separately for the surface- and volume-based analysis.

19.2.10 Setup the pipeline

The nodes created above do not describe the flow of data. They merely describe the parameters used for each function. In this section we setup the connections between the nodes such that appropriate outputs from nodes are piped into appropriate inputs of other nodes.

Use the nipype.pipeline.engine.Workflow to create a graph-based execution pipeline for first level analysis.

19.2. Defining the workflow 207
19.2.11 Store the output

Create a datasink node to store the contrast images and registration info

```python
datasink = pe.Node(interface=nio.DataSink(), name="datasink")
datasink.inputs.base_directory = os.path.abspath('volsurf_tutorial/l1out')
datasink.inputs.substitutions = []

def getsubs(subject_id):
    subs = [('_subject_id_%s/' % subject_id, '')]
    return subs
```

```python
# store relevant outputs from various stages of the 1st level analysis
levell.connect([[(infosource, datasink, [('subject_id', 'container'), ('subject_id', getsubs), ('substitutions')]),
                ([llpipeline, datasink, ('surfanalysis.contrastestimate.con_images', 'contrasts'),
                  ('preproc.surfregister.out_reg_file', 'registrations'),]])])
```

Run the analysis pipeline and also create a dot+png (if graphviz is available) rskflow.

```python
if __name__ == '__main__':
    levell.run()
    levell.write_graph(graph2use='flat')
```

19.2.12 Level2 surface-based pipeline

Create a level2 workflow

```python
l2flow = pe.Workflow(name='l2out')
l2flow.base_dir = os.path.abspath('volsurf_tutorial')
```

Setup a dummy node to iterate over contrasts and hemispheres

```python
l2inputnode = pe.Node(interface=util.IdentityInterface(fields=['contrasts', 'hemi']), name='inputnode')
l2inputnode.iterables = [('contrasts', list(range(1, len(contrasts) + 1))),
 ('hemi', ['lh', 'rh'])]
```

Use a datagrabber node to collect contrast images and registration files
```python
l2source = pe.Node(
    interface=nio.DataGrabber(infields=['con_id'], outfields=['con', 'reg']),
    name='l2source')
l2source.inputs.base_directory = os.path.abspath('volsurf_tutorial/l1out')
l2source.inputs.template = '*'
l2source.inputs.field_template = dict(
    con='*/contrasts/con_%04d.img', reg='*/registrations/*.dat')
l2source.inputs.template_args = dict(con=[['con_id']], reg=[[]])
l2source.inputs.sort_filelist = True
l2flow.connect(l2inputnode, 'contrasts', l2source, 'con_id')

Merge contrast images and registration files

mergenode = pe.Node(interface=util.Merge(2, axis='hstack'), name='merge')

def ordersubjects(files, subj_list):
    outlist = []
    for s in subj_list:
        for f in files:
            if '/%s/' % s in f:
                outlist.append(f)
                continue
    print(outlist)
    return outlist

l2flow.connect(l2source, ('con', ordersubjects, subject_list), mergenode, 'in1')
l2flow.connect(l2source, ('reg', ordersubjects, subject_list), mergenode, 'in2')

Concatenate contrast images projected to fsaverage

l2concat = pe.Node(interface=fs.MRISPreproc(), name='concat')
l2concat.inputs.target = 'fsaverage'
l2concat.inputs.fwhm = 5

def list2tuple(listoflist):
    return [tuple(x) for x in listoflist]

l2flow.connect(l2inputnode, 'hemi', l2concat, 'hemi')
l2flow.connect(mergenode, ('out', list2tuple), l2concat, 'vol_measure_file')

Perform a one sample t-test

l2ttest = pe.Node(interface=fs.OneSampleTTest(), name='onesample')
l2flow.connect(l2concat, 'out_file', l2ttest, 'in_file')

Run the analysis pipeline and also create a dot+png (if graphviz is available) that visually represents the workflow.

if __name__ == '__main__':
    l2flow.run()
    l2flow.write_graph(graph2use='flat')
```

### Example source code

19.2. Defining the workflow 209
You can download the full source code of this example. This same script is also included in the Nipype source distribution under the examples directory.
A workflow that uses fsl to perform a first level analysis on the nipype tutorial data set:

```
python fmri_fsl.py
```

First tell python where to find the appropriate functions.

```python
from __future__ import print_function
from __future__ import division
from builtins import str
from builtins import range

import os  # system functions

import nipype.interfaces.io as nio  # Data i/o
import nipype.interfaces.fsl as fsl  # fsl
import nipipe.interfaces.utility as util  # utility
import nipipe.pipeline.engine as pe  # pipeline engine
import nipipe.algorithms.modelgen as model  # model generation
import nipipe.algorithms.rapidart as ra  # artifact detection
```

## 20.1 Preliminaries

Setup any package specific configuration. The output file format for FSL routines is being set to compressed NIFTI.

```
fsl.FSLCommand.set_default_output_type('NIFTI_GZ')
```

## 20.2 Setting up workflows

In this tutorial we will be setting up a hierarchical workflow for fsl analysis. This will demonstrate how pre-defined workflows can be setup and shared across users, projects and labs.
20.3 Setup preprocessing workflow

This is a generic fsl feat preprocessing workflow encompassing skull stripping, motion correction and smoothing operations.

```python
preproc = pe.Workflow(name='preproc')
```

Set up a node to define all inputs required for the preprocessing workflow

```python
inputnode = pe.Node(
    interface=util.IdentityInterface(fields=['func', 'struct']),
    name='inputspec')
```

Convert functional images to float representation. Since there can be more than one functional run we use a MapNode to convert each run.

```python
img2float = pe.MapNode(
    interface=fsl.ImageMaths(
        out_data_type='float', op_string='', suffix='_dtype'),
    iterfield=['in_file'],
    name='img2float')
preproc.connect(inputnode, 'func', img2float, 'in_file')
```

Extract the middle volume of the first run as the reference

```python
extract_ref = pe.Node(interface=fsl.ExtractROI(t_size=1), name='extractref')
```

Define a function to pick the first file from a list of files

```python
def pickfirst(files):
    if isinstance(files, list):
        return files[0]
    else:
        return files
```

```python
preproc.connect(img2float, ('out_file', pickfirst), extract_ref, 'in_file')
```

Define a function to return the 1 based index of the middle volume

```python
def getmiddlevolume(func):
    from nibabel import load
    from nipype.utils import NUMPY_MMAP
    funcfile = func
    if isinstance(func, list):
        funcfile = func[0]
    _, _, _, timepoints = load(funcfile, mmap=NUMPY_MMAP).shape
    return int(timepoints / 2) - 1
```

```python
preproc.connect(inputnode, ('func', getmiddlevolume), extract_ref, 't_min')
```

Realign the functional runs to the middle volume of the first run

```python
motion_correct = pe.MapNode(
    interface=fsl.MCFLIRT(save_mats=True, save_plots=True),
    name='realign',
    iterfield=['in_file'])
```

(continues on next page)
preproc.connect(img2float, 'out_file', motion_correct, 'in_file')
preproc.connect(extract_ref, 'roi_file', motion_correct, 'ref_file')

Plot the estimated motion parameters

```python
plot_motion = pe.MapNode(
    interface=fsl.PlotMotionParams(in_source='fsl'),
    name='plot_motion',
    iterfield=['in_file'])
plot_motion.iterables = ('plot_type', ["rotations", "translations"])
preproc.connect(motion_correct, 'par_file', plot_motion, 'in_file')
```

Extract the mean volume of the first functional run

```python
meanfunc = pe.Node(
    interface=fsl.ImageMaths(op_string='-Tmean', suffix='_mean'),
    name='meanfunc')
preproc.connect(motion_correct, ('out_file', pickfirst), meanfunc, 'in_file')
```

Strip the skull from the mean functional to generate a mask

```python
meanfuncmask = pe.Node(
    interface=fsl.BET(mask=True, no_output=True, frac=0.3),
    name='meanfuncmask')
preproc.connect(meanfunc, 'out_file', meanfuncmask, 'in_file')
```

Mask the functional runs with the extracted mask

```python
maskfunc = pe.MapNode(
    interface=fsl.ImageMaths(suffix='_bet', op_string='-mas'),
    iterfield=['in_file'],
    name='maskfunc')
preproc.connect(motion_correct, 'out_file', maskfunc, 'in_file')
pureproc.connect(meanfuncmask, 'mask_file', maskfunc, 'in_file2')
```

Determine the 2nd and 98th percentile intensities of each functional run

```python
getthresh = pe.MapNode(
    interface=fsl.ImageStats(op_string='-p 2 -p 98'),
    iterfield=['in_file'],
    name='getthreshold')
preproc.connect(maskfunc, 'out_file', getthresh, 'in_file')
```

Threshold the first run of the functional data at 10% of the 98th percentile

```python
threshold = pe.Node(
    interface=fsl.ImageMaths(out_data_type='char', suffix='_thresh'),
    name='threshold')
preproc.connect(maskfunc, ('out_file', pickfirst), threshold, 'in_file')
```

Define a function to get 10% of the intensity

```python
def getthreshop(thresh):
    return '-thr %.10f -Tmin -bin' % (0.1 * thresh[0][1])
```

```python
preproc.connect(getthresh, ('out_stat', getthreshop), threshold, 'op_string')
```

Determine the median value of the functional runs using the mask

20.3. Setup preprocessing workflow
medianval = pe.MapNode(
    interface=fsl.ImageStats(op_string='\texttt{-k $s -p 50\%}$'),
    iterfield=['\texttt{in\_file}'],
    name='medianval')

preproc.connect(motion_correct, 'out\_file', medianval, 'in\_file')
preproc.connect(threshold, 'out\_file', medianval, 'mask\_file')

Dilate the mask

dilatemask = pe.Node(
    interface=fsl.ImageMaths(suffix='\_dil', op_string='\texttt{-dilF}'),
    name='dilatemask')

preproc.connect(threshold, 'out\_file', dilatemask, 'in\_file')

Mask the motion corrected functional runs with the dilated mask

maskfunc2 = pe.MapNode(
    interface=fsl.ImageMaths(suffix='\_mask', op_string='\texttt{-mas}'),
    iterfield=['\texttt{in\_file}'],
    name='maskfunc2')

preproc.connect(motion_correct, 'out\_file', maskfunc2, 'in\_file')
preproc.connect(dilatemask, 'out\_file', maskfunc2, 'in\_file2')

Determine the mean image from each functional run

meanfunc2 = pe.MapNode(
    interface=fsl.ImageMaths(op_string='\texttt{-Tmean}'), suffix='\_mean'),
    iterfield=['\texttt{in\_file}'],
    name='meanfunc2')

preproc.connect(maskfunc2, 'out\_file', meanfunc2, 'in\_file')

Merge the median values with the mean functional images into a coupled list

mergenode = pe.Node(interface=util.Merge(2, axis='hstack'), name='merge')

preproc.connect(meanfunc2, 'out\_file', mergenode, 'in1')
preproc.connect(medianval, ('out\_stat', getbtthresh), smooth, 'brightness\_threshold')
preproc.connect(mergenode, ('out', getusans), smooth, 'usans')

Smooth each run using SUSAN with the brightness threshold set to 75% of the median value for each run and a mask constituting the mean functional

smooth = pe.MapNode(
    interface=fsl.SUSAN(),
    iterfield=['\texttt{in\_file}', 'brightness\_threshold', 'usans'],
    name='smooth')

Define a function to get the brightness threshold for SUSAN

def getbtthresh(medianvals):
    return [0.75 * val for val in medianvals]

def getusans(x):
    return [[tuple([val[0], 0.75 * val[1]]) for val in x]

preproc.connect(maskfunc2, 'out\_file', smooth, 'in\_file')
preproc.connect(medianval, ('out\_stat', getbtthresh), smooth, 'brightness\_threshold')
preproc.connect(mergenode, ('out', getusans), smooth, 'usans')

Mask the smoothed data with the dilated mask
Scale each volume of the run so that the median value of the run is set to 10000

Define a function to get the scaling factor for intensity normalization

```python
def getinormscale(medianvals):
    return [-mul %.10f % (10000. / val) for val in medianvals]
```

Perform temporal highpass filtering on the data

```python
highpass = pe.MapNode(
    interface=fsl.ImageMaths(suffix='_tempfilt'),
    iterfield=['in_file'],
    name='highpass')
preproc.connect(intnorm, 'out_file', highpass, 'in_file')
```

Generate a mean functional image from the first run

```python
meanfunc3 = pe.MapNode(
    interface=fsl.ImageMaths(op_string='-Tmean', suffix='_mean'),
    iterfield=['in_file'],
    name='meanfunc3')
preproc.connect(highpass, 'out_file', meanfunc3, 'in_file')
```

Strip the structural image and coregister the mean functional image to the structural image

```python
nosestrip = pe.Node(interface=fsl.BET(frac=0.3), name='nosestrip')
skullstrip = pe.Node(interface=fsl.BET(mask=True), name='stripstruct')
coregister = pe.Node(interface=fsl.FLIRT(dof=6), name='coregister')
```

Use `nipype.algorithms.rapidart` to determine which of the images in the functional series are outliers based on deviations in intensity and/or movement.

```python
art = pe.MapNode(
    interface=ra.ArtifactDetect(
        use_differences=[True, False],
        use_norm=True,
        norm_threshold=1,
        zintensity_threshold=3,
        parameter_source='FSL',
        mask_type='file'),
    iterfield=['realigned_files', 'realignment_parameters'],
    name='art')
```

(continues on next page)
20.4 Set up model fitting workflow

```python
modelfit = pe.Workflow(name='modelfit')

Use nipype.algorithms.modelgen.SpecifyModel to generate design information.

```modelspec = pe.Node(interface=model.SpecifyModel(), name="modelspec")

Use nipype.interfaces.fsl.Level1Design to generate a run specific fsf file for analysis

```level1design = pe.Node(interface=fsl.Level1Design(), name="level1design")

Use nipype.interfaces.fsl.FEATModel to generate a run specific mat file for use by FILMGLS

```modelgen = pe.MapNode(
    interface=fsl.FEATModel(),
    name='modelgen',
    iterfield=['fsf_file', 'ev_files'])

Use nipype.interfaces.fsl.FILMGLS to estimate a model specified by a mat file and a functional run

```modelestimate = pe.MapNode(
    interface=fsl.FILMGLS(smooth_autocorr=True, mask_size=5, threshold=1000),
    name='modelestimate',
    iterfield=['design_file', 'in_file'])

Use nipype.interfaces.fsl.ContrastMgr to generate contrast estimates

```conestimate = pe.MapNode(
    interface=fsl.ContrastMgr(),
    name='conestimate',
    iterfield=[
        'tcon_file', 'param_estimates', 'sigmasquareds', 'corrections',
        'dof_file']
)

modelfit.connect(
    [modelspec, level1design, [('session_info', 'session_info')]),
    (level1design, modelgen, [('fsf_files', 'fsf_file'), ('ev_files',
        'ev_files')]),
    (modelgen, modelestimate, [('design_file', 'design_file')]),
    (modelgen, conestimate, [('design_file', 'design_file')]),
    (modelestimate, conestimate,
        [('param_estimates', 'param_estimates'), ('sigmasquareds',
            'sigmasquareds'),
        ('corrections', 'corrections'), ('dof_file', 'dof_file')])
)
20.5 Set up fixed-effects workflow

```python
fixed_fx = pe.Workflow(name='fixedfx')

Use nipype.interfaces.fsl.Merge to merge the copes and varcopes for each condition

```python
copemerge = pe.MapNode(
    interface=fsl.Merge(dimension='t'),
    iterfield=['in_files'],
    name="copemerge")

varcopemerge = pe.MapNode(
    interface=fsl.Merge(dimension='t'),
    iterfield=['in_files'],
    name="varcopemerge")

Use nipype.interfaces.fsl.L2Model to generate subject and condition specific level 2 model design files

```python
level2model = pe.Node(interface=fsl.L2Model(), name='l2model')

Use nipype.interfaces.fsl.FLAMEO to estimate a second level model

```python
flameo = pe.MapNode(
    interface=fsl.FLAMEO(run_mode='fe'),
    name="flameo",
    iterfield=['cope_file', 'var_cope_file'])

fixed_fx.connect(
    [(copemerge, flameo, [('merged_file', 'cope_file')]),
     (varcopemerge, flameo, [('merged_file', 'var_cope_file')]),
     (level2model, flameo, [('design_mat', 'design_file'),
                            ('design_con', 't_con_file'), ('design_grp',
                             'cov_split_file')])]
)

20.6 Set up first-level workflow

```python

def sort_copes(files):
    numelements = len(files[0])
    outfiles = []
    for i in range(numelements):
        outfiles.insert(i, [])
        for j, elements in enumerate(files):
            outfiles[i].append(elements[i])
    return outfiles

def num_copes(files):
    return len(files)

firstlevel = pe.Workflow(name='firstlevel')
firstlevel.connect(
    [(preproc, modelfit, [('highpass.out_file', 'modelspec.functional_runs'),
                           ('art.outlier_files', 'modelspec.outlier_files')]),
]"

(continues on next page)
20.7 Experiment specific components

The nipype tutorial contains data for two subjects. Subject data is in two subdirectories, s1 and s2. Each subject directory contains four functional volumes: f3.nii, f5.nii, f7.nii, f10.nii. And one anatomical volume named struct.nii.

Below we set some variables to inform the datasource about the layout of our data. We specify the location of the data, the subject sub-directories and a dictionary that maps each run to a mnemonic (or field) for the run type (struct or func). These fields become the output fields of the datasource node in the pipeline.

In the example below, run ‘f3’ is of type ‘func’ and gets mapped to a nifti filename through a template ‘%s.nii’. So ‘f3’ would become ‘f3.nii’.

```python
# Specify the location of the data.
data_dir = os.path.abspath('data')
# Specify the subject directories
subject_list = ['s1', 's2']
# Map field names to individual subject runs.
info = dict(
    func=[('subject_id', ['f3', 'f5', 'f7', 'f10'])],
    struct=[('subject_id', 'struct')])

infosource = pe.Node(
    interface=util.IdentityInterface(fields=['subject_id']), name="infosource")
```

Here we set up iteration over all the subjects. The following line is a particular example of the flexibility of the system. The datasource attribute iterables tells the pipeline engine that it should repeat the analysis on each of the items in the subject_list. In the current example, the entire first level preprocessing and estimation will be repeated for each subject contained in subject_list.

```python
infosource.iterables = ('subject_id', subject_list)
```

Now we create a nipype.interfaces.io.DataSource object and fill in the information from above about the layout of our data. The nipype.pipeline.NodeWrapper module wraps the interface object and provides additional housekeeping and pipeline specific functionality.

```python
datasource = pe.Node(
    interface=nio.DataGrabber(
        infields=['subject_id'], outfields=['func', 'struct']),
    name='datasource')
```

Use the get_node function to retrieve an internal node by name. Then set the iterables on this node to perform two different extents of smoothing.

```python
smoothnode = firstlevel.get_node('preproc.smooth')
```

```python
assert (str(smoothnode) == 'preproc.smooth')
```
smoothnode.iterables = ('fwhm', [5., 10.])
hpcutoff = 120
TR = 3.  # ensure float
firstlevel.inputs.preproc.highpass.suffix = '_hpf'
firstlevel.inputs.preproc.highpass.op_string = '-bptf $d -1' % (hpcutoff / TR)

Setup a function that returns subject-specific information about the experimental paradigm. This is used by the nipype.interfaces.spm.SpecifyModel to create the information necessary to generate an SPM design matrix. In this tutorial, the same paradigm was used for every participant. Other examples of this function are available in the doc/examples folder. Note: Python knowledge required here.

def subjectinfo(subject_id):
    from nipype.interfaces.base import Bunch
    from copy import deepcopy
    print("Subject ID: \$s\n" % str(subject_id))
    output = []
    names = ['Task-Odd', 'Task-Even']
    for r in range(4):
        onsets = [list(range(15, 240, 60)), list(range(45, 240, 60))]
        output.insert(r,
            Bunch(
                conditions=names,
                onsets=deepcopy(onsets),
                durations=[[15] for s in names],
                amplitudes=None,
                tmod=None,
                pmod=None,
                regressor_names=None,
                regressors=None))
    return output

Setup the contrast structure that needs to be evaluated. This is a list of lists. The inner list specifies the contrasts and has the following format - [Name,Stat,[list of condition names],[weights on those conditions]. The condition names must match the names listed in the subjectinfo function described above.

cont1 = ['Task>Baseline', 'T', ['Task-Odd', 'Task-Even'], [0.5, 0.5]]
cont2 = ['Task-Odd>Task-Even', 'T', ['Task-Odd', 'Task-Even'], [1, -1]]
cont3 = ['Task', 'F', [cont1, cont2]]
contrasts = [cont1, cont2]

firstlevel.inputs.modelfit.modelspec.input_units = 'secs'
firstlevel.inputs.modelfit.modelspec.time_repetition = TR
firstlevel.inputs.modelfit.modelspec.high_pass_filter_cutoff = hpcutoff

firstlevel.inputs.modelfit.level1design.interscan_interval = TR
firstlevel.inputs.modelfit.level1design.bases = {'dgamma': {'derivs': False}}
firstlevel.inputs.modelfit.level1design.contrasts = contrasts
firstlevel.inputs.modelfit.level1design.model_serial_correlations = True

20.7.1 Set up complete workflow

llpipeline = pe.Workflow(name="levell")
llpipeline.base_dir = os.path.abspath('./fsl/workingdir')
llpipeline.config = {
    "execution": { (continues on next page)
20.8 Execute the pipeline

The code discussed above sets up all the necessary data structures with appropriate parameters and the connectivity between the processes, but does not generate any output. To actually run the analysis on the data the `nipype.pipeline.engine.Pipeline.Run` function needs to be called.

```python
if __name__ == '__main__':
    l1pipeline.write_graph()
    outgraph = l1pipeline.run()
    # l1pipeline.run(plugin='MultiProc', plugin_args={'n_procs':2})
```

Example source code

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the examples directory.
A pipeline example that data from the FSL FEEDS set. Single subject, two stimuli.
You can find it at http://www.fmrib.ox.ac.uk/fsl/feeds/doc/index.html

```python
from __future__ import division
from builtins import range
import os  # system functions
from nipype.interfaces import io as nio  # Data i/o
from nipype.interfaces import utility as niu  # Utilities
from nipype.interfaces import fsl  # fsl
from nipype.pipeline import engine as pe  # pypeline engine
from nipype.algorithms import modelgen as model  # model generation
from nipype.workflows.fmri.fsl import (create_featreg_preproc, create_modelfit_workflow, create_reg_workflow)
from nipype.interfaces.base import Bunch
```

### 21.1 iminaries

Setup any package specific configuration. The output file format for FSL routines is being set to compressed NIFTI.

```python
fsl.FSLCommand.set_default_output_type('NIFTI_GZ')
```

### 21.2 Experiment specific components

This tutorial does a single subject analysis so we are not using infosource and iterables

```python
# Specify the location of the FEEDS data. You can find it at http://www.fmrib.ox.ac.uk/fsl/feeds/doc/index.html
inputnode = pe.Node(
    niu.IdentityInterface(fields=['in_data']),
    name='inputnode')
# Specify the subject directories
```
# Map field names to individual subject runs.
info = dict(func=['fmri'], struct=['structural'])

Now we create a `nipype.interfaces.io.DataSource` object and fill in the information from above about the layout of our data. The `nipype.pipeline.Node` module wraps the interface object and provides additional housekeeping and pipeline specific functionality.

datasource = pe.Node(
    interface=nio.DataGrabber(outfields=['func', 'struct'], name='datasource'),
    datasource.inputs.template = 'feeds/data/%s.nii.gz',
    datasource.inputs.template_args = info,
    datasource.inputs.sort_filelist = True
)

preproc = create_featreg_preproc(whichvol='first')
TR = 3.
preproc.inputs.inputspec.fwhm = 5
preproc.inputs.inputspec.highpass = 100. / TR

modelspec = pe.Node(interface=model.SpecifyModel(), name='modelspec')
modelspec.inputs.input_units = 'secs'
modelspec.inputs.time_repetition = TR
modelspec.inputs.high_pass_filter_cutoff = 100
modelspec.inputs.subject_info = [
    Bunch(
        conditions=['Visual', 'Auditory'],
        onsets=[
            list(range(0, int(180 * TR), 60)),
            list(range(0, int(180 * TR), 90))
        ],
        durations=[[30], [45]],
        amplitudes=None,
        tmod=None,
        pmod=None,
        regressor_names=None,
        regressors=None
    )
]

modelfit = create_modelfit_workflow(f_contrasts=True)
modelfit.inputs.inputspec.interscan_interval = TR
modelfit.inputs.inputspec.model_serial_correlations = True
modelfit.inputs.inputspec.bases = {'dgamma': {'derivs': True}}
cont1 = ['Visual>Baseline', 'T', ['Visual', 'Auditory'], [1, 0]]
cont2 = ['Auditory>Baseline', 'T', ['Visual', 'Auditory'], [0, 1]]
cont3 = ['Task', 'F', [cont1, cont2]]
modelfit.inputs.inputspec.contrasts = [cont1, cont2, cont3]

registration = create_reg_workflow()
registration.inputs.inputspec.target_image = fsl.Info.standard_image('MNI152_T1_2mm.nii.gz')
registration.inputs.inputspec.target_image_brain = fsl.Info.standard_image('MNI152_T1_2mm_brain.nii.gz')
registration.inputs.inputspec.config_file = 'T1_2_MNI152_2mm'
21.2.1 Set up complete workflow

```python
l1pipeline = pe.Workflow(name="level1")
l1pipeline.base_dir = os.path.abspath('./fsl_feeds/workingdir')
l1pipeline.config = {
    "execution": {
        "crashdump_dir": os.path.abspath('./fsl_feeds/crashdumps')
    }
}
l1pipeline.connect(inputnode, 'in_data', datasource, 'base_directory')
l1pipeline.connect(datasource, 'func', preproc, 'inputspec.func')
l1pipeline.connect(preproc, 'outputspec.highpassed_files', modelspec, 'functional_runs')
l1pipeline.connect(preproc, 'outputspec.motion_parameters', modelspec, 'realignment_parameters')
l1pipeline.connect(modelspec, 'session_info', modelfit, 'inputspec.session_info')
l1pipeline.connect(preproc, 'outputspec.highpassed_files', modelfit, 'inputspec.functional_data')
l1pipeline.connect(preproc, 'outputspec.mean', registration, 'inputspec.mean_image')
l1pipeline.connect(datasource, 'struct', registration, 'inputspec.anatomical_image')
l1pipeline.connect(modelfit, 'outputspec.zfiles', registration, 'inputspec.source_files')
```

Setup the datasink

```python
datasink = pe.Node(
    interface=nio.DataSink(parameterization=False), name="datasink")
datasink.inputs.base_directory = os.path.abspath('./fsl_feeds/l1out')
datasink.inputs.substitutions = [
    ('fmri_dtype_mcf_mask_smooth_mask_gms_mean_warp', 'meanfunc')
]  # store relevant outputs from various stages of the 1st level analysis
l1pipeline.connect(registration, 'outputspec.transformed_files', datasink, 'level1.@Z')
l1pipeline.connect(registration, 'outputspec.transformed_mean', datasink, 'meanfunc')
```

21.3 Execute the pipeline

The code discussed above sets up all the necessary data structures with appropriate parameters and the connectivity between the processes, but does not generate any output. To actually run the analysis on the data the `nipype.pipeline.engine.Pipeline.Run` function needs to be called.

```python
if __name__ == '__main__':
    l1pipeline.inputs.inputnode.in_data = os.path.abspath('feeds/data')
    l1pipeline.run()
```

Example source code
You can download the full source code of this example. This same script is also included in the Nipype source distribution under the `examples` directory.
A workflow that uses fsl to perform a first level analysis on the nipype tutorial data set:

```python
python fmri_fsl_reuse.py
```

First tell python where to find the appropriate functions.

```python
from __future__ import print_function
from __future__ import division
from builtins import str
from builtins import range

import os  # system functions
import nipype.interfaces.io as nio  # Data i/o
import nipype.interfaces.fsl as fsl  # fsl
from nipype.interfaces import utility as niu  # Utilities
import nipype.pipeline.engine as pe  # pypeline engine
import nipype.algorithms.modelgen as model  # model generation
import nipype.algorithms.rapidart as ra  # artifact detection

from nipype.workflows.fmri.fsl import (create_featreg_preproc,
                                      create_modelfit_workflow,
                                      create_fixed_effects_flow)
```

### 22.1 Preliminaries

Setup any package specific configuration. The output file format for FSL routines is being set to compressed NIFTI.

```python
fsl.FSLCommand.set_default_output_type('NIFTI_GZ')
level1_workflow = pe.Workflow(name='level1flow')
preproc = create_featreg_preproc(whichvol='first')
modelfit = create_modelfit_workflow()
```
fixed_fx = create_fixed_effects_flow()

Add artifact detection and model specification nodes between the preprocessing and modelfitting workflows.

```python
art = pe.MapNode(
    ra.ArtifactDetect(
        use_differences=[True, False],
        use_norm=True,
        norm_threshold=1,
        zintensity_threshold=3,
        parameter_source='FSL',
        mask_type='file'),
    iterfield=['realigned_files', 'realignment_parameters', 'mask_file'],
    name="art")
modelspec = pe.Node(model.SpecifyModel(), name="modelspec")

levell_workflow.connect(
    [(preproc, art,
      [('outputspec.motion_parameters', 'realignment_parameters'),
       ('outputspec.realigned_files', 'realigned_files'), ('outputspec.mask', 'mask_file')]),
     (preproc, modelspec, [('outputspec.highpassed_files', 'functional_runs'),
                            ('outputspec.motion_parameters', 'realignment_parameters')]), (art, modelspec,
      [('outlier_files', 'outlier_files')]),
     (modelspec, modelfit, [('session_info', 'inputspec.session_info')]),
     (preproc, modelfit, [('outputspec.highpassed_files', 'inputspec.functional_data')]))
```

### 22.2 Set up first-level workflow

```python
def sort_copes(files):
    numelements = len(files[0])
    outfiles = []
    for i in range(numelements):
        outfiles.insert(i, [])
        for j, elements in enumerate(files):
            outfiles[i].append(elements[i])
    return outfiles

def num_copes(files):
    return len(files)

pickfirst = lambda x: x[0]

levell_workflow.connect(
    [(preproc, fixed_fx, ['outputspec.mask', pickfirst], 'flameo.mask_file')],
    (modelfit, fixed_fx, [('outputspec.copes', sort_copes), 'inputspec.copes'],
                          ('outputspec.dof_file', 'inputspec.dof_files'),
```

(continues on next page)
22.3 Experiment specific components

The nipype tutorial contains data for two subjects. Subject data is in two subdirectories, s1 and s2. Each subject directory contains four functional volumes: f3.nii, f5.nii, f7.nii, f10.nii. And one anatomical volume named struct.nii.

Below we set some variables to inform the datasource about the layout of our data. We specify the location of the data, the subject sub-directories and a dictionary that maps each run to a mnemonic (or field) for the run type (struct or func). These fields become the output fields of the datasource node in the pipeline. In the example below, run ‘f3’ is of type ‘func’ and gets mapped to a nifti filename through a template ‘%s.nii’. So ‘f3’ would become ‘f3.nii’.

```python
inputnode = pe.Node(niu.IdentityInterface(fields=['in_data']), name='inputnode')

# Specify the subject directories
subject_list = ['s1']  # , 's3']
# Map field names to individual subject runs.
info = dict(
    func=[['subject_id', ['f3', 'f5', 'f7', 'f10']]],
    struct=[['subject_id', 'struct']])

infosource = pe.Node(niu.IdentityInterface(fields=['subject_id']), name='infosource')

Here we set up iteration over all the subjects. The following line is a particular example of the flexibility of the system. The datasource attribute iterables tells the pipeline engine that it should repeat the analysis on each of the items in the subject_list. In the current example, the entire first level preprocessing and estimation will be repeated for each subject contained in subject_list.

```python
infosource.iterables = ('subject_id', subject_list)
``` 

Now we create a nipype.interfaces.io.DataSource object and fill in the information from above about the layout of our data. The nipype.pipeline.NodeWrapper module wraps the interface object and provides additional housekeeping and pipeline specific functionality.

```python
datasource = pe.Node(
    nio.DataGrabber(infields=['subject_id'], outfields=['func', 'struct']),
    name='datasource')
datasource.inputs.template = 'nipype-tutorial/data/%s/%s.nii'
datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True
```

Use the get_node function to retrieve an internal node by name. Then set the iterables on this node to perform two different extents of smoothing.

```python
featinput = level1_workflow.get_node('featpreproc.inputspec')
featinput.iterables = ('fwhm', [5., 10.])
```

hpcutoff = 120.
TR = 3.
featinput.inputs.highpass = hpcutoff / (2. * TR)

Setup a function that returns subject-specific information about the experimental paradigm. This is used by
the `nipype.modelgen.SpecifyModel` to create the information necessary to generate an SPM design matrix. In this tutorial, the same paradigm was used for every participant. Other examples of this function are available in the `doc/examples` folder. Note: Python knowledge required here.

```python
def subjectinfo(subject_id):
    from nipype.interfaces.base import Bunch
    from copy import deepcopy
    print("Subject ID: \$s\n" % str(subject_id))
    output = []
    names = ['Task-Odd', 'Task-Even']
    for r in range(4):
        onsets = [list(range(15, 240, 60)), list(range(45, 240, 60))]
        output.insert(r,
            Bunch(
                conditions=names,
                onsets=deepcopy(onsets),
                durations=[[15] for s in names]))
    return output
```

Setup the contrast structure that needs to be evaluated. This is a list of lists. The inner list specifies the contrasts and has the following format - [Name,Stat,[list of condition names],[weights on those conditions]. The condition names must match the names listed in the `subjectinfo` function described above.

```
cont1 = ['Task>Baseline', 'T', ['Task-Odd', 'Task-Even'], [0.5, 0.5]]
cont2 = ['Task-Odd>Task-Even', 'T', ['Task-Odd', 'Task-Even'], [1, -1]]
cont3 = ['Task', 'F', [cont1, cont2]]
contrasts = [cont1, cont2]
```

```
modelspec.inputs.input_units = 'secs'
modelspec.inputs.time_repetition = TR
modelspec.inputs.high_pass_filter_cutoff = hpcutoff
modelfit.inputs.inputspec.interscan_interval = TR
modelfit.inputs.inputspec.bases = {'dgamma': {'derivs': False}}
modelfit.inputs.inputspec.contrasts = contrasts
modelfit.inputs.inputspec.model_serial_correlations = True
modelfit.inputs.inputspec.film_threshold = 1000
```

```
level1_workflow.base_dir = os.path.abspath('./fsl/workingdir')
level1_workflow.config['execution'] = dict(
    crashdump_dir=os.path.abspath('./fsl/crashdumps'))
```

```
level1_workflow.connect(
    (inputnode, datasource, [('in_data', 'base_directory')]),
    (infosource, datasource, [('subject_id', 'subject_id')]),
    (infosource, modelspec, [('subject_id', subjectinfo), ('subject_info')]),
    (datasource, preproc, [('func', 'inputspec.func')]),
)
```

### 22.4 Execute the pipeline

The code discussed above sets up all the necessary data structures with appropriate parameters and the connectivity between the processes, but does not generate any output. To actually run the analysis on the data the `nipype.pipeline.engine.Pipeline.Run` function needs to be called.

```
if __name__ == '__main__':
    # level1_workflow.write_graph()
```
Example source code
You can download the full source code of this example. This same script is also included in the Nipype source distribution under the examples directory.
fMRI: NiPy GLM, SPM

The fmri_nipy_glm.py integrates several interfaces to perform a first level analysis on a two-subject data set. It is very similar to the spm_tutorial with the difference of using nipy for fitting GLM model and estimating contrasts. The tutorial can be found in the examples folder. Run the tutorial from inside the nipype tutorial directory:

```python
from __future__ import print_function
from builtins import str
from builtins import range
from nipype.interfaces.nipy.model import FitGLM, EstimateContrast
from nipype.interfaces.nipy.preprocess import ComputeMask
```

Import necessary modules from nipype.

```python
import nipype.interfaces.io as nio # Data i/o
import nipype.interfaces.spm as spm # spm
import nipype.interfaces.matlab as mlab # how to run matlab
import nipype.interfaces.utility as util # utility
import nipype.pipeline.engine as pe # pipeline engine
import nipype.algorithms.rapidart as ra # artifact detection
import nipype.algorithms.modelgen as model # model specification
import os # system functions
```

### 23.1 Preliminaries

Set any package specific configuration. The output file format for FSL routines is being set to uncompressed NIFTI and a specific version of matlab is being used. The uncompressed format is required because SPM does not handle compressed NIFTI.

```python
# Set the way matlab should be called
mlab.MatlabCommand.set_default_matlab_cmd("matlab -nodesktop -nosplash")
```

The nipype tutorial contains data for two subjects. Subject data is in two subdirectories, s1 and s2. Each subject directory contains four functional volumes: f3.nii, f5.nii, f7.nii, f10.nii. And one anatomical volume.
named struct.nii. Below we set some variables to inform the datasource about the layout of our data. We specify the location of the data, the subject sub-directories and a dictionary that maps each run to a mnemonic (or field) for the run type (struct or func). These fields become the output fields of the datasource node in the pipeline. In the example below, run ‘f3’ is of type ‘func’ and gets mapped to a nifti filename through a template ‘%s.nii’. So ‘f3’ would become

```python
# Specify the location of the data.
data_dir = os.path.abspath('data')
# Specify the subject directories
subject_list = ['s1']
# Map field names to individual subject runs.
info = dict(
    func=[['subject_id', ['f3', 'f5', 'f7', 'f10']]],
    struct=[['subject_id', 'struct']])

# Specify the location of the data.
data_dir = os.path.abspath('data')
subject_list = ['s1']
info = dict(
    func=[['subject_id', ['f3', 'f5', 'f7', 'f10']]],
    struct=[['subject_id', 'struct']])

infosource = pe.Node(
    interface=util.IdentityInterface(fields=['subject_id']), name="infosource")
```

Here we set up iteration over all the subjects. The following line is a particular example of the flexibility of the system. The datasource attribute iterables tells the pipeline engine that it should repeat the analysis on each of the items in the subject_list. In the current example, the entire first level preprocessing and estimation will be repeated for each subject contained in subject_list.

```python
infosource.iterables = ('subject_id', subject_list)
```

### 23.2 Preprocessing pipeline nodes

Now we create a nipype.interfaces.io.DataSource object and fill in the information from above about the layout of our data. The nipype.pipeline.NodeWrapper module wraps the interface object and provides additional housekeeping and pipeline specific functionality.

```python
datasource = pe.Node(
    interface=nio.DataGrabber(
        infields=['subject_id'], outfields=['func', 'struct'],
        name='datasource'))
datasource.inputs.base_directory = data_dir
datasource.inputs.template = '%s/%s.nii'
datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True
```

Use nipype.interfaces.spm.Realign for motion correction and register all images to the mean image.

```python
realign = pe.Node(interface=spm.Realign(), name="realign")
realign.inputs.register_to_mean = True
```

Use nipype.algorithms.rapidart to determine which of the images in the functional series are outliers based on deviations in intensity or movement.

```python
art = pe.Node(interface=ra.ArtifactDetect(), name="art")
art.inputs.use_differences = [True, False]
art.inputs.use_norm = True
art.inputs.norm_threshold = 1
art.inputs.zintensity_threshold = 3
art.inputs.mask_type = 'file'
art.inputs.parameter_source = 'SPM'
```
Use nipype.interfaces.spm.Coregister to perform a rigid body registration of the functional data to the structural data.

```python
coregister = pe.Node(interface=spm.Coregister(), name="coregister")
coregister.inputs.jobtype = 'estimate'
```

Smooth the functional data using nipype.interfaces.spm.Smooth.

```python
smooth = pe.Node(interface=spm.Smooth(), name="smooth")
smooth.inputs.fwhm = 4
```

### 23.3 Set up analysis components

Here we create a function that returns subject-specific information about the experimental paradigm. This is used by the nipype.interfaces.spm.SpecifyModel to create the information necessary to generate an SPM design matrix. In this tutorial, the same paradigm was used for every participant.

```python
def subjectinfo(subject_id):
    from nipype.interfaces.base import Bunch
    from copy import deepcopy
    print("Subject ID: %s\n") % str(subject_id)
    output = []
    names = ['Task-Odd', 'Task-Even']
    for r in range(4):
        onsets = [list(range(15, 240, 60)), list(range(45, 240, 60))]
        output.insert(r,
            Bunch(
                conditions=names,
                onsets=deepcopy(onsets),
                durations=[[15] for s in names],
                amplitudes=None,
                tmod=None,
                pmod=None,
                regressor_names=None,
                regressors=None))
    return output
```

Setup the contrast structure that needs to be evaluated. This is a list of lists. The inner list specifies the contrasts and has the following format - [Name,Stat,[list of condition names],[weights on those conditions]. The condition names must match the names listed in the subjectinfo function described above.

```python
cont1 = ('Task>Baseline', 'T', ['Task-Odd', 'Task-Even'], [0.5, 0.5])
cont2 = ('Task-Odd>Task-Even', 'T', ['Task-Odd', 'Task-Even'], [1, -1])
contrasts = [cont1, cont2]
```

Generate design information using nipype.interfaces.spm.SpecifyModel. nipy accepts only design specified in seconds so “output_units” has always have to be set to “secs”.

```python
modelspec = pe.Node(interface=model.SpecifySPMModel(), name="modelspec")
modelspec.inputs.concatenate_runs = True
modelspec.inputs.input_units = 'secs'
modelspec.inputs.output_units = 'secs'
modelspec.inputs.time_repetition = 3.
modelspec.inputs.high_pass_filter_cutoff = 120
```

Fit the GLM model using nipy and ordinary least square method
model_estimate = pe.Node(interface=FitGLM(), name="model_estimate")
model_estimate.inputs.TR = 3.
model_estimate.inputs.model = "spherical"
model_estimate.inputs.method = "ols"

Estimate the contrasts. The format of the contrasts definition is the same as for FSL and SPM

contrast_estimate = pe.Node(
   interface=EstimateContrast(), name="contrast_estimate")
cont1 = ('Task>Baseline', 'T', ['Task-Odd', 'Task-Even'], [0.5, 0.5])
cont2 = ('Task-Odd>Task-Even', 'T', ['Task-Odd', 'Task-Even'], [1, -1])
contrast_estimate.inputs.contrasts = [cont1, cont2]

23.4 Setup the pipeline

The nodes created above do not describe the flow of data. They merely describe the parameters used for each function. In this section we setup the connections between the nodes such that appropriate outputs from nodes are piped into appropriate inputs of other nodes.

Use the nipype.pipeline.engine.Pipeline to create a graph-based execution pipeline for first level analysis. The config options tells the pipeline engine to use workdir as the disk location to use when running the processes and keeping their outputs. The use_parameterized_dirs tells the engine to create sub-directories under workdir corresponding to the iterables in the pipeline. Thus for this pipeline there will be subject specific sub-directories.

The nipype.pipeline.engine.Pipeline.connect function creates the links between the processes, i.e., how data should flow in and out of the processing nodes.

l1pipeline = pe.Workflow(name="level1")
l1pipeline.base_dir = os.path.abspath('nipy_tutorial/workingdir')
l1pipeline.connect(
   [(infosource, datasource, [('subject_id', 'subject_id')]),
    (datasource, realign, [('func', 'in_files')]),
    (realign, compute_mask, [('mean_image', 'mean_volume')]),
    (realign, coregister, [('mean_image', 'source'),
                            ('realigned_files', 'apply_to_files')]),
    (datasource, coregister, [('struct', 'target')]),
    (coregister, smooth, [('realignment_parameters', 'realignment_parameters')]),
    (smooth, modelspec, [('smoothed_files', 'functional_runs')]),
    (art, realign, [('realignment_parameters', 'realignment_parameters')]),
    (coregister, art, [('coregistered_files', 'realigned_files')]),
    (compute_mask, art, [('brain_mask', 'mask_file')]),
    (art, modelspec, [('outlier_files', 'outlier_files')]),
    (infosource, modelspec, [('subject_id', subjectinfo), 'subject_info'])],
   (modelspec, model_estimate, [('session_info', 'session_info')]),
   (compute_mask, model_estimate, [('brain_mask', 'mask')]),
   (model_estimate, contrast_estimate, [('beta', 'beta'), ('nvbeta', 'nvbeta'),
                                        ('s2', 's2'), ('dof', 'dof'),
                                        ('axis', 'axis'), ('constants', 'constants'), ('reg_names', 'reg_names')])

(continues on next page)
```python
if __name__ == '__main__':
    llpipeline.run()
```

---

**Example source code**

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the `examples` directory.

---

**Example source code**

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the `examples` directory.
CHAPTER 24

fMRI: Coregistration - Slicer, BRAINS

This is currently not working and will raise an exception in release 0.3. It will be fixed in a later release:

```
python fmri_slicer_coregistration.py
```

```
# raise RuntimeWarning, 'Slicer not fully implemented'
from nipype.interfaces.slicer import BRAINSFit, BRAINSResample
```

Import necessary modules from nipype.

```
import nipype.interfaces.io as nio  # Data i/o
import nipype.interfaces.utility as util  # utility
import nipype.pipeline.engine as pe  # pipeline engine
import os  # system functions
```

24.1 Preliminaries

Confirm package dependencies are installed. (This is only for the tutorial, rarely would you put this in your own code.)

```
from nipype.utils.misc import package_check
package_check('numpy', '1.3', 'tutorial1')
package_check('scipy', '0.7', 'tutorial1')
package_check('IPython', '0.10', 'tutorial1')
```

The nipype tutorial contains data for two subjects. Subject data is in two subdirectories, s1 and s2. Each subject directory contains four functional volumes: f3.nii, f5.nii, f7.nii, f10.nii. And one anatomical volume named struct.nii.

Below we set some variables to inform the datasource about the layout of our data. We specify the location of the data, the subject sub-directories and a dictionary that maps each run to a mnemonic (or field) for the run type (struct or func). These fields become the output fields of the datasource node in the pipeline.

In the example below, run ‘f3’ is of type ‘func’ and gets mapped to a nifti filename through a template ‘%s.nii’. So ‘f3’ would become ‘f3.nii’.

```
# Specify the location of the data.
data_dir = os.path.abspath('data')
```
# Specify the subject directories
subject_list = ['s1', 's3']

# Map field names to individual subject runs.
info = dict(func=[['subject_id', 'f3']], struct=[['subject_id', 'struct']])

infosource = pe.Node(interface=util.IdentityInterface(fields=['subject_id']), name='infosource')

Here we set up iteration over all the subjects. The following line is a particular example of the flexibility of the system. The datasource attribute iterables tells the pipeline engine that it should repeat the analysis on each of the items in the subject_list. In the current example, the entire first level preprocessing and estimation will be repeated for each subject contained in subject_list.

```python
infosource.iterables = ('subject_id', subject_list)
```

## 24.2 Preprocessing pipeline nodes

Now we create a nipype.interfaces.io.DataSource object and fill in the information from above about the layout of our data. The nipype.pipeline.NodeWrapper module wraps the interface object and provides additional housekeeping and pipeline specific functionality.

```python
datasource = pe.Node(interface=nio.DataGrabber(infields=['subject_id'], outfields=['func', 'struct'], name='datasource'))
datasource.inputs.base_directory = data_dir
datasource.inputs.template = '%s/%s.nii'
datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True
coregister = pe.Node(interface=BRAINSFit(), name='coregister')
coregister.inputs.outputTransform = True
coregister.inputs.outputVolume = True
coregister.inputs.transformType = ['Affine']
reslice = pe.Node(interface=BRAINSResample(), name='reslice')
reslice.inputs.outputVolume = True
pipeline = pe.Workflow(name='pipeline')
pipeline.base_dir = os.path.abspath('slicer_tutorial/workingdir')
pipeline.connect([(infosource, datasource, [('subject_id', 'subject_id')]),
                 (datasource, coregister, [('func', 'movingVolume')]),
                 (datasource, coregister, [('struct', 'fixedVolume')]),
                 (coregister, reslice, [('outputTransform', 'warpTransform')]),
                 (datasource, reslice, [('func', 'inputVolume')]),
                 (datasource, reslice, [('struct', 'referenceVolume')])])
```

```python
if __name__ == '__main__':
    pipeline.run()
    pipeline.write_graph()
```

### Example source code

You can download the full source code of this example. This same script is also included in
the Nipype source distribution under the `examples` directory.
CHAPTER 25

fMRI: SPM, FSL

The fmri_spm.py integrates several interfaces to perform a first and second level analysis on a two-subject data set. The tutorial can be found in the examples folder. Run the tutorial from inside the nipype tutorial directory:

```
python fmri_spm.py
```

Import necessary modules from nipype.

```python
from __future__ import print_function
from builtins import str
from builtins import range
import os  # system functions
from nipype import config
# config.enable_provenance()
from nipype.interfaces import spm, fsl
import nipype.interfaces.io as nio  # Data i/o
import nipype.interfaces.utility as util  # utility
import nipype.pipeline.engine as pe  # pipeline engine
import nipype.algorithms.rapidart as ra  # artifact detection
import nipype.algorithms.modelgen as model  # model specification
import nipype.interfaces.matlab as mlab
```

### 25.1 Preliminaries

Set any package specific configuration. The output file format for FSL routines is being set to uncompressed NIFTI and a specific version of matlab is being used. The uncompressed format is required because SPM does not handle compressed NIFTI.
# Tell fsl to generate all output in uncompressed nifti format
fsl.FSLCommand.set_default_output_type('NIFTI')

# Set the way matlab should be called
# import nipype.interfaces.matlab as mlab  # how to run matlab
# mlab.MatlabCommand.set_default_matlab_cmd("matlab -nodesktop -nosplash")

# In case a different path is required
# mlab.MatlabCommand.set_default_paths('/software/matlab/spm12b/spm12b_r5918')

The nipype tutorial contains data for two subjects. Subject data is in two subdirectories, s1 and s2. Each subject directory contains four functional volumes: f3.nii, f5.nii, f7.nii, f10.nii. And one anatomical volume named struct.nii. Below we set some variables to inform the datasource about the layout of our data. We specify the location of the data, the subject sub-directories and a dictionary that maps each run to a mnemonic (or field) for the run type (struct or func). These fields become the output fields of the datasource node in the pipeline. In the example below, run ‘f3’ is of type ‘func’ and gets mapped to a nifti filename through a template ‘%s.nii’. So ‘f3’ would become

# Specify the location of the data.
data_dir = os.path.abspath('data')
# Specify the subject directories
subject_list = ['s1', 's3']
# Map field names to individual subject runs.
info = dict(
    func=[['subject_id', ['f3', 'f5', 'f7', 'f10']]],
    struct=[['subject_id', 'struct']])

infosource = pe.Node(interface=util.IdentityInterface(fields=['subject_id']), name="infosource")

Here we set up iteration over all the subjects. The following line is a particular example of the flexibility of the system. The datasource attribute iterables tells the pipeline engine that it should repeat the analysis on each of the items in the subject_list. In the current example, the entire first level preprocessing and estimation will be repeated for each subject contained in subject_list.

infosource.iterables = ('subject_id', subject_list)

## 25.2 Preprocessing pipeline nodes

Now we create a nipype.interfaces.io.DataSource object and fill in the information from above about the layout of our data. The nipype.pipeline.NodeWrapper module wraps the interface object and provides additional housekeeping and pipeline specific functionality.

datasource = pe.Node(
    interface=nio.DataGrabber(
        infields=['subject_id'], outfields=['func', 'struct'],
        name='datasource')

datasource.inputs.base_directory = data_dir
datasource.inputs.template = '%s/%s.nii'
datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True

Use nipype.interfaces.spm.Realign for motion correction and register all images to the mean image.

realign = pe.Node(interface=spm.Realign(), name="realign")
realign.inputs.register_to_mean = True
Use `nipype.algorithms.rapidart` to determine which of the images in the functional series are outliers based on deviations in intensity or movement.

```
art = pe.Node(interface=ra.ArtifactDetect(), name="art")
art.inputs.use_differences = [True, False]
art.inputs.use_norm = True
art.inputs.norm_threshold = 1
art.inputs.zintensity_threshold = 3
art.inputs.mask_type = 'file'
art.inputs.parameter_source = 'SPM'
```

Skull strip structural images using `nipype.interfaces.fsl.BET`.

```
skullstrip = pe.Node(interface=fsl.BET(), name="skullstrip")
skullstrip.inputs.mask = True
```

Use `nipype.interfaces.spm.Coregister` to perform a rigid body registration of the functional data to the structural data.

```
coregister = pe.Node(interface=spm.Coregister(), name="coregister")
coregister.inputs.jobtype = 'estimate'
```

Warp functional and structural data to SPM's T1 template using `nipype.interfaces.spm.Normalize`. The tutorial data set includes the template image, T1.nii.

```
normalize = pe.Node(interface=spm.Normalize(), name="normalize")
normalize.inputs.template = os.path.abspath('data/T1.nii')
```

Smooth the functional data using `nipype.interfaces.spm.Smooth`.

```
smooth = pe.Node(interface=spm.Smooth(), name="smooth")
fwhmlist = [4]
smooth.iterables = ('fwhm', fwhmlist)
```

### 25.3 Set up analysis components

Here we create a function that returns subject-specific information about the experimental paradigm. This is used by `nipype.interfaces.spm.SpecifyModel` to create the information necessary to generate an SPM design matrix. In this tutorial, the same paradigm was used for every participant.

```
def subjectinfo(subject_id):
    from nipype.interfaces.base import Bunch
    from copy import deepcopy
    print('Subject ID: %s

output = []
names = ['Task-Odd', 'Task-Even']
for r in range(4):
    onsets = [[list(range(15, 240, 60)), list(range(45, 240, 60))]
    output.insert(r,
    Bunch(
        conditions=names,
        onsets=deepcopy(onsets),
        durations=[[15] for s in names]))
return output
```

Setup the contrast structure that needs to be evaluated. This is a list of lists. The inner list specifies the contrasts and has the following format - [Name,Stat,[list of condition names],[weights on those conditions]. The condition names must match the `names` listed in the `subjectinfo` function described above.
cont1 = ('Task>Baseline', 'T', ['Task-Odd', 'Task-Even'], [0.5, 0.5])
cont2 = ('Task-Odd>Task-Even', 'T', ['Task-Odd', 'Task-Even'], [1, -1])
contrasts = [cont1, cont2]

Generate SPM-specific design information using `nipype.interfaces.spm.SpecifyModel`.

```python
modelspec = pe.Node(interface=model.SpecifySPMModel(), name="modelspec")
modelspec.inputs.concatenate_runs = False
modelspec.inputs.input_units = 'secs'
modelspec.inputs.output_units = 'secs'
modelspec.inputs.time_repetition = 3.
modelspec.inputs.high_pass_filter_cutoff = 120
```

Generate a first level SPM.mat file for analysis using `nipype.interfaces.spm.Level1Design`.

```python
level1design = pe.Node(interface=spm.Level1Design(), name="level1design")
level1design.inputs.timing_units = modelspec.inputs.output_units
level1design.inputs.interscan_interval = modelspec.inputs.time_repetition
level1design.inputs.bases = {'hrf': {'derivs': [0, 0]}}
```

Use `nipype.interfaces.spm.EstimateModel` to determine the parameters of the model.

```python
level1estimate = pe.Node(interface=spm.EstimateModel(), name="level1estimate")
level1estimate.inputs.estimation_method = {'Classical': 1}
```

Use `nipype.interfaces.spm.EstimateContrast` to estimate the first level contrasts specified in a few steps above.

```python
contrastestimate = pe.Node(interface=spm.EstimateContrast(), name="contrastestimate")
contrastestimate.inputs.contrasts = contrasts
contrastestimate.overwrite = True
contrastestimate.config = {'execution': {'remove_unnecessary_outputs': False}}
```

### 25.4 Setup the pipeline

The nodes created above do not describe the flow of data. They merely describe the parameters used for each function. In this section we setup the connections between the nodes such that appropriate outputs from nodes are piped into appropriate inputs of other nodes.

Use `nipype.pipeline.engine.Pipeline` to create a graph-based execution pipeline for first level analysis. The config options tells the pipeline engine to use `workdir` as the disk location to use when running the processes and keeping their outputs. The `use_parameterized_dirs` tells the engine to create sub-directories under `workdir` corresponding to the iterables in the pipeline. Thus for this pipeline there will be subject specific sub-directories.

The `nipype.pipeline.engine.Pipeline.connect` function creates the links between the processes, i.e., how data should flow in and out of the processing nodes.

```python
l1pipeline = pe.Workflow(name="level1")
l1pipeline.base_dir = os.path.abspath('spm_tutorial/workingdir')
```

(continues on next page)
25.5 Setup storage results

Use nipype.interfaces.io.DataSink to store selected outputs from the pipeline in a specific location. This allows the user to selectively choose important output bits from the analysis and keep them. The first step is to create a datasink node and then to connect outputs from the modules above to storage locations. These take the following form directory_name[@]subdir where parts between [] are optional. For example 'realign.@mean' below creates a directory called realign in 'l1output/subject_id/' and stores the mean image output from the Realign process in the realign directory. If the @ is left out, then a sub-directory with the name 'mean' would be created and the mean image would be copied to that directory.

```python
datasink = pe.Node(interface=nio.DataSink(), name="datasink")
datasink.inputs.base_directory = os.path.abspath('spm_tutorial/l1output')

def getstripdir(subject_id):
    import os
    return os.path.join(os.path.abspath('spm_tutorial/workingdir'), '_subject_id_%s' % subject_id)

# store relevant outputs from various stages of the 1st level analysis
l1pipeline.connect([
    (infosource, datasink, [('subject_id', 'container')],
     ('subject_id', getstripdir), 'strip_dir')],
    (realign, datasink, [('mean_image', 'realign.@mean'),
        ('realignment_parameters', 'realign.@param')],
    (art, datasink, [('outlier_files', 'art.@outliers'), ('statistic_files',
        'art.@stats')],
    (level1design, datasink, [('spm_mat_file', 'model.pre-estimate')])),
    (levelestim, contrastestimate, [('spm_mat_file', 'model.@spm'), ('beta_images', 'model.@beta'), ('mask_image', 'model.@mask'), ('residual_image', 'model.@res'), ('RPVimage', 'model.@rpv')]),
    (contrastestimate, datasink, [('con_images', 'contrasts.@con'), ('spmT_images', 'contrasts.@T')])]
```

(continues on next page)
Use `nipype.interfaces.io.DataGrabber` to extract the contrast images across a group of first level subjects. Unlike the previous pipeline that iterated over subjects, this pipeline will iterate over all contrast images:

```python
# collect all the con images for each contrast.
contrast_ids = list(range(1, len(contrasts) + 1))
l2source = pe.Node(nio.DataGrabber(infields=['fwhm', 'con']), name="l2source")
# we use .*i* to capture both .img (SPM8) and .nii (SPM12)
l2source.inputs.template = os.path.abspath('spm_tutorial/l1output/*/con*/*_fwhm_\%d/con_\%04d.*i*')
# iterate over all contrast images
l2source.iterables = [('fwhm', fwhmlist), ('con', contrast_ids)]
l2source.inputs.sort_filelist = True
```

Use `nipype.interfaces.spm.OneSampleTTestDesign` to perform a simple statistical analysis of the contrasts from the group of n:

```python
# setup a 1-sample t-test node
onesamptstestdes = pe.Node(
    interface=spm.OneSampleTTestDesign(), name="onesamptstestdes")
l2estimate = pe.Node(interface=spm.EstimateModel(), name="level2estimate")
l2estimate.inputs.estimation_method = {'Classical': 1}
l2conestimate = pe.Node(interface=spm.EstimateContrast(), name="level2conestimate")
cont1 = [('Group', 'T', ['mean'], [1])]
l2conestimate.inputs.contrasts = [cont1]
l2conestimate.inputs.group_contrast = True
```

As before, we setup a pipeline to connect these two nodes (l2source lettest):

```python
l2pipeline = pe.Workflow(name="level2")
l2pipeline.base_dir = os.path.abspath('spm_tutorial/l2output')
l2pipeline.connect(
    (l2source, onesamptstestdes, [('outfiles', 'in_files')]),
    (onesamptstestdes, l2estimate, [('spm_mat_file', 'spm_mat_file')]),
    (l2estimate, l2conestimate, [('spm_mat_file', 'spm_mat_file'), ('beta_images', 'beta_images'),
    ('residual_image', 'residual_image')]),
)
```

### 25.6 Execute the pipeline

The code discussed above sets up all the necessary data structures with appropriate parameters and the connectivity between the processes, but does not generate any output. To actually run the analysis on the data the `nipype.pipeline.engine.Pipeline.Run` function needs to be called.

```python
if __name__ == '__main__':
    l1pipeline.run('MultiProc')
    l2pipeline.run('MultiProc')
```

#### Example source code

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the examples directory.
26.1 Introduction

The fmri_spm_auditory.py recreates the classical workflow described in the SPM8 manual using auditory dataset that can be downloaded from http://www.fil.ion.ucl.ac.uk/spm/data/auditory/:

```
python fmri_spm_auditory.py
```

Import necessary modules from nipype.

```
from builtins import range
import nipype.interfaces.io as nio # Data i/o
import nipype.interfaces.spm as spm # spm
import nipype.interfaces.fsl as fsl # fsl
import nipype.interfaces.matlab as mlab # how to run matlab
import nipype.interfaces.utility as util # utility
import nipype.pipeline.engine as pe # pipeline engine
import nipype.algorithms.modelgen as model # model specification
import os # system functions
```

26.1.1 Preliminaries

```
# Set the way matlab should be called
mlab(MatlabCommand.set_default_matlab_cmd("matlab -nodesktop -nosplash")
```

26.1.2 lows

In this tutorial we will be setting up a hierarchical workflow for spm analysis. This will demonstrate how pre-defined workflows can be setup and shared across users, projects and labs. eprocessing workflow ———— This is a generic preprocessing workflow that can be used by different analyses

```
preproc = pe.Workflow(name='preproc')
```

We strongly encourage to use 4D files insteead of series of 3D for fMRI analyses for many reasons (cleanness and saving and filesystem inodes are among them). However, the the workflow presented in the SPM8 manual which this tutorial is based on uses 3D files. Therefore we leave converting to 4D as an option. We are using
merge_to_4d variable, because switching between 3d and 4d requires some additional steps (explained later on). Use `nipype.interfaces.fsl.Merge` to merge a series of 3D files along the time dimension creating a 4d file.

```python
merge_to_4d = True
if merge_to_4d:
    merge = pe.Node(interface=fsl.Merge(), name="merge")
    merge.inputs.dimension = "t"
```

Use `nipype.interfaces.spm.Realign` for motion correction and register all images to the mean image.

```python
realign = pe.Node(interface=spm.Realign(), name="realign")
```

Use `nipype.interfaces.spm.Coregister` to perform a rigid body registration of the functional data to the structural data.

```python
coregister = pe.Node(interface=spm.Coregister(), name="coregister")
coregister.inputs.jobtype = 'estimate'
segment = pe.Node(interface=spm.Segment(), name="segment")
```

Uncomment the following line for faster execution

```python
# segment.inputs.gaussians_per_class = [1, 1, 1, 4]
```

Warp functional and structural data to SPM's T1 template using `nipype.interfaces.spm.Normalize`. The tutorial data set emplate image, T1.nii.

```python
normalize_func = pe.Node(interface=spm.Normalize(), name="normalize_func")
normalize_func.inputs.jobtype = "write"
normalize_struc = pe.Node(interface=spm.Normalize(), name="normalize_struc")
normalize_struc.inputs.jobtype = "write"
```

Smooth the functional data using `nipype.interfaces.spm.Smooth`.

```python
smooth = pe.Node(interface=spm.Smooth(), name="smooth")
```

`write_voxel_sizes` is the input of the normalize interface that is recommended to be set to the voxel sizes of the target volume. There is no need to set it manually since we can infer it from data using the following function:

```python
def get_vox_dims(volume):
    import nibabel as nb
    from nipype.utils import NUMPY_MMAP
    if isinstance(volume, list):
        volume = volume[0]
    nii = nb.load(volume, mmap=NUMPY_MMAP)
    hdr = nii.header
    voxdims = hdr.get_zooms()
    return [float(voxdims[0]), float(voxdims[1]), float(voxdims[2])]
```

Here we are connecting all the nodes together. Notice that we add the merge node only if you choose to use 4D. Also `get_vox_dims` function is passed along the input volume of normalise to set the optimal voxel sizes.

```python
if merge_to_4d:
    preproc.connect([(merge, realign, [('merged_file', 'in_files')])])
preproc.connect([(realign, coregister, [('mean_image', 'target')]),
                 (coregister, segment, [('coregistered_source', 'data')]),
```

(continues on next page)
26.1.3 Set up analysis workflow

```python
llanalisis = pe.Workflow(name='analysis')
Generate SPM-specific design information using nipype.interfaces.spm.SpecifyModel.
modelspec = pe.Node(interface=model.SpecifySPMModel(), name=“modelspec”)
Generate a first level SPM.mat file for analysis nipype.interfaces.spm.Level1Design.
level1design = pe.Node(interface=spm.Level1Design(), name=“level1design”)
level1design.inputs.bases = {‘hrf’: {‘derivs’: [0, 0]}}
Use nipype.interfaces.spm.EstimateModel to determine the parameters of the model.
level1estimate = pe.Node(interface=spm.EstimateModel(), name=“level1estimate”)
level1estimate.inputs.estimation_method = {‘Classical’: 1}
threshold = pe.Node(interface=spm.Threshold(), name=“threshold”)
Use nipype.interfaces.spm.EstimateContrast to estimate the first level contrasts specified in a few steps above.
contrastestimate = pe.Node(interface=spm.EstimateContrast(), name=“contrastestimate”)
llanalisis.connect([(modelspec, level1design, [‘session_info’, ‘session_info’])],
[level1design, level1estimate, [‘spm_mat_file’, ‘spm_mat_file’]),
level1estimate, contrastestimate,
[‘spm_mat_file’, ‘spm_mat_file’], [‘beta_images’, ‘beta_images’],
[‘residual_image’, ‘residual_image’]),
contrastestimate, threshold, [‘spm_mat_file’, ‘spm_mat_file’],
[‘spamT_images’, ‘stat_image’]),
])
```

26.1.4 Preproc + Analysis pipeline

```python
llpipeline = pe.Workflow(name=‘firstlevel’)
llpipeline.connect([(preproc, llanalisis, [‘realign.realignment_parameters’,
modelspec.realignment_parameters’)])])
Plugging in functional_runs is a bit more complicated, because model spec expects a list of runs. Every run can be a 4D file or a list of 3D files. Therefore for 3D analysis we need a list of lists and to make one we need a helper function.
if merge_to_4d:
    l1pipeline.connect(((preproc, llanalysis,
                      [('smooth.smoothed_files',
                        'modelspec.functional_runs')]))
else:
    def makelist(item):
        return [item]
    l1pipeline.connect(((preproc, llanalysis,
                      [('smooth.smoothed_files', makelist),
                        'modelspec.functional_runs')]))

26.1.5 Data specific components

In this tutorial there is only one subject M00223. Below we set some variables to inform the datasource about the layout of our data. We specify the location of the data, the subject sub-directories and a dictionary that maps each run to a mnemonic (or field) for the run type (struct or func). These fields become the output fields of the datasource node in the pipeline.

```python
# Specify the location of the data downloaded from http://www.fil.ion.ucl.ac.uk/
# spm/data/auditory/
data_dir = os.path.abspath('spm_auditory_data')
# Specify the subject directories
subject_list = ['M00223']
# Map field names to individual subject runs.
info = dict(
    func=[['f', 'subject_id', 'f', 'subject_id',
           list(range(16, 100))]],
    struct=[['s', 'subject_id', 's', 'subject_id', 2]])
infosource = pe.Node(
    interface=util.IdentityInterface(fields=['subject_id']), name="infosource")

Here we set up iteration over all the subjects. The following line is a particular example of the flexibility of the system. The datasource attribute iterables tells the pipeline engine that it should repeat the analysis on each of the items in the subject_list. In the current example, the entire first level preprocessing and estimation will be repeated for each subject contained in subject_list.

```python
infosource.iterables = ('subject_id', subject_list)
```  
Now we create a nipype.interfaces.io.DataGrabber object and fill in the information from above about the layout of our data. The nipype.pipeline.NodeWrapper module wraps the interface object and provides additional housekeeping and pipeline specific functionality.

```python
datasource = pe.Node(
    interface=nio.DataGrabber(
        infields=['subject_id'], outfields=['func', 'struct'],
        name='datasource')
datasource.inputs.base_directory = data_dir
datasource.inputs.template = '%s%s_%03d.img'
datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True
```
26.1.6 Experimental paradigm specific components

Here we create a structure that provides information about the experimental paradigm. This is used by the `nipype.interfaces.spm.SpecifyModel` to create the information necessary to generate an SPM design matrix.

```python
from nipype.interfaces.base import Bunch
subjectinfo = [
    Bunch(
        conditions=['Task'], onsets=[list(range(6, 84, 12))], durations=[[6]])
]
```

Setup the contrast structure that needs to be evaluated. This is a list of lists. The inner list specifies the contrasts and has the following format - [Name,Stat,[list of condition names],[weights on those conditions]. The condition names must match the names listed in the `subjectinfo` function described above.

```python
cont1 = ('active > rest', 'T', ['Task'], [1])
contrasts = [cont1]
```

26.1.7 Setup the pipeline

The nodes created above do not describe the flow of data. They merely describe the parameters used for each function. In this section we setup the connections between the nodes such that appropriate outputs from nodes are piped into appropriate inputs of other nodes. Use the `nipype.pipeline.engine.Pipeline` to create a graph-based execution pipeline for first level analysis. The config options tells the pipeline engine to use `workdir` as the disk location to use when running the processes and keeping their outputs. The `use_parameterized_dirs` tells the engine to create sub-directories under `workdir` corresponding to the iterables in the pipeline. Thus for this pipeline there will be subject specific sub-directories.

The `nipype.pipeline.engine.Pipeline.connect` function creates the links between the processes, i.e., how data should flow in and out of the processing nodes.

```python
level1 = pe.Workflow(name="level1")
level1.base_dir = os.path.abspath('spm_auditory_tutorial/workingdir')
level1.connect([(infosource, datasource, [('subject_id', 'subject_id')]),
                (datasource, l1pipeline, [('struct', 'preproc.coregister.source')])])
if merge_to_4d:
    level1.connect([(datasource, l1pipeline, [('func', 'preproc.merge.in_files')])])
```

(continues on next page)
26.1.8 Setup storage results

Use nipype.interfaces.io.DataSink to store selected outputs from the pipeline in a specific location. This allows the user to selectively choose important output bits from the analysis and keep them.

The first step is to create a datasink node and then to connect outputs from the modules above to storage locations. These take the following form directory_name[@@subdir] where parts between [ ] are optional. For example ‘realign.@mean’ below creates a directory called realign in ‘l1output/subject_id/’ and stores the mean image output from the Realign process in the realign directory. If the @ is left out, then a sub-directory with the name ‘mean’ would be created and the mean image would be copied to that directory.

```python
datasink = pe.Node(interface=nio.DataSink(), name='datasink')
datasink.inputs.base_directory = os.path.abspath('spm_auditory_tutorial/l1output')

def getstripdir(subject_id):
    import os
    return os.path.join(
        os.path.abspath('spm_auditory_tutorial/workingdir'),
        '_subject_id_%s' % subject_id)

# store relevant outputs from various stages of the 1st level analysis
level1.connect([
    (infosource, datasink, [('subject_id', 'container'),
        ('subject_id', getstripdir), 'strip_dir')],
    (l1pipeline, datasink,
        [('analysis.contrastestimate.con_images', 'contrasts.@con'),
        ('analysis.contrastestimate.spmT_images', 'contrasts.@T')])),
])
```

Code discussed above sets up all the necessary data structures appropriate parameters and the connectivity between the sses, but does not generate any output. To actually run the sis on the data the nipype.pipeline.engine.Pipeline.Run

```python
if __name__ == '__main__':
    level1.run()
    level1.write_graph()
```

Example source code

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the examples directory.
The fmri_spm_dartel.py integrates several interfaces to perform a first and second level analysis on a two-
subject data set. The tutorial can be found in the examples folder. Run the tutorial from inside the nipype
tutorial directory:

```python
python fmri_spm_dartel.py
```

Import necessary modules from nipype.

```python
from __future__ import print_function
from builtins import str
from builtins import range

import nipype.interfaces.io as nio  # Data i/o
import nipype.interfaces.spm as spm  # spm
import nipype.workflows.fmri.spm as spm_wf  # spm
import nipype.interfaces.fsl as fsl  # fsl
from nipype.interfaces import utility as niu  # Utilities
import nipype.pipeline.engine as pe  # pipeline engine
import nipype.algorithms.rapidart as ra  # artifact detection
import nipype.algorithms.modelgen as model  # model specification
import os  # system functions
```

## 27.1 Preliminaries

Set any package specific configuration. The output file format for FSL routines is being set to uncompressed
NIFTI and a specific version of matlab is being used. The uncompressed format is required because SPM does
not handle compressed NIFTI.

```bash
# Tell fsl to generate all output in uncompressed nifti format
fsl.FSLCommand.set_default_output_type('NIFTI')

# Set the way matlab should be called
# mlab.MatlabCommand.set_default_matlab_cmd("matlab -nodesktop -nosplash")
# mlab.MatlabCommand.set_default_paths('/software/spm8')
```
27.2. lows

In this tutorial we will be setting up a hierarchical workflow for SPM analysis. This will demonstrate how pre-defined workflows can be setup and shared across users, projects and labs. 

```python
preproc = pe.Workflow(name='preproc')
```

Use `nipype.interfaces.spm.Realign` for motion correction and register all images to the mean image.

```python
realign = pe.Node(spm.Realign(), name="realign")
realign.inputs.register_to_mean = True
```

Use `nipype.algorithms.rapidart` to determine which of the images in the functional series are outliers based on deviations in intensity or movement.

```python
art = pe.Node(ra.ArtifactDetect(), name="art")
art.inputs.use_differences = [True, False]
art.inputs.use_norm = True
art.inputs.norm_threshold = 1
art.inputs.zintensity_threshold = 3
art.inputs.mask_type = 'file'
art.inputs.parameter_source = 'SPM'
```

Skull strip structural images using `nipype.interfaces.fsl.BET`.

```python
skullstrip = pe.Node(fsl.BET(), name="skullstrip")
skullstrip.inputs.mask = True
```

Use `nipype.interfaces.spm.Coregister` to perform a rigid body registration of the functional data to the structural data.

```python
coregister = pe.Node(spm.Coregister(), name="coregister")
coregister.inputs.jobtype = 'estimate'
```

Normalize and smooth functional data using DARTEL template

```python
normalize_and_smooth_func = pe.Node(
    spm.DARTELNorm2MNI(modulate=True), name='normalize_and_smooth_func')
fwhmlist = [4]
normalize_and_smooth_func.iterables = ('fwhm', fwhmlist)
```

Normalize structural data using DARTEL template

```python
normalize_struct = pe.Node(
    spm.DARTELNorm2MNI(modulate=True), name='normalize_struct')
```

```python
preproc.connect([(realign, coregister, [('mean_image', 'source'), ('realigned_files', 'apply_to_files')]),
                 (coregister, normalize_and_smooth_func, [('coregistered_files', 'apply_to_files')]),
                 (normalize_struct, skullstrip, [('normalized_files', 'in_file')]),
                 (realign, art, [('realignment_parameters', 'realignment_parameters')]),
                 (normalize_and_smooth_func, art, [('normalized_files', 'realigned_files')]),
                 (skullstrip, art, [('mask_file', 'mask_file')])])
```
27.3 Set up analysis workflow

```python
l1analysis = pe.Workflow(name='analysis')

Generate SPM-specific design information using nipype.interfaces.spm.SpecifyModel.

modelspec = pe.Node(model.SpecifySPMModel(), name="modelspec")
modelspec.inputs.concatenate_runs = True

Generate a first level SPM.mat file for analysis nipype.interfaces.spm.Level1Design.

level1design = pe.Node(spm.Level1Design(), name="level1design")
level1design.inputs.bases = {'hrf': {'derivs': [0, 0]})

Use nipype.interfaces.spm.EstimateModel to determine the parameters of the model.

level1estimate = pe.Node(spm.EstimateModel(), name="level1estimate")
level1estimate.inputs.estimation_method = {'Classical': 1}

Use nipype.interfaces.spm.EstimateContrast to estimate the first level contrasts specified in a few steps above.

contrastestimate = pe.Node(spm.EstimateContrast(), name="contrastestimate")

Use :class: nipype.interfaces.utility.Select to select each contrast for reporting.

selectcontrast = pe.Node(niu.Select(), name="selectcontrast")

Use nipype.interfaces.fsl.Overlay to combine the statistical output of the contrast estimate and a background image into one volume.

overlaystats = pe.Node(fsl.Overlay(), name="overlaystats")
overlaystats.inputs.stat_thresh = (3, 10)
overlaystats.inputs.show_negative_stats = True
overlaystats.inputs.auto_thresh_bg = True

Use nipype.interfaces.fsl.Slicer to create images of the overlaid statistical volumes for a report of the first-level results.

slicestats = pe.Node(fsl.Slicer(), name="slicestats")
slicestats.inputs.all_axial = True
slicestats.inputs.image_width = 750

l1analysis.connect([(modelspec, level1design, [('session_info', 'session_info')]), (level1design, level1estimate, [('spm_mat_file', 'spm_mat_file')]), (level1estimate, contrastestimate, [('spm_mat_file', 'spm_mat_file'), ('beta_images', 'beta_images')]), (contrastestimate, selectcontrast, [('spmT_images', 'inlist')]), (selectcontrast, overlaystats, [('out', 'stat_image')]), (overlaystats, slicestats, [('out_file', 'in_file')])])
```
27.4 Preproc + Analysis pipeline

```python
l1pipeline = pe.Workflow(name='firstlevel')
l1pipeline.connect(
    [(preproc, l1analysis,
      [('realign.realignment_parameters', 'modelspec.realignment_parameters'),
       ('normalize_and_smooth_func.normalized_files',
        'modelspec.functional_runs'), ('art.outlier_files',
        'modelspec.outlier_files'),
       ('skullstrip.mask_file',
        'level1design.mask_image'), ('normalize_struct.normalized_files',
        'overlaystats.background_image'))],
  )
```

27.5 Data specific components

The nipype tutorial contains data for two subjects. Subject data is in two subdirectories, s1 and s2. Each subject directory contains four functional volumes: f3.nii, f5.nii, f7.nii, f10.nii. And one anatomical volume named struct.nii.

Below we set some variables to inform the datasource about the layout of our data. We specify the location of the data, the subject sub-directories and a dictionary that maps each run to a mnemonic (or field) for the run type (struct or func). These fields become the output fields of the datasource node in the pipeline.

In the example below, run ‘f3’ is of type ‘func’ and gets mapped to a nifti filename through a template ‘%s.nii’. So ‘f3’ would become ‘f3.nii’.

```python
# Specify the location of the data.
# data_dir = os.path.abspath('data')
# Specify the subject directories
subject_list = ['s1', 's3']
# Map field names to individual subject runs.
info = dict(
    func=[['subject_id', ['f3', 'f5', 'f7', 'f10']]],
    struct=[['subject_id', 'struct']])

infosource = pe.Node(
    niu.IdentityInterface(fields=['subject_id']), name="infosource")
```

Here we set up iteration over all the subjects. The following line is a particular example of the flexibility of the system. The datasource attribute iterables tells the pipeline engine that it should repeat the analysis on each of the items in the subject_list. In the current example, the entire first level preprocessing and estimation will be repeated for each subject contained in subject_list.

```python
infosource.iterables = ('subject_id', subject_list)
```

Now we create a nipype.interfaces.io.DataGrabber object and fill in the information from above about the layout of our data. The nipype.pipeline.NodeWrapper module wraps the interface object and provides additional housekeeping and pipeline specific functionality.

```python
inputnode = pe.Node(
    niu.IdentityInterface(fields=['in_data']), name='inputnode')
datasource = pe.Node(
    nio.DataGrabber(infields=['subject_id'], outfields=['func', 'struct']),
    name='datasource')
datasource.inputs.template = 'nipype-tutorial/data/%s/%s.nii'
datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True
```
nipype Documentation, Release 1.2.1

We need to create a separate workflow to make the DARTEL template
datasource_dartel = pe.MapNode(
nio.DataGrabber(infields=['subject_id'], outfields=['struct']),
name='datasource_dartel',
iterfield=['subject_id'])
datasource_dartel.inputs.template = 'nipype-tutorial/data/%s/%s.nii'
datasource_dartel.inputs.template_args = dict(
struct=[['subject_id', 'struct']])
datasource_dartel.inputs.sort_filelist = True
datasource_dartel.inputs.subject_id = subject_list

Here we make sure that struct files have names corresponding to the subject ids. This way we will be able to
pick the right field flows later.
rename_dartel = pe.MapNode(
niu.Rename(format_string="subject_id_%(subject_id)s_struct"),
iterfield=['in_file', 'subject_id'],
name='rename_dartel')
rename_dartel.inputs.subject_id = subject_list
rename_dartel.inputs.keep_ext = True
dartel_workflow = spm_wf.create_DARTEL_template(name='dartel_workflow')
dartel_workflow.inputs.inputspec.template_prefix = "template"

This function will allow to pick the right field flow for each subject
def pickFieldFlow(dartel_flow_fields, subject_id):
from nipype.utils.filemanip import split_filename
for f in dartel_flow_fields:
_, name, _ = split_filename(f)
if name.find("subject_id_%s" % subject_id):
return f
raise Exception

pick_flow = pe.Node(
niu.Function(
input_names=['dartel_flow_fields', 'subject_id'],
output_names=['dartel_flow_field'],
function=pickFieldFlow),
name="pick_flow")

27.6 Experimental paradigm specific components
Here we create a function that returns subject-specific information about the experimental paradigm. This is
used by the nipype.interfaces.spm.SpecifyModel to create the information necessary to generate
an SPM design matrix. In this tutorial, the same paradigm was used for every participant.
def subjectinfo(subject_id):
from nipype.interfaces.base import Bunch
from copy import deepcopy
print("Subject ID: %s\n" % str(subject_id))
output = []
names = ['Task-Odd', 'Task-Even']
for r in range(4):
(continues on next page)

27.6. Experimental paradigm specific components

257


onsets = [list(range(15, 240, 60)), list(range(45, 240, 60))]
output.insert(r,
    Bunch(
        conditions=names,
        onsets=deepcopy(onsets),
        durations=[[15] for s in names],
        amplitudes=None,
        tmod=None,
        pmod=None,
        regressor_names=None,
        regressors=None)

return output

Setup the contrast structure that needs to be evaluated. This is a list of lists. The inner list specifies the contrasts and has the following format - [Name, Stat, [list of condition names], [weights on those conditions]. The condition names must match the names listed in the subjectinfo function described above.

cont1 = ('Task>Baseline', 'T', ['Task-Odd', 'Task-Even'], [0.5, 0.5])
cont2 = ('Task-Odd>Task-Even', 'T', ['Task-Odd', 'Task-Even'], [1, -1])
contrasts = [cont1, cont2]

# set up node specific inputs
modelspecref = l1pipeline.inputs.analysis.modelspec
modelspecref.input_units = 'secs'
modelspecref.output_units = 'secs'
modelspecref.time_repetition = 3.
modelspecref.high_pass_filter_cutoff = 120
l1designref = l1pipeline.inputs.analysis.level1design
l1designref.timing_units = modelspecref.output_units
l1designref.interscan_interval = modelspecref.time_repetition
l1pipeline.inputs.analysis.contrastestimate.contrasts = contrasts

# Iterate over each contrast and create report images.
selectcontrast.iterables = ('index', [i for i in range(len(contrasts))])

The nodes created above do not describe the flow of data. They merely describe the parameters used for each function. In this section we setup the connections between the nodes such that appropriate outputs from nodes are piped into appropriate inputs of other nodes. Use the nipype.pipeline.engine.Pipeline to create a graph-based execution pipeline for first level analysis. The config options tells the pipeline engine to use workdir as the disk location to use when running the processes and keeping their outputs. The use_parameterized_dirs tells the engine to create sub-directories under workdir corresponding to the iterables in the pipeline. Thus for this pipeline there will be subject specific The nipype.pipeline.engine.Pipeline.connect function creates the links between the processes, i.e., how data should flow in and out of

levell = pe.Workflow(name="levell")
levell.base_dir = os.path.abspath('spm_dartel_tutorial/workingdir')
levell.connect([
    (inputnode, datasource, [('in_data', 'base_directory')]),
    (inputnode, datasource_dartel, [('in_data', 'base_directory')]),
    (datasource_dartel, rename_dartel, [('struct', 'in_file')]),
    (rename_dartel, dartel_workflow, [('out_file',
        'inputspec.structural_files')]),
    (infosource, datasource, [('subject_id', 'subject_id')]),
    (continues on next page)
27.7 Setup storage results

Use nipype.interfaces.io.DataSink to store selected outputs from the pipeline in a specific location. This allows the user to selectively choose important output bits from the analysis and keep them. The first step is to create a datasink node and then to connect outputs from the modules above to storage locations. These take the following form directory_name[@@subdir] where parts between [] are optional. For example ‘realign.@mean’ below creates a directory called realign in ‘l1output/subject_id/’ and stores the mean image output from the Realign process in the realign directory. If the @ is left out, then a sub-directory with the name ‘mean’ would be created and the mean image would be copied to that directory.

```python
# store relevant outputs from various stages of the 1st level analysis
level1.connect(
    (infosource, datasink, [('subject_id', 'container'),
                             ('subject_id', getstripdir), 'strip_dir')],
    (l1pipeline, datasink,
     [('analysis.contrastestimate.con_images', 'contrasts.@con'),
      ('analysis.contrastestimate.spmT_images', 'contrasts.@T')]),
    (infosource, report, [('subject_id', 'container'),
                           ('subject_id', getstripdir), 'strip_dir'])),
    (l1pipeline, report, [('analysis.slicestats.out_file', '@report')]),
)
```

(continues on next page)
The code discussed above sets up all the necessary data structures, appropriate parameters, and the connectivity between the sses, but does not generate any output. To actually run the pipeline on the data, the `nipype.pipeline.engine.Pipeline.Run` is used.

```python
if __name__ == '__main__':
    level1.run(plugin_args={'n_procs': 4})
    level1.write_graph()
```

### 27.8 Setup level 2 pipeline

Use `nipype.interfaces.io.DataGrabber` to extract the contrast images across a group of first level subjects. Unlike the previous pipeline that iterated over subjects, this pipeline will iterate over contrasts.

```python
# collect all the con images for each contrast.
contrast_ids = list(range(1, len(contrasts) + 1))
l2source = pe.Node(nio.DataGrabber(infields=['fwhm', 'con']), name="l2source")
# we use .*i* to capture both .img (SPM8) and .nii (SPM12)
l2source.inputs.template = os.path.abspath('spm_dartel_tutorial/l1output/*/con*/*/_fwhm_%d/con_%04d.*i*')
# iterate over all contrast images
l2source.iterables = [('fwhm', fwhmlist), ('con', contrast_ids)]
l2source.inputs.sort_filelist = True
```

Use `nipype.interfaces.spm.OneSampleTTestDesign` to perform a simple statistical analysis of the contrasts from the group of n in this example).

```python
# setup a 1-sample t-test node
onesamplettestdes = pe.Node(spm.OneSampleTTestDesign(), name="onesamplettestdes")
l2estimate = pe.Node(spm.EstimateModel(), name="level2estimate")
l2estimate.inputs.estimation_method = {'Classical': 1}
l2conestimate = pe.Node(spm.EstimateContrast(), name="level2conestimate")
cont1 = ('Group', 'T', ['mean'], [1])
l2conestimate.inputs.contrasts = [cont1]
l2conestimate.inputs.group_contrast = True
```

As before, we setup a pipeline to connect these two nodes (l2source lettest).

```python
l2pipeline = pe.Workflow(name="level2")
l2pipeline.base_dir = os.path.abspath('spm_dartel_tutorial/l2output')
l2pipeline.connect([
    (l2source, onesamplettestdes, [('outfiles', 'in_files')]),
    (onesamplettestdes, l2estimate, [('spm_mat_file', 'spm_mat_file')]),
    (l2estimate, l2conestimate, [('spm_mat_file', 'spm_mat_file')]),
    (l2conestimate, 'beta_images', 'beta_images'),
    (l2conestimate, 'residual_image', 'residual_image')])
```

### 27.9 Execute the second level pipeline

```python
if __name__ == '__main__':
    l2pipeline.run()
```

Example source code
You can download the full source code of this example. This same script is also included in the Nipype source distribution under the examples directory.
28.1 Introduction

The fmri_spm_face.py recreates the classical workflow described in the SPM8 manual using face dataset that can be downloaded from http://www.fil.ion.ucl.ac.uk/spm/data/face_rep/:

```python
python fmri_spm.py
```

Import necessary modules from nipype.

```python
from __future__ import division
from builtins import range
import os  # system functions
import nipype.interfaces.io as nio  # Data i/o
import nipype.interfaces.spm as spm  # spm
import nipype.interfaces.matlab as mlab  # how to run matlab
import nipype.interfaces.utility as util  # utility
import nipype.pipeline.engine as pe  # pipeline engine
import nipype.algorithms.modelgen as model  # model specification
```

28.1.1 iminaries

Set any package specific configuration. The output file format for FSL routines is being set to uncompressed NIFTI and a specific version of matlab is being used. The uncompressed format is required because SPM does not handle compressed NIFTI.

```python
# Set the way matlab should be called
mlab.MatlabCommand.set_default_matlab_cmd("matlab -nodesktop -nosplash")
# If SPM is not in your MATLAB path you should add it here
# mlab.MatlabCommand.set_default_paths('/path/to/your/spm8')
```

28.1.2 lows

In this tutorial we will be setting up a hierarchical workflow for spm analysis. It one is slightly different then the one used in spm_tutorial2. eprocessing workflow ——————– This is a generic preprocessing workflow that can be used by different analyses
Use `nipype.interfaces.spm.Realign` for motion correction and register all images to the mean image.

```python
realign = pe.Node(interface=spm.Realign(), name="realign"
```

Use `nipype.interfaces.spm.SliceTiming()` to perform a rigid body registration of the functional data to the structural data.

```python
coregister = pe.Node(interface=spm.Coregister(), name="coregister"
coregister.inputs.jobtype = 'estimate'
```

Uncomment the following line for faster execution

```python
# segment.inputs.gaussians_per_class = [1, 1, 1, 4]
```

Warp functional and structural data to SPM's T1 template using `nipype.interfaces.spm.Normalize`.

```python
normalize_func = pe.Node(interface=spm.Normalize(), name="normalize_func"
```

Smooth the functional data using `nipype.interfaces.spm.Smooth`.

```python
smooth = pe.Node(interface=spm.Smooth(), name="smooth"
```

`write_voxel_sizes` is the input of the normalize interface that is recommended to be set to the voxel sizes of the target volume. There is no need to set it manually since we van infer it from data using the following function:

```python
def get_vox_dims(volume):
    import nibabel as nb
    from nipype.utils import NUMPY_MMAP
    if isinstance(volume, list):
        volume = volume[0]
    nii = nb.load(volume, mmap=NUMPY_MMAP)
    hdr = nii.header
    voxdims = hdr.get_zooms()
    return [float(voxdims[0]), float(voxdims[1]), float(voxdims[2])]
```

Here we are connecting all the nodes together. Notice that we add the merge node only if you choose to use 4D. Also `get_vox_dims` function is passed along the input volume of normalise to set the optimal voxel sizes.

```python
preproc.connect(
    [realign, coregister, [('mean_image', 'target')]),
    [coregister, segment, [('coregistered_source', 'data')]),
    [segment, normalize_func, [('transformation_mat', 'parameter_file')]),
    [segment, normalize_struc, [('transformation_mat', 'parameter_file')], ('bias_corrected_image',
    [apply_to_files'])),
    [('bias_corrected_image', get_vox_dims), 'write_voxel_sizes')]),
    [realign, slice_timing, [('realigned_files', 'in_files')]),
    [slice_timing, normalize_func, [('timecorrected_files', 'apply_to_files')],
    [(timecorrected_files', get_vox_dims),
```

(continues on next page)
28.1.3 Set up analysis workflow

```python
l1analysis = pe.Workflow(name='analysis')

Generate SPM-specific design information using `nipype.interfaces.spm.SpecifyModel`.
```modelspec = pe.Node(interface=model.SpecifySPMModel(), name="modelspec")

Generate a first level SPM.mat file for analysis `nipype.interfaces.spm.Level1Design`.
```level1design = pe.Node(interface=spm.Level1Design(), name="level1design")

Use `nipype.interfaces.spm.EstimateModel` to determine the parameters of the model.
```level1estimate = pe.Node(interface=spm.EstimateModel(), name="level1estimate")
level1estimate.inputs.estimation_method = {'Classical': 1}
threshold = pe.Node(interface=spm.Threshold(), name="threshold")

Use `nipype.interfaces.spm.EstimateContrast` to estimate the first level contrasts specified in a few steps above.
```contrastestimate = pe.Node(
    interface=spm.EstimateContrast(), name="contrastestimate")

def pickfirst(l):
    return l[0]

l1analysis.connect([
    (modelspec, level1design, [('session_info', 'session_info')]),
    (level1design, level1estimate, [('spm_mat_file', 'spm_mat_file')]),
    (level1estimate, contrastestimate, [('spm_mat_file', 'spm_mat_file'), ('beta_images', 'beta_images'), ('residual_image', 'residual_image')]),
    (contrastestimate, threshold, [('spmT_images', pickfirst), ('stat_image')]),
])
```

28.1.4 Preproc + Analysis pipeline

```python
l1pipeline = pe.Workflow(name='firstlevel')
l1pipeline.connect([(preproc, l1analysis,
    [('realign.realignment_parameters', 'modelspec.realignment_parameters')])])
```

Plugging in `functional_runs` is a bit more complicated, because model spec expects a list of runs. Every run can be a 4D file or a list of 3D files. Therefore for 3D analysis we need a list of lists and to make one we need a helper function.
def makelist(item):
    return [item]

llpipeline.connect([(preproc, llanalysis, [{('smooth.smoothed_files',
                  makelist),
                   'modelspec.functional_runs'}])])

28.1.5 Data specific components

In this tutorial there is only one subject M03953. Below we set some variables to inform the datasource about the layout of our data. We specify the location of the data, the subject sub-directories and a dictionary that maps each run to a mnemonic (or field) for the run type (struct or func). These fields become the output fields of the datasource node in the pipeline.

```python
# Specify the location of the data downloaded from http://www.fil.ion.ucl.ac.uk/
# spm/data/face_rep/face_rep_SPM5.html
data_dir = os.path.abspath('spm_face_data')

# Specify the subject directories
subject_list = ['M03953']

# Map field names to individual subject runs.
info = dict(
    func=[['RawEPI', 'subject_id', 5, ['_%$04d' % i for i in range(6, 357)]],
          struct=[['Structural', 'subject_id', 7, '']])

infosource = pe.Node(
    interface=util.IdentityInterface(fields=['subject_id']), name=" infosource")
```

Here we set up iteration over all the subjects. The following line is a particular example of the flexibility of the system. The datasource attribute iterables tells the pipeline engine that it should repeat the analysis on each of the items in the subject_list. In the current example, the entire first level preprocessing and estimation will be repeated for each subject contained in subject_list.

```python
infosource.iterables = ('subject_id', subject_list)
```

Now we create a nipype.interfaces.io.DataGrabber object and fill in the information from above about the layout of our data. The nipype.pipeline.NodeWrapper module wraps the interface object and provides additional housekeeping and pipeline specific functionality.

```python
datasource = pe.Node(
    interface=nio.DataGrabber(
        infields=['subject_id'], outfields=['func', 'struct'],
        name='datasource')
datasource.inputs.base_directory = data_dir
datasource.inputs.template = '%s/s%s_%04d%s.img'
datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True
```

28.1.6 Experimental paradigm specific components

Here we create a structure that provides information about the experimental paradigm. This is used by the nipype.interfaces.spm.SpecifyModel to create the information necessary to generate an SPM design matrix.

```python
from nipype.interfaces.base import Bunch
```

We’re importing the onset times from a mat file (found on http://www.fil.ion.ucl.ac.uk/spm/data/face_rep/).
from scipy.io.matlab import loadmat
mat = loadmat(os.path.join(data_dir, "sots.mat"), struct_as_record=False)
sot = mat['sot'][0]
itemlag = mat['itemlag'][0]

subjectinfo = [
    Bunch(
        conditions=['N1', 'N2', 'F1', 'F2'],
        onsets=[sot[0], sot[1], sot[2], sot[3]],
        durations=[[0], [0], [0], [0]],
        amplitudes=None,
        tmod=None,
        pmod=None,
        regressor_names=None,
        regressors=None)
]

# Setup the contrast structure that needs to be evaluated. This is a list of lists. The inner list specifies the contrasts
# cond1 = ('positive effect of condition', 'T',
#       ['N1*bf(1)', 'N2*bf(1)', 'F1*bf(1)', 'F2*bf(1)'], [1, 1, 1, 1])
# cond2 = ('positive effect of condition_dtemp', 'T',
#       ['N1*bf(2)', 'N2*bf(2)', 'F1*bf(2)', 'F2*bf(2)'], [1, 1, 1, 1])
# cond3 = ('positive effect of condition_ddisp', 'T',
#       ['N1*bf(3)', 'N2*bf(3)', 'F1*bf(3)', 'F2*bf(3)'], [1, 1, 1, 1])

Setup the contrast structure that needs to be evaluated. This is a list of lists. The inner list specifies the contrasts
and has the following format - [Name,Stat,[list of condition names],[weights on those conditions]. The condition
names must match the names listed in the subjectinfo function described above.

# non-famous > famous
fam1 = ('positive effect of Fame', 'T',
       ['N1*bf(1)', 'N2*bf(1)', 'F1*bf(1)', 'F2*bf(1)'], [1, 1, -1, -1])

fam2 = ('positive effect of Fame_dtemp', 'T',
       ['N1*bf(2)', 'N2*bf(2)', 'F1*bf(2)', 'F2*bf(2)'], [1, 1, -1, -1])

fam3 = ('positive effect of Fame_ddisp', 'T',
       ['N1*bf(3)', 'N2*bf(3)', 'F1*bf(3)', 'F2*bf(3)'], [1, 1, -1, -1])

# repl1 > repl2
rep1 = ('positive effect of Rep', 'T',
       ['N1*bf(1)', 'N2*bf(1)', 'F1*bf(1)', 'F2*bf(1)'], [1, -1, 1, -1])

rep2 = ('positive effect of Rep_dtemp', 'T',
       ['N1*bf(2)', 'N2*bf(2)', 'F1*bf(2)', 'F2*bf(2)'], [1, -1, 1, -1])

rep3 = ('positive effect of Rep_ddisp', 'T',
       ['N1*bf(3)', 'N2*bf(3)', 'F1*bf(3)', 'F2*bf(3)'], [1, -1, 1, -1])

int1 = ('positive interaction of Fame x Rep', 'T',
       ['N1*bf(1)', 'N2*bf(1)', 'F1*bf(1)', 'F2*bf(1)'], [-1, -1, -1, 1])

int2 = ('positive interaction of Fame x Rep_dtemp', 'T',
       ['N1*bf(2)', 'N2*bf(2)', 'F1*bf(2)', 'F2*bf(2)'], [-1, -1, -1, 1])

int3 = ('positive interaction of Fame x Rep_ddisp', 'T',
       ['N1*bf(3)', 'N2*bf(3)', 'F1*bf(3)', 'F2*bf(3)'], [-1, -1, -1, 1])

contf1 = ['average effect condition', 'F', [cond1, cond2, cond3]]
contf2 = ['main effect Fam', 'F', [fam1, fam2, fam3]]
contf3 = ['main effect Rep', 'F', [rep1, rep2, rep3]]
contf4 = ['interaction: Fam x Rep', 'F', [int1, int2, int3]]

contrasts = [
    cond1, cond2, cond3, fam1, fam2, fam3, rep1, rep2, rep3, int1, int2, int3,
    contf1, contf2, contf3, contf4]

Setting up nodes inputs

28.1. Introduction
num_slices = 24
TR = 2.

slice_timingref = l1pipeline.inputs.preproc.slice_timing
slice_timingref.num_slices = num_slices
slice_timingref.time_repetition = TR
slice_timingref.time_acquisition = TR - TR / float(num_slices)
slice_timingref.slice_order = list(range(num_slices, 0, -1))
slice_timingref.ref_slice = int(num_slices / 2)

l1pipeline.inputs.preproc.smooth.fwhm = [8, 8, 8]

# set up node specific inputs
modelspecref = l1pipeline.inputs.analysis.modelspec
modelspecref.input_units = 'scans'
modelspecref.output_units = 'scans'
modelspecref.time_repetition = TR
modelspecref.high_pass_filter_cutoff = 120

l1designref = l1pipeline.inputs.analysis.level1design
l1designref.timing_units = modelspecref.output_units
l1designref.interscan_interval = modelspecref.time_repetition
l1designref.microtime_resolution = slice_timingref.num_slices
l1designref.microtime_onset = slice_timingref.ref_slice
l1designref.bases = {'hrf': {'derivs': [1, 1]}}
The following lines automatically inform SPM to create a default set of contrats for a factorial design.

# l1designref.factor_info = [dict(name = 'Fame', levels = 2),
#                            dict(name = 'Rep', levels = 2)]

l1pipeline.inputs.analysis.modelspec.subject_info = subjectinfo
l1pipeline.inputs.analysis.contrastestimate.contrasts = contrasts
l1pipeline.inputs.analysis.threshold.contrast_index = 1
Use derivative estimates in the non-parametric model

l1pipeline.inputs.analysis.contrastestimate.use_derivs = True

Setting up parametric variation of the model

subjectinfo_param = [
    Bunch(
        conditions=['N1', 'N2', 'F1', 'F2'],
        onsets=[sot[0], sot[1], sot[2], sot[3]],
        durations=[[0], [0], [0], [0]],
        amplitudes=None,
        tmod=None,
        pmod=[
            None,
            Bunch(name='Lag', param=itemlag[1].tolist(), poly=[2]), None,
            Bunch(name='Lag', param=itemlag[3].tolist(), poly=[2])
        ],
        regressor_names=None,
        regressors=None
    )
]
cont1 = ('Famous_lag1', 'T', ['F2xLag^1'], [1])
cont2 = ('Famous_lag2', 'T', ['F2xLag^2'], [1])

(continues on next page)
(continued from previous page)

```python
fcont1 = ('Famous Lag', 'F', [cont1, cont2])
paramcontrasts = [cont1, cont2, fcont1]

paramanalysis = llanalysis.clone(name='paramanalysis')

paramanalysis.inputs.level1design.bases = {'hrf': {'derivs': [0, 0]}}
paramanalysis.inputs.modelspec.subject_info = subjectinfo_param
paramanalysis.inputs.level1design.bases = {'hrf': {'derivs': [0, 0]}}
paramanalysis.inputs.contrastestimate.contrasts = paramcontrasts
paramanalysis.inputs.contrastestimate.use_derivs = False

l1pipeline.connect(
    [(preproc, paramanalysis, 
      ('realign.realignment_parameters', 'modelspec.realignment_parameters'),
      ('smooth.smoothed_files', makelist), 'modelspec.functional_runs)])
```

### 28.1.7 Setup the pipeline

The nodes created above do not describe the flow of data. They merely describe the parameters used for each function. In this section we setup the connections between the nodes such that appropriate outputs from nodes are piped into appropriate inputs of other nodes.

Use the `nipype.pipeline.engine.Pipeline` to create a graph-based execution pipeline for first level analysis. The config options tells the pipeline engine to use `workdir` as the disk location to use when running the processes and keeping their outputs. The `use_parameterized_dirs` tells the engine to create sub-directories under `workdir` corresponding to the iterables in the pipeline. Thus for this pipeline there will be subject specific sub-directories.

The `nipype.pipeline.engine.Pipeline.connect` function creates the links between the processes, i.e., how data should flow in and out of the processing nodes.

```python
level1 = pe.Workflow(name="level1")
level1.base_dir = os.path.abspath('spm_face_tutorial/workingdir')

level1.connect([(infosource, datasource, 
                 ('subject_id', 'subject_id'))],
               (datasource, l1pipeline,
                ('struct', 'preproc.coregister.source'),
                ('func', 'preproc.realign.in_files')))
```

### 28.1.8 Setup storage results

Use `nipype.interfaces.io.DataSink` to store selected outputs from the pipeline in a specific location. This allows the user to selectively choose important output bits from the analysis and keep them.

The first step is to create a datasink node and then to connect outputs from the modules above to storage locations. These take the following form directory_name[@subdir] where parts between [] are optional. For example `realign.@mean` below creates a directory called realign in `l1output/subject_id/` and stores the mean image output from the Realign process in the realign directory. If the @ is left out, then a sub-directory with the name `mean` would be created and the mean image would be copied to that directory.

```python
datasink = pe.Node(interface=nio.DataSink(), name="datasink")
datasink.inputs.base_directory = os.path.abspath('spm_auditory_tutorial/l1output')

def getstripdir(subject_id):
    import os
    return os.path.join(
```
os.path.abspath('spm_auditory_tutorial/workingdir'),
'_subject_id_%s' % subject_id)

# store relevant outputs from various stages of the 1st level analysis
level1.connect(
    (infosource, datasink, [('subject_id', 'container'),
      ('subject_id', getstripdir), 'strip_dir')],
    (llpipeline, datasink,
     [('analysis.contrastestimate.con_images', 'contrasts.@con'),
      ('analysis.contrastestimate.spmT_images', 'contrasts.@T'),
      ('paramanalysis.contrastestimate.con_images',
       'paramcontrasts.@con'), ('paramanalysis.contrastestimate.spmT_images',
       'paramcontrasts.@T')],
    )
)

code discussed above sets up all the necessary data structures

opriate parameters and the connectivity between the sses, but does not generate any output. To actually run the
sis on the data the nipype.pipeline.engine.Pipeline.Run

if __name__ == '__main__':
    level1.run()
    level1.write_graph()

Example source code
You can download the full source code of this example. This same script is also included in
the Nipype source distribution under the examples directory.
The fmri_spm.py integrates several interfaces to perform a first and second level analysis on a two-subject data set. The tutorial can be found in the examples folder. Run the tutorial from inside the nipype tutorial directory:

```python
python fmri_spm_nested.py
```

Import necessary modules from nipype.

```python
from __future__ import print_function
from builtins import str
from builtins import range
import os.path as op  # system functions
from nipype.interfaces import io as nio  # Data i/o
from nipype.interfaces import spm as spm  # spm
# from nipype.interfaces import matlab as mlab  # how to run matlab
from nipype.interfaces import fsl as fsl  # fsl
from nipype.interfaces import utility as niu  # utility
from nipype.pipeline import engine as pe  # pypeline engine
from nipype.algorithms import rapidart as ra  # artifact detection
from nipype.algorithms import modelgen as model  # model specification
```

### 29.1 Preliminaries

Set any package specific configuration. The output file format for FSL routines is being set to uncompressed NIFTI and a specific version of matlab is being used. The uncompressed format is required because SPM does not handle compressed NIFTI.

```python
# Tell fsl to generate all output in uncompressed nifti format
fsl.FSLCommand.set_default_output_type('NIFTI')

# Set the way matlab should be called
# mlab.MatlabCommand.set_default_matlab_cmd("matlab -nodesktop -nosplash")
# mlab.MatlabCommand.set_default_paths('/software/spm8')
```
29.2 lows

In this tutorial we will be setting up a hierarchical workflow for spm analysis. This will demonstrate how pre-defined workflows can be setup and shared across users, projects and labs. Example of how to inline functions in connect()

```python
def _template_path(in_data):
    import os.path as op
    return op.abspath(op.join(in_data, 'nipype-tutorial/data/T1.nii'))
```

29.3 Set-up preprocessing workflow

This is a generic preprocessing workflow that can be used by different analyses

```python
preproc = pe.Workflow(name='preproc')
```

A node called inputnode is set to designate the path in which input data are located:

```python
inputnode = pe.Node(
    niu.IdentityInterface(fields=['in_data']), name='inputnode')
```

Use nipype.interfaces.spm.Realign for motion correction and register all images to the mean image.

```python
realign = pe.Node(spm.Realign(), name="realign")
realign.inputs.register_to_mean = True
```

Use nipype.algorithms.rapidart to determine which of the images in the functional series are outliers based on deviations in intensity or movement.

```python
art = pe.Node(ra.ArtifactDetect(), name="art")
art.inputs.use_differences = [True, False]
art.inputs.use_norm = True
art.inputs.norm_threshold = 1
art.inputs.zintensity_threshold = 3
art.inputs.mask_type = 'file'
art.inputs.parameter_source = 'SPM'
```

Skull strip structural images using nipype.interfaces.fsl.BET.

```python
skullstrip = pe.Node(fsl.BET(), name="skullstrip")
skullstrip.inputs.mask = True
```

Use nipype.interfaces.spm.Coregister to perform a rigid body registration of the functional data to the structural data.

```python
coregister = pe.Node(spm.Coregister(), name="coregister")
coregister.inputs.jobtype = 'estimate'
```

Warp functional and structural data to SPM's T1 template using nipype.interfaces.spm.Normalize. The tutorial data set includes the template image, T1.nii.

```python
normalize = pe.Node(spm.Normalize(), name="normalize")
```

Smooth the functional data using nipype.interfaces.spm.Smooth.

```python
smooth = pe.Node(spm.Smooth(), name="smooth")
fwhmlist = [4]
smooth.iterables = ('fwhm', fwhmlist)
```

(continues on next page)
29.4 Set up analysis workflow

```python
l1analysis = pe.Workflow(name='analysis')

Generate SPM-specific design information using nipype.interfaces.spm.SpecifyModel.

modelspec = pe.Node(model.SpecifySPMModel(), name="modelspec")
modelspec.inputs.concatenate_runs = True

Generate a first level SPM.mat file for analysis nipype.interfaces.spm.Level1Design.

level1design = pe.Node(spm.Level1Design(), name="level1design")
level1design.inputs.bases = {'hrf': {'derivs': [0, 0]}}

Use nipype.interfaces.spm.EstimateModel to determine the parameters of the model.

level1estimate = pe.Node(spm.EstimateModel(), name="level1estimate")
level1estimate.inputs.estimation_method = {'Classical': 1}

Use nipype.interfaces.spm.EstimateContrast to estimate the first level contrasts specified in a few steps above.

contrastestimate = pe.Node(spm.EstimateContrast(), name="contrastestimate")

Use :class: nipype.interfaces.utility.Select to select each contrast for reporting.

selectcontrast = pe.Node(niu.Select(), name="selectcontrast")

Use nipype.interfaces.fsl.Overlay to combine the statistical output of the contrast estimate and a background image into one volume.

overlaystats = pe.Node(fsl.Overlay(), name="overlaystats")
overlaystats.inputs.stat_thresh = (3, 10)
overlaystats.inputs.show_negative_stats = True
overlaystats.inputs.auto_thresh_bg = True

Use nipype.interfaces.fsl.Slicer to create images of the overlaid statistical volumes for a report of the first-level results.

slicestats = pe.Node(fsl.Slicer(), name="slicestats")
slicestats.inputs.all_axial = True
slicestats.inputs.image_width = 750
```

(continues on next page)
29.5 Preproc + Analysis pipeline

```python
l1pipeline = pe.Workflow(name='firstlevel')
l1pipeline.connect([(
    preproc, l1analysis,
    [('realign.realignment_parameters', 'modelspec.realignment_parameters'),
     ('smooth.smoothed_files', 'modelspec.functional_runs'), ('art.outlier_files', 'modelspec.outlier_files'),
     ('skullstrip.mask_file', 'level1design.mask_image'), ('normalize.normalized_source', 'overlaystats.background_image')],
]
```

29.6 Data specific components

The nipype tutorial contains data for two subjects. Subject data is in two subdirectories, s1 and s2. Each subject directory contains four functional volumes: f3.nii, f5.nii, f7.nii, f10.nii. And one anatomical volume named struct.nii.

Below we set some variables to inform the datasource about the layout of our data. We specify the location of the data, the subject sub-directories and a dictionary that maps each run to a mnemonic (or field) for the run type (struct or func). These fields become the output fields of the datasource node in the pipeline.

In the example below, run ‘f3’ is of type ‘func’ and gets mapped to a nifti filename through a template ‘%s.nii’. So ‘f3’ would become ‘f3.nii’.

```python
# Specify the subject directories
subject_list = ['s1', 's3']
# Map field names to individual subject runs.
info = dict(  
    func=[['subject_id', ['f3', 'f5', 'f7', 'f10']]],  
    struct=[['subject_id', 'struct']])

infosource = pe.Node(  
    niu.IdentityInterface(fields=['subject_id']), name="infosource")
```

Here we set up iteration over all the subjects. The following line is a particular example of the flexibility of the system. The datasource attribute iterables tells the pipeline engine that it should repeat the analysis on each of the items in the subject_list. In the current example, the entire first level preprocessing and estimation will be repeated for each subject contained in subject_list.

```python
infosource.iterables = ('subject_id', subject_list)
```
Now we create a `nipype.interfaces.io.DataGrabber` object and fill in the information from above about the layout of our data. The `nipype.pipeline.NodeWrapper` module wraps the interface object and provides additional housekeeping and pipeline specific functionality.

```python
datasource = pe.Node(
    nio.DataGrabber(infields=['subject_id'], outfields=['func', 'struct']),
    name='datasource')
datasource.inputs.template = 'nipype-tutorial/data/%s/
    %s.nii'
datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True
```

### 29.7 Experimental paradigm specific components

Here we create a function that returns subject-specific information about the experimental paradigm. This is used by the `nipype.interfaces.spm.SpecifyModel` to create the information necessary to generate an SPM design matrix. In this tutorial, the same paradigm was used for every participant.

```python
def subjectinfo(subject_id):
    from nipype.interfaces.base import Bunch
    from copy import deepcopy

    print("Subject ID: \%s\n" % str(subject_id))
    output = []
    names = ['Task-Odd', 'Task-Even']
    for r in range(4):
        onsets = [list(range(15, 240, 60)), list(range(45, 240, 60))]
        output.insert(r,
            Bunch(
                conditions=names,
                onsets=deepcopy(onsets),
                durations=[[15] for s in names],
                amplitudes=None,
                tmod=None,
                pmod=None,
                regressor_names=None,
                regressors=None))

    return output
```

Setup the contrast structure that needs to be evaluated. This is a list of lists. The inner list specifies the contrasts and has the following format - [Name,Stat,[list of condition names],[weights on those conditions]. The condition names must match the `names` listed in the `subjectinfo` function described above.

```python
cont1 = ('Task>Baseline', 'T', ['Task-Odd', 'Task-Even'], [0.5, 0.5])
cont2 = ('Task-Odd>Task-Even', 'T', ['Task-Odd', 'Task-Even'], [1, -1])
contrasts = [cont1, cont2]
```

# set up node specific inputs
modelspecref = llpipeline.inputs.analysis.modelspec
modelspecref.input_units = 'secs'
modelspecref.output_units = 'secs'
modelspecref.time_repetition = 3.
modelspecref.high_pass_filter_cutoff = 120
lldesignref = llpipeline.inputs.analysis.levelldesign
lldesignref.timing_units = modelspecref.output_units
lldesignref.interscan_interval = modelspecref.time_repetition
llpipeline.inputs.analysis.contrastestimate.contrasts = contrasts
```

(continues on next page)
The nodes created above do not describe the flow of data. They merely describe the parameters used for each function. In this section we setup the connections between the nodes such that appropriate outputs from nodes are piped into appropriate inputs of other nodes. Use the `nipype.pipeline.engine.Pipeline` to create a graph-based execution pipeline for first level analysis. The config options tells the pipeline engine to use `workdir` as the disk location to use when running the processes and keeping their outputs. The `use_parameterized_dirs` tells the engine to create sub-directories under `workdir` corresponding to the iterables in the pipeline. Thus for this pipeline there will be subject specific The `nipype.pipeline.engine.Pipeline.connect` function creates the links between the processes, i.e., how data should flow in and out of

```python
levell = pe.Workflow(name="levell")
levell.base_dir = op.abspath('spm_tutorial2/workingdir')

levell.connect(
    ([inputnode, datasource, [('in_data', 'base_directory')]),
    (infosource, datasource, [('subject_id', 'subject_id')]),
    (datasource, l1pipeline, [('func', 'preproc.realign.in_files'),
                              ('struct', 'preproc.coregister.target'),
                              ('struct', 'preproc.normalize.source')]),
    (infosource, l1pipeline, [('subject_id', subjectinfo),
                              'analysis.modspec.subject_info'])),
)
```

### 29.8 Setup storage results

Use `nipype.interfaces.io.DataSink` to store selected outputs from the pipeline in a specific location. This allows the user to selectively choose important output bits from the analysis and keep them. The first step is to create a datasink node and then to connect outputs from the modules above to storage locations. These take the following form `directory_name[@subdir]` where parts between `[]` are optional. For example ‘realign.@mean’ below creates a directory called realign in ‘l1output/subject_id/’ and stores the mean image output from the Realign process in the realign directory. If the `@` is left out, then a sub-directory with the name ‘mean’ would be created and the mean image would be copied to that directory.

```python
datasink = pe.Node(nio.DataSink(), name="datasink")
datasink.inputs.base_directory = op.abspath('spm_tutorial2/l1output')
report = pe.Node(nio.DataSink(), name='report')
report.inputs.base_directory = op.abspath('spm_tutorial2/report')
report.inputs.parameterization = False

def getstripdir(subject_id):
    import os.path as op
    return op.join(op.abspath('spm_tutorial2/workingdir'), '_subject_id_%s' % subject_id)

# store relevant outputs from various stages of the 1st level analysis
levell.connect(
    (infosource, datasink, [('subject_id', 'container')],
     ('subject_id', getstripdir), 'strip_dir'))
```

(continues on next page)
code discussed above sets up all the necessary data structures appropriate parameters and the connectivity between the sses, but does not generate any output. To actually run the sis on the data the nipype.pipeline.engine.Pipeline.Run

```python
if __name__ == '__main__':
    level1.run('MultiProc')
    level1.write_graph()
```

## 29.9 Setup level 2 pipeline

Use nipype.interfaces.io.DataGrabber to extract the contrast images across a group of first level subjects. Unlike the previous pipeline that iterated over subjects, this pipeline will iterate over contrasts.

```python
# collect all the con images for each contrast.
contrast_ids = list(range(1, len(contrasts) + 1))
l2source = pe.Node(nio.DataGrabber(infields=['fwhm', 'con']), name="l2source")
# we use .*i* to capture both .img (SPM8) and .nii (SPM12)
l2source.inputs.template = op.abspath('spm_tutorial2/l1output/*/con*/*/_fwhm_%d/con_%04d.*i*')
# iterate over all contrast images
l2source.iterables = [('fwhm', fwhmlist), ('con', contrast_ids)]
l2source.inputs.sort_filelist = True
```

Use nipype.interfaces.spm.OneSampleTTestDesign to perform a simple statistical analysis of the contrasts from the group of n this example).

```python
# setup a 1-sample t-test node
onesampttestdes = pe.Node(spm.OneSampleTTestDesign(), name="onesampttestdes")
l2estimate = pe.Node(spm.EstimateModel(), name="level2estimate")
l2estimate.inputs.estimation_method = {'Classical': 1}
l2conestimate = pe.Node(spm.EstimateContrast(), name="level2conestimate")
cont1 = ('Group', 'T', ['mean'], [1])
l2conestimate.inputs.contrasts = [cont1]
l2conestimate.inputs.group_contrast = True
```

As before, we setup a pipeline to connect these two nodes (l2source lettest).

```python
l2pipeline = pe.Workflow(name="level2")
l2pipeline.base_dir = op.abspath('spm_tutorial2/l2output')
l2pipeline.connect([
    (l2source, onesampttestdes, [('outfiles', 'in_files')]),
    (onesampttestdes, l2estimate, [('spm_mat_file', 'spm_mat_file')]),
    (l2estimate, l2conestimate, [('spm_mat_file', 'spm_mat_file')], ('beta_images', 'beta_images'),
     ('residual_image', 'residual_image')])
]}
```
29.10 Execute the second level pipeline

```python
if __name__ == '__main__':
    l2pipeline.run('MultiProc')
```

**Example source code**
You can download the full source code of this example. This same script is also included in the Nipype source distribution under the examples directory.
Using nipype in an imperative way: caching without workflow
Note that in the following example, we are calling command-lines with disk I/O that persists across runs, but we never have to worry about the file names or the directories.
The disk location of the persistence is encoded by hashes. To find out where an operation has been persisted, simply look in it’s output variable:

```
out.runtime.cwd
```

```python
from nipype.interfaces import fsl
fsl.FSLCommand.set_default_output_type('NIFTI')

from nipype.caching import Memory

import glob

# First retrieve the list of files that we want to work upon
in_files = glob.glob('data/*/f3.nii')

# Create a memory context
mem = Memory('.')

# Apply an arbitrary (and pointless, here) threshold to the files
threshold = [mem.cache(fsl.Threshold)(in_file=f, thresh=i)
             for i, f in enumerate(in_files)]

# Merge all these files along the time dimension
out_merge = mem.cache(fsl.Merge)(
    dimension="t",
    in_files=[t.outputs.out_file for t in threshold],
)

# And finally compute the mean
out_mean = mem.cache(fsl.MeanImage)(in_file=out_merge.outputs.merged_file)

# To avoid having increasing disk size we can keep only what was touched
```

(continues on next page)
# in this run
# mem.clear_previous_runs()

# or what wasn’t used since the start of 2011
# mem.clear_runs_since(year=2011)

**Example source code**

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the examples directory.
A preprocessing workflow for Siemens resting state data.
This workflow makes use of:
- ANTS
- FreeSurfer
- FSL
- SPM
- CompCor

For example:

```python
python rsfmri_preprocessing.py -d /data/12345-34-1.dcm -f /data/Resting.nii
   -s subj001 -o output -p PBS --plugin_args "dict(qsub_args='--q many')"
```
or

```python
python rsfmri_vol_surface_preprocessing.py -f SUB_1024011/E?/func/rest.nii
   --subjects_dir fsdata --slice_times 0 17 18 2 19 3 20 4 21 5 22 6 23
   7 24 8 25 9 26 10 27 11 28 12 29 13 30 14 31 15 32 16 -o .
```

This workflow takes resting timeseries and a Siemens dicom file corresponding to it and preprocesses it to produce timeseries coordinates or grayordinates.
This workflow also requires 2mm subcortical atlas and templates that are available from:
http://mindboggle.info/data.html specifically the 2mm versions of:
- Joint Fusion Atlas
- MNI template

```python
from __future__ import division, unicode_literals
from builtins import open, range, str
import os

from nipype.interfaces.base import CommandLine
CommandLine.set_default_terminal_output('allatonce')

from dicom import read_file
```

(continues on next page)
from nipype.interfaces import (spm, fsl, Function, ants, freesurfer)
from nipype.interfaces.c3 import C3dAffineTool
fsl.FSLCommand.set_default_output_type('NIFTI')

from nipype import Workflow, Node, MapNode
from nipype.interfaces import matlab as mlab
mlab.MatlabCommand.set_default_matlab_cmd("matlab -nodisplay")
# If SPM is not in your MATLAB path you should add it here
# mlab.MatlabCommand.set_default_paths('/software/matlab/spm12')

from nipype.algorithms.rapidart import ArtifactDetect
from nipype.algorithms.misc import TSNR, CalculateMedian
from nipype.interfaces.utility import Rename, Merge, IdentityInterface
from nipype.interfaces.io import DataSink, FreeSurferSource

import numpy as np
import scipy as sp
import nibabel as nb

imports = ['import os', 'import nibabel as nb', 'import numpy as np',
           'import scipy as sp',
           'from nipype.utils.filemanip import filename_to_list, list_to_filename, split_
            →filename',
           'from scipy.special import legendre']

def get_info(dicom_files):
    from dcmstack.extract import default_extractor
    
    """Given a Siemens dicom file return metadata
    Returns
    ----------
    RepetitionTime
    Slice Acquisition Times
    Spacing between slices
    ""
    meta = default_extractor(
        read_file(  
            filename_to_list(dicom_files)[0],
            stop_before_pixels=True,
            force=True)
    return (meta['RepetitionTime'] / 1000., meta['CsaImage.MosaicRefAcqTimes'],
            meta['SpacingBetweenSlices'])

def median(in_files):
    """Computes an average of the median of each realigned timeseries
    Parameters
    ----------
    """
in_files: one or more realigned Nifti 4D time series

Returns
-------
out_file: a 3D Nifti file

""
import numpy as np
import nibabel as nb
from nipype.utils import NUMPY_MMAP
average = None
for idx, filename in enumerate(filename_to_list(in_files)):
    img = nb.load(filename, mmap=NUMPY_MMAP)
    data = np.median(img.get_data(), axis=3)
    if average is None:
        average = data
    else:
        average += data
median_img = nb.Nifti1Image(average / float(idx + 1), img.affine, img.header)
filename = os.path.join(os.getcwd(), 'median.nii.gz')
median_img.to_filename(filename)
return filename

def bandpass_filter(files, lowpass_freq, highpass_freq, fs):
    """Bandpass filter the input files

    Parameters
    ----------
    files: list of 4d nifti files
    lowpass_freq: cutoff frequency for the low pass filter (in Hz)
    highpass_freq: cutoff frequency for the high pass filter (in Hz)
    fs: sampling rate (in Hz)
    """
from nipype.utils.filemanip import split_filename, list_to_filename
import numpy as np
import nibabel as nb
from nipype.utils import NUMPY_MMAP
out_files = []
for filename in filename_to_list(files):
    path, name, ext = split_filename(filename)
    out_file = os.path.join(os.getcwd(), name + '_bp' + ext)
    img = nb.load(filename, mmap=NUMPY_MMAP)
    timepoints = img.shape[-1]
    F = np.zeros((timepoints))
    lowidx = int(timepoints / 2) + 1
    if lowpass_freq > 0:
        lowidx = np.round(lowpass_freq / fs * timepoints)
    highidx = int(timepoints / 2) + 1
    if highpass_freq > 0:
        highidx = np.round(highpass_freq / fs * timepoints)
    F[highidx] = 1
    F = (F + F[::-1]) > 0).astype(int)
    data = img.get_data()
    if np.all(F == 1):
        filtered_data = data
else:
    filtered_data = np.real(np.fft.ifftn(np.fft.fftn(data) * F))
    img_out = nb.Nifti1Image(filtered_data, img.affine, img.header)
    out_files.append(out_file)
return list_to_filename(out_files)

def motion_regressors(motion_params, order=0, derivatives=1):
    """Compute motion regressors up to given order and derivative
    motion + d(motion)/dt + d2(motion)/dt2 (linear + quadratic)"
    import numpy as np
    out_files = []
    for idx, filename in enumerate(filename_to_list(motion_params)):
        params = np.genfromtxt(filename)
        out_params = params
        for d in range(1, derivatives + 1):
            cparams = np.vstack((np.repeat(params[0, :][None, :], d, axis=0),
                                 params))
            out_params = np.hstack((out_params, np.diff(cparams, d, axis=0)))
        out_params2 = out_params
        for i in range(2, order + 1):
            out_params2 = np.hstack((out_params2, np.power(out_params, i)))
        filename = os.path.join(os.getcwd(), "motion_regressor%02d.txt" % idx)
        np.savetxt(filename, out_params2, fmt=b"%.10f")
        out_files.append(filename)
    return out_files

def build_filter1(motion_params, comp_norm, outliers, detrend_poly=None):
    """Builds a regressor set comprising motion parameters, composite norm and outliers
    The outliers are added as a single time point column for each outlier
    """
    import numpy as np
    import nibabel as nb
    from scipy.special import legendre
    out_files = []
    for idx, filename in enumerate(filename_to_list(motion_params)):
        params = np.genfromtxt(filename)
        norm_val = np.genfromtxt(filename_to_list(comp_norm)[idx])
        # ...
```
out_params = np.hstack((params, norm_val[:, None]))
try:
    outlier_val = np.genfromtxt(filename_to_list(outliers)[idx])
except IOError:
    outlier_val = np.empty((0))
for index in np.atleast_1d(outlier_val):
    outlier_vector = np.zeros((out_params.shape[0], 1))
    outlier_vector[index] = 1
    out_params = np.hstack((out_params, outlier_vector))
if detrend_poly:
    timepoints = out_params.shape[0]
    X = np.empty((timepoints, 0))
    for i in range(detrend_poly):
        X = np.hstack((X, legendre(i + 1)(np.linspace(-1, 1, timepoints))[:, None]))
    out_params = np.hstack((out_params, X))
filename = os.path.join(os.getcwd(), "filter_regressor%02d.txt" % idx)
np.savetxt(filename, out_params, fmt=b"%.10f")
out_files.append(filename)
return out_files
```

```python
def extract_noise_components(realigned_file, mask_file, num_components=5, extra_regressors=None):
    """Derive components most reflective of physiological noise
    Parameters
    ----------
    realigned_file: a 4D Nifti file containing realigned volumes
    mask_file: a 3D Nifti file containing white matter + ventricular masks
    num_components: number of components to use for noise decomposition
    extra_regressors: additional regressors to add
    Returns
    -------
    components_file: a text file containing the noise components
    """
    from scipy.linalg.decomp_svd import svd
    import numpy as np
    import nibabel as nb
    from nipype.utils import NUMPY_MMAP
    import os
    imgseries = nb.load(realigned_file, mmap=NUMPY_MMAP)
    components = None
    for filename in filename_to_list(mask_file):
        mask = nb.load(filename, mmap=NUMPY_MMAP).get_data()
        if len(np.nonzero(mask > 0)[0]) == 0:
            continue
        voxel_timecourses = imgseries.get_data()[mask > 0]
        voxel_timecourses[np.isnan(np.sum(voxel_timecourses, axis=1)), :] = 0
        # remove mean and normalize by variance
        X = voxel_timecourses.T
        stdX = np.std(X, axis=0)
        stdX[stdX == 0] = 1.
```
stdX[np.isnan(stdX)] = 1.
stdX[np.isinf(stdX)] = 1.
X = (X - np.mean(X, axis=0)) / stdX
u, _, _ = svd(X, full_matrices=False)
if components is None:
    components = u[:, :num_components]
else:
    components = np.hstack((components, u[:, :num_components]))
if extra_regressors:
    regressors = np.genfromtxt(extra_regressors)
    components = np.hstack((components, regressors))
components_file = os.path.join(os.getcwd(), 'noise_components.txt')
np.savetxt(components_file, components, fmt='%.10f')
return components_file

def rename(in_files, suffix=None):
    from nipype.utils.filemanip import filename_to_list, split_filename, list_to_filename
    out_files = []
    for idx, filename in enumerate(filename_to_list(in_files)):
        _, name, ext = split_filename(filename)
        if suffix is None:
            out_files.append(name + ('_%03d' % idx) + ext)
        else:
            out_files.append(name + suffix + ext)
    return list_to_filename(out_files)

def get_aparc_aseg(files):
    """Return the aparc+aseg.mgz file""
    for name in files:
        if 'aparc+aseg.mgz' in name:
            return name
    raise ValueError('aparc+aseg.mgz not found')

def extract_subrois(timeseries_file, label_file, indices):
    """Extract voxel time courses for each subcortical roi index

Parameters
----------

timeseries_file: a 4D Nifti file
label_file: a 3D file containing rois in the same space/size of the 4D file
indices: a list of indices for ROIs to extract.

Returns
------

out_file: a text file containing time courses for each voxel of each roi

The first four columns are: freesurfer index, i, j, k positions in the
label file
"""
from nipype.utils.filemanip import split_filename
import nibabel as nb
from nipype.utils import NUMPY_MMAP
import os

(continues on next page)
img = nb.load(timeseries_file, mmap=NUMPY_MMAP)
data = img.get_data()
roiimg = nb.load(label_file, mmap=NUMPY_MMAP)
rois = roiimg.get_data()
prefix = split_filename(timeseries_file)[1]
out_ts_file = os.path.join(os.getcwd(), ' %s _subcortical_ts.txt' % prefix)

with open(out_ts_file, 'wt') as fp:
    for fsindex in indices:
        ijk = np.nonzero(rois == fsindex)
        ts = data[ijk]
        for i0, row in enumerate(ts):
            fp.write(' %d, %d, %d, %d,
            for val in row) + ','

return out_ts_file

def combine_hemi(left, right):
    """Combine left and right hemisphere time series into a single text file
    """
    import os
    import numpy as np
    from nipype.utils import NUMPY_MMAP
    lh_data = nb.load(left, mmap=NUMPY_MMAP).get_data()
rh_data = nb.load(right, mmap=NUMPY_MMAP).get_data()

    indices = np.vstack((1000000 + np.arange(0, lh_data.shape[0])[::, None],
                         2000000 + np.arange(0, rh_data.shape[0])[::, None]))
    all_data = np.hstack((indices,
                          np.vstack((lh_data.squeeze(), rh_data.squeeze()))))
    filename = left.split('.')[1] + '_combined.txt'
np.savetxt(filename, all_data, fmt=','.join(['%d'] + ['%.10f'] * (all_data.shape[1] - 1)))
return os.path.abspath(filename)

def create_reg_workflow(name='registration'):
    """Create a FEAT preprocessing workflow together with freesurfer
    Parameters
    ----------

    name : name of workflow (default: 'registration')

    Inputs::

    inputspec.source_files : files (filename or list of filenames to register)
    inputspec.mean_image : reference image to use
    inputspec.anatomical_image : anatomical image to coregister to
    inputspec.target_image : registration target

    Outputs::

    outputspec.func2anat_transform : FLIRT transform

    """
outputs.spec.anat2target_transform : FLIRT+FNIRT transform
outputs.spec.transformed_files : transformed files in target space
outputs.spec.transformed_mean : mean image in target space

""

register = Workflow(name=name)

inputnode = Node(
    interface=IdentityInterface(fields=[
        'source_files', 'mean_image', 'subject_id', 'subjects_dir',
        'target_image'
    ]),
    name='inputspec')

outputnode = Node(
    interface=IdentityInterface(fields=[
        'func2anat_transform', 'out_reg_file', 'anat2target_transform',
        'transforms', 'transformed_mean', 'segmentation_files',
        'anat2target', 'aparc'
    ]),
    name='outputspec')

# Get the subject’s freesurfer source directory
fssource = Node(FreeSurferSource(), name='fssource')
fssource.run_without_submitting = True
register.connect(inputnode, 'subject_id', fssource, 'subject_id')
register.connect(inputnode, 'subjects_dir', fssource, 'subjects_dir')

convert = Node(freesurfer.MRIConvert(out_type='nii'), name="convert")
register.connect(fssource, 'T1', convert, 'in_file')

# Coregister the median to the surface
bbregister = Node(freesurfer.BBRegister(), name='bbregister')
bbregister.inputs.init = 'fsl'
bbregister.inputs.contrast_type = 't2'
bbregister.inputs.out_fsl_file = True
bbregister.inputs.epi_mask = True
register.connect(inputnode, 'subject_id', bbregister, 'subject_id')
register.connect(inputnode, 'mean_image', bbregister, 'source_file')
register.connect(inputnode, 'subjects_dir', bbregister, 'subjects_dir')

""
Estimate the tissue classes from the anatomical image. But use spm's segment as FSL appears to be breaking.
""

stripper = Node(fsl.BET(), name='stripper')
register.connect(convert, 'out_file', stripper, 'in_file')
fast = Node(fsl.FAST(), name='fast')
register.connect(stripper, 'out_file', fast, 'in_files')

""
Binarize the segmentation
""

binarize = MapNode(
    fsl.ImageMaths(op_string='-nan -thr 0.9 -ero -bin'),
    iterfield=['in_file'],
    name='binarize')

(continues on next page)
register.connect(fast, 'partial_volume_files', binarize, 'in_file')

```python
applyxfm = MapNode(
    freesurfer.ApplyVolTransform(inverse=True, interp='nearest'),
    iterfield=['target_file'],
    name='inverse_transform')
```

register.connect(inputnode, 'subjects_dir', applyxfm, 'subjects_dir')
register.connect(bbregister, 'out_reg_file', applyxfm, 'reg_file')
register.connect(binarize, 'out_file', applyxfm, 'target_file')
register.connect(inputnode, 'mean_image', applyxfm, 'source_file')

```python
aparcxfm = Node(
    freesurfer.ApplyVolTransform(inverse=True, interp='nearest'),
    name='aparc_inverse_transform')
```

register.connect(inputnode, 'subjects_dir', aparcxfm, 'subjects_dir')
register.connect(bbregister, 'out_reg_file', aparcxfm, 'reg_file')
register.connect(fssource, ('aparc_aseg', get_aparc_aseg), aparcxfm, 'target_file')
register.connect(inputnode, 'mean_image', aparcxfm, 'source_file')

```python
convert2itk = Node(C3dAffineTool(), name='convert2itk')
convert2itk.inputs.fsl2ras = True
convert2itk.inputs.itk_transform = True
```

register.connect(bbregister, 'out_fsl_file', convert2itk, 'transform_file')
register.connect(inputnode, 'mean_image', convert2itk, 'source_file')
register.connect(stripper, 'out_file', convert2itk, 'reference_file')

```python
reg = Node(ants.Registration(), name='antsRegister')
```

reg.inputs.output_transform_prefix = "output_"
reg.inputs.transforms = ['Rigid', 'Affine', 'SyN']
reg.inputs.transform_parameters = [(0.1, ), (0.1, ), (0.2, 3.0, 0.0)]
reg.inputs.number_of_iterations = [[100, 30, 20]]
reg.inputs.dimension = 3
reg.inputs.write_composite_transform = True
reg.inputs.collapse_output_transforms = True
reg.inputs.initial_moving_transform_com = True
reg.inputs.metric = ['Mattes'] * 2 + [['Mattes', 'CC']]
reg.inputs.metric_weight = [1] * 2 + [[0.5, 0.5]]

(continues on next page)
reg.inputs.radius_or_number_of_bins = [32] * 2 + [[32, 4]]
reg.inputs.sampling_strategy = ['Regular'] * 2 + [[None, None]]
reg.inputs.sampling_percentage = [0.3] * 2 + [[None, None]]
reg.inputs.convergence_threshold = [1.e-8] * 2 + [-0.01]
reg.inputs.smoothing_sigmas = [[4, 2, 1]] * 2 + [[1, 0.5, 0]]
reg.inputs.sigma_units = ['vox'] * 3
reg.inputs.shrink_factors = [[3, 2, 1]] * 2 + [[4, 2, 1]]
reg.inputs.use_estimate_learning_rate_once = [True] * 3
reg.inputs.use_histogram_matching = [False] * 2 + [True]
reg.inputs.winsorize_lower_quantile = 0.005
reg.inputs.winsorize_upper_quantile = 0.995
reg.inputs.float = True
reg.inputs.output_warped_image = 'output_warped_image.nii.gz'
reg.inputs.num_threads = 4
reg.plugin_args = {'qsub_args': '-l nodes=1:ppn=4'}
register.connect(stripper, 'out_file', reg, 'moving_image')
register.connect(inputnode, 'target_image', reg, 'fixed_image')

"""
Concatenate the affine and ants transforms into a list
"""
merge = Node(ols(2), iterfield=['in2'], name='mergexfm')
register.connect(convert2itk, 'itk_transform', merge, 'in2')
register.connect(reg, 'composite_transform', merge, 'in1')

"""
Transform the mean image. First to anatomical and then to target
"""
warpmean = Node(ants.ApplyTransforms(), name='warpmean')
warpmean.inputs.input_image_type = 3
warpmean.inputs.interpolation = 'Linear'
warpmean.inputs.invert_transform_flags = [False, False]
warpmean.terminal_output = 'file'
warpmean.inputs.args = '--float'
warpmean.inputs.num_threads = 4
register.connect(inputnode, 'target_image', warpmean, 'reference_image')
register.connect(inputnode, 'mean_image', warpmean, 'input_image')
register.connect(merge, 'out', warpmean, 'transforms')

"""
Assign all the output files
"""
register.connect(reg, 'warped_image', outputnode, 'anat2target')
register.connect(warpmean, 'output_image', outputnode, 'transformed_mean')
register.connect(applyxfm, 'transformed_file', outputnode, 'segmentation_files')
register.connect(aparcxfm, 'transformed_file', outputnode, 'aparc')
register.connect(bbregister, 'out_fsl_file', outputnode, 'func2anat_transform')
register.connect(bbregister, 'out_reg_file', outputnode, 'out_reg_file')
register.connect(reg, 'composite_transform', outputnode, 'anat2target_transform')
register.connect(merge, 'out', outputnode, 'transforms')

return register
Creates the main preprocessing workflow

def create_workflow(files, target_file, subject_id, TR, slice_times, norm_threshold=1, num_components=5, vol_fwhm=None, surf_fwhm=None, lowpass_freq=-1, highpass_freq=-1, subjects_dir=None, sink_directory=os.getcwd(), target_subject=['fsaverage3', 'fsaverage4'], name='resting'):

    wf = Workflow(name=name)
    # Rename files in case they are named identically
    name_unique = MapNode(
        Rename(format_string='rest_%(run)02d'),
        iterfield=['in_file', 'run'],
        name='rename')
    name_unique.inputs.keep_ext = True
    name_unique.inputs.run = list(range(1, len(files) + 1))
    name_unique.inputs.in_file = files

    realign = Node(interface=spm.Realign(), name="realign")
    realign.inputs.jobtype = 'estwrite'

    num_slices = len(slice_times)
    slice_timing = Node(interface=spm.SliceTiming(), name="slice_timing")
    slice_timing.inputs.num_slices = num_slices
    slice_timing.inputs.time_repetition = TR
    slice_timing.inputs.time_acquisition = TR - TR / float(num_slices)
    slice_timing.inputs.slice_order = (np.argsort(slice_times) + 1).tolist()
    slice_timing.inputs.ref_slice = int(num_slices / 2)
    slice_timing.inputs.timecorrected_files = slice_timing.inputs.timecorrected_files

    tsnr = MapNode(TSNR(regress_poly=2), iterfield=['in_file'], name='tsnr')
    wf.connect(slice_timing, 'timecorrected_files', tsnr, 'in_file')

    # Compute TSNR on realigned data regressing polynomials up to order 2
    tsnr = MapNode(TSNR(regress_poly=2), iterfield=['in_file'], name='tsnr')
    wf.connect(slice_timing, 'timecorrected_files', tsnr, 'in_file')

    # Compute the median image across runs
    calc_median = Node(CalculateMedian(), name='median')
    wf.connect(tsnr, 'detrended_file', calc_median, 'in_files')

    registration = create_reg_workflow(name='registration')
    wf.connect(calc_median, 'median_file', registration, inputspec.mean_image)

    registration.inputs.inputspec.subject_id = subject_id
    registration.inputs.inputspec.subjects_dir = subjects_dir
    registration.inputs.inputspec.target_image = target_file

    registration.inputs.inputspec.target_subject = target_subject
    registration.inputs.inputspec.name = name

(continues on next page)
intensity or movement.
"

art = Node(interface=ArtifactDetect(), name="art")
art.inputs.use_differences = [True, True]
art.inputs.use_norm = True
art.inputs.norm_threshold = norm_threshold
art.inputs.zintensity_threshold = 9
art.inputs.mask_type = 'spm_global'
art.inputs.parameter_source = 'SPM'
"

"""Here we are connecting all the nodes together. Notice that we add the merge node only if you choose to use 4D. Also 'get_vox_dims' function is passed along the input volume of normalise to set the optimal voxel sizes.
"

wf.connect([
    (name_unique, realign, [('out_file', 'in_files')]),
    (realign, slice_timing, [('realigned_files', 'in_files')]),
    (slice_timing, art, [('timecorrected_files', 'realigned_files')]),
    (realign, art, [('realignment_parameters', 'realignment_parameters')])],

def selectindex(files, idx):
    import numpy as np
    from nipype.utils.filemanip import filename_to_list, list_to_filename
    return list_to_filename(np.array(filename_to_list(files))[idx].tolist())

mask = Node(fsl.BET(), name='getmask')
mask.inputs.mask = True
wf.connect(calc_median, 'median_file', mask, 'in_file')
# get segmentation in normalized functional space

def merge_files(in1, in2):
    out_files = filename_to_list(in1)
    out_files.extend(filename_to_list(in2))
    return out_files
# filter some noise
# Compute motion regressors
motreg = Node(
    Function(
        input_names=['motion_params', 'order', 'derivatives'],
        output_names=['out_files'],
        function=motion_regressors,
        imports=imports),
    name='getmotionregress')
wf.connect(realign, 'realignment_parameters', motreg, 'motion_params')
# Create a filter to remove motion and art confounds
createfilter1 = Node(
    Function(
        input_names=[]
'motion_params', 'comp_norm', 'outliers', 'detrend_poly'
],
output_names=['out_files'],
function=build_filter1,
imports=imports),
name='makemotionbasedfilter')
createfilter1.inputs.detrend_poly = 2
wf.connect(motreg, 'out_files', createfilter1, 'motion_params')
wf.connect(art, 'norm_files', createfilter1, 'comp_norm')
wf.connect(art, 'outlier_files', createfilter1, 'outliers')

filter1 = MapNode(fsl.GLM(
    out_f_name='F_mcart.nii', out_pf_name='pF_mcart.nii', demean=True),
    iterfield=['in_file', 'design', 'out_res_name'],
    name='filtermotion')
wf.connect(slice_timing, 'timecorrected_files', filter1, 'in_file')
wf.connect(slice_timing, ('timecorrected_files', rename, '_filtermotart'),
    filter1, 'out_res_name')
wf.connect(createfilter1, 'out_files', filter1, 'design')

createfilter2 = MapNode(
    Function(
        input_names=['realigned_file', 'mask_file', 'num_components',
                     'extra_regressors'],
        output_names=['out_files'],
        function=extract_noise_components,
        imports=imports),
    iterfield=['realigned_file', 'extra_regressors'],
    name='makecompcorrfilter')
createfilter2.inputs.num_components = num_components
wf.connect(createfilter1, 'out_files', createfilter2, 'realigned_file')
wf.connect(filter1, 'out_res', createfilter2, 'realigned_file')
wf.connect(registration,
    ('outputspec.segmentation_files', selectindex, [0, 2]),
    createfilter2, 'mask_file')

filter2 = MapNode(fsl.GLM(out_f_name='F.nii', out_pf_name='pF.nii', demean=True),
    iterfield=['in_file', 'design', 'out_res_name'],
    name='filter_noise_nosmooth')
wf.connect(filter1, 'out_res', filter2, 'in_file')
wf.connect(filter1, ('out_res', rename, '_cleaned'), filter2,
    'out_res_name')
wf.connect(createfilter2, 'out_files', filter2, 'design')
wf.connect(mask, 'mask_file', filter2, 'mask')

bandpass = Node(
    Function(
        input_names=['files', 'lowpass_freq', 'highpass_freq', 'fs'],
        output_names=['out_files'],
        function=bandpass_filter,
        imports=imports),
(continues on previous page)
name='bandpass_unsmooth')
bandpass.inputs.fs = 1. / TR
bandpass.inputs.highpass_freq = highpass_freq
bandpass.inputs.lowpass_freq = lowpass_freq
wf.connect(filter2, 'out_res', bandpass, 'files')

"""Smooth the functional data using :
class:`nipype.interfaces.spm.Smooth`.
"""

smooth = Node(interface=spm.Smooth(), name="smooth")
smooth.inputs.fwhm = vol_fwhm
wf.connect(bandpass, 'out_files', smooth, 'in_files')

collector = Node(Merge(2), name='collect_streams')
wf.connect(smooth, 'smoothed_files', collector, 'in1')
wf.connect(bandpass, 'out_files', collector, 'in2')

"""Transform the remaining images. First to anatomical and then to target
"""

warpall = MapNode(
    ants.ApplyTransforms(), iterfield=['input_image'], name='warpall')
warpall.inputs.input_image_type = 3
warpall.inputs.interpolation = 'Linear'
warpall.inputs.invert_transform_flags = [False, False]
warpall.terminal_output = 'file'
warpall.inputs.reference_image = target_file
warpall.inputs.args = '--float'
warpall.inputs.num_threads = 1

# transform to target
wf.connect(collector, 'out', warpall, 'input_image')
wf.connect(registration, 'outputspec.transforms', warpall, 'transforms')

mask_target = Node(fsl.ImageMaths(op_string='--bin'), name='target_mask')
wf.connect(registration, 'outputspec.anat2target', mask_target, 'in_file')
maskts = MapNode(fsl.ApplyMask(), iterfield=['in_file'], name='ts_masker')
wf.connect(warpall, 'output_image', maskts, 'in_file')
wf.connect(mask_target, 'out_file', maskts, 'mask_file')

# map to surface
# extract aparc+aseg ROIs
# extract subcortical ROIs
# extract target space ROIs
# combine subcortical and cortical rois into a single cifti file

#######
# Convert aparc to subject functional space

# Sample the average time series in aparc ROIs
sampleaparc = MapNode(
    freesurfer.SegStats(default_color_table=True),
    iterfield=['in_file', 'summary_file', 'avgwf_txt_file'],
    name='aparc_ts')
sampleaparc.inputs.segment_id = (
    [8] + list(range(10, 14)) + [17, 18, 26, 47] + list(range(49, 55)) + 
    [58] + list(range(1001, 1036)) + list(range(2001, 2036)))

wf.connect(registration, 'outputspec.aparc', sampleaparc,
            'segmentation_file')
wf.connect(collector, 'out', sampleaparc, 'in_file')

def get_names(files, suffix):
    """Generate appropriate names for output files
    """
    from nipype.utils.filemanip import split_filename, filename_to_list,
                                        list_to_filename
    out_names = []
    for filename in files:
        _, name, _ = split_filename(filename)
        out_names.append(name + suffix)
    return list_to_filename(out_names)

wf.connect(collector, ('out', get_names, '_avgwf.txt'), sampleaparc,
            'avgwf_txt_file')
wf.connect(collector, ('out', get_names, '_summary.stats'), sampleaparc,
            'summary_file')

# Sample the time series onto the surface of the target surface. Performs
# sampling into left and right hemisphere
target = Node(IdentityInterface(fields=['target_subject']), name='target')
target.iterables = ('target_subject', filename_to_list(target_subject))
samplerlh = MapNode(freesurfer.SampleToSurface(),
                    iterfield=['source_file'],
                    name='sampler_lh')
samplerlh.inputs.sampling_method = "average"
samplerlh.inputs.sampling_range = (0.1, 0.9, 0.1)
samplerlh.inputs.sampling_units = "frac"
samplerlh.inputs.interp_method = "trilinear"
samplerlh.inputs.smooth_surf = surf_fwhm
# samplerlh.inputs.cortex_mask = True
samplerlh.inputs.out_type = 'niigz'
samplerlh.inputs.subjects_dir = subjects_dir
samplerrh = samplerlh.clone('sampler_rh')
samplerlh.inputs.hemi = 'lh'
fw.connect(collector, 'out', samplerlh, 'source_file')
wf.connect(registration, 'outputspec.out_reg_file', samplerlh, 'reg_file')
wf.connect(target, 'target_subject', samplerlh, 'target_subject')
samplerrh.set_input('hemi', 'rh')
fw.connect(collector, 'out', samplerrh, 'source_file')
wf.connect(registration, 'outputspec.out_reg_file', samplerrh, 'reg_file')
wf.connect(target, 'target_subject', samplerrh, 'target_subject')

# Combine left and right hemisphere to text file
combiner = MapNode(Function(
    (continues on next page)
input_names=['left', 'right'],
output_names=['out_file'],
function=combine_hemi,
imports=imports),
iterfield=['left', 'right'],
name="combiner")
wf.connect(samplerlh, 'out_file', combiner, 'left')
wf.connect(samplerrh, 'out_file', combiner, 'right')

# Sample the time series file for each subcortical roi
ts2txt = MapNode(
    Function(
        input_names=['timeseries_file', 'label_file', 'indices'],
        output_names=['out_file'],
        function=extract_subrois,
        imports=imports),
    iterfield=['timeseries_file'],
    name='getsubcortts')
ts2txt.inputs.indices = [8] + list(range(10, 14)) + [17, 18, 26, 47] +
    list(range(49, 55)) + [58]
ts2txt.inputs.label_file = \
    os.path.abspath(('OASIS-TRT-20_jointfusion_DKT31_CMA_labels_in_MNI152_'\
    '2mm_v2.nii.gz'))
wf.connect(maskts, 'out_file', ts2txt, 'timeseries_file')

substitutions = 
    [('_target_subject_', ''), ('_filtermotart_cleaned_bp_trans_masked', ''),
     ('_filtermotart_cleaned_bp', '')]
regex_subs = [
    ('_ts_masker.*/sar', '/smooth/'),
    ('_ts_masker.*/ar', '/unsmooth/'),
    ('_combiner.*/sar', '/smooth/'),
    ('_combiner.*/ar', '/unsmooth/'),
    ('_aparc_ts.*/sar', '/smooth/'),
    ('_aparc_ts.*/ar', '/unsmooth/'),
    ('_getsubcortts.*/sar', '/smooth/'),
    ('_getsubcortts.*/ar', '/unsmooth/'),
    ('series/sar', 'series/smooth/'),
    ('series/ar', 'series/unsmooth/'),
    ('_inverse_transform./', '')]

# Save the relevant data into an output directory
datasink = Node(interface=DataSink(), name="datasink")
datasink.inputs.base_directory = sink_directory
datasink.inputs.container = subject_id
datasink.inputs.substitutions = substitutions
datasink.inputs.regexp_substitutions = regex_subs
wf.connect(realign, 'realignment_parameters', datasink, 'resting.qa.motion')
wf.connect(art, 'norm_files', datasink, 'resting.qa.art.@norm')
wf.connect(art, 'intensity_files', datasink, 'resting.qa.art.@intensity')
wf.connect(art, 'outlier_files', datasink, 'resting.qa.art.@outlier_files')
wf.connect(registration, 'outputspec.segmentation_files', datasink, 'resting.mask_files')
wf.connect(registration, 'outputspec.anat2target', datasink, 'resting.qa.ants')
wf.connect(mask, 'mask_file', datasink, 'resting.mask_files.@brainmask')
wf.connect(mask_target, 'out_file', datasink, 'resting.mask_files.target')
wf.connect(filter1, 'out_f', datasink, 'resting.qa.compmaps.@mc_F')
wf.connect(filter1, 'out_pf', datasink, 'resting.qa.compmaps.@mc_pF')
wf.connect(filter2, 'out_f', datasink, 'resting.qa.compmaps')
wf.connect(filter2, 'out_pf', datasink, 'resting.qa.compmaps.@p')
wf.connect(bandpass, 'out_files', datasink, 'resting.timeseries.@bandpassed')
wf.connect(smooth, 'smoothed_files', datasink, 'resting.timeseries.@smoothed')
wf.connect(createfilter1, 'out_files', datasink, 'resting.regress.@regressors')
wf.connect(createfilter2, 'out_files', datasink, 'resting.regress.@compcorr')
wf.connect(maskts, 'out_file', datasink, 'resting.timeseries.target')
wf.connect(sampleaparc, 'summary_file', datasink, 'resting.parcellations.aparc')
wf.connect(sampleaparc, 'avgwf_txt_file', datasink, 'resting.parcellations.aparc.@avgwf')
wf.connect(ts2txt, 'out_file', datasink, 'resting.parcellations.grayo.@subcortical')

datasink2 = Node(interface=DataSink(), name="datasink2")
datasink2.inputs.base_directory = sink_directory
datasink2.inputs.container = subject_id
datasink2.inputs.substitutions = substitutions
datasink2.inputs.regexp_substitutions = regex_subs
wf.connect(combiner, 'out_file', datasink2, 'resting.parcellations.grayo.@surface')
return wf

Creates the full workflow including getting information from dicom files

def create_resting_workflow(args, name=None):
    TR = args.TR
    slice_times = args.slice_times
    if args.dicom_file:
        TR, slice_times, slice_thickness = get_info(args.dicom_file)
        slice_times = (np.array(slice_times) / 1000.).tolist()
    if name is None:
        name = 'resting_' + args.subject_id
    kwargs = dict(
        files=[os.path.abspath(filename) for filename in args.files],
        target_file=os.path.abspath(args.target_file),
        subject_id=args.subject_id,
        TR=TR,
        slice_times=slice_times,
        vol_fwhm=args.vol_fwhm,
        surf_fwhm=args.surf_fwhm,
        norm_threshold=2.,
        subjects_dir=os.path.abspath(args.fsdir),
        target_subject=args.target_surfs,
        lowpass_freq=args.lowpass_freq,
        highpass_freq=args.highpass_freq,
sink_directory=os.path.abspath(args.sink),
name=name)
wf = create_workflow(**kwargs)
return wf

if __name__ == '__main__':
    from argparse import ArgumentParser, RawTextHelpFormatter
defstr = ' (default %(default)s)
parser = ArgumentParser(
    description=__doc__, formatter_class=RawTextHelpFormatter)
parser.add_argument(
    '-d',
    '--dicom_file',
dest="dicom_file",
    help="an example dicom file from the resting series")
parser.add_argument(
    '-f',
    '--files',
dest="files",
nargs="*",
    help="4d nifti files for resting state",
    required=True)
parser.add_argument(
    '-t',
    '--target',
dest="target_file",
    help="Target in MNI space. Best to use the MindBoggle "
    "template - "
    "OASIS-30_Atropos_template_in_MNI152_2mm.nii.gz"),
    required=True)
parsed.add_argument(
    '-s',
    '--subject_id',
dest="subject_id",
    help="FreeSurfer subject id",
    required=True)
parsed.add_argument(
    '--subcom_direc',
    dest="fsdir",
    help="FreeSurfer subject directory",
    required=True)
parsed.add_argument(
    '--target_surfaces',
dest="target_surfs",
nargs="*",
default=["fsaverage5"],
    help="FreeSurfer target surfs" + defstr)
parsed.add_argument(
    '--TR',
dest="TR",
default=None,
type=float,
    help="TR if dicom not provided in seconds")
parsed.add_argument(
    '--slice_times',
dest="slice_times",}
nargs="+",  
type=float,  
help="Slice onset times in seconds")
parser.add_argument(  
'--vol_fwhm',  
default=6.,  
dest='vol_fwhm',  
type=float,  
help="Spatial FWHM" + defstr)
parser.add_argument(  
'--surf_fwhm',  
default=15.,  
dest='surf_fwhm',  
type=float,  
help="Spatial FWHM" + defstr)
parser.add_argument(  
'-l',  
'--lowpass_freq',  
default=0.1,  
dest='lowpass_freq',  
type=float,  
help="Low pass frequency (Hz)" + defstr)
parser.add_argument(  
'-u',  
'--highpass_freq',  
default=0.01,  
dest='highpass_freq',  
type=float,  
help="High pass frequency (Hz)" + defstr)
parser.add_argument(  
'-o',  
'--output_dir',  
default='sink',  
dest='sink',  
help="Output directory base",  
required=True)
parser.add_argument(  
'-w',  
'--work_dir',  
help="Output directory base")
parser.add_argument(  
'-p',  
'--plugin',  
default='Linear',  
dest='plugin',  
help="Plugin to use")
parser.add_argument(  
'--plugin_args',  
help="Plugin arguments")
args = parser.parse_args()
wf = create_resting_workflow(args)

if
    work_dir = os.path.abspath(args.work_dir)
else:
    work_dir = os.getcwd()
wf.base_dir = work_dir
if
    wf.run(args.plugin, plugin_args=eval(args.plugin_args))

(continues on next page)
```python
else:
    wf.run(args.plugin)
```

### Example source code

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the `examples` directory.
A preprocessing workflow for Siemens resting state data. This workflow makes use of:

- ANTS
- FreeSurfer
- FSL
- NiPy
- aCompCor

For example:

```
python rsfmri_preprocessing.py -d /data/12345-34-1.dcm -f /data/Resting.nii
-s subj001 -o output -p PBS --plugin_args "dict(qsub_args=''-q many'')"
```

or:

```
python rsfmri_vol_surface_preprocessing.py -f SUB_1024011/E?/func/rest.nii
-t OASIS-30_Atrropos_template_in_MNI152_2mm.nii.gz --TR 2 -s SUB_1024011
--subjects_dir fsdata --slice_times 0 17 1 18 2 19 3 20 4 21 5 22 6 23
7 24 8 25 9 26 10 27 11 28 12 29 13 30 14 31 15 32 16 -o .
```

This workflow takes resting timeseries and a Siemens dicom file corresponding to it and preprocesses it to produce timeseries coordinates or grayordinates. For non-Siemens dicoms, provide slice times instead, since the dicom extractor is not guaranteed to work. This workflow also requires 2mm subcortical atlas and templates that are available from:

http://mindboggle.info/data.html

specifically the 2mm versions of:

- Joint Fusion Atlas
- MNI template

Import necessary modules from nipype.

```
from __future__ import division, unicode_literals
from builtins import open, range, str
import os

from nipype.interfaces.base import CommandLine
CommandLine.set_default_terminal_output('allatonce')
```

(continues on next page)
# https://github.com/moloney/dcmstack
from dcmstack.extract import default_extractor
# pip install pydicom
from dicom import read_file
from nipype.interfaces import (fsl, Function, ants, freesurfer, nipy)
from nipype.interfaces.c3 import C3dAffineTool
fsl.FSLCommand.set_default_output_type('NIFTI_GZ')
from nipype import Workflow, Node, MapNode
from nipype.algorithms.rapidart import ArtifactDetect
from nipipe.algorithms.misc import TSNR, CalculateMedian
from nipipe.algorithms.confounds import ACompCor
from nipipe.interfaces.utility import Rename, Merge, IdentityInterface
from nipipe.utils.filemanip import filename_to_list
from nipipe.interfaces.io import DataSink, FreeSurferSource
import nipype.interfaces.freesurfer as fs
import numpy as np
import scipy as sp
import nibabel as nb
from nipipe.utils.config import NUMPY_MMAP

A list of modules and functions to import inside of nodes

```python
imports = [
    'import os',
    'import nibabel as nb',
    'import numpy as np',
    'import scipy as sp',
    'from nipype.utils.filemanip import filename_to_list, list_to_filename, split_
→filename',
    'from scipy.special import legendre'
]
```

Define utility functions for use in workflow nodes

```python
def get_info(dicom_files):
    """Given a Siemens dicom file return metadata
    ""
    Returns
    --------
    RepetitionTime
    Slice Acquisition Times
    Spacing between slices
    """
    meta = default_extractor(
        read_file(  
            filename_to_list(dicom_files)[0],  
            stop_before_pixels=True,  
            force=True)
    )
    return (meta['RepetitionTime'] / 1000., meta['CsaImage.MosaicRefAcqTimes'],
            meta['SpacingBetweenSlices'])

def median(in_files):
    (continues on next page)
Computes an average of the median of each realigned timeseries

Parameters
-----------

in_files: one or more realigned Nifti 4D time series

Returns
-------

out_file: a 3D Nifti file

```python
average = None
for idx, filename in enumerate(filename_to_list(in_files)):
    img = nb.load(filename, mmap=NUMPY_MMAP)
    data = np.median(img.get_data(), axis=3)
    if average is None:
        average = data
    else:
        average = average + data
median_img = nb.Nifti1Image(average / float(idx + 1), img.affine, img.header)
filename = os.path.join(os.getcwd(), 'median.nii.gz')
median_img.to_filename(filename)
return filename
```

def bandpass_filter(files, lowpass_freq, highpass_freq, fs):
    """Bandpass filter the input files""

    Parameters
               --------
    files: list of 4d nifti files
    lowpass_freq: cutoff frequency for the low pass filter (in Hz)
    highpass_freq: cutoff frequency for the high pass filter (in Hz)
    fs: sampling rate (in Hz)

    out_files = []
    for filename in filename_to_list(files):
        path, name, ext = split_filename(filename)
        out_file = os.path.join(os.getcwd(), name + '_bp' + ext)
        img = nb.load(filename, mmap=NUMPY_MMAP)
        timepoints = img.shape[-1]
        F = np.zeros((timepoints))
        lowidx = int(timepoints / 2) + 1
        if lowpass_freq > 0:
            lowidx = np.round(float(lowpass_freq) / fs * timepoints)
        highidx = 0
        if highpass_freq > 0:
            highidx = np.round(float(highpass_freq) / fs * timepoints)
        F[highidx:lowidx] = 1
        F = (F + F[::-1]) > 0).astype(int)
        data = img.get_data()
        if np.all(F == 1):
            filtered_data = data
        else:
            filtered_data = np.real(np.fft.ifftn(np.fft.fftn(data) * F))
```

(continues on next page)
```
img_out = nb.Nifti1Image(filtered_data, img.affine, img.header)
img_out.to_filename(out_file)
out_files.append(out_file)
return list_to_filename(out_files)

def motion_regressors(motion_params, order=0, derivatives=1):
    """Compute motion regressors upto given order and derivative
    motion + d(motion)/dt + d2(motion)/dt2 (linear + quadratic)
    ""
    out_files = []
    for idx, filename in enumerate(filename_to_list(motion_params)):
        params = np.genfromtxt(filename)
        out_params = params
        for d in range(1, derivatives + 1):
            cparams = np.vstack((np.repeat(params[0, :], d, axis=0), params))
            out_params = np.hstack((out_params, np.diff(cparams, d, axis=0)))
        out_params2 = out_params
        for i in range(2, order + 1):
            out_params2 = np.hstack((out_params2, np.power(out_params, i)))
        filename = os.path.join(os.getcwd(), "motion_regressor%02d.txt" % idx)
        np.savetxt(filename, out_params2, fmt="%.10f")
        out_files.append(filename)
    return out_files

def build_filter1(motion_params, comp_norm, outliers, detrend_poly=None):
    """Builds a regressor set comprisong motion parameters, composite norm and outliers
    The outliers are added as a single time point column for each outlier
    Parameters
    ----------
    motion_params: a text file containing motion parameters and its derivatives
    comp_norm: a text file containing the composite norm
    outliers: a text file containing 0-based outlier indices
    detrend_poly: number of polynomials to add to detrend
    Returns
    -------
    components_file: a text file containing all the regressors
    ""
    out_files = []
    for idx, filename in enumerate(filename_to_list(motion_params)):
        params = np.genfromtxt(filename)
        norm_val = np.genfromtxt(filename_to_list(comp_norm)[idx])
        out_params = np.hstack((params, norm_val[:, None]))
        try:
            outlier_val = np.genfromtxt(filename_to_list(outliers)[idx])
        except IOError:
            outlier_val = np.empty((0))
        for index in np.atleast_1d(outlier_val):
```
outlier_vector = np.zeros((out_params.shape[0], 1))
outlier_vector[index] = 1
out_params = np.hstack((out_params, outlier_vector))
if detrend_poly:
    timepoints = out_params.shape[0]
    X = np.empty((timepoints, 0))
    for i in range(detrend_poly):
        X = np.hstack((X, legendre(i + 1)(np.linspace(-1, 1, timepoints))[:, None]))
    out_params = np.hstack((out_params, X))
filename = os.path.join(os.getcwd(), "filter_regressor%02d.txt" % idx)
np.savetxt(filename, out_params, fmt="%.10f")
out_files.append(filename)
return out_files

def rename(in_files, suffix=None):
    from nipype.utils.filemanip import filename_to_list, split_filename, list_to_filename
    out_files = []
    for idx, filename in enumerate(filename_to_list(in_files)):
        _, name, ext = split_filename(filename)
        if suffix is None:
            out_files.append(name + ('_%03d' % idx) + ext)
        else:
            out_files.append(name + suffix + ext)
    return list_to_filename(out_files)

def get_aparc_aseg(files):
    """Return the aparc+aseg.mgz file""
    for name in files:
        if 'aparc+aseg.mgz' in name:
            return name
    raise ValueError('aparc+aseg.mgz not found')

def extract_subrois(timeseries_file, label_file, indices):
    """Extract voxel time courses for each subcortical roi index

    Parameters
    ----------

    timeseries_file: a 4D Nifti file
    label_file: a 3D file containing rois in the same space/size of the 4D file
    indices: a list of indices for ROIs to extract.

    Returns
    -------

    out_file: a text file containing time courses for each voxel of each roi
              The first four columns are: freesurfer index, i, j, k positions in the
              label file
    """
    img = nb.load(timeseries_file, mmap=NUMPY_MMAP)
data = img.get_data()
roiimg = nb.load(label_file, mmap=NUMPY_MMAP)
rois = roiimg.get_data()
prefix = split_filename(timeseries_file)[1]
out_ts_file = os.path.join(os.getcwd(), '%s_subcortical_ts.txt' % prefix)

with open(out_ts_file, 'wt') as fp:
    for fsindex in indices:
        ijk = np.nonzero(rois == fsindex)
        ts = data[ijk]
        for i0, row in enumerate(ts):
            fp.write('%d, %d, %d, %d,
                      %s
                    ' % (fsindex, ijk[0][i0], ijk[1][i0],
                          ijk[2][i0]) + ','.join(["%.10f" % val
                          for val in row]) + '

return out_ts_file

def combine_hemi(left, right):
    """Combine left and right hemisphere time series into a single text file""
    lh_data = nb.load(left, mmap=NUMPY_MMAP).get_data()
    rh_data = nb.load(right, mmap=NUMPY_MMAP).get_data()

    indices = np.vstack((1000000 + np.arange(0, lh_data.shape[0])[:,
                          None],
                        2000000 + np.arange(0, rh_data.shape[0])[:,
                          None]))
    all_data = np.hstack((indices,
                          np.vstack((lh_data.squeeze(), rh_data.squeeze()))))

    filename = left.split('.')[1] + '_combined.txt'
    np.savetxt(filename,
               all_data,
               fmt=',' + ('%.10f' * (all_data.shape[1] - 1)))
    return os.path.abspath(filename)

Create a Registration Workflow

def create_reg_workflow(name='registration'):
    """Create a FEAT preprocessing workflow together with freesurfer"

    Parameters
              ---------
        name : name of workflow (default: 'registration')

    Inputs:

        inputspec.source_files : files (filename or list of filenames to register)
        inputspec.mean_image : reference image to use
        inputspec.anatomical_image : anatomical image to coregister to
        inputspec.target_image : registration target

    Outputs:

        outputspec.func2anat_transform : FLIRT transform
        outputspec.anat2target_transform : FLIRT+FNIRT transform
        outputspec.transformed_files : transformed files in target space
        outputspec.transformed_mean : mean image in target space

    Example
            --------
        See code below
register = Workflow(name=name)

inputnode = Node(
    interface=IdentityInterface(fields=[
        'source_files', 'mean_image', 'subject_id', 'subjects_dir',
        'target_image'
    ]),
    name='inputspec')

outputnode = Node(
    interface=IdentityInterface(fields=[
        'func2anat_transform', 'out_reg_file', 'anat2target_transform',
        'transforms', 'transformed_mean', 'segmentation_files',
        'anat2target', 'aparc', 'min_cost_file'
    ]),
    name='outputspec')

# Get the subject's freesurfer source directory
fssource = Node(FreeSurferSource(), name='fssource')
fssource.run_withoutSubmitting = True
register.connect(inputnode, 'subject_id', fssource, 'subject_id')
register.connect(inputnode, 'subjects_dir', fssource, 'subjects_dir')

convert = Node(freesurfer.MRIConvert(out_type='nii'), name='convert')
register.connect(fssource, 'T1', convert, 'in_file')

# Coregister the median to the surface
bbregister = Node(freesurfer.BBRegister(), name='bbregister')
bbregister.inputs.init = 'fsl'
bbregister.inputs.contrast_type = 't2'
bbregister.inputs.out_fsl_file = True
bbregister.inputs.epi_mask = True
register.connect(inputnode, 'subject_id', bbregister, 'subject_id')
register.connect(inputnode, 'mean_image', bbregister, 'source_file')
register.connect(inputnode, 'subjects_dir', bbregister, 'subjects_dir')

```python
Estimate the tissue classes from the anatomical image. But use aparc+aseg's brain mask
```
binarize = MapNode(
    fsl.ImageMaths(op_string='nan -thr 0.9 -ero -bin'),
    iterfield=["in_file"],
    name='binarize')
register.connect(fast, 'partial_volume_files', binarize, 'in_file')

""" Apply inverse transform to take segmentations to functional space """

applyxfm = MapNode(
    freesurfer.ApplyVolTransform(inverse=True, interp='nearest'),
    iterfield=['target_file'],
    name='inverse_transform')
register.connect(inputnode, 'subjects_dir', applyxfm, 'subjects_dir')
register.connect(bbregister, 'out_reg_file', applyxfm, 'reg_file')
register.connect(binarize, 'out_file', applyxfm, 'target_file')
register.connect(inputnode, 'mean_image', applyxfm, 'source_file')

""" Apply inverse transform to aparc file """

aparcxfm = Node(
    freesurfer.ApplyVolTransform(inverse=True, interp='nearest'),
    name='aparc_inverse_transform')
register.connect(inputnode, 'subjects_dir', aparcxfm, 'subjects_dir')
register.connect(bbregister, 'out_reg_file', aparcxfm, 'reg_file')
register.connect(fssource, ('aparc_aseg', get_aparc_aseg), aparcxfm, 'target_file')
register.connect(inputnode, 'mean_image', aparcxfm, 'source_file')

""" Convert the BBRegister transformation to ANTS ITK format """

convert2itk = Node(C3dAffineTool(), name='convert2itk')
convert2itk.inputs.fsl2ras = True
convert2itk.inputs.itk_transform = True
register.connect(bbregister, 'out_fsl_file', convert2itk, 'transform_file')
register.connect(inputnode, 'mean_image', convert2itk, 'source_file')
register.connect(stripper, 'out_file', convert2itk, 'reference_file')

""" Compute registration between the subject's structural and MNI template """

* All parameters are set using the example from:
  #https://github.com/stnava/ANTS/blob/master/Scripts/newAntsExample.sh
* This is currently set to perform a very quick registration. However, the
  registration can be made significantly more accurate for cortical
  structures by increasing the number of iterations.

reg = Node(ants.Registration(), name='antsRegister')
reg.inputs.output_transform_prefix = "output_"
reg.inputs.transforms = ['Rigid', 'Affine', 'SyN']
reg.inputs.transform_parameters = [[0.1, ], [0.1, ], [0.2, 3.0, 0.0]]
reg.inputs.number_of_iterations = [[10000, 11110, 11110]] * 2 + [[
  100, 30, 20]]
reg.inputs.dimension = 3
reg.inputs.write_composite_transform = True
reg.inputs.collapse_output_transforms = True
reg.inputs.initial_moving_transform_com = True
reg.inputs.metric = ['Mattes'] * 2 + [['Mattes', 'CC']]
reg.inputs.metric_weight = [1] * 2 + [[0.5, 0.5]]
reg.inputs.radius_or_number_of_bins = [32] * 2 + [[32, 4]]
reg.inputs.sampling_strategy = ['Regular'] * 2 + [[None, None]]
reg.inputs.sampling_percentage = [0.3] * 2 + [[None, None]]
reg.inputs.convergence_threshold = [1.e-8] * 2 + [-0.01]
reg.inputs.smoothing_sigmas = [[4, 2, 1]] * 2 + [[1, 0.5, 0]]
reg.inputs.sigma_units = ['vox'] * 3
reg.inputs.shrink_factors = [[3, 2, 1]] * 2 + [[4, 2, 1]]
reg.inputs.use_estimate_learning_rate_once = [True] * 3
reg.inputs.use_histogram_matching = [False] * 2 + [True]
reg.inputs.winsorize_lower_quantile = 0.005
reg.inputs.winsorize_upper_quantile = 0.995
reg.inputs.float = True
reg.inputs.output_warped_image = 'output_warped_image.nii.gz'
reg.inputs.num_threads = 4
reg.plugin_args = {'sbatch_args': '-c %d' % 4}

register.connect(stripper, 'out_file', reg, 'moving_image')
register.connect(inputnode, 'target_image', reg, 'fixed_image')

Concatenate the affine and ants transforms into a list
merge = Node(Merge(2), iterfield=['in2'], name='mergexfm')
register.connect(convert2itk, 'itk_transform', merge, 'in2')
register.connect(reg, ('composite_transform', pickfirst), merge, 'in1')

Transform the mean image. First to anatomical and then to target
warpmean = Node(ants.ApplyTransforms(), name='warpmean')
warpmean.inputs.input_image_type = 3
warpmean.inputs.interpolation = 'Linear'
warpmean.inputs.invert_transform_flags = [False, False]
warpmean.terminal_output = 'file'
warpmean.inputs.args = '--float'
warpmean.inputs.num_threads = 4
warpmean.plugin_args = ('sbatch_args': '-c%d %d')
register.connect(inputnode, 'target_image', warpmean, 'reference_image')
register.connect(inputnode, 'mean_image', warpmean, 'input_image')
register.connect(merge, 'out', warpmean, 'transforms')

Assign all the output files
register.connect(reg, 'warped_image', outputnode, 'anat2target')
register.connect(warpmean, 'output_image', outputnode, 'transformed_mean')
register.connect(applyxfm, 'transformed_file', outputnode, 'segmentation_files')
register.connect(aparcxfm, 'transformed_file', outputnode, 'aparc')
register.connect(bbregister, 'out_fsl_file', outputnode, 'func2anat_transform')
register.connect(bbregister, 'out_reg_file', outputnode, 'out_reg_file')
register.connect(reg, 'composite_transform', outputnode, 'anat2target_transform')
register.connect(merge, 'out', outputnode, 'transforms')
register.connect(bbregister, 'min_cost_file', outputnode, 'min_cost_file')

return register

Creates the main preprocessing workflow

def create_workflow(files, target_file, subject_id, TR, slice_times, norm_threshold=1, num_components=5, vol_fwhm=None, surf_fwhm=None, lowpass_freq=-1, highpass_freq=-1, subjects_dir=None, sink_directory=os.getcwd(), target_subject=['fsaverage3', 'fsaverage4'], name='resting'):

    wf = Workflow(name=name)

    # Rename files in case they are named identically
    name_unique = MapNode(
        Rename(format_string='rest_%(run)02d'),
        iterfield=['in_file', 'run'],
        name='rename')
    name_unique.inputs.keep_ext = True
    name_unique.inputs.run = list(range(1, len(files) + 1))
    name_unique.inputs.in_file = files

    realign = Node(nipy.SpaceTimeRealigner(), name="spacetime_realign")
    realign.inputs.slice_times = slice_times
    realign.inputs.tr = TR
    realign.inputs.slice_info = 2
    realign.plugin_args = {'sbatch_args': '-c %d' % 4}

    # Compute TSNR on realigned data regressing polynomials up to order 2
    tsnr = MapNode(TSNR(regress_poly=2), iterfield=['in_file'], name='tsnr')
    wf.connect(realign, "out_file", tsnr, "in_file")

    # Compute the median image across runs
    calc_median = Node(CalculateMedian(), name='median')
    wf.connect(tsnr, 'detrended_file', calc_median, 'in_files')

    Segment and Register

    registration = create_reg_workflow(name='registration')
    wf.connect(calc_median, 'median_file', registration, 'inputspec.mean_image')
    registration.inputs.inputspec.subject_id = subject_id
    registration.inputs.inputspec.subjects_dir = subjects_dir
    registration.inputs.inputspec.target_image = target_file

    Quantify TSNR in each freesurfer ROI
get_roi_tsnr = MapNode(
    name='get_roi_tsnr',
    iterfield=['in_file'],
    function=fs.SegStats(default_color_table=True),
    name='get_aparc_tsnr')

get_roi_tsnr.inputs.avg_wf_txt_file = True
wf.connect(tsnr, 'tsnr_file', get_roi_tsnr, 'in_file')
wf.connect(registration, 'outputspec.aparc', get_roi_tsnr, 'segmentation_file')

Use nipype.algorithms.rapidart to determine which of the images in the functional series are outliers based on deviations in intensity or movement.

art = Node(interface=ArtifactDetect(), name='art')
art.inputs.use_differences = [True, True]
art.inputs.use_norm = True
art.inputs.norm_threshold = norm_threshold
art.inputs.zintensity_threshold = 9
art.inputs.mask_type = 'spm_global'
art.inputs.parameter_source = 'NiPy'

Here we are connecting all the nodes together. Notice that we add the merge node only if you choose to use 4D. Also get_vox_dims function is passed along the input volume of normalise to set the optimal voxel sizes.

wf.connect([
    (name_unique, realign, [('out_file', 'in_file')]),
    (realign, art, [('out_file', 'realigned_files')]),
    (realign, art, [('par_file', 'realignment_parameters')]),
])

def select_index(files, idx):
    import numpy as np
    from nipype.utils.filemanip import filename_to_list, list_to_filename
    return list_to_filename(np.array(filename_to_list(files))[idx].tolist())

mask = Node(fsl.BET(), name='getmask')
mask.inputs.mask = True
wf.connect(calc_median, 'median_file', mask, 'in_file')

# get segmentation in normalized functional space

def merge_files(in1, in2):
    out_files = filename_to_list(in1)
    out_files.extend(filename_to_list(in2))
    return out_files

# filter some noise

# Compute motion regressors
motreg = Node(f
    Function(
        input_names=['motion_params', 'order', 'derivatives'],
        output_names=['out_files'],
        function=motion_regressors,
        imports=imports),
    name='getmotionregress')
wf.connect(realign, 'par_file', motreg, 'motion_params')
# Create a filter to remove motion and art confounds

```python
createfilter1 = Node(
    Function(
        input_names=['motion_params', 'comp_norm', 'outliers', 'detrend_poly'],
        output_names=['out_files'],
        function=build_filter1,
        imports=imports,
        name='makemotionbasedfilter')
createfilter1.inputs.detrend_poly = 2
```

```python
wf.connect(motreg, 'out_files', createfilter1, 'motion_params')
wf.connect(art, 'norm_files', createfilter1, 'comp_norm')
wf.connect(art, 'outlier_files', createfilter1, 'outliers')
```

```python
filter1 = MapNode(
    fsl.GLM(out_f_name='F_mcart.nii.gz',
            out_pf_name='pF_mcart.nii.gz',
            demean=True),
    iterfield=['in_file', 'design', 'out_res_name'],
    name='filtermotion')
```

```python
wf.connect(realign, 'out_file', filter1, 'in_file')
wf.connect(realign, ('out_file', rename, '_filtermotart'), filter1, 'out_res_name')
wf.connect(createfilter1, 'out_files', filter1, 'design')
```

```python
createfilter2 = MapNode(
    ACompCor(),
    iterfield=['realigned_file', 'extra_regressors'],
    name='makecompccorrrfilter')
createfilter2.inputs.components_file = 'noise_components.txt'
createfilter2.inputs.num_components = num_components
```

```python
wf.connect(createfilter1, 'out_files', createfilter2, 'extra_regressors')
wf.connect(filter1, 'out_res', createfilter2, 'realigned_file')
wf.connect(registration, ('outputspec.segmentation_files', selectindex, [0, 2]),
            createfilter2, 'mask_file')
```

```python
filter2 = MapNode(
    fsl.GLM(out_f_name='F.nii.gz', out_pf_name='pF.nii.gz', demean=True),
    iterfield=['in_file', 'design', 'out_res_name'],
    name='filter_noise_nosmooth')
```

```python
wf.connect(filter1, 'out_res', filter2, 'in_file')
wf.connect(filter1, ('out_res', rename, '_cleaned'), filter2, 'out_res_name')
wf.connect(createfilter2, 'components_file', filter2, 'design')
wf.connect(mask, 'mask_file', filter2, 'mask')
```

```python
bandpass = Node(
    Function(
        input_names=['files', 'lowpass_freq', 'highpass_freq', 'fs'],
        output_names=['out_files'],
        function=bandpass_filter,
        imports=imports),
    )
```

(continues on next page)
name='bandpass_unsmooth')
b bandpass.inputs.fs = 1. / TR
bandpass.inputs.highpass_freq = highpass_freq
bandpass.inputs.lowpass_freq = lowpass_freq
wf.connect(filter2, 'out_res', bandpass, 'files')

'''Smooth the functional data using :class:`nipype.interfaces.fsl.IsotropicSmooth`.''
'''

smooth = MapNode(
    interface=fsl.IsotropicSmooth(), name="smooth", iterfield=['in_file'])
smooth.inputs.fwhm = vol_fwhm
wf.connect(bandpass, 'out_files', smooth, 'in_file')

collector = Node(Merge(2), name='collect_streams')
wf.connect(smooth, 'out_file', collector, 'in1')
wf.connect(bandpass, 'out_files', collector, 'in2')

'''Transform the remaining images. First to anatomical and then to target
'''

warpall = MapNode(
    ants.ApplyTransforms(), iterfield=['input_image'], name='warpall')
warpall.inputs.input_image_type = 3
warpall.inputs.interpolation = 'Linear'
warpall.inputs.invert_transform_flags = [False, False]
warpall.terminal_output = 'file'
warpall.inputs.reference_image = target_file
warpall.inputs.args = '--float'
warpall.inputs.num_threads = 2
warpall.plugin_args = {"sbatch_args": '-c %d' % 2}

# transform to target
wf.connect(collector, 'out', warpall, 'input_image')
wf.connect(registration, 'outputspec.transforms', warpall, 'transforms')

mask_target = Node(fsl.ImageMaths(op_string='-bin'), name='target_mask')
wf.connect(registration, 'outputspec.anat2target', mask_target, 'in_file')

maskts = MapNode(fsl.ApplyMask(), iterfield=['in_file'], name='ts_masker')
wf.connect(warpall, 'output_image', maskts, 'in_file')
wf.connect(mask_target, 'out_file', maskts, 'mask_file')

# map to surface
# extract aparc+aseg ROIs
# extract subcortical ROIs
# extract target space ROIs
# combine subcortical and cortical rois into a single cifti file

#####
# Convert aparc to subject functional space

# Sample the average time series in aparc ROIs
sampleaparc = MapNode(
    freesurfer.SegStats(default_color_table=True),
    interface=freesurfer.SegStats(), name="sample")

iterfield=['in_file', 'summary_file', 'avgwf_txt_file'],
name='aparc_ts')
sampleaparc.inputs.segment_id = (  [8] + list(range(10, 14)) + [17, 18, 26, 47] + list(range(49, 55)) + [58] + list(range(1001, 1036)) + list(range(2001, 2036)))

wf.connect(registration, 'outputspec.aparc', sampleaparc, 'segmentation_file')
wf.connect(collector, 'out', sampleaparc, 'in_file')

def get_names(files, suffix):
    """Generate appropriate names for output files"
    ""
    from nipype.utils.filemanip import (split_filename, filename_to_list, list_to_filename)
    import os
    out_names = []
    for filename in files:
        path, name, _ = split_filename(filename)
        out_names.append(os.path.join(path, name + suffix))
    return list_to_filename(out_names)

wf.connect(collector, ('out', get_names, '_avgwf.txt'), sampleaparc, 'avgwf_txt_file')
wf.connect(collector, ('out', get_names, '_summary.stats'), sampleaparc, 'summary_file')
# Sample the time series onto the surface of the target surface. Performs
# sampling into left and right hemisphere

target = Node(IdentityInterface(fields=['target_subject']), name='target')
target.iterables = ('target_subject', filename_to_list(target_subject))
samplerlh = MapNode(
    freesurfer.SampleToSurface(),
    iterfield=['source_file'],
    name='sampler_lh')
samplerlh.inputs.sampling_method = "average"
samplerlh.inputs.sampling_range = (0.1, 0.9, 0.1)
samplerlh.inputs.sampling_units = "frac"
samplerlh.inputs.interp_method = "trilinear"
samplerlh.inputs.smooth_surf = surf_fwhm
# samplerlh.inputs.cortex_mask = True
samplerlh.inputs.out_type = 'niigz'
samplerlh.inputs.subjects_dir = subjects_dir
samplerrh = samplerlh.clone('sampler_rh')
samplerlh.inputs.hemi = 'lh'
wf.connect(collector, 'out', samplerlh, 'source_file')
wf.connect(registration, 'outputspec.out_reg_file', samplerlh, 'reg_file')
wf.connect(target, 'target_subject', samplerlh, 'target_subject')
samplerrh.set_input('hemi', 'rh')
wf.connect(collector, 'out', samplerrh, 'source_file')
wf.connect(registration, 'outputspec.out_reg_file', samplerrh, 'reg_file')
wf.connect(target, 'target_subject', samplerrh, 'target_subject')
# Combine left and right hemisphere to text file
combiner = MapNode(
    Function(
        input_names=['left', 'right'],
        output_names=['out_file'],
        function=combine_hemi,
        imports=imports),
    iterfield=['left', 'right'],
    name="combiner")
wf.connect(samplerlh, 'out_file', combiner, 'left')
wf.connect(samplerrh, 'out_file', combiner, 'right')

# Sample the time series file for each subcortical roi
ts2txt = MapNode(
    Function(
        input_names=['timeseries_file', 'label_file', 'indices'],
        output_names=['out_file'],
        function=extract_subrois,
        imports=imports),
    iterfield=['timeseries_file'],
    name='getsubcortts')
ts2txt.inputs.indices = [8] + list(range(10, 14)) + [17, 18, 26, 47] +
    list(range(49, 55)) + [58]
ts2txt.inputs.label_file = os.path.abspath('OASIS-TRT-20_jointfusion_DKT31_CMA_labels_in_MNI152_'
    '2mm_v2.nii.gz')
wf.connect(maskts, 'out_file', ts2txt, 'timeseries_file')

substitutions = [
    ('_target_subject_', ''),
    ('filtermotart_cleaned_bp_trans_masked', ''),
    ('filtermotart_cleaned_bp', ''),
]
substitutions += ["_smooth%d" % i, "") for i in range(11)[::-1]]
substitutions += ["_ts_masker%d" % i, "") for i in range(11)[::-1]]
substitutions += ["_getsubcortts%d" % i, "") for i in range(11)[::-1]]
substitutions += ["_filtermotion%d" % i, "") for i in range(11)[::-1]]
substitutions += ["_get_aparc_tsnr%d/" % i, "run%d_"] % (i + 1))
    for i in range(11)[::-1]]
substitutions += ["_makecompcorfilter%d" % i, "")
    for i in range(11)[::-1]]
substitutions += ["_get_aparc_tsnr%d/" % i, "run%d_"] % (i + 1))
    for i in range(11)[::-1]]
substitutions += ["T1_outBrain_pve_0_maths_warped", "compcor_csf"],
    ("T1_outBrain_pve_1_maths_warped", "compcor_gm"),
    ("T1_outBrain_pve_2_maths_warped", "compcor_wm"),
    ("output_warped_image_maths", "target_brain_mask"), ("median_brain_mask", "native_brain_mask"),
    ("corr", "]
regex_subs = [
(continues on next page)
(continued from previous page)

`('_combiner.*sar', '/smooth/'),
('._combiner.*ar', '/unsmooth/'),
('._aparc_ts.*sar', '/smooth/'),
('._aparc_ts.*ar', '/unsmooth/'),
('._getsubcortts.*sar', '/smooth/'),
('._getsubcortts.*ar', '/unsmooth/'),
('series/sar', 'series/smooth/'),
('series/ar', 'series/unsmooth/'),
('_inverse_transform./', ''),
]

# Save the relevant data into an output directory

datasink = Node(interface=DataSink(), name="datasink")
datasink.inputs.base_directory = sink_directory
datasink.inputs.container = subject_id
datasink.inputs.substitutions = substitutions
datasink.inputs.regexp_substitutions = regex_subs

wf.connect(realign, 'par_file', datasink, 'resting.qa.motion')
wf.connect(art, 'norm_files', datasink, 'resting.qa.art.@norm')
wf.connect(art, 'intensity_files', datasink, 'resting.qa.art.@intensity')
wf.connect(art, 'outlier_files', datasink, 'resting.qa.art.@outlier_files')
wf.connect(registration, 'outputspec.segmentation_files', datasink, 'resting.mask_files')
wf.connect(registration, 'outputspec.anat2target', datasink, 'resting.qa.ants')
wf.connect(mask, 'mask_file', datasink, 'resting.mask_files.@brainmask')
wf.connect(mask_target, 'out_file', datasink, 'resting.mask_files.target')
wf.connect(filter1, 'out_f', datasink, 'resting.qa.compmaps.@mc_F')
wf.connect(filter1, 'out_pf', datasink, 'resting.qa.compmaps.@mc_pF')
wf.connect(filter2, 'out_f', datasink, 'resting.qa.compmaps')
wf.connect(filter2, 'out_pf', datasink, 'resting.qa.compmaps.@p')
wf.connect(registration, 'outputspec.min_cost_file', datasink, 'resting.qa.mincost')
wf.connect(tsnr, 'tsnr_file', datasink, 'resting.qa.tsnr.@map')
wf.connect([[get_roi_tsnr, datasink, ['avgwf_txt_file', 'summary_file'],
('summary_file', 'resting.qa.tsnr.@summary')]])

wf.connect(bandpass, 'out_files', datasink, 'resting.timeseries.@bandpassed')
wf.connect(smooth, 'out_file', datasink, 'resting.timeseries.@smoothed')
wf.connect(createfilter1, 'out_files', datasink, 'resting.regress.@regressors')
wf.connect(createfilter2, 'components_file', datasink, 'resting.regress.@compcorr')
wf.connect(maskts, 'out_file', datasink, 'resting.timeseries.target')
wf.connect(sampleaparc, 'summary_file', datasink, 'resting.parcellations.aparc')
wf.connect(sampleaparc, 'avgwf_txt_file', datasink, 'resting.parcellations.aparc.@avgwf')
wf.connect(ts2txt, 'out_file', datasink, 'resting.parcellations.grayo.@subcortical')

datasink2 = Node(interface=DataSink(), name="datasink2")
datasink2.inputs.base_directory = sink_directory
datasink2.inputs.container = subject_id
datasink2.inputs.substitutions = substitutions
datasink2.inputs.regexp_substitutions = regex_subs

(continues on next page)
wf.connect(combiner, 'out_file', datasink2, 'resting.parcellations.grayo.@surface')
return wf

Creates the full workflow including getting information from dicom files

def create_resting_workflow(args, name=None):
    TR = args.TR
    slice_times = args.slice_times
    if args.dicom_file:
        TR, slice_times, slice_thickness = get_info(args.dicom_file)
        slice_times = (np.array(slice_times) / 1000.).tolist()
    if name is None:
        name = 'resting_' + args.subject_id
    kwargs = dict(
        files=[os.path.abspath(filename) for filename in args.files],
        target_file=os.path.abspath(args.target_file),
        subject_id=args.subject_id,
        TR=TR,
        slice_times=slice_times,
        vol_fwhm=args.vol_fwhm,
        surf_fwhm=args.surf_fwhm,
        norm_threshold=2.,
        subjects_dir=os.path.abspath(args.fsdir),
        target_subject=args.target_surfs,
        lowpass_freq=args.lowpass_freq,
        highpass_freq=args.highpass_freq,
        sink_directory=os.path.abspath(args.sink),
        name=name)
    wf = create_workflow(**kwargs)
    return wf

if __name__ == "__main__":
    from argparse import ArgumentParser, RawTextHelpFormatter
defstr = " (default %(default)s)"
parser = ArgumentParser(
    description=__doc__, formatter_class=RawTextHelpFormatter)
parser.add_argument(  
    "-d",  
    "--dicom_file",  
    dest="dicom_file",  
    help="a SIEMENS example dicom file from the resting series")
parser.add_argument(  
    "-f",  
    "--files",  
    dest="files",  
    nargs="+",  
    help="4d nifti files for resting state",  
    required=True)
parser.add_argument(  
    "-t",  
    "--target",  
    dest="target_file",  
    help="Target in MNI space. Best to use the MindBoggle "
    "template - "

(continues on next page)
"OASIS-30_Atropos_template_in_MNI152_2mm.nii.gz"),
required=True)
parser.add_argument(
    "-s",
    "--subject_id",
    dest="subject_id",
    help="FreeSurfer subject id",
    required=True)
parser.add_argument(
    "--subjects_dir",
    dest="fsdir",
    help="FreeSurfer subject directory",
    required=True)
parser.add_argument(
    "--target_surfaces",
    dest="target_surfs",
    nargs="*",
    default=['fsaverage5'],
    help="FreeSurfer target surfaces" + defstr)
parser.add_argument(
    "--TR",
    dest="TR",
    default=None,
    type=float,
    help="TR if dicom not provided in seconds")
parser.add_argument(
    "--slice_times",
    dest="slice_times",
    nargs="*",
    type=float,
    help="Slice onset times in seconds")
parser.add_argument(
    "--vol_fwhm",
    default=6.,
    dest='vol_fwhm',
    type=float,
    help="Spatial FWHM" + defstr)
parser.add_argument(
    "--surf_fwhm",
    default=15.,
    dest='surf_fwhm',
    type=float,
    help="Spatial FWHM" + defstr)
parser.add_argument(
    "-l",
    "--lowpass_freq",
    dest="lowpass_freq",
    default=0.1,
    type=float,
    help="Low pass frequency (Hz)" + defstr)
parser.add_argument(
    "-u",
    "--highpass_freq",
    dest="highpass_freq",
    default=0.01,
    type=float,
    help="High pass frequency (Hz)" + defstr)
parser.add_argument(
    "-o",
    "--output_dir",
    dest="sink",
    help="Output directory base",
    required=True
)
parser.add_argument(
    "-w", "--work_dir", dest="work_dir", help="Output directory base")
parser.add_argument(
    "-p",
    "--plugin",
    dest="plugin",
    default='Linear',
    help="Plugin to use")
parser.add_argument(
    "--plugin_args", dest="plugin_args", help="Plugin arguments")
args = parser.parse_args()
wf = create_resting_workflow(args)

if args.work_dir:
    work_dir = os.path.abspath(args.work_dir)
else:
    work_dir = os.getcwd()
wf.base_dir = work_dir
if args.plugin_args:
    wf.run(args.plugin, plugin_args=eval(args.plugin_args))
else:
    wf.run(args.plugin)

---

**Example source code**
You can download the full source code of this example. This same script is also included in the Nipype source distribution under the examples directory.
from builtins import range
import nipype.interfaces.io as nio  # Data i/o
import nipype.interfaces.spm as spm  # spm
import nipype.interfaces.freesurfer as fs  # freesurfer
import nipipe.interfaces.nipy as nipy
import nipipe.interfaces.utility as util
import nipipe.pipeline.engine as pe  # pypeline engine
import nipipe.algorithms.modelgen as model  # model specification
import nipipe.workflows.fmri.fsl as fsl_wf
import os  # system functions

preprocessing = pe.Workflow(name="preprocessing")
iter_fwhm = pe.Node(
    interface=util.IdentityInterface(fields=["fwhm"]), name="iter_fwhm")
iter_fwhm.iterables = ["fwhm", [4, 8]]

iter_smoothing_method = pe.Node(
    interface=util.IdentityInterface(fields=["smoothing_method"]),
    name="iter_smoothing_method")
iter_smoothing_method.iterables = [:smoothing_method", ["isotropic_voxel", "anisotropic_voxel", "isotropic_surface"]]

realign = pe.Node(interface=spm.Realign(), name="realign")
realign.inputs.register_to_mean = True

isotropic_voxel_smooth = pe.Node(
    interface=spm.Smooth(), name="isotropic_voxel_smooth")
preprocessing.connect(realign, "realigned_files", isotropic_voxel_smooth, "in_files")
preprocessing.connect(iter_fwhm, "fwhm", isotropic_voxel_smooth, "fwhm")

compute_mask = pe.Node(interface=nipy.ComputeMask(), name="compute_mask")
(continues on next page)
preprocessing.connect(realign, "mean_image", compute_mask, "mean_volume")

anisotropic_voxel_smooth = fsl_wf.create_susan_smooth(
    name="anisotropic_voxel_smooth", separate_masks=False)
anisotropic_voxel_smooth.inputs.smooth.output_type = 'NIFTI'
preprocessing.connect(realign, "realigned_files", anisotropic_voxel_smooth,
    "inputnode.in_files")
preprocessing.connect(iter_fwhm, "fwhm", anisotropic_voxel_smooth,
    "inputnode.fwhm")
preprocessing.connect(compute_mask, "brain_mask", anisotropic_voxel_smooth,
    'inputnode.mask_file')
recon_all = pe.Node(interface=fs.ReconAll(), name="recon_all")

surfregister = pe.Node(interface=fs.BBRegister(), name='surfregister')
surfregister.inputs.init = 'fsl'
surfregister.inputs.contrast_type = 't2'
preprocessing.connect(realign, 'mean_image', surfregister, 'source_file')
preprocessing.connect(recon_all, 'subject_id', surfregister, 'subject_id')
preprocessing.connect(recon_all, 'subjects_dir', surfregister, 'subjects_dir')

isotropic_surface_smooth = pe.MapNode(
    interface=fs.Smooth(proj_frac_avg=(0, 1, 0.1)),
    iterfield=['in_file'],
    name="isotropic_surface_smooth")
preprocessing.connect(surfregister, 'out_reg_file', isotropic_surface_smooth,
    'reg_file')
preprocessing.connect(realign, "realigned_files", isotropic_surface_smooth,
    "in_file")
preprocessing.connect(iter_fwhm, "fwhm", isotropic_surface_smooth,
    "surface_fwhm")
preprocessing.connect(iter_fwhm, "fwhm", isotropic_surface_smooth, "vol_fwhm")
preprocessing.connect(recon_all, 'subjects_dir', isotropic_surface_smooth,
    'subjects_dir')

merge_smoothed_files = pe.Node(
    interface=util.Merge(3), name="merge_smoothed_files")
preprocessing.connect(isotropic_voxel_smooth, 'smoothed_files',
    merge_smoothed_files, 'in1')
preprocessing.connect(anisotropic_voxel_smooth, 'outputnode.smoothed_files',
    merge_smoothed_files, 'in2')
preprocessing.connect(isotropic_surface_smooth, 'smoothed_file',
    merge_smoothed_files, 'in3')

select_smoothed_files = pe.Node(
    interface=util.Select(), name="select_smoothed_files")
preprocessing.connect(merge_smoothed_files, 'out', select_smoothed_files,
    'inlist')

def choose_index(roi):
    return {
        'isotropic_voxel': list(range(0, 4)),
        'anisotropic_voxel': list(range(4, 8)),
        'isotropic_surface': list(range(8, 12))
    }[roi]
preprocessing.connect(iter_smoothing_method, ("smoothing_method", chooseindex),
select_smoothed_files, 'index')

rename = pe.MapNode(
    util.Rename(format_string="%(orig)s"),
    name="rename",
    iterfield=['in_file'])
rename.inputs.parse_string = "(?P<orig>.*)"
preprocessing.connect(select_smoothed_files, 'out', rename, 'in_file')

specify_model = pe.Node(interface=model.SpecifyModel(), name="specify_model")
specify_model.inputs.input_units = 'secs'
specify_model.inputs.time_repetition = 3
specify_model.inputs.high_pass_filter_cutoff = 120
specify_model.inputs.subject_info = [
    Bunch(
        conditions=['Task-Odd', 'Task-Even'],
        onsets=[list(range(15, 240, 60)),
                list(range(45, 240, 60))],
        durations=[[15], [15]])
] * 4
level1design = pe.Node(interface=spm.Level1Design(), name="level1design")
level1design.inputs.bases = {'hrf': {'derivs': [0, 0]}}
level1design.inputs.timing_units = 'secs'
level1design.inputs.interscan_interval = specify_model.inputs.time_repetition
level1estimate = pe.Node(interface=spm.EstimateModel(), name="level1estimate")
level1estimate.inputs.estimation_method = {'Classical': 1}
contrastestimate = pe.Node(
    interface=spm.EstimateContrast(), name="contrastestimate")
contrastestimate.inputs.contrasts = [('Task>Baseline', 'T',
    ['Task-Odd', 'Task-Even'], [0.5, 0.5])]
modelling = pe.Workflow(name="modelling")
modelling.connect(specify_model, 'session_info', level1design, 'session_info')
modelling.connect(level1design, 'spm_mat_file', level1estimate, 'spm_mat_file')
modelling.connect(level1estimate, 'spm_mat_file', contrastestimate, 'spm_mat_file')
modelling.connect(level1estimate, 'beta_images', contrastestimate, 'beta_images')
modelling.connect(level1estimate, 'residual_image', contrastestimate, 'residual_image')
main_workflow = pe.Workflow(name="main_workflow")
main_workflow.base_dir = "smoothing_comparison_workflow"
main_workflow.connect(preprocessing, "realign.realignment_parameters",
modelling, "specify_model.realignment_parameters")
main_workflow.connect(preprocessing, "select_smoothed_files.out", modelling, "specify_model.functional_runs")
main_workflow.connect(preprocessing, "compute_mask.brain_mask", modelling, "level1design.mask_image")
interface=nio.DataGrabber(
    infields=['subject_id'], outfields=['func', 'struct'],
    name='datasource')
datasource.inputs.base_directory = os.path.abspath('data')
datasource.inputs.template = '%$/$s.nii'
datasource.inputs.template_args = info = dict(
    func=[['subject_id', ['f3', 'f5', 'f7', 'f10']]],
    struct=[['subject_id', 'struct']])
datasource.inputs.subject_id = 's1'
datasource.inputs.sort_filelist = True

main_workflow.connect(datasource, 'func', preprocessing, 'realign.in_files')
main_workflow.connect(datasource, 'struct', preprocessing, 'recon_all.T1_files')

datasink = pe.Node(interface=nio.DataSink(), name="datasink")
datasink.inputs.base_directory = os.path.abspath('smoothing_comparison_workflow/output')
datasink.inputs.regexp_substitutions = [('_rename[0-9]', '')]
main_workflow.connect(modelling, 'contrastestimate.spmT_images', datasink, 'contrasts')
main_workflow.connect(preprocessing, 'rename.out_file', datasink, 'smoothed_epi')

main_workflow.run()
main_workflow.write_graph()

Example source code
You can download the full source code of this example. This same script is also included in the Nipype source distribution under the examples directory.
sMRI: Using new ANTS for creating a T1 template

In this tutorial we will use ANTS (old version aka “ANTS”) based workflow to create a template out of multiple T1 volumes.

1. Tell python where to find the appropriate functions.

```python
from __future__ import print_function, unicode_literals
from builtins import open
from future import standard_library
standard_library.install_aliases()

import os
import nipype.interfaces.utility as util
import nipype.interfaces.ants as ants
import nipype.interfaces.io as io
import nipype.pipeline.engine as pe  # pipeline engine

from nipype.workflows.smri.ants import ANTSTemplateBuildSingleIterationWF
```

2. Download T1 volumes into home directory

```python
import urllib.request
import urllib.error
import urllib.parse

homeDir = os.getenv("HOME")
requestedPath = os.path.join(homeDir, 'nipypeTestPath')
mydatadir = os.path.realpath(requestedPath)
if not os.path.exists(mydatadir):
    os.makedirs(mydatadir)
print(mydatadir)

MyFileURLs = [
    ('http://slicer.kitware.com/midas3/download?bitstream=13121', '01_T1_half.nii.gz'),
    ('http://slicer.kitware.com/midas3/download?bitstream=13122', '02_T1_half.nii.gz'),
    ('http://slicer.kitware.com/midas3/download?bitstream=13124', '03_T1_half.nii.gz'),
    (continues on next page)
for tt in MyFileURLs:
    myURL = tt[0]
    localFilename = os.path.join(mydatadir, tt[1])
    if not os.path.exists(localFilename):
        remotefile = urllib.request.urlopen(myURL)
        localFile = open(localFilename, 'wb')
        localFile.write(remotefile.read())
        localFile.close()
        print("Downloaded file: {0}".format(localFilename))
    else:
        print("File previously downloaded {0}".format(localFilename))

input_images = [
    os.path.join(mydatadir, '01_T1_half.nii.gz'),
    os.path.join(mydatadir, '02_T1_half.nii.gz'),
    os.path.join(mydatadir, '03_T1_half.nii.gz')
]
input_passive_images = [{
    'INV_T1': os.path.join(mydatadir, '01_T1_inv_half.nii.gz')
}, {
    'INV_T1': os.path.join(mydatadir, '02_T1_inv_half.nii.gz')
}, {
    'INV_T1': os.path.join(mydatadir, '03_T1_inv_half.nii.gz')
}]

3. Define the workflow and its working directory

tbuilder = pe.Workflow(name="ANTSTemplateBuilder")
tbuilder.base_dir = requestedPath

4. Define data sources. In real life these would be replace by DataGrabbers

datasource = pe.Node(
    interface=util.IdentityInterface(
        fields=['imageList', 'passiveImagesDictionariesList']),
    run_without_subitting=True,
    name='InputImages')
datasource.inputs.imageList = input_images
datasource.inputs.passiveImagesDictionariesList = input_passive_images
datasource.inputs.sort_filelist = True

5. Template is initialized by a simple average

initAvg = pe.Node(interface=ants.AverageImages(), name='initAvg')
initAvg.inputs.dimension = 3
initAvg.inputs.normalize = True
tbuilder.connect(datasource, "imageList", initAvg, "images")

6. Define the first iteration of template building
7. Define the second iteration of template building

```python
buildTemplateIteration2 = ANTSTemplateBuildSingleIterationWF('iteration02')
tbuilder.connect(buildTemplateIteration1, 'outputspec.template',
                 buildTemplateIteration2, 'inputspec.fixed_image')
tbuilder.connect(datasource, 'imageList', buildTemplateIteration2,
                 'inputspec.images')
tbuilder.connect(datasource, 'passiveImagesDictionariesList',
                 buildTemplateIteration2,
                 'inputspec.ListOfPassiveImagesDictionaries')
```

8. Move selected files to a designated results folder

```python
datasink = pe.Node(io.DataSink(), name="datasink")
datasink.inputs.base_directory = os.path.join(requestedPath, "results")
tbuilder.connect(buildTemplateIteration2, 'outputspec.template', datasink,
                 'PrimaryTemplate')
tbuilder.connect(buildTemplateIteration2, 'outputspec.passive_deformed_templates', datasink,
                 'PassiveTemplate')
tbuilder.connect(initAvg, 'output_average_image', datasink,
                 'PreRegisterAverage')
```

8. Run the workflow

```python
tbuilder.run()
```

**Example source code**

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the examples directory.
In this simple tutorial we will use the Registration interface from ANTS to coregister two T1 volumes.

1. Tell python where to find the appropriate functions.

```python
from __future__ import print_function, unicode_literals
from builtins import open
from future import standard_library
standard_library.install_aliases()

import os
import urllib.request
import urllib.error
import urllib.parse
from nipype.interfaces.ants import Registration
from nipype.testing import example_data
```

2. Download T1 volumes into home directory

```python
homeDir = os.getenv("HOME")
requestedPath = os.path.join(homeDir, 'nipypeTestPath')
mydatadir = os.path.realpath(requestedPath)
if not os.path.exists(mydatadir):
    os.makedirs(mydatadir)
print(mydatadir)

MyFileURLs = [
    ('http://slicer.kitware.com/midas3/download?bitstream=13121', '01_T1_half.nii.gz'),
    ('http://slicer.kitware.com/midas3/download?bitstream=13122', '02_T1_half.nii.gz'),
]
for tt in MyFileURLs:
    myURL = tt[0]
    localFilename = os.path.join(mydatadir, tt[1])
    if not os.path.exists(localFilename):
        remotefile = urllib.request.urlopen(myURL)
```

(continues on next page)
localFile = open(localFilename, 'wb')
localFile.write(remotefile.read())
localFile.close()

print("Downloaded file: {0}".format(localFilename))
else:
    print("File previously downloaded {0}".format(localFilename))

input_images = [
    os.path.join(mydatadir, '01_T1_half.nii.gz'),
    os.path.join(mydatadir, '02_T1_half.nii.gz'),
]

3. Define the parameters of the registration. Settings are found in the file
smri_ants_registration_settings.json distributed with the example_data of nipype.

reg = Registration(
    from_file=example_data('smri_ants_registration_settings.json'))
reg.inputs.fixed_image = input_images[0]
reg.inputs.moving_image = input_images[1]

Alternatively to the use of the from_file feature to load ANTs settings, the user can manually set all those
inputs instead:

reg.inputs.output_transform_prefix = 'thisTransform'
reg.inputs.output_warped_image = 'INTERNAL_WARPED.nii.gz'
reg.inputs.output_transform_prefix = "output_"
reg.inputs.transforms = [\'Translation\', \'Rigid\', \'Affine\', \'SyN\']
reg.inputs.transform_parameters = [((0.1,), (0.1,), (0.1,), (0.2, 3.0, 0.0))]
reg.inputs.number_of_iterations = (((10000, 111110, 111110)) * 3 +
    [[100, 50, 30]])
reg.inputs.dimension = 3
reg.inputs.write_composite_transform = True
reg.inputs.collapse_output_transforms = False
reg.inputs.metric = [\'Mattes\'] * 3 + [\'Mattes\', \'CC\']
reg.inputs.metric_weight = [1] * 3 + [[0.5, 0.5]]
reg.inputs.radius_or_number_of_bins = [32] * 3 + [[32, 4]]
reg.inputs.sampling_strategy = [\'Regular\'] * 3 + [[None, None]]
reg.inputs.sampling_percentage = [0.3] * 3 + [[None, None]]
reg.inputs.convergence_threshold = [1.e-8] * 3 + [-0.01]
reg.inputs.smoothing_sigmas = [4, 2, 1]) * 3 + [[1, 0.5, 0]]
reg.inputs.sigma_units = [\'vox\'] * 4
reg.inputs.shrink_factors = [[6, 4, 2]] + [[3, 2, 1]] * 2 + [[4, 2, 1]]
reg.inputs.use_estimate_learning_rate_once = [True] * 4
reg.inputs.use_histogram_matching = [False] * 3 + [True]
reg.inputs.initial_moving_transform_com = True

print(reg.cmdline)

3. Run the registration

reg.run()

Example source code
You can download the full source code of this example. This same script is also included in
the Nipype source distribution under the examples directory.
**sMRI: Using new ANTS for creating a T1 template (ITK4)**

In this tutorial we will use ANTS (new ITK4 version aka “antsRegistration”) based workflow to create a template out of multiple T1 volumes. We will also showcase how to fine tune SGE jobs requirements.

1. Tell python where to find the appropriate functions.

```python
from __future__ import print_function
from future import standard_library
standard_library.install_aliases()
import os
import nipype.interfaces.utility as util
import nipype.interfaces.ants as ants
import nipype.interfaces.io as io
import nipype.pipeline.engine as pe  # pypeline engine

from nipype.workflows.smri.ants import antsRegistrationTemplateBuildSingleIterationWF
```

2. Download T1 volumes into home directory

```python
import urllib.request
import urllib.error
import urllib.parse

homeDir = os.getenv("HOME")
requestedPath = os.path.join(homeDir, 'nipypeTestPath')
mydatadir = os.path.realpath(requestedPath)
if not os.path.exists(mydatadir):
    os.makedirs(mydatadir)
print(mydatadir)

MyFileURLs = [
     '01_T1_half.nii.gz'),
     '02_T1_half.nii.gz'),
     '03_T1_half.nii.gz'),
     '04_T1_half.nii.gz')
]
```
for tt in MyFileURLs:
    myURL = tt[0]
    localFilename = os.path.join(mydatadir, tt[1])
    if not os.path.exists(localFilename):
        remotefile = urllib.request.urlopen(myURL)
        localFile = open(localFilename, 'wb')
        localFile.write(remotefile.read())
        localFile.close()
        print("Downloaded file: {0}"").format(localFilename))
    else:
        print("File previously downloaded {0}"").format(localFilename))

ListOfImagesDictionaries - a list of dictionaries where each dictionary is for one scan session, and the mappings in the dictionary are for all the co-aligned images for that one scan session

ListOfImagesDictionaries = [{
    'T1':
        os.path.join(mydatadir, '01_T1_half.nii.gz'),
    'INV_T1':
        os.path.join(mydatadir, '01_T1_inv_half.nii.gz'),
    'LABEL_MAP':
        os.path.join(mydatadir, '01_T1_inv_half.nii.gz')
},
{ 
    'T1':
        os.path.join(mydatadir, '02_T1_half.nii.gz'),
    'INV_T1':
        os.path.join(mydatadir, '02_T1_inv_half.nii.gz'),
    'LABEL_MAP':
        os.path.join(mydatadir, '02_T1_inv_half.nii.gz')
},
{ 
    'T1':
        os.path.join(mydatadir, '03_T1_half.nii.gz'),
    'INV_T1':
        os.path.join(mydatadir, '03_T1_inv_half.nii.gz'),
    'LABEL_MAP':
        os.path.join(mydatadir, '03_T1_inv_half.nii.gz')
}]

input_passive_images = [{
    'INV_T1':
        os.path.join(mydatadir, '01_T1_inv_half.nii.gz')
},
{ 
    'INV_T1':
        os.path.join(mydatadir, '02_T1_inv_half.nii.gz')
},
{ 
    'INV_T1':
        os.path.join(mydatadir, '03_T1_inv_half.nii.gz')
}]

registrationImageTypes - A list of the image types to be used actively during the estimation process of registration, any image type not in this list will be passively resampled with the estimated transforms. ['T1','T2']
registrationImageTypes = ['T1']

interpolationMap - A map of image types to interpolation modes. If an image type is not listed, it will be linearly interpolated. { 'labelmap': 'NearestNeighbor', 'FLAIR': 'WindowedSinc'}

interpolationMapping = {
    'INV_T1': 'LanczosWindowedSinc',
    'LABEL_MAP': 'NearestNeighbor',
    'T1': 'Linear'
}

3. Define the workflow and its working directory

tbuilder = pe.Workflow(name="antsRegistrationTemplateBuilder")
tbuilder.base_dir = requestedPath

4. Define data sources. In real life these would be replace by DataGrabbers

InitialTemplateInputs = [mdict['T1'] for mdict in ListOfImagesDictionaries]
datasource = pe.Node(
    interface=util.IdentityInterface(fields=['InitialTemplateInputs', 'ListOfImagesDictionaries',
                                             'registrationImageTypes', 'interpolationMapping']),
    run_without_submitting=True,
    name='InputImages')
datasource.inputs.InitialTemplateInputs = InitialTemplateInputs
datasource.inputs.ListOfImagesDictionaries = ListOfImagesDictionaries
datasource.inputs.registrationImageTypes = registrationImageTypes
datasource.inputs.interpolationMapping = interpolationMapping
datasource.inputs.sort_filelist = True

5. Template is initialized by a simple average in this simple example, any reference image could be used (i.e. a previously created template)

initAvg = pe.Node(interface=ants.AverageImages(), name='initAvg')
initAvg.inputs.dimension = 3
initAvg.inputs.normalize = True

tbuilder.connect(datasource, "InitialTemplateInputs", initAvg, "images")

6. Define the first iteration of template building

buildTemplateIteration1 = antsRegistrationTemplateBuildSingleIterationWF(
    'iteration01')

Here we are fine tuning parameters of the SGE job (memory limit, number of cores etc.)

BeginANTS = buildTemplateIteration1.get_node("BeginANTS")
BeginANTS.plugin_args = {
    'qsub_args':
        '-S /bin/bash -pe smp1 8-12 -l mem_free=6000M -o /dev/null -e /dev/null queue_-
        "name",
        'overwrite':
            True
}

tbuilder.connect(initAvg, 'output_average_image', buildTemplateIteration1,
    'inputspec.fixed_image')

(continues on next page)
7. Define the second iteration of template building

```python
tbuilder.connect(datasource, 'ListOfImagesDictionaries',
                 buildTemplateIteration1, 'inputspec.ListOfImagesDictionaries')
tbuilder.connect(datasource, 'registrationImageTypes', buildTemplateIteration1,
                 'inputspec.registrationImageTypes')
tbuilder.connect(datasource, 'interpolationMapping', buildTemplateIteration1,
                 'inputspec.interpolationMapping')
```

8. Move selected files to a designated results folder

```python
datasink = pe.Node(io.DataSink(), name="datasink")
datasink.inputs.base_directory = os.path.join(requestedPath, "results")
```

```python
tbuilder.connect(buildTemplateIteration2, 'outputspec.template',
                 datasink, 'PrimaryTemplate')
tbuilder.connect(buildTemplateIteration2,
                 'outputspec.passive_deformed_templates', datasink,
                 'PassiveTemplate')
tbuilder.connect(initAvg, 'output_average_image', datasink,
                 'PreRegisterAverage')
```

9. Run the workflow

```python
tbuilder.run(plugin="SGE")
```

**Example source code**

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the examples directory.
This simple workflow uses SPECTRE2010 algorithm to skullstrip an MP2RAGE anatomical scan.

```python
import nipype.pipeline.engine as pe
from nipype.interfaces.mipav.developer import JistIntensityMp2rageMasking, MedicAlgorithmSPECTRE2010

wf = pe.Workflow("skullstripping")

mask = pe.Node(JistIntensityMp2rageMasking(), name="masking")
folder_path = '/Users/filo/7t_trt/niftis/sub001/session_1/
mask.inputs.inSecond = folder_path + "MP2RAGE_INV2.nii.gz"
mask.inputs.inQuantitative = folder_path + "MP2RAGE_UNI.nii.gz"
mask.inputs.inT1weighted = folder_path + "MP2RAGE_T1.nii.gz"
mask.inputs.outMasked = True
mask.inputs.outMasked2 = True
mask.inputs.outSignal = True
mask.inputs.outSignal2 = True

skullstrip = pe.Node(MedicAlgorithmSPECTRE2010(), name="skullstrip")
skullstrip.inputs.outStripped = True
skullstrip.inputs.xDefaultMem = 6000

wf.connect(mask, 'outMasked', skullstrip, 'inInput')
wf.run()
```

**Example source code**

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the examples directory.
This script, smri_freesurfer.py, demonstrates the ability to call reconall on a set of subjects and then make an average subject:

```python
python smri_freesurfer.py
```

Import necessary modules from nipype.

```python
import os
import nipype.pipeline.engine as pe
import nipype.interfaces.io as nio
from nipype.interfaces.freesurfer.preprocess import ReconAll
from nipype.interfaces.freesurfer.utils import MakeAverageSubject
```

```python
subject_list = ['s1', 's3']
data_dir = os.path.abspath('data')
subjects_dir = os.path.abspath('amri_freesurfer_tutorial/subjects_dir')
wf = pe.Workflow(name='l1workflow')
wf.base_dir = os.path.abspath('amri_freesurfer_tutorial/workdir')
```

Grab data

```python
datasource = pe.MapNode(
    interface=nio.DataGrabber(infields=['subject_id'], outfields=['struct']),
    name='datasource',
    iterfield=['subject_id'])
datasource.inputs.base_directory = data_dir
datasource.inputs.template = '%s/%s.nii'
datasource.inputs.template_args = dict(struct=[['subject_id', 'struct']])
datasource.inputs.subject_id = subject_list
datasource.inputs.sort_filelist = True
```

Run recon-all

```python
recon_all = pe.MapNode(
    interface=ReconAll(),
    name='recon_all',
    iterfield=['subject_id', 'T1_files'])
```

(continues on next page)
recon_all.inputs.subject_id = subject_list
if not os.path.exists(subjects_dir):
    os.mkdir(subjects_dir)
recon_all.inputs.subjects_dir = subjects_dir
wf.connect(datasource, 'struct', recon_all, 'T1_files')

Make average subject

average = pe.Node(interface=MakeAverageSubject(), name="average")
average.inputs.subjects_dir = subjects_dir
wf.connect(recon_all, 'subject_id', average, 'subjects_ids')
wf.run("MultiProc", plugin_args={'n_procs': 4})

Example source code
You can download the full source code of this example. This same script is also included in the Nipype source distribution under the examples directory.
This script, smri_fsreconall.py, demonstrates the ability to use the create_reconall_workflow function to create a workflow and then run it on a set of subjects and then make an average subject:

```
python smri_fsreconall.py
```

For an example on how to call FreeSurfer’s reconall script in Nipype see smri_freesurfer.py. Import necessary modules from nipype.

```python
import os
import nipype.pipeline.engine as pe
import nipype.interfaces.io as nio
from nipype.workflows.smri.freesurfer import create_reconall_workflow
from nipype.interfaces.freesurfer.utils import MakeAverageSubject
from nipype.interfaces.utility import IdentityInterface
```

Assign the tutorial directory

```python
tutorial_dir = os.path.abspath('smri_fsreconall_tutorial')
if not os.path.isdir(tutorial_dir):
    os.mkdir(tutorial_dir)
```

Define the workflow directories

```python
subject_list = ['s1', 's3']
data_dir = os.path.abspath('data')
subjects_dir = os.path.join(tutorial_dir, 'subjects_dir')
if not os.path.exists(subjects_dir):
    os.mkdir(subjects_dir)

wf = pe.Workflow(name="l1workflow")
wf.base_dir = os.path.join(tutorial_dir, 'workdir')
```

Create inputspec

```python
inputspec = pe.Node(
    interface=IdentityInterface(["subject_id"], name="inputspec")
inputspec.iterables = ("subject_id", subject_list)
```

Grab data
**datasource = pe.Node(**
  **interface=nio.DataGrabber(**
    **infields=['subject_id'],**
    **outfields=['struct'],**
    **name='datasource')**
**datasource.inputs.base_directory = data_dir**
**datasource.inputs.template = '%s/%s.nii'**
**datasource.inputs.template_args = dict(struct=[['subject_id', 'struct']])**
**datasource.inputs.subject_id = subject_list**
**datasource.inputs.sort_filelist = True**
**wf.connect(inputspec, 'subject_id', datasource, 'subject_id')**

Run recon-all

**recon_all = create_reconall_workflow()**
**recon_all.inputs.inputspec.subjects_dir = subjects_dir**
**wf.connect(datasource, 'struct', recon_all, 'inputspec.T1_files')**
**wf.connect(inputspec, 'subject_id', recon_all, 'inputspec.subject_id')**

Make average subject

**average = pe.JoinNode(**
  **interface=MakeAverageSubject(),**
  **joinsource="inputspec",**
  **joinfield="subjects_ids",**
  **name="average")**
**average.inputs.subjects_dir = subjects_dir**
**wf.connect(recon_all, 'postdatasink_outputspec.subject_id', average, 'subjects_ids')**
**wf.run("MultiProc", plugin_args={"n_procs": 4})**

**Example source code**

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the examples directory.
CHAPTER 40

sMRI: Regional Tessellation and Surface Smoothing

40.1 Introduction

This script, tessellation_tutorial.py, demonstrates the use of create_tessellation_flow from nipype.workflows.smri.freesurfer, and it can be run with:

```
python tessellation_tutorial.py
```

This example requires that the user has Freesurfer installed, and that the Freesurfer directory for ‘fsaverage’ is present.

See also:

ConnectomeViewer  The Connectome Viewer connects Multi-Modal Multi-Scale Neuroimaging and Network Datasets For Analysis and Visualization in Python.

http://www.geuz.org/gmsh/  Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities

http://www.blender.org/  Blender is the free open source 3D content creation suite, available for all major operating systems under the GNU General Public License.

Warning:  This workflow will take several hours to finish entirely, since smoothing the larger cortical surfaces is very time consuming.

40.2 Packages and Data Setup

Import the necessary modules and workflow from nipype.

```
import nipype.pipeline.engine as pe  # pypeline engine
import nipype.interfaces.cmtk as cmtk
import nipype.interfaces.io as nio  # Data i/o
import os
import os.path as op
from nipype.workflows.smri.freesurfer import create_tessellation_flow
```
40.3 Tories

Set the default directory and lookup table (LUT) paths

```python
fs_dir = os.environ['FREESURFER_HOME']
lookup_file = op.join(fs_dir, 'FreeSurferColorLUT.txt')
subjects_dir = op.join(fs_dir, 'subjects/')
output_dir = './tessellate_tutorial'
```

40.4 Inputs

Create the tessellation workflow and set inputs Here we will choose Gifti (gii) as the output format, because we want to able to view the surface in ConnectomeViewer.
In you intend to view the meshes in gmsh or Blender, you should change the workflow creation to use stereolithographic (stl) format.

```python
tessflow = create_tessellation_flow(name='tessflow', out_format='gii')
tessflow.inputs.inputspec.subject_id = 'fsaverage'
tessflow.inputs.inputspec.subjects_dir = subjects_dir
tessflow.inputs.inputspec.lookup_file = lookup_file
```

We also create a conditional node to package the surfaces for ConnectomeViewer. Simply set cff to “False” to ignore this step.

```python
cff = True
if cff:
    cff = pe.Node(interface=cmtk.CFFConverter(), name='cff')
    cff.inputs.out_file = 'Meshes.cff'
```

40.5 Outputs

Create a datasink to organize the smoothed meshes Using regular-expression substitutions we can remove the extraneous folders generated by the mapnode.

```python
datasink = pe.Node(interface=nio.DataSink(), name='datasink')
datasink.inputs.base_directory = 'meshes'
datasink.inputs.regexp_substitutions = [('_smoother[\d]*/', '')]
```

40.6 Execution

Finally, create and run another pipeline that connects the workflow and datasink

```python
tesspipe = pe.Workflow(name='tessellate_tutorial')
tesspipe.base_dir = output_dir
tesspipe.connect([(tessflow, datasink, [('outputspec.meshes', '@meshes.all')])])
```

If the surfaces are to be packaged, this will connect the CFFConverter node to the tessellation and smoothing workflow, as well as to the datasink.

```python
if cff:
    tesspipe.connect([(tessflow, cff, [('outputspec.meshes', 'gifti_surfaces')])])
    tesspipe.connect([(cff, datasink, [('connectome_file', '@cff')])])
```

(continues on next page)
tesspipe.run()

---

**Example source code**

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the examples directory.

---

**Example source code**

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the examples directory.
CHAPTER 41

Workflow from scratch

from builtins import range

import nipype.interfaces.io as nio  # Data i/o
import nipype.interfaces.spm as spm  # spm
import nipype.pipeline.engine as pe  # pypeline engine
import nipype.algorithms.modelgen as model  # model specification
from nipype.interfaces.base import Bunch
import os  # system functions

In the following section, to showcase NiPyPe, we will describe how to create and extend a typical fMRI processing pipeline. We will begin with a basic processing layout and follow with extending it by adding/exchanging different components. Most fMRI pipeline can be divided into two sections - preprocessing and modelling. First one deals with cleaning data from confounds and noise and the second one fits a model based on the experimental design. Preprocessing stage in our first iteration of a pipeline will consist of only two steps: realignment and smoothing. In NiPyPe Every processing step consist of an Interface (which defines how to execute corresponding software) encapsulated in a Node (which defines for example a unique name). For realignment (motion correction achieved by coregistering all volumes to the mean) and smoothing (convolution with 3D Gaussian kernel) we will use SPM implementation. Definition of appropriate nodes can be found in Listing 1 (TODO). Inputs (such as register_to_mean from listing 1) of nodes are accessible through the inputs property. Upon setting any input its type is verified to avoid errors during the execution.

realign = pe.Node(interface=spm.Realign(), name="realign")
realign.inputs.register_to_mean = True

smooth = pe.Node(interface=spm.Smooth(), name="smooth")
smooth.inputs.fwhm = 4

To connect two nodes a Workflow has to be created. connect() method of a Workflow allows to specify which outputs of which Nodes should be connected to which inputs of which Nodes (see Listing 2). By connecting realigned_files output of realign to in_files input of Smooth we have created a simple preprocessing workflow (see Figure TODO).

preprocessing = pe.Workflow(name="preprocessing")
preprocessing.connect(realign, "realigned_files", smooth, "in_files")

Creating a modelling workflow which will define the design, estimate model and contrasts follows the same suite. We will again use SPM implementations. NiPyPe, however, adds extra abstraction layer to model def-
inition which allows using the same definition for many model estimation implementations (for example one from FSL or nipype). Therefore we will need four nodes: SpecifyModel (NiPyPe specific abstraction layer), Level1Design (SPM design definition), ModelEstimate, and ContrastEstimate. The connected modelling Work-flow can be seen on Figure TODO. Model specification supports block, event and sparse designs. Contrasts provided to ContrastEstimate are defined using the same names of regressors as defined in the SpecifyModel.

```
specify_model = pe.Node(interface=model.SpecifyModel(), name="specify_model")
specify_model.inputs.input_units = 'secs'
specify_model.inputs.time_repetition = 3.
specify_model.inputs.high_pass_filter_cutoff = 120
specify_model.inputs.subject_info = [
    Bunch(
        conditions=['Task-Odd', 'Task-Even'],
        onsets=[list(range(15, 240, 60)),
                list(range(45, 240, 60))],
        durations=[[15], [15]])
] * 4

level1design = pe.Node(interface=spm.Level1Design(), name="level1design")
level1design.inputs.bases = {'hrf': {'derivs': [0, 0]}}
level1design.inputs.timing_units = 'secs'
level1design.inputs.interscan_interval = specify_model.inputs.time_repetition

level1estimate = pe.Node(interface=spm.EstimateModel(), name="level1estimate")
level1estimate.inputs.estimation_method = {'Classical': 1}

contrastestimate = pe.Node(interface=spm.EstimateContrast(), name="contrastestimate")
cont1 = ('Task>Baseline', 'T', ['Task-Odd', 'Task-Even'], [0.5, 0.5])
cont2 = ('Task-Odd>Task-Even', 'T', ['Task-Odd', 'Task-Even'], [1, -1])
contrastestimate.inputs.contrasts = [cont1, cont2]

modelling = pe.Workflow(name="modelling")
modelling.connect(specify_model, 'session_info', level1design, 'session_info')
modelling.connect(level1design, 'spm_mat_file', level1estimate, 'spm_mat_file')
modelling.connect(level1estimate, 'beta_images', contrastestimate, 'beta_images')
modelling.connect(level1estimate, 'residual_image', contrastestimate, 'residual_image')
```

Having preprocessing and modelling workflows we need to connect them together, add data grabbing facility and save the results. For this we will create a master Workflow which will host preprocessing and model Workflows as well as DataGrabber and DataSink Nodes. NiPyPe allows connecting Nodes between Workflows. We will use this feature to connect realignment_parameters and smoothed_files to modelling workflow.

```
main_workflow = pe.Workflow(name="main_workflow")
main_workflow.base_dir = "workflow_from_scratch"
main_workflow.connect(preprocessing, "realign.realignment_parameters", modelling, "specify_model.realignment_parameters")
main_workflow.connect(preprocessing, "smooth.smoothed_files", modelling, "specify_model.functional_runs")
```

DataGrabber allows to define flexible search patterns which can be parameterized by user defined inputs (such as subject ID, session etc.). This allows to adapt to a wide range of file layouts. In our case we will parameterize it with subject ID. In this way we will be able to run it for different subjects. We can automate this by iterating over a list of subject IDs, by setting an iterables property on the subject_id input of DataGrabber. Its output will be connected to realignment node from preprocessing workflow.

```
DataGrabber allows to define flexible search patterns which can be parameterized by user defined inputs (such as subject ID, session etc.). This allows to adapt to a wide range of file layouts. In our case we will parameterize it with subject ID. In this way we will be able to run it for different subjects. We can automate this by iterating over a list of subject IDs, by setting an iterables property on the subject_id input of DataGrabber. Its output will be connected to realignment node from preprocessing workflow.
```
datasource = pe.Node(
    interface=nio.DataGrabber(infields=['subject_id'], outfields=['func']),
    name='datasource')
datasource.inputs.base_directory = os.path.abspath('data')
datasource.inputs.template = '%s/%s.nii'
datasource.inputs.template_args = dict(
    func=[['subject_id', ['f3', 'f5', 'f7', 'f10']]])
datasource.inputs.subject_id = 's1'
datasource.inputs.sort_filelist = True

main_workflow.connect(datasource, 'func', preprocessing, 'realign.in_files')

Datasink on the other side provides means to storing selected results to a specified location. It supports automatic creation of folder stricter and regular expression based substitutions. In this example we will store T maps.

datasink = pe.Node(interface=nio.DataSink(), name="datasink")
datasink.inputs.base_directory = os.path.abspath('workflow_from_scratch/output')

main_workflow.connect(modelling, 'contrastestimate.spmT_images', datasink, 'contrasts.@T')

main_workflow.run()
main_workflow.write_graph()

Example source code
You can download the full source code of this example. This same script is also included in the Nipype source distribution under the examples directory.
First let’s go to the directory with the data we’ll be working on and start the interactive python interpreter (with some nipype specific configuration). Note that nipype does not need to be run through ipython - it is just much nicer to do interactive work in it.

```bash
cd $TDPATH
ipython -p nipype
```

For every neuroimaging procedure supported by nipype there exists a wrapper - a small piece of code managing the underlying software (FSL, SPM, AFNI etc.). We call those interfaces. They are standardised so we can hook them up together. Let’s have a look at some of them.

```python
In [1]: import nipype.interfaces.fsl as fsl
In [2]: fsl.BET.help()

Inputs
------
 Mandatory:
  in_file: input file to skull strip

Optional:
  args: Additional parameters to the command
  center: center of gravity in voxels
  environ: Environment variables (default={})
  frac: fractional intensity threshold
  functional: apply to 4D fMRI data
  mutually exclusive: functional, reduce_bias
  mask: create binary mask image
  mesh: generate a vtk mesh brain surface
  no_output: Don't generate segmented output
  out_file: name of output skull stripped image
  outline: create surface outline image
  output_type: FSL output type
  radius: head radius
  reduce_bias: bias field and neck cleanup
  mutually exclusive: functional, reduce_bias
  skull: create skull image
```

(continues on next page)
threshold: apply thresholding to segmented brain image and mask
verticalGradient: vertical gradient in fractional intensity threshold (-1, 1)

Outputs
-------
maskFile: path/name of binary brain mask (if generated)
meshFile: path/name of vtk mesh file (if generated)
outFile: path/name of skullstripped file
outlineFile: path/name of outline file (if generated)

In [3]: import nipype.interfaces.freesurfer as fs

In [4]: fs.Smooth.help()

Inputs
------
Mandatory:
inFile: source volume
num_iters: number of iterations instead of fwhm
mutually exclusive: surface_fwhm
regFile: registers volume to surface anatomical
surface_fwhm: surface FWHM in mm
mutually exclusive: num_iters
requires: regFile

Optional:
args: Additional parameters to the command
environ: Environment variables (default={})
proj_frac: project frac of thickness a long surface normal
mutually exclusive: proj_frac_avg
proj_frac_avg: average a long normal min max delta
mutually exclusive: proj_frac
smoothedFile: output volume
subjects_dir: subjects directory
vol_fwhm: volumesmoothing outside of surface

Outputs
-------
args: Additional parameters to the command
environ: Environment variables
smoothedFile: smoothed input volume
subjects_dir: subjects directory

You can read about all of the interfaces implemented in nipype at our online documentation at http://nipy.sourceforge.net/nipype/documentation.html#documentation. Check it out now.

42.1 Using interfaces

Having interfaces allows us to use third party software (like FSL BET) as function. Look how simple it is.

```python
from __future__ import print_function
from builtins import str

import nipype.interfaces.fsl as fsl
result = fsl.BET(in_file='data/s1/struct.nii').run()
print(result)
```
Running a single program is not much of a breakthrough. Let's run motion correction followed by smoothing (isotropic - in other words not using SUSAN). Notice that in the first line we are setting the output data type for all FSL interfaces.

```python
fsl.FSLCommand.set_default_output_type('NIFTI_GZ')
result1 = fsl.MCFLIRT(in_file='data/s1/f3.nii').run()
result2 = fsl.Smooth(in_file='f3_mcf.nii.gz', fwhm=6).run()
```

### 42.2 Simple workflow

In the previous example we knew that `fsl.MCFLIRT` will produce a file called `f3_mcf.nii.gz` and we have hard coded this as an input to `fsl.Smooth`. This is quite limited, but luckily `nipype` supports joining interfaces in pipelines. This way output of one interface will be used as an input of another without having to hard code anything. Before connecting Interfaces we need to put them into (separate) Nodes and give them unique names. This way every interface will process data in a separate folder.

```python
import nipype.pipeline.engine as pe
import os

motion_correct = pe.Node(interface=fsl.MCFLIRT(in_file=os.path.abspath('data/s1/f3.nii')), name="motion_correct")
smooth = pe.Node(interface=fsl.Smooth(fwhm=6), name="smooth")

motion_correct_and_smooth = pe.Workflow(name="motion_correct_and_smooth")
motion_correct_and_smooth.base_dir = os.path.abspath('.')
# define where will be the root folder for the workflow
motion_correct_and_smooth.connect([(motion_correct, smooth, [('out_file', 'in_file')])])
# we are connecting 'out_file' output of motion_correct to 'in_file' input of smooth
motion_correct_and_smooth.run()
```

### 42.3 Another workflow

Another example of a simple workflow (calculate the mean of fMRI signal and subtract it). This time we'll be assigning inputs after defining the workflow.

```python
calc_mean = pe.Node(interface=fsl.ImageMaths(), name="calc_mean")
calc_mean.inputs.op_string = "-Tmean"
subtract = pe.Node(interface=fsl.ImageMaths(), name="subtract")
subtract.inputs.op_string = "-sub"
demean = pe.Workflow(name="demean")
demean.base_dir = os.path.abspath('.')
demean.connect([(calc_mean, subtract, [('out_file', 'in_file2')])])

demean.inputs.calc_mean.in_file = os.path.abspath('data/s1/f3.nii')
demean.inputs.subtract.in_file = os.path.abspath('data/s1/f3.nii')
demean.run()
```
42.4 Reusing workflows

The beauty of the workflows is that they are reusable. We can just import a workflow made by someone else and feed it with our data.

```python
from fmri_fsl import preproc
preproc.base_dir = os.path.abspath('.
preproc.inputs.inputspec.func = os.path.abspath('data/s1/f3.nii')
preproc.inputs.inputspec.struct = os.path.abspath('data/s1/struct.nii')
preproc.run()
```

... and we can run it again and it won’t actually rerun anything because none of the parameters have changed.

```python
preproc.run()
```

... and we can change a parameter and run it again. Only the dependent nodes are rerun and that too only if the input state has changed.

```python
preproc.inputs.meanfuncmask.frac = 0.5
preproc.run()
```

42.5 Visualizing workflows 1

So what did we run in this precanned workflow

```python
preproc.write_graph()
```

42.6 Datasink

Datasink is a special interface for copying and arranging results.

```python
import nipype.interfaces.io as nio
preproc.inputs.inputspec.func = os.path.abspath('data/s1/f3.nii')
preproc.inputs.inputspec.struct = os.path.abspath('data/s1/struct.nii')
datasink = pe.Node(interface=nio.DataSink(), name='sinker')
preprocess = pe.Workflow(name='preprocout')
preprocess.base_dir = os.path.abspath('.
preprocess.connect([(preproc, datasink, 
('meanfunc2.out_file', 'meanfunc'),
('maskfunc3.out_file', 'funcruns')])])
preprocess.run()
```

42.7 Datagrabber

Datagrabber is (surprise, surprise) an interface for collecting files from hard drive. It is very flexible and supports almost any file organisation of your data you can imagine.

```python
datasource1 = nio.DataGrabber()
datasource1.inputs.template = 'data/s1/f3.nii'
datasource1.inputs.sort_filelist = True
results = datasource1.run()
print(results.outputs)
```

(continues on next page)
datasource2.inputs.template = 'data/s*/f*.nii'
datasource2.inputs.sort_filelist = True
results = datasource2.run()
print(results.outputs)

datasource3 = nio.DataGrabber(infields=['run'])
datasource3.inputs.template = 'data/s1/f%d.nii'
datasource3.inputs.sort_filelist = True
datasource3.inputs.run = [3, 7]
results = datasource3.run()
print(results.outputs)

datasource4 = nio.DataGrabber(infields=['subject_id', 'run'])
datasource4.inputs.template = 'data/%s/f%d.nii'
datasource4.inputs.sort_filelist = True
datasource4.inputs.run = [3, 7]
datasource4.inputs.subject_id = ['s1', 's3']
results = datasource4.run()
print(results.outputs)

42.8 Iterables

Iterables is a special field of the Node class that enables to iterate all workflows/nodes connected to it over some parameters. Here we’ll use it to iterate over two subjects.

import nipype.interfaces.utility as util
infosource = pe.Node(
    interface=util.IdentityInterface(fields=['subject_id']), name="infosource")
infosource.iterables = ('subject_id', ['s1', 's3'])

datasource = pe.Node(
    nio.DataGrabber(infields=['subject_id'], outfields=['func', 'struct']),
    name="datasource")
datasource.inputs.template = '%s/%s.nii'
datasource.inputs.base_directory = os.path.abspath('data')
datasource.inputs.template_args = dict(
    func=[['subject_id', 'f3']], struct=[['subject_id', 'struct']])
datasource.inputs.sort_filelist = True

my_workflow = pe.Workflow(name="my_workflow")
my_workflow.base_dir = os.path.abspath('.')

my_workflow.connect((
    (infosource, datasource, ['subject_id', 'subject_id']),
    (datasource, preproc, [['func', 'inputspec.func'], [
        'struct', 'inputspec.struct']]))
my_workflow.run()

and we can change a node attribute and run it again

smoothnode = my_workflow.get_node('preproc.smooth')
assert (str(smoothnode) == 'preproc.smooth')
smoothnode.iterables = ('fwhm', [5., 10.])
my_workflow.run()
42.9 Visualizing workflows 2

In the case of nested workflows, we might want to look at expanded forms of the workflow.

Example source code
You can download the full source code of this example. This same script is also included in the Nipype source distribution under the examples directory.

• Interfaces
43.1 ACompCor

Anatomical compcor: for inputs and outputs, see CompCor. When the mask provided is an anatomical mask, then CompCor is equivalent to ACompCor.

Inputs:

[Mandatory]
realigned_file: (a pathlike object or string representing an existing file)
    already realigned brain image (4D)

[Optional]
mask_files: (a list of items which are a pathlike object or string representing an existing file)
    One or more mask files that determines ROI (3D). When more that one file is provided `merge_method` or `merge_index` must be provided
merge_method: ('union' or 'intersect' or 'none')
    Merge method if multiple masks are present - `union` uses voxels included in at least one input mask, `intersect` uses only voxels present in all input masks, `none` performs CompCor on each mask individually
    mutually_exclusive: mask_index
    requires: mask_files
mask_index: (a long integer >= 0)
    Position of mask in `mask_files` to use - first is the default.
    mutually_exclusive: merge_method
    requires: mask_files
mask_names: (a list of items which are a unicode string)
    Names for provided masks (for printing into metadata). If provided, it must be as long as the final mask list (after any merge and indexing operations).
components_file: (a unicode string, nipytype default value: components_file.txt)
    Filename to store physiological components
num_components: (a long integer >= 1 or 'all')

(continues on next page)
Number of components to return from the decomposition. If
'tnum_components' is 'all', then all components will be retained.
mutually_exclusive: variance_threshold

variance_threshold: (0.0 < a floating point number < 1.0)
Select the number of components to be returned automatically based
on their ability to explain variance in the dataset.
'tvariance_threshold' is a fractional value between 0 and 1; the
number of components retained will be equal to the minimum number of
components necessary to explain the provided fraction of variance in
the masked time series.
mutually_exclusive: num_components

pre_filter: ('polynomial' or 'cosine' or False, nipype default value:
polynomial)
Detrend time series prior to component extraction
use_regress_poly: (a boolean)
use polynomial regression pre-component extraction
regress_poly_degree: (a long integer >= 1, nipype default value: 1)
the degree polynomial to use
header_prefix: (a unicode string)
the desired header for the output tsv file (one column). If
undefined, will default to "CompCor"
high_pass_cutoff: (a float, nipype default value: 128)
Cutoff (in seconds) for "cosine" pre-filter
repetition_time: (a float)
Repetition time (TR) of series - derived from image header if
unspecified
save_pre_filter: (a boolean or a pathlike object or string
representing a file, nipype default value: False)
Save pre-filter basis as text file
save_metadata: (a boolean or a pathlike object or string representing
a file, nipype default value: False)
Save component metadata as text file
ignore_initial_volumes: (a long integer >= 0, nipype default value:
0)
Number of volumes at start of series to ignore
failure_mode: ('error' or 'NaN', nipype default value: error)
When no components are found or convergence fails, raise an error or
silently return columns of NaNs.

43.1.1 References:
None

43.2 CompCor

Link to code
Interface with core CompCor computation, used in aCompCor and tCompCor

CompCor provides three pre-filter options, all of which include per-voxel mean removal:

- polynomial: Legendre polynomial basis
- cosine: Discrete cosine basis
- False: mean-removal only

In the case of polynomial and cosine filters, a pre-filter file may be saved with a row for each volume/timepoint, and a column for each non-constant regressor. If no non-constant (mean-removal) columns are used, this file may be empty.

If ignore_initial_volumes is set, then the specified number of initial volumes are excluded both from pre-filtering and CompCor component extraction. Each column in the components and pre-filter files are prefixed with zeros for each excluded volume so that the number of rows continues to match the number of volumes in the input file. In addition, for each excluded volume, a column is added to the pre-filter file with a 1 in the corresponding row.

### 43.2.1 Example

```python
>>> ccinterface = CompCor()
>>> ccinterface.inputs.realigned_file = 'functional.nii'
>>> ccinterface.inputs.mask_files = 'mask.nii'
>>> ccinterface.inputs.num_components = 1
>>> ccinterface.inputs.pre_filter = 'polynomial'
>>> ccinterface.inputs.regress_poly_degree = 2
```

**Inputs:**

- **realigned_file**: (a pathlike object or string representing an existing file)
  - already realigned brain image (4D)

- **mask_files**: (a list of items which are a pathlike object or string representing an existing file)
  - One or more mask files that determines ROI (3D). When more than one file is provided, `merge_method` or `merge_index` must be provided
  - **merge_method**: ('union' or 'intersect' or 'none')
    - Merge method if multiple masks are present - 'union' uses voxels included in at least one input mask, 'intersect' uses only voxels present in all input masks, 'none' performs CompCor on each mask individually
    - mutually_exclusive: mask_index
      - Position of mask in `mask_files` to use - first is the default.
      - requires: mask_files

- **mask_index**: (a long integer >= 0)
  - Position of mask in `mask_files` to use - first is the default.
  - requires: mask_files

- **mask_names**: (a list of items which are a unicode string)
  - Names for provided masks (for printing into metadata). If provided, it must be as long as the final mask list (after any merge and indexing operations).

- **components_file**: (a unicode string, nipype default value: `components_file.txt`)
  - Filename to store physiological components

- **num_components**: (a long integer >= 1 or 'all')
  - Number of components to return from the decomposition. If `num_components` is 'all', then all components will be retained.
mutually_exclusive: variance_threshold

variance_threshold: (0.0 < a floating point number < 1.0)
    Select the number of components to be returned automatically based on their ability to explain variance in the dataset. 'variance_threshold' is a fractional value between 0 and 1; the number of components retained will be equal to the minimum number of components necessary to explain the provided fraction of variance in the masked time series.

mutually_exclusive: num_components

pre_filter: ('polynomial' or 'cosine' or False, nipype default value: polynomial)
    Detrend time series prior to component extraction

use_regress_poly: (a boolean)
    use polynomial regression pre-component extraction

regrass_poly_degree: (a long integer >= 1, nipype default value: 1)
    the degree polynomial to use

header_prefix: (a unicode string)
    the desired header for the output tsv file (one column). If undefined, will default to "CompCor"

high_pass_cutoff: (a float, nipype default value: 128)
    Cutoff (in seconds) for "cosine" pre-filter

repetition_time: (a float)
    Repetition time (TR) of series - derived from image header if unspecified

save_pre_filter: (a boolean or a pathlike object or string representing a file, nipype default value: False)
    Save pre-filter basis as text file

save_metadata: (a boolean or a pathlike object or string representing a file, nipype default value: False)
    Save component metadata as text file

ignore_initial_volumes: (a long integer >= 0, nipype default value: 0)
    Number of volumes at start of series to ignore

failure_mode: ('error' or 'NaN', nipype default value: error)
    When no components are found or convergence fails, raise an error or silently return columns of NaNs.

Outputs:

components_file: (a pathlike object or string representing an existing file)
    text file containing the noise components

pre_filter_file: (a pathlike object or string representing a file)
    text file containing high pass filter basis

metadata_file: (a pathlike object or string representing a file)
    text file containing component metadata

43.2.2 References:

None

43.3 ComputeDVARS

Link to code
Computes the DVARS.

Inputs:
[Mandatory]
in_file: (a pathlike object or string representing an existing file)
    functional data, after HMC
in_mask: (a pathlike object or string representing an existing file)
    a brain mask

[Optional]
remove_zerovariance: (a boolean, nipype default value: True)
    remove voxels with zero variance
save_std: (a boolean, nipype default value: True)
    save standardized DVARS
save_nstd: (a boolean, nipype default value: False)
    save non-standardized DVARS
save_vxstd: (a boolean, nipype default value: False)
    save voxel-wise standardized DVARS
save_all: (a boolean, nipype default value: False)
    output all DVARS
series_tr: (a float)
    repetition time in sec.
save_plot: (a boolean, nipype default value: False)
    write DVARS plot
figdpi: (an integer (int or long), nipype default value: 100)
    output dpi for the plot
figsize: (a tuple of the form: (a float, a float), nipype default value: (11.7, 2.3))
    output figure size
figformat: ('png' or 'pdf' or 'svg', nipype default value: png)
    output format for figures
intensity_normalization: (a float, nipype default value: 1000.0)
    Divide value in each voxel at each timepoint by the median
calculated across all voxels and timepoints within the mask (if
specified) and then multiply by the value specified by this parameter.
By using the default (1000) output DVARS will be expressed in x10 %
BOLD units compatible with Power et al. 2012. Set this to 0 to
disable intensity normalization altogether.

Outputs:

out_std: (a pathlike object or string representing an existing file)
    output text file
out_nstd: (a pathlike object or string representing an existing file)
    output text file
out_vxstd: (a pathlike object or string representing an existing file)
    output text file
out_all: (a pathlike object or string representing an existing file)
    output text file
avg_std: (a float)
avg_nstd: (a float)
avg_vxstd: (a float)
fig_std: (a pathlike object or string representing an existing file)
    output DVARS plot
fig_nstd: (a pathlike object or string representing an existing file)
    output DVARS plot
fig_vxstd: (a pathlike object or string representing an existing file)
    output DVARS plot
43.3.1 References:
None

43.4 FramewiseDisplacement

Link to code
Calculate the FD (framewise displacement) as in [Power2012]. This implementation reproduces the calculation in fsl_motion_outliers
Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  motion parameters
parameter_source: ('FSL' or 'AFNI' or 'SPM' or 'FSFAST' or 'NIPY')
  Source of movement parameters

[Optional]
radius: (a float, nipype default value: 50)
  radius in mm to calculate angular FDs, 50mm is the default since it
  is used in Power et al. 2012
out_file: (a pathlike object or string representing a file, nipype
  default value: fd_power_2012.txt)
  output file name
out_figure: (a pathlike object or string representing a file, nipype
  default value: fd_power_2012.pdf)
  output figure name
series_tr: (a float)
  repetition time in sec.
save_plot: (a boolean, nipype default value: False)
  write FD plot
normalize: (a boolean, nipype default value: False)
  calculate FD in mm/s
figdpi: (an integer (int or long), nipype default value: 100)
  output dpi for the FD plot
figsize: (a tuple of the form: (a float, a float), nipype default
  value: (11.7, 2.3))
  output figure size

Outputs:

out_file: (a pathlike object or string representing a file)
  calculated FD per timestep
out_figure: (a pathlike object or string representing a file)
  output image file
fd_average: (a float)
  average FD

43.4.1 References:
None

43.5 NonSteadyStateDetector

Link to code
Returns the number of non-steady state volumes detected at the beginning of the scan.
Inputs:
43.6 TCompCor

Link to code
Interface for tCompCor. Computes a ROI mask based on variance of voxels.

43.6.1 Example

```python
>>> ccinterface = TCompCor()
>>> ccinterface.inputs.realigned_file = 'functional.nii'
>>> ccinterface.inputs.mask_files = 'mask.nii'
>>> ccinterface.inputs.num_components = 1
>>> ccinterface.inputs.pre_filter = 'polynomial'
>>> ccinterface.inputs.regress_poly_degree = 2
>>> ccinterface.inputs.percentile_threshold = .03
```

Inputs:

[Mandatory]
realigned_file: (a pathlike object or string representing an existing file)
   already realigned brain image (4D)

[Optional]
percentile_threshold: (0.0 < a floating point number < 1.0, nipype default value: 0.02)
   the percentile used to select highest-variance voxels, represented by a number between 0 and 1, exclusive. By default, this value is set to .02. That is, the 2% of voxels with the highest variance are used.

mask_files: (a list of items which are a pathlike object or string representing an existing file)
   One or more mask files that determines ROI (3D). When more that one file is provided 'merge_method' or 'merge_index' must be provided

merge_method: ('union' or 'intersect' or 'none')
   Merge method if multiple masks are present - 'union' uses voxels included in at least one input mask, 'intersect' uses only voxels present in all input masks, 'none' performs CompCor on each mask individually
   mutually_exclusive: mask_index
   requires: mask_files

mask_index: (a long integer >= 0)
   Position of mask in 'mask_files' to use - first is the default.
   mutually_exclusive: merge_method
   requires: mask_files

mask_names: (a list of items which are a unicode string)
   Names for provided masks (for printing into metadata). If provided,
   (continues on next page)
it must be as long as the final mask list (after any merge and
indexing operations).
components_file: (a unicode string, nipype default value:
    components_file.txt)
    Filename to store physiological components
num_components: (a long integer >= 1 or 'all')
    Number of components to return from the decomposition. If
    'num_components' is 'all', then all components will be retained.
    mutually_exclusive: variance_threshold
variance_threshold: (0.0 < a floating point number < 1.0)
    Select the number of components to be returned automatically based
    on their ability to explain variance in the dataset.
    'variance_threshold' is a fractional value between 0 and 1; the
    number of components retained will be equal to the minimum number of
    components necessary to explain the provided fraction of variance in
    the masked time series.
    mutually_exclusive: num_components
pre_filter: ('polynomial' or 'cosine' or False, nipype default value:
    polynomial)
    Detrend time series prior to component extraction
use_regress_poly: (a boolean)
    use polynomial regression pre-component extraction
regress_poly_degree: (a long integer >= 1, nipype default value: 1)
    the degree polynomial to use
header_prefix: (a unicode string)
    the desired header for the output tsv file (one column). If
    undefined, will default to "CompCor"
high_pass_cutoff: (a float, nipype default value: 128)
    Cutoff (in seconds) for "cosine" pre-filter
repetition_time: (a float)
    Repetition time (TR) of series - derived from image header if
    unspecified
save_pre_filter: (a boolean or a pathlike object or string
    representing a file, nipype default value: False)
    Save pre-filter basis as text file
save_metadata: (a boolean or a pathlike object or string representing
    a file, nipype default value: False)
    Save component metadata as text file
ignore_initial_volumes: (a long integer >= 0, nipype default value:
    0)
    Number of volumes at start of series to ignore
failure_mode: ('error' or 'NaN', nipype default value: error)
    When no components are found or convergence fails, raise an error or
    silently return columns of NaNs.

Outputs:

high_variance_masks: (a list of items which are a pathlike object or
    string representing an existing file)
    voxels exceeding the variance threshold
components_file: (a pathlike object or string representing an
    existing file)
    text file containing the noise components
pre_filter_file: (a pathlike object or string representing a file)
    text file containing high-pass filter basis
metadata_file: (a pathlike object or string representing a file)
    text file containing component metadata
43.6.2 References:
None

43.7 TSNR

Link to code
Computes the time-course SNR for a time series
Typically you want to run this on a realigned time-series.

43.7.1 Example

```python
>>> tsnr = TSNR()
>>> tsnr.inputs.in_file = 'functional.nii'
>>> res = tsnr.run()  # doctest: +SKIP
```

Inputs:

<table>
<thead>
<tr>
<th>Mandatory</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in_file</code></td>
<td>(a list of items which are a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td></td>
<td>realigned 4D file or a list of 3D files</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optional</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>regress_poly</code></td>
<td>(a long integer &gt;= 1)</td>
</tr>
<tr>
<td></td>
<td>Remove polynomials</td>
</tr>
<tr>
<td><code>tsnr_file</code></td>
<td>(a pathlike object or string representing a file, nipype default value: tsnr.nii.gz)</td>
</tr>
<tr>
<td></td>
<td>output tSNR file</td>
</tr>
<tr>
<td><code>mean_file</code></td>
<td>(a pathlike object or string representing a file, nipype default value: mean.nii.gz)</td>
</tr>
<tr>
<td></td>
<td>output mean file</td>
</tr>
<tr>
<td><code>stddev_file</code></td>
<td>(a pathlike object or string representing a file, nipype default value: stddev.nii.gz)</td>
</tr>
<tr>
<td></td>
<td>output tSNR file</td>
</tr>
<tr>
<td><code>detrended_file</code></td>
<td>(a pathlike object or string representing a file, nipype default value: detrend.nii.gz)</td>
</tr>
<tr>
<td></td>
<td>input file after detrending</td>
</tr>
</tbody>
</table>

Outputs:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tsnr_file</code></td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td></td>
<td>tsnr image file</td>
</tr>
<tr>
<td><code>mean_file</code></td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td></td>
<td>mean image file</td>
</tr>
<tr>
<td><code>stddev_file</code></td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td></td>
<td>std dev image file</td>
</tr>
<tr>
<td><code>detrended_file</code></td>
<td>(a pathlike object or string representing a file)</td>
</tr>
<tr>
<td></td>
<td>detrended input file</td>
</tr>
</tbody>
</table>

43.8 combine_mask_files()

Link to code
Combines input mask files into a single nibabel image
A helper function for CompCor

**mask_files:** a list
One or more binary mask files

**mask_method:** enum (`union`, `intersect`, `none`) determines how to combine masks

**mask_index:** an integer
Determines which file to return (mutually exclusive with **mask_method**)

returns: a list of nibabel images

### 43.9 compute_dvars()

**Link to code**

Compute the DVARS (D referring to temporal derivative of timecourses, VARS referring to RMS variance over voxels) \[Power2012\].

Particularly, the standardized DVARS \[Nichols2013\] are computed.

**Note:** Implementation details
Uses the implementation of the Yule-Walker equations from nitime for the AR (auto-regressive) filtering of the fMRI signal.

```python
param numpy.ndarray func
   functional data, after head-motion-correction.
param numpy.ndarray mask
   a 3D mask of the brain
param bool output_all
   write out all dvars
param str out_file
   a path to which the standardized dvars should be saved.
return the standardized DVARS
```

### 43.10 compute_noise_components()

**Link to code**

Compute the noise components from the imgsseries for each mask

#### 43.10.1 Parameters

**imgs serie s:** nibabel image
Time series data to be decomposed.

**mask_images:** list
List of nibabel images. Time series data from **img_series** is subset according to the spatial extent of each mask, and the subset data is then decomposed using principal component analysis. Masks should be coextensive with either anatomical or spatial noise ROIs.

**components_criterion:** float
Number of noise components to return. If this is a decimal value between 0 and 1, then **create_noise_components** will instead return the smallest number of components necessary to explain the indicated fraction of variance. If **components_criterion** is all, then all components will be returned.

**filter_type:** str
Type of filter to apply to time series before computing noise components.
- `'polynomial'` - Legendre polynomial basis
- `'cosine'` - Discrete cosine (DCT) basis
- `False` - None (mean-removal only)

**failure mode:** str
Action to be taken in the event that any decomposition fails to identify any components. `error` indicates that the routine should raise an exception and exit, while any other value indicates that the routine should return a matrix of NaN values equal in size to the requested decomposition matrix.

**mask_names:** list or None
List of names for each image in **mask_images**. This should be equal in length to **mask_images**, with the ith element of **mask_names** naming the ith element of **mask_images**.

Filter options:

**degree:** int
Order of polynomial used to remove trends from the timeseries

**period_cut:** float
Minimum period (in sec) for DCT high-pass filter

**repetition_time:** float
Time (in sec) between volume acquisitions. This must be defined if the **filter_type** is cosi ne.
43.10.2 Returns

components: numpy array  Numpy array containing the requested set of noise components
basis: numpy array  Numpy array containing the (non-constant) filter regressors

43.11 cosine_filter()

Link to code

43.12 fallback_svd()

Link to code

43.13 is_outlier()

Link to code
Returns a boolean array with True if points are outliers and False otherwise.

param nparray points  an numobservations by numdimensions numpy array of observations
param float thresh  the modified z-score to use as a threshold. Observations with a modified z-score (based on the median absolute deviation) greater than this value will be classified as outliers.

return  A bolean mask, of size numobservations-length array.

Note: References

43.14 plot_confound()

Link to code
A helper function to plot fMRI (functional MRI) confounds.

43.15 regress_poly()

Link to code
Returns data with degree polynomial regressed out.

param bool remove_mean  whether or not demean data (i.e. degree 0),
param int axis  numpy array axes along which regression is performed
44.1 ICC

Link to code

Inputs:

[Mandatory]
subjects_sessions: (a list of items which are a list of items which are a pathlike object or string representing an existing file)

mask: (a pathlike object or string representing an existing file)

Outputs:

icc_map: (a pathlike object or string representing an existing file)

session_var_map: (a pathlike object or string representing an existing file)

subject_var_map: (a pathlike object or string representing an existing file)

44.2 ICC_rep_anova()

Link to code
the data Y are entered as a ‘table’ ie subjects are in rows and repeated measures in columns

One Sample Repeated measure ANOVA

Y = XB + E with X = [Factor / Subjects]
45.1 ComputeMeshWarp

Link to code
Calculates a the vertex-wise warping to get surface2 from surface1. It also reports the average distance of vertices, using the norm specified as input.

Example:

```python
import nipype.algorithms.mesh as m
dist = m.ComputeMeshWarp()
dist.inputs.surface1 = 'surf1.vtk'
dist.inputs.surface2 = 'surf2.vtk'
res = dist.run()
```

Inputs:

- **surface1**: (a pathlike object or string representing an existing file)
  Reference surface (vtk format) to which compute distance.

- **surface2**: (a pathlike object or string representing an existing file)
  Test surface (vtk format) from which compute distance.

- **metric**: ('euclidean' or 'sqeuclidean', nipype default value: euclidean)
  norm used to report distance

- **weighting**: ('none' or 'area', nipype default value: none)
  "none": no weighting is performed, surface": edge distance is weighted by the corresponding surface area

- **out_warp**: (a pathlike object or string representing a file, nipype default value: surfwarp.vtk)
  vtk file based on surface1 and warpings mapping it to surface2

- **out_file**: (a pathlike object or string representing a file, nipype default value: distance.npy)
  numpy file keeping computed distances and weights

Outputs:
45.2 MeshWarpMaths

**Link to code**
Performs the most basic mathematical operations on the warping field defined at each vertex of the input surface. A surface with scalar or vector data can be used as operator for non-uniform operations.

**Example:**

```python
import nipype.algorithms.mesh as m
mmath = m.MeshWarpMaths()
mmath.inputs.in_surf = 'surf1.vtk'
mmath.inputs.operator = 'surf2.vtk'
mmath.inputs.operation = 'mul'
res = mmath.run()
```

**Inputs:**

- **in_surf**: (a pathlike object or string representing an existing file)
  - Input surface in vtk format, with associated warp field as point data (i.e. from ComputeMeshWarp)
- **operator**: (a float or a tuple of the form: (a float, a float, a float) or a pathlike object or string representing an existing file, nipype default value: 1.0)
  - Image, float or tuple of floats to act as operator
- **float_trait**: (a float or a tuple of the form: (a float, a float, a float))
- **operation**: ("sum" or "sub" or "mul" or "div", nipype default value: sum)
  - Operation to be performed
- **out_warp**: (a pathlike object or string representing a file, nipype default value: warp_maths.vtk)
  - VTK file based on in_surf and warpings mapping it to out_file
- **out_file**: (a pathlike object or string representing an existing file, nipype default value: warped_surf.vtk)
  - VTK with surface warped

**Outputs:**

- **out_warp**: (a pathlike object or string representing an existing file)
  - VTK file with the vertex-wise mapping of surface1 to surface2
- **out_file**: (a pathlike object or string representing an existing file)
  - VTK with surface warped

45.3 P2PDistance

**Link to code**
Calculates a point-to-point (p2p) distance between two corresponding VTK-readable meshes or contours.
A point-to-point correspondence between nodes is required

Deprecated since version 1.0-dev: Use ComputeMeshWarp instead.

Inputs:

[Mandatory]

- surface1: (a pathlike object or string representing an existing file)
  Reference surface (vtk format) to which compute distance.
- surface2: (a pathlike object or string representing an existing file)
  Test surface (vtk format) from which compute distance.

[Optional]

- metric: ('euclidean' or 'sqeuclidean', nipype default value: euclidean)
  norm used to report distance
- weighting: ('none' or 'area', nipype default value: none)
  "none": no weighting is performed, surface": edge distance is weighted by the corresponding surface area
- out_warp: (a pathlike object or string representing a file, nipype default value: surfwarp.vtk)
  vtk file based on surface1 and warpings mapping it to surface2
- out_file: (a pathlike object or string representing a file, nipype default value: distance.npy)
  numpy file keeping computed distances and weights

Outputs:

- distance: (a float)
  computed distance
- out_warp: (a pathlike object or string representing an existing file)
  vtk file with the vertex-wise mapping of surface1 to surface2
- out_file: (a pathlike object or string representing an existing file)
  numpy file keeping computed distances and weights

45.4 TVTKBaseInterface

Link to code

A base class for interfaces using VTK

Inputs:

None

Outputs:

None

45.5 WarpPoints

Link to code

Applies a displacement field to a point set given in vtk format. Any discrete deformation field, given in physical coordinates and which volume covers the extent of the vtk point set, is a valid warp file. FSL interfaces are compatible, for instance any field computed with nipype.interfaces.fsl.utils.ConvertWarp.

Example:

```python
from nipype.algorithms.mesh import WarpPoints
wp = WarpPoints()
```
```
wp.inputs.points = 'surf1.vtk'
wp.inputs.warp = 'warpfield.nii'
res = wp.run()
```

**Inputs:**

<table>
<thead>
<tr>
<th>Mandatory</th>
</tr>
</thead>
<tbody>
<tr>
<td>points: (a pathlike object or string representing an existing file) file containing the point set</td>
</tr>
<tr>
<td>warp: (a pathlike object or string representing an existing file) dense deformation field to be applied</td>
</tr>
<tr>
<td>interp: ('cubic' or 'nearest' or 'linear', nipype default value: cubic) interpolation</td>
</tr>
</tbody>
</table>

[Optional]

|out_points: (a pathlike object or string representing a file) the warped point set|

**Outputs:**

|out_points: (a pathlike object or string representing a file) the warped point set|
46.1 Distance

Link to code
Calculates distance between two volumes.

Inputs:

[Mandatory]
volume1: (a pathlike object or string representing an existing file)
Has to have the same dimensions as volume2.
volume2: (a pathlike object or string representing an existing file)
Has to have the same dimensions as volume1.

[Optional]
method: ('eucl_min' or 'eucl_cog' or 'eucl_mean' or 'eucl_wmean' or 'eucl_max', nipype default value: eucl_min)

"eucl_min": Euclidean distance between two closest points
"eucl_cog": mean Euclidian distance between the Center of Gravity of volume1 and CoGs of volume2
"eucl_mean": mean Euclidian minimum distance of all volume2 voxels to volume1
"eucl_wmean": mean Euclidian minimum distance of all volume2 voxels to volume1 weighted by their values
"eucl_max": maximum over minimum Euclidian distances of all volume2 voxels to volume1 (also known as the Hausdorff distance)

mask_volume: (a pathlike object or string representing an existing file)
calculate overlap only within this mask.

Outputs:

distance: (a float)
point1: (an array with shape (3,))
point2: (an array with shape (3,))
histogram: (a pathlike object or string representing a file)
46.2 ErrorMap

Link to code
Calculates the error (distance) map between two input volumes.

46.2.1 Example

```python
>>> errormap = ErrorMap()
>>> errormap.inputs.in_ref = 'cont1.nii'
>>> errormap.inputs.in_tst = 'cont2.nii'
>>> res = errormap.run()  # doctest: +SKIP
```

Inputs:

- **in_ref**: (a pathlike object or string representing an existing file)
  Reference image. Requires the same dimensions as in_tst.
- **in_tst**: (a pathlike object or string representing an existing file)
  Test image. Requires the same dimensions as in_ref.
- **metric**: ("sqeuclidean" or "euclidean", nipype default value: sqeuclidean)
  Error map metric (as implemented in scipy cdist)
- **mask**: (a pathlike object or string representing an existing file)
  Calculate overlap only within this mask.
- **out_map**: (a pathlike object or string representing a file)
  Name for the output file

Outputs:

- **out_map**: (a pathlike object or string representing an existing file)
  Resulting error map
- **distance**: (a float)
  Average distance between volume 1 and 2

46.3 FuzzyOverlap

Link to code
Calculates various overlap measures between two maps, using the fuzzy definition proposed in: Crum et al., Generalized Overlap Measures for Evaluation and Validation in Medical Image Analysis, IEEE Trans. Med. Im. 25(11), pp 1451-1461, Nov. 2006.
in_ref and in_tst are lists of 2/3D images, each element on the list containing one volume fraction map of a class in a fuzzy partition of the domain.

46.3.1 Example

```python
>>> overlap = FuzzyOverlap()
>>> overlap.inputs.in_ref = [ 'ref_class0.nii', 'ref_class1.nii' ]
>>> overlap.inputs.in_tst = [ 'tst_class0.nii', 'tst_class1.nii' ]
>>> overlap.inputs.weighting = 'volume'
>>> res = overlap.run()  # doctest: +SKIP
```

Inputs:
46.4 Overlap

Link to code

Calculates Dice and Jaccard’s overlap measures between two ROI maps. The interface is backwards compatible with the former version in which only binary files were accepted.

The averaged values of overlap indices can be weighted. Volumes now can be reported in $mm^3$, although they are given in voxels to keep backwards compatibility.

46.4.1 Example

```python
>>> overlap = Overlap()
>>> overlap.inputs.volume1 = 'cont1.nii'
>>> overlap.inputs.volume2 = 'cont2.nii'
>>> res = overlap.run() # doctest: +SKIP
```

Inputs:

```python
[Mandatory]
volume1: (a pathlike object or string representing an existing file)
    Has to have the same dimensions as volume2.
volume2: (a pathlike object or string representing an existing file)
    Has to have the same dimensions as volume1.
bg_overlap: (a boolean, nipype default value: False)
    consider zeros as a label
```
vol_units: ('voxel' or 'mm', nipype default value: voxel)

units for volumes

[Optional]

mask_volume: (a pathlike object or string representing an existing file)
calculate overlap only within this mask.

tf_file: (a pathlike object or string representing a file, nipype default value: diff.nii)

weighting: ('none' or 'volume' or 'squared_vol', nipype default value: none)

'none': no class-overlap weighting is performed. 'volume': computed class-overlaps are weighted by class volume
'squared_vol': computed class-overlaps are weighted by the squared volume of the class

Outputs:

jaccard: (a float)
averaged jaccard index
dice: (a float)
averaged dice index

roi_ji: (a list of items which are a float)
the Jaccard index (JI) per ROI

roi_di: (a list of items which are a float)
the Dice index (DI) per ROI

volume_difference: (a float)
averaged volume difference

roi_voldiff: (a list of items which are a float)
volume differences of ROIs

labels: (a list of items which are an integer (int or long))
detected labels

diff_file: (a pathlike object or string representing an existing file)
error map of differences

46.5 Similarity

Link to code
Calculates similarity between two 3D or 4D volumes. Both volumes have to be in the same coordinate system, same space within that coordinate system and with the same voxel dimensions.

Note: This interface is an extension of nipype.interfaces.nipy.utils.Similarity to support 4D files. Requires nipy

46.5.1 Example

```python
>>> from nipype.algorithms.metrics import Similarity
>>> similarity = Similarity()
>>> similarity.inputs.volume1 = 'rc1s1.nii'
>>> similarity.inputs.volume2 = 'rc1s2.nii'
>>> similarity.inputs.mask1 = 'mask.nii'
>>> similarity.inputs.mask2 = 'mask.nii'
>>> similarity.inputs.metric = 'cr'
```
>>> res = similarity.run()  # doctest: +SKIP

Inputs:

[Mandatory]
volume1: (a pathlike object or string representing an existing file)
  3D/4D volume
volume2: (a pathlike object or string representing an existing file)
  3D/4D volume

[Optional]
mask1: (a pathlike object or string representing an existing file)
  3D volume
mask2: (a pathlike object or string representing an existing file)
  3D volume
metric: ('cc' or 'cr' or 'crl1' or 'mi' or 'nmi' or 'slr' or a
  callable value, nipype default value: None)
  str or callable
  Cost-function for assessing image similarity. If a string,
  one of 'cc': correlation coefficient, 'cr': correlation
  ratio, 'crl1': L1-norm based correlation ratio, 'mi': mutual
  information, 'nmi': normalized mutual information, 'slr':
  supervised log-likelihood ratio. If a callable, it should
  take a two-dimensional array representing the image joint
  histogram as an input and return a float.

Outputs:

similarity: (a list of items which are a float)
47.1 AddCSVColumn

Link to code
Short interface to add an extra column and field to a text file

47.1.1 Example

```python
>>> from nipype.algorithms import misc
>>> addcol = misc.AddCSVColumn()
>>> addcol.inputs.in_file = 'degree.csv'
>>> addcol.inputs.extra_column_heading = 'group'
>>> addcol.inputs.extra_field = 'male'
>>> addcol.run()  # doctest: +SKIP
```

Inputs:

[Optional]
out_file: (a pathlike object or string representing a file, nipype
default value: extra_heading.csv)
Output filename for merged CSV file
extra_column_heading: (a unicode string)
New heading to add for the added field.
extra_field: (a unicode string)
New field to add to each row. This is useful for saving the group or
subject ID in the file.

Outputs:

csv_file: (a pathlike object or string representing a file)
Output CSV file containing columns
47.2 AddCSVRow

Link to code
Simple interface to add an extra row to a csv file

Note: Requires pandas

Warning: Multi-platform thread-safe execution is possible with lockfile. Please recall that (1) this module is alpha software; and (2) it should be installed for thread-safe writing. If lockfile is not installed, then the interface is not thread-safe.

47.2.1 Example

```python
>>> from nipype.algorithms import misc
>>> addrow = misc.AddCSVRow()
>>> addrow.inputs.in_file = 'scores.csv'
>>> addrow.inputs.si = 0.74
>>> addrow.inputs.di = 0.93
>>> addrow.inputs.subject_id = 'S400'
>>> addrow.inputs.list_of_values = [ 0.4, 0.7, 0.3 ]
>>> addrow.run() # doctest: +SKIP
```

Inputs:

- in_file: (a pathlike object or string representing a file)
  - Input comma-separated value (CSV) files

- _outputs: (a dictionary with keys which are any value and with values which are any value, nipype default value: {})

Outputs:

- csv_file: (a pathlike object or string representing a file)
  - Output CSV file containing rows

47.3 AddNoise

Link to code
Corrupts with noise the input image

47.3.1 Example

```python
>>> from nipype.algorithms.misc import AddNoise
>>> noise = AddNoise()
>>> noise.inputs.in_file = 'T1.nii'
>>> noise.inputs.in_mask = 'mask.nii'
>>> noise.snr = 30.0
>>> noise.run() # doctest: +SKIP
```

Inputs:
[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  input image that will be corrupted with noise
dist: ('normal' or 'rician', nipype default value: normal)
  desired noise distribution
bg_dist: ('normal' or 'rayleigh', nipype default value: normal)
  desired noise distribution, currently only normal is implemented

[Optional]
in_mask: (a pathlike object or string representing an existing file)
  input mask, voxels outside this mask will be considered background
snr: (a float, nipype default value: 10.0)
  desired output SNR in dB
out_file: (a pathlike object or string representing a file)
  desired output filename

Outputs:

out_file: (a pathlike object or string representing an existing file)
  corrupted image

47.4 CalculateMedian

Link to code
Computes an average of the median across one or more 4D Nifti timeseries

47.4.1 Example

```python
>>> from nipype.algorithms.misc import CalculateMedian
>>> mean = CalculateMedian()
>>> mean.inputs.in_files = 'functional.nii'
>>> mean.run()  # doctest: +SKIP
```

Inputs:

[Optional]
in_files: (a list of items which are a pathlike object or string
  representing an existing file)
median_file: (a unicode string)
  Filename prefix to store median images
median_per_file: (a boolean, nipype default value: False)
  Calculate a median file for each Nifti

Outputs:

median_files: (a list of items which are a pathlike object or string
  representing an existing file)
  One or more median images

47.5 CalculateNormalizedMoments

Link to code
Calculates moments of timeseries.
47.5.1 Example

```python
>>> from nipype.algorithms import misc
>>> skew = misc.CalculateNormalizedMoments()
>>> skew.inputs.moment = 3
>>> skew.inputs.timeseries_file = 'timeseries.txt'
>>> skew.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
timeseries_file: (a pathlike object or string representing an existing file)
  Text file with timeseries in columns and timepoints in rows, whitespace separated
moment: (an integer (int or long))
  Define which moment should be calculated, 3 for skewness, 4 for kurtosis.

Outputs:

moments: (a list of items which are a float)
  Moments

47.6 CreateNifti

Link to code
Creates a nifti volume

Inputs:

[Mandatory]
data_file: (a pathlike object or string representing an existing file)
  ANALYZE img file
header_file: (a pathlike object or string representing an existing file)
  corresponding ANALYZE hdr file

[Optional]
affine: (an array)
  affine transformation array

Outputs:

nifti_file: (a pathlike object or string representing an existing file)

47.7 Distance

Link to code
Calculates distance between two volumes.
Deprecated since version 0.10.0: Use nipype.algorithms.metrics.Distance instead.

Inputs:

[Mandatory]
volume1: (a pathlike object or string representing an existing file)
 Has to have the same dimensions as volume2.

volume2: (a pathlike object or string representing an existing file)
Has to have the same dimensions as volume1.

[Optional]
method: ('eucl_min' or 'eucl_cog' or 'eucl_mean' or 'eucl_wmean' or 'eucl_max', nipype default value: eucl_min)

"eucl_min": Euclidean distance between two closest points
"eucl_cog": mean Euclidian distance between the Center of Gravity of volume1 and CoGs of volume2
"eucl_mean": mean Euclidian minimum distance of all volume2 voxels to volume1
"eucl_wmean": mean Euclidian minimum distance of all volume2 voxels to volume1 weighted by their values
"eucl_max": maximum over minimum Euclidian distances of all volume2 voxels to volume1 (also known as the Hausdorff distance)

mask_volume: (a pathlike object or string representing an existing file)
calculate overlap only within this mask.

Outputs:

distance: (a float)
point1: (an array with shape (3,))
point2: (an array with shape (3,))
histogram: (a pathlike object or string representing a file)

47.8 FuzzyOverlap

Link to code
Calculates various overlap measures between two maps, using a fuzzy definition.
Deprecated since version 0.10.0: Use nipype.algorithms.metrics.FuzzyOverlap instead.

Inputs:

[Mandatory]
in_ref: (a list of items which are a pathlike object or string representing an existing file)
Reference image. Requires the same dimensions as in_tst.
in_tst: (a list of items which are a pathlike object or string representing an existing file)
Test image. Requires the same dimensions as in_ref.

[Optional]
in_mask: (a pathlike object or string representing an existing file)
calculate overlap only within mask
weighting: ('none' or 'volume' or 'squared_vol', nipype default value: none)

'none': no class-overlap weighting is performed.
'volume': computed class-overlaps are weighted by class volume
'squared_vol': computed class-overlaps are weighted by the squared volume of the class

out_file: (a pathlike object or string representing a file, nipype default value: diff.nii)
alternative name for resulting difference-map

Outputs:
jaccard: (a float)
Fuzzy Jaccard Index (fJI), all the classes
dice: (a float)
Fuzzy Dice Index (fDI), all the classes
class_fji: (a list of items which are a float)
Array containing the fJIs of each computed class
class_fdi: (a list of items which are a float)
Array containing the fDIs of each computed class

47.9 Gunzip

Link to code
Gunzip wrapper

```python
>>> from nipype.algorithms.misc import Gunzip
>>> gunzip = Gunzip(in_file='tpms_msk.nii.gz')
>>> res = gunzip.run()
>>> res.outputs.out_file  # doctest: +ELLIPSIS
'.../tpms_msk.nii'
```

```python
>>> os.unlink('tpms_msk.nii')
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)

Outputs:

out_file: (a pathlike object or string representing an existing file)

47.10 Matlab2CSV

Link to code
Simple interface to save the components of a MATLAB .mat file as a text file with comma-separated values (CSVs).
CSV files are easily loaded in R, for use in statistical processing. For further information, see cran.r-project.org/doc/manuals/R-data.pdf

47.10.1 Example

```python
>>> from nipype.algorithms.misc import misc
>>> mat2csv = misc.Matlab2CSV()
>>> mat2csv.inputs.in_file = 'cmatrix.mat'
>>> mat2csv.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
Input MATLAB .mat file

[Optional]
reshape_matrix: (a boolean, nipype default value: True)
The output of this interface is meant for R, so matrices will be reshaped to vectors by default.

**Outputs:**

| csv_files: (a list of items which are a pathlike object or string representing a file) |

### 47.11 MergeCSVFiles

**Link to code**

This interface is designed to facilitate data loading in the R environment. It takes input CSV files and merges them into a single CSV file. If provided, it will also incorporate column heading names into the resulting CSV file.

CSV files are easily loaded in R, for use in statistical processing. For further information, see cran.r-project.org/doc/manuals/R-data.pdf

#### 47.11.1 Example

```python
>>> from nipype.algorithms import misc
>>> mat2csv = misc.MergeCSVFiles()
>>> mat2csv.inputs.in_files = ['degree.mat','clustering.mat']
>>> mat2csv.inputs.column_headings = ['degree','clustering']
>>> mat2csv.run() # doctest: +SKIP
```

**Inputs:**

- [Mandatory]
  - `in_files`: (a list of items which are a pathlike object or string representing an existing file)
    - Input comma-separated value (CSV) files

- [Optional]
  - `out_file`: (a pathlike object or string representing a file, nipype default value: merged.csv)
    - Output filename for merged CSV file
  - `column_headings`: (a list of items which are a unicode string)
    - List of column headings to save in merged CSV file (must be equal to number of input files). If left undefined, these will be pulled from the input filenames.
  - `row_headings`: (a list of items which are a unicode string)
    - List of row headings to save in merged CSV file (must be equal to number of rows in the input files).
  - `row_heading_title`: (a unicode string, nipype default value: label)
    - Column heading for the row headings added
  - `extra_column_heading`: (a unicode string)
    - New heading to add for the added field.
  - `extra_field`: (a unicode string)
    - New field to add to each row. This is useful for saving the group or subject ID in the file.

**Outputs:**

- `csv_file`: (a pathlike object or string representing a file)
  - Output CSV file containing columns
47.12 MergeROIs

Link to code
Splits a 3D image in small chunks to enable parallel processing. ROIs keep time series structure in 4D images.

47.12.1 Example

```python
>>> from nipype.algorithms import misc
>>> rois = misc.MergeROIs()
>>> rois.inputs.in_files = ['roi%02d.nii' % i for i in range(1, 6)]
>>> rois.inputs.in_reference = 'mask.nii'
>>> rois.inputs.in_index = ['roi%02d_idxx.npz' % i for i in range(1, 6)]
>>> rois.run() # doctest: +SKIP
```

Inputs:
[Optional]
- `in_files`: (a list of items which are a pathlike object or string representing an existing file)
- `in_index`: (a list of items which are a pathlike object or string representing an existing file)
- `in_reference`: (a pathlike object or string representing an existing file)

Outputs:
- `merged_file`: (a pathlike object or string representing an existing file)
  - the recomposed file

47.13 ModifyAffine

Link to code
Left multiplies the affine matrix with a specified values. Saves the volume as a nifti file.

Inputs:
[Mandatory]
- `volumes`: (a list of items which are a pathlike object or string representing an existing file)
  - volumes which affine matrices will be modified

[Optional]
- `transformation_matrix`: (an array with shape (4, 4), nipype default value: `<bound method AbstractArray.copy_default_value of <traits.trait_numeric.Array object at 0x7f99411a2128>>`,
  - transformation matrix that will be left multiplied by the affine matrix

Outputs:
- `transformed_volumes`: (a list of items which are a pathlike object or string representing a file)
47.14 NormalizeProbabilityMapSet

Link to code
Returns the input tissue probability maps (tpms, aka volume fractions) normalized to sum up 1.0 at each voxel within the mask.

Note: Please recall this is not a spatial normalization algorithm.

47.14.1 Example

```python
>>> from nipype.algorithms.misc import NormalizeProbabilityMapSet
>>> normalize = NormalizeProbabilityMapSet()
>>> normalize.inputs.in_files = ['tpm_00.nii.gz', 'tpm_01.nii.gz', 'tpm_02.nii.gz']
>>> normalize.inputs.in_mask = 'tpms_msk.nii.gz'
>>> normalize.run()  # doctest: +SKIP
```

Inputs:

<table>
<thead>
<tr>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_files:</td>
<td>(a list of items which are a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>in_mask:</td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td></td>
<td>Masked voxels must sum up 1.0, 0.0 otherwise.</td>
</tr>
</tbody>
</table>

Outputs:

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out_files:</td>
<td>(a list of items which are a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td></td>
<td>normalized maps</td>
</tr>
</tbody>
</table>

47.15 Overlap

Link to code
Calculates various overlap measures between two maps.
Deprecated since version 0.10.0: Use nipype.algorithms.metrics.Overlap instead.

Inputs:

<table>
<thead>
<tr>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>volume1:</td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td></td>
<td>Has to have the same dimensions as volume2.</td>
</tr>
<tr>
<td>volume2:</td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td></td>
<td>Has to have the same dimensions as volume1.</td>
</tr>
<tr>
<td>bg_overlap:</td>
<td>(a boolean, nipype default value: False)</td>
</tr>
<tr>
<td></td>
<td>consider zeros as a label</td>
</tr>
<tr>
<td>vol_units:</td>
<td>('voxel' or 'mm', nipype default value: voxel)</td>
</tr>
<tr>
<td></td>
<td>units for volumes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mask_volume:</td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td></td>
<td>calculate overlap only within this mask.</td>
</tr>
<tr>
<td>out_file:</td>
<td>(a pathlike object or string representing a file, nipype default value: diff.nii)</td>
</tr>
</tbody>
</table>

(continues on next page)
weighting: ('none' or 'volume' or 'squared_vol', nipype default value: none)
   'none': no class-overlap weighting is performed. 'volume': computed class-overlaps are weighted by class volume 'squared_vol': computed class-overlaps are weighted by the squared volume of the class

Outputs:

jaccard: (a float)
   averaged jaccard index
dice: (a float)
   averaged dice index
roi_ji: (a list of items which are a float)
   the Jaccard index (JI) per ROI
roi_di: (a list of items which are a float)
   the Dice index (DI) per ROI
volume_difference: (a float)
   averaged volume difference
roi_voldiff: (a list of items which are a float)
   volume differences of ROIs
labels: (a list of items which are an integer (int or long))
   detected labels
diff_file: (a pathlike object or string representing an existing file)
   error map of differences

47.16 PickAtlas

Link to code
Returns ROI masks given an atlas and a list of labels. Supports dilation and left right masking (assuming the atlas is properly aligned).

Inputs:

[Mandatory]
atlas: (a pathlike object or string representing an existing file)
   Location of the atlas that will be used.
labels: (an integer (int or long) or a list of items which are an integer (int or long))
   Labels of regions that will be included in the mask. Must be compatible with the atlas used.

[Optional]
hemi: ('both' or 'left' or 'right', nipype default value: both)
   Restrict the mask to only one hemisphere: left or right
dilation_size: (an integer (int or long), nipype default value: 0)
   Defines how much the mask will be dilated (expanded in 3D)
output_file: (a pathlike object or string representing a file)
   Where to store the output mask.

Outputs:

mask_file: (a pathlike object or string representing an existing file)
   output mask file
47.17 SimpleThreshold

Link to code
Applies a threshold to input volumes

Inputs:

[Mandatory]
volumes: (a list of items which are a pathlike object or string representing an existing file)
  volumes to be thresholded
threshold: (a float)
  volumes to be thresholded everything below this value will be set to zero

Outputs:

thresholded_volumes: (a list of items which are a pathlike object or string representing an existing file)
  thresholded volumes

47.18 SplitROIs

Link to code
Splits a 3D image in small chunks to enable parallel processing. ROIs keep time series structure in 4D images.

47.18.1 Example

>>> from nipype.algorithms import misc
>>> rois = misc.SplitROIs()
>>> rois.inputs.in_file = 'diffusion.nii'
>>> rois.inputs.in_mask = 'mask.nii'
>>> rois.run()  # doctest: +SKIP

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  file to be splitted

[Optional]
in_mask: (a pathlike object or string representing an existing file)
  only process files inside mask
roi_size: (a tuple of the form: (an integer (int or long), an integer
  (int or long), an integer (int or long)))
  desired ROI size

Outputs:

out_files: (a list of items which are a pathlike object or string representing an existing file)
  the resulting ROIs
out_masks: (a list of items which are a pathlike object or string representing an existing file)
  a mask indicating valid values
out_index: (a list of items which are a pathlike object or string representing an existing file)
  arrays keeping original locations
47.19 TSNR

Link to code
Deprecated since version 0.12.1: Use nipype.algorithms.confounds.TSNR instead

Inputs:

[Mandatory]
in_file: (a list of items which are a pathlike object or string representing an existing file)
realigned 4D file or a list of 3D files

[Optional]
regress_poly: (a long integer >= 1)
Remove polynomials
tsnr_file: (a pathlike object or string representing a file, nipype default value: tsnr.nii.gz)
output tSNR file
mean_file: (a pathlike object or string representing a file, nipype default value: mean.nii.gz)
output mean file
stddev_file: (a pathlike object or string representing a file, nipype default value: stdev.nii.gz)
output std dev image file
detrended_file: (a pathlike object or string representing a file, nipype default value: detrend.nii.gz)
input file after detrending

Outputs:

tsnr_file: (a pathlike object or string representing an existing file)
tsnr image file
mean_file: (a pathlike object or string representing an existing file)
mean image file
stddev_file: (a pathlike object or string representing an existing file)
std dev image file
detrended_file: (a pathlike object or string representing a file)
detrended input file

47.20 calc_moments()

Link to code
Returns nth moment (3 for skewness, 4 for kurtosis) of timeseries (list of values; one per timeseries).
Keyword arguments: timeseries_file – text file with white space separated timepoints in rows

47.21 makefmtlist()

Link to code

47.22 maketypelist()

Link to code
47.23  **matlab2csv()**
Link to code

47.24  **merge_csvs()**
Link to code

47.25  **merge_rois()**
Link to code
Re-builds an image resulting from a parallelized processing

47.26  **normalize_tpms()**
Link to code
Returns the input tissue probability maps (tpms, aka volume fractions) normalized to sum up 1.0 at each voxel within the mask.

47.27  **remove_identical_paths()**
Link to code

47.28  **replaceext()**
Link to code

47.29  **split_rois()**
Link to code
Splits an image in ROIs for parallel processing
CHAPTER 48

48.1 SpecifyModel

Link to code
Makes a model specification compatible with spm/fsl designers.
The subject_info field should contain paradigm information in the form of a Bunch or a list of Bunch. The
Bunch should contain the following information:

[Mandatory]
- conditions : list of names
- onsets : lists of onsets corresponding to each condition
- durations : lists of durations corresponding to each condition. Should be
  left to a single 0 if all events are being modelled as impulses.

[Optional]
- regressor_names : list of str
  list of names corresponding to each column. Should be None if
  automatically assigned.
- regressors : list of lists
  values for each regressor - must correspond to the number of
  volumes in the functional run
- amplitudes : lists of amplitudes for each event. This will be ignored by
  SPM’s Level1Design.

The following two (tmod, pmod) will be ignored by any Level1Design class
other than SPM:

- tmod : lists of conditions that should be temporally modulated. Should
default to None if not being used.
- pmod : list of Bunch corresponding to conditions
  - name : name of parametric modulator
  - param : values of the modulator
  - poly : degree of modulation

Alternatively, you can provide information through event files.
The event files have to be in 1, 2 or 3 column format with the columns corresponding to Onsets, Durations and
Amplitudes and they have to have the name event_name.runXXX... e.g.: Words.run001.txt. The event_name
part will be used to create the condition names.
48.1.1 Examples

```python
>>> from nipype.algorithms import modelgen
>>> from nipype.interfaces.base import Bunch

>>> s = modelgen.SpecifyModel()
>>> s.inputs.input_units = 'secs'
>>> s.inputs.functional_runs = ['functional2.nii', 'functional3.nii']
>>> s.inputs.time_repetition = 6
>>> s.inputs.high_pass_filter_cutoff = 128.
>>> evs_run2 = Bunch(conditions=['cond1'], onsets=[[2, 50, 100, 180]],
   durations=[[1]])
>>> evs_run3 = Bunch(conditions=['cond1'], onsets=[[30, 40, 100, 150]],
   durations=[[1]])
>>> s.inputs.subject_info = [evs_run2, evs_run3]

Using pmod:

>>> evs_run2 = Bunch(conditions=['cond1', 'cond2'], onsets=[[2, 50], [100, 180]],
   durations=[[0], [0]], pmod=[Bunch(name=['amp'], poly=[2], param=[[1, 2]]), None])
>>> evs_run3 = Bunch(conditions=['cond1', 'cond2'], onsets=[[20, 120], [80, 160]],
   durations=[[0], [0]], pmod=[Bunch(name=['amp'], poly=[2], param=[[1, 2]]), None])
>>> s.inputs.subject_info = [evs_run2, evs_run3]
```

Inputs:

[Mandatory]

- **subject_info:** (a list of items which are a Bunch or None)
  - Bunch or List(Bunch) subject-specific condition information. see :ref:`SpecifyModel` or SpecifyModel.__doc__ for details
  - mutually_exclusive: subject_info, event_files, bids_event_file
- **event_files:** (a list of items which are a list of items which are a pathlike object or string representing an existing file)
  - List of event description files 1, 2 or 3 column format corresponding to onsets, durations and amplitudes
  - mutually_exclusive: subject_info, event_files, bids_event_file
- **bids_event_file:** (a list of items which are a pathlike object or string representing an existing file)
  - TSV event file containing common BIDS fields: `onset`, `duration`, and categorization and amplitude columns
  - mutually exclusive: subject_info, event_files, bids_event_file
- **functional_runs:** (a list of items which are a list of items which are a pathlike object or string representing an existing file or a pathlike object or string representing an existing file)
  - Data files for model. List of 4D files or list of list of 3D files per session
- **input_units:** ('secs' or 'scans')
  - Units of event onsets and durations ( secs or scans ). Output units are always in secs
- **high_pass_filter_cutoff:** (a float)
  - High-pass filter cutoff in secs
- **time_repetition:** (a float)
  - Time between the start of one volume to the start of the next image volume.

[Optional]

- **bids_condition_column:** (a unicode string, nipype default value: trial_type)
Column of the file passed to 'bids_event_file' to the unique values of which events will be assigned to regressors

**bids_amplitude_column**: (a unicode string)
Column of the file passed to 'bids_event_file' according to which to assign amplitudes to events

**realignment_parameters**: (a list of items which are a pathlike object or string representing an existing file)
Realignment parameters returned by motion correction algorithm

**parameter_source**: {'SPM' or 'FSL' or 'AFNI' or 'FSFAST' or 'NIPY', nipype default value: SPM}
Source of motion parameters

**outlier_files**: (a list of items which are a pathlike object or string representing an existing file)
Files containing scan outlier indices that should be tossed

---

**Outputs:**

<table>
<thead>
<tr>
<th>session_info: (any value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session info for level1designs</td>
</tr>
</tbody>
</table>

---

**48.2 SpecifySPMModel**

**Link to code**

Add SPM specific options to SpecifyModel

**adds:**

- concatenate_runs
- output_units

**48.2.1 Examples**

```python
>>> from nipype.algorithms import modelgen
>>> from nipype.interfaces.base import Bunch

>>> s = modelgen.SpecifySPMModel()
>>> s.inputs.input_units = 'secs'
>>> s.inputs.output_units = 'scans'
>>> s.inputs.high_pass_filter_cutoff = 128.
>>> s.inputs.functional_runs = ['functional12.nii', 'functional13.nii']
>>> s.inputs.time_repetition = 6
>>> s.inputs.concatenate_runs = True
>>> evs_run2 = Bunch(conditions=['cond1'], onsets=[[2, 50, 100, 180]], durations=[[1]])
>>> evs_run3 = Bunch(conditions=['cond1'], onsets=[[30, 40, 100, 150]], durations=[[1]])
>>> s.inputs.subject_info = [evs_run2, evs_run3]
```

**Inputs:**

- **subject_info**: (a list of items which are a Bunch or None)
  
  Bunch or List(Bunch) subject-specific condition information. see :ref:`SpecifyModel` or SpecifyModel.__doc__ for details
  
  mutually_exclusive: subject_info, event_files, bids_event_file

- **event_files**: (a list of items which are a list of items which are a pathlike object or string representing an existing file)
  
  List of event description files 1, 2 or 3 column format

(continues on next page)
corresponding to onsets, durations and amplitudes

mutually_exclusive: subject_info, event_files, bids_event_file

bids_event_file: (a list of items which are a pathlike object or
string representing an existing file)
TSV event file containing common BIDS fields: `onset`, `duration`,
and categorization and amplitude columns

mutually_exclusive: subject_info, event_files, bids_event_file

functional_runs: (a list of items which are a list of items which are
a pathlike object or string representing an existing file or a
pathlike object or string representing an existing file)
Data files for model. List of 4D files or list of list of 3D files
per session

input_units: (`secs` or `scans`)
Units of event onsets and durations (secs or scans). Output units
are always in secs

high_pass_filter_cutoff: (a float)
High-pass filter cutoff in secs

time_repetition: (a float)
Time between the start of one volume to the start of the next image
volume.

[Optional]
concatenate_runs: (a boolean, nipype default value: False)
Concatenate all runs to look like a single session.

output_units: (`secs` or `scans`, nipype default value: `secs`)
Units of design event onsets and durations (secs or scans)

bids_condition_column: (a unicode string, nipype default value:
trial_type)
Column of the file passed to `bids_event_file` to the unique values
of which events will be assigned to regressors

bids_amplitude_column: (a unicode string)
Column of the file passed to `bids_event_file` according to which to
assign amplitudes to events

realignment_parameters: (a list of items which are a pathlike object
or string representing an existing file)
Realignment parameters returned by motion correction algorithm

parameter_source: (`SPM` or `FSL` or `AFNI` or `FSFAST` or `NIPY`,
nipype default value: SPM)
Source of motion parameters

outlier_files: (a list of items which are a pathlike object or string
representing an existing file)
Files containing scan outlier indices that should be tossed

Outputs:

session_info: (any value)
  Session info for level1 designs

48.3 SpecifySparseModel

Link to code
Specify a sparse model that is compatible with spm/fsl designers
48.3.1 References


48.3.2 Examples

```python
>>> from nipype.algorithms import modelgen
>>> from nipype.interfaces.base import Bunch
>>> s = modelgen.SpecifySparseModel()
>>> s.inputs.input_units = 'secs'
>>> s.inputs.functional_runs = ['functional2.nii', 'functional3.nii']
>>> s.inputs.time_repetition = 6
>>> s.inputs.time_acquisition = 2
>>> s.inputs.high_pass_filter_cutoff = 128.
>>> s.inputs.model_hrf = True
>>> evs_run2 = Bunch(conditions=['cond1'], onsets=[[2, 50, 100, 180]], durations=[[1]])
>>> evs_run3 = Bunch(conditions=['cond1'], onsets=[[30, 40, 100, 150]], durations=[[1]])
>>> s.inputs.subject_info = [evs_run2, evs_run3]
```

Inputs:

[Mandatory]

time_acquisition: (a float)
   Time in seconds to acquire a single image volume

subject_info: (a list of items which are a Bunch or None)
   Bunch or List(Bunch) subject-specific condition information. see :ref:`SpecifyModel` or SpecifyModel.__doc__ for details
   mutually_exclusive: subject_info, event_files, bids_event_file

event_files: (a list of items which are a list of items which are a pathlike object or string representing an existing file)
   list of event description files 1, 2 or 3 column format corresponding to onsets, durations and amplitudes
   mutually_exclusive: subject_info, event_files, bids_event_file

bids_event_file: (a list of items which are a pathlike object or string representing an existing file)
   TSV event file containing common BIDS fields: `onset`, `duration`, and categorization and amplitude columns
   mutually_exclusive: subject_info, event_files, bids_event_file

functional_runs: (a list of items which are a list of items which are a pathlike object or string representing an existing file or a pathlike object or string representing an existing file)
   Data files for model. List of 4D files or list of list of 3D files per session

input_units: ('secs' or 'scans')
   Units of event onsets and durations (secs or scans). Output units are always in secs

high_pass_filter_cutoff: (a float)
   High-pass filter cutoff in secs

time_repetition: (a float)
   Time between the start of one volume to the start of the next image volume.

[Optional]

volumes_in_cluster: (a long integer >= 1, nipype default value: 1)
   Number of scan volumes in a cluster

(continues on next page)
model_hrf: (a boolean)
  Model sparse events with hrf
stimuli_as_impulses: (a boolean, nipype default value: True)
  Treat each stimulus to be impulse-like
use_temporal_deriv: (a boolean)
  Create a temporal derivative in addition to regular regressor
  requires: model_hrf
scale_regressors: (a boolean, nipype default value: True)
  Scale regressors by the peak
scan_onset: (a float, nipype default value: 0.0)
  Start of scanning relative to onset of run in secs
save_plot: (a boolean)
  Save plot of sparse design calculation (requires matplotlib)
bids_condition_column: (a unicode string, nipype default value: trial_type)
  Column of the file passed to `bids_event_file` to the unique values
  of which events will be assigned to regressors
bids_amplitude_column: (a unicode string)
  Column of the file passed to `bids_event_file` according to which to
  assign amplitudes to events
realignment_parameters: (a list of items which are a pathlike object
  or string representing an existing file)
  Realignment parameters returned by motion correction algorithm
parameter_source: (‘SPM’ or ‘FSL’ or ‘AFNI’ or ‘FSLFAST’ or ‘NIPY’),
  nipype default value: SPM
  Source of motion parameters
outlier_files: (a list of items which are a pathlike object or string
  representing an existing file)
  Files containing scan outlier indices that should be tossed

Outputs:
sparse_png_file: (a pathlike object or string representing a file)
  PNG file showing sparse design
sparse_svg_file: (a pathlike object or string representing a file)
  SVG file showing sparse design
session_info: (any value)
  Session info for level1designs

48.4 bids_gen_info()

Link to code
Generate subject_info structure from a list of BIDS .tsv event files.

48.4.1 Parameters

bids_event_files [list of str] Filenames of BIDS .tsv event files containing columns including: ‘onset’, ‘duration’, and ‘trial_type’ or the condition_column value.
condition_column [str] Column of files in bids_event_files based on the values of which events will be sorted into different regressors
amplitude_column [str] Column of files in bids_event_files based on the values of which to apply amplitudes to events. If unspecified, all events will be represented with an amplitude of 1.

48.4.2 Returns
list of Bunch
48.5 **gcd()**

**Link to code**

Returns the greatest common divisor of two integers

uses Euclid's algorithm

```python
>>> gcd(4, 5)
1
>>> gcd(4, 8)
4
>>> gcd(22, 55)
11
```

48.6 **gen_info()**

**Link to code**

Generate subject_info structure from a list of event files

48.7 **orth()**

**Link to code**

Orthogonalize y_in with respect to x_in.

```python
>>> orth_expected = np.array([1.7142857142857144, 0.42857142857142883,
                           -0.85714285714285676])
>>> err = np.abs(np.array(orth([1, 2, 3],[4, 5, 6]) - orth_expected))
>>> all(err < np.finfo(float).eps)
True
```

48.8 **scale_timings()**

**Link to code**

Scales timings given input and output units (scans secs)

48.8.1 Parameters

- `timelist`: list of times to scale
- `input_units`: 'secs' or 'scans'
- `output_units`: Ibid.
- `time_repetition`: float in seconds

48.9 **spm_hrf()**

**Link to code**

Python implementation of spm_hrf

see spm_hrf for implementation details

% RT - scan repeat time % p - parameters of the response function (two gamma % functions) % defaults (seconds) % p(0) - delay of response (relative to onset) 6 % p(1) - delay of undershoot (relative to onset) 16 % p(2) - dispersion of response 1 % p(3) - dispersion of undershoot 1 % p(4) - ratio of response to undershoot 6 % p(5) - onset (seconds) 0 % p(6) - length of kernel (seconds) 32 ~ hrf - hemodynamic response function % p - parameters of the response function

the following code using scipy.stats.distributions.gamma doesn’t return the same result as the spm_Gpdf function
hrf = gamma.pdf(u, p[0]/p[2], scale=dt/p[2]) -
    gamma.pdf(u, p[1]/p[3], scale=dt/p[3])/p[4]

>>> print(spm_hrf(2))
[ 0.00000000e+00  8.65660810e-02  3.74888236e-01  3.84923382e-01
  2.16117316e-01  7.68695653e-02  1.62017720e-03 -3.06078117e-02
 -3.73060781e-02 -3.08373716e-02 -2.05161334e-02 -1.16441637e-02
 -5.82063147e-03 -2.61854250e-03 -1.07732374e-03 -4.10443522e-04
 -1.46257507e-04]
49.1 ArtifactDetect

Link to code
Detects outliers in a functional imaging series
Uses intensity and motion parameters to infer outliers. If `use_norm` is True, it computes the movement of the center of each face a cuboid centered around the head and returns the maximal movement across the centers. If you wish to use individual thresholds instead, import `Undefined` from `nipype.interfaces.base` and set 

```python
... .inputs.use_norm = Undefined
```

### 49.1.1 Examples

```python
>>> ad = ArtifactDetect()
>>> ad.inputs.realigned_files = 'functional.nii'
>>> ad.inputs.realignment_parameters = 'functional.par'
>>> ad.inputs.parameter_source = 'FSL'
>>> ad.inputs.norm_threshold = 1
>>> ad.inputs.use_differences = [True, False]
>>> ad.inputs.zintensity_threshold = 3
>>> ad.run()  # doctest: +SKIP
```

**Inputs:**

[Mandatory]
- `realigned_files`: (a list of items which are a pathlike object or string representing an existing file)
  Names of realigned functional data files
- `realignment_parameters`: (a list of items which are a pathlike object or string representing an existing file)
  Names of realignment parameters corresponding to the functional data files
- `parameter_source`: ('SPM' or 'FSL' or 'AFNI' or 'NiPy' or 'FSFAST')
  Source of movement parameters
- `norm_threshold`: (a float)
  Threshold to use to detect motion-related outliers when composite motion is being used
mutually_exclusive: rotation_threshold, translation_threshold
rotation_threshold: (a float)
    Threshold (in radians) to use to detect rotation-related outliers
mutually_exclusive: norm_threshold
translation_threshold: (a float)
    Threshold (in mm) to use to detect translation-related outliers
mutually_exclusive: norm_threshold
zintensity_threshold: (a float)
    Intensity Z-threshold use to detection images that deviate from the mean
mask_type: ('spm_global' or 'file' or 'thresh')
    Type of mask that should be used to mask the functional data.
    *spm_global* uses an spm_global like calculation to determine the brain mask. *file* specifies a brain mask file (should be an image file consisting of 0s and 1s). *thresh* specifies a threshold to use. By default all voxels are used, unless one of these mask types are defined

[Optional]
use_differences: (a list of items which are a bool or None, nipype default value: [True, False])
    Use differences between successive motion (first element) and intensity parameter (second element) estimates in order to determine outliers. (default is [True, False])
use_norm: (a boolean, nipype default value: True)
    Uses a composite of the motion parameters in order to determine outliers.
    requires: norm_threshold
mask_file: (a pathlike object or string representing an existing file)
    Mask file to be used if mask_type is 'file'.
mask_threshold: (a float)
    Mask threshold to be used if mask_type is 'thresh'.
intersect_mask: (a boolean, nipype default value: True)
    Intersect the masks when computed from spm_global.
save_plot: (a boolean, nipype default value: True)
    save plots containing outliers
plot_type: ('png' or 'svg' or 'eps' or 'pdf', nipype default value: png)
    file type of the outlier plot
bound_by_brainmask: (a boolean, nipype default value: False)
    use the brain mask to determine bounding box for composite norm
    (works for SPM and Nipy - currently inaccurate for FSL, AFNI)
global_threshold: (a float, nipype default value: 8.0)
    use this threshold when mask type equal's spm_global

Outputs:

outlier_files: (a list of items which are a pathlike object or string representing an existing file)
    One file for each functional run containing a list of 0-based indices corresponding to outlier volumes
intensity_files: (a list of items which are a pathlike object or string representing an existing file)
    One file for each functional run containing the global intensity values determined from the brainmask
norm_files: (a list of items which are a pathlike object or string
representing a file)
One file for each functional run containing the composite norm

statistic_files: (a list of items which are a pathlike object or
string representing an existing file)
One file for each functional run containing information about the
different types of artifacts and if design info is provided then
details of stimulus correlated motion and a listing or artifacts by
event type.

plot_files: (a list of items which are a pathlike object or string
representing a file)
One image file for each functional run containing the detected
outliers

mask_files: (a list of items which are a pathlike object or string
representing a file)
One image file for each functional run containing the mask used for
global signal calculation

displacement_files: (a list of items which are a pathlike object or
string representing a file)
One image file for each functional run containing the voxel
displacement timeseries

49.2 StimulusCorrelation

Link to code
Determines if stimuli are correlated with motion or intensity parameters.
Currently this class supports an SPM generated design matrix and requires intensity parameters. This implies
that one must run ArtifactDetect and Level1Design prior to running this or provide an SPM.mat file and intensity
parameters through some other means.

49.2.1 Examples

```python
>>> sc = StimulusCorrelation()
>>> sc.inputs.realignment_parameters = 'functional.par'
>>> sc.inputs.intensity_values = 'functional.rms'
>>> sc.inputs.spm_mat_file = 'SPM.mat'
>>> sc.inputs.concatenated_design = False
>>> sc.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
realignment_parameters: (a list of items which are a pathlike object
or string representing an existing file)
Names of realignment parameters corresponding to the functional data
files

intensity_values: (a list of items which are a pathlike object or
string representing an existing file)
Name of file containing intensity values

spm_mat_file: (a pathlike object or string representing an existing
file)
SPM mat file (use pre-estimate SPM.mat file)

concatenated_design: (a boolean)
state if the design matrix contains concatenated sessions

Outputs:
**stimcorr_files**: (a list of items which are a pathlike object or string representing an existing file)

List of files containing correlation values
50.1 ActivationCount

Link to code
Calculate a simple Activation Count Maps
Adapted from: https://github.com/poldracklab/CNP_task_analysis/blob/61c27f5992db9d880884f8fcee73e6957db8af/CNP_2nd_level_ACM.py

Inputs:

```
[Mandatory]
in_files: (a list of items which are a pathlike object or string representing an existing file)
          input file, generally a list of z-stat maps
threshold: (a float)
          binarization threshold. E.g. a threshold of 1.65 corresponds to a two-sided Z-test of p<.10
```

Outputs:

```
out_file: (a pathlike object or string representing an existing file)
          output activation count map
acm_pos: (a pathlike object or string representing an existing file)
          positive activation count map
acm_neg: (a pathlike object or string representing an existing file)
          negative activation count map
```
51.1 interfaces.afni.base

51.1.1 AFNIPythonCommand

Link to code
A subtype of AFNI command line for Python scripts.

Inputs:

- **num_threads**: (an integer (int or long), nipype default value: 1)
  - set number of threads
- **outputtype**: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
  - AFNI output filetype
- **out_file**: (a pathlike object or string representing a file)
  - output image file name
  - argument: ``-prefix %s``
- **args**: (a unicode string)
  - Additional parameters to the command
  - argument: ``%s``
- **environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  - Environment variables

Outputs:

**None**

References:

None None
51.2 interfaces.afni.model

51.2.1 Deconvolve

Link to code
Wraps the executable command 3dDeconvolve.
Performs OLS regression given a 4D neuroimage file and stimulus timings
For complete details, see the 3dDeconvolve Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> deconvolve = afni.Deconvolve()
>>> deconvolve.inputs.in_files = ['functional.nii', 'functional2.nii']
>>> deconvolve.inputs.out_file = 'output.nii'
>>> deconvolve.inputs.x1D = 'output.1D'
>>> stim_times = [(1, 'timeseries.txt', 'SPMG1(4)')]
>>> deconvolve.inputs.stim_times = stim_times
>>> deconvolve.inputs.stim_label = [(1, 'Houses')]
>>> deconvolve.inputs.gltsym = ['SYM: +Houses']
>>> deconvolve.inputs.glt_label = [(1, 'Houses')]
>>> deconvolve.cmdline
"3dDeconvolve -input functional.nii functional2.nii -bucket output.nii -x1D output.1D -num_stimts 1 -stim_times 1 timeseries.txt 'SPMG1(4)' -stim_label 1 'Houses' -num_glt 1 -gltsym 'SYM: +Houses' -glt_label 1 'Houses"
>>> res = deconvolve.run()  # doctest: +SKIP
```

Inputs:

[Optional]
in_files: (a list of items which are a pathlike object or string representing an existing file)
filename of 3D+time input datasets. More than one filename can be
given and the datasets will be auto-catenated in time. You can input
a 1D time series file here, but the time axis should run along the
ROW direction, not the COLUMN direction as in the 'input1D' option.
argument: ``-input %s``, position: 1
sat: (a boolean)
check the dataset time series for initial saturation transients,
which should normally have been excised before data analysis.
argument: ``-sat``
mutually_exclusive: trans
trans: (a boolean)
check the dataset time series for initial saturation transients,
which should normally have been excised before data analysis.
argument: ``-trans``
mutually_exclusive: sat
noblock: (a boolean)
normally, if you input multiple datasets with 'input', then the separate datasets are taken to be separate image runs that get separate baseline models. Use this options if you want to have the program consider these to be all one big run.* If any of the input dataset has only 1 sub-brick, then this option is automatically invoked!* If the auto-catenation feature isn't used, then this option has no effect, no how, no way.
argument: ``-noblock``
force_TR: (a float)
use this value instead of the TR in the 'input' dataset. (It's better to fix the input using Refit.)
argument: `'--force_TR %f'`, position: 0

input1D: (a pathlike object or string representing an existing file)
filename of single (fMRI) .1D time series where time runs down the column.
argument: `'--input1D %s'`

TR_1D: (a float)
TR to use with 'input1D'. This option has no effect if you do not also use 'input1D'.
argument: `'--TR_1D %f'`

legendre: (a boolean)
use Legendre polynomials for null hypothesis (baseline model)
argument: `'--legendre'`

nolegendre: (a boolean)
use power polynomials for null hypotheses. Don't do this unless you are crazy!
argument: `'--nolegendre'`

donodmbase: (a boolean)
don't de-mean baseline time series
argument: `'--nodmbase'`
dmbase: (a boolean)
de-mean baseline time series (default if 'polort' >= 0)
argument: `'--dmbase'`

svd: (a boolean)
use SVD instead of Gaussian elimination (default)
argument: `'--svd'`

nosvd: (a boolean)
use Gaussian elimination instead of SVD
argument: `'--nosvd'`

rmsmin: (a float)
minimum rms error to reject reduced model (default = 0; don't use this option normally!)
argument: `'--rmsmin %f'`

nocond: (a boolean)
DON'T calculate matrix condition number
argument: `'--nocond'`

singvals: (a boolean)
print out the matrix singular values
argument: `'--singvals'`

goforit: (an integer (int or long))
use this to proceed even if the matrix has bad problems (e.g., duplicate columns, large condition number, etc.).
argument: `'--GOFORIT %i'`

allzero_OK: (a boolean)
don't consider all zero matrix columns to be the type of error that 'goforit' is needed to ignore.
argument: `'--allzero_OK'`

dname: (a tuple of the form: (a unicode string, a unicode string))
set environmental variable to provided value
argument: `'-%D%s=%s'`

mask: (a pathlike object or string representing an existing file)
filename of 3D mask dataset; only data time series from within the mask will be analyzed; results for voxels outside the mask will be set to zero.
argument: `'--mask %s'`

automask: (a boolean)
build a mask automatically from input data (will be slow for long
time series datasets)
argument: `-automask`

STATmask: (a pathlike object or string representing an existing file)
built a mask from provided file, and use this mask for the purpose
of reporting truncation-to float issues AND for computing the FDR
curves. The actual results ARE not masked with this option (only
with 'mask' or 'automask' options).
argument: `-STATmask %s`

censor: (a pathlike object or string representing an existing file)
filename of censor .1D time series. This is a file of 1s and 0s,
indicating which time points are to be included (1) and which are to
be excluded (0).
argument: `-censor %s`

polort: (an integer (int or long))
degree of polynomial corresponding to the null hypothesis [default: 1]
argument: `-polort %d`

 ortvec: (a tuple of the form: (a pathlike object or string
 representing an existing file, a unicode string))
this option lets you input a rectangular array of 1 or more baseline
vectors from a file. This method is a fast way to include a lot of
baseline regressors in one step.
argument: `-ortvec %s %s`

 x1D: (a pathlike object or string representing a file)
specify name for saved X matrix
argument: `-x1D %s`

 x1D-stop: (a boolean)
stop running after writing .xmat.1D file
argument: `-x1D_stop`

cbucket: (a unicode string)
Name for dataset in which to save the regression coefficients (no
statistics). This dataset will be used in a -xrestore run [not yet
implemented] instead of the bucket dataset, if possible.
argument: `-cbucket %s`

out_file: (a pathlike object or string representing a file)
output statistics file
argument: `-bucket %s`

num_threads: (an integer (int or long))
run the program with provided number of sub-processes
argument: `-jobs %d`

fout: (a boolean)
output F-statistic for each stimulus
argument: `-fout`

rout: (a boolean)
output the R^2 statistic for each stimulus
argument: `-rout`

tout: (a boolean)
output the T-statistic for each stimulus
argument: `-tout`

vout: (a boolean)
output the sample variance (MSE) for each stimulus
argument: `-vout`

nofdr: (a boolean)
Don't compute the statistic-vs-FDR curves for the bucket dataset.
argument: `-noFDR`

global_times: (a boolean)
use global timing for stimulus timing files
argument: `'--global_times'`
mutually_exclusive: local_times

local_times: (a boolean)
use local timing for stimulus timing files
argument: `'--local_times'`
mutually_exclusive: global_times

num_stimts: (an integer (int or long))
number of stimulus timing files
argument: `'--num_stimts %d'`, position: -6

stim_times: (a list of items which are a tuple of the form: (an integer (int or long), a pathlike object or string representing an existing file, a unicode string))
generate a response model from a set of stimulus times given in file.
argument: `'--stim_times %d %s ...'`, position: -5

stim_label: (a list of items which are a tuple of the form: (an integer (int or long), a unicode string))
label for kth input stimulus (e.g., Label1)
argument: `'--stim_label %d %s ...'`, position: -4
requires: stim_times

stim_times_subtract: (a float)
this option means to subtract specified seconds from each time encountered in any 'stim_times' option. The purpose of this option is to make it simple to adjust timing files for the removal of images from the start of each imaging run.
argument: `'--stim_times_subtract %f'`

num_glt: (an integer (int or long))
number of general linear tests (i.e., contrasts)
argument: `'--num_glt %d'`, position: -3

gltsym: (a list of items which are a unicode string)
general linear tests (i.e., contrasts) using symbolic conventions (e.g., '+Label1 -Label2')
argument: `'--gltsym SYM: %s ...'`, position: -2

glt_label: (a list of items which are a tuple of the form: (an integer (int or long), a unicode string))
general linear test (i.e., contrast) labels
argument: `'--glt_label %d %s ...'`, position: -1
requires: gltsym

outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
AFNI output filetype

args: (a unicode string)
Additional parameters to the command
argument: `'%s'`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
output statistics file

reml_script: (a pathlike object or string representing an existing file)
automatical generated script to run 3dREMLfit

x1D: (a pathlike object or string representing an existing file)
save out X matrix
cbucket: (a pathlike object or string representing a file)
output regression coefficients file (if generated)

References:
None

51.2.2 Remlfit

Link to code
Wraps the executable command `3dREMLfit`.
Performs Generalized least squares time series fit with Restricted Maximum Likelihood (REML) estimation of the temporal auto-correlation structure.
For complete details, see the `3dREMLfit` Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> remlfit = afni.Remlfit()
>>> remlfit.inputs.in_files = ['functional.nii', 'functional2.nii']
>>> remlfit.inputs.out_file = 'output.nii'
>>> remlfit.inputs.matrix = 'output.1D'
>>> remlfit.inputs.gltsym = [('SYM: +Lab1 -Lab2', 'TestSYM'), ('timeseries.txt', 'TestFile')]
>>> remlfit.cmdline
'3dREMLfit -gltsym "SYM: +Lab1 -Lab2" TestSYM -gltsym "timeseries.txt" TestFile -input "functional.nii functional2.nii" -matrix output.1D -Rbuck output.nii'
>>> res = remlfit.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_files: (a list of items which are a pathlike object or string representing an existing file)
  Read time series dataset
  argument: `''-input "%s"'`
matrix: (a pathlike object or string representing a file)
  the design matrix file, which should have been output from
  Deconvolve via the 'x1D' option
  argument: `''-matrix %s'`

[Optional]
polort: (an integer (int or long))
  if no 'matrix' option is given, AND no 'matim' option, create a
  matrix with Legendre polynomial regressors up to the specified order.
  The default value is 0, which produces a matrix with a single column
  of all ones
  argument: `''-polort %d'`
  mutually_exclusive: matrix
matim: (a pathlike object or string representing a file)
  read a standard file as the matrix. You can use only Col as a name
  in GLTs with these nonstandard matrix input methods, since the other
  names come from the 'matrix' file. These mutually exclusive options
  are ignored if 'matrix' is used.
argument: ``-matim %s``
mutually_exclusive: matrix

mask: (a pathlike object or string representing an existing file)
filename of 3D mask dataset; only data time series from within the
mask will be analyzed; results for voxels outside the mask will be
set to zero.
argument: ``-mask %s``

automask: (a boolean, nipype default value: False)
build a mask automatically from input data (will be slow for long
time series datasets)
argument: ``-automask``

STATmask: (a pathlike object or string representing an existing file)
filename of 3D mask dataset to be used for the purpose of reporting
truncation-to float issues AND for computing the FDR curves. The
actual results ARE not masked with this option (only with 'mask' or
'automask' options).
argument: ``-STATmask %s``

addbase: (a list of items which are a pathlike object or string
representing an existing file)
file(s) to add baseline model columns to the matrix with this
option. Each column in the specified file(s) will be appended to the
matrix. File(s) must have at least as many rows as the matrix does.
argument: ``-addbase %s``

slibase: (a list of items which are a pathlike object or string
representing an existing file)
similar to 'addbase' in concept, BUT each specified file must have
an integer multiple of the number of slices in the input dataset(s);
then, separate regression matrices are generated for each slice,
with the first column of the file appended to the matrix for the
first slice of the dataset, the second column of the file appended
to the matrix for the first slice of the dataset, and so on.
Intended to help model physiological noise in fMRI, or other effects
you want to regress out that might change significantly in the
inter-slice time intervals. This will slow the program down, and
make it use a lot more memory (to hold all the matrix stuff).
argument: ``-slibase %s``

slibase_sm: (a list of items which are a pathlike object or string
representing an existing file)
similar to 'slibase', BUT each file much be in slice major order
(i.e. all slice0 columns come first, then all slice1 columns, etc).
argument: ``-slibase_sm %s``

usetemp: (a boolean)
write intermediate stuff to disk, to economize on RAM. Using this
option might be necessary to run with 'slibase' and with 'Grid'
values above the default, since the program has to store a large
number of matrices for such a problem: two for every slice and for
every (a,b) pair in the ARMA parameter grid. Temporary files are
written to the directory given in environment variable TMPDIR, or in
/tmp, or in ./ (preference is in that order)
argument: ``-usetemp``

nodmbase: (a boolean)
by default, baseline columns added to the matrix via 'addbase' or
'slibase' or 'dsort' will each have their mean removed (as is done
in Deconvolve); this option turns this centering off
argument: ``-nodmbase``
requires: addbase, dsort

dsort: (a pathlike object or string representing an existing file)
4D dataset to be used as voxelwise baseline regressor
argument: ```-dsort %s```

dsort_nods: (a boolean)
if 'dsort' option is used, this command will output additional
results files excluding the 'dsort' file
argument: ```-dsort_nods```
requires: dsort

fout: (a boolean)
output F-statistic for each stimulus
argument: ```-fout```

rout: (a boolean)
output the R^2 statistic for each stimulus
argument: ```-rout```

tout: (a boolean)
output the T-statistic for each stimulus; if you use 'out_file' and
do not give any of 'fout', 'tout', or 'rout', then the program
assumes 'fout' is activated.
argument: ```-tout```

nofdr: (a boolean)
do NOT add FDR curve data to bucket datasets; FDR curves can take a
long time if 'tout' is used
argument: ```-noFDR```

nobout: (a boolean)
do NOT add baseline (null hypothesis) regressor betas to the
'rbeta_file' and/or 'obeta_file' output datasets.
argument: ```-nobout```

gltsym: (a list of items which are a tuple of the form: (a pathlike
object or string representing an existing file, a unicode string)
or a tuple of the form: (a unicode string, a unicode string))
read a symbolic GLT from input file and associate it with a label.
As in Deconvolve, you can also use the 'SYM:' method to provide the
definition of the GLT directly as a string (e.g., with 'SYM: +Label1
-Label2'). Unlike Deconvolve, you MUST specify 'SYM: ' if providing
the GLT directly as a string instead of from a file
argument: ```-gltsym "%s" %s...```

out_file: (a pathlike object or string representing a file)
output dataset for beta + statistics from the REML estimation; also
contains the results of any GLT analysis requested in the Deconvolve
setup, similar to the 'bucket' output from Deconvolve. This dataset
does NOT get the betas (or statistics) of those regressors marked as
'baseline' in the matrix file.
argument: ```-Rbuck %s```

var_file: (a pathlike object or string representing a file)
output dataset for REML variance parameters
argument: ```-Rvar %s```

rbeta_file: (a pathlike object or string representing a file)
output dataset for beta weights from the REML estimation, similar to
the 'bucket' output from Deconvolve. This dataset will contain all
the beta weights, for baseline and stimulus regressors alike, unless
the '-nobout' option is given -- in that case, this dataset will
only get the betas for the stimulus regressors.
argument: ```-Rbeta %s```

glt_file: (a pathlike object or string representing a file)
output dataset for beta + statistics from the REML estimation, but
ONLY for the GLTs added on the REMLfit command line itself via
'gltsym'; GLTs from Deconvolve's command line will NOT be included.
argument: ```-Rglt %s```
fitts_file: (a pathlike object or string representing a file)
output dataset for REML fitted model
argument: ‘-Rfitts %s’
errts_file: (a pathlike object or string representing a file)
output dataset for REML residuals = data - fitted model
argument: ‘-Rerrts %s’
wherr_file: (a pathlike object or string representing a file)
dataset for REML residual, whitened using the estimated ARMA(1,1)
correlation matrix of the noise
argument: ‘-Rwherr %s’
quiet: (a boolean)
turn off most progress messages
argument: ‘-quiet’
verb: (a boolean)
turns on more progress messages, including memory usage progress
reports at various stages
argument: ‘-verb’
ovar: (a pathlike object or string representing a file)
dataset for OLSQ st.dev. parameter (kind of boring)
argument: ‘-Ovar %s’
obeta: (a pathlike object or string representing a file)
dataset for beta weights from the OLSQ estimation
argument: ‘-Obeta %s’
obuck: (a pathlike object or string representing a file)
dataset for beta + statistics from the OLSQ estimation
argument: ‘-Obuck %s’
oglt: (a pathlike object or string representing a file)
dataset for beta + statistics from 'gltsym' options
argument: ‘-Oglt %s’
ofitts: (a pathlike object or string representing a file)
dataset for OLSQ fitted model
argument: ‘-Ofitts %s’
oerrts: (a pathlike object or string representing a file)
dataset for OLSQ residuals (data - fitted model)
argument: ‘-Oerrts %s’
num_threads: (an integer (int or long), nipype default value: 1)
set number of threads
outputtype: (‘NIFTI’ or ‘AFNI’ or ‘NIFTI_GZ’)
AFNI output filetype
args: (a unicode string)
Additional parameters to the command
argument: ‘%s’
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {}) Environment variables

Outputs:

out_file: (a pathlike object or string representing a file)
dataset for beta + statistics from the REML estimation (if generated)
var_file: (a pathlike object or string representing a file)
dataset for REML variance parameters (if generated)
rbeta_file: (a pathlike object or string representing a file)
output dataset for beta weights from the REML estimation (if
generated)
glt_file: (a pathlike object or string representing a file)
output dataset for beta + statistics from the REML estimation, but
ONLY for the GLTs added on the REMLfit command line itself via
'gltsym' (if generated)
fitts_file: (a pathlike object or string representing a file)
output dataset for REML fitted model (if generated)
errts_file: (a pathlike object or string representing a file)
output dataset for REML residuals = data - fitted model (if
generated
wherr_file: (a pathlike object or string representing a file)
dataset for REML residual, whitened using the estimated ARMA(1,1)
correlation matrix of the noise (if generated)
ovar: (a pathlike object or string representing a file)
dataset for OLSQ st.dev. parameter (if generated)
obeta: (a pathlike object or string representing a file)
dataset for beta weights from the OLSQ estimation (if generated)
obuck: (a pathlike object or string representing a file)
dataset for beta + statistics from the OLSQ estimation (if
generated
oglt: (a pathlike object or string representing a file)
dataset for beta + statistics from 'gltsym' options (if generated
ofitts: (a pathlike object or string representing a file)
dataset for OLSQ fitted model (if generated)
oerrts: (a pathlike object or string representing a file)
dataset for OLSQ residuals = data - fitted model (if generated

References:
None None

51.2.3 Synthesize

Link to code
Wraps the executable command 3dSynthesize.
Reads a `-cbucket` dataset and a `.xmat.1D` matrix from 3dDeconvolve, and synthesizes a fit dataset using
user-selected sub-bricks and matrix columns.
For complete details, see the 3dSynthesize Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> synthesize = afni.Synthesize()
>>> synthesize.inputs.cbucket = 'functional.nii'
>>> synthesize.inputs.matrix = 'output.1D'
>>> synthesize.inputs.select = ['baseline']
>>> synthesize.cmdline
'3dSynthesize -cbucket functional.nii -matrix output.1D -select baseline'
>>> syn = synthesize.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
cbucket: (a pathlike object or string representing a file)
Read the dataset output from 3dDeconvolve via the '-cbucket' option.
argument: `'-cbucket %s'`
matrix: (a pathlike object or string representing a file)
Read the matrix output from 3dDeconvolve via the '-x1D' option.
argument: `'-matrix %s'`

select: (a list of items which are a unicode string)
A list of selected columns from the matrix (and the corresponding
coefficient sub-bricks from the cbucket). Valid types include
'baseline', 'polort', 'allfunc', 'allstim', 'all', Can also provide
'something' where something matches a stim_label from 3dDeconvolve,
and 'digits' where digits are the numbers of the select matrix
columns by numbers (starting at 0), or number ranges of the form
'3..7' and '3-7'.
argument: `'-select %s'`

[Optional]
out_file: (a pathlike object or string representing a file)
output dataset prefix name (default 'syn')
argument: `'-prefix %s'`
dry_run: (a boolean)
Don't compute the output, just check the inputs.
argument: `'-dry'`
TR: (a float)
TR to set in the output. The default value of TR is read from the
header of the matrix file.
argument: `'-TR %f'`
cenfill: ('zero' or 'nbhr' or 'none')
Determines how censored time points from the 3dDeconvolve run will
be filled. Valid types are 'zero', 'nbhr' and 'none'.
argument: `'-cenfill %s'`
num_threads: (an integer (int or long), nipype default value: 1)
set number of threads
outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
AFNI output filetype
args: (a unicode string)
Additional parameters to the command
argument: `'\%s'`
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
output file

References:
None

51.3 interfaces.afni.preprocess

51.3.1 AlignEpiAnatPy

Link to code
Wraps the executable command align_epi_anat.py.
Align EPI to anatomical datasets or vice versa. This Python script computes the alignment between two datasets,
typically an EPI and an anatomical structural dataset, and applies the resulting transformation to one or the other
to bring them into alignment.

This script computes the transforms needed to align EPI and anatomical datasets using a cost function designed for this purpose. The script combines multiple transformations, thereby minimizing the amount of interpolation applied to the data.

**Basic Usage:** `align_epi_anat.py -anat anat+orig -epi epi+orig -epi_base 5`

The user must provide EPI and anatomical datasets and specify the EPI sub-brick to use as a base in the alignment.

Internally, the script always aligns the anatomical to the EPI dataset, and the resulting transformation is saved to a 1D file. As a user option, the inverse of this transformation may be applied to the EPI dataset in order to align it to the anatomical data instead.

This program generates several kinds of output in the form of datasets and transformation matrices which can be applied to other datasets if needed. Time-series volume registration, oblique data transformations and Talairach (standard template) transformations will be combined as needed and requested (with options to turn on and off each of the steps) in order to create the aligned datasets.

For complete details, see the `align_epi_anat.py` Documentation.

### Examples

```python
>>> from nipype.interfaces import afni
>>> al_ea = afni.AlignEpiAnatPy()
>>> al_ea.inputs.anat = "structural.nii"
>>> al_ea.inputs.in_file = "functional.nii"
>>> al_ea.inputs.epi_base = 0
>>> al_ea.inputs.epi_strip = '3dAutomask'
>>> al_ea.inputs.volreg = 'off'
>>> al_ea.inputs.tshift = 'off'
>>> al_ea.inputs.save_skullstrip = True
>>> al_ea.cmdline
# doctest: +ELLIPSIS
'python2 ...align_epi_anat.py -anat structural.nii -epi_base 0 -epi_strip
˓→3dAutomask -epi functional.nii -save_skullstrip -suffix _al -tshift off -volreg
˓→off'
>>> res = alineate.run()  # doctest: +SKIP
```

### Inputs:

- **in_file**: (a pathlike object or string representing an existing file)
  EPI dataset to align
  argument: `'-epi %s'`
- **anat**: (a pathlike object or string representing an existing file)
  name of structural dataset
  argument: `'-anat %s'`
- **epi_base**: (a long integer >= 0 or 'mean' or 'median' or 'max')
  the epi base used in alignment should be one of
  (0/mean/median/max/subbrick#)
  argument: `'-epi_base %s'`
- **anat2epi**: (a boolean)
  align anatomical to EPI dataset (default)
  argument: `'-anat2epi'`
- **epi2anat**: (a boolean)
  align EPI to anatomical dataset
  argument: `'-epi2anat'`
- **save_skullstrip**: (a boolean)
  save skull-stripped (not aligned)

(continues on next page)
argument: ``-save_skullstrip``

suffix: (a unicode string, nipype default value: _al)
append suffix to the original anat/epi dataset to use in the
resulting dataset names (default is "_al")
argument: ``-suffix %s``

epi_strip: ('3dSkullStrip' or '3dAutomask' or 'None')
method to mask brain in EPI datashould be one
of [3dSkullStrip]/3dAutomask/None)
argument: ``-epi_strip %s``

volreg: {'on' or 'off', nipype default value: on}
do volume registration on EPI dataset before alignmentshould be 'on'
or 'off', defaults to 'on'
argument: ``-volreg %s``

tshift: {'on' or 'off', nipype default value: on}
do time shifting of EPI dataset before alignmentshould be 'on'
or 'off', defaults to 'on'
argument: ``-tshift %s``

outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
AFNI output filetype

py27_path: (a pathlike object or string representing an existing file
or 'python2', nipype default value: python2)

args: (a unicode string)
Additional parameters to the command
argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {}))
Environment variables

Outputs:

anat_al_orig: (a pathlike object or string representing a file)
A version of the anatomy that is aligned to the EPI
epi_al_orig: (a pathlike object or string representing a file)
A version of the EPI dataset aligned to the anatomy
epi_tlrc_al: (a pathlike object or string representing a file)
A version of the EPI dataset aligned to a standard template
anat_al_mat: (a pathlike object or string representing a file)
matrix to align anatomy to the EPI
epi_al_mat: (a pathlike object or string representing a file)
matrix to align EPI to anatomy
epi_vr_al_mat: (a pathlike object or string representing a file)
matrix to volume register EPI
epi_reg_al_mat: (a pathlike object or string representing a file)
matrix to volume register and align epi to anatomy
epi_al_tlrc_mat: (a pathlike object or string representing a file)
matrix to volume register and align epi to anatomy and put into
standard space
epi_vr_motion: (a pathlike object or string representing a file)
motion parameters from EPI time-seriesregistration (tsh included in
name if slicetiming correction is also included).
skullstrip: (a pathlike object or string representing a file)
skull-stripped (not aligned) volume

References:

None None

51.3. interfaces.afni.preprocess
51.3.2 Allineate

Link to code
Wraps the executable command 3dAllineate.
Program to align one dataset (the ‘source’) to a base dataset
For complete details, see the 3dAllineate Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> allineate = afni.Allineate()
>>> allineate.inputs.in_file = 'functional.nii'
>>> allineate.inputs.in_matrix = 'cmatrix.mat'
>>> allineate.cmdline
'3dAllineate -source functional.nii -prefix functional_allineate.nii -1Dmatrix_apply cmatrix.mat'
>>> res = allineate.run()  # doctest: +SKIP
```

```python
>>> allineate = afni.Allineate()
>>> allineate.inputs.in_file = 'functional.nii'
>>> allineate.inputs.reference = 'structural.nii'
>>> allineate.cmdline
'3dAllineate -source functional.nii -base structural.nii -allcostx |& tee out.allcostX.txt'
>>> res = allineate.run()  # doctest: +SKIP
```

```python
>>> allineate = afni.Allineate()
>>> allineate.inputs.in_file = 'functional.nii'
>>> allineate.inputs.nwarp_fixmot = ['X', 'Y']
>>> allineate.cmdline
>>> res = allineate.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  input file to 3dAllineate
  argument: `--source %s`

[Optional]
reference: (a pathlike object or string representing an existing file)
  file to be used as reference, the first volume will be used if not given the reference will be the first volume of in_file.
  argument: `--base %s`
out_file: (a pathlike object or string representing a file)
  output file from 3dAllineate
  argument: `--prefix %s`
mutually_exclusive: allcostx
out_param_file: (a pathlike object or string representing a file)
  Save the warp parameters in ASCII (.1D) format.
  argument: `--1Dparam_save %s`

(continues on next page)
mutually_exclusive: in_param_file, allcostx

in_param_file: (a pathlike object or string representing an existing file)
Read warp parameters from file and apply them to the source dataset, and produce a new dataset
argument: ```-1Dparam_apply %s```

mutually_exclusive: out_param_file

out_matrix: (a pathlike object or string representing a file)
Save the transformation matrix for each volume.
argument: ```-1Dmatrix_save %s```

mutually_exclusive: in_matrix, allcostx

in_matrix: (a pathlike object or string representing a file)
matrix to align input file
argument: ```-1Dmatrix_apply %s``, position: -3

mutually_exclusive: out_matrix

overwrite: (a boolean)
overwrite output file if it already exists
argument: ```-overwrite```

allcostx: (a pathlike object or string representing a file)
Compute and print ALL available cost functionals for the un-warped inputs AND THEN QUIT. If you use this option none of the other expected outputs will be produced
argument: ```-allcostx |& tee %s```, position: -1

mutually_exclusive: out_file, out_matrix, out_param_file, out_weight_file

cost: ('leastsq' or 'ls' or 'mutualinfo' or 'mi' or 'corratio_mul' or 'crM' or 'norm_mutualinfo' or 'nmi' or 'hellinger' or 'hel' or 'corratio_add' or 'crA' or 'corratio_uns' or 'crU')
Defines the 'cost' function that defines the matching between the source and the base
argument: ```-cost %s```

interpolation: ('nearestneighbour' or 'linear' or 'cubic' or 'quintic')
Defines interpolation method to use during matching
argument: ```-interp %s```

final_interpolation: ('nearestneighbour' or 'linear' or 'cubic' or 'quintic' or 'wsinc5')
Defines interpolation method used to create the output dataset
argument: ```-final %s```

nmatch: (an integer (int or long))
Use at most n scattered points to match the datasets.
argument: ```-nmatch %d```

no_pad: (a boolean)
Do not use zero-padding on the base image.
argument: ```-nopad```

zclip: (a boolean)
Replace negative values in the input datasets (source & base) with zero.
argument: ```-zclip```

convergence: (a float)
Convergence test in millimeters (default 0.05mm).
argument: ```-conv %f```

usetemp: (a boolean)
temporary file use
argument: ```-usetemp```

check: (a list of items which are 'leastsq' or 'ls' or 'mutualinfo'
or 'mi' or 'corratio_mul' or 'crM' or 'norm_mutualinfo' or 'nmi' (continues on next page)
or 'hellinger' or 'hel' or 'corratio_add' or 'crA' or 'corratio_uns' or 'crU')

After cost functional optimization is done, start at the final parameters and RE-optimize using this new cost functions. If the results are too different, a warning message will be printed. However, the final parameters from the original optimization will be used to create the output dataset.

argument: `-check %s`

one_pass: (a boolean)
Use only the refining pass -- do not try a coarse resolution pass first. Useful if you know that only small amounts of image alignment are needed.

argument: `-onepass`

two_pass: (a boolean)
Use a two pass alignment strategy for all volumes, searching for a large rotation+shift and then refining the alignment.

argument: `-twopass`

two_blur: (a float)
Set the blurring radius for the first pass in mm.

argument: `-twoblur %f`

two_first: (a boolean)
Use -twopass on the first image to be registered, and then on all subsequent images from the source dataset, use results from the first image's coarse pass to start the fine pass.

argument: `-twofirst`

two_best: (an integer (int or long))
In the coarse pass, use the best 'bb' set of initial points to search for the starting point for the fine pass. If bb==0, then no search is made for the best starting point, and the identity transformation is used as the starting point. [Default=5; min=0 max=11]

argument: `-twobest %d`

fine_blur: (a float)
Set the blurring radius to use in the fine resolution pass to 'x' mm. A small amount (1-2 mm?) of blurring at the fine step may help with convergence, if there is some problem, especially if the base volume is very noisy. [Default == 0 mm = no blurring at the final alignment pass]

argument: `-fineblur %f`

center_of_mass: (a unicode string)
Use the center-of-mass calculation to bracket the shifts.

argument: `-cmass%s`

autoweight: (a unicode string)
Compute a weight function using the 3dAutomask algorithm plus some blurring of the base image.

argument: `-autoweight%s`

automask: (an integer (int or long))
Compute a mask function, set a value for dilation or 0.

argument: `-automask+%d`

autobox: (a boolean)
Expand the -automask function to enclose a rectangular box that holds the irregular mask.

argument: `-autobox`

nomask: (a boolean)
Don't compute the autoweight/mask; if -weight is not also used, then every voxel will be counted equally.

argument: `-nomask`

weight_file: (a pathlike object or string representing an existing file)

(continues on next page)
weight: (a pathlike object or string representing an existing file or a float)
   Set the weighting for each voxel in the base dataset; larger weights mean that voxel count more in the cost function. If an image file is given, the volume must be defined on the same grid as the base dataset.
   argument: "-weight %s"
out_weight_file: (a pathlike object or string representing a file)
   Write the weight volume to disk as a dataset
   argument: "-wtprefix %s"
mutually_exclusive: allcostx
source_mask: (a pathlike object or string representing an existing file)
   mask the input dataset
   argument: "-source_mask %s"
source_automask: (an integer (int or long))
   Automatically mask the source dataset with dilation or 0.
   argument: "-source_automask+%d"
warp_type: ('shift_only' or 'shift_rotate' or 'shift_rotate_scale' or 'affine_general')
   Set the warp type.
   argument: "-warp %s"
warpfreeze: (a boolean)
   Freeze the non-rigid body parameters after first volume.
   argument: "-warpfreeze"
replacebase: (a boolean)
   If the source has more than one volume, then after the first volume is aligned to the base.
   argument: "-replacebase"
replacemeth: ('leastsq' or 'ls' or 'mutualinfo' or 'mi' or 'corratio_mul' or 'crM' or 'norm_mutualinfo' or 'nmi' or 'hellinger' or 'hel' or 'corratio_add' or 'crA' or 'corratio_add' or 'crU')
   After first volume is aligned, switch method for later volumes. For use with "-replacebase".
   argument: "-replacemeth %s"
epi: (a boolean)
   Treat the source dataset as being composed of warped EPI slices, and the base as comprising anatomically 'true' images. Only phase-encoding direction image shearing and scaling will be allowed with this option.
   argument: "-EPI"
maxrot: (a float)
   Maximum allowed rotation in degrees.
   argument: "-maxrot %f"
maxshf: (a float)
   Maximum allowed shift in mm.
   argument: "-maxshf %f"
maxscl: (a float)
   Maximum allowed scaling factor.
   argument: "-maxscl %f"
Maximum allowed shearing factor.
argument: `''-maxshr %f''`

master: (a pathlike object or string representing an existing file)
Write the output dataset on the same grid as this file.
argument: `''-master %s''`

newgrid: (a float)
Write the output dataset using isotropic grid spacing in mm.
argument: `''-newgrid %f''`

nwarp: ('bilinear' or 'cubic' or 'quintic' or 'heptic' or 'nonic' or
'poly3' or 'poly5' or 'poly7' or 'poly9')
Experimental nonlinear warping: bilinear or legendre poly.
argument: `''-nwarp %s''`

nwarp_fixmot: (a list of items which are 'X' or 'Y' or 'Z' or 'I' or
'J' or 'K')
To fix motion along directions.
argument: `''-nwarp_fixmot%s...''`

nwarp_fixdep: (a list of items which are 'X' or 'Y' or 'Z' or 'I' or
'J' or 'K')
To fix non-linear warp dependency along directions.
argument: `''-nwarp_fixdep%s...''`

verbose: (a boolean)
Print out verbose progress reports.
argument: `''-verb''`

quiet: (a boolean)
Don't print out verbose progress reports.
argument: `''-quiet''`

num_threads: (an integer (int or long), nipype default value: 1)
set number of threads

outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
AFNI output filetype

args: (a unicode string)
Additional parameters to the command
argument: `''%s''`

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
output image file name

out_matrix: (a pathlike object or string representing an existing file)
matrix to align input file

out_param_file: (a pathlike object or string representing an existing file)
warp parameters

out_weight_file: (a pathlike object or string representing an existing file)
weight volume

allcostx: (a pathlike object or string representing a file)
Compute and print ALL available cost functionals for the un-warped inputs

References:

None None
51.3.3 AutoTLRC

Link to code
Wraps the executable command @auto_tlrc. A minimal wrapper for the AutoTLRC script. The only option currently supported is no_ss. For complete details, see the 3dQwarp Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> autoTLRC = afni.AutoTLRC()
>>> autoTLRC.inputs.in_file = 'structural.nii'
>>> autoTLRC.inputs.no_ss = True
>>> autoTLRC.inputs.base = "TT_N27+tlrc"
>>> autoTLRC.cmdline
'@auto_tlrc -base TT_N27+tlrc -input structural.nii -no_ss'
>>> res = autoTLRC.run() # doctest: +SKIP
```

Inputs:

[Mandatory]

in_file: (a pathlike object or string representing an existing file)
Original anatomical volume (+orig). The skull is removed by this script unless instructed otherwise (-no_ss). argument: ```-input %s```

base: (a unicode string)
Reference anatomical volume. Usually this volume is in some standard space like TLRC or MNI space and with afni dataset view of (+tlrc). Preferably, this reference volume should have had the skull removed but that is not mandatory. AFNI's distribution contains several templates. For a longer list, use ```whereami -show_templates``` TT_N27+tlrc --> Single subject, skull stripped volume. This volume is also known as N27_SurfVol_NoSkull+tlrc elsewhere in AFNI and SUMA land. (www.loni.ucla.edu, www.bic.mni.mcgill.ca) This template has a full set of FreeSurfer (surfer.nmr.mgh.harvard.edu) surface models that can be used in SUMA. For details, see Talairach-related link: https://afni.nimh.nih.gov/afni/sumaTT_icbm452+tlrc --> Average volume of 452 normal brains. Skull Stripped. (www.loni.ucla.edu) TT_avg152T1+tlrc --> Average volume of 152 normal brains. Skull Stripped. (www.bic.mni.mcgill.ca) TT_EPI+tlrc --> EPI template from spm2, masked as TT_avg152T1 TT_avg152 and TT_EPI volume sources are from SPM's distribution. (www.fil.ion.ucl.ac.uk/spm/) If you do not specify a path for the template, the script will attempt to locate the template AFNI's binaries directory. NOTE: These datasets have been slightly modified from their original size to match the standard TLRC dimensions (Jean Talairach and Pierre Tournoux Co-Planar Stereotaxic Atlas of the Human Brain Thieme Medical Publishers, New York, 1988). That was done for internal consistency in AFNI. You may use the original form of these volumes if you choose but your TLRC coordinates will not be consistent with AFNI's TLRC database (San Antonio Talairach Daemon database), for example. argument: ```-input %s```

[Optional]
outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
AFNI output filetype
no_ss: (a boolean)
Do not strip skull of input data set (because skull has already been
removed or because template still has the skull) NOTE: The -no_ss
option is not all that optional. Here is a table of when you should
and should not use -no_ss Template Template WITH skull WITHOUT skull
Dset. WITH skull -no_ss xxx WITHOUT skull No Cigar -no_ss Template
means: Your template of choice Dset. means: Your anatomical dataset
-no_ss means: Skull stripping should not be attempted on Dset xxx
means: Don't put anything, the script will strip Dset No Cigar
means: Don't try that combination, it makes no sense.
argument: `-no_ss`
args: (a unicode string)
  Additional parameters to the command
  argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
Environment variables

Outputs:
out_file: (a pathlike object or string representing an existing file)
  output file

References:
None

51.3.4 AutoTcorrelate

Link to code
Wraps the executable command 3dAutoTcorrelate.
Computes the correlation coefficient between the time series of each pair of voxels in the input dataset, and
stores the output into a new anatomical bucket dataset [scaled to shorts to save memory space].
For complete details, see the 3dAutoTcorrelate Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> corr = afni.AutoTcorrelate()
>>> corr.inputs.in_file = 'functional.nii'
>>> corr.inputs.polort = -1
>>> corr.inputs.eta2 = True
>>> corr.inputs.mask = 'mask.nii'
>>> corr.inputs.mask_only_targets = True
>>> corr.cmdline  # doctest: +ELLIPSIS
'3dAutoTcorrelate -eta2 -mask mask.nii -mask_only_targets -prefix functional_ →similarity_matrix.1D -polort -1 functional.nii'
>>> res = corr.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
timeseries x space (volume or surface) file
  argument: `%s`, position: -1
[Optional]
polort: (an integer (int or long))
   Remove polynomial trend of order m or -1 for no detrending
   argument: `--polort %d`
eta2: (a boolean)
   eta^2 similarity
   argument: `--eta2`
mask: (a pathlike object or string representing an existing file)
   mask of voxels
   argument: `--mask %s`
mask_only_targets: (a boolean)
   use mask only on targets voxels
   argument: `--mask_only_targets`
   mutually_exclusive: mask_source
mask_source: (a pathlike object or string representing an existing file)
   mask for source voxels
   argument: `--mask_source %s`
   mutually_exclusive: mask_only_targets
out_file: (a pathlike object or string representing a file)
   output image file name
   argument: `--prefix %s`
num_threads: (an integer (int or long), nipype default value: 1)
   set number of threads
outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
   AFNI output filetype
args: (a unicode string)
   Additional parameters to the command
   argument: `%%%s`
environ: (a dictionary with keys which are a bytes or None or a value
   of class 'str' and with values which are a bytes or None or a
   value of class 'str', nipype default value: {})
   Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
   output file

References:

None None

51.3.5 Automask

Link to code
Wraps the executable command 3dAutomask.
Create a brain-only mask of the image using AFNI 3dAutomask command
For complete details, see the 3dAutomask Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> automask = afni.Automask()
```
>>> automask.inputs.in_file = 'functional.nii'
>>> automask.inputs.dilate = 1
>>> automask.inputs.outputtype = 'NIFTI'
>>> automask.cmdline  # doctest: +ELLIPSIS
'3dAutomask -apply_prefix functional_masked.nii -dilate 1 -prefix functional_mask.nii'
>>> res = automask.run()  # doctest: +SKIP

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
    input file to 3dAutomask
    argument: ``'%s'``, position: -1

[Optional]
out_file: (a pathlike object or string representing a file)
    output image file name
    argument: ``'-prefix %s'``
brain_file: (a pathlike object or string representing a file)
    output file from 3dAutomask
    argument: ``'-apply_prefix %s'``
clfrac: (a float)
    sets the clip level fraction (must be 0.1-0.9). A small value will
    tend to make the mask larger [default = 0.5].
    argument: ``'-clfrac %s'``
dilate: (an integer (int or long))
    dilate the mask outwards
    argument: ``'-dilate %s'``
erode: (an integer (int or long))
    erode the mask inwards
    argument: ``'-erode %s'``
num_threads: (an integer (int or long), nipype default value: 1)
    set number of threads
outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
    AFNI output filetype
args: (a unicode string)
    Additional parameters to the command
    argument: ``'%s'``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
    mask file
brain_file: (a pathlike object or string representing an existing file)
    brain file (skull stripped)

References:

None None
51.3.6 Bandpass

Link to code  
Wraps the executable command 3dBandpass.  
Program to lowpass and/or highpass each voxel time series in a dataset, offering more/different options than Fourier.  
For complete details, see the 3dBandpass Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> from nipype.testing import example_data
>>> bandpass = afni.Bandpass()
>>> bandpass.inputs.in_file = 'functional.nii'
>>> bandpass.inputs.highpass = 0.005
>>> bandpass.inputs.lowpass = 0.1
>>> bandpass.cmdline
'3dBandpass -prefix functional_bp 0.005000 0.100000 functional.nii'
>>> res = bandpass.run()  # doctest: +SKIP
```

Inputs:

<table>
<thead>
<tr>
<th>Mandatory</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in_file: (a pathlike object or string representing an existing file)</td>
<td>input file to 3dBandpass</td>
</tr>
<tr>
<td></td>
<td>argument: &quot;%s&quot;, position: -1</td>
</tr>
<tr>
<td>lowpass: (a float)</td>
<td>lowpass</td>
</tr>
<tr>
<td></td>
<td>argument: &quot;%f&quot;, position: -2</td>
</tr>
<tr>
<td>highpass: (a float)</td>
<td>highpass</td>
</tr>
<tr>
<td></td>
<td>argument: &quot;%f&quot;, position: -3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optional</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>out_file: (a pathlike object or string representing a file)</td>
<td>output file from 3dBandpass</td>
</tr>
<tr>
<td></td>
<td>argument: &quot;-prefix %s&quot;, position: 1</td>
</tr>
<tr>
<td>mask: (a pathlike object or string representing an existing file)</td>
<td>mask file</td>
</tr>
<tr>
<td></td>
<td>argument: &quot;-mask %s&quot;, position: 2</td>
</tr>
<tr>
<td>despike: (a boolean)</td>
<td>Despike each time series before other processing. Hopefully, you don't actually need to do this, which is why it is optional.</td>
</tr>
<tr>
<td></td>
<td>argument: &quot;-despike&quot;</td>
</tr>
<tr>
<td>orthogonalize_file: (a list of items which are a pathlike object or string representing an existing file)</td>
<td>Also orthogonalize input to columns in f.1D. Multiple '-ort' options are allowed.</td>
</tr>
<tr>
<td></td>
<td>argument: &quot;-ort %s&quot;</td>
</tr>
<tr>
<td>orthogonalize_dset: (a pathlike object or string representing an existing file)</td>
<td>Orthogonalize each voxel to the corresponding voxel time series in dataset 'fset', which must have the same spatial and temporal grid structure as the main input dataset. At present, only one '-dsort' option is allowed.</td>
</tr>
<tr>
<td></td>
<td>argument: &quot;-dsort %s&quot;</td>
</tr>
<tr>
<td>no_detrend: (a boolean)</td>
<td>Skip the quadratic detrending of the input that occurs before the</td>
</tr>
</tbody>
</table>
FFT-based bandpassing. You would only want to do this if the dataset had been detrended already in some other program. 

argument: `\'-nodetrend` 

tr: (a float) 
Set time step (TR) in sec [default=from dataset header]. 
argument: `\'-dt %f` 

nfft: (an integer (int or long)) 
Set the FFT length [must be a legal value]. 
argument: `\'-nfft %d` 

normalize: (a boolean) 
Make all output time series have L2 norm = 1 (i.e., sum of squares = 1). 
argument: `\'-norm` 

automask: (a boolean) 
Create a mask from the input dataset. 
argument: `\'-automask` 

blur: (a float) 
Blur (inside the mask only) with a filter width (FWHM) of 'fff' millimeters. 
argument: `\'-blur %f` 

localPV: (a float) 
Replace each vector by the local Principal Vector (AKA first singular vector) from a neighborhood of radius 'rrr' millimeters. 
Note that the PV time series is L2 normalized. This option is mostly for Bob Cox to have fun with. 
argument: `\'-localPV %f` 

notrans: (a boolean) 
Don't check for initial positive transients in the data. The test is a little slow, so skipping it is OK, if you KNOW the data time series are transient-free. 
argument: `\'-notrans` 

num_threads: (an integer (int or long), nipype default value: 1) 
set number of threads 

outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ') 
AFNI output filetype 

args: (a unicode string) 
Additional parameters to the command 
argument: `\'%s` 

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) 
Environment variables 

Outputs:

out_file: (a pathlike object or string representing an existing file) 
output file 

References:

None

51.3.7 BlurInMask

Link to code
Wraps the executable command 3dBlurInMask. 
Blurs a dataset spatially inside a mask. That’s all. Experimental.
For complete details, see the 3dBlurInMask Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> bim = afni.BlurInMask()
>>> bim.inputs.in_file = 'functional.nii'
>>> bim.inputs.mask = 'mask.nii'
>>> bim.inputs.fwhm = 5.0
>>> bim.cmdline  # doctest: +ELLIPSIS
'3dBlurInMask -input functional.nii -FWHM 5.000000 -mask mask.nii -prefix_˓→functional_blur'
>>> res = bim.run()  # doctest: +SKIP
```

## Inputs:

[Mandatory]
- `in_file`: (a pathlike object or string representing an existing file)
  - input file to 3dSkullStrip
    - argument: `'-input %s'`, position: 1
  - fwhm: (a float)
    - fwhm kernel size
    - argument: `'-FWHM %f'`

[Optional]
- `out_file`: (a pathlike object or string representing a file)
  - output to the file
    - argument: `'-prefix %s'`, position: -1
- `mask`: (a pathlike object or string representing a file)
  - Mask dataset, if desired. Blurring will occur only within the mask.
    - Voxels NOT in the mask will be set to zero in the output.
    - argument: `'-mask %s'`
- `multimask`: (a pathlike object or string representing a file)
  - Multi-mask dataset -- each distinct nonzero value in dataset will be treated as a separate mask for blurring purposes.
    - argument: `'-Mmask %s'`
- `automask`: (a boolean)
  - Create an automask from the input dataset.
    - argument: `'-automask'`
- `preserve`: (a boolean)
  - Normally, voxels not in the mask will be set to zero in the output.
    - If you want the original values in the dataset to be preserved in the output, use this option.
    - argument: `'-preserve'`
- `float_out`: (a boolean)
  - Save dataset as floats, no matter what the input data type is.
    - argument: `'-float'`
- `options`: (a unicode string)
  - options
    - argument: `'%s'`, position: 2
- `num_threads`: (an integer (int or long), nipype default value: 1)
  - set number of threads
- `outputtype`: (’NIFTI’ or ’AFNI’ or ’NIFTI_GZ’)
  - AFNI output filetype
- `args`: (a unicode string)
  - Additional parameters to the command
    - argument: `'%s'`
- `environ`: (a dictionary with keys which are a bytes or None or a value
(continues on next page)
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
output file

References:

None None

51.3.8 BlurToFWHM

Link to code
Wraps the executable command 3dBlurToFWHM.
Blurs a ‘master’ dataset until it reaches a specified FWHM smoothness (approximately).
For complete details, see the 3dBlurToFWHM Documentation

Examples

```python
>>> from nipype.interfaces import afni
>>> blur = afni.preprocess.BlurToFWHM()
>>> blur.inputs.in_file = 'epi.nii'
>>> blur.inputs.fwhm = 2.5
>>> blur.cmdline  # doctest: +ELLIPSIS
'3dBlurToFWHM -FWHM 2.500000 -input epi.nii -prefix epi_afni'
>>> res = blur.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
The dataset that will be smoothed
argument: ``-input %s``

[Optional]
automask: (a boolean)
Create an automask from the input dataset.
argument: ``-automask``
fwhm: (a float)
Blur until the 3D FWHM reaches this value (in mm)
argument: ``-FWHM %f``
fwhmxy: (a float)
Blur until the 2D (x,y)-plane FWHM reaches this value (in mm)
argument: ``-FWHMxy %f``
blurmaster: (a pathlike object or string representing an existing file)
The dataset whose smoothness controls the process.
argument: ``-blurmaster %s``
mask: (a pathlike object or string representing an existing file)
Mask dataset, if desired. Voxels NOT in mask will be set to zero in output.
argument: ``-mask %s``
um_threads: (an integer (int or long), nipype default value: 1)
set number of threads
outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
    AFNI output filetype
out_file: (a pathlike object or string representing a file)
    output image file name
    argument: ```-prefix %s```
args: (a unicode string)
    Additional parameters to the command
    argument: ```%s```
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables

Outputs:
out_file: (a pathlike object or string representing an existing file)
    output file

References:
None None

51.3.9 ClipLevel

Link to code
Wraps the executable command 3dClipLevel.
Estimates the value at which to clip the anatomical dataset so that background regions are set to zero.
For complete details, see the 3dClipLevel Documentation.

Examples

```python
>>> from nipype.interfaces.afni import preprocess
>>> cliplevel = preprocess.ClipLevel()
>>> cliplevel.inputs.in_file = 'anatomical.nii'
>>> cliplevel.cmdline
'3dClipLevel anatomical.nii'
>>> res = cliplevel.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
    input file to 3dClipLevel
    argument: ```%s``', position: -1

[Optional]
mfrac: (a float)
    Use the number ff instead of 0.50 in the algorithm
    argument: ```-mfrac %s``', position: 2
doall: (a boolean)
    Apply the algorithm to each sub-brick separately.
    argument: ```-doall``', position: 3
    mutually_exclusive: g, r, a, d
grad: (a pathlike object or string representing a file)
    Also compute a 'gradual' clip level as a function of voxel position,
and output that to a dataset.
argument: `--grad %s'`, position: 3
mutually_exclusive: d, o, a, l, l
args: (a unicode string)
    Additional parameters to the command
argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})
Environment variables

Outputs:

clip_val: (a float)
    output

51.3.10 DegreeCentrality

Link to code
Wraps the executable command 3dDegreeCentrality.
Performs degree centrality on a dataset using a given maskfile via 3dDegreeCentrality
For complete details, see the 3dDegreeCentrality Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> degree = afni.DegreeCentrality()
>>> degree.inputs.in_file = 'functional.nii'
>>> degree.inputs.mask = 'mask.nii'
>>> degree.inputs.sparsity = 1  # keep the top one percent of connections
>>> degree.inputs.out_file = 'out.nii'
>>> degree.cmdline
'3dDegreeCentrality -mask mask.nii -prefix out.nii -sparsity 1.000000 functional.nii'
>>> res = degree.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
    input file to 3dDegreeCentrality
    argument: `%s'`, position: -1

[Optional]
sparsity: (a float)
    only take the top percent of connections
    argument: `--sparsity %f'
oned_file: (a unicode string)
    output filepath to text dump of correlation matrix
    argument: `--out1D %s`
mask: (a pathlike object or string representing an existing file)
    mask file to mask input data
    argument: `--mask %s`
thresh: (a float)
    threshold to exclude connections where corr <= thresh
    argument: `--thresh %f`

(continues on next page)
polort: (an integer (int or long))
  argument: ``-polort %d``
autoclip: (a boolean)
  Clip off low-intensity regions in the dataset
  argument: ``-autoclip``
automask: (a boolean)
  Mask the dataset to target brain-only voxels
  argument: ``-automask``
num_threads: (an integer (int or long), nipype default value: 1)
  set number of threads
outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
  AFNI output filetype
out_file: (a pathlike object or string representing a file)
  output image file name
  argument: ``-prefix %s``
args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
  Environment variables

Outputs:

oned_file: (a pathlike object or string representing a file)
  The text output of the similarity matrix computed after thresholding
  with one-dimensional and ijk voxel indices, correlations, image
  extents, and affine matrix.
out_file: (a pathlike object or string representing an existing file)
  output file

References:

None

51.3.11 Despike

Link to code

Wraps the executable command 3dDespike.
Removes 'spikes' from the 3D+time input dataset
For complete details, see the 3dDespike Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> despike = afni.Despike()
>>> despike.inputs.in_file = 'functional.nii'
>>> despike.cmdline
'3dDespike -prefix functional_despike functional.nii'
>>> res = despike.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
input file to 3dDespike
argument: `"%s"`, position: -1

[Optional]
out_file: (a pathlike object or string representing a file)
output image file name
argument: `-prefix %s`
num_threads: (an integer (int or long), nipype default value: 1)
set number of threads
outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
AFNI output filetype
args: (a unicode string)
Additional parameters to the command
argument: `"%s`
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
output file

References:
None

51.3.12 Detrend

Link to code
Wraps the executable command 3dDetrend.
This program removes components from voxel time series using linear least squares
For complete details, see the 3dDetrend Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> detrend = afni.Detrend()
>>> detrend.inputs.in_file = 'functional.nii'
>>> detrend.inputs.args = '-polort 2'
>>> detrend.inputs.outputtype = 'AFNI'
>>> detrend.cmdline
'3dDetrend -polort 2 -prefix functional_detrend functional.nii'
>>> res = detrend.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
input file to 3dDetrend
argument: `"%s"`, position: -1

[Optional]
out_file: (a pathlike object or string representing a file)
output image file name

(continues on next page)
argument: `'-prefix %s'`
num_threads: (an integer (int or long), nipype default value: 1)
    set number of threads
outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
    AFNI output filetype
args: (a unicode string)
    Additional parameters to the command
    argument: `'-%s'`
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})
    Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
    output file

References:

None

51.3.13 ECM

Link to code

Wraps the executable command 3dECM.
Performs degree centrality on a dataset using a given maskfile via the 3dECM command
For complete details, see the 3dECM Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> ecm = afni.ECM()
>>> ecm.inputs.in_file = 'functional.nii'
>>> ecm.inputs.mask = 'mask.nii'
>>> ecm.inputs.sparsity = 0.1 # keep top 0.1% of connections
>>> ecm.inputs.out_file = 'out.nii'
>>> ecm.cmdline
'3dECM -mask mask.nii -prefix out.nii -sparsity 0.100000 functional.nii'
>>> res = ecm.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
    input file to 3dECM
    argument: `'-%s'`, position: -1

[Optional]
sparsity: (a float)
    only take the top percent of connections
    argument: `'-sparsity %f'`
full: (a boolean)
    Full power method; enables thresholding; automatically selected if
    -thresh or -sparsity are set
    argument: `'-full'`
fecm: (a boolean)

Fast centrality method; substantial speed increase but cannot
accomodate thresholding; automatically selected if -thresh or
-sparcity are not set
argument: `--fecm`

shift: (a float)

shift correlation coefficients in similarity matrix to enforce non-
negativity, s >= 0.0; default = 0.0 for -full, 1.0 for -fecm
argument: `--shift %f`

scale: (a float)

scale correlation coefficients in similarity matrix to after
shifting, x >= 0.0; default = 1.0 for -full, 0.5 for -fecm
argument: `--scale %f`

eps: (a float)

sets the stopping criterion for the power iteration; l2|v_old -
v_new| < eps*|v_old|; default = 0.001
argument: `--eps %f`

max_iter: (an integer (int or long))

sets the maximum number of iterations to use in the power iteration;
default = 1000
argument: `--max_iter %d`

memory: (a float)

Limit memory consumption on system by setting the amount of GB to
limit the algorithm to; default = 2GB
argument: `--memory %f`

mask: (a pathlike object or string representing an existing file)

mask file to mask input data
argument: `--mask %s`

desc: (a float)

description to exclude connections where corr <= thresh
argument: `--thresh %f`

polort: (an integer (int or long))

argument: `--polort %d`

autoclip: (a boolean)

Clip off low-intensity regions in the dataset
argument: `--autoclip`

automask: (a boolean)

Mask the dataset to target brain-only voxels
argument: `--automask`

num_threads: (an integer (int or long), nipype default value: 1)

set number of threads

outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')

AFNI output filetype

out_file: (a pathlike object or string representing a file)

output image file name
argument: `--prefix %s`

args: (a unicode string)

Additional parameters to the command
argument: `'%s'`

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {}))

Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
51.3.14 Fim

Link to code
Wraps the executable command `3dfim+`.
Program to calculate the cross-correlation of an ideal reference waveform with the measured FMRI time series for each voxel.
For complete details, see the 3dfim+ Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> fim = afni.Fim()
>>> fim.inputs.in_file = 'functional.nii'
>>> fim.inputs.ideal_file = 'seed.1D'
>>> fim.inputs.out_file = 'functional_corr.nii'
>>> fim.inputs.out = 'Correlation'
>>> fim.inputs.fim_thr = 0.0009
>>> fim.cmdline
'3dfim+ -input functional.nii -ideal_file seed.1D -fim_thr 0.000900 -out Correlation -bucket functional_corr.nii'
>>> res = fim.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  input file to 3dfim+
  argument: `'-input %s'`, position: 1
ideal_file: (a pathlike object or string representing an existing file)
  ideal time series file name
  argument: `'-ideal_file %s'`, position: 2

[Optional]
out_file: (a pathlike object or string representing a file)
  output image file name
  argument: `'-bucket %s'`
fim_thr: (a float)
  fim internal mask threshold value
  argument: `'-fim_thr %f'`, position: 3
out: (a unicode string)
  Flag to output the specified parameter
  argument: `'-out %s'`, position: 4
num_threads: (an integer (int or long), nipype default value: 1)
  set number of threads
outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
  AFNI output filetype
args: (a unicode string)
  Additional parameters to the command
51.3.15 Fourier

Link to code
Wraps the executable command 3dFourier.
Program to lowpass and/or highpass each voxel time series in a dataset, via the FFT.
For complete details, see the 3dFourier Documentation.

Examples

```python
def main():
    import nipype
    import nipype.interfaces.afni
    from nipype import Node, IdentityInterface, MapNode
    import os
    refnode = Node(nipype.interfaces.afni.Fourier(), name='FourierNode')
    refnode.inputs.in_file = 'functional.nii'
    refnode.inputs.out_file = 'functional_fourier.nii'
    refnode.inputs.highpass = 0.005
    refnode.inputs.lowpass = 0.1
    refnode.inputs.retrend = True
    res = refnode.run()
```

```
>>> from nipype.interfaces import afni
>>> fourier = afni.Fourier()
>>> fourier.inputs.in_file = 'functional.nii'
>>> fourier.inputs.retrend = True
>>> fourier.inputs.highpass = 0.005
>>> fourier.inputs.lowpass = 0.1
>>> fourier.cmdline
'3dFourier -highpass 0.005000 -lowpass 0.100000 -prefix functional_fourier -\n-\nretrend functional.nii'
>>> res = fourier.run()  # doctest: +SKIP
```

References:

None

51.3.15 Fourier

Link to code
Wraps the executable command 3dFourier.
Program to lowpass and/or highpass each voxel time series in a dataset, via the FFT.
For complete details, see the 3dFourier Documentation.

Examples

```
>>> from nipype.interfaces import afni
>>> fourier = afni.Fourier()
>>> fourier.inputs.in_file = 'functional.nii'
>>> fourier.inputs.retrend = True
>>> fourier.inputs.highpass = 0.005
>>> fourier.inputs.lowpass = 0.1
>>> fourier.cmdline
'3dFourier -highpass 0.005000 -lowpass 0.100000 -prefix functional_fourier -\n-\nretrend functional.nii'
>>> res = fourier.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
    input file to 3dFourier
    argument: ``%s``
    position: -1
lowpass: (a float)
    lowpass
    argument: ``-lowpass %f``
highpass: (a float)
    highpass
    argument: ``-highpass %f``

[Optional]
out_file: (a pathlike object or string representing a file)
    output image file name
    argument: ``-prefix %s``
retrend: (a boolean)
    Any mean and linear trend are removed before filtering. This will restore the trend after filtering.
    argument: ``-retrend``
num_threads: (an integer (int or long), nipype default value: 1)  
set number of threads

outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')  
AFNI output filetype

args: (a unicode string)  
Additional parameters to the command  
argument: '%s'

environ: (a dictionary with keys which are a bytes or None or a value  
of class 'str' and with values which are a bytes or None or a value  
of class 'str', nipype default value: {}))  
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)  
output file

References:

None None

51.3.16 Hist

Link to code
Wraps the executable command 3dHist.  
Computes average of all voxels in the input dataset which satisfy the criterion in the options list  
For complete details, see the 3dHist Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> hist = afni.Hist()
>>> hist.inputs.in_file = 'functional.nii'
>>> hist.cmdline
'3dHist -input functional.nii -prefix functional_hist'
>>> res = hist.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)  
input file to 3dHist  
argument: '-input %s', position: 1

[Optional]
out_file: (a pathlike object or string representing a file)  
Write histogram to niml file with this prefix  
argument: '-prefix %s'

showhist: (a boolean, nipype default value: False)  
write a text visual histogram  
argument: '-showhist'

out_show: (a pathlike object or string representing a file)  
output image file name  
argument: '- > %s', position: -1

mask: (a pathlike object or string representing an existing file)  
matrix to align input file
argument: ``-mask %s``

nbin: (an integer (int or long))
  number of bins
  argument: ``-nbin %d``

max_value: (a float)
  maximum intensity value
  argument: ``-max %f``

min_value: (a float)
  minimum intensity value
  argument: ``-min %f``

bin_width: (a float)
  bin width
  argument: ``-binwidth %f``

args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})

Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
  output file

out_show: (a pathlike object or string representing a file)
  output visual histogram

51.3.17 LFCD

Link to code
Wraps the executable command 3dLFCD.
Performs degree centrality on a dataset using a given maskfile via the 3dLFCD command
For complete details, see the 3dLFCD Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> lfcd = afni.LFCD()
>>> lfcd.inputs.in_file = 'functional.nii'
>>> lfcd.inputs.mask = 'mask.nii'
>>> lfcd.inputs.thresh = 0.8  # keep all connections with corr >= 0.8
>>> lfcd.inputs.out_file = 'out.nii'
>>> lfcd.cmdline
'3dLFCD -mask mask.nii -prefix out.nii -thresh 0.800000 functional.nii'
>>> res = lfcd.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  input file to 3dLFCD
  argument: ```%s``'', position: -1

[Optional]
mask: (a pathlike object or string representing an existing file)
mask file to mask input data
argument: ```-mask %s```

thresh: (a float)
threshold to exclude connections where corr <= thresh
argument: ```-thresh %f```

polort: (an integer (int or long))
argument: ```-polort %d```

autoclip: (a boolean)
Clip off low-intensity regions in the dataset
argument: ```-autoclip```

automask: (a boolean)
Mask the dataset to target brain-only voxels
argument: ```-automask```

num_threads: (an integer (int or long), nipype default value: 1)
set number of threads

outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
AFNI output filetype

out_file: (a pathlike object or string representing a file)
output image file name
argument: ```-prefix %s```

args: (a unicode string)
Additional parameters to the command
argument: ```%s```

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
output file

References:

None None

51.3.18 Maskave

Link to code
Wraps the executable command 3dmaskave.
Computes average of all voxels in the input dataset which satisfy the criterion in the options list
For complete details, see the 3dmaskave Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> maskave = afni.Maskave()
>>> maskave.inputs.in_file = 'functional.nii'
>>> maskave.inputs.mask = 'seed_mask.nii'
>>> maskave.inputs.quiet = True
>>> maskave.cmdline  # doctest: +ELLIPSIS
'3dmaskave -mask seed_mask.nii -quiet functional.nii > functional_maskave.1D'
>>> res = maskave.run()  # doctest: +SKIP
```
**in_file**: (a pathlike object or string representing an existing file)

input file to 3dmaskave

argument: `"%s"`, position: -2

**out_file**: (a pathlike object or string representing a file)

output image file name

argument: `>` %s`, position: -1

**mask**: (a pathlike object or string representing an existing file)

matrix to align input file

argument: `"-mask %s"`, position: 1

**quiet**: (a boolean)

matrix to align input file

argument: `"-quiet"`, position: 2

**num_threads**: (an integer (int or long), nipype default value: 1)

set number of threads

**outputtype**: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')

AFNI output filetype

**args**: (a unicode string)

Additional parameters to the command

argument: `"%s"`

**environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

### Outputs:

**out_file**: (a pathlike object or string representing an existing file)

output file

### References:

None None

### 51.3.19 Means

**Link to code**

Wraps the executable command 3dMean.

Takes the voxel-by-voxel mean of all input datasets using 3dMean

For complete details, see the 3dMean Documentation.

### Examples

```python
>>> from nipype.interfaces import afni
>>> means = afni.Means()
>>> means.inputs.in_file_a = 'im1.nii'
>>> means.inputs.in_file_b = 'im2.nii'
>>> means.inputs.out_file = 'output.nii'
>>> means.cmdline
'3dMean -prefix output.nii im1.nii im2.nii'
>>> res = means.run()  # doctest: +SKIP
```

```python
>>> from nipype.interfaces import afni
>>> means = afni.Means()
```
>>> means.inputs.in_file_a = 'im1.nii'
>>> means.inputs.out_file = 'output.nii'
>>> means.inputs.datum = 'short'
>>> means.cmdline
'3dMean -datum short -prefix output.nii im1.nii'
>>> res = means.run()  # doctest: +SKIP

Inputs:

[Mandatory]
in_file_a: (a pathlike object or string representing an existing file)
    input file to 3dMean
    argument: ``\%s``, position: -2

[Optional]
in_file_b: (a pathlike object or string representing an existing file)
    another input file to 3dMean
    argument: ``\%s``, position: -1
datum: (a unicode string)
    Sets the data type of the output dataset
    argument: ``\-datum \%s``
out_file: (a pathlike object or string representing a file)
    output image file name
    argument: ``\-prefix \%s``
scale: (a unicode string)
    scaling of output
    argument: ``\-%sscale``
non_zero: (a boolean)
    use only non-zero values
    argument: ``\-non_zero``
std_dev: (a boolean)
    calculate std dev
    argument: ``\-stdev``
sqr: (a boolean)
    mean square instead of value
    argument: ``\-sqr``
summ: (a boolean)
    take sum, (not average)
    argument: ``\-sum``
count: (a boolean)
    compute count of non-zero voxels
    argument: ``\-count``
mask_inter: (a boolean)
    create intersection mask
    argument: ``\-mask_inter``
mask_union: (a boolean)
    create union mask
    argument: ``\-mask_union``
num_threads: (an integer (int or long), nipype default value: 1)
    set number of threads
outputtype: (``NIFTI`` or ``AFNI`` or ``NIFTI_GZ``)
    AFNI output filetype
args: (a unicode string)
    Additional parameters to the command
    argument: ``\%s``
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})

Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
output file

References:
None None

51.3.20 OutlierCount

Link to code
Wraps the executable command 3dToutcount.
Calculates number of 'outliers' at each time point of a 3D+time dataset.
For complete details, see the 3dToutcount Documentation

Examples

```python
>>> from nipype.interfaces import afni
>>> toutcount = afni.OutlierCount()
>>> toutcount.inputs.in_file = 'functional.nii'
>>> toutcount.cmdline  # doctest: +ELLIPSIS
'3dToutcount -qthr 0.00100 functional.nii'
>>> res = toutcount.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
input dataset
argument: ``-%s``, position: -2

[Optional]
mask: (a pathlike object or string representing an existing file)
only count voxels within the given mask
argument: ``-mask %s``
mutually_exclusive: autoclip, automask
qthr: (0.0 <= a floating point number <= 1.0, nipype default value: 0.001)
indicate a value for q to compute alpha
argument: ``-qthr %.5f``
autoclip: (a boolean, nipype default value: False)
clip off small voxels
argument: ``-autoclip``
mutually_exclusive: mask
automask: (a boolean, nipype default value: False)
clip off small voxels
argument: ``-automask``
mutually_exclusive: mask
fraction: (a boolean, nipype default value: False)
write out the fraction of masked voxels which are outliers at each
timepoint
argument: ```-fraction```
interval: (a boolean, nipype default value: False)
write out the median + 3.5 MAD of outlier count with each timepoint
argument: ```-range```
save_outliers: (a boolean, nipype default value: False)
enables out_file option
outliers_file: (a pathlike object or string representing a file)
output image file name
argument: ```-save %s```
polort: (an integer (int or long))
detrend each voxel timeseries with polynomials
argument: ```-polort %d```
legendre: (a boolean, nipype default value: False)
use Legendre polynomials
argument: ```-legendre```
out_file: (a pathlike object or string representing a file)
capture standard output
args: (a unicode string)
Additional parameters to the command
argument: ```%s```
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})}
Environment variables

Outputs:

out_outliers: (a pathlike object or string representing an existing
file)
output image file name
out_file: (a pathlike object or string representing a file)
capture standard output

51.3.21 QualityIndex

Link to code
Wraps the executable command 3dTqual.
Computes a ‘quality index’ for each sub-brick in a 3D+time dataset. The output is a 1D time series with the
index for each sub-brick. The results are written to stdout.
For complete details, see the 3dTQual Documentation

Examples

```python
>>> from nipype.interfaces import afni
>>> tqual = afni.QualityIndex()
>>> tqual.inputs.in_file = 'functional.nii'
>>> tqual.cmdline  # doctest: +ELLIPSIS
'3dTqual functional.nii > functional_tqual'
>>> res = tqual.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
input dataset

(continues on next page)
argument: `'\%s'`, position: -2

[Optional]
mask: (a pathlike object or string representing an existing file)
compute correlation only across masked voxels
argument: `'\-mask \%s'`
mutually_exclusive: autoclip, automask

spearman: (a boolean, nipype default value: False)
Quality index is 1 minus the Spearman (rank) correlation coefficient
each sub-brick with the median sub-brick. (default).
argument: `'\-spearman'`

quadrant: (a boolean, nipype default value: False)
Similar to -spearman, but using 1 minus the quadrant correlation
coefficient as the quality index.
argument: `'\-quadrant'`

autoclip: (a boolean, nipype default value: False)
clip off small voxels
argument: `'\-autoclip'`
mutually_exclusive: mask

automask: (a boolean, nipype default value: False)
clip off small voxels
argument: `'\-automask'`
mutually_exclusive: mask

clip: (a float)
clip off values below
argument: `'\-clip %f'`

interval: (a boolean, nipype default value: False)
write out the median + 3.5 MAD of outlier count with each timepoint
argument: `'\-range'`

out_file: (a pathlike object or string representing a file)
capture standard output
argument: `'\> %s'`, position: -1

args: (a unicode string)
Additional parameters to the command
argument: `'\%s'`

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing a file)
file containing the captured standard output

51.3.22  Qwarp

Link to code
Wraps the executable command 3dQwarp.
A version of 3dQwarp Allinante your images prior to passing them to this workflow.
For complete details, see the 3dQwarp Documentation.

Examples
>>> from nipype.interfaces import afni
>>> qwarp = afni.Qwarp()
>>> qwarp.inputs.in_file = 'sub-01_dir-LR_epi.nii.gz'
>>> qwarp.inputs.nopadWARP = True
>>> qwarp.inputs.base_file = 'sub-01_dir-RL_epi.nii.gz'
>>> qwarp.inputs.plusminus = True
>>> qwarp.cmdline
'3dQwarp -base sub-01_dir-RL_epi.nii.gz -source sub-01_dir-LR_epi.nii.gz -nopadWARP -prefix ppp_sub-01_dir-LR_epi -plusminus'
>>> res = qwarp.run() # doctest: +SKIP

>>> from nipype.interfaces import afni
>>> qwarp = afni.Qwarp()
>>> qwarp.inputs.in_file = 'structural.nii'
>>> qwarp.inputs.base_file = 'mni.nii'
>>> qwarp.inputs.resample = True
>>> qwarp.cmdline
'3dQwarp -base mni.nii -source structural.nii -prefix ppp_structural -resample'
>>> res = qwarp.run() # doctest: +SKIP

>>> from nipype.interfaces import afni
>>> qwarp = afni.Qwarp()
>>> qwarp.inputs.in_file = 'structural.nii'
>>> qwarp.inputs.base_file = 'epi.nii'
>>> qwarp.inputs.out_file = 'anatSSQ.nii.gz'
>>> qwarp.inputs.resample = True
>>> qwarp.inputs.lpc = True
>>> qwarp.inputs.verb = True
>>> qwarp.inputs.iwarp = True
>>> qwarp.inputs.blur = [0,3]
>>> qwarp.cmdline
'3dQwarp -base epi.nii -blur 0.0 3.0 -source structural.nii -iwarp -prefix anatSSQ.nii.gz -resample -verb -lpc'
>>> res = qwarp.run() # doctest: +SKIP

>>> from nipype.interfaces import afni
>>> qwarp = afni.Qwarp()
>>> qwarp.inputs.in_file = 'structural.nii'
>>> qwarp.inputs.base_file = 'mni.nii'
>>> qwarp.inputs.duplo = True
>>> qwarp.inputs.minpatch = 25
>>> qwarp.inputs.blur = [0,3]
>>> qwarp.cmdline
'3dQwarp -base mni.nii -blur 0.0 3.0 -duplo -source structural.nii -minpatch 25 -prefix Q25'
>>> res = qwarp.run() # doctest: +SKIP

(continues on next page)
>>> res = qwarp.run()  # doctest: +SKIP
>>> qwarp2 = afni.Qwarp()
>>> qwarp2.inputs.in_file = 'structural.nii'
>>> qwarp2.inputs.base_file = 'mni.nii'
>>> qwarp2.inputs.blur = [0,2]
>>> qwarp2.inputs.out_file = 'Q11'
>>> qwarp2.inputs.inilev = 7
>>> qwarp2.inputs.iniwarp = ['Q25_warp+t1rc.HEAD']
>>> qwarp2.cmdline
'3dQwarp -base mni.nii -blur 0.0 2.0 -source structural.nii -inilev 7 -iniwarp -Q25_warp+t1rc.HEAD -prefix Q11'
>>> res2 = qwarp2.run()  # doctest: +SKIP
>>> res2 = qwarp2.run()  # doctest: +SKIP
>>> qwarp3 = afni.Qwarp()
>>> qwarp3.inputs.in_file = 'structural.nii'
>>> qwarp3.inputs.base_file = 'mni.nii'
>>> qwarp3.inputs.allineate = True
>>> qwarp3.inputs.allineate_opts = '-cose lpa -verb'
>>> qwarp3.cmdline
'3dQwarp -allineate -allineate_opts '-cose lpa -verb' -base mni.nii -source structural.nii -prefix ppp_structural'
>>> res3 = qwarp3.run()  # doctest: +SKIP

Inputs:

[Mandatory]

in_file: (a pathlike object or string representing an existing file)

   Source image (opposite phase encoding direction than base image).

   argument: `'-source %s'`

base_file: (a pathlike object or string representing an existing file)

   Base image (opposite phase encoding direction than source image).

   argument: `'-base %s'`

[Optional]

out_file: (a pathlike object or string representing a file)

   Sets the prefix/suffix for the output datasets.
   * The source dataset is warped to match the base and gets prefix 'ppp'. (Except if '-plusminus' is used
   * The final interpolation to this output dataset is done using the 'wsinc5' method. See the output of
     3dAllineate -HELP
     (in the "Modifying '-final wsinc5'" section) for the lengthy technical details.
   * The 3D warp used is saved in a dataset with
     prefix 'ppp_WARP' -- this dataset can be used
   * To be clear, this is the warp from source dataset
     coordinates to base dataset coordinates, where the
     values at each base grid point are the xyz displacements
     needed to move that grid point's xyz values to the corresponding
     xyz values in the source dataset:
     base( (x,y,z) + WARP(x,y,z) ) matches source(x,y,z)
   * Another way to think of this warp is that it 'pulls' values back from source space to base space.
   * 3dNwarpApply would use 'ppp_WARP' to transform datasets
     aligned with the source dataset to be aligned with the

(continues on next page)
If you do NOT want this warp saved, use the option '-nowarp'.

If you want to calculate and save the inverse 3D warp, use the option '-iwarp'. This inverse warp will then be saved in a dataset with prefix 'ppp_WARPINV'.

This inverse warp could be used to transform data from base space to source space, if you need to do such an operation.

You can easily compute the inverse later, say by a command like

```
3dResample -prefix Z_WARPINV 'INV(Z_WARP+tlrc)'
```
or the inverse can be computed as needed in 3dNwarpApply, like

```
3dNwarpApply -nwarp 'INV(Z_WARP+tlrc)' -source Dataset.nii ...
```

argument: ``-prefix %s``

This option simply resamples the source dataset to match the base dataset grid. You can use this if the two datasets overlap well (as seen in the AFNI GUI), but are not on the same 3D grid. If they don't overlap well, allineate them first. The resampling here is done with the 'wsinc5' method, which has very little blurring artifact.

If the base and source datasets are on the same 3D grid, then the -resample option will be ignored. You can use -resample with these 3dQwarp options:

```
argument: ``-resample``
```

This option will make 3dQwarp run 3dAllineate first, to align the source dataset to the base with an affine transformation. It will then use that alignment as a starting point for the nonlinear warping.

argument: ``-allineate``

add extra options to the 3dAllineate command to be run by 3dQwarp.

argument: ``-allineate_opts %s``

requires: allineate

Do not save the _WARP file.

argument: ``-nowarp``

Do compute and save the _WARPINV file.

argument: ``-iwarp``

Use strict Pearson correlation for matching. Not usually recommended, since the 'clipped Pearson' method used by default will reduce the impact of outlier values.

argument: ``-pear``

Replace negative values in either input volume with 0. If there ARE negative input values, and you do NOT use -noneg, then strict Pearson correlation will be used, since the 'clipped' method only is implemented for non-negative volumes. '-noneg' is not the default, since there might be situations where you want to align datasets with positive and negative values mixed. But, in many cases, the negative values in a dataset are just the result of interpolation artifacts (or other peculiarities), and so they should be ignored. That is what '-noneg' is for.

argument: ``-noneg``
nopenalty: (a boolean)
Replace negative values in either input volume with 0.* If there ARE
negative input values, and you do NOT use -noneg, then strict Pearson
correlation will be used, since the 'clipped' method only is
implemented for non-negative volumes.* '-noneg' is not the default,
since there might be situations where you want to align datasets with
positive and negative values mixed.* But, in many cases, the
negative values in a dataset are just the result of interpolation
artifacts (or other peculiarities), and so they should be ignored.
That is what '-noneg' is for.
argument: "-nopenalty"

penfac: (a float)
Use this value to weight the penalty. The default value is 1. Larger
values mean the penalty counts more, reducing grid
distortions, insha'Allah; '-nopenalty' is the same as '-penfac 0'.

weight: (a pathlike object or string representing an existing file)
Instead of computing the weight from the base dataset, directly input
the weight volume from dataset 'www'. Useful if you know what parts
of the base image you want to emphasize or de-emphasize.
argument: "-weight %s"

noweight: (a boolean)
If you want a binary weight (the old default), use this option. That
is, each voxel in the base volume automask will be weighted the same
in the computation of the cost functional.
argument: "-noweight"

wball: (a list of from 5 to 5 items which are an integer (int or
long))
-wball x y z r f Enhance automatic weight from '-useweight' by a
factor of $f \times \text{Gaussian}(\text{FWHM}=r)$ centered in the base image at DICOM
coordinates $(x, y, z)$ and with radius $r$. The goal of this option is
to try and make the alignment better in a specific part of the
brain.* Example: -wball 0 14 6 30 40 to emphasize the thalamic area
(in MNI/Talairach space).* The $f$ parameter must be positive!* The
$r$ parameter must be between 1 and 100 (inclusive).* '-wball' does
nothing if you input your own weight with the '-weight' option.*
'-wball' does change the binary weight created by the '-noweight'
option.* You can only use '-wball' once in a run of 3dQwarp.*** The
effect of '-wball' is not dramatic. The example above makes the
average brain image across a collection of subjects a little sharper
in the thalamic area, which might have some small value. If you care
enough about alignment to use '-wball', then you should examine
the results from 3dQwarp for each subject, to see if the alignments
are good enough for your purposes.
argument: "-wball %s"

wmask: (a tuple of the form: (a pathlike object or string (continues on next page)
representing an existing file, a float))
-wmask ws fSimilar to '-wball', but here, you provide a dataset
'ws' that indicates where to increase the weight.* The 'ws' dataset
must be on the same 3D grid as the base dataset.* 'ws' is treated as
a mask -- it only matters where it is nonzero -- otherwise, the
values inside are not used.* After 'ws' comes the factor 'f' by
which to increase the automatically computed weight. Where 'ws' is
nonzero, the weighting will be multiplied by (1+f).* As with
'-wball', the factor 'f' should be between 1 and 100.* You cannot
use '-wball' and '-wmask' together!
argument: `\`-wpass %s \f`'

out_weight_file: (a pathlike object or string representing a file)
Write the weight volume to disk as a dataset
argument: `\`-wtprefix %s`'

blur: (a list of from 1 to 2 items which are a float)
Gaussian blur the input images by 'bb' (FWHM) voxels before doing
the alignment (the output dataset will not be blurred). The default is
2.345 (for no good reason).* Optionally, you can provide 2 values
for 'bb', and then the first one is applied to the base volume, the
second to the source volume.---*** e.g., `\`-blur 0 3`' to skip blurring
the base image (if the base is a blurry template, for example).* A
negative blur radius means to use 3D median filtering, rather than
Gaussian blurring. This type of filtering will better preserve edges,
which can be important in alignment.* If the base is a template
volume that is already blurry, you probably don't want to blur it
again, but blurring the source volume a little is probably a good
idea, to help the program avoid trying to match tiny features.* Note
that -duplo will blur the volumes some extra amount for the initial
small-scale warping, to make that phase of the program converge more
rapidly.
argument: `\`-blur %s`'
pblur: (a list of from 1 to 2 items which are a float)
Use progressive blurring; that is, for larger patch sizes, the amount
of blurring is larger. The general idea is to avoid trying to match
finer details when the patch size and incremental warps are coarse.
When '-blur' is used as well, it sets a minimum amount of blurring
that will be used. [06 Aug 2014 -- '-pblur' may become the default
someday].* You can optionally give the fraction of the patch size
that is used for the progressive blur by providing a value between 0
and 0.25 after '-pblur'. If you provide TWO values, the first
fraction is used for progressively blurring the base image and the
second for the source image. The default parameters when just
'-pblur' is given is the same as giving the options as '-pblur 0.09
0.09'.* '-pblur' is useful when trying to match 2 volumes with
high amounts of detail; e.g, warping one subject's brain image
to match another's, or trying to warp to match a detailed template.*
Note that using negative values with '-blur' means that
progressive blurring will be done with median filters, rather than
Gaussian linear blurring.---*** The combination of the -allineate
and -pblur options will make the results of using 3dQwarp to align to
a template somewhat less sensitive to initial head position and
scaling.
argument: `\`-pblur %s`'
emask: (a pathlike object or string representing an existing file)
Here, 'ee' is a dataset to specify a mask of voxelsto EXCLUDE from
the analysis -- all voxels in 'ee' that are NONZERO will not be used
in the alignment.* The base image always automasked --- the emask
isextra, to indicate voxels you definitely DON'T want included in the matching process, even if they are inside the brain.
argument: `'-emask %s`

noXdis: (a boolean)
Warp will not displace in x direction
argument: `'-noXdis'`

noYdis: (a boolean)
Warp will not displace in y direction
argument: `'-noYdis'`

noZdis: (a boolean)
Warp will not displace in z direction
argument: `'-noZdis'`

iniwarp: (a list of items which are a pathlike object or string representing an existing file)
A dataset with an initial nonlinear warp to use.* If this option is not used, the initial warp is the identity.* You can specify a catenation of warps (in quotes) here, as in program 3dNwarpApply.* As a special case, if you just input an affine matrix in a .1D file, that will work also -- it is treated as giving the initial warp via the string "IDENT(base_dataset) matrix_file.aff12.1D".* You CANNOT use this option with -duplo !* -iniwarp is usually used with -inilev to re-start 3dQwarp from a previous stopping point.
argument: `'-iniwarp %s'`
mutually_exclusive: duplo

inilev: (an integer (int or long))
The initial refinement 'level' at which to start.* Usually used with -iniwarp; CANNOT be used with -duplo.* The combination of -inilev and -iniwarp lets you take the results of a previous 3dQwarp run and refine them further:Note that the source dataset in the second run is the SAME asin the first run. If you don't see why this is necessary, then you probably need to seek help from an AFNI guru.
argument: `'-inilev %d'`
mutually_exclusive: duplo

minpatch: (an integer (int or long))
* The value of mm should be an odd integer.* The default value of mm is 25.* For more accurate results than mm=25, try 19 or 13.* The smallest allowed patch size is 5.* You may want to stop at a larger patch size (say 7 or 9) and use the -Qfinal option to run that final level with quintic warps, which might run faster and provide the same degree of warp detail.* Trying to make two different brain volumes match in fine detail is usually a waste of time, especially in humans. There is too much variability in anatomy to match gyrius accurately. For this reason, the default minimum patch size is 25 voxels. Using a smaller '-minpatch' might try to force the warp to match features that do not match, and the result can be useless image distortions -- another reason to LOOK AT THE RESULTS.
argument: `'-minpatch %d'`

maxlev: (an integer (int or long))
The initial refinement 'level' at which to start.* Usually used with -iniwarp; CANNOT be used with -duplo.* The combination of -inilev and -iniwarp lets you take the results of a previous 3dQwarp run and refine them further:Note that the source dataset in the second run is the SAME asin the first run. If you don't see why this is necessary, then you probably need to seek help from an AFNI guru.
argument: `'-maxlev %d'`, position: -1
mutually_exclusive: duplo

gridlist: (a pathlike object or string representing an existing file)
This option provides an alternate way to specify the patchgrid sizes used in the warp optimization process. 'gl' is a 1D file with a list of patches to use -- in most cases, you will want to use it in the following form: `-gridlist '1D: 0 151 101 75 51'`. Here, a 0 patch size means the global domain. Patch sizes otherwise should be odd integers >= 5. If you use the '0' patch size again after the first position, you will actually get an iteration at the size of the default patch level 1, where the patch sizes are 75% of the volume dimension. There is no way to force the program to literally repeat the sui generis step of lev=0. You cannot use `-gridlist` with `-duplo` or `-plusminus`.

Argument: `'-gridlist %s'`

Mutually exclusive: `duplo`, `plusminus`

`allsave`: (a boolean)

This option lets you save the output warps from each level of the refinement process. Mostly used for experimenting. Cannot be used with `-nopadWARP`, `-duplo`, or `-plusminus`. Will only save all the outputs if the program terminates normally -- if it crashes, or freezes, then all these warps are lost.

Argument: `'-allsave'`

Mutually exclusive: `nopadWARP`, `duplo`, `plusminus`

`duplo`: (a boolean)

Start off with 1/2 scale versions of the volumes, for getting a speedy coarse first alignment. Then scales back up to register the full volumes. The goal is greater speed, and it seems to help this positively piggish program to be more expeditious. However, accuracy is somewhat lower with '-duplo', for reasons that currently elude Zhark; for this reason, the Emperor does not usually use '-duplo'.

Argument: `'-duplo'`

Mutually exclusive: `gridlist`, `maxlev`, `inilev`, `iniwarp`, `plusminus`, `allsave`

`workhard`: (a boolean)

Iterate more times, which can help when the volumes are hard to align at all, or when you hope to get a more precise alignment. Slows the program down (possibly a lot), of course. When you combine '-workhard' with '-duplo', only the full size volumes get the extra iterations. For finer control over which refinement levels work hard, you can use this option in the form (for example) `-workhard:4:7` which implies the extra iterations will be done at levels 4, 5, 6, and 7, but not otherwise. You can also use '-superhard' to iterate even more, but this extra option will REALLY slow things down. Under most circumstances, you should not need to use either `-workhard` or `-superhard`.

Argument: `'-workhard'`

Mutually exclusive: `boxopt`, `ballopt`

`Qfinal`: (a boolean)

At the finest patch size (the final level), use Hermite quintic polynomials for the warp instead of cubic polynomials. In a 3D 'patch', there are 2x2x2x3=24 cubic polynomial basis function parameters over which to optimize (2 polynomials dependent on each of the x,y,z directions, and 3 different directions of displacement). There are 3x3x3x3=81 quintic polynomial parameters per patch. With
-Qfinal, the final level will have more detail in the allowed warps, at the cost of yet more CPU time.* However, no patch below 7x7x7 in size will be done with quintic polynomials.* This option is also not usually needed, and is experimental.

argument: `--Qfinal`

Qonly: (a boolean)

Use Hermite quintic polynomials at all levels.* Very slow (about 4 times longer). Also experimental.* Will produce a (discrete representation of a) C2 warp.

argument: `--Qonly`

plusminus: (a boolean)

Normally, the warp displacements \( \text{dis}(x) \) are defined to match \( \text{base}(x) \) to source\( (x+\text{dis}(x)) \). With this option, the match is between base\( (x-\text{dis}(x)) \) and source\( (x+\text{dis}(x)) \) -- the two images 'meet in the middle'.* One goal is to mimic the warping done to MRI EPI data by field inhomogeneities, when registering between a 'blip up' and a 'blip down' down volume, which will have opposed distortions.* Define \( W_p(x) = x + \text{dis}(x) \) and \( W_m(x) = x - \text{dis}(x) \). Then since base \( W_m(x) \) matches source \( W_p(x) \), by substituting \( \text{INV}(W_m(x)) \) wherever we see \( x \), we have base\( (x) \) matches source\( (W_p(\text{INV}(W_m(x)))) \); that is, the warp \( V(x) \) that one would get from the 'usual' way of running 3dQwarp is \( V(x) = W_p(\text{INV}(W_m(x))) \).* Conversely, we can calculate \( W_p(x) \) in terms of \( V(x) \) as follows: If \( V(x) = x + dv(x) \), define \( V_h(x) = x + \frac{dv(x)}{2} \); then \( W_p(x) = V(\text{INV}(V_h(x))) \).* With the above formulas, it is possible to compute \( W_p(x) \) from \( V(x) \) and vice-versa, using program 3dNwarpCalc. The requisite commands are left as an exercise for the aspiring AFNI Jedi Master.* You can use the semi-secret `--pmBASE` option to get the \( V(x) \) warp and the source dataset warped to base space, in addition to the \( W_p(x) \) '_PLUS' and \( W_m(x) \) '_MINUS' warps.* Alas: `--plusminus` does not work with `--duplo` or `--allsave` :-(* However, you can use `--iniwarp` with `--plusminus` :-(*) The outputs have _PLUS (from the source dataset) and _MINUS (from the base dataset) in their filenames, in addition to the prefix. The `--iwire` option, if present, will be ignored.

argument: `--plusminus`

mutually_exclusive: duplo, allsave, iwire

nopad: (a boolean)

Do NOT use zero-padding on the 3D base and source images.[Default == zero-pad, if needed]* The underlying model for deformations goes to zero at the edge of the volume being warped. However, if there is significant data near an edge of the volume, then it won't get displaced much, and so the results might not be good.* Zero padding is designed as a way to work around this potential problem. You should NOT need the `--nopad` option for any reason that Zhark can think of, but it is here to be symmetrical with 3dAllineate.* Note that the output (warped from source) dataset will be on the base dataset grid whether or not zero-padding is allowed. However, unless you use the following option, allowing zero-padding (i.e., the default operation) will make the output WARP dataset(s) be on a larger grid (also see `--expad` below).

argument: `--nopad`

nopadWARP: (a boolean)

If for some reason you require the warp volume to match the base volume, then use this option to have the output WARP dataset(s) truncated.

argument: `--nopadWARP`

mutually_exclusive: allsave, expad

(continues on next page)
expad: (an integer (int or long))
This option instructs the program to pad the warp by an extra 'EE'
voxels (and then 3DQwarp starts optimizing it).* This option is
seldom needed, but can be useful if you might later catenate the
nonlinear warp -- via 3DQwarpCat -- with an affine transformation
that contains a large shift. Under that circumstance, the nonlinear
warp might be shifted partially outside its original grid, so
expanding that grid can avoid this problem.* Note that this option
perforce turns off '-nopadWARP'.
argument: `-expad %d`
mutually_exclusive: nopadWARP

ballopt: (a boolean)
Normally, the incremental warp parameters are optimized inside a
rectangular 'box' (24 dimensional for cubic patches, 81 for quintic
patches), whose limits define the amount of distortion allowed at
each step. Using '-ballopt' switches these limits to be applied to a
'ball' (interior of a hypersphere), which can allow for larger
incremental displacements. Use this option if you think things need
to be able to move farther.
argument: `-ballopt`
mutually_exclusive: workhard, ballopt

boxopt: (a boolean)
Use the 'box' optimization limits instead of the 'ball'[this is the
default at present].* Note that if '-workhard' is used, then ball
and box optimization are alternated in the different iterations at
each level, so these two options have no effect in that case.
argument: `-boxopt`
mutually_exclusive: workhard, ballopt

verb: (a boolean)
more detailed description of the process
argument: `-verb`
mutually_exclusive: quiet

quiet: (a boolean)
Cut out most of the fun fun fun progress messages :-(
argument: `-quiet`
mutually_exclusive: verb

overwrite: (a boolean)
Overwrite outputs
argument: `-overwrite`

lpc: (a boolean)
Local Pearson minimization (i.e., EPI-T1 registration) This option
has not been extensively tested If you use '-lpc', then '-maxlev 0' is
automatically set. If you want to go to more refined levels, you can
set '-maxlev' This should be set up to have lpc as the second to last
argument and maxlev as the second to last argument, as needed by
AFNI Using maxlev > 1 is not recommended for EPI-T1 alignment.
argument: `-lpc` position: -2
mutually_exclusive: nmi, mi, hel, lpa, pear

lpa: (a boolean)
Local Pearson maximization This option has not been extensively tested
argument: `-lpa`
mutually_exclusive: nmi, mi, lpc, hel, pear

hel: (a boolean)
Hellinger distance: a matching function for the adventurous This
option has NOT be extensively tested for usefulness and should be
considered experimental at this infundibulum.
argument: `-hel`
mutually_exclusive: nmi, mi, lpc, lpa, pear

mi: (a boolean)
   Mutual Information: a matching function for the adventurous. This option has NOT be extensively tested for usefulness and should be considered experimental at this infundibulum.
   argument: `-mi`
   mutually_exclusive: mi, hel, lpc, lpa, pear

nmi: (a boolean)
   Normalized Mutual Information: a matching function for the adventurous. This option has NOT be extensively tested for usefulness and should be considered experimental at this infundibulum.
   argument: `-nmi`
   mutually_exclusive: nmi, hel, lpc, lpa, pear

num_threads: (an integer (int or long), nipype default value: 1)
   set number of threads

outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
   AFNI output filetype

args: (a unicode string)
   Additional parameters to the command
   argument: `%-s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
   Environment variables

Outputs:

warped_source: (a pathlike object or string representing a file)
   Warped source file. If plusminus is used, this is the undistorted source file.

warped_base: (a pathlike object or string representing a file)
   Undistorted base file.

source_warp: (a pathlike object or string representing a file)
   Displacement in mm for the source image. If plusminus is used, this is the field susceptibility correction warp (in 'mm') for source image.

base_warp: (a pathlike object or string representing a file)
   Displacement in mm for the base image. If plus minus is used, this is the field susceptibility correction warp (in 'mm') for base image. This is only output if plusminus or iwarp options are passed.

weights: (a pathlike object or string representing a file)
   Auto-computed weight volume.

References:

None None

51.3.23 QwarpPlusMinus

Link to code
Wraps the executable command 3dQwarp.
A version of 3dQwarp for performing field susceptibility correction using two images with opposing phase encoding directions.
For complete details, see the 3dQwarp Documentation.
Examples

```python
>>> from nipype.interfaces import afni
>>> qwarp = afni.QwarpPlusMinus()
>>> qwarp.inputs.in_file = 'sub-01_dir-LR_epi.nii.gz'
>>> qwarp.inputs.nopadWARP = True
>>> qwarp.inputs.base_file = 'sub-01_dir-RL_epi.nii.gz'
>>> qwarp.cmdline
'3dQwarp -prefix Qwarp.nii.gz -plusminus -base sub-01_dir-RL_epi.nii.gz -source sub-01_dir-LR_epi.nii.gz -nopadWARP'
>>> res = warp.run()  # doctest: +SKIP
```

Inputs:

<table>
<thead>
<tr>
<th>Mandatory</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_file: (a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>Source image (opposite phase encoding direction than base image).</td>
</tr>
<tr>
<td>argument: <code>'-source %s'</code></td>
</tr>
<tr>
<td>base_file: (a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>Base image (opposite phase encoding direction than source image).</td>
</tr>
<tr>
<td>argument: <code>'-base %s'</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_file: (a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>Source image (opposite phase encoding direction than base image)</td>
</tr>
<tr>
<td>argument: <code>'-source %s'</code></td>
</tr>
<tr>
<td>out_file: (a pathlike object or string representing a file, nipype default value: Qwarp.nii.gz)</td>
</tr>
<tr>
<td>Output file</td>
</tr>
<tr>
<td>argument: <code>'-prefix %s'</code>, position: 0</td>
</tr>
<tr>
<td>plusminus: (a boolean, nipype default value: True)</td>
</tr>
<tr>
<td>Normally, the warp displacements ( \text{dis}(x) ) are defined to match ( \text{base}(x) ) to ( \text{source}(x+\text{dis}(x)) ). With this option, the match is between ( \text{base}(x-\text{dis}(x)) ) and ( \text{source}(x+\text{dis}(x)) ) -- the two images 'meet in the middle'. For more info, view Qwarp interface</td>
</tr>
<tr>
<td>argument: <code>'-plusminus'</code>, position: 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>mutually_exclusive: duplo, allsave, iwarp</th>
</tr>
</thead>
<tbody>
<tr>
<td>resample: (a boolean)</td>
</tr>
<tr>
<td>This option simply resamples the source dataset to match the base dataset grid. You can use this if the two datasetsoverlap well (as seen in the AFNI GUI), but are not on thesame 3D grid. If they don't overlap well, alignate them first. The reampling here is done with the 'wsubic5' method, which has very little blurring artifact. If the base and source datasets ARE on the same 3D grid, then the -resample option will be ignored. You CAN use -resample with these 3dQwarp options: -plusminus -inilev -iniwarp -duplo</td>
</tr>
<tr>
<td>argument: <code>'-resample'</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>allinate: (a boolean)</th>
</tr>
</thead>
<tbody>
<tr>
<td>This option will make 3dQwarp run 3dAllinate first, to align the source dataset to the base with an affine transformation. It will then use that alignment as a starting point for the nonlinear warping.</td>
</tr>
<tr>
<td>argument: <code>'-allinate'</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>allinate_opts: (a unicode string)</th>
</tr>
</thead>
<tbody>
<tr>
<td>add extra options to the 3dAllinate command to be run by 3dQwarp.</td>
</tr>
<tr>
<td>argument: <code>'-allinate_opts %s'</code></td>
</tr>
</tbody>
</table>

(continues on next page)
requires: allineate
nowarp: (a boolean)
    Do not save the _WARP file.
    argument: `-nowarp`

iwarp: (a boolean)
    Do compute and save the _WARPINV file.
    argument: `-iwarp`
    mutually_exclusive: plusminus

pear: (a boolean)
    Use strict Pearson correlation for matching. Not usually recommended, since the 'clipped Pearson' method used by default will reduce the impact of outlier values.
    argument: `-pear`

noneg: (a boolean)
    Replace negative values in either input volume with 0. If there ARE negative input values, and you do NOT use -noneg, then strict Pearson correlation will be used, since the 'clipped' method only is implemented for non-negative volumes. `-noneg` is not the default, since there might be situations where you want to align datasets with positive and negative values mixed. But, in many cases, the negative values in a dataset are just the result of interpolation artifacts (or other peculiarities), and so they should be ignored. That is what `-noneg` is for.
    argument: `-noneg`

nopenalty: (a boolean)
    Replace negative values in either input volume with 0. If there ARE negative input values, and you do NOT use -noneg, then strict Pearson correlation will be used, since the 'clipped' method only is implemented for non-negative volumes. `-noneg` is not the default, since there might be situations where you want to align datasets with positive and negative values mixed. But, in many cases, the negative values in a dataset are just the result of interpolation artifacts (or other peculiarities), and so they should be ignored. That is what `-noneg` is for.
    argument: `-nopenalty`

penfac: (a float)
    Use this value to weight the penalty. The default value is 1. Larger values mean the penalty counts more, reducing grid distortions, insha'Allah; `-nopenalty` is the same as `-penfac 0`. 
    [23 Sep 2013] -- Zbark increased the default value of the penalty by a factor of 5, and also made it get progressively larger with each level of refinement. Thus, warping results will vary from earlier instances of 3dQwarp. The progressive increase in the penalty at higher levels means that the 'cost function' can actually look like the alignment is getting worse when the levels change. * IF you wish to turn off this progression, for whatever reason (e.g., to keep compatibility with older results), use the option `-penold`. To be completely compatible with the older 3dQwarp, you'll also have to use `-penfac 0.2`.
    argument: `-penfac %f`

noweight: (a boolean)
    If you want a binary weight (the old default), use this option. That is, each voxel in the base volume automask will be weighted the same in the computation of the cost functional.
    argument: `-noweight`

weight: (a pathlike object or string representing an existing file)
    Instead of computing the weight from the base dataset, directly input
the weight volume from dataset 'www'.* Useful if you know what over
parts of the base image you want to emphasize or de-emphasize the
matching functional.
argument: ``-weight %s``

wball: (a list of from 5 to 5 items which are an integer (int or
long))
- wball x y z r fEnhance automatic weight from '-useweight' by a
factor of 1+f*Gaussian(FWHM=r) centered in the base image at DICOM
coordinates (x,y,z) and with radius 'r'. The goal of this option is
to try and make the alignment better in a specific part of the
brain.* Example: -wball 0 14 6 30 40 to emphasize the thalamic area
(in MNI/Talairach space).* The 'r' parameter must be positive!* The
'f' parameter must be between 1 and 100 (inclusive).* '-wball' does
nothing if you input your own weight with the '-weight' option.*
'-wball' does change the binary weight created by the '-neweight'
option.* You can only use '-wball' once in a run of 3dQwarp.** The
effect of '-wball' is not dramatic. The example above makes the
average brain image across a collection of subjects a little sharper
in the thalamic area, which might have some small value. If you care
even about alignment to use '-wball', then you should examine
the results from 3dQwarp for each subject, to see if the alignments
are good enough for your purposes.
argument: ``-wball %s``

wmask: (a tuple of the form: (a pathlike object or string
representing an existing file, a float))
- wmask ws fSimilar to '-wball', but here, you provide a dataset
'ws' that indicates where to increase the weight.* The 'ws' dataset
must be on the same 3D grid as the base dataset.* 'ws' is treated as
a mask -- it only matters where it is nonzero -- otherwise, the
values inside are not used.* After 'ws' comes the factor 'f' by
which to increase the automatically computed weight. Where 'ws'
is nonzero, the weighting will be multiplied by (1+f).* As with
'-wball', the factor 'f' should be between 1 and 100.* You cannot
use '-wball' and '-wmask' together!
argument: ``-wpass %s %f``

out_weight_file: (a pathlike object or string representing a file)
Write the weight volume to disk as a dataset
argument: ``-wtprefix %s``

blur: (a list of from 1 to 2 items which are a float)
Gaussian blur the input images by 'bb' (FWHM) voxels before doing
the alignment (the output dataset will not be blurred). The default is
2.345 (for no good reason).* Optionally, you can provide 2 values
for 'bb', and then the first one is applied to the base volume, the
second to the source volume.--- e.g., '-blur 0 3' to skip blurring
the base image (if the base is a blurry template, for example).* A
negative blur radius means to use 3D median filtering, rather than
Gaussian blurring. This type of filtering will better preserve edges,
which can be important in alignment.* If the base is a template
volume that is already blurry, you probably don't want to blur it
again, but blurring the source volume a little is probably a good
idea, to help the program avoid trying to match tiny features.* Note
that -duplo will blur the volumes some extra amount for the initial
small-scale warping, to make that phase of the program converge more
rapidly.
argument: ``-blur %s``
pblur: (a list of from 1 to 2 items which are a float)
Use progressive blurring; that is, for larger patch sizes, the amount
(continues on next page)
of blurring is larger. The general idea is to avoid trying to match finer details when the patch size and incremental warps are coarse. When 'blur' is used as well, it sets a minimum amount of blurring that will be used. [06 Aug 2014 -- 'blur' may become the default someday].

You can optionally give the fraction of the patch size that is used for the progressive blur by providing a value between 0 and 0.25 after 'blur'. If you provide two values, the first fraction is used for progressively blurring the base image and the second for the source image. The default parameters when just 'blur' is given is the same as giving the options as 'blur 0.09 0.09'. 'blur' is useful when trying to match two volumes with high amounts of detail; e.g., warping one subject's brain image to match another's, or trying to warp to match a detailed template.

Note that using negative values with 'blur' means that the progressive blurring will be done with median filters, rather than Gaussian linear blurring.

The combination of the -allineate and -blur options will make the results of using 3dQwarp to align to a template somewhat less sensitive to initial head position and scaling.

You can optionally give the fraction of the patch size that is used for the progressive blur by providing a value between 0 and 0.25 after 'blur'. If you provide two values, the first fraction is used for progressively blurring the base image and the second for the source image. The default parameters when just 'blur' is given is the same as giving the options as 'blur 0.09 0.09'. 'blur' is useful when trying to match two volumes with high amounts of detail; e.g., warping one subject's brain image to match another's, or trying to warp to match a detailed template.

Note that using negative values with 'blur' means that the progressive blurring will be done with median filters, rather than Gaussian linear blurring.

The combination of the -allineate and -blur options will make the results of using 3dQwarp to align to a template somewhat less sensitive to initial head position and scaling.

You can optionally give the fraction of the patch size that is used for the progressive blur by providing a value between 0 and 0.25 after 'blur'. If you provide two values, the first fraction is used for progressively blurring the base image and the second for the source image. The default parameters when just 'blur' is given is the same as giving the options as 'blur 0.09 0.09'. 'blur' is useful when trying to match two volumes with high amounts of detail; e.g., warping one subject's brain image to match another's, or trying to warp to match a detailed template.

Note that using negative values with 'blur' means that the progressive blurring will be done with median filters, rather than Gaussian linear blurring.

The combination of the -allineate and -blur options will make the results of using 3dQwarp to align to a template somewhat less sensitive to initial head position and scaling.
minpatch: (an integer (int or long))
* The value of mm should be an odd integer.* The default value of mm
is 25.* For more accurate results than mm=25, try 19 or 13.* The
smallest allowed patch size is 5.* You may want stop at a larger
patch size (say 7 or 9) and use the -Qfinal option to run that final
level with quintic warps, which might run faster and provide the same
degree of warp detail.* Trying to make two different brain volumes
match in fine detail is usually a waste of time, especially in
humans. There is too much variability in anatomy to match gyri to
gyrius accurately. For this reason, the default minimum patch size is
25 voxels. Using a smaller '-minpatch' might try to force the warp
tomatch features that do not match, and the result can be
useless image distortions — another reason to LOOK AT THE RESULTS.
argument: '--minpatch %d'

maxlev: (an integer (int or long))
The initial refinement 'level' at which to start.* Usually used with
-iniwarp; CANNOT be used with -duplo.* The combination of -inilev
and -iniwarp lets you take the results of a previous 3dQwarp run and
refine them further: Note that the source dataset in the second run
is the SAME as the first run. If you don't see why this is
necessary, then you probably need to seek help from an AFNI guru.
argument: '--maxlev %d', position: -1

gridlist: (a pathlike object or string representing an existing file)
This option provides an alternate way to specify the patchgrid sizes
used in the warp optimization process. 'gl' is a 1D file with a list
of patches to use — in most cases, you will want to use it in the
following form: -gridlist '1D: 0 151 101 75 51' Here, a 0 patch size
means the global domain. Patch sizes otherwise should be odd integers
>= 5.* If you use the '0' patch size again after the first
position, you will actually get an iteration at the size of
the default patch level 1, where the patch sizes are 75% of the volume
dimension. There is no way to force the program to literally repeat
the sui generis step of lev=0.* You cannot use -gridlist with -duplo
or -plusminus!
argument: '--gridlist $s'

allsave: (a boolean)
This option lets you save the output warps from each level of the
refinement process. Mostly used for experimenting.* Cannot be used with
-nopadWARP, -duplo, or -plusminus.* Will only save all the
outputs if the program terminates normally — if it crashes, or
freezes, then all these warps are lost.
argument: '--allsave'

duplo: (a boolean)
Start off with 1/2 scale versions of the volumes, for getting a
speedy coarse first alignment.* Then scales back up to register the
full volumes. The goal is greater speed, and it seems to help
this positively piggish program to be more expeditious.* However,
accuracy is somewhat lower with '-duplo', for reasons that currently
evade Zhark; for this reason, the Emperor does not usually use
'-duplo'.
argument: '--duplo'
mutable_exclusive: gridlist, maxlev, inilev, iniwarp, plusminus, allsave

workhard: (a boolean)
Iterate more times, which can help when the volumes are hard to align at all, or when you hope to get a more precise alignment. Slows the program down (possibly a lot), of course. When you combine `-workhard` with `-duplo`, only the full size volumes get the extra iterations. For finer control over which refinement levels work hard, you can use this option in the form (for example)

```
-workhard:4:7
```

which implies the extra iterations will be done at levels 4, 5, 6, and 7, but not otherwise. You can also use `-superhard` to iterate even more, but this extra option will REALLY slow things down. Under most circumstances, you should not need to use either `-workhard` or `-superhard`. The fastest way to register to a template image is via the `-duplo` option, and without the `-workhard` or `-superhard` options.

- **Qfinal**: (a boolean)
  - At the finest patch size (the final level), use Hermite quintic polynomials for the warp instead of cubic polynomials. In a 3D `patch`, there are $2 \times 2 \times 2 = 8$ cubic polynomial basis function parameters over which to optimize (2 polynomials dependent on each of the x, y, z directions, and 3 different directions of displacement). There are $3 \times 3 \times 3 = 27$ quintic polynomial parameters per patch. With `-Qfinal`, the final level will have more detail in the allowed warps, at the cost of yet more CPU time. However, no patch below 7x7x7 in size will be done with quintic polynomials. This option is also not usually needed, and is experimental.

- **Qonly**: (a boolean)
  - Use Hermite quintic polynomials at all levels. Very slow (about 4 times longer). Also experimental. Will produce a (discrete representation of a) C2 warp.

- **nopad**: (a boolean)
  - Do NOT use zero-padding on the 3D base and source images. [Default == zero-pad, if needed] The underlying model for deformations goes to zero at the edge of the volume being warped. However, if there is significant data near an edge of the volume, then it won't get displaced much, and so the results might not be good. Zero padding is designed as a way to work around this potential problem. You should NOT need the `-nopad` option for any reason that 2hark can think of, but it is here to be symmetrical with 3dAllineate. Note that the output (warped from source) dataset will be on the base dataset grid whether or not zero-padding is allowed. However, unless you use the following option, allowing zero-padding (i.e., the default operation) will make the output WARP dataset(s) be on a larger grid (also see `-expad` below).

- **nopadWARP**: (a boolean)
  - If for some reason you require the warp volume to match the base volume, then use this option to have the output WARP dataset(s) truncated.

- **expad**: (an integer (int or long))
  - This option instructs the program to pad the warp by an extra 'EE'
voxels (and then 3dQwarp starts optimizing it). * This option is seldom needed, but can be useful if you might later catenate the nonlinear warp -- via 3dNwarpCat -- with an affine transformation that contains a large shift. Under that circumstance, the nonlinear warp might be shifted partially outside its original grid, so expanding that grid can avoid this problem. * Note that this option performsturns off `-nopadWARP'.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-expad %d</code></td>
<td>Mutually exclusive: <code>nopadWARP</code></td>
</tr>
<tr>
<td><code>-ballopt</code></td>
<td>Normally, the incremental warp parameters are optimized inside a rectangular 'box' (24 dimensional for cubic patches, 81 for quintic patches), whose limits define the amount of distortion allowed at each step. Using <code>-ballopt</code> switches these limits to be applied to a 'ball' (interior of a hypersphere), which can allow for larger incremental displacements. Use this option if you think things need to be able to move farther.</td>
</tr>
<tr>
<td><code>-boxopt</code></td>
<td>Use the 'box' optimization limits instead of the 'ball' [this is the default at present]. * Note that if <code>-workhard</code> is used, then ball and box optimization are alternated in the different iterations at each level, so these two options have no effect in that case.</td>
</tr>
<tr>
<td><code>-verb</code></td>
<td>More detailed description of the process</td>
</tr>
<tr>
<td><code>-quiet</code></td>
<td>Cut out most of the fun fun fun progress messages :-(</td>
</tr>
<tr>
<td><code>-overwrite</code></td>
<td>Overwrite outputs</td>
</tr>
<tr>
<td><code>-lpc</code></td>
<td>Local Pearson minimization (i.e., EPI-T1 registration) This option has not be extensively tested If you use <code>-lpc</code>, then <code>-maxlev 0</code> is automatically set. If you want to go to more refined levels, you can set <code>-maxlev</code> This should be set up to have <code>lpc</code> as the second to last argument and <code>maxlev</code> as the second to last argument, as needed by AFNI Using <code>maxlev &gt; 1</code> is not recommended for EPI-T1 alignment.</td>
</tr>
<tr>
<td><code>-lpa</code></td>
<td>Local Pearson maximization This option has not be extensively tested</td>
</tr>
<tr>
<td><code>-hel</code></td>
<td>Hellinger distance: a matching function for the adventurous This option has NOT be extensively tested for usefulness and should be considered experimental at this infundibulum.</td>
</tr>
</tbody>
</table>

(continues on next page)
Mutual Information: a matching function for the adventurous
This option has NOT be extensively tested for usefulness and should be
considered experimental at this infundibulum.
argument: ``-mi``
mutually_exclusive: mi, hel, lpc, lpa, pear

nmi: (a boolean)
Normalized Mutual Information: a matching function for the
adventurous
This option has NOT be extensively tested for usefulness and should be
considered experimental at this infundibulum.
argument: ``-nmi``
mutually_exclusive: nmi, hel, lpc, lpa, pear

num_threads: (an integer (int or long), nipype default value: 1)
set number of threads

outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
AFNI output filetype

args: (a unicode string)
Additional parameters to the command
argument: ``%s``

eviron: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

warped_source: (a pathlike object or string representing a file)
Warped source file. If plusminus is used, this is the
undistorted source file.

warped_base: (a pathlike object or string representing a file)
Undistorted base file.

source_warp: (a pathlike object or string representing a file)
Displacement in mm for the source image. If plusminus is used this is
the field susceptibility correction warp (in 'mm') for source image.

base_warp: (a pathlike object or string representing a file)
Displacement in mm for the base image. If plusminus is used, this is
the field susceptibility correction warp (in 'mm') for base image.
This is only output if plusminus or iwarp options are passed

weights: (a pathlike object or string representing a file)
Auto-computed weight volume.

References:
None None

51.3.24 ROIStats

Link to code
Wraps the executable command 3dROIstats.
Display statistics over masked regions
For complete details, see the 3dROIstats Documentation

Examples
>>> from nipype.interfaces import afni
>>> roistats = afni.ROIStats()
>>> roistats.inputs.in_file = 'functional.nii'
>>> roistats.inputs.mask_file = 'skeleton_mask.nii.gz'
>>> roistats.inputs.stat = ['mean', 'median', 'voxels']
>>> roistats.inputs.nomeanout = True
>>> roistats.cmdline
'3dROIstats -mask skeleton_mask.nii.gz -nomeanout -nzmean -nzmedian -nzvoxels_→functional.nii > functional_roistat.1D'
>>> res = roistats.run()  # doctest: +SKIP

Inputs:

[Mandatory]

in_file: (a pathlike object or string representing an existing file)
input dataset
argument: `"%s"`, position: -2

[Optional]

mask: (a pathlike object or string representing an existing file)
input mask
argument: `"-mask %s"`, position: 3

mask_file: (a pathlike object or string representing an existing file)
input mask
argument: `"-mask %s"

mask_f2short: (a boolean)
Tell the program to convert a float mask to short integers, by simple rounding.
argument: `"-mask_f2short`

num_roi: (an integer (int or long))
Forces the assumption that the mask dataset's ROIs are denoted by 1 to n inclusive. Normally, the program figures out the ROIs on its own. This option is useful if a) you are certain that the mask dataset has no values outside the range [0 n], b) there may be some ROIs missing between [1 n] in the mask data-set and c) you want those columns in the output any-way so the output lines up with the output from other invocations of 3dROIstats.
argument: `"-numroi %s`

zerofill: (a unicode string)
For ROI labels not found, use the provided string instead of a '0' in the output file. Only active if `num_roi` is enabled.
argument: `"-zerofill %s`
requires: num_roi

roisel: (a pathlike object or string representing an existing file)
only considers ROIs denoted by values found in the specified file.
Note that the order of the ROIs as specified in the file is not preserved. So an SEL.1D of '2 8 20' produces the same output as '8 20 2'
argument: `"-roisel %s`

debug: (a boolean)
print debug information
argument: `"-debug`

quiet: (a boolean)
execute quietly
argument: `"-quiet`

nomeanout: (a boolean)
Do not include the (zero-inclusive) mean among computed stats

(continues on next page)
argument: `\`-nomeanout\`

nobriklab: (a boolean)
Do not print the sub-brick label next to its index
argument: `\`-nobriklab\`

format1D: (a boolean)
Output results in a 1D format that includes commented labels
argument: `\`-1Dformat\`
mutually_exclusive: format1DR

format1DR: (a boolean)
Output results in a 1D format that includes uncommented labels. May not work optimally with typical 1D functions, but is useful for R functions.
argument: `\`-1DRformat\`
mutually_exclusive: format1D

stat: (a list of items which are 'mean' or 'sum' or 'voxels' or 'minmax' or 'sigma' or 'median' or 'mode' or 'summary' or 'zerominmax' or 'zerosigma' or 'zeromedian' or 'zeromode')
statistics to compute. Options include:
* mean = Compute the mean using only non_zero voxels. Implies the opposite for the mean computed by default.
* median = Compute the median of nonzero voxels
* mode = Compute the mode of nonzero voxels. (integral valued sets only)
* minmax = Compute the min/max of nonzero voxels
* sum = Compute the sum using only nonzero voxels.
* voxels = Compute the number of nonzero voxels
* sigma = Compute the standard deviation of nonzero voxels

Statistics that include zero-valued voxels:
* zerominmax = Compute the min/max of all voxels.
* zerosigma = Compute the standard deviation of all voxels.
* zeromedian = Compute the median of all voxels.
* zeromode = Compute the mode of all voxels.
* summary = Only output a summary line with the grand mean across all briks in the input dataset. This option cannot be used with nomeanout.
More that one option can be specified.
argument: `\`%s...\`

out_file: (a pathlike object or string representing a file)
output file
argument: `\>` %s\`, position: -1

args: (a unicode string)
Additional parameters to the command
argument: `\`%s\`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
output tab-separated values file

51.3.25 Retroicor

Link to code
Wraps the executable command 3dretroicor.
Performs Retrospective Image Correction for physiological motion effects, using a slightly modified version of the RETROICOR algorithm

The durations of the physiological inputs are assumed to equal the duration of the dataset. Any constant sampling rate may be used, but 40 Hz seems to be acceptable. This program’s cardiac peak detection algorithm is rather simplistic, so you might try using the scanner’s cardiac gating output (transform it to a spike wave if necessary).

This program uses slice timing information embedded in the dataset to estimate the proper cardiac/respiratory phase for each slice. It makes sense to run this program before any program that may destroy the slice timings (e.g. 3dvolreg for motion correction).

For complete details, see the 3dretroicor Documentation.

### Examples

```python
>>> from nipype.interfaces import afni
>>> ret = afni.Retroicor()
>>> ret.inputs.in_file = 'functional.nii'
>>> ret.inputs.card = 'mask.1D'
>>> ret.inputs.resp = 'resp.1D'
>>> ret.inputs.outputtype = 'NIFTI'
>>> ret.cmdline
'3dretroicor -prefix functional_retroicor.nii -resp resp.1D -card mask.1D -outputtype NIFTI -functional.nii'
>>> res = ret.run()  # doctest: +SKIP
```

### Inputs:

- **in_file**: (a pathlike object or string representing an existing file)
  - input file to 3dretroicor
  - argument: ``-in %s``, position: -1

- **out_file**: (a pathlike object or string representing a file)
  - output image file name
  - argument: ``-prefix %s``, position: 1

- **card**: (a pathlike object or string representing an existing file)
  - 1D cardiac data file for cardiac correction
  - argument: ``-card %s``, position: -2

- **resp**: (a pathlike object or string representing an existing file)
  - 1D respiratory waveform data for correction
  - argument: ``-resp %s``, position: -3

- **threshold**: (an integer (int or long))
  - Threshold for detection of R-wave peaks in input (Make sure it is above the background noise level, Try 3/4 or 4/5 times range plus minimum)
  - argument: ``-threshold %d``, position: -4

- **order**: (an integer (int or long))
  - The order of the correction (2 is typical)
  - argument: ``-order %s``, position: -5

- **cardphase**: (a pathlike object or string representing a file)
  - Filename for 1D cardiac phase output
  - argument: ``-cardphase %s``, position: -6

- **respphase**: (a pathlike object or string representing a file)
  - Filename for 1D resp phase output
  - argument: ``-respphase %s``, position: -7

- **num_threads**: (an integer (int or long), nipype default value: 1)
  - set number of threads

- **outputtype**: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')

(continues on next page)
AFNI output filetype

Output:

```
AFNI output filetype
```

```
args: (a unicode string)
	Additional parameters to the command
argument: ``\%s``
environ: (a dictionary with keys which are a bytes or None or a value
	of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
	Environment variables
```

Outputs:

```
out_file: (a pathlike object or string representing an existing file)
```

References:

None None

51.3.26 Seg

Link to code

Wraps the executable command 3dSeg. 3dSeg segments brain volumes into tissue classes. The program allows for adding a variety of global and voxelwise priors. However for the moment, only mixing fractions and MRF are documented. For complete details, see the 3dSeg Documentation.

Examples

```
>>> from nipype.interfaces.afni import preprocess
>>> seg = preprocess.Seg()
>>> seg.inputs.in_file = 'structural.nii'
>>> seg.inputs.mask = 'AUTO'
>>> seg.cmdline
'3dSeg -mask AUTO -anat structural.nii'
>>> res = seg.run()  # doctest: +SKIP
```

Inputs:

```
[Mandatory]
in_file: (a pathlike object or string representing an existing file)
	ANAT is the volume to segment
argument: ``\-anat \%s```, position: -1
mask: ("AUTO" or a pathlike object or string representing an existing file)
	only non-zero voxels in mask are analyzed. mask can either be a dataset or the string "AUTO" which would use AFNI's automask function to create the mask.
argument: ``\-mask \%s```, position: -2
```

[Optional]
blur_meth: ('BFT' or 'BIM')
	set the blurring method for bias field estimation
argument: ``\-blur_meth \%s``

bias_fwhm: (a float)
	The amount of blurring used when estimating the field bias with the Wells method

(continues on next page)
argument: `--bias_fwhm %f`
classes: (a unicode string)
  CLASS_STRING is a semicolon delimited string of class labels
  argument: `--classes %s`
bmrf: (a float)
  Weighting factor controlling spatial homogeneity of the
  classifications
  argument: `--bmrf %f`
bias_classes: (a unicode string)
  A semicolon delimited string of classes that contribute to the
  estimation of the bias field
  argument: `--bias_classes %s`
prefix: (a unicode string)
  the prefix for the output folder containing all output volumes
  argument: `--prefix %s`
mixfrac: (a unicode string)
  MIXFRAC sets up the volume-wide (within mask) tissue fractions while
  initializing the segmentation (see IGNORE for exception)
  argument: `--mixfrac %s`
mixfloor: (a float)
  Set the minimum value for any class's mixing fraction
  argument: `--mixfloor %f`
main_N: (an integer (int or long))
  Number of iterations to perform.
  argument: `--main_N %d`
args: (a unicode string)
  Additional parameters to the command
  argument: `%%s`
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a value
  of class 'str', nipype default value: {})
  Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
  output file

51.3.27 SkullStrip

Link to code
Wraps the executable command 3dSkullStrip.
A program to extract the brain from surrounding tissue from MRI T1-weighted images. TODO Add optional arguments.
For complete details, see the 3dSkullStrip Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> skullstrip = afni.SkullStrip()
>>> skullstrip.inputs.in_file = 'functional.nii'
>>> skullstrip.inputs.args = '-o_ply'
>>> skullstrip.cmdline
'3dSkullStrip -input functional.nii -o_ply -prefix functional_skullstrip'
>>> res = skullstrip.run()  # doctest: +SKIP
```

Inputs:
in_file: (a pathlike object or string representing an existing file)
  input file to 3dSkullStrip
  argument: `\"-input %s\``, position: 1

out_file: (a pathlike object or string representing a file)
  output image file name
  argument: `\"-prefix %s\``

num_threads: (an integer (int or long), nipype default value: 1)
  set number of threads

outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
  AFNI output filetype

args: (a unicode string)
  Additional parameters to the command
  argument: `\"%s\``

envir: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
  Environment variables

51.3.28 TCorr1D

Link to code
Wraps the executable command 3dTcorr1D.
Computes the correlation coefficient between each voxel time series in the input 3D+time dataset.
For complete details, see the 3dTcorr1D Documentation.

```python
>>> from nipype.interfaces import afni
>>> tcorr1D = afni.TCorr1D()
>>> tcorr1D.inputs.xset= 'u_rcrlsl_Template.nii'
>>> tcorr1D.inputs.y_1d = 'seed.1D'
>>> tcorr1D.cmdline
'3dTcorr1D -prefix u_rcrlsl_Template_correlation.nii.gz u_rcrlsl_Template.nii -
→seed.1D'
>>> res = tcorr1D.run()  # doctest: +SKIP
```
out_file: (a pathlike object or string representing a file)
    output filename prefix
    argument: ``-prefix %s``
pearson: (a boolean)
    Correlation is the normal Pearson correlation coefficient
    argument: `` -pearson``, position: 1
    mutually_exclusive: spearman, quadrant, ktaub
spearman: (a boolean)
    Correlation is the Spearman (rank) correlation coefficient
    argument: `` -spearman``, position: 1
    mutually_exclusive: pearson, quadrant, ktaub
quadrant: (a boolean)
    Correlation is the quadrant correlation coefficient
    argument: `` -quadrant``, position: 1
    mutually_exclusive: pearson, spearman, ktaub
ktaub: (a boolean)
    Correlation is the Kendall’s tau_b correlation coefficient
    argument: `` -ktaub``, position: 1
    mutually_exclusive: pearson, spearman, quadrant
num_threads: (an integer (int or long), nipype default value: 1)
    set number of threads
outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
    AFNI output filetype
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})
    Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
    output file containing correlations

References:

None

51.3.29 TCorrMap

Link to code

Wraps the executable command 3dTcorrMap.

For each voxel time series, computes the correlation between it and all other voxels, and combines this set of
values into the output dataset(s) in some way.

For complete details, see the 3dTcorrMap Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> tcm = afni.TCorrMap()
>>> tcm.inputs.in_file = 'functional.nii'
>>> tcm.inputs.mask = 'mask.nii'
>>> tcm.mean_file = 'functional_meancorr.nii'
```
>>> tcm.cmdline # doctest: +SKIP
'3dTcorrMap -input functional.nii -mask mask.nii -Mean functional_meancorr.nii'
>>> res = tcm.run() # doctest: +SKIP

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  argument: `'-input %s'`

[Optional]
seeds: (a pathlike object or string representing an existing file)
  argument: `'-seed %s'
  mutually_exclusive: s, e, d, s, _w, i, d, t, h
mask: (a pathlike object or string representing an existing file)
  argument: `'-mask %s'
automask: (a boolean)
  argument: `'-automask`
polort: (an integer (int or long))
  argument: `'-polort %d'
bandpass: (a tuple of the form: (a float, a float))
  argument: `'-bpass %f %f'
regress_out_timeseries: (a pathlike object or string representing an existing file)
  argument: `'-ort %s'
blur_fwhm: (a float)
  argument: `'-Gblur %f'
seeds_width: (a float)
  argument: `'-Mseed %f'
  mutually_exclusive: s, e, d, s
mean_file: (a pathlike object or string representing a file)
  argument: `'-Mean %s'
zmean: (a pathlike object or string representing a file)
  argument: `'-Zmean %s'
qmean: (a pathlike object or string representing a file)
  argument: `'-Qmean %s'
pmean: (a pathlike object or string representing a file)
  argument: `'-Pmean %s'
thresholds: (a list of items which are an integer (int or long))
absolute_threshold: (a pathlike object or string representing a file)
  argument: `'-Thresh %f %s'
  mutually_exclusive: absolute_threshold, var_absolute_threshold,
  var_absolute_threshold_normalize
var_absolute_threshold: (a pathlike object or string representing a file)
  argument: `'-VarThresh %f %f %f %s'
  mutually_exclusive: absolute_threshold, var_absolute_threshold,
  var_absolute_threshold_normalize
var_absolute_threshold_normalize: (a pathlike object or string representing a file)
  argument: `'-VarThreshN %f %f %f %f %f %s'
  mutually_exclusive: absolute_threshold, var_absolute_threshold,
  var_absolute_threshold_normalize
correlation_maps: (a pathlike object or string representing a file)
  argument: `'-CorrMap %s'
correlation_maps_masked: (a pathlike object or string representing a file)
argument: ``--CorrMask %s``
expr: (a unicode string)
average_expr: (a pathlike object or string representing a file)
    argument: ``--Aexpr %s %s``
    mutually_exclusive: average_expr, average_expr_nonzero, sum_expr
average_expr_nonzero: (a pathlike object or string representing a file)
    argument: ``--Cexpr %s %s``
    mutually_exclusive: average_expr, average_expr_nonzero, sum_expr
sum_expr: (a pathlike object or string representing a file)
    argument: ``--Sexpr %s %s``
    mutually_exclusive: average_expr, average_expr_nonzero, sum_expr
histogram_bin_numbers: (an integer (int or long))
histogram: (a pathlike object or string representing a file)
    argument: ``--Hist %d %s``
num_threads: (an integer (int or long), nipype default value: 1)
    set number of threads
outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
    AFNI output filetype
out_file: (a pathlike object or string representing a file)
    output image file name
    argument: ``--prefix %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
    Environment variables

Outputs:

mean_file: (a pathlike object or string representing a file)
zmean: (a pathlike object or string representing a file)
qmean: (a pathlike object or string representing a file)
pmean: (a pathlike object or string representing a file)
absolute_threshold: (a pathlike object or string representing a file)
var_absolute_threshold: (a pathlike object or string representing a file)
var_absolute_threshold_normalize: (a pathlike object or string representing a file)
correlation_maps: (a pathlike object or string representing a file)
correlation_maps_masked: (a pathlike object or string representing a file)
average_expr: (a pathlike object or string representing a file)
average_expr_nonzero: (a pathlike object or string representing a file)
sum_expr: (a pathlike object or string representing a file)
histogram: (a pathlike object or string representing a file)

References:

None None

51.3.30 TCorrelate

Link to code
Wraps the executable command 3dTcorrelate.
Computes the correlation coefficient between corresponding voxel time series in two input 3D+time datasets 'xset' and 'yset'
For complete details, see the 3dTcorrelate Documentation.

Examples

```python
>>> from nipype.interfaces import afni

>>> tcorrelate = afni.TCorrelate()

>>> tcorrelate.inputs.xset = 'u_rc1s1_Template.nii'

>>> tcorrelate.inputs.yset = 'u_rc1s2_Template.nii'

>>> tcorrelate.inputs.out_file = 'functional_tcorrelate.nii.gz'

>>> tcorrelate.inputs.polort = -1

>>> tcorrelate.inputs.pearson = True

>>> tcorrelate.cmdline

'3dTcorrelate -prefix functional_tcorrelate.nii.gz -pearson -polort -1 u_rc1s1_Template.nii u_rc1s2_Template.nii'

>>> res = tcorrelate.run()  # doctest: +SKIP
```

Inputs:

<table>
<thead>
<tr>
<th>[Mandatory]</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>xset:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>input xset</td>
<td></td>
<td></td>
</tr>
<tr>
<td>argument:</td>
<td><code>\</code>%s``, position: -2</td>
<td></td>
</tr>
<tr>
<td>yset:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>input yset</td>
<td></td>
<td></td>
</tr>
<tr>
<td>argument:</td>
<td><code>\</code>%s``, position: -1</td>
<td></td>
</tr>
</tbody>
</table>

| [Optional] |   |   |
| out_file:   |   |   |
| output file |   |   |
| argument:   | `\`-prefix %s\`\` |
| pearson:    | (a boolean) | Correlation is the normal Pearson correlation coefficient |
| argument:   | `\`-pearson\`\` |
| polort:     | (an integer (int or long)) | Remove polynomial trend of order m |
| argument:   | `\`-polort %d\`\` |
| num_threads: | (an integer (int or long), nipype default value: 1) | set number of threads |
| outputtype: | ('NIFTI' or 'AFNI' or 'NIFTI_GZ') | AFNI output filetype |
| args:       | (a unicode string) | Additional parameters to the command |
| argument:   | `\`%s\`\` |
| environ:    | (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) | Environment variables |

Outputs:

| out_file: | (a pathlike object or string representing an existing file) | output file |
References:
None None

51.3.31 TNorm

Link to code
Wraps the executable command 3dTnorm. Shifts voxel time series from input so that separate slices are aligned to the same temporal origin. For complete details, see the 3dTnorm Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> tnorm = afni.TNorm()
>>> tnorm.inputs.in_file = 'functional.nii'
>>> tnorm.inputs.norm2 = True
>>> tnorm.inputs.out_file = 'rm.errts.unit errts+tlrc'
>>> tnorm.cmdline
'3dTnorm -norm2 -prefix rm.errts.unit errts+tlrc functional.nii'
>>> res = tshift.run()  # doctest: +SKIP
```

Inputs:

[Required]
in_file: (a pathlike object or string representing an existing file)
  - input file to 3dTNorm
  - argument: ``'-%s'``, position: -1

[Optional]
out_file: (a pathlike object or string representing a file)
  - output image file name
  - argument: ``'-prefix %s'``
norm2: (a boolean)
  - L2 normalize (sum of squares = 1) [DEFAULT]
  - argument: ``'-norm2'``
normR: (a boolean)
  - normalize so sum of squares = number of time points * e.g., so RMS = 1.
  - argument: ``'-normR'``
norm1: (a boolean)
  - L1 normalize (sum of absolute values = 1)
  - argument: ``'-norm1'``
normx: (a boolean)
  - Scale so max absolute value = 1 (L_infinity norm)
  - argument: ``'-normx'``
polort: (an integer (int or long))
  - Detrend with polynomials of order p before normalizing
    - [DEFAULT = don't do this]
    - * Use `-polort 0` to remove the mean, for example
    - argument: ``'-polort %s'``
L1fit: (a boolean)
  - Detrend with L1 regression (L2 is the default)
  - * This option is here just for the hell of it
  - argument: ``'-L1fit'``
num_threads: (an integer (int or long), nipype default value: 1)
  - set number of threads
outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
   AFNI output filetype
args: (a unicode string)
   Additional parameters to the command
   argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
   of class 'str' and with values which are a bytes or None or a
   value of class 'str', nipype default value: {})
   Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
   output file

References:
None None

51.3.32 TProject

Link to code
Wraps the executable command 3dTproject.
This program projects (detrends) out various ‘nuisance’ time series from each voxel in the input dataset. Note
that all the projections are done via linear regression, including the frequency-based options such as ‘-passband’.
In this way, you can bandpass time-censored data, and at the same time, remove other time series of no interest
(e.g., physiological estimates, motion parameters). Shifts voxel time series from input so that separate slices are
aligned to the same temporal origin.
For complete details, see the 3dTproject Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> tproject = afni.TProject()
>>> tproject.inputs.in_file = 'functional.nii'
>>> tproject.inputs.bandpass = (0.00667, 99999)
>>> tproject.inputs.polort = 3
>>> tproject.inputs.automask = True
>>> tproject.inputs.out_file = 'projected.nii.gz'
>>> tproject.cmdline
'3dTproject -input functional.nii -automask -bandpass 0.00667 99999 -polort 3 -prefix projected.nii.gz'
>>> res = tproject.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
   input file to 3dTproject
   argument: ``'-input %s'``, position: 1

[Optional]
out_file: (a pathlike object or string representing a file)
   output image file name
   argument: ``'-prefix %s'``, position: -1
censor: (a pathlike object or string representing an existing file)

    filename of censor .1D time series

    * This is a file of 1s and 0s, indicating which
time points are to be included (1) and which are
to be excluded (0).
    argument: ``-censor %s``

censortr: (a list of items which are a unicode string)

    list of strings that specify time indexes
to be removed from the analysis. Each string is
of one of the following forms:
37 => remove global time index #37
2:37 => remove time index #37 in run #2
37..47 => remove global time indexes #37-47
37-47 => same as above
2:37..47 => remove time indexes #37-47 in run #2
*0-2 => remove time indexes #0-2 in all runs
+Time indexes within each run start at 0.
+Run indexes start at 1 (just be to confusing).
+N.B.: 2:37,47 means index #37 in run #2 and
global time index 47; it does NOT mean
index #37 in run #2 AND index #47 in run #2.
    argument: ``-CENSORTR %s``

cenmode: ('KILL' or 'ZERO' or 'NTRP')

    specifies how censored time points are treated in
the output dataset:
+ mode = ZERO ==> put zero values in their place
==> output dataset is same length as input
+ mode = KILL ==> remove those time points
==> output dataset is shorter than input
+ mode = NTRP ==> censored values are replaced by interpolated
neighboring (in time) non-censored values,
BEFORE any projections, and then the
analysis proceeds without actual removal
of any time points -- this feature is to
keep the Spanish Inquisition happy.
    * The default mode is KILL !!!
    argument: ``-cenmode %s``

concat: (a pathlike object or string representing an existing file)

    The catenation file, as in 3dDeconvolve, containing the
TR indexes of the start points for each contiguous run
within the input dataset (the first entry should be 0).
++ Also as in 3dDeconvolve, if the input dataset is
automatically catenated from a collection of datasets,
then the run start indexes are determined directly,
and '-concat' is not needed (and will be ignored).
++ Each run must have at least 9 time points AFTER
censoring, or the program will not work!
++ The only use made of this input is in setting up
the bandpass/stopband regressors.
++ '-ort' and '-dsort' regressors run through all time
points, as read in. If you want separate projections
in each run, then you must either break these ort files
into appropriate components, OR you must run 3dTproject
for each run separately, using the appropriate pieces
from the ort files via the '{...}' selector for the
1D files and the '[...]' selector for the datasets.
    argument: ``-concat %s``
noblock: (a boolean)
    Also as in 3dDeconvolve, if you want the program to treat
    an auto-catenated dataset as one long run, use this option.
    ++ However, '-noblock' will not affect catenation if you use
    the '-concat' option.
    argument: -noblock

ort: (a pathlike object or string representing an existing file)
    Remove each column in file
    ++ Each column will have its mean removed.
    argument: -ort %s

polort: (an integer (int or long))
    Remove polynomials up to and including degree pp.
    ++ Default value is 2.
    ++ It makes no sense to use a value of pp greater than
    2, if you are bandpassing out the lower frequencies!
    ++ For catenated datasets, each run gets a separate set
    set of pp+1 Legendre polynomial regressors.
    ++ Use of -polort -1 is not advised (if data mean != 0),
    even if -ort contains constant terms, as all means are
    removed.
    argument: -polort %d

dsort: (a list of items which are a pathlike object or string
    representing an existing file)
    Remove the 3D+time time series in dataset fset.
    ++ That is, 'fset' contains a different nuisance time
    series for each voxel (e.g., from AnatICOR).
    ++ Multiple -dsort options are allowed.
    argument: -dsort %s...

bandpass: (a tuple of the form: (a float, a float))
    Remove all frequencies EXCEPT those in the range
    argument: -bandpass %g %g

stopband: (a tuple of the form: (a float, a float))
    Remove all frequencies in the range
    argument: -stopband %g %g

TR: (a float)
    Use time step dd for the frequency calculations,
    rather than the value stored in the dataset header.
    argument: -TR %g

mask: (a pathlike object or string representing an existing file)
    Only operate on voxels nonzero in the mset dataset.
    ++ Voxels outside the mask will be filled with zeros.
    ++ If no masking option is given, then all voxels
    will be processed.
    argument: -mask %s

automask: (a boolean)
    Generate a mask automatically
    argument: -automask
    mutually_exclusive: mask

blur: (a float)
    Blur (inside the mask only) with a filter that has
    width (FWHM) of fff millimeters.
    ++ Spatial blurring (if done) is after the time
    series filtering.
    argument: -blur %g

norm: (a boolean)
    Normalize each output time series to have sum of
    squares = 1. This is the LAST operation.
argument: `--norm`
num_threads: (an integer (int or long), nipype default value: 1)
set number of threads
outputtype: (`NIFTI' or 'AFNI' or 'NIFTI_GZ')
   AFNI output filetype
args: (a unicode string)
   Additional parameters to the command
argument: `-%s`
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:
out_file: (a pathlike object or string representing an existing file)
   output file

References:
None None

51.3.33 TShift

Link to code
Wraps the executable command 3dTshift.
Shifts voxel time series from input so that separate slices are aligned to the same temporal origin.
For complete details, see the 3dTshift Documentation.

Examples
Slice timing details may be specified explicitly via the slice_timing input:

```python
>>> from nipype.interfaces import afni
>>> TR = 2.5
>>> tshift = afni.TShift()
>>> tshift.inputs.in_file = 'functional.nii'
>>> tshift.inputs.tzero = 0.0
>>> tshift.inputs.tr = '%.1f' % TR
>>> tshift.inputs.slice_timing = list(np.arange(40) / TR)
>>> tshift.cmdline
'3dTshift -prefix functional_tshift -tpattern @slice_timing.1D -TR 2.5s -tzero 0.0 functional.nii'
```

When the slice_timing input is used, the timing_file output is populated, in this case with the generated file.

```python
>>> tshift._list_outputs()['timing_file'] # doctest: +ELLIPSIS
'.../slice_timing.1D'
```

```python
>>> np.loadtxt(tshift._list_outputs()['timing_file']).tolist()[:5]
[0.0, 0.4, 0.8, 1.2, 1.6]
```

If slice_encoding_direction is set to `k-`, the slice timing is reversed:

```python
>>> tshift.inputs.slice_encoding_direction = 'k-
```

(continues on next page)
This method creates a `slice_timing.1D` file to be passed to 3dTshift. A pre-existing slice-timing file may be used in the same way:

```
>>> tshift = afni.TShift()
>>> tshift.inputs.in_file = 'functional.nii'
>>> tshift.inputs.tzero = 0.0
>>> tshift.inputs.tr = '%.1f' % TR
>>> tshift.inputs.slice_timing = '@slice_timing.1D'
>>> tshift.cmdline
'3dTshift -prefix functional_tshift -tpattern @slice_timing.1D -TR 2.5s -tzero 0.0
  functional.nii'
```

When a pre-existing file is provided, `timing_file` is simply passed through.

```
>>> tshift._list_outputs()['timing_file']  # doctest: +ELLIPSIS
'..\slice_timing.1D'
```

Alternatively, pre-specified slice timing patterns may be specified with the `tpattern` input. For example, to specify an alternating, ascending slice timing pattern:

```
>>> tshift = afni.TShift()
>>> tshift.inputs.in_file = 'functional.nii'
>>> tshift.inputs.tzero = 0.0
>>> tshift.inputs.tr = '%.1f' % TR
>>> tshift.inputs.tpattern = 'alt+z'
>>> tshift.cmdline
'3dTshift -prefix functional_tshift -tpattern alt+z -TR 2.5s -tzero 0.0
  functional.nii'
```

For backwards compatibility, `tpattern` may also take filenames prefixed with `@`. However, in this case, filenames are not validated, so this usage will be deprecated in future versions of Nipype.

```
>>> tshift = afni.TShift()
>>> tshift.inputs.in_file = 'functional.nii'
>>> tshift.inputs.tzero = 0.0
>>> tshift.inputs.tr = '%.1f' % TR
>>> tshift.inputs.tpattern = '@slice_timing.1D'
>>> tshift.cmdline
'3dTshift -prefix functional_tshift -tpattern @slice_timing.1D -TR 2.5s -tzero 0.
  functional.nii'
```

In these cases, `timing_file` is undefined.

```
>>> tshift._list_outputs()['timing_file']  # doctest: +ELLIPSIS
<undefined>
```

In any configuration, the interface may be run as usual:

```
>>> res = tshift.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]

- `in_file`: (a pathlike object or string representing an existing file)
  input file to 3dTshift

(continues on next page)
argument: `\`%s\``, position: -1

[Optional]
out_file: (a pathlike object or string representing a file)
  output image file name
  argument: `\`-prefix %s\``
tr: (a unicode string)
  manually set the TR. You can attach suffix "s" for seconds or "ms"
  for milliseconds.
  argument: `\`-TR %s\``
tzero: (a float)
  align each slice to given time offset
  argument: `\`-tzero %s\``
mutually_exclusive: tslice
tslice: (an integer (int or long))
  align each slice to time offset of given slice
  argument: `\`-slice %s\``
mutually_exclusive: tzero
ignore: (an integer (int or long))
  ignore the first set of points specified
  argument: `\`-ignore %s\``
interp: ('Fourier' or 'linear' or 'cubic' or 'quintic' or 'heptic')
  different interpolation methods (see 3dTshift for details) default = Fourier
  argument: `\`-%s\``

mutually_exclusive: tslice

[Optional] tslice: (an integer (int or long))
  align each slice to time offset of given slice
  argument: `\`-slice %s\``
mutually_exclusive: tzero
ignore: (an integer (int or long))
  ignore the first set of points specified
  argument: `\`-ignore %s\``
interp: ('Fourier' or 'linear' or 'cubic' or 'quintic' or 'heptic')
  different interpolation methods (see 3dTshift for details) default = Fourier
  argument: `\`-%s\``

tpattern: ('alt+z' or 'altplus' or 'alt+z2' or 'alt-z' or 'altminus' 
  or 'alt-z2' or 'seq+z' or 'seqplus' or 'seq-z' or 'seqminus' or a 
  unicode string)
  use specified slice time pattern rather than one in header
  argument: `\`-tpattern %s\``
mutually_exclusive: slice_timing

slice_timing: (a pathlike object or string representing an existing 
  file or a list of items which are a float)
  time offsets from the volume acquisition onset for each slice
  argument: `\`-tpattern @$s\``
mutually_exclusive: tpattern

slice_encoding_direction: ('k' or 'k-', nipype default value: k)
  Direction in which slice_timing is specified (default: k). If 
  negative, slice_timing is defined in reverse order, that is, the 
  first entry corresponds to the slice with the largest index, and the 
  final entry corresponds to slice index zero. Only in effect when 
  slice_timing is passed as list, not when it is passed as file.

rlt: (a boolean)
  Before shifting, remove the mean and linear trend
  argument: `\`-rlt\``

rltplus: (a boolean)
  Before shifting, remove the mean and linear trend and later put back 
  the mean
  argument: `\`-rlt+\``

num_threads: (an integer (int or long), nipype default value: 1)
  set number of threads

outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
  AFNI output filetype

args: (a unicode string)
  Additional parameters to the command
  argument: `\`%s\``
environ: (a dictionary with keys which are a bytes or None or a value 

of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

Environment variables

Outputs:

timing_file: (a pathlike object or string representing a file)
   AFNI formatted timing file, if ```slice_timing``` is a list
out_file: (a pathlike object or string representing an existing file)
   output file

References:

None None

51.3.34 Volreg

Link to code

Wraps the executable command 3dvolreg.
Register input volumes to a base volume using AFNI 3dvolreg command
For complete details, see the 3dvolreg Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> volreg = afni.Volreg()
>>> volreg.inputs.in_file = 'functional.nii'
>>> volreg.inputs.args = '-Fourier -twopass'
>>> volreg.inputs.zpad = 4
>>> volreg.inputs.outputtype = 'NIFTI'
>>> volreg.cmdline  # doctest: +ELLIPSIS
'3dvolreg -Fourier -twopass -1Dfile functional.1D -1Dmatrix_save functional.aff12.1D -prefix functional_volreg.nii -zpad 4 -maxdisp1D functional_md.1D functional.nii'
>>> res = volreg.run()  # doctest: +SKIP
```

```python
>>> from nipype.interfaces import afni
>>> volreg = afni.Volreg()
>>> volreg.inputs.in_file = 'functional.nii'
>>> volreg.inputs.interp = 'cubic'
>>> volreg.inputs.verbose = True
>>> volreg.inputs.zpad = 1
>>> volreg.inputs.basefile = 'functional.nii'
>>> volreg.inputs.out_file = 'rm.epi.volreg.r1'
>>> volreg.inputs.oned_file = 'dfile.r1.1D'
>>> volreg.inputs.oned_matrix_save = 'mat.r1.tshift+orig.1D'
>>> volreg.cmdline
'3dvolreg -cubic -1Dfile dfile.r1.1D -1Dmatrix_save mat.r1.tshift+orig.1D -prefix rm.epi.volreg.r1 -verbose -base functional.nii -zpad 1 -maxdisp1D functional_md.1D functional.nii'
>>> res = volreg.run()  # doctest: +SKIP
```

Inputs:
[Mandatory]

in_file: (a pathlike object or string representing an existing file)
  input file to 3dvolreg
  argument: ``-s``, position: -1

[Optional]

in_weight_volume: (a tuple of the form: (a pathlike object or string
  representing an existing file, an integer (int or long)) or a
  pathlike object or string representing an existing file)
  weights for each voxel specified by a file with an optional volume
  number (defaults to 0)
  argument: ``-weight %s``

out_file: (a pathlike object or string representing a file)
  output image file name
  argument: ``-prefix %s``

basefile: (a pathlike object or string representing an existing file)
  base file for registration
  argument: ``-base %s``, position: -6

zpad: (an integer (int or long))
  Zeropad around the edges by 'n' voxels during rotations
  argument: ``-zpad %d``, position: -5

md1d_file: (a pathlike object or string representing a file)
  max displacement output file
  argument: ``-maxdisp1D %s``, position: -4

oned_file: (a pathlike object or string representing a file)
  1D movement parameters output file
  argument: ``-1Dfile %s``

verbose: (a boolean)
  more detailed description of the process
  argument: ``-verbose``

timeshift: (a boolean)
  time shift to mean slice time offset
  argument: ``-tshift 0``

copyorigin: (a boolean)
  copy base file origin coords to output
  argument: ``-twodup``

oned_matrix_save: (a pathlike object or string representing a file)
  Save the matrix transformation
  argument: ``-1Dmatrix_save %s``

interp: ('Fourier' or 'cubic' or 'heptic' or 'quintic' or 'linear')
  spatial interpolation methods [default = heptic]
  argument: ``-%s``

num_threads: (an integer (int or long), nipype default value: 1)
  set number of threads

outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
  AFNI output filetype

args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a value
  of class 'str', nipype default value: {})
  Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
  registered file

(continues on next page)
mdld_file: (a pathlike object or string representing an existing file)
   max displacement info file
oned_file: (a pathlike object or string representing an existing file)
   movement parameters info file
oned_matrix_save: (a pathlike object or string representing an existing file)
   matrix transformation from base to input

References:
None None

51.3.35 Warp

Link to code
Wraps the executable command 3dWarp.
Use 3dWarp for spatially transforming a dataset
For complete details, see the 3dWarp Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> warp = afni.Warp()
>>> warp.inputs.in_file = 'structural.nii'
>>> warp.inputs.deoblique = True
>>> warp.inputs.out_file = 'trans.nii.gz'
>>> warp.cmdline
'3dWarp -deoblique -prefix trans.nii.gz structural.nii'
>>> res = warp.run()  # doctest: +SKIP

>>> warp_2 = afni.Warp()
>>> warp_2.inputs.in_file = 'structural.nii'
>>> warp_2.inputs.newgrid = 1.0
>>> warp_2.inputs.out_file = 'trans.nii.gz'
>>> warp_2.cmdline
'3dWarp -newgrid 1.000000 -prefix trans.nii.gz structural.nii'
>>> res = warp_2.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
   input file to 3dWarp
   argument: ``\`\%s\``', position: -1

[Optional]
out_file: (a pathlike object or string representing a file)
   output image file name
   argument: ``\`-prefix %s``
tta2mni: (a boolean)
   transform dataset from Talairach to MNI152
   argument: ``\`-tta2mni\``
mni2tta: (a boolean)

(continues on next page)
transform dataset from MNI152 to Talaraich
argument: ``-mni2tta``

matparent: (a pathlike object or string representing an existing file)
apply transformation from 3dWarpDrive
argument: ``-matparent %s``

oblique_parent: (a pathlike object or string representing an existing file)
Read in the oblique transformation matrix from an oblique dataset and make cardinal dataset oblique to match
argument: ``-oblique_parent %s``

deblique: (a boolean)
transform dataset from oblique to cardinal
argument: ``-deoblique``

interp: ('linear' or 'cubic' or 'NN' or 'quintic')
spatial interpolation methods [default = linear]
argument: ``-%s``

gridset: (a pathlike object or string representing an existing file)
copy grid of specified dataset
argument: ``-gridset %s``

newgrid: (a float)
specify grid of this size (mm)
argument: ``-newgrid %f``

zpad: (an integer (int or long))
pad input dataset with N planes of zero on all sides.
argument: ``-zpad %d``

verbose: (a boolean)
Print out some information along the way.
argument: ``-verb``

save_warp: (a boolean)
save warp as .mat file
requires: verbose

num_threads: (an integer (int or long), nipype default value: 1)
set number of threads

outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
AFNI output filetype

args: (a unicode string)
Additional parameters to the command
argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
Warped file.
warp_file: (a pathlike object or string representing a file)
warp transform .mat file

References:
None

None
51.4 interfaces.afni.svm

51.4.1 SVMTest

Link to code
Wraps the executable command 3dsvm.
Temporally predictive modeling with the support vector machine SVM Test Only For complete details, see the 3dsvm Documentation.

Examples

```python
>>> from nipype.interfaces import afni as afni
>>> svmTest = afni.SVMTest()
>>> svmTest.inputs.in_file= 'run2+orig'
>>> svmTest.inputs.model= 'run1+orig_model'
>>> svmTest.inputs.testlabels= 'run2_categories.1D'
>>> svmTest.inputs.out_file= 'pred2_model1'
>>> res = svmTest.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
model: (a unicode string)
modname is the basename for the brik containing the SVM model
argument: ``-model %s``
in_file: (a pathlike object or string representing an existing file)
A 3D or 3D+t AFNI brik dataset to be used for testing.
argument: ``-testvol %s``

[Optional]
out_file: (a pathlike object or string representing a file)
filename for .1D prediction file(s).
argument: ``-predictions %s``
testlabels: (a pathlike object or string representing an existing file)
*true* class category .1D labels for the test dataset. It is used to calculate the prediction accuracy performance
argument: ``-testlabels %s``
classout: (a boolean)
Flag to specify that pname files should be integer-valued, corresponding to class category decisions.
argument: ``-classout``
nopredcensord: (a boolean)
Flag to prevent writing predicted values for censored time-points
argument: ``-nopredcensord``
nodetrend: (a boolean)
Flag to specify that pname files should not be linearly detrended
argument: ``-nodetrend``
multiclass: (a boolean)
Specifies multiclass algorithm for classification
argument: ``-multiclass %s``
options: (a unicode string)
additional options for SVM-light
argument: ``%s``
um_threads: (an integer (int or long), nipype default value: 1)
set number of threads
outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
AFNI output filetype
args: (a unicode string)
    Additional parameters to the command
    argument: `\`%s\`'
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})
      Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
    output file

References:
None None

51.4.2 SVMTrain

Link to code
Wraps the executable command 3dsvm.
Temporally predictive modeling with the support vector machine SVM Train Only For complete details, see the 3dsvm Documentation.

Examples

```python
>>> from nipype.interfaces import afni as afni
>>> svmTrain = afni.SVMTrain()
>>> svmTrain.inputs.in_file = 'run1+orig'
>>> svmTrain.inputs.trainlabels = 'run1_categories.1D'
>>> svmTrain.inputs.ttype = 'regression'
>>> svmTrain.inputs.mask = 'mask.nii'
>>> svmTrain.inputs.model = 'model_run1'
>>> svmTrain.inputs.alphas = 'alphas_run1'
>>> res = svmTrain.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
ttype: (a unicode string)
    tname: classification or regression
    argument: `\`-type %s\`'
in_file: (a pathlike object or string representing an existing file)
    A 3D+t AFNI brik dataset to be used for training.
    argument: `\`-trainvol %s\`'

[Optional]
out_file: (a pathlike object or string representing a file)
    output sum of weighted linear support vectors file name
    argument: `\`-bucket %s\`
model: (a pathlike object or string representing a file)
    basename for the brik containing the SVM model
    argument: `\`-model %s\`
alphas: (a pathlike object or string representing a file)
    output alphas file name
argument: `\'-alpha %s\'`

mask: (a pathlike object or string representing an existing file)
byte-format brik file used to mask voxels in the analysis
argument: `\'-mask %s\'`, position: -1

nomodelmask: (a boolean)
Flag to enable the omission of a mask file
argument: `\'-nomodelmask\'`

trainlabels: (a pathlike object or string representing an existing file)
.1D labels corresponding to the stimulus paradigm for the training data.
argument: `\'-trainlabels %s\'`

censor: (a pathlike object or string representing an existing file)
.1D censor file that allows the user to ignore certain samples in the training data.
argument: `\'-censor %s\'`

kernel: (a unicode string)
string specifying type of kernel function:linear, polynomial, rbf, sigmoid
argument: `\'-kernel %s\'`

max_iterations: (an integer (int or long))
Specify the maximum number of iterations for the optimization.
argument: `\'-max_iterations %d\'`

w_out: (a boolean)
output sum of weighted linear support vectors
argument: `\'-wout\'`

options: (a unicode string)
additional options for SVM-light
argument: `\'%s\'`

num_threads: (an integer (int or long), nipype default value: 1)
set number of threads

output_type: (`NIFTI' or 'AFNI' or 'NIFTI_GZ')
AFNI output filetype

args: (a unicode string)
Additional parameters to the command
argument: `\'%s\'`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: { })
Environment variables

Outputs:

out_file: (a pathlike object or string representing a file)
sum of weighted linear support vectors file name

model: (a pathlike object or string representing a file)
brik containing the SVM model file name

alphas: (a pathlike object or string representing a file)
output alphas file name

References:

None None
51.5 interfaces.afni.utils

51.5.1 ABoverlap

Link to code
Wraps the executable command 3dABoverlap. Output (to screen) is a count of various things about how the automasks of datasets A and B overlap or don’t overlap. For complete details, see the 3dABoverlap Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> aoverlap = afni.ABoverlap()
>>> aoverlap.inputs.in_file_a = 'functional.nii'
>>> aoverlap.inputs.in_file_b = 'structural.nii'
>>> aoverlap.inputs.out_file = 'out.mask_ae_overlap.txt'
>>> aoverlap.cmdline
'3dABoverlap functional.nii structural.nii |& tee out.mask_ae_overlap.txt'
>>> res = aoverlap.run()  # doctest: +SKIP
```

Inputs:

<table>
<thead>
<tr>
<th>Mandatory</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_file_a: (a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>input file A</td>
</tr>
<tr>
<td>argument: <code>'-%s'</code>, position: <code>-3</code></td>
</tr>
<tr>
<td>in_file_b: (a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>input file B</td>
</tr>
<tr>
<td>argument: <code>'-%s'</code>, position: <code>-2</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>out_file: (a pathlike object or string representing a file)</td>
</tr>
<tr>
<td>collect output to a file</td>
</tr>
<tr>
<td>argument: `''</td>
</tr>
<tr>
<td>no_automask: (a boolean)</td>
</tr>
<tr>
<td>consider input datasets as masks</td>
</tr>
<tr>
<td>argument: <code>''-no_automask''</code></td>
</tr>
<tr>
<td>quiet: (a boolean)</td>
</tr>
<tr>
<td>be as quiet as possible (without being entirely mute)</td>
</tr>
<tr>
<td>argument: <code>''-quiet''</code></td>
</tr>
<tr>
<td>verb: (a boolean)</td>
</tr>
<tr>
<td>print out some progress reports (to stderr)</td>
</tr>
<tr>
<td>argument: <code>''-verb''</code></td>
</tr>
<tr>
<td>num_threads: (an integer (int or long), nipype default value: 1)</td>
</tr>
<tr>
<td>set number of threads</td>
</tr>
<tr>
<td>outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')</td>
</tr>
<tr>
<td>AFNI output filetype</td>
</tr>
<tr>
<td>args: (a unicode string)</td>
</tr>
<tr>
<td>Additional parameters to the command</td>
</tr>
<tr>
<td>argument: <code>''%s''</code></td>
</tr>
<tr>
<td>environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: <code>{}</code>)</td>
</tr>
<tr>
<td>Environment variables</td>
</tr>
</tbody>
</table>
Outputs:

out_file: (a pathlike object or string representing an existing file)
output file

References:
None None

51.5.2 AFNItoNIFTI

Link to code
Wraps the executable command 3dAFNItoNIFTI.
Converts AFNI format files to NIFTI format. This can also convert 2D or 1D data, which you can
numpy.squeeze() to remove extra dimensions.
For complete details, see the 3dAFNItoNIFTI Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> a2n = afni.AFNItoNIFTI()
>>> a2n.inputs.in_file = 'afni_output.3D'
>>> a2n.inputs.out_file = 'afni_output.nii'
>>> a2n.cmdline
'3dAFNItoNIFTI -prefix afni_output.nii afni_output.3D'
>>> res = a2n.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
input file to 3dAFNItoNIFTI
argument: ``%s``, position: -1

[Optional]
out_file: (a pathlike object or string representing a file)
output image file name
argument: ``-prefix %s``
pure: (a boolean)
Do NOT write an AFNI extension field into the output file. Only use
this option if needed. You can also use the 'nifti_tool' program to
strip extensions from a file.
argument: ``-pure``
denote: (a boolean)
When writing the AFNI extension field, remove text notes that might
contain subject identifying information.
argument: ``-denote``
oldid: (a boolean)
Give the new dataset the input datasets AFNI ID code.
argument: ``-oldid``
mutually_exclusive: newid
newid: (a boolean)
Give the new dataset a new AFNI ID code, to distinguish it from the
input dataset.
argument: ``-newid``
mutually_exclusive: oldid
num_threads: (an integer (int or long), nipype default value: 1)
set number of threads
outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
AFNI output filetype
args: (a unicode string)
   Additional parameters to the command
   argument: `\%s`
environ: (a dictionary with keys which are a bytes or None or a value
   of class 'str' and with values which are a bytes or None or a
   value of class 'str', nipype default value: {})
   Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
   output file

References:

None None

51.5.3 Autobox

Link to code
Wraps the executable command 3dAutobox.
Computes size of a box that fits around the volume. Also can be used to crop the volume to that box.
For complete details, see the 3dAutobox Documentation.

Examples

>>> from nipype.interfaces import afni
>>> abox = afni.Autobox()
>>> abox.inputs.in_file = 'structural.nii'
>>> abox.inputs.padding = 5
>>> abox.cmdline
'3dAutobox -input structural.nii -prefix structural_autobox -npad 5'
>>> res = abox.run()  # doctest: +SKIP

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
   input file
   argument: `\-input %s`'

[Optional]
padding: (an integer (int or long))
   Number of extra voxels to pad on each side of box
   argument: `\-npad %d`
out_file: (a pathlike object or string representing a file)
   argument: `\-prefix %s`
no_clustering: (a boolean)
   Don't do any clustering to find box. Any non-zero voxel will be
   preserved in the cropped volume. The default method uses some
   clustering to find the cropping box, and will clip off small
   isolated blobs.
   argument: `\-noclust`
References:

None None

51.5.4 Axialize

Link to code
Wraps the executable command 3daxialize.
Read in a dataset and write it out as a new dataset with the data brick oriented as axial slices.
For complete details, see the 3dcopy Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> axial3d = afni.Axialize()
>>> axial3d.inputs.in_file = 'functional.nii'
>>> axial3d.inputs.out_file = 'axialized.nii'
>>> axial3d.cmdline
'3daxialize -prefix axialized.nii functional.nii'
>>> res = axial3d.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  input file to 3daxialize
  argument: `` annexation '', position: -2

[Optional]
out_file: (a pathlike object or string representing a file)
  output image file name
  argument: annexation prefix annexation
verb: (a boolean)
Print out a progress report
argument: ``-verb``

sagittal: (a boolean)
  Do sagittal slice order [-orient ASL]
  argument: ``-sagittal``
  mutually_exclusive: coronal, axial

coronal: (a boolean)
  Do coronal slice order [-orient RSA]
  argument: ``-coronal``
  mutually_exclusive: sagittal, axial

axial: (a boolean)
  Do axial slice order [-orient RAI]
  This is the default AFNI axial order, and is the one currently required by the volume rendering plugin; this is also the default orientation output by this program (hence the program's name).
  argument: ``-axial``
  mutually_exclusive: coronal, sagittal

orientation: (a unicode string)
  new orientation code
  argument: ``-orient %s``

num_threads: (an integer (int or long), nipype default value: 1)
  set number of threads

outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
  AFNI output filetype

args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
  output file

References:

None None

51.5.5 BrickStat

Link to code

Wraps the executable command 3dBrickStat.

Computes maximum and/or minimum voxel values of an input dataset. TODO Add optional arguments.

For complete details, see the 3dBrickStat Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> brickstat = afni.BrickStat()
>>> brickstat.inputs.in_file = 'functional.nii'
>>> brickstat.inputs.mask = 'skeleton_mask.nii.gz'
>>> brickstat.inputs.min = True
```
>>> brickstat.cmdline
'3dBrickStat -min -mask skeleton_mask.nii.gz functional.nii'

>>> res = brickstat.run()  # doctest: +SKIP

# Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
    input file to 3dmaskave
    argument: "-s", position: -1

[Optional]
mask: (a pathlike object or string representing an existing file)
    -mask dset = use dset as mask to include/exclude voxels
    argument: "-mask %s", position: 2

min: (a boolean)
    print the minimum value in dataset
    argument: "-min"

slow: (a boolean)
    read the whole dataset to find the min and max values
    argument: "-slow"

max: (a boolean)
    print the maximum value in the dataset
    argument: "-max"

mean: (a boolean)
    print the mean value in the dataset
    argument: "-mean"

sum: (a boolean)
    print the sum of values in the dataset
    argument: "-sum"

var: (a boolean)
    print the variance in the dataset
    argument: "-var"

percentile: (a tuple of the form: (a float, a float, a float))
    p0 ps p1 write the percentile values starting at p0% and ending at
    p1% at a step of ps%. only one sub-brick is accepted.
    argument: "-percentile %.3f %.3f %.3f"

args: (a unicode string)
    Additional parameters to the command
    argument: "%s"

environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})
    Environment variables

# Outputs:

min_val: (a float)
    output

---

51.5.6 Bucket

Link to code
Wraps the executable command 3dbucket.
Concatenate sub-bricks from input datasets into one big ‘bucket’ dataset.
For complete details, see the 3dbucket Documentation.
Examples

```python
>>> from nipype.interfaces import afni
>>> bucket = afni.Bucket()
>>> bucket.inputs.in_file = [('functional.nii','\{2..$\}'), ('functional.nii','\{1\}')]
>>> bucket.inputs.out_file = 'vr_base'
>>> bucket.cmdline
"3dbucket -prefix vr_base functional.nii'\{2..$\}' functional.nii'\{1\}'"
>>> res = bucket.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a list of items which are a tuple of the form: (a pathlike
    object or string representing an existing file, a unicode string))
    List of tuples of input datasets and subbrick selection strings
    described in more detail in the following afni help string
    Input dataset specified using one of these forms: 'prefix+view',
    'prefix+view.HEAD', or 'prefix+view.BRIK'. You can also add a sub-
    brick selection list after the end of the dataset name. This allows
    only a subset of the sub-bricks to be included into the output (by
    default, all of the input datasets is copied into the output). A sub-
    brick selection list looks like one of the following forms:
    fred+orig[5] ==> use only sub-brick #5 fred+orig[5,9,17] ==> use #5,
    #9, and #17 fred+orig[5..8] or [5-8] ==> use #5, #6, #7, and #8
    fred+orig[5..13(2)] or [5-13(2)] ==> use #5, #7, #9, #11, and
    #13 Sub-brick indexes start at 0. You can use the character '$'
    to indicate the last sub-brick in a dataset; for example, you can
    select every third sub-brick by using the selection list
    fred+orig[0..$(3)]N.B.: The sub-bricks are output in the order
    specified, which may not be the order in the original datasets. For
    example, using fred+orig[0..$(2),1..$(2)] will cause the sub-bricks
    in fred+orig to be output into the new dataset in an interleaved
    fashion. Using fred+orig[$..0] will reverse the order of the sub-
    bricks in the output. N.B.: Bucket datasets have multiple sub-bricks,
    but do NOT have a time dimension. You can input sub-bricks from a
    3D+time dataset into a bucket dataset. You can use the '3dinfo
    program to see how many sub-bricks a 3D+time or a bucket dataset
    contains. N.B.: In non-bucket functional datasets (like the 'fico'
    datasets output by FIM, or the 'fitt' datasets output by 3dttest),
    sub-brick [0] is the 'intensity' and sub-brick [1] is the
    statistical parameter used as a threshold. Thus, to create a bucket
    dataset using the intensity from dataset A and the threshold from
dataset B, and calling the output dataset C, you would type 3dbucket
-pref C -fbuc 'A+orig[0]' -fbuc 'B+orig[1]' WARNING: using this
program, it is possible to create a dataset that has different basic
datum types for different sub-bricks (e.g., shorts for brick 0,
floats for brick 1). Do NOT do this! Very few AFNI programs will
work correctly with such datasets!
argument: `\`%s\``, position: -1

[Optional]
out_file: (a pathlike object or string representing a file)
argument: `\`-prefix %s\``
num_threads: (an integer (int or long), nipype default value: 1)
    set number of threads
outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
AFNI output filetype
args: (a unicode string)
   Additional parameters to the command
   argument: `\$s`'
environ: (a dictionary with keys which are a bytes or None or a value
   of class 'str' and with values which are a bytes or None or a
   value of class 'str', nipype default value: {})
   Environment variables

Outputs:
out_file: (a pathlike object or string representing an existing file)
   output file

References:
None None

51.5.7 Calc

Link to code
Wraps the executable command 3dcalc.
This program does voxel-by-voxel arithmetic on 3D datasets.
For complete details, see the 3dcalc Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> calc = afni.Calc()
>>> calc.inputs.in_file_a = 'functional.nii'
>>> calc.inputs.in_file_b = 'functional2.nii'
>>> calc.inputs.expr = 'a*b'
>>> calc.inputs.out_file = 'functional_calc.nii.gz'
>>> calc.inputs.outputtype = 'NIFTI'
>>> calc.cmdline
   # doctest: +ELLIPSIS
   '3dcalc -a functional.nii -b functional2.nii -expr "a*b" -prefix functional_calc.
    ...ni.gz'
>>> res = calc.run() # doctest: +SKIP
```

```python
>>> from nipype.interfaces import afni
>>> calc = afni.Calc()
>>> calc.inputs.in_file_a = 'functional.nii'
>>> calc.inputs.expr = '1'
>>> calc.inputs.out_file = 'rm.epi.alll'
>>> calc.inputs.overwrite = True
>>> calc.cmdline
   '3dcalc -a functional.nii -expr "1" -prefix rm.epi.alll -overwrite'
>>> res = calc.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file_a: (a pathlike object or string representing an existing file)
   input file to 3dcalc

(continues on next page)
argument: ``-a %s``, position: 0
expr: (a unicode string)
  expr
  argument: ``-expr "%s"``, position: 3

[Optional]
in_file_b: (a pathlike object or string representing an existing file)
  operand file to 3dcalc
  argument: ``-b %s``, position: 1
in_file_c: (a pathlike object or string representing an existing file)
  operand file to 3dcalc
  argument: ``-c %s``, position: 2
out_file: (a pathlike object or string representing a file)
  output image file name
  argument: ``-prefix %s``
start_idx: (an integer (int or long))
  start index for in_file_a
  requires: stop_idx
stop_idx: (an integer (int or long))
  stop index for in_file_a
  requires: start_idx
single_idx: (an integer (int or long))
  volume index for in_file_a
overwrite: (a boolean)
  overwrite output
  argument: ``-overwrite``
other: (a pathlike object or string representing a file)
  other options
num_threads: (an integer (int or long), nipype default value: 1)
  set number of threads
outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
  AFNI output filetype
args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
  output file

References:
None None

51.5.8 Cat

Link to code
Wraps the executable command 1dcat.
1dcat takes as input one or more 1D files, and writes out a 1D file containing the side-by-side concatenation of all or a subset of the columns from the input files.
Examples

```python
>>> from nipype.interfaces import afni
>>> cat1d = afni.Cat()
>>> cat1d.inputs.sel = "'[0,2]'"
>>> cat1d.inputs.in_files = ['f1.1D', 'f2.1D']
>>> cat1d.inputs.out_file = 'catout.1d'
>>> cat1d.cmdline
''1dcat -sel '[0,2]' f1.1D f2.1D > catout.1d''
>>> res = cat1d.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]

- **in_files**: (a list of items which are a pathlike object or string representing an existing file)
  - argument: ``"%s"``, position: -2
- **out_file**: (a pathlike object or string representing a file, nipype default value: catout.1d)
  - output (concatenated) file name
  - argument: ``"> %s"``, position: -1

[Optional]

- **omitconst**: (a boolean)
  - Omit columns that are identically constant from output.
  - argument: ``"-nonconst"``
- **keepfree**: (a boolean)
  - Keep only columns that are marked as 'free' in the 3dAllineate header from '-1Dparam_save'. If there is no such header, all columns are kept.
  - argument: ``"-nonfixed"``
- **out_format**: ('int' or 'nice' or 'double' or 'fint' or 'cint')
  - specify data type for output. Valid types are 'int', 'nice', 'double', 'fint', and 'cint'.
  - argument: ``"-form %s"``
- **stack**: (a boolean)
  - Stack the columns of the resultant matrix in the output.
  - argument: ``"-stack"``
- **sel**: (a unicode string)
  - Apply the same column/row selection string to all filenames on the command line.
  - argument: ``"-sel %s"``
- **out_int**: (a boolean)
  - specify int data type for output
  - argument: ``"-i"``
- **out_nice**: (a boolean)
  - specify nice data type for output
  - argument: ``"-n"``
- **out_double**: (a boolean)
  - specify double data type for output
  - argument: ``"-d"``
- **out_fint**: (a boolean)
  - specify float int data type for output
  - argument: ``"-f"``
- **out_cint**: (a boolean)
  - specify cint data type for output
  - argument: ``"-c"``
argument: ``-d``
mutually_exclusive: out_format, out_nice, out_int, out_fint,
out_cint

out_fint: (a boolean)
specify int, rounded down, data type for output
argument: ``-f``
mutually_exclusive: out_format, out_nice, out_double, out_int,
out_cint

out_cint: (a boolean)
specify int, rounded up, data type for output
mutually_exclusive: out_format, out_nice, out_double, out_fint,
out_int

num_threads: (an integer (int or long), nipype default value: 1)
set number of threads

outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
AFNI output filetype

args: (a unicode string)
Additional parameters to the command
argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
output file

References:

None None

51.5.9 CatMatvec

Link to code
Wraps the executable command cat_matvec.
Catenates 3D rotation+shift matrix+vector transformations.
For complete details, see the cat_matvec Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> cmv = afni.CatMatvec()
>>> cmv.inputs.in_file = [('structural.BRIK::WARP_DATA', 'I')]
>>> cmv.inputs.out_file = 'warp.anat.Xat.1D'
>>> cmv.cmdline
'cat_matvec structural.BRIK::WARP_DATA -I > warp.anat.Xat.1D'
>>> res = cmv.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]

in_file: (a list of items which are a tuple of the form: (a unicode
string, a unicode string))
list of tuples of mfiles and associated opkeys
argument: ``%s``\, position: -2

out_file: (a pathlike object or string representing a file)
File to write concatenated matvecs to
argument: `>`\, position: -1

[Optional]

matrix: (a boolean)
indicates that the resulting matrix will be written to outfile in the
'MATRIX(...)\' format (FORM 3). This feature could be used, with
clever scripting, to input a matrix directly on the command line to
program 3dWarp.
argument: `\-MATRIX`\`
mutually_exclusive: oneline, fourxfour

oneline: (a boolean)
indicates that the resulting matrix will simply be written as 12
numbers on one line.
argument: `\-ONELINE`\`
mutually_exclusive: matrix, fourxfour

fourxfour: (a boolean)
Output matrix in augmented form (last row is 0 0 0 1). This option
does not work with \-MATRIX or \-ONELINE
argument: `\-4x4`\`
mutually_exclusive: matrix, oneline

num_threads: (an integer (int or long), nipype default value: 1)
set number of threads

outputtype: ('NIFTI' or 'AFNI' or 'NIFTI\_GZ')
AFNI output filetype

args: (a unicode string)
Additional parameters to the command
argument: `'\%s'`

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a value
of class 'str', nipype default value: { })
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
output file

References:
None None

51.5.10 CenterMass

Link to code
Wraps the executable command 3dCM.
Computes center of mass using 3dCM command

Note: By default, the output is (x,y,z) values in DICOM coordinates. But as of Dec, 2016, there are now
command line switches for other options.

For complete details, see the 3dCM Documentation.
Examples

```python
>>> from nipype.interfaces import afni
>>> cm = afni.CenterMass()
>>> cm.inputs.in_file = 'structural.nii'
>>> cm.inputs.cm_file = 'cm.txt'
>>> cm.inputs.roi_vals = [2, 10]
>>> cm.cmdline
'3dCM -roi_vals 2 10 structural.nii > cm.txt'
>>> res = 3dcm.run()  # doctest: +SKIP
```

Inputs:

- **in_file**: (a pathlike object or string representing an existing file)
  - input file to 3dCM
  - argument: ```%s```, position: -2
- **cm_file**: (a pathlike object or string representing a file)
  - File to write center of mass to
  - argument: ```> %s```, position: -1
- **mask_file**: (a pathlike object or string representing an existing file)
  - Only voxels with nonzero values in the provided mask will be averaged.
  - argument: ```-mask %s```
- **automask**: (a boolean)
  - Generate the mask automatically
  - argument: ```-automask```
- **set_cm**: (a tuple of the form: (a float, a float, a float))
  - After computing the center of mass, set the origin fields in the header so that the center of mass will be at (x,y,z) in DICOM coords.
  - argument: ```-set %f %f %f```
- **local_ijk**: (a boolean)
  - Output values as (i,j,k) in local orientation
  - argument: ```-local_ijk```
- **roi_vals**: (a list of items which are an integer (int or long))
  - Compute center of mass for each blob with voxel value of v0, v1, v2, etc. This option is handy for getting ROI centers of mass.
  - argument: ```-roi_vals %s```
- **all_rois**: (a boolean)
  - Don't bother listing the values of ROIs you want: The program will find all of them and produce a full list.
  - argument: ```-all_rois```
- **args**: (a unicode string)
  - Additional parameters to the command
  - argument: ```%s```
- **environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) 
  - Environment variables

Outputs:

- **out_file**: (a pathlike object or string representing an existing file)
  - output file
- **cm_file**: (a pathlike object or string representing a file)
51.5.11 ConvertDset

Link to code
Wraps the executable command ConvertDset.
Converts a surface dataset from one format to another.
For complete details, see the ConvertDset Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> convertdset = afni.ConvertDset()
>>> convertdset.inputs.in_file = 'lh.pial_converted.gii'
>>> convertdset.inputs.out_type = 'niml_asc'
>>> convertdset.inputs.out_file = 'lh.pial_converted.niml.dset'
>>> convertdset.cmdline
'ConvertDset -o_niml_asc -input lh.pial_converted.gii -prefix lh.pial_converted.˓
→niml.dset'
>>> res = convertdset.run()  # doctest: +SKIP
```

Inputs:

- **in_file**: (a pathlike object or string representing an existing file)
  input file to ConvertDset
  argument: `'-input %s'`, position: -2
- **out_file**: (a pathlike object or string representing a file)
  output file for ConvertDset
  argument: `'-prefix %s'`, position: -1
- **out_type**: ('niml' or 'niml_asc' or 'niml_bi' or '1D' or '1Dp' or
  '1Dpt' or 'gii' or 'gii_asc' or 'gii_b64' or 'gii_b64gz')
  output type
  argument: `'-o_%s'`, position: 0

Optional:

- **num_threads**: (an integer (int or long), nipype default value: 1)
  set number of threads
- **outputtype**: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
  AFNI output filetype
- **args**: (a unicode string)
  Additional parameters to the command
  argument: `''`
- **environ**: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {}) 
  Environment variables

Outputs:

- **out_file**: (a pathlike object or string representing an existing file)
  output file
51.5.12 Copy

Wraps the executable command 3dcopy.
Copies an image of one type to an image of the same or different type using 3dcopy command
For complete details, see the 3dcopy Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> copy3d = afni.Copy()
>>> copy3d.inputs.in_file = 'functional.nii'
>>> copy3d.cmdline = '3dcopy functional.nii functional_copy'
>>> res = copy3d.run()  # doctest: +SKIP
```

```python
>>> from copy import deepcopy
>>> copy3d_2 = deepcopy(copy3d)
>>> copy3d_2.inputs.outputtype = 'NIFTI'
>>> copy3d_2.cmdline = '3dcopy functional.nii functional_copy.nii'
>>> res = copy3d_2.run()  # doctest: +SKIP
```

```python
>>> copy3d_3 = deepcopy(copy3d)
>>> copy3d_3.inputs.outputtype = 'NIFTI_GZ'
>>> copy3d_3.cmdline = '3dcopy functional.nii functional_copy.nii.gz'
>>> res = copy3d_3.run()  # doctest: +SKIP
```

```python
>>> copy3d_4 = deepcopy(copy3d)
>>> copy3d_4.inputs.out_file = 'new_func.nii'
>>> copy3d_4.cmdline = '3dcopy functional.nii new_func.nii'
>>> res = copy3d_4.run()  # doctest: +SKIP
```

Inputs:

- **in_file**: (a pathlike object or string representing an existing file)
  - input file to 3dcopy
  - argument: ``%s``, position: -2

- **out_file**: (a pathlike object or string representing a file)
  - output image file name
  - argument: ``%s``, position: -1

- **verbose**: (a boolean)
  - print progress reports
  - argument: ``-verb``

- **num_threads**: (an integer (int or long), nipype default value: 1)
  - set number of threads

- **outputtype**: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
  - AFNI output filetype

- **args**: (a unicode string)
  - Additional parameters to the command
  - argument: ``%s``

(continues on next page)
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
    output file

References:

None None

51.5.13 Dot

Link to code
Wraps the executable command 3dDot.
Correlation coefficient between sub-brick pairs. All datasets in in_files list will be concatenated. You can use sub-brick selectors in the file specification. Note: This program is not efficient when more than two subbricks are input. For complete details, see the 3ddot Documentation.

>>> from nipype.interfaces import afni
>>> dot = afni.Dot()
>>> dot.inputs.in_files = ['functional.nii[0]', 'structural.nii']
>>> dot.inputs.dodice = True
>>> dot.inputs.out_file = 'out.mask_ae_dice.txt'
>>> dot.cmdline
'3dDot -dodice functional.nii[0] structural.nii |& tee out.mask_ae_dice.txt'
>>> res = copy3d.run()  # doctest: +SKIP

Inputs:

[Optional]
in_files: (a list of items which are a pathlike object or string representing a file)
    list of input files, possibly with subbrick selectors
    argument: '%s ...', position: -2
out_file: (a pathlike object or string representing a file)
    collect output to a file
    argument: '%s |& tee %s', position: -1
mask: (a pathlike object or string representing a file)
    Use this dataset as a mask
    argument: '-mask %s'
mrange: (a tuple of the form: (a float, a float))
    Means to further restrict the voxels from 'mset' so that only those mask values within this range (inclusive) will be used.
    argument: '-mrange %s %s'
demean: (a boolean)
    Remove the mean from each volume prior to computing the correlation
    argument: '-demean'
docor: (a boolean)
    Return the correlation coefficient (default).
    argument: '-docor'
dodot: (a boolean)
    Return the dot product (unscaled).
    argument: '-dodot'
docoef: (a boolean)
  Return the least square fit coefficients \{\(a, b\)\} so that \(dset2\) is approximately \(a + b \cdot dset1\)
  argument: `--docoef`

dosums: (a boolean)
  Return the 6 numbers \(xbar = <x>\), \(ybar = <y>\), \((x-xbar)^2\), \((y-ybar)^2\), \((x-xbar)(y-ybar)\) and the correlation coefficient.
  argument: `--dosums`

dodice: (a boolean)
  Return the Dice coefficient (the Sorensen-Dice index).
  argument: `--dodice`

doeta2: (a boolean)
  Return \(\eta^2\) (Cohen, NeuroImage 2008).
  argument: `--doeta2`

full: (a boolean)
  Compute the whole matrix. A waste of time, but handy for parsing.
  argument: `--full`

show_labels: (a boolean)
  Print sub-brick labels to help identify what is being correlated.
  This option is useful when you have more than 2 sub-bricks at input.
  argument: `--show_labels`

upper: (a boolean)
  Compute upper triangular matrix
  argument: `--upper`

num_threads: (an integer (int or long), nipype default value: 1)
  set number of threads

outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
  AFNI output filetype

args: (a unicode string)
  Additional parameters to the command
  argument: `--%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
  output file

References:

None None

51.5.14 Edge3

Link to code
Wraps the executable command 3dedge3.
Does 3D Edge detection using the library 3DEdge by Gregoire Malandain (gregoire.malandain@sophia.inria.fr).
For complete details, see the 3dedge3 Documentation.

references = [{'entry': BibTeX('@article{Deriche1987,' 'author={R. Deriche},' 'title={Optimal edge detection using recursive filtering},' 'journal={International Journal of Computer Vision},' 'volume={2},' 'pages={167-187},' 'year={1987},'}'), 'tags': ['method'], },

51.5. interfaces.afni.utils 507
Examples

```python
>>> from nipype.interfaces import afni
>>> edge3 = afni.Edge3()
>>> edge3.inputs.in_file = 'functional.nii'
>>> edge3.inputs.out_file = 'edges.nii'
>>> edge3.inputs.datum = 'byte'
>>> edge3.cmdline
'3dedge3 -input functional.nii -datum byte -prefix edges.nii'
>>> res = edge3.run()  # doctest: +SKIP
```

Inputs:

[Optional]
out_file: (a pathlike object or string representing a file)
output image file name
argument: `-prefix %s`, position: -1

datum: ('byte' or 'short' or 'float')
specify data type for output. Valid types are 'byte', 'short' and 'float'.
argument: `-datum %s`

fscale: (a boolean)
Force scaling of the output to the maximum integer range.
argument: `-fscale`
mutually_exclusive: gscale, nscale, scale_floats

gscale: (a boolean)
Same as `-fscale`, but also forces each output sub-brick to get the same scaling factor.
argument: `-gscale`
mutually_exclusive: fscale, nscale, scale_floats

nscale: (a boolean)
Don't do any scaling on output to byte or short datasets.
argument: `-nscale`
mutually_exclusive: fscale, gscale, scale_floats

scale_floats: (a float)
Multiply input by VAL, but only if the input datum is float. This is needed when the input dataset has a small range, like 0 to 2.0 for instance. With such a range, very few edges are detected due to what I suspect to be truncation problems. Multiplying such a dataset by 10000 fixes the problem and the scaling is undone at the output.
argument: `-scale_floats %f`
mutually_exclusive: fscale, gscale, nscale

verbose: (a boolean)
Print out some information along the way.
argument: `-verbose`

(continues on next page)
num_threads: (an integer (int or long), nipype default value: 1)  
set number of threads

outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')  
AFNI output filetype

args: (a unicode string)  
Additional parameters to the command  
argument: '``%s``'

environ: (a dictionary with keys which are a bytes or None or a value  
of class 'str' and with values which are a bytes or None or a  
value of class 'str', nipype default value: { })
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)  
output file

References:

None None

51.5.15 Eval

Link to code
Wraps the executable command 1deval.  
Evaluates an expression that may include columns of data from one or more text files.  
For complete details, see the 1deval Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> eval = afni.Eval()
>>> eval.inputs.in_file_a = 'seed.1D'
>>> eval.inputs.in_file_b = 'resp.1D'
>>> eval.inputs.expr = 'a*b'
>>> eval.inputs.out1D = True
>>> eval.inputs.out_file = 'data_calc.1D'
>>> eval.cmdline
'1deval -a seed.1D -b resp.1D -expr "a*b" -1D -prefix data_calc.1D'
>>> res = eval.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file_a: (a pathlike object or string representing an existing file)  
input file to 1deval  
argument: '``-a %s``', position: 0

expr: (a unicode string)  
expr  
argument: '``-expr "%s"``', position: 3

[Optional]
in_file_b: (a pathlike object or string representing an existing file)  
operand file to 1deval
**51.5.16 FWHMx**

**Link to code**
Warms the executable command 3dFWHMx.

Unlike the older 3dFWHM, this program computes FWHMs for all sub-bricks in the input dataset, each one separately. The output for each one is written to the file specified by `-out`. The mean (arithmetic or geometric) of all the FWHMs along each axis is written to stdout. (A non-positive output value indicates something bad happened; e.g., FWHM in z is meaningless for a 2D dataset; the estimation method computed incoherent intermediate results.)

For complete details, see the 3dFWHMx Documentation.

---

### 3dFWHMx Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-b %s</code></td>
<td>operand file to 1deval</td>
<td>yes</td>
</tr>
<tr>
<td><code>-c %s</code></td>
<td>output image file name</td>
<td>yes</td>
</tr>
<tr>
<td><code>-prefix %s</code></td>
<td>output in 1D</td>
<td>yes</td>
</tr>
<tr>
<td><code>-1D</code></td>
<td>output in 1D</td>
<td>yes</td>
</tr>
<tr>
<td><code>-prefix %s</code></td>
<td>output in 1D</td>
<td>yes</td>
</tr>
<tr>
<td><code>-1D</code></td>
<td>output in 1D</td>
<td>yes</td>
</tr>
<tr>
<td><code>-start_idx</code></td>
<td>start index for input file</td>
<td>yes</td>
</tr>
<tr>
<td><code>-stop_idx</code></td>
<td>stop index for input file</td>
<td>yes</td>
</tr>
<tr>
<td><code>-single_idx</code></td>
<td>volume index for input file</td>
<td>yes</td>
</tr>
<tr>
<td><code>-other</code></td>
<td>other options</td>
<td>no</td>
</tr>
<tr>
<td><code>-num_threads</code></td>
<td>set number of threads</td>
<td>no</td>
</tr>
<tr>
<td><code>-outputtype</code></td>
<td>AFNI output filetype</td>
<td>no</td>
</tr>
<tr>
<td><code>-args</code></td>
<td>additional parameters to the command</td>
<td>no</td>
</tr>
<tr>
<td><code>-environ</code></td>
<td>environment variables</td>
<td>no</td>
</tr>
</tbody>
</table>

### Outputs:

- **out_file**: (a pathlike object or string representing an existing file)
  - output file

### References:

None None
Examples

```python
>>> from nipype.interfaces import afni
>>> fwhm = afni.FWHMx()
>>> fwhm.inputs.in_file = 'functional.nii'
>>> fwhm.cmdline
'3dFWHMx -input functional.nii -out functional_subbricks.out > functional_fwhmx.out'
>>> res = fwhm.run()  # doctest: +SKIP
```

(Classic) METHOD:
• Calculate ratio of variance of first differences to data variance.
• Should be the same as 3dFWHM for a 1-brick dataset. (But the output format is simpler to use in a script.)

**Note:** IMPORTANT NOTE [AFNI > 16]
A completely new method for estimating and using noise smoothness values is now available in 3dFWHMx and 3dClustSim. This method is implemented in the `-acf` options to both programs. ‘ACF’ stands for (spatial) AutoCorrelation Function, and it is estimated by calculating moments of differences out to a larger radius than before.

Notably, real FMRI data does not actually have a Gaussian-shaped ACF, so the estimated ACF is then fit (in 3dFWHMx) to a mixed model (Gaussian plus mono-exponential) of the form

\[
ACF(r) = a \cdot \exp(-r \cdot r / (2 \cdot b \cdot b)) + (1 - a) \cdot \exp(-r / c)
\]

where \(r\) is the radius, and \(a, b, c\) are the fitted parameters. The apparent FWHM from this model is usually somewhat larger in real data than the FWHM estimated from just the nearest-neighbor differences used in the ‘classic’ analysis.

The longer tails provided by the mono-exponential are also significant. 3dClustSim has also been modified to use the ACF model given above to generate noise random fields.

**Note:** TL;DR or summary
The take-away message is that the ‘classic’ 3dFWHMx and 3dClustSim analysis, using a pure Gaussian ACF, is not very correct for FMRI data – I cannot speak for PET or MEG data.

**Warning:** Do NOT use 3dFWHMx on the statistical results (e.g., ‘-bucket’) from 3dDeconvolve or 3dREMLfit!!! The function of 3dFWHMx is to estimate the smoothness of the time series NOISE, not of the statistics. This proscription is especially true if you plan to use 3dClustSim next!!

**Note:** Recommendations
• For FMRI statistical purposes, you DO NOT want the FWHM to reflect the spatial structure of the underlying anatomy. Rather, you want the FWHM to reflect the spatial structure of the noise. This means that the input dataset should not have anatomical (spatial) structure.
• One good form of input is the output of ‘3dDeconvolve -errts’, which is the dataset of residuals left over after the GLM fitted signal model is subtracted out from each voxel’s time series.
• If you don’t want to go to that much trouble, use ‘-detrend’ to approximately subtract out the anatomical spatial structure, OR use the output of 3dDetrend for the same purpose.
• If you do not use ‘-detrend’, the program attempts to find non-zero spatial structure in the input, and will print a warning message if it is detected.

**Note:** Notes on -demend
I recommend this option, and it is not the default only for historical compatibility reasons. It may become the default someday.

It is already the default in program 3dBlurToFWHM. This is the same detrending as done in 3dDespike; using \(2q+3\) basis functions for \(q > 0\).

If you don’t use ‘-detrend’, the program now [Aug 2010] checks if a large number of voxels are have significant nonzero means. If so, the program will print a warning message suggesting the use of ‘-detrend’, since inherent spatial structure in the image will bias the estimation of the FWHM of the image time series NOISE (which is usually the point of using 3dFWHMx).

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
   input dataset
   argument: `'-input %s'`

[Optional]
out_file: (a pathlike object or string representing a file)
   output file
   argument: `' > %s'`, position: -1
out_subbricks: (a pathlike object or string representing a file)
   output file listing the subbricks FWHM
   argument: `' -out %s'`
mask: (a pathlike object or string representing an existing file)
   use only voxels that are nonzero in mask
   argument: `' -mask %s'`
automask: (a boolean, nipype default value: False)
   compute a mask from THIS dataset, a la 3dAutomask
   argument: `' -automask'`
detrend: (a boolean or an integer (int or long), nipype default
   value: False)
   instead of demed (0th order detrending), detrend to the specified
   order. If order is not given, the program picks \(q=NT/30\). -detrend
   disables -demed, and includes -unif.
   argument: `' -detrend'`
   mutually_exclusive: demed
demed: (a boolean)
   If the input dataset has more than one sub-brick (e.g., has a time
   axis), then subtract the median of each voxel’s time series before
   processing FWHM. This will tend to remove intrinsic spatial
   structure and leave behind the noise.
   argument: `' -demed'`
   mutually_exclusive: detrend
unif: (a boolean)
   If the input dataset has more than one sub-brick, then normalize
   each voxel’s time series to have the same MAD before processing
   FWHM.
   argument: `' -unif'`
out_detrend: (a pathlike object or string representing a file)
   Save the detrended file into a dataset
   argument: `' -detprefix %s'`
geom: (a boolean)
   if in_file has more than one sub-brick, compute the final estimate
   as the geometric mean of the individual sub-brick FWHM estimates
   argument: `' -geom'`
   mutually_exclusive: arith
arith: (a boolean)
if in_file has more than one sub-brick, compute the final estimate as the arithmetic mean of the individual sub-brick FWHM estimates
argument: `--arith`
mutually_exclusive: geom
combine: (a boolean)
combine the final measurements along each axis
argument: `--combine`
compat: (a boolean)
be compatible with the older 3dFWHM
argument: `--compat`
acf: (a boolean or a pathlike object or string representing a file or
a tuple of the form: (a pathlike object or string representing an
eexisting file, a float), nipype default value: False)
computes the spatial autocorrelation
argument: `--acf`
args: (a unicode string)
Additional parameters to the command
argument: `-%s`
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
output file
out_subbricks: (a pathlike object or string representing an existing
file)
output file (subbricks)
out_detrend: (a pathlike object or string representing a file)
output file, detrended
fwhm: (a tuple of the form: (a float, a float, a float) or a tuple of
the form: (a float, a float, a float, a float))
FWHM along each axis
acf_param: (a tuple of the form: (a float, a float, a float) or a
tuple of the form: (a float, a float, a float, a float))
fitted ACF model parameters
out_acf: (a pathlike object or string representing an existing file)
output acf file

References:
None

51.5.17 GCOR

Link to code
Wraps the executable command @compute_gcor.
Computes the average correlation between every voxel and every other voxel, over any give mask.
For complete details, see the @compute_gcor Documentation.

Examples
>>> from nipype.interfaces import afni
>>> gcor = afni.GCOR()
>>> gcor.inputs.in_file = 'structural.nii'
>>> gcor.inputs.nfirst = 4
>>> gcor.cmdline
'@compute_gcor -nfirst 4 -input structural.nii'
>>> res = gcor.run()  # doctest: +SKIP

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  input dataset to compute the GCOR over
  argument: ``-input %s``, position: -1

[Optional]
mask: (a pathlike object or string representing an existing file)
  mask dataset, for restricting the computation
  argument: ``-mask %s``
nfirst: (an integer (int or long))
  specify number of initial TRs to ignore
  argument: ``-nfirst %d``
no_demean: (a boolean)
  do not (need to) demean as first step
  argument: ``-no_demean``
args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
  Environment variables

Outputs:

out: (a float)
  global correlation value

51.5.18 LocalBistat

Link to code

Wraps the executable command 3dLocalBistat. 3dLocalBistat - computes statistics between 2 datasets, at each voxel, based on a local neighborhood of that voxel. For complete details, see the 3dLocalBistat Documentation.

Examples

>>> from nipype.interfaces import afni
>>> bistat = afni.LocalBistat()
>>> bistat.inputs.in_file1 = 'functional.nii'
>>> bistat.inputs.in_file2 = 'structural.nii'
>>> bistat.inputs.neighborhood = ('SPHERE', 1.2)
>>> bistat.inputs.stat = 'pearson'
>>> bistat.inputs.outputtype = 'NIFTI'
>>> bistat.cmdline
(continues on next page)
"3dLocalBistat -prefix functional_bistat.nii -nbhd 'SPHERE(1.2)' -stat pearson functional.nii structural.nii"

>>> res = automask.run()  # doctest: +SKIP

Inputs:

[Mandatory]

in_file1: (a pathlike object or string representing an existing file)
  Filename of the first image
  argument: `'\%s'`, position: -2

in_file2: (a pathlike object or string representing an existing file)
  Filename of the second image
  argument: `'\%s'`, position: -1

neighborhood: (a tuple of the form: ('SPHERE' or 'RHDD' or 'TOHD', a float) or a tuple of the form: ('RECT', a tuple of the form: (a float, a float, a float)))
  The region around each voxel that will be extracted for the statistics calculation. Possible regions are: 'SPHERE', 'RHDD'
  (rhombic dodecahedron), 'TOHD' (truncated octahedron) with a given radius in mm or 'RECT' (rectangular block) with dimensions to
  specify in mm.
  argument: `'-nbhd \%s(\%)'`

stat: (a list of items which are 'pearson' or 'spearman' or 'quadrant' or 'mutinfo' or 'normuti' or 'jointent' or 'hellinger'
  or 'crU' or 'crM' or 'crA' or 'L2slope' or 'L1slope' or 'num' or 'ALL')
  statistics to compute. Possible names are: * pearson = Pearson correlation coefficient * spearman = Spearman correlation
  coefficient * quadrant = Quadrant correlation coefficient * mutinfo = Mutual Information * normuti = Normalized Mutual Information *
  jointent = Joint entropy * hellinger= Hellinger metric * crU = Correlation ratio (Unsymmetric) * crM = Correlation ratio
  (symmetrized by Multiplication) * crA = Correlation ratio (symmetrized by Addition) * crR = Correlation ratio
  (symmetrized by Addition) * L2slope = slope of least-squares (L2) linear regression of the data from dataset1 vs. the dataset2 (i.e.,
  d2 = a + b*d1 ==> this is 'b') * L1slope = slope of least-absolute-
  sum (L1) linear regression of the data from dataset1 vs. the
  dataset2 * num = number of the values in the region: with the use of
  -mask or -automask, the size of the region around any given voxel
  will vary; this option lets you map that size. * ALL = all of the
  above, in that orderMore than one option can be used.
  argument: `'-stat %s...'`

[Optional]

mask_file: (a pathlike object or string representing an existing file)
  mask image file name. Voxels NOT in the mask will not be used in the neighborhood of any voxel. Also, a voxel NOT in the mask will have
  its statistic(s) computed as zero (0).
  argument: `'-mask %s'`

automask: (a boolean)
  Compute the mask as in program 3dAutomask.
  argument: `'-automask'`

mutually_exclusive: weight_file

weight_file: (a pathlike object or string representing an existing file)
  File name of an image to use as a weight. Only applies to 'pearson'
Localstat

Link to code

Wraps the executable command 3dLocalstat.

3dLocalstat - computes statistics at each voxel, based on a local neighborhood of that voxel.

For complete details, see the 3dLocalstat Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> localstat = afni.Localstat()
>>> localstat.inputs.in_file = 'functional.nii'
>>> localstat.inputs.mask_file = 'skeleton_mask.nii.gz'
>>> localstat.inputs.neighborhood = ('SPHERE', 45)
>>> localstat.inputs.stat = 'mean'
>>> localstat.inputs.nonmask = True
>>> localstat.inputs.outputtype = 'NIFTI_GZ'
>>> localstat.cmdline
"3dLocalstat -prefix functional_localstat.nii -mask skeleton_mask.nii.gz -nbhd SPHERE(45.0) -use_nonmask -stat mean functional.nii"
>>> res = localstat.run() # doctest: +SKIP
```

Inputs:

<table>
<thead>
<tr>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_file: (a pathlike object or string representing an existing file)</td>
<td>input dataset</td>
</tr>
<tr>
<td>neighborhood: (a tuple of the form: ('SPHERE' or 'RHDD' or 'TOHD', a number))</td>
<td></td>
</tr>
</tbody>
</table>

References:

None

51.5.19 Localstat

Link to code

Wraps the executable command 3dLocalstat.

3dLocalstat - computes statistics at each voxel, based on a local neighborhood of that voxel.

For complete details, see the 3dLocalstat Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> localstat = afni.Localstat()
>>> localstat.inputs.in_file = 'functional.nii'
>>> localstat.inputs.mask_file = 'skeleton_mask.nii.gz'
>>> localstat.inputs.neighborhood = ('SPHERE', 45)
>>> localstat.inputs.stat = 'mean'
>>> localstat.inputs.nonmask = True
>>> localstat.inputs.outputtype = 'NIFTI_GZ'
>>> localstat.cmdline
"3dLocalstat -prefix functional_localstat.nii -mask skeleton_mask.nii.gz -nbhd SPHERE(45.0) -use_nonmask -stat mean functional.nii"
>>> res = localstat.run() # doctest: +SKIP
```

Inputs:

<table>
<thead>
<tr>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_file: (a pathlike object or string representing an existing file)</td>
<td>input dataset</td>
</tr>
<tr>
<td>neighborhood: (a tuple of the form: ('SPHERE' or 'RHDD' or 'TOHD', a number))</td>
<td></td>
</tr>
</tbody>
</table>
float) or a tuple of the form: ('RECT', a tuple of the form: (a float, a float, a float))

The region around each voxel that will be extracted for the statistics calculation. Possible regions are: 'SPHERE', 'RHDD' (rhombic dodecahedron), 'TOHD' (truncated octahedron) with a given radius in mm or 'RECT' (rectangular block) with dimensions to specify in mm.

argument: `^-nbhd '%%s(%s)'`

stat: (a list of items which are 'mean' or 'stdev' or 'var' or 'cvar'
or 'median' or 'MAD' or 'min' or 'max' or 'absmax' or 'num' or
'sum' or 'FWHM' or 'FWHMbar' or 'rank' or 'frank' or 'P2skew' or
'ALL' or 'mMP2s' or 'mmMP2s' or a tuple of the form: ('perc', a
tuple of the form: (a float, a float, a float)))

statistics to compute. Possible names are:
* mean = average of the values
* stdev = standard deviation
* var = variance (stdev*stdev)
* cvar = coefficient of variation = stdev/fabs(mean)
* median = median of the values
* MAD = median absolute deviation
* min = minimum
* max = maximum
* absmax = maximum of the absolute values
* num = number of the values in the region:

with the use of -mask or -automask, the size of the region around any given voxel will vary; this option lets you map that size. It may be useful if you plan to compute a t-statistic (say) from the mean and stdev outputs.
* sum = sum of the values in the region
* FWHM = compute (like 3dFWHM) image smoothness inside each voxel's neighborhood. Results are in 3 sub-bricks: FWHMx, FHWMy, and FWHMz. Places where an output is -1 are locations where the FWHM value could not be computed (e.g., outside the mask).
* FWHMbar = Compute just the average of the 3 FWHM values (normally would NOT do this with FWHM also).
* perc:P0:P1:Pstep = Compute percentiles between P0 and P1 with a step of Pstep. Default P1 is equal to P0 and default P2 = 1
* rank = rank of the voxel's intensity
* frank = rank / number of voxels in neighborhood
* P2skew = Pearson's second skewness coefficient 3 * (mean - median) / stdev
* ALL = all of the above, in that order (except for FWHMbar and perc).
* mMP2s = Exactly the same output as: median, MAD, P2skew, but a little faster
* mmMP2s = Exactly the same output as: mean, median, MAD, P2skew

More than one option can be used.

argument: `^-stat %s...'`

[Optional]

mask_file: (a pathlike object or string representing an existing file)

Mask image file name. Voxels NOT in the mask will not be used in the neighborhood of any voxel. Also, a voxel NOT in the mask will have its statistic(a) computed as zero (0) unless the parameter 'nonmask' is set to true.
argument: `'-mask %s'`
automask: (a boolean)
  Compute the mask as in program 3dAutomask.
  argument: `'-automask'`
nonmask: (a boolean)
  Voxels not in the mask WILL have their local statistics computed
  from all voxels in their neighborhood that ARE in the mask.
  * For instance, this option can be used to compute the average
    local white matter time series, even at non-WM voxels.
  argument: `'-use_nonmask'`
reduce_grid: (a float or a tuple of the form: (a float, a float, a
  float))
  Compute output on a grid that is reduced by the specified factors.
  If a single value is passed, output is resampled to the specified
  isotropic grid. Otherwise, the 3 inputs describe the reduction in
  the X, Y, and Z directions. This option speeds up computations at
  the expense of resolution. It should only be used when the nbhd is
  quite large with respect to the input's resolution, and the
  resultant stats are expected to be smooth.
  argument: `'-reduce_grid %s'`
mutually_exclusive: reduce_restore_grid, reduce_max_vox
reduce_restore_grid: (a float or a tuple of the form: (a float, a
  float))
  Like reduce_grid, but also resample output back to inputgrid.
  argument: `'-reduce_restore_grid %s'`
mutually_exclusive: reduce_max_vox, reduce_grid
reduce_max_vox: (a float)
  Like reduce_restore_grid, but automatically set Rx Ry Rz sothat the
  computation grid is at a resolution of nbhd/MAX_VOXvoxels.
  argument: `'-reduce_max_vox %s'`
mutually_exclusive: reduce_restore_grid, reduce_grid
grid_rmode: ("NN" or 'Li' or 'Cu' or 'Bk')
  Interpolant to use when resampling the output with
  thereduce_restore_grid option. The resampling method string RESAM
  should come from the set {'NN', 'Li', 'Cu', 'Bk'}). These stand for
  'Nearest Neighbor', 'Linear', 'Cubic', and 'Blocky' interpolation,
  respectively.
  argument: `'-grid_rmode %s'`
  requires: reduce_restore_grid
quiet: (a boolean)
  Stop the highly informative progress reports.
  argument: `'-quiet'`
overwrite: (a boolean)
  overwrite output file if it already exists
  argument: `'-overwrite'`
out_file: (a pathlike object or string representing a file)
  Output dataset.
  argument: `'-prefix %s'`, position: 0
num_threads: (an integer (int or long), nipype default value: 1)
  set number of threads
outputtype: ("NIFTI" or 'AFNI' or 'NIFTI_GZ')
  AFNI output filetype
args: (a unicode string)
  Additional parameters to the command
  argument: `'%s'`
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {}}

Environment variables

Outputs:

| out_file: (a pathlike object or string representing an existing file) |
| output file |

References:

None None

51.5.20 MaskTool

Link to code

Wraps the executable command 3dmask_tool.

3dmask_tool - for combining/dilating/eroding/filling masks

For complete details, see the 3dmask_tool Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> masktool = afni.MaskTool()
>>> masktool.inputs.in_file = 'functional.nii'
>>> masktool.inputs.outputtype = 'NIFTI'
>>> masktool.cmdline
'3dmask_tool -prefix functional_mask.nii -input functional.nii'
>>> res = automask.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]

| in_file: (a list of items which are a pathlike object or string representing an existing file) |
| input file or files to 3dmask_tool |
| argument: `'-input %s'`, position: -1 |

[Optional]

| out_file: (a pathlike object or string representing a file) |
| output image file name |
| argument: `'-prefix %s'` |

| count: (a boolean) |
| Instead of created a binary 0/1 mask dataset, create one with counts of voxel overlap, i.e., each voxel will contain the number of masks that it is set in. |
| argument: `'-count'`, position: 2 |

| datum: ('byte' or 'short' or 'float') |
| specify data type for output. Valid types are 'byte', 'short' and 'float'. |
| argument: `'-datum %s'` |

| dilate_inputs: (a unicode string) |
| Use this option to dilate and/or erode datasets as they are read. ex. '5 -5' to dilate and erode 5 times |
| argument: `'-dilate_inputs %s'` |

| dilate_results: (a unicode string) |
| dilate and/or erode combined mask at the given levels. |

(continues on next page)
### 51.5.21 Merge

Wraps the executable command 3dmerge.

#### Link to code

Merge or edit volumes using AFNI 3dmerge command

For complete details, see the 3dmerge Documentation.
Examples

```python
>>> from nipype.interfaces import afni
>>> merge = afni.Merge()
>>> merge.inputs.in_files = ['functional.nii', 'functional2.nii']
>>> merge.inputs.blitrfwhm = 4
>>> merge.inputs.doall = True
>>> merge.inputs.out_file = 'e7.nii'
>>> merge.cmdline
'3dmerge -1blur_fwhm 4 -doall -prefix e7.nii functional.nii functional2.nii'
>>> res = merge.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]

- **in_files**: (a list of items which are a pathlike object or string representing an existing file)
  - argument: `%-s`, position: -1

[Optional]

- **out_file**: (a pathlike object or string representing a file)
  - output image file name
  - argument: `%-prefix %s`
- **doall**: (a boolean)
  - apply options to all sub-bricks in dataset
  - argument: `%-doall`
- **blurfwhm**: (an integer (int or long))
  - FWHM blur value (mm)
  - argument: `%-blur_fwhm %d`
- **num_threads**: (an integer (int or long), nipype default value: 1)
  - set number of threads
- **outputtype**: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
  - AFNI output filetype
- **args**: (a unicode string)
  - Additional parameters to the command
  - argument: `%-s`
- **environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})  
  - Environment variables

Outputs:

- **out_file**: (a pathlike object or string representing an existing file)
  - output file

References:

None None

51.5.22 Notes

Link to code

Wraps the executable command 3dNotes.

A program to add, delete, and show notes for AFNI datasets.

For complete details, see the 3dNotes Documentation.
Examples

```python
>>> from nipype.interfaces import afni
>>> notes = afni.Notes()
>>> notes.inputs.in_file = 'functional.HEAD'
>>> notes.inputs.add = 'This note is added.'
>>> notes.inputs.add_history = 'This note is added to history.'
>>> notes.cmdline
'3dNotes -a "This note is added." -h "This note is added to history." functional.HEAD'
>>> res = notes.run()  # doctest: +SKIP
```

Inputs:

- **In_file**: (a pathlike object or string representing an existing file)
  - input file to 3dNotes
  - argument: '``%s``', position: -1

- **Optional**
  - **add**: (a unicode string)
    - note to add
    - argument: '``-a "%s"``'
  - **add_history**: (a unicode string)
    - note to add to history
    - argument: '``-h "%s"``'
    - mutually_exclusive: rep_history
  - **rep_history**: (a unicode string)
    - note with which to replace history
    - argument: '``-HH "%s"``'
    - mutually_exclusive: add_history
  - **delete**: (an integer (int or long))
    - delete note number num
    - argument: '``-d %d``'
  - **ses**: (a boolean)
    - print to stdout the expanded notes
    - argument: '``-ses``'
  - **out_file**: (a pathlike object or string representing a file)
    - output image file name
    - argument: '``%s``'
  - **num_threads**: (an integer (int or long), nipype default value: 1)
    - set number of threads
  - **outputtype**: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
    - AFNI output filetype
  - **args**: (a unicode string)
    - Additional parameters to the command
    - argument: '``%s``'
  - **environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
    - Environment variables

Outputs:

- **out_file**: (a pathlike object or string representing an existing file)
  - output file
51.5.23 NwarpAdjust

Link to code
Wraps the executable command 3dNwarpAdjust.
This program takes as input a bunch of 3D warps, averages them, and computes the inverse of this average warp. It then composes each input warp with this inverse average to 'adjust' the set of warps. Optionally, it can also read in a set of 1-brick datasets corresponding to the input warps, and warp each of them, and average those. For complete details, see the 3dNwarpAdjust Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> adjust = afni.NwarpAdjust()
>>> adjust.inputs.warps = ['func2anat_InverseWarp.nii.gz', 'func2anat_InverseWarp.nii.gz', 'func2anat_InverseWarp.nii.gz', 'func2anat_InverseWarp.nii.gz', 'func2anat_InverseWarp.nii.gz']
>>> adjust.cmdline
'3dNwarpAdjust -nwarp func2anat_InverseWarp.nii.gz func2anat_InverseWarp.nii.gz func2anat_InverseWarp.nii.gz func2anat_InverseWarp.nii.gz func2anat_InverseWarp.nii.gz'
>>> res = adjust.run()  # doctest: +SKIP
```

Inputs:

- **warps**: (a list of at least 5 items which are a pathlike object or string representing an existing file)
  - List of input 3D warp datasets
  - argument: ``-nwarp %s``

- **in_files**: (a list of at least 5 items which are a pathlike object or string representing an existing file)
  - List of input 3D datasets to be warped by the adjusted warp datasets. There must be exactly as many of these datasets as there are input warps.
  - argument: ``-source %s``

- **out_file**: (a pathlike object or string representing a file)
  - Output mean dataset, only needed if in_files are also given. The output dataset will be on the common grid shared by the source datasets.
  - argument: ``-prefix %s``
  - requires: in_files

- **num_threads**: (an integer (int or long), nipype default value: 1)
  - set number of threads

- **outputtype**: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
  - AFNI output filetype

- **args**: (a unicode string)
  - Additional parameters to the command
  - argument: ``%s``

- **environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  - Environment variables

Outputs:
nipype Documentation, Release 1.2.1

out_file: (a pathlike object or string representing an existing file)
output file

51.5.24 NwarpApply
Link to code
Wraps the executable command 3dNwarpApply.
Program to apply a nonlinear 3D warp saved from 3dQwarp (or 3dNwarpCat, etc.) to a 3D dataset, to produce
a warped version of the source dataset.
For complete details, see the 3dNwarpApply Documentation.
Examples
>>> from nipype.interfaces import afni
>>> nwarp = afni.NwarpApply()
>>> nwarp.inputs.in_file = 'Fred+orig'
>>> nwarp.inputs.master = 'NWARP'
>>> nwarp.inputs.warp = "'Fred_WARP+tlrc Fred.Xaff12.1D'"
>>> nwarp.cmdline
"3dNwarpApply -source Fred+orig -interp wsinc5 -master NWARP -prefix Fred+orig_
˓→Nwarp -nwarp 'Fred_WARP+tlrc Fred.Xaff12.1D'"
>>> res = nwarp.run() # doctest: +SKIP

Inputs:
[Mandatory]
in_file: (a pathlike object or string representing an existing file
or a list of items which are a pathlike object or string
representing an existing file)
the name of the dataset to be warped can be multiple datasets
argument: ``-source %s``
warp: (a string)
the name of the warp dataset. multiple warps can be concatenated
(make sure they exist)
argument: ``-nwarp %s``
[Optional]
inv_warp: (a boolean)
After the warp specified in '-nwarp' is computed, invert it
argument: ``-iwarp``
master: (a pathlike object or string representing an existing file)
the name of the master dataset, which defines the output grid
argument: ``-master %s``
interp: ('wsinc5' or 'NN' or 'nearestneighbour' or 'nearestneighbor'
or 'linear' or 'trilinear' or 'cubic' or 'tricubic' or 'quintic'
or 'triquintic', nipype default value: wsinc5)
defines interpolation method to use during warp
argument: ``-interp %s``
ainterp: ('NN' or 'nearestneighbour' or 'nearestneighbor' or 'linear'
or 'trilinear' or 'cubic' or 'tricubic' or 'quintic' or
'triquintic' or 'wsinc5')
specify a different interpolation method than might be used for the
warp
argument: ``-ainterp %s``
out_file: (a pathlike object or string representing a file)
output image file name
(continues on next page)

524

Chapter 51. interfaces.afni


argument: `'-prefix %s'`
short: (a boolean)
Write output dataset using 16-bit short integers, rather than the usual 32-bit floats.
argument: `'-short'`
quiet: (a boolean)
don't be verbose :
argument: `'-quiet'`
mutually_exclusive: verb
verb: (a boolean)
be extra verbose :)
argument: `'-verb'`
mutually_exclusive: quiet
args: (a unicode string)
Additional parameters to the command
argument: `'%s'`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
output file

51.5.25 NwarpCat

Link to code
Wraps the executable command 3dNwarpCat.
Catenates (composes) 3D warps defined on a grid, OR via a matrix.

Note:
- All transformations are from DICOM xyz (in mm) to DICOM xyz.
- Matrix warps are in files that end in '.1D' or in '.txt'. A matrix warp file should have 12 numbers in it, as output (for example), by `3dAllineate -1Dmatrix_save`.
- Nonlinear warps are in dataset files (AFNI .HEAD/.BRIK or NIfTI .nii) with 3 sub-bricks giving the DICOM order xyz grid displacements in mm.
- If all the input warps are matrices, then the output is a matrix and will be written to the file `prefix.aff12.1D`. Unless the prefix already contains the string `.1D`, in which case the filename is just the prefix.
- If `prefix` is just `stdout`, then the output matrix is written to standard output. In any of these cases, the output format is 12 numbers in one row.
- If any of the input warps are datasets, they must all be defined on the same 3D grid! And of course, then the output will be a dataset on the same grid. However, you can expand the grid using the `'-expad'` option.
- The order of operations in the final (output) warp is, for the case of 3 input warps:
  \[
  \text{OUTPUT}(x) = \text{warp3}( \text{warp2}( \text{warp1}(x) ) )
  \]
That is, warp1 is applied first, then warp2, et cetera. The 3D x coordinates are taken from each grid location in the first dataset defined on a grid.

For complete details, see the 3dNwarpCat Documentation.
Examples

```python
>>> from nipype.interfaces import afni
>>> nwarpcat = afni.NwarpCat()
>>> nwarpcat.inputs.in_files = ['Q25_warp+tlrc.HEAD', ('IDENT', 'structural.nii')]
>>> nwarpcat.inputs.out_file = 'Fred_total_WARP'
>>> nwarpcat.cmdline
"3dNwarpCat -interp wsinc5 -prefix Fred_total_WARP Q25_warp+tlrc.HEAD
→'IDENT(structural.nii)'
```

```
>>> res = nwarpcat.run()  # doctest: +SKIP
```

Inputs:

- **in_files**: (a list of items which are a pathlike object or string representing a file or a tuple of the form: ('IDENT' or 'INV' or 'SQRT' or 'SQRTINV', a pathlike object or string representing a file))
  
  list of tuples of 3D warps and associated functions

  argument: `"%s"`, position: -1

- **space**: (a string)
  
  string to attach to the output dataset as its atlas space marker.

  argument: `"-space %s"`

- **inv_warp**: (a boolean)
  
  invert the final warp before output

  argument: `"-iwarp"`

- **interp**: ('wsinc5' or 'linear' or 'quintic', nipype default value: wsinc5)
  
  specify a different interpolation method than might be used for the warp

  argument: `"-interp %s"`

- **expad**: (an integer (int or long))
  
  Pad the nonlinear warps by the given number of voxels voxels in all directions. The warp displacements are extended by linear extrapolation from the faces of the input grid.

  argument: `"-expad %d"`

- **out_file**: (a pathlike object or string representing a file)
  
  output image file name

  argument: `"-prefix %s"`

- **verb**: (a boolean)
  
  be verbose

  argument: `"-verb"`

- **num_threads**: (an integer (int or long), nipype default value: 1)
  
  set number of threads

- **outputtype**: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
  
  AFNI output filetype

- **args**: (a unicode string)
  
  Additional parameters to the command

  argument: `"%s"`

- **environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  
  Environment variables

Outputs:
out_file: (a pathlike object or string representing an existing file)
  output file

References:
None None

51.5.26 OneDTolPy

Link to code
Wraps the executable command 1d_tool.py.
This program is meant to read/manipulate/write/diagnose 1D datasets. Input can be specified using AFNI sub-
brick[/time] selectors.

```python
>>> from nipype.interfaces import afni
>>> odt = afni.OneDTolPy()
>>> odt.inputs.in_file = 'f1.1D'
>>> odt.inputs.set_nruns = 3
>>> odt.inputs.demean = True
>>> odt.inputs.out_file = 'motion_dmean.1D'
>>> odt.cmdline # doctest: +ELLIPSIS
'python2 ...1d_tool.py -demean -infile f1.1D -write motion_dmean.1D -set_nruns 3'
>>> res = odt.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  input file to OneDTol
  argument: '--in_file %s'

[Optional]
set_nruns: (an integer (int or long))
  treat the input data as if it has nruns
  argument: '--set_nruns %d'
derivative: (a boolean)
  take the temporal derivative of each vector (done as first backward
difference)
  argument: '--derivative'
demean: (a boolean)
  demean each run (new mean of each run = 0.0)
  argument: '--demean'
out_file: (a pathlike object or string representing a file)
  write the current 1D data to FILE
  argument: '--write %s'
mutually_exclusive: show_cormat_warnings

show_censor_count: (a boolean)
  display the total number of censored TRs Note: if input is a valid
xmat.1D dataset, then the count will come from the header. Otherwise
the input is assumed to be a binary censorfile, and zeros are simply
counted.
  argument: '--show_censor_count'
censor_motion: (a tuple of the form: (a float, a pathlike object or
  string representing a file))
  Tuple of motion limit and outfile prefix. need to also set set_nruns
  -r set_run_lengths
  argument: '--censor_motion %f %s'

(continues on next page)
censor_prev_TR: (a boolean)
  for each censored TR, also censor previous
  argument: ``--censor_prev_TR``
show_trs_uncensored: ("comma" or "space" or "encoded" or "verbose")
  display a list of TRs which were not censored in the specified style
  argument: ``--show_trs_uncensored %s``
show_cormat_warnings: (a pathlike object or string representing a
  file)
  Write cormat warnings to a file
  argument: ``--show_cormat_warnings |& tee %s``, position: -1
  mutually_exclusive: out_file
show_indices_interest: (a boolean)
  display column indices for regs of interest
  argument: ``--show_indices_interest``
show_trs_run: (an integer (int or long))
  restrict -show_trs_[un]censored to the given 1-based run
  argument: ``--show_trs_run %d``
outputtype: ("NIFTI" or "AFNI" or "NIFTI_GZ")
  AFNI output filetype
py27_path: (a pathlike object or string representing an existing file
  or 'python2', nipype default value: python2)
args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
in_file: (a pathlike object or string representing an existing file)
  input dataset
  argument: `--inset %s`, position: 1

[Optional]
out_file: (a pathlike object or string representing a file)
  Output dataset.
  argument: `--prefix %s`, position: 0

chi_sq: (a boolean)
  Output the Friedman chi-squared value in addition to the Kendall's
  W. This option is currently compatible only with the AFNI
  (BRIK/HEAD) output type; the chi-squared value will be the second
  sub-brick of the output dataset.
  argument: `--chi_sq`

mask_file: (a pathlike object or string representing a file)
  Mask within which ReHo should be calculated voxelwise
  argument: `--mask %s`

neighborhood: ('faces' or 'edges' or 'vertices')
  voxels in neighborhood. can be: * faces (for voxel and 6 facewise
  neighbors, only),
  * edges (for voxel and 18 face- and edge-wise neighbors),
  * vertices (for voxel and 26 face-, edge-, and node-wise neighbors).
  argument: `--nneigh %s`

sphere: (a float)
  for additional voxelwise neighborhood control, the radius R of a
  desired neighborhood can be put in; R is a floating point number,
  and must be >1. Examples of the numbers of voxels in a given radius
  are as follows (you can roughly approximate with the ol'
  4*PI*(R^3)/3 thing):
  R=2.0 -> V=33,
  R=2.3 -> V=57,
  R=2.9 -> V=93,
  R=3.1 -> V=123,
  R=3.9 -> V=251,
  R=4.5 -> V=389,
  R=6.1 -> V=949,
  but you can choose most any value.
  argument: `--neigh_RAD %s`

ellipsoid: (a tuple of the form: (a float, a float, a float))
  Tuple indicating the x, y, and z radius of an ellipsoid defining the
  neighbourhood of each voxel.
  The 'hood is then made according to the following relation:(i/A)^2 +
  (j/B)^2 + (k/C)^2 <=1.
  which will have approx. V=4*PI*A*B*C/3. The impetus for this freedom
  was for use with data having anisotropic voxel edge lengths.
  argument: `--neigh_X %s --neigh_Y %s --neigh_Z %s`

label_set: (a pathlike object or string representing an existing
  file)
  a set of ROIs, each labelled with distinct integers. ReHo will then
  be calculated per ROI.
  argument: `--in_rois %s`

overwrite: (a boolean)
  overwrite output file if it already exists

(continues on next page)
argument: `-overwrite`
args: (a unicode string)
   Additional parameters to the command
   argument: `-%s`
environ: (a dictionary with keys which are a bytes or None or a value
   of class 'str' and with values which are a bytes or None or a
   value of class 'str', nipype default value: {})
   Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
   Voxelwise regional homogeneity map
out_vals: (a pathlike object or string representing a file)
   Table of labelwise regional homogeneity values

51.5.28 Refit

Link to code
Wraps the executable command 3drefit.
Changes some of the information inside a 3D dataset's header
For complete details, see the 3drefit Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> refit = afni.Refit()
>>> refit.inputs.in_file = 'structural.nii'
>>> refit.inputs.deoblique = True
>>> refit.cmdline
'3drefit -deoblique structural.nii'
>>> res = refit.run()  # doctest: +SKIP
```

```python
>>> refit_2 = afni.Refit()
>>> refit_2.inputs.in_file = 'structural.nii'
>>> refit_2.inputs.atrfloat = ('IJK_TO_DICOM_REAL', '1 0.2 0 0 -0.2 1 0 0 0 0 1 0')
>>> refit_2.cmdline
"3drefit -atrfloat IJK_TO_DICOM_REAL '1 0.2 0 0 -0.2 1 0 0 0 0 1 0' structural.nii"
>>> res = refit_2.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
   input file to 3drefit
   argument: `-%s`, position: -1

[Optional]
deoblique: (a boolean)
   replace current transformation matrix with cardinal matrix
   argument: `-%deoblique`
xorigin: (a unicode string)
   x distance for edge voxel offset

(continues on next page)
argument: `-xorigin %s`
yorigin: (a unicode string)
y distance for edge voxel offset
argument: `-yorigin %s`
zorigin: (a unicode string)
z distance for edge voxel offset
argument: `-zorigin %s`
duporigin_file: (a pathlike object or string representing an existing file)
Copies the xorigin, yorigin, and zorigin values from the header of the given dataset
argument: `--duporigin %s`
xdel: (a float)
new x voxel dimension in mm
argument: `--xdel %f`
ydel: (a float)
new y voxel dimension in mm
argument: `--ydel %f`
zdel: (a float)
new z voxel dimension in mm
argument: `--zdel %f`
xyzscale: (a float)
Scale the size of the dataset voxels by the given factor
argument: `--xyzscale %f`

space: ('TLRC' or 'MNI' or 'ORIG')
Associates the dataset with a specific template type, e.g. TLRC, MNI, ORIG
argument: `--space %s`
atrcopy: (a tuple of the form: (a pathlike object or string representing an existing file, a unicode string))
Copy AFNI header attribute from the given file into the header of the dataset(s) being modified. For more information on AFNI header attributes, see documentation file README.attributes. More than one `-atrcopy` option can be used. For AFNI advanced users only. Do NOT use `-atrcopy` or `-atrstring` with other modification options. See also `-copyaux`.
argument: `--atrcopy %s %s`
atrstring: (a tuple of the form: (a unicode string, a unicode string))
Copy the last given string into the dataset(s) being modified, giving it the attribute name given by the last string. To be safe, the last string should be in quotes.
argument: `--atrstring %s %s`
atrfloat: (a tuple of the form: (a unicode string, a unicode string))
Create or modify floating point attributes. The input values may be specified as a single string in quotes or as a 1D filename or string, example '1 0.2 0 0 -0.2 1 0 0 0 0 1 0' or flipZ.1D or '1D:1,0.2,200,-0.2,1,200,200,1,0'
argument: `--atrfloat %s %s`
atrint: (a tuple of the form: (a unicode string, a unicode string))
Create or modify integer attributes. The input values may be specified as a single string in quotes or as a 1D filename or string, example '1 0 0 0 0 1 0 0 0 0 1 0' or flipZ.1D or '1D:1,0,200,-0,1,200,200,1,0'
argument: `--atrint %s %s`
saveatr: (a boolean)
(default) Copy the attributes that are known to AFNI into the
dset->dblk structure thereby forcing changes to known attributes to be present in the output. This option only makes sense with -atrcopy.

nosaveatr: (a boolean)

Opposite of -saveatr

args: (a unicode string)

Additional parameters to the command

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}))

Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)

output file

51.5.29 Resample

Link to code

Wraps the executable command 3dresample. Resample or reorient an image using AFNI 3dresample command

For complete details, see the 3dresample Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> resample = afni.Resample()
>>> resample.inputs.in_file = 'functional.nii'
>>> resample.inputs.orientation = 'RPI'
>>> resample.inputs.outputtype = 'NIFTI'
>>> resample.cmdline
'3dresample -orient RPI -prefix functional_resample.nii -inset functional.nii'
>>> res = resample.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]

in_file: (a pathlike object or string representing an existing file)

input file to 3dresample

argument: `-inset %s`, position: -1

[Optional]

out_file: (a pathlike object or string representing a file)

output image file name

argument: `-prefix %s`

orientation: (a unicode string)

new orientation code

argument: `-orient %s`

resample_mode: ('NN' or 'Li' or 'Cu' or 'Bk')

resampling method from set {'NN', 'Li', 'Cu', 'Bk'}. These are for "Nearest Neighbor", "Linear", "Cubic" and "Blocky" interpolation, respectively. Default is NN.
argument: ``-rmode %s``

voxel size: (a tuple of the form: (a float, a float, a float))
resample to new dx, dy and dz
argument: ``-dxyz %f %f %f``

master: (a pathlike object or string representing a file)
aligned dataset grid to a reference file
argument: ``-master %s``

num_threads: (an integer (int or long), nipype default value: 1)
set number of threads

outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
AFNI output filetype

args: (a unicode string)
Additional parameters to the command
argument: '%s'

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {}))
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
output file

References:

None None

51.5.30 TCat

Link to code
Wraps the executable command 3dTcat.
Concatenate sub-bricks from input datasets into one big 3D+time dataset.
TODO Replace InputMultiPath in_files with Traits.List, if possible. Current version adds extra whitespace.
For complete details, see the 3dTcat Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> tcat = afni.TCat()
>>> tcat.inputs.in_files = ['functional.nii', 'functional2.nii']
>>> tcat.inputs.out_file = 'functional_tcat.nii'
>>> tcat.inputs.rlt = '+'
>>> tcat.cmdline
'3dTcat -rlt+ -prefix functional_tcat.nii functional.nii functional2.nii'
>>> res = tcat.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_files: (a list of items which are a pathlike object or string
representing an existing file)
input file to 3dTcat
argument: `` %s`` , position: -1

[Optional]
out_file: (a pathlike object or string representing a file)
    output image file name
    argument: `'-prefix %s``
rlt: ('' or '+' or '++')
    Remove linear trends in each voxel time series loaded from each
    input dataset, SEPARATELY. Option -rlt removes the least squares fit
    of 'a+b*t' to each voxel time series. Option -rlt+ adds dataset mean
    back in. Option -rlt++ adds overall mean of all dataset timeseries
    back in.
    argument: `'-rlt%s``, position: 1
verbose: (a boolean)
    Print out some verbose output as the program
    argument: `'-verb``
um_threads: (an integer (int or long), nipype default value: 1)
    set number of threads
outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
    AFNI output filetype
args: (a unicode string)
    Additional parameters to the command
    argument: `'`$
    environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {}) Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
    output file

References:

None None

51.5.31 TCatSubBrick

Link to code

Wraps the executable command 3dTcat.
Hopefully a temporary function to allow sub-brick selection until afni file managment is improved.
For complete details, see the 3dTcat Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> tcsb = afni.TCatSubBrick()
>>> tcsb.inputs.in_files = [('functional.nii', '{2..$}')], ('functional2.nii', '{2..$}')]
>>> tcsb.inputs.out_file= 'functional_tcat.nii'
>>> tcsb.inputs.rlt = '+'
>>> tcsb.cmdline
"3dTcat -rlt+ -prefix functional_tcat.nii functional.nii{2..$}' functional2.nii' →{2..$}' "
>>> res = tcsb.run()  # doctest: +SKIP
```

Inputs:
### [Mandatory]

**in_files**: (a list of items which are a tuple of the form: (a pathlike object or string representing an existing file, a unicode string))

List of tuples of file names and subbrick selectors as strings. Don't forget to protect the single quotes in the subbrick selector so the contents are protected from the command line interpreter.

argument: ``%s%s ...``, position: -1

### [Optional]

**out_file**: (a pathlike object or string representing a file)

output image file name

argument: ``-prefix %s``

**rlt**: ('' or '+' or '++')

Remove linear trends in each voxel time series loaded from each input dataset, SEPARATELY. Option -rlt removes the least squares fit of 'a+b*t' to each voxel time series. Option -rlt+ adds dataset mean back in. Option -rlt++ adds overall mean of all dataset timeseries back in.

argument: ``-rlt%s``, position: 1

**num_threads**: (an integer (int or long), nipype default value: 1)

set number of threads

**outputtype**: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')

AFNI output filetype

**args**: (a unicode string)

Additional parameters to the command

argument: ``%s``

**environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

Environment variables

### Outputs:

<table>
<thead>
<tr>
<th><strong>out_file</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>output file</td>
</tr>
</tbody>
</table>

### References:

None

### 51.5.32 TStat

**Link to code**

Wraps the executable command 3dTstat.

**Compute voxel-wise statistics using AFNI 3dTstat command**

For complete details, see the 3dTstat Documentation.

### Examples

```python
>>> from nipype.interfaces import afni
>>> tstat = afni.TStat()
>>> tstat.inputs.in_file = 'functional.nii'
>>> tstat.inputs.args = '-mean'
>>> tstat.inputs.out_file = 'stats'
>>> tstat.cmdline
'3dTstat -mean -prefix stats functional.nii'
>>> res = tstat.run()  # doctest: +SKIP
```
Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  input file to 3dTstat
  argument: ``-%s``, position: -1

[Optional]
out_file: (a pathlike object or string representing a file)
  output image file name
  argument: ``-prefix %s``
mask: (a pathlike object or string representing an existing file)
  mask file
  argument: ``-mask %s``
options: (a unicode string)
  selected statistical output
  argument: ``%s``
num_threads: (an integer (int or long), nipype default value: 1)
  set number of threads
outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
  AFNI output filetype
args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
  Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
  output file

References:

None None

51.5.33 To3D

Link to code
Wraps the executable command to3d.
Create a 3D dataset from 2D image files using AFNI to3d command
For complete details, see the to3d Documentation

Examples

```python
>>> from nipype.interfaces import afni
>>> to3d = afni.To3D()
>>> to3d.inputs.datatype = 'float'
>>> to3d.inputs.in_folder = '.
>>> to3d.inputs.out_file = 'dicomdir.nii'
>>> to3d.inputs.filetype = 'anat'
>>> to3d.cmdline  # doctest: +ELLIPSIS
'to3d -datum float -anat -prefix dicomdir.nii ./*.dcm'
>>> res = to3d.run()  # doctest: +SKIP
```
in_folder: (a pathlike object or string representing an existing directory)
folder with DICOM images to convert
argument: ``-%s/*.dcm``, position: -1

[Optional]
out_file: (a pathlike object or string representing a file)
output image file name
argument: ``-prefix %s``

filetype: ('spgr' or 'fse' or 'epan' or 'anat' or 'ct' or 'spct' or 'pet' or 'mra' or 'bmap' or 'diff' or 'omri' or 'abuc' or 'fim' or 'fith' or 'fico' or 'fitt' or 'fift' or 'fict' or 'fibt' or 'fibrn' or 'figt' or 'fit' or 'fbuc')
type of datafile being converted
argument: ``-%s``

skipoutliers: (a boolean)
skip the outliers check
argument: ``-skip_outliers``

assumemosaic: (a boolean)
assume that Siemens image is mosaic
argument: ``-assume_dicom_mosaic``

datatype: ('short' or 'float' or 'byte' or 'complex')
set output file datatype
argument: ``-datum %s``

funcparams: (a unicode string)
parameters for functional data
argument: ``-time:zt %s alt+z2``

num_threads: (an integer (int or long), nipype default value: 1)
set number of threads

outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
AFNI output filetype

args: (a unicode string)
Additional parameters to the command
argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
output file

References:
None None

51.5.34 Undump

Link to code
Wraps the executable command 3dUndump.
3dUndump - Assembles a 3D dataset from an ASCII list of coordinates and (optionally) values.
The input file(s) are ASCII files, with one voxel specification per line. A voxel specification is 3 numbers (-ijk or -xyz coordinates), with an optional 4th number giving the voxel value. For example: 1 2 3 2 1 5 5.3 6.2 3.7 // this line illustrates a comment
The first line puts a voxel (with value given by `'-dval'`) at point (1,2,3). The second line puts a voxel (with value 5) at point (3,2,1). The third line puts a voxel (with value given by `'-dval'`) at point (5.3,6.2,3.7). If `-ijk` is in effect, and fractional coordinates are given, they will be rounded to the nearest integers; for example, the third line would be equivalent to (i,j,k) = (5,6,4).

For complete details, see the **3dUndump Documentation**.

### Examples

```python
>>> from nipype.interfaces import afni
>>> unndump = afni.Undump()
>>> unndump.inputs.in_file = 'structural.nii'
>>> unndump.inputs.out_file = 'structural_undumped.nii'
>>> unndump.cmdline
'3dUndump -prefix structural_undumped.nii -master structural.nii'
>>> res = unndump.run() # doctest: +SKIP
```

### Inputs:

<table>
<thead>
<tr>
<th>[Mandatory]</th>
<th>[Optional]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in_file</strong>: (a pathlike object or string representing an existing file)</td>
<td><strong>out_file</strong>: (a pathlike object or string representing a file)</td>
</tr>
<tr>
<td>input file to 3dUndump, whose geometry will determinethe geometry of the output</td>
<td>output image file name</td>
</tr>
<tr>
<td>argument: <code>'-master %s'</code>, position: -1</td>
<td>argument: <code>'-prefix %s'</code></td>
</tr>
<tr>
<td><strong>mask_file</strong>: (a pathlike object or string representing a file)</td>
<td><strong>default_value</strong>: (a float)</td>
</tr>
<tr>
<td>mask image file name. Only voxels that are nonzero in the mask can be set.</td>
<td>default value stored in each input voxel that does not have a value supplied in the input file</td>
</tr>
<tr>
<td>argument: <code>'-mask %s'</code></td>
<td>argument: <code>'-datum %s'</code></td>
</tr>
<tr>
<td><strong>datatype</strong>: ('short' or 'float' or 'byte')</td>
<td><strong>fill_value</strong>: (a float)</td>
</tr>
<tr>
<td>set output file datatype</td>
<td>value, used for each voxel in the output dataset that is NOT listed in the input file</td>
</tr>
<tr>
<td>argument: <code>'-datum %s'</code></td>
<td>argument: <code>'-fval %f'</code></td>
</tr>
<tr>
<td><strong>coordinates_specification</strong>: ('ijk' or 'xyz')</td>
<td><strong>coordinates_specification</strong>: ('ijk' or 'xyz')</td>
</tr>
<tr>
<td>Coordinates in the input file as index triples (i, j, k) or spatial coordinates (x, y, z) in mm</td>
<td>Coordinates in the input file as index triples (i, j, k) or spatial coordinates (x, y, z) in mm</td>
</tr>
<tr>
<td>argument: <code>'-%s'</code></td>
<td>argument: <code>'-%s'</code></td>
</tr>
<tr>
<td><strong>srad</strong>: (a float)</td>
<td><strong>srad</strong>: (a float)</td>
</tr>
<tr>
<td>radius in mm of the sphere that will be filled about each input (x,y,z) or (i,j,k) voxel. If the radius is not given, or is 0, then each input data line sets the value in only one voxel.</td>
<td>radius in mm of the sphere that will be filled about each input (x,y,z) or (i,j,k) voxel. If the radius is not given, or is 0, then each input data line sets the value in only one voxel.</td>
</tr>
<tr>
<td>argument: <code>'-srad %f'</code></td>
<td>argument: <code>'-srad %f'</code></td>
</tr>
<tr>
<td><strong>orient</strong>: (a tuple of the form: ('R' or 'L', 'A' or 'P', 'I' or 'S'))</td>
<td><strong>orient</strong>: (a tuple of the form: ('R' or 'L', 'A' or 'P', 'I' or 'S'))</td>
</tr>
<tr>
<td>Specifies the coordinate order used by -xyz. The code must be 3 letters, one each from the pairs {R,L} {A,P} {I,S}. The first letter gives the orientation of the x-axis, the second the orientation of the y-axis, the third the z-axis: R = right-to-left L = left-to-</td>
<td>Specifies the coordinate order used by -xyz. The code must be 3 letters, one each from the pairs {R,L} {A,P} {I,S}. The first letter gives the orientation of the x-axis, the second the orientation of the y-axis, the third the z-axis: R = right-to-left L = left-to-</td>
</tr>
<tr>
<td>(continues on next page)</td>
<td>(continues on next page)</td>
</tr>
</tbody>
</table>
right A = anterior-to-posterior P = posterior-to-anterior I = inferior-to-superior S = superior-to-inferior
If -orient isn't used, then the coordinate order of the -master (in_file) dataset is used to interpret (x,y,z) inputs.
argument: `'-orient %s'`
head_only: (a boolean)
    create only the .HEAD file which gets exploited by the AFNI matlab library function New_HEAD.m
    argument: `'-head_only'`
um_threads: (an integer (int or long), nipype default value: 1)
    set number of threads
outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
    AFNI output filetype
args: (a unicode string)
    Additional parameters to the command
    argument: `'%s'
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
    Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
    assembled file

References:

None

51.5.35 Unifize

Link to code

Wraps the executable command 3dUnifize.
3dUnifize - for uniformizing image intensity
• The input dataset is supposed to be a T1-weighted volume, possibly already skull-stripped (e.g., via 3dSkull-Strip). However, this program can be a useful step to take BEFORE 3dSkullStrip, since the latter program can fail if the input volume is strongly shaded – 3dUnifize will (mostly) remove such shading artifacts.
• The output dataset has the white matter (WM) intensity approximately uniformized across space, and scaled to peak at about 1000.
• The output dataset is always stored in float format!
• If the input dataset has more than 1 sub-brick, only sub-brick #0 will be processed!
• Want to correct EPI datasets for nonuniformity? You can try the new and experimental [Mar 2017] `-EPI` option.
• The principal motive for this program is for use in an image registration script, and it may or may not be useful otherwise.
• This program replaces the older (and very different) 3dUniformize, which is no longer maintained and may sublimate at any moment. (In other words, we do not recommend the use of 3dUniformize.)

For complete details, see the 3dUnifize Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> unifize = afni.Unifize()
>>> unifize.inputs.in_file = 'structural.nii'
```
>>> unifize.inputs.out_file = 'structural_unifized.nii'
>>> unifize.cmdline
'3dUnifize -prefix structural_unifized.nii -input structural.nii'
>>> res = unifize.run()  # doctest: +SKIP

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
   input file to 3dUnifize
   argument: `'-input %s'`, position: -1

[Optional]
out_file: (a pathlike object or string representing a file)
   output image file name
   argument: `'-prefix %s'`

t2: (a boolean)
   Treat the input as if it were T2-weighted, rather than T1-weighted.
   This processing is done simply by inverting the image contrast, processing it as if that result were T1-weighted, and then re-inverting the results counts of voxel overlap, i.e., each voxel will contain the number of masks that it is set in.
   argument: `'-T2'`

gm: (a boolean)
   Also scale to unifize 'gray matter' = lower intensity voxels (to aid in registering images from different scanners).
   argument: `'-GM'`

urad: (a float)
   Sets the radius (in voxels) of the ball used for the sneaky trick. Default value is 18.3, and should be changed proportionally if the dataset voxel size differs significantly from 1 mm.
   argument: `'-Urad %s'`

scale_file: (a pathlike object or string representing a file)
   output file name to save the scale factor used at each voxel
   argument: `'-ssave %s'`

no_duplo: (a boolean)
   Do NOT use the 'duplo down' step; this can be useful for lower resolution datasets.
   argument: `'-noduplo'`

epi: (a boolean)
   Assume the input dataset is a T2 (or T2*) weighted EPI time series. After computing the scaling, apply it to ALL volumes (TRs) in the input dataset. That is, a given voxel will be scaled by the same factor at each TR. This option also implies '-noduplo' and '-T2'. This option turns off '-GM' if you turned it on.
   argument: `'-EPI'`
   mutually_exclusive: gm
   requires: no_duplo, t2

rbt: (a tuple of the form: (a float, a float, a float))
   Option for AFNI experts only. Specify the 3 parameters for the algorithm:
   R = radius; same as given by option '-Urad', [default=18.3]
   b = bottom percentile of normalizing data range, [default=70.0]
   r = top percentile of normalizing data range, [default=80.0]
   argument: `'-rbt %f %f %f'`

t2_up: (a float)
   Option for AFNI experts only. Set the upper percentile point used for...


T2-T1 inversion. Allowed to be anything between 90 and 100 (inclusive), with default to 98.5 (for no good reason).

- **`-T2up %f`**

**cl_frac**: (a float)
Option for AFNI experts only. Set the automask 'clip level fraction'. Must be between 0.1 and 0.9. A small fraction means to make the initial threshold for clipping (a la 3dClipLevel) smaller, which will tend to make the mask larger. [default=0.1]

- **`-clfrac %f`**

**quiet**: (a boolean)
Don't print the progress messages.

- **`-quiet`**

**num_threads**: (an integer (int or long), nipype default value: 1)
set number of threads

**outputtype**: (`'NIFTI'` or `'AFNI'` or `'NIFTI_GZ'`)
AFNI output filetype

**args**: (a unicode string)
Additional parameters to the command

- **`%s`**

**environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

### Outputs:

- **scale_file**: (a pathlike object or string representing a file)
  scale factor file
- **out_file**: (a pathlike object or string representing an existing file)
  unifized file

### References:

None

#### 51.5.36 ZCutUp

**Link to code**

Wraps the executable command 3dZcutup. Cut z-slices from a volume using AFNI 3dZcutup command.

For complete details, see the 3dZcutup Documentation.

### Examples

```python
>>> from nipype.interfaces import afni
>>> zcutup = afni.ZCutUp()
>>> zcutup.inputs.in_file = 'functional.nii'
>>> zcutup.inputs.out_file = 'functional_zcutup.nii'
>>> zcutup.inputs.keep = '0 10'
>>> zcutup.cmdline
'3dZcutup -keep 0 10 -prefix functional_zcutup.nii functional.nii'
>>> res = zcutup.run()  # doctest: +SKIP
```

Inputs:
51.5.37 Zcat

Link to code
Wraps the executable command 3dZcat.
Copies an image of one type to an image of the same or different type using 3dZcat command
For complete details, see the 3dZcat Documentation.

Examples

```python
>>> from nipype.interfaces import afni
>>> zcat = afni.Zcat()
>>> zcat.inputs.in_files = ['./functional2.nii', './functional3.nii']
>>> zcat.inputs.out_file = 'cat_functional.nii'
>>> zcat.cmdline
>>> res = zcat.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]

in_files: (a list of items which are a pathlike object or string representing an existing file)
    input file to 3dZcutup
    argument: ``"%s```, position: -1

[Optional]

out_file: (a pathlike object or string representing a file)
    output image file name
    argument: ``-prefix %s``

keep: (a unicode string)
    slice range to keep in output
    argument: ``-keep %s``
	num_threads: (an integer (int or long), nipype default value: 1)
    set number of threads

toutputtype: ("NIFTI" or "AFNI" or "NIFTI_GZ")
    AFNI output filetype

args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})
    Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
    output file
Optional

out_file: (a pathlike object or string representing a file)
  output dataset prefix name (default 'zcat')
  argument: `'-prefix %s'`
datum: ('byte' or 'short' or 'float')
  specify data type for output. Valid types are 'byte', 'short' and
  'float'.
  argument: `'-datum %s'`
verb: (a boolean)
  print out some verbositiness as the program proceeds.
  argument: `'-verb'`
fscale: (a boolean)
  Force scaling of the output to the maximum integer range. This only
  has effect if the output datum is byte or short (either forced or
  defaulted). This option is sometimes necessary to eliminate
  unpleasant truncation artifacts.
  argument: `'-fscale'`
mutually_exclusive: nscale
nscale: (a boolean)
  Don't do any scaling on output to byte or short datasets. This may
  be especially useful when operating on mask datasets whose output
  values are only 0's and 1's.
  argument: `'-nscale'`
mutually_exclusive: fscale
num_threads: (an integer (int or long), nipype default value: 1)
  set number of threads
outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
  AFNI output filetype
args: (a unicode string)
  Additional parameters to the command
  argument: `'%s'`
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
  Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
  output file

References:
None

51.5.38 Zeropad

Link to code
Wraps the executable command `3dZeropad`.
Adds planes of zeros to a dataset (i.e., pads it out).
For complete details, see the `3dZeropad Documentation`.

Examples
>>> from nipype.interfaces import afni
>>> zeropad = afni.Zeropad()
>>> zeropad.inputs.in_files = 'functional.nii'
>>> zeropad.inputs.out_file = 'pad_functional.nii'
>>> zeropad.inputs.I = 10
>>> zeropad.inputs.S = 10
>>> zeropad.inputs.A = 10
>>> zeropad.inputs.P = 10
>>> zeropad.inputs.R = 10
>>> zeropad.inputs.L = 10
>>> zeropad.cmdline
'3dZeropad -A 10 -I 10 -L 10 -P 10 -R 10 -S 10 -prefix pad_functional.nii
→functional.nii'
>>> res = zeropad.run()  # doctest: +SKIP

Inputs:

[Mandatory]
in_files: (a pathlike object or string representing an existing file)
    input dataset
    argument: ``%s``, position: -1

[Optional]
out_file: (a pathlike object or string representing a file)
    output dataset prefix name (default 'zeropad')
    argument: ``-prefix %s``
I: (an integer (int or long))
    adds 'n' planes of zero at the Inferior edge
    argument: ``-I %i``
    mutually_exclusive: master
S: (an integer (int or long))
    adds 'n' planes of zero at the Superior edge
    argument: ``-S %i``
    mutually_exclusive: master
A: (an integer (int or long))
    adds 'n' planes of zero at the Anterior edge
    argument: ``-A %i``
    mutually_exclusive: master
P: (an integer (int or long))
    adds 'n' planes of zero at the Posterior edge
    argument: ``-P %i``
    mutually_exclusive: master
L: (an integer (int or long))
    adds 'n' planes of zero at the Left edge
    argument: ``-L %i``
    mutually_exclusive: master
R: (an integer (int or long))
    adds 'n' planes of zero at the Right edge
    argument: ``-R %i``
    mutually_exclusive: master
z: (an integer (int or long))
    adds 'n' planes of zero on EACH of the dataset z-axis (slice-
    direction) faces
    argument: ``-z %i``
    mutually_exclusive: master
RL: (an integer (int or long))
    specify that planes should be added or cut symmetrically to make the
    resulting volume haveN slices in the right-left direction

(continues on next page)
argument: `''-RL %i''`
mutually_exclusive: master

AP: (an integer (int or long))
specify that planes should be added or cut symmetrically to make the
resulting volume have N slices in the anterior-posterior direction
argument: `''-AP %i''`
mutually_exclusive: master

IS: (an integer (int or long))
specify that planes should be added or cut symmetrically to make the
resulting volume have N slices in the inferior-superior direction
argument: `''-IS %i''`
mutually_exclusive: master

mm: (a boolean)
pad counts 'n' are in mm instead of slices, where each 'n' is an
integer and at least 'n' mm of slices will be added/removed; e.g., n
= 3 and slice thickness = 2.5 mm ==> 2 slices added
argument: `''-mm''`
mutually_exclusive: master

master: (a pathlike object or string representing a file)
match the volume described in dataset 'mset', where mset must have
the same orientation and grid spacing as dataset to be padded. the
goal of -master is to make the output dataset from 3dZeropad match
the spatial 'extents' of mset by adding or subtracting slices as
needed. you can't use -I,-S,..., or -mm with -master
argument: `''-master %s''`
mutually_exclusive: I, S, A, P, L, R, z, RL, AP, IS, mm

num_threads: (an integer (int or long), nipype default value: 1)
set number of threads

outputtype: ('NIFTI' or 'AFNI' or 'NIFTI_GZ')
AFNI output filetype

args: (a unicode string)
Additional parameters to the command
argument: `''%s''`

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
output file

References:

None None
52.1 interfaces.ants.legacy

52.1.1 GenWarpFields

Link to code
Wraps the executable command antsIntroduction.sh.

Inputs:

[Mandatory]
reference_image: (a pathlike object or string representing an existing file)
    template file to warp to
    argument: ``-r %s``

input_image: (a pathlike object or string representing an existing file)
    input image to warp to template
    argument: ``-i %s``

[Optional]
dimension: (3 or 2, nipype default value: 3)
    image dimension (2 or 3)
    argument: ``-d %d``, position: 1

force_proceed: (a boolean)
    force script to proceed even if headers may be incompatible
    argument: ``-f 1``

inverse_warp_template_labels: (a boolean)
    Applies inverse warp to the template labels to estimate label positions in target space (use for template-based segmentation)
    argument: ``-l``

max_iterations: (a list of items which are an integer (int or long))
    maximum number of iterations (must be list of integers in the form [J,K,L...]: J = coarsest resolution iterations, K = middle resolution iterations, L = fine resolution iterations)
    argument: ``-m %s``

bias_field_correction: (a boolean)
    Applies bias field correction to moving image
argument: `'-n l'`
similarity_metric: (`PR' or 'CC' or 'MI' or 'MSQ')
Type of similarity metric used for registration (CC = cross correlation, MI = mutual information, PR = probability mapping, MSQ = mean square difference)
argument: `'-s %s'`
transformation_model: (`GR' or 'EL' or 'SY' or 'S2' or 'EX' or 'DD' or 'RI' or 'RA', nipype default value: GR)
Type of transformation model used for registration (EL = elastic transformation model, SY = SyN with time, arbitrary number of time points, S2 = SyN with time optimized for 2 time points, GR = greedy SyN, EX = exponential, DD = diffeomorphic demons style exponential mapping, RI = purely rigid, RA = affine rigid)
argument: `'-t %s'`
out_prefix: (a unicode string, nipype default value: ants_)
Prefix that is prepended to all output files (default = ants_)
argument: `'-o %s'`
quality_check: (a boolean)
Perform a quality check of the result
argument: `'-q l'`
um_threads: (an integer (int or long), nipype default value: 1)
Number of ITK threads to use
args: (a unicode string)
Additional parameters to the command
argument: `'%s'`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

affine_transformation: (a pathlike object or string representing an existing file)
    affine (prefix_Affine.txt)
warp_field: (a pathlike object or string representing an existing file)
    warp field (prefix_Warp.nii)
inverse_warp_field: (a pathlike object or string representing an existing file)
    inverse warp field (prefix_InverseWarp.nii)
input_file: (a pathlike object or string representing an existing file)
    input image (prefix_repaired.nii)
output_file: (a pathlike object or string representing an existing file)
    output image (prefix_deformed.nii)

52.1.2 antsIntroduction

Link to code
Wraps the executable command antsIntroduction.sh.
Uses ANTS to generate matrices to warp data from one space to another.
Examples

```python
>>> from nipype.interfaces.ants.legacy import antsIntroduction
>>> warp = antsIntroduction()
>>> warp.inputs.reference_image = 'Template_6.nii'
>>> warp.inputs.input_image = 'structural.nii'
>>> warp.inputs.max_iterations = [30,90,20]
>>> warp.cmdline
'antsIntroduction.sh -d 3 -i structural.nii -m 30x90x20 -o ants_-r Template_6.
˓
→nii -t GR'
```

Inputs:

[Mandatory]

- **reference_image**: (a pathlike object or string representing an existing file)
  - template file to warp to
  - argument: `'-r %s'`
- **input_image**: (a pathlike object or string representing an existing file)
  - input image to warp to template
  - argument: `'-i %s'`

[Optional]

- **dimension**: (3 or 2, nipype default value: 3)
  - image dimension (2 or 3)
  - argument: `'-d %d'`, position: 1
- **force_proceed**: (a boolean)
  - force script to proceed even if headers may be incompatible
  - argument: `'-f 1'`
- **inverse_warp_template_labels**: (a boolean)
  - Applies inverse warp to the template labels to estimate label positions in target space (use for template-based segmentation)
  - argument: `'-l'`
- **max_iterations**: (a list of items which are an integer (int or long))
  - maximum number of iterations (must be list of integers in the form [J,K,L...]: J = coarsest resolution iterations, K = middle resolution iterations, L = fine resolution iterations)
  - argument: `'-m %s'`
- **bias_field_correction**: (a boolean)
  - Applies bias field correction to moving image
  - argument: `'-n 1'`
- **similarity_metric**: ('PR' or 'CC' or 'MI' or 'MSQ')
  - Type of similarity metric used for registration (CC = cross correlation, MI = mutual information, PR = probability mapping, MSQ = mean square difference)
  - argument: `'-s %s'`
- **transformation_model**: ('GR' or 'EL' or 'SY' or 'S2' or 'EX' or 'DD' or 'RI' or 'RA', nipype default value: GR)
  - Type of transformation model used for registration (EL = elastic transformation model, SY = SyN with time, arbitrary number of time points, S2 = SyN with time optimized for 2 time points, GR = greedy SyN, EX = exponential, DD = diffeomorphic demons style exponential mapping, RI = purely rigid, RA = affine rigid)
  - argument: `'-t %s'`
- **out_prefix**: (a unicode string, nipype default value: ants_)
  - Prefix that is prepended to all output files (default = ants_)
  - argument: `'-o %s'`
quality_check: (a boolean)
Perform a quality check of the result
argument: `--q 1`

num_threads: (an integer (int or long), nipype default value: 1)
Number of ITK threads to use

args: (a unicode string)
Additional parameters to the command
argument: `-%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str'
and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

affine_transformation: (a pathlike object or string representing an existing file)
affine (prefix_Affine.txt)
warp_field: (a pathlike object or string representing an existing file)
warp field (prefix_Warp.nii)
inverse_warp_field: (a pathlike object or string representing an existing file)
inverse warp field (prefix_InverseWarp.nii)
input_file: (a pathlike object or string representing an existing file)
input image (prefix_repaired.nii)
output_file: (a pathlike object or string representing an existing file)
output image (prefix_deformed.nii)

52.1.3 buildtemplateparallel

Link to code
Wraps the executable command `buildtemplateparallel.sh`.
Generate a optimal average template

**Warning:** This can take a VERY long time to complete

Examples

```python
>>> from nipype.interfaces.ants.legacy import buildtemplateparallel
>>> tmpl = buildtemplateparallel()
>>> tmpl.inputs.in_files = ['T1.nii', 'structural.nii']
>>> tmpl.inputs.max_iterations = [30, 90, 20]
>>> tmpl.cmdline
'buildtemplateparallel.sh -d 3 -i 4 -m 30x90x20 -o antsTMPL_ -c 0 -t GR T1.nii structural.nii'
```

Inputs:

[Mandatory]
in_files: (a list of items which are a pathlike object or string representing an existing file)
list of images to generate template from
argument: '%s', position: -1

[Optional]
dimension: (3 or 2 or 4, nipype default value: 3)
    image dimension (2, 3 or 4)
    argument: '--d %d', position: 1
out_prefix: (a unicode string, nipype default value: antsTMPL_)
    Prefix that is prepended to all output files (default = antsTMPL_)
    argument: '--o %s'
parallelization: (0 or 1 or 2, nipype default value: 0)
    control for parallel processing (0 = serial, 1 = use PBS, 2 = use
    PEXEC, 3 = use Apple XGrid
    argument: '--c %d'
gradient_step_size: (a float)
    smaller magnitude results in more cautious steps (default = .25)
    argument: '--g %f'
iteration_limit: (an integer (int or long), nipype default value: 4)
    iterations of template construction
    argument: '--i %d'
um_cores: (an integer (int or long))
    Requires parallelization = 2 (PEXEC). Sets number of cpu cores to
    use
    argument: '--j %d'
    requires: parallelization
max_iterations: (a list of items which are an integer (int or long))
    maximum number of iterations (must be list of integers in the form
    [J,K,L...] : J = coarsest resolution iterations, K = middle
    resolution iterations, L = fine resolution iterations
    argument: '--m %s'
bias_field_correction: (a boolean)
    Applies bias field correction to moving image
    argument: '--n 1'
rigid_body_registration: (a boolean)
    registers inputs before creating template (useful if no initial
    template available)
    argument: '--r 1'
similarity_metric: ('PR' or 'CC' or 'MI' or 'MSQ')
    Type of similarity metric used for registration (CC = cross
    correlation, MI = mutual information, PR = probability mapping, MSQ
    = mean square difference)
    argument: '--s %s'
transformation_model: ('GR' or 'EL' or 'SY' or 'S2' or 'EX' or 'DD',
    nipype default value: GR)
    Type of transformation model used for registration (EL = elastic
    transformation model, SY = SyN with time, arbitrary number of time
    points, S2 = SyN with time optimized for 2 time points, GR = greedy
    SyN, EX = exponential, DD = diffeomorphic demons style exponential
    mapping
    argument: '--t %s'
use_first_as_target: (a boolean)
    uses first volume as target of all inputs. When not used, an
    unbiased average image is used to start.
num_threads: (an integer (int or long), nipype default value: 1)
    Number of ITK threads to use
args: (a unicode string)
    Additional parameters to the command
    argument: '%s'
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

final_template_file: (a pathlike object or string representing an existing file)
final ANTS template
template_files: (a list of items which are a pathlike object or string representing an existing file)
Templates from different stages of iteration
subject_outfiles: (a list of items which are a pathlike object or string representing an existing file)
Outputs for each input image. Includes warp field, inverse warp, Affine, original image (repaired) and warped image (deformed)

52.2 interfaces.ants.registration

52.2.1 CompositeTransformUtil

Link to code
Wraps the executable command CompositeTransformUtil. ANTs utility which can combine or break apart transform files into their individual constituent components.

Examples

```python
>>> from nipype.interfaces.ants import CompositeTransformUtil
>>> tran = CompositeTransformUtil()
>>> tran.inputs.process = 'disassemble'
>>> tran.inputs.in_file = 'output_Composite.h5'
>>> tran.cmdline
'CompositeTransformUtil --disassemble output_Composite.h5 transform'
>>> tran.run()  # doctest: +SKIP
```

example for assembling transformation files

```python
>>> from nipype.interfaces.ants import CompositeTransformUtil
>>> tran = CompositeTransformUtil()
>>> tran.inputs.process = 'assemble'
>>> tran.inputs.out_file = 'my.h5'
>>> tran.inputs.in_file = ['AffineTransform.mat', 'DisplacementFieldTransform.nii.gz']
>>> tran.cmdline
'CompositeTransformUtil --assemble my.h5 AffineTransform.mat DisplacementFieldTransform.nii.gz '
>>> tran.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a list of items which are a pathlike object or string representing an existing file)
Input transform file(s)
argument: ``%s...``, position: 3
52.2.2 MeasureImageSimilarity

Link to code

Wraps the executable command MeasureImageSimilarity.

Examples

```python
>>> from nipype.interfaces.ants import MeasureImageSimilarity
>>> sim = MeasureImageSimilarity()
>>> sim.inputs.dimension = 3
>>> sim.inputs.metric = 'MI'
>>> sim.inputs.fixed_image = 'T1.nii'
>>> sim.inputs.moving_image = 'resting.nii'
>>> sim.inputs.metric_weight = 1.0
>>> sim.inputs.radius_or_number_of_bins = 5
>>> sim.inputs.sampling_strategy = 'Regular'
>>> sim.inputs.sampling_percentage = 1.0
>>> sim.inputs.fixed_image_mask = 'mask.nii'
>>> sim.inputs.moving_image_mask = 'mask.nii.gz'
>>> sim.cmdline
'MeasureImageSimilarity --dimensionality 3 --masks ["mask.nii", "mask.nii.gz"] --metric MI["T1.nii", "resting.nii", 1.0, 5, Regular, 1.0]'
```
fixed_image: (a pathlike object or string representing an existing file)
  Image to which the moving image is warped
moving_image: (a pathlike object or string representing an existing file)
  Image to apply transformation to (generally a coregistered functional)
metric: ('CC' or 'MI' or 'Mattes' or 'MeanSquares' or 'Demons' or 'GC')
  argument: ``%s``
radius_or_number_of_bins: (an integer (int or long))
  The number of bins in each stage for the MI and Mattes metric, or the radius for other metrics
  requires: metric
sampling_percentage: (0.0 <= a floating point number <= 1.0)
  Percentage of points accessible to the sampling strategy over which to optimize the metric.
  requires: metric

[Optional]
dimension: (2 or 3 or 4)
  Dimensionality of the fixed/moving image pair
  argument: ``--dimensionality %d``, position: 1
metric_weight: (a float, nipype default value: 1.0)
  The "metricWeight" variable is not used.
  requires: metric
metric_sampling_strategy: ('None' or 'Regular' or 'Random', nipype default value: None)
  Manner of choosing point set over which to optimize the metric.
  Defaults to "None" (i.e. a dense sampling of one sample per voxel).
  requires: metric
fixed_image_mask: (a pathlike object or string representing an existing file)
  Mask used to limit metric sampling region of the fixed image
  argument: ``%s``
moving_image_mask: (a pathlike object or string representing an existing file)
  Mask used to limit metric sampling region of the moving image
  requires: fixed_image_mask
num_threads: (an integer (int or long), nipype default value: 1)
  Number of ITK threads to use
args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:
similarity: (a float)

52.2.3 Registration

Link to code
Wraps the executable command antsRegistration.
ANTS Registration command for registration of images

`antsRegistration` registers a `moving_image` to a `fixed_image`, using a predefined (sequence of) cost function(s) and transformation operations. The cost function is defined using one or more ‘metrics’, specifically local cross-correlation (CC), Mean Squares (MeanSquares), Demons (Demons), global correlation (GC), or Mutual Information (Mattes or MI).

ANTS can use both linear (Translation, Rigid, Affine, CompositeAffine, or Translation) and non-linear transformations (BSpline, GaussianDisplacementField, TimeVaryingVelocityField, TimeVaryingBSplineVelocityField, SyN, BSplineSyN, Exponential, or BSplineExponential). Usually, registration is done in multiple stages. For example first an Affine, then a Rigid, and ultimately a non-linear (Syn)-transformation.

`antsRegistration` can be initialized using one or more transforms from `moving_image` to `fixed_image` with the `initial_moving_transform`-input. For example, when you already have a warprfield that corrects for geometrical distortions in an EPI (functional) image, that you want to apply before an Affine registration to a structural image. You could put this transform into ‘intial_moving_transform’.

The Registration-interface can output the resulting transform(s) that map `moving_image` to `fixed_image` in a single file as a `composite_transform` (if `write_composite_transform` is set to True), or a list of transforms as `forwards_transforms`. It can also output inverse transforms (from `fixed_image` to `moving_image`) in a similar fashion using `inverse_composite_transform`. Note that the order of `forward_transforms` is in ‘natural’ order: the first element should be applied first, the last element should be applied last.

Note, however, that ANTS tools always apply lists of transformations in reverse order (the last transformation in the list is applied first). Therefore, if the output `forward_transforms` is a list, one can not directly feed it into, for example, `ants.ApplyTransforms`. To make `ants.ApplyTransforms` apply the transformations in the same order as `ants.Registration`, you have to provide the list of transformations in reverse order from `forward_transforms`. `reverse_forward_transforms` outputs `forward_transforms` in reverse order and can be used for this purpose. Note also that, because `composite_transform` is always a single file, this output is preferred for most use-cases.

More information can be found in the ANTS manual. See below for some useful examples.

**Examples**

Set up a Registration node with some default settings. This Node registers ‘fixed1.nii’ to ‘moving1.nii’ by first fitting a linear ‘Affine’ transformation, and then a non-linear ‘SyN’ transformation, both using the Mutual Information-cost metric.

The registration is initialized by first applying the (linear) transform `trans.mat`.

```python
>>> import copy, pprint
>>> from nipype.interfaces.ants import Registration
>>> reg = Registration()
>>> reg.inputs.fixed_image = 'fixed1.nii'
>>> reg.inputs.moving_image = 'moving1.nii'
>>> reg.inputs.output_transform_prefix = "output_"
>>> reg.inputs.initial_moving_transform = 'trans.mat'
>>> reg.inputs.transforms = ['Affine', 'SyN']
>>> reg.inputs.transform_parameters = [(2.0,), (0.25, 3.0, 0.0)]
>>> reg.inputs.number_of_iterations = [[1500, 200], [100, 50, 30]]
>>> reg.inputs.dimension = 3
>>> reg.inputs.write_composite_transform = True
>>> reg.inputs.collapse_output_transforms = False
>>> reg.inputs.initialize_transforms_per_stage = False
>>> reg.inputs.metric = ['Mattes']*2
>>> reg.inputs.metric_weight = [1]*2 # Default (value ignored currently by ANTs)
>>> reg.inputs.radius_or_number_of_bins = [32]*2
>>> reg.inputs.sampling_strategy = ['Random', None]
```
>>> reg.inputs.sampling_percentage = [0.05, None]
>>> reg.inputs.convergence_threshold = [1.e-8, 1.e-9]
>>> reg.inputs.convergence_window_size = [20]*2
>>> reg.inputs.smoothing_sigmas = [[1,0], [2,1,0]]
>>> reg.inputs.sigma_units = ['vox'] * 2
>>> reg.inputs.shrink_factors = [[2,1], [3,2,1]]
>>> reg.inputs.use_estimate_learning_rate_once = [True, True]  # This is the default
>>> reg.inputs.use_histogram_matching = [True, True]
>>> reg.inputs.output_warped_image = 'output_warped_image.nii.gz'
>>> reg.cmdline
'antsRegistration --collapse-output-transforms 0 --dimensionality 3 --initial-moving-transform [ trans.mat, 0 ] --initialize-transforms-per-stage 0 --interpolation Linear --output [ output_, output_warped_image.nii.gz ] --transform Affine[ 2.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1, 32, Random, 0.05 ] --convergence [ 1500x200, 1e-08, 20 ] --smoothing-sigmas 1.0x0.0vox --shrink-factors 2x1 --use-estimate-learning-rate-once 1 --use-histogram-matching --transform SyN[ 0.25, 3.0, 0.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1, 32 ] --convergence [ 100x50x30, 1e-09, 20 ] --smoothing-sigmas 2.0x1.0x0.0vox --shrink-factors 3x2x1 --use-estimate-learning-rate-once 1 --use-histogram-matching 1 --winsorize-image-intensities [ 0.0, 1.0 ] --write-composite-transform 1'
>>> reg.run()  # doctest: +SKIP

Same as reg1, but first invert the initial transform ('trans.mat') before applying it.

>>> reg1 = copy.deepcopy(reg)
>>> reg1.inputs.invert_initial_moving_transform = True

>>> reg1.cmdline
'antsRegistration --collapse-output-transforms 0 --dimensionality 3 --initial-moving-transform [ trans.mat, 1 ] --initialize-transforms-per-stage 0 --interpolation Linear --output [ output_, output_warped_image.nii.gz ] --transform Affine[ 2.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1, 32, Random, 0.05 ] --convergence [ 1500x200, 1e-08, 20 ] --smoothing-sigmas 1.0x0.0vox --shrink-factors 2x1 --use-estimate-learning-rate-once 1 --use-histogram-matching --transform SyN[ 0.25, 3.0, 0.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1, 32 ] --convergence [ 100x50x30, 1e-09, 20 ] --smoothing-sigmas 2.0x1.0x0.0vox --shrink-factors 3x2x1 --use-estimate-learning-rate-once 1 --use-histogram-matching 1 --winsorize-image-intensities [ 0.025, 1.0 ] --write-composite-transform 1'
>>> reg1.run()  # doctest: +SKIP

Clip extremely high intensity data points using winsorize_upper_quantile. All data points higher than the 0.975 quantile are set to the value of the 0.975 quantile.

>>> reg2 = copy.deepcopy(reg)
>>> reg2.inputs.winsorize_upper_quantile = 0.975
>>> reg2.cmdline
'antsRegistration --collapse-output-transforms 0 --dimensionality 3 --initial-moving-transform [ trans.mat, 1 ] --initialize-transforms-per-stage 0 --interpolation Linear --output [ output_, output_warped_image.nii.gz ] --transform Affine[ 2.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1, 32, Random, 0.05 ] --convergence [ 1500x200, 1e-08, 20 ] --smoothing-sigmas 1.0x0.0vox --shrink-factors 2x1 --use-estimate-learning-rate-once 1 --use-histogram-matching --transform SyN[ 0.25, 3.0, 0.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1, 32 ] --convergence [ 100x50x30, 1e-09, 20 ] --smoothing-sigmas 2.0x1.0x0.0vox --shrink-factors 3x2x1 --use-estimate-learning-rate-once 1 --use-histogram-matching 1 --winsorize-image-intensities [ 0.0, 0.975 ] --write-composite-transform 1'
(continues on next page)
Clip extremely low intensity data points using winsorize_lower_quantile. All data points lower than the 0.025 quantile are set to the original value at the 0.025 quantile.

```python
>>> reg3 = copy.deepcopy(reg)
>>> reg3.inputs.winsorize_lower_quantile = 0.025
>>> reg3.inputs.winsorize_upper_quantile = 0.975
>>> reg3.cmdline
'antsRegistration --collapse-output-transforms 0 --dimensionality 3 --initial-moving-transform [ trans.mat, 1 ] --initialize-transforms-per-stage 0 -- interpolation Linear --output [ output_, output_warped_image.nii.gz ] -- transform Affine[ 2.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1, 32, Random, 0.05 ] --convergence [ 1500x200, 1e-08, 20 ] --smoothing-sigmas 1.0x0.0vox -- shrink-factors 2x1 --use-estimate-learning-rate-once 1 --use-histogram-matching 1 --winsorize-image-intensities [ 0.025, 0.975 ] --write-composite-transform 1'
```

Use float instead of double for computations (saves memory usage)

```python
>>> reg3a = copy.deepcopy(reg)
>>> reg3a.inputs.float = True
>>> reg3a.cmdline
'antsRegistration --collapse-output-transforms 0 --dimensionality 3 --float 1 -- initial-moving-transform [ trans.mat, 1 ] --initialize-transforms-per-stage 0 -- interpolation Linear --output [ output_, output_warped_image.nii.gz ] -- transform Affine[ 2.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1, 32, Random, 0.05 ] --convergence [ 1500x200, 1e-08, 20 ] --smoothing-sigmas 1.0x0.0vox -- shrink-factors 2x1 --use-estimate-learning-rate-once 1 --use-histogram-matching 1 --winsorize-image-intensities [ 0.0, 1.0 ] --write-composite-transform 1'
```

Force to use double instead of float for computations (more precision and memory usage).

```python
>>> reg3b = copy.deepcopy(reg)
>>> reg3b.inputs.float = False
>>> reg3b.cmdline
'antsRegistration --collapse-output-transforms 0 --dimensionality 3 --float 0 -- initial-moving-transform [ trans.mat, 1 ] --initialize-transforms-per-stage 0 -- interpolation Linear --output [ output_, output_warped_image.nii.gz ] -- transform Affine[ 2.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1, 32, Random, 0.05 ] --convergence [ 1500x200, 1e-08, 20 ] --smoothing-sigmas 1.0x0.0vox -- shrink-factors 2x1 --use-estimate-learning-rate-once 1 --use-histogram-matching 1 --winsorize-image-intensities [ 0.0, 1.0 ] --write-composite-transform 1'
```

‘collapse_output_transforms’ can be used to put all transformation in a single ‘composite_transform’-file. Note that forward_transforms will now be an empty list.
>>> # Test collapse transforms flag
>>> reg4 = copy.deepcopy(reg)
>>> reg4.inputs.save_state = 'trans.mat'
>>> reg4.inputs.restore_state = 'trans.mat'
>>> reg4.inputs.initialize_transforms_per_stage = True
>>> reg4.inputs.collapse_output_transforms = True
>>> outputs = reg4._list_outputs()
>>> pprint.pprint(outputs)  # doctest: +ELLIPSIS,
{'composite_transform': '...data/output_Composite.h5',
 'elapsed_time': <undefined>,
 'forward_invert_flags': [],
 'forward_transforms': [],
 'inverse_composite_transform': '...data/output_InverseComposite.h5',
 'inverse_warped_image': <undefined>,
 'metric_value': <undefined>,
 'reverse_invert_flags': [],
 'reverse_transforms': [],
 'save_state': '...data/trans.mat',
 'warped_image': '...data/output_warped_image.nii.gz'}

>>> reg4.cmdline
'antsRegistration --collapse-output-transforms 1 --dimensionality 3 --initial-
  moving-transform [ trans.mat, 1 ] --initialize-transforms-per-stage 1 --
  interpolation Linear --output [ output_, output_warped_image.nii.gz ] --restore-
  state trans.mat --save-state trans.mat --transform Affine[ 2.0 ] --metric_Cp
  Mattes[ fixed1.nii, moving1.nii, 1, 32, Random, 0.05 ] --convergence [ 1500x200,
  1e-08, 20 ] --smoothing-sigmas 1.0x0.0vox --shrink-factors 2x1 --use-estimate-
  learning-rate-once 1 --use-histogram-matching 1 --transform SyN[ 0.25, 3.0, 0.0
  ] --metric Mattes[ fixed1.nii, moving1.nii, 1, 32 ] --convergence [ 100x50x30,
  1e-09, 20 ] --smoothing-sigmas 2.0x1.0x0.0vox --shrink-factors 3x2x1 --use-
  estimate-learning-rate-once 1 --use-histogram-matching 1 --winsorize-image-
  intensities [ 0.0, 1.0 ] --write-composite-transform 1'

>>> # Test collapse transforms flag
>>> reg4b = copy.deepcopy(reg4)
>>> reg4b.inputs.write_composite_transform = False
>>> outputs = reg4b._list_outputs()
>>> pprint.pprint(outputs)  # doctest: +ELLIPSIS,
{'composite_transform': <undefined>,
 'elapsed_time': <undefined>,
 'forward_invert_flags': [False, False],
 'forward_transforms': ['...data/output_0GenericAffine.mat',
 '...data/output_1Warp.nii.gz'],
 'inverse_composite_transform': <undefined>,
 'inverse_warped_image': <undefined>,
 'metric_value': <undefined>,
 'reverse_invert_flags': [True, False],
 'reverse_transforms': ['...data/output_0InverseWarp.nii.gz',
 '...data/output_warped_image.nii.gz'],
 'save_state': '...data/trans.mat',
 'warped_image': '...data/output_warped_image.nii.gz'}

>>> reg4b.aggregate_outputs()  # doctest: +SKIP

>>> reg4b.cmdline
'antsRegistration --collapse-output-transforms 1 --dimensionality 3 --initial-
  moving-transform [ trans.mat, 1 ] --initialize-transforms-per-stage 1 --
  interpolation Linear --output [ output_, output_warped_image.nii.gz ] --restore-
  state trans.mat --save-state trans.mat --transform Affine[ 2.0 ] --metric_Cp
  Mattes[ fixed1.nii, moving1.nii, 1, 32, Random, 0.05 ] --convergence [ 1500x200,
  1e-08, 20 ] --smoothing-sigmas 1.0x0.0vox --shrink-factors 2x1 --use-estimate-
  learning-rate-once 1 --use-histogram-matching 1 --transform SyN[ 0.25, 3.0, 0.0
  ] --metric Mattes[ fixed1.nii, moving1.nii, 1, 32 ] --convergence [ 100x50x30,
  1e-09, 20 ] --smoothing-sigmas 2.0x1.0x0.0vox --shrink-factors 3x2x1 --use-
  estimate-learning-rate-once 1 --use-histogram-matching 1 --winsorize-image-
  intensities [ 0.0, 1.0 ] --write-composite-transform 0'}
One can use multiple similarity metrics in a single registration stage. The Node below first performs a linear registration using only the Mutual Information ('Mattes')-metric. In a second stage, it performs a non-linear registration ('Syn') using both a Mutual Information and a local cross-correlation ('CC')-metric. Both metrics are weighted equally ('metric_weight' is .5 for both). The Mutual Information-metric uses 32 bins. The local cross-correlations (correlations between every voxel’s neighborhoods) is computed with a radius of 4.

```python
>>> # Test multiple metrics per stage
>>> reg5 = copy.deepcopy(reg)
>>> reg5.inputs.fixed_image = 'fixed1.nii'
>>> reg5.inputs.moving_image = 'moving1.nii'
>>> reg5.inputs.metric = ['Mattes', 'CC']
>>> reg5.inputs.metric_weight = [.5, .5]
>>> reg5.inputs.radius_or_number_of_bins = [32, 4]
>>> reg5.inputs.sampling_strategy = ['Random', None]
>>> reg5.inputs.sampling_percentage = [0.05, 0.10]
>>> reg5.cmdline
'antsRegistration --collapse-output-transforms 0 --dimensionality 3 --initial-moving-transform [ trans.mat, 1 ] --initialize-transforms-per-stage 0 -- interpolation Linear --output [ output_, output_warped_image.nii.gz ] -- transform Affine [ 2.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1, 32, Random, 0.05 ] --convergence [ 1500x200, 1e-08, 20 ] --smoothing-sigmas 1.0x0.0vox -- shrink-factors 2x1 --use-estimate-learning-rate-once 1 --use-histogram-matching -- transform Syn[ 0.25, 3.0, 0.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 0.5, 32, None, 0.05 ] --metric CC[ fixed1.nii, moving1.nii, 0.5, 4, None, 0.1 ] -- convergence [ 100x50x30, 1e-09, 20 ] --smoothing-sigmas 2.0x1.0x0.0vox -- shrink-factors 3x2x1 --use-estimate-learning-rate-once 1 --use-histogram-matching -- winsorize-image-intensities [ 0.0, 1.0 ] --write-composite-transform 1'
```

ANTS Registration can also use multiple modalities to perform the registration. Here it is assumed that fixed1.nii and fixed2.nii are in the same space, and so are moving1.nii and moving2.nii. First, a linear registration is performed matching fixed1.nii to moving1.nii, then a non-linear registration is performed to match fixed2.nii to moving2.nii, starting from the transformation of the first step.

```python
>>> # Test multiple inputs
>>> reg6 = copy.deepcopy(reg5)
>>> reg6.inputs.fixed_image = ['fixed1.nii', 'fixed2.nii']
>>> reg6.inputs.moving_image = ['moving1.nii', 'moving2.nii']
>>> reg6.cmdline
'antsRegistration --collapse-output-transforms 0 --dimensionality 3 --initial-moving-transform [ trans.mat, 1 ] --initialize-transforms-per-stage 0 -- interpolation Linear --output [ output_, output_warped_image.nii.gz ] -- transform Affine [ 2.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1, 32, Random, 0.05 ] --convergence [ 1500x200, 1e-08, 20 ] --smoothing-sigmas 1.0x0.0vox -- shrink-factors 2x1 --use-estimate-learning-rate-once 1 --use-histogram-matching -- transform Syn[ 0.25, 3.0, 0.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 0.5, 32, None, 0.05 ] --metric CC[ fixed1.nii, moving1.nii, 0.5, 4, None, 0.1 ] -- convergence [ 100x50x30, 1e-09, 20 ] --smoothing-sigmas 2.0x1.0x0.0vox -- shrink-factors 3x2x1 --use-estimate-learning-rate-once 1 --use-histogram-matching -- winsorize-image-intensities [ 0.0, 1.0 ] --write-composite-transform 1'
```

Different methods can be used for the interpolation when applying transformations.

```python
>>> # Test Interpolation Parameters (BSpline)
>>> reg7a = copy.deepcopy(reg)
>>> reg7a.inputs.interpolation = 'BSpline'
>>> reg7a.cmdline
'antsRegistration --collapse-output-transforms 0 --dimensionality 3 --initial-moving-transform [ trans.mat, 1 ] --initialize-transforms-per-stage 0 -- interpolation Linear --output [ output_, output_warped_image.nii.gz ] -- transform Affine [ 2.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1, 32, Random, 0.05 ] --convergence [ 1500x200, 1e-08, 20 ] --smoothing-sigmas 1.0x0.0vox -- shrink-factors 2x1 --use-estimate-learning-rate-once 1 --use-histogram-matching -- transform Syn[ 0.25, 3.0, 0.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 0.5, 32, None, 0.05 ] --metric CC[ fixed2.nii, moving2.nii, 0.5, 4, None, 0.1 ] -- convergence [ 100x50x30, 1e-09, 20 ] --smoothing-sigmas 2.0x1.0x0.0vox -- shrink-factors 3x2x1 --use-estimate-learning-rate-once 1 --use-histogram-matching -- winsorize-image-intensities [ 0.0, 1.0 ] --write-composite-transform 1'
```
>>> reg7a.inputs.interpolation_parameters = (3,)
>>> reg7a.cmdline
'antsRegistration --collapse-output-transforms 0 --dimensionality 3 --initial-
→moving-transform [ trans.mat, 1 ] --initialize-transforms-per-stage 0 --
→transform Affine[ 2.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1, 32, Random,
→ 0.05 ] --convergence [ 1500x200, 1e-08, 20 ] --smoothing-sigmas 1.0x0.0vox --
→shrink-factors 2x1 --use-estimate-learning-rate-one 1 --use-histogram-matching,
→1 --transform Syn[ 0.25, 3.0, 0.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1,
→ 32 ] --convergence [ 100x50x30, 1e-09, 20 ] --smoothing-sigmas 2.0x1.0x0.0vox --
→shrink-factors 3x2x1 --use-estimate-learning-rate-one 1 --use-histogram-
→matching 1 --winsorize-image-intensities [ 0.0, 1.0 ] --write-composite-
→transform 1'

>>> # Test Interpolation Parameters (MultiLabel/Gaussian)
>>> reg7b = copy.deepcopy(reg)
>>> reg7b.inputs.interpolation = 'Gaussian'
>>> reg7b.inputs.interpolation_parameters = (1.0, 1.0)
>>> reg7b.cmdline
'antsRegistration --collapse-output-transforms 0 --dimensionality 3 --initial-
→moving-transform [ trans.mat, 1 ] --initialize-transforms-per-stage 0 --
→interpolation Gaussian[ 1.0, 1.0 ] --output [ output_, output_warped_image.nii.
→gz ] --transform Affine[ 2.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1, 32,
→Random, 0.05 ] --convergence [ 1500x200, 1e-08, 20 ] --smoothing-sigmas 1.0x0.0vox --
→shrink-factors 2x1 --use-estimate-learning-rate-one 1 --use-histogram-matching,
→1 --transform Syn[ 0.25, 3.0, 0.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1,
→32 ] --convergence [ 100x50x30, 1e-09, 20 ] --smoothing-sigmas 2.0x1.0x0.0vox --
→shrink-factors 3x2x1 --use-estimate-learning-rate-one 1 --use-histogram-
→matching 1 --winsorize-image-intensities [ 0.0, 1.0 ] --write-composite-
→transform 1'

BSplineSyN non-linear registration with custom parameters.

>>> # Test Extended Transform Parameters
>>> reg8 = copy.deepcopy(reg)
>>> reg8.inputs.transforms = ['Affine', 'BSplineSyN']
>>> reg8.inputs.transform_parameters = [(2.0,), (0.25, 26, 0, 3)]
>>> reg8.cmdline
'antsRegistration --collapse-output-transforms 0 --dimensionality 3 --initial-
→moving-transform [ trans.mat, 1 ] --initialize-transforms-per-stage 0 --
→interpolation Linear --output [ output_, output_warped_image.nii.gz ] --
→transform Affine[ 2.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1, 32, Random,
→ 0.05 ] --convergence [ 1500x200, 1e-08, 20 ] --smoothing-sigmas 1.0x0.0vox --
→shrink-factors 2x1 --use-estimate-learning-rate-one 1 --use-histogram-matching,
→1 --transform BSplineSyN[ 0.25, 26, 0, 3 ] --metric Mattes[ fixed1.nii, moving1.
→nii, 1, 32 ] --convergence [ 100x50x30, 1e-09, 20 ] --smoothing-sigmas 2.0x1.0
→0x0.0vox --shrink-factors 3x2x1 --use-estimate-learning-rate-one 1 --use-
→histogram-matching 1 --winsorize-image-intensities [ 0.0, 1.0 ] --write-
→composite-transform 1'

Mask the fixed image in the second stage of the registration (but not the first).

>>> # Test masking
>>> reg9 = copy.deepcopy(reg)
>>> reg9.inputs.fixed_image_masks = ['NULL', 'fixed1.nii']
>>> reg9.cmdline
'antsRegistration --collapse-output-transforms 0 --dimensionality 3 --initial-
→moving-transform [ trans.mat, 1 ] --initialize-transforms-per-stage 0 --
→interpolation Linear --output [ output_, output_warped_image.nii.gz ] --
→transform Affine[ 2.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1, 32, Random,
→ 0.05 ] --convergence [ 1500x200, 1e-08, 20 ] --smoothing-sigmas 1.0x0.0vox --
→shrink-factors 2x1 --use-estimate-learning-rate-one 1 --use-histogram-matching,
→1 --masks [ NULL, NULL ] --transform Syn[ 0.25, 3.0, 0.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1, 32 ] --convergence [ 100x50x30, 1e-09, 20 ] --
→smoothing-sigmas 2.0x1.0x0.0vox --shrink-factors 3x2x1 --use-estimate-learning-
→
Here we use both a warpfield and a linear transformation, before registration commences. Note that the first transformation that needs to be applied (‘ants_Warp.nii.gz’) is last in the list of ‘initial_moving_transform’.

```python
>>> # Test initialization with multiple transforms matrices (e.g., un warp and
--> affine transform)
>>> reg10 = copy.deepcopy(reg)
>>> reg10.inputs.initial_moving_transform = ["func_to_struct.mat", 'ants_Warp.nii.
--> gz']
>>> reg10.inputs.invert_initial_moving_transform = [False, False]
```  
```
reg10.cmdline
'antsRegistration --colla peg-output-transforms 0 --dimensionality 3 --initial-
--> moving-transform [ func_to_struct.mat, 0 ] [ ants_Warp.nii.gz, 0 ] -- initial-
--> transforms-per-stage 0 --interpolation Linear --output [ output_, output_warped_ 
--> nii, 1, 32, Random, 0.05 ] --convergence [ 1500x200, 1e-08, 20 ] -- smoothing-
--> sigmas 1.0x0.0vox --shrink-factors 2x1 --use-estimate-learning-rate-once 1 -- 
--> use histogram-matching 1 --transform SyN[ 0.25, 3.0, 0.0 ] --metric Mattes[ 
--> fixed1.nii, moving1.nii, 1, 32 ] --convergence [ 100x50x30, 1e-09, 20 ] -- 
--> smoothing-sigmas 2.0x1.0x0.0vox --shrink-factors 3x2x1 --use-estimate-learning-
--> rate-once 1 --use histogram-matching 1 --winsorize-image-intensities [ 0.0, 1.0, 
--> ] --write-composite-transform 1'
```

Inputs:

[Optional]
**dimension:** (3 or 2, nipype default value: 3)

**fixed_image:** (a list of items which are a pathlike object or string 
representing an existing file)

*Image to which the moving_image should be transformed (usually a structural image)*

**moving_image:** (a list of items which are a pathlike object or string 
representing an existing file)

*Image that will be registered to the space of fixed_image. This is the image on which the transformations will be applied to*

**metric:** (a list of items which are 'CC' or 'MeanSquares' or 'Demons', 
or 'GC' or 'MI' or 'Mattes' or a list of items which are 'CC' or 'MeanSquares' or 'Demons' or 'GC' or 'MI' or 'Mattes')

*the metric(s) to use for each stage. Note that multiple metrics per stage are not supported in ANTS 1.9.1 and earlier.*

**metric_weight:** (a list of items which are a float or a list of items 
which are a float, nipype default value: [1.0])

*the metric weight(s) for each stage. The weights must sum to 1 per stage.*

**requires:** metric

**transforms:** (a list of items which are 'Rigid' or 'Affine' or 
'CompositeAffine' or 'Similarity' or 'Translation' or 'BSpline' or 
'GaussianDisplacementField' or 'TimeVaryingVelocityField' or 
'TimeVaryingBSplineVelocityField' or 'SyN' or 'BSplineSyN' or 
'Exponential' or 'BSplineExponential')

*argument: ‘%s’*

**smoothing_sigm a:** (a list of items which are a list of items which 
are a float)

**shrink_factors:** (a list of items which are a list of items which are 
an integer (int or long))
image dimension (2 or 3)
argument: `''--dimensionality %d''`

fixed_image_mask: (a pathlike object or string representing an existing file)
Mask used to limit metric sampling region of the fixed image in all stages
argument: `''%s''`
mutually_exclusive: fixed_image_masks

fixed_image_masks: (a list of items which are a pathlike object or string representing an existing file or 'NULL')
Masks used to limit metric sampling region of the fixed image, defined per registration stage (Use "NULL" to omit a mask at a given stage)
mutually_exclusive: fixed_image_mask

moving_image_mask: (a pathlike object or string representing an existing file)
mask used to limit metric sampling region of the moving image in all stages
mutually_exclusive: moving_image_masks
requires: fixed_image_mask

moving_image_masks: (a list of items which are a pathlike object or string representing an existing file or 'NULL')
Masks used to limit metric sampling region of the moving image, defined per registration stage (Use "NULL" to omit a mask at a given stage)
mutually_exclusive: moving_image_mask

save_state: (a pathlike object or string representing a file)
Filename for saving the internal restorable state of the registration
argument: `''--save-state %s''`

restore_state: (a pathlike object or string representing an existing file)
Filename for restoring the internal restorable state of the registration
argument: `''--restore-state %s''`

initial_moving_transform: (a list of items which are a pathlike object or string representing an existing file)
A transform or a list of transforms that should be applied before the registration begins. Note that, when a list is given, the transformations are applied in reverse order.
argument: `''%s''`
mutable_exclusive: initial_moving_transform_com

invert_initial_moving_transform: (a list of items which are a boolean)
One boolean or a list of booleans that indicate whether the inverse(s) of the transform(s) defined in initial_moving_transform should be used.
mutable_exclusive: initial_moving_transform_com
requires: initial_moving_transform

initial_moving_transform_com: (0 or 1 or 2)
Align the moving image and fixed image before registration using the geometric center of the images (=0), the image intensities (=1), or the origin of the images (=2).
argument: `''%s''`
mutable_exclusive: initial_moving_transform_com

metric_item_trait: ('CC' or 'MeanSquares' or 'Demons' or 'GC' or 'MI' or 'Mattes')
metric_stage_trait: ("CC" or "MeanSquares" or "Demons" or "GC" or "MI" or "Mattes" or a list of items which are "CC" or "MeanSquares" or "Demons" or "GC" or "MI" or "Mattes")

metric_weight_item_trait: (a float, nipype default value: 1.0)

metric_weight_stage_trait: (a float or a list of items which are a float)

radius_bins_item_trait: (an integer (int or long), nipype default value: 5)

radius_bins_stage_trait: (an integer (int or long) or a list of items which are an integer (int or long))

radius_or_number_of_bins: (a list of items which are an integer (int or long) or a list of items which are an integer (int or long), nipype default value: [5])

the number of bins in each stage for the MI and Mattes metric, the radius for other metrics

requires: metric_weight

sampling_strategy_item_trait: ("None" or "Regular" or "Random" or None)

sampling_strategy_stage_trait: ("None" or "Regular" or "Random" or None or a list of items which are "None" or "Regular" or "Random" or None)

sampling_strategy: (a list of items which are "None" or "Regular" or "Random" or None or a list of items which are "None" or "Regular" or "Random" or None)

the metric sampling strategy (strategies) for each stage

requires: metric_weight

sampling_percentage_item_trait: (0.0 <= a floating point number <= 1.0 or None)

sampling_percentage_stage_trait: (0.0 <= a floating point number <= 1.0 or None or a list of items which are 0.0 <= a floating point number <= 1.0 or None)

sampling_percentage: (a list of items which are 0.0 <= a floating point number <= 1.0 or None or a list of items which are 0.0 <= a floating point number <= 1.0 or None)

the metric sampling percentage(s) to use for each stage

requires: sampling_strategy

use_estimate_learning_rate_once: (a list of items which are a boolean)

use_histogram_matching: (a boolean or a list of items which are a boolean, nipype default value: True)

Histogram match the images before registration.

interpolation: ("Linear" or "NearestNeighbor" or "CosineWindowedSinc" or "WelchWindowedSinc" or "HammingWindowedSinc" or "LanczosWindowedSinc" or "Bspline" or "MultiLabel" or "Gaussian"), nipype default value: Linear)

argument: "--%s"

interpolation_parameters: (a tuple of the form: (an integer (int or long)) or a tuple of the form: (a float, a float))

write_composite_transform: (a boolean, nipype default value: False)

argument: "--write-composite-transform %d"

collapse_output_transforms: (a boolean, nipype default value: True)

Collapse output transforms. Specifically, enabling this option combines all adjacent linear transforms and composes all adjacent displacement field transforms before writing the results to disk.

argument: "--collapse-output-transforms %d"

initialize_transforms_per_stage: (a boolean, nipype default value: False)
Initialize linear transforms from the previous stage. By enabling this option, the current linear stage transform is directly initialized from the previous stages linear transform; this allows multiple linear stages to be run where each stage directly updates the estimated linear transform from the previous stage. (e.g. Translation -> Rigid -> Affine).

**argument:** `--initialize-transforms-per-stage %d`

**float:** (a boolean)
Use float instead of double for computations.
**argument:** `--float %d`

**transform_parameters:** (a list of items which are a tuple of the form:
(a float) or a tuple of the form: (a float, a float, a float) or a tuple of the form: (a float, an integer (int or long), an integer (int or long)) or a tuple of the form: (a float, an integer (int or long), a float, a float, a float) or a tuple of the form: (a float, a float, a float, an integer (int or long)) or a tuple of the form: (a float, an integer (int or long), an integer (int or long), an integer (int or long), an integer (int or long)))

**restrict_deformation:** (a list of items which are a list of items which are 0 or 1)
This option allows the user to restrict the optimization of the displacement field, translation, rigid or affine transform on a per-component basis. For example, if one wants to limit the deformation or rotation of 3-D volume to the first two dimensions, this is possible by specifying a weight vector of '1x1x0' for a deformation field or '1x1x0x1x1x0' for a rigid transformation. Low-dimensional restriction only works if there are no preceding transformations.

**number_of_iterations:** (a list of items which are a list of items which are an integer (int or long))

**sigma_units:** (a list of items which are 'mm' or 'vox')
units for smoothing sigmas

**convergence_threshold:** (a list of at least 1 items which are a float, nipype default value: [1e-06])

**convergence_window_size:** (a list of at least 1 items which are an integer (int or long), nipype default value: [10])

**output_transform_prefix:** (a unicode string, nipype default value: transform)
**argument:** `-%s`

**output_warped_image:** (a boolean or a pathlike object or string representing a file)

**output_inverse_warped_image:** (a boolean or a pathlike object or string representing a file)

**winsorize_upper_quantile:** (0.0 <= a floating point number <= 1.0, nipype default value: 1.0)
The Upper quantile to clip image ranges
**argument:** `-%s`

**winsorize_lower_quantile:** (0.0 <= a floating point number <= 1.0, nipype default value: 0.0)
The Lower quantile to clip image ranges
**argument:** `-%s`

**verbose:** (a boolean, nipype default value: False)
**argument:** `-%v`
**num_threads:** (an integer (int or long), nipype default value: 1)
   Number of ITK threads to use

**args:** (a unicode string)
   Additional parameters to the command
   argument: `'\%s'`

**environ:** (a dictionary with keys which are a bytes or None or a value
   of class 'str' and with values which are a bytes or None or a
   value of class 'str', nipype default value: {})
   Environment variables

#### Outputs:

- **forward_transforms:** (a list of items which are a pathlike object or
  string representing an existing file)
  List of output transforms for forward registration

- **reverse_transforms:** (a list of items which are a pathlike object or
  string representing an existing file)
  List of output transforms for reverse registration

- **forward_invert_flags:** (a list of items which are a boolean)
  List of flags corresponding to the forward transforms

- **reverse_invert_flags:** (a list of items which are a boolean)
  List of flags corresponding to the reverse transforms

- **composite_transform:** (a pathlike object or string representing an
  existing file)
  Composite transform file

- **inverse_composite_transform:** (a pathlike object or string representing a file)
  Inverse composite transform file

- **warped_image:** (a pathlike object or string representing a file)
  Outputs warped image

- **inverse_warped_image:** (a pathlike object or string representing a file)
  Outputs the inverse of the warped image

- **save_state:** (a pathlike object or string representing a file)
  The saved registration state to be restored

- **metric_value:** (a float)
  the final value of metric

- **elapsed_time:** (a float)
  the total elapsed time as reported by ANTs

**52.2.4 RegistrationSynQuick**

**Link to code**

Wraps the executable command `antsRegistrationSyNQuick.sh`.


**Examples**

```python
>>> from nipype.interfaces.ants import RegistrationSynQuick
>>> reg = RegistrationSynQuick()
>>> reg.inputs.fixed_image = 'fixed1.nii'
>>> reg.inputs.moving_image = 'moving1.nii'
>>> reg.inputs.num_threads = 2
>>> reg.cmdline
```

(continues on next page)
'antsRegistrationSyNQuick.sh -d 3 -f fixed1.nii -r 32 -m moving1.nii -n 2 -o transform -p d -s 26 -t s'

example for multiple images

```python
from nipype.interfaces.ants import RegistrationSyNQuick

reg = RegistrationSyNQuick()
reg.inputs.fixed_image = ['fixed1.nii', 'fixed2.nii']
reg.inputs.moving_image = ['moving1.nii', 'moving2.nii']
reg.inputs.num_threads = 2
reg.cmdline

# doctest: +SKIP
```

Inputs:

- **fixed_image**: (a list of items which are a pathlike object or string representing an existing file)
  - Fixed image or source image or reference image
  - Argument: `-f %s...`

- **moving_image**: (a list of items which are a pathlike object or string representing an existing file)
  - Moving image or target image
  - Argument: `-m %s...`

- **dimension**: (3 or 2, nipype default value: 3)
  - Image dimension (2 or 3)
  - Argument: `-d %d`

- **output_prefix**: (a unicode string, nipype default value: transform)
  - A prefix that is prepended to all output files
  - Argument: `-o %s`

- **num_threads**: (an integer (int or long), nipype default value: 1)
  - Number of threads (default = 1)
  - Argument: `-n %d`

- **transform_type**: ('s' or 't' or 'r' or 'a' or 'sr' or 'b' or 'br', nipype default value: s)
  - Transform type
  - t: translation
  - r: rigid
  - a: rigid + affine
  - s: rigid + affine + deformable syn (default)
  - sr: rigid + deformable syn
  - b: rigid + affine + deformable b-spline syn
  - br: rigid + deformable b-spline syn
  - Argument: `-t %s`

- **use_histogram_matching**: (a boolean)
  - Use histogram matching
  - Argument: `-j %d`

- **histogram_bins**: (an integer (int or long), nipype default value: 32)
  - Histogram bins for mutual information in SyN stage (default = 32)
  - Argument: `-r %d`

- **spline_distance**: (an integer (int or long), nipype default value: 26)
  - Spline distance for deformable B-spline SyN transform (default = 26)
  - Argument: `-s %d`
precision_type: ('double' or 'float', nipype default value: double)
  precision type (default = double)
  argument: `'-p %s'`
args: (a unicode string)
  Additional parameters to the command
  argument: `'%s'`
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
  Environment variables

Outputs:

warped_image: (a pathlike object or string representing an existing
  file)
Warped image
inverse_warped_image: (a pathlike object or string representing an
  existing file)
Inverse warped image
out_matrix: (a pathlike object or string representing an existing
  file)
Affine matrix
forward_warp_field: (a pathlike object or string representing an
  existing file)
Forward warp field
inverse_warp_field: (a pathlike object or string representing an
  existing file)
Inverse warp field

52.3 interfaces.ants.resampling

52.3.1 ApplyTransforms

Link to code
Wraps the executable command antsApplyTransforms.
ApplyTransforms, applied to an input image, transforms it according to a reference image and a transform (or a set of transforms).

Examples

```python
>>> from nipype.interfaces.ants import ApplyTransforms
>>> at = ApplyTransforms()
>>> at.inputs.input_image = 'moving1.nii'
>>> at.inputs.reference_image = 'fixed1.nii'
>>> at.inputs.transforms = 'identity'
>>> at.cmdline
'antsApplyTransforms --default-value 0 --float 0 --input moving1.nii --
  --interpolation Linear --output moving1_trans.nii --reference-image fixed1.nii -t
  --identity'
```

```python
>>> at = ApplyTransforms()
>>> at.inputs.dimension = 3
>>> at.inputs.input_image = 'moving1.nii'
>>> at.inputs.reference_image = 'fixed1.nii'
```
at.inputs.output_image = 'deformed_moving1.nii'
at.inputs.interpolation = 'Linear'
at.inputs.default_value = 0
at.inputs.transforms = ['ants_Warp.nii.gz', 'trans.mat']
at.inputs.invert_transform_flags = [False, False]
at.cmdline
'antsApplyTransforms --default-value 0 --dimensionality 3 --float 0 --input moving1.nii --interpolation Linear --output deformed_moving1.nii --reference-image fixed1.nii --transform [ ants_Warp.nii.gz, 0 ] --transform [ trans.mat, 0 ]'

at1 = ApplyTransforms()
at1.inputs.dimension = 3
at1.inputs.input_image = 'moving1.nii'
at1.inputs.reference_image = 'fixed1.nii'
at1.inputs.output_image = 'deformed_moving1.nii'
at1.inputs.interpolation = 'BSpline'
at1.inputs.interpolation_parameters = (5,)
at1.inputs.default_value = 0
at1.inputs.transforms = ['ants_Warp.nii.gz', 'trans.mat']
at1.inputs.invert_transform_flags = [False, False]
at1.cmdline
'antsApplyTransforms --default-value 0 --dimensionality 3 --float 0 --input moving1.nii --interpolation BSpline[ 5 ] --output deformed_moving1.nii --reference-image fixed1.nii --transform [ ants_Warp.nii.gz, 0 ] --transform [ trans.mat, 0 ]'

Inputs:

[Mandatory]

input_image: (a pathlike object or string representing an existing file)
   image to apply transformation to (generally a coregistered functional)
   argument: `''--input %s''`
reference_image: (a pathlike object or string representing an existing file)
   reference image space that you wish to warp INTO
   argument: `''--reference-image %s''`
transforms: (a list of items which are a pathlike object or string representing an existing file or 'identity')
   transform files: will be applied in reverse order. For example, the last specified transform will be applied first.
   argument: `''%s''`

[Optional]

dimension: (2 or 3 or 4)
   This option forces the image to be treated as a specified-dimensional image. If not specified, antsWarp tries to infer the dimensionality from the input image.
   argument: `''--dimensionality %d''`
input_image_type: (0 or 1 or 2 or 3)
   Option specifying the input image type of scalar (default), vector, tensor, or time series.
   argument: `''--input-image-type %d''`
output_image: (a unicode string)
   output file name
(continues on next page)
52.3.2 ApplyTransformsToPoints

Link to code

Wraps the executable command `antsApplyTransformsToPoints`. ApplyTransformsToPoints, applied to an CSV file, transforms coordinates using provided transform (or a set of transforms).

Examples

```python
g from nipype.interfaces.ants import ApplyTransforms
g at = ApplyTransformsToPoints()
g at.inputs.dimension = 3
g at.inputs.input_file = 'moving.csv'
g at.inputs.transforms = ['trans.mat', 'ants_Warp.nii.gz']
g at.inputs.invert_transform_flags = [False, False]
g at.cmdline
'antsApplyTransformsToPoints --dimensionality 3 --input moving.csv --output
  moving_transformed.csv --transform [ trans.mat, 0 ] --transform [ ants_Warp.nii.
  gz, 0 ]'
```

Inputs:
input_file: (a pathlike object or string representing an existing file)
Currently, the only input supported is a csv file with columns including x,y (2D), x,y,z (3D) or x,y,z,t,label (4D) column headers. The points should be defined in physical space. If in doubt how to convert coordinates from your files to the space required by antsApplyTransformsToPoints try creating/drawing a simple label volume with only one voxel set to 1 and all others set to 0. Write down the voxel coordinates. Then use ImageMaths LabelStats to find out what coordinates for this voxel antsApplyTransformsToPoints is expecting.

transforms: (a list of items which are a pathlike object or string representing an existing file)
transforms that will be applied to the points

dimension: (2 or 3 or 4)
This option forces the image to be treated as a specified-dimensional image. If not specified, antsWarp tries to infer the dimensionality from the input image.

output_file: (a unicode string)
Name of the output CSV file

invert_transform_flags: (a list of items which are a boolean)
list indicating if a transform should be reversed

num_threads: (an integer (int or long), nipype default value: 1)
Number of ITK threads to use

args: (a unicode string)
Additional parameters to the command

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

output_file: (a pathlike object or string representing an existing file)
csv file with transformed coordinates

52.3.3 WarpImageMultiTransform

Link to code
Wraps the executable command WarpImageMultiTransform. Warps an image from one space to another

Examples

```python
>>> from nipype.interfaces.ants import WarpImageMultiTransform
>>> wint = WarpImageMultiTransform()
>>> wint.inputs.input_image = 'structural.nii'
```
>>> wimt.inputs.reference_image = 'ants_deformed.nii.gz'
>>> wimt.inputs.transformation_series = ['ants_Warp.nii.gz','ants_Affine.txt']
>>> wimt.cmdline
'WarpImageMultiTransform 3 structural.nii structural_wimt.nii -R ants_deformed.nii.gz ants_Warp.nii.gz ants_Affine.txt'

>>> wimt = WarpImageMultiTransform()
>>> wimt.inputs.input_image = 'diffusion_weighted.nii'
>>> wimt.inputs.reference_image = 'functional.nii'
>>> wimt.inputs.transformation_series = ['func2anat_coreg_Affine.txt','func2anat_InverseWarp.nii.gz', 'dwi2anat_Warp.nii.gz','dwi2anat_coreg_Affine.txt']
>>> wimt.inputs.invert_affine = [1]  # this will invert the 1st Affine file:
   ->'func2anat_coreg_Affine.txt'
>>> wimt.cmdline
'WarpImageMultiTransform 3 diffusion_weighted.nii diffusion_weighted_wimt.nii -R functional.nii -i func2anat_coreg_Affine.txt func2anat_InverseWarp.nii.gz dwi2anat_Warp.nii.gz dwi2anat_coreg_Affine.txt'

Inputs:

[Mandatory]

input_image: (a pathlike object or string representing a file)
   image to apply transformation to (generally a coregistered functional)
   argument: `'\$s'`, position: 2

transformation_series: (a list of items which are a pathlike object or string representing an existing file)
   transformation file(s) to be applied
   argument: `'\$s'`, position: -1

[Optional]

dimension: (3 or 2, nipype default value: 3)
   image dimension (2 or 3)
   argument: `'\$d'`, position: 1

output_image: (a pathlike object or string representing a file)
   name of the output warped image
   argument: `'\$s'`, position: 3

mutually_exclusive: out_postfix
   Postfix that is prepended to all output files (default = _wimt)

out_postfix: (a pathlike object or string representing a file, nipype default value: _wimt)
   Postfix that is prepended to all output files (default = _wimt)

reference_image: (a pathlike object or string representing a file)
   reference image space that you wish to warp INTO
   argument: `'-R %s`

mutually_exclusive: tightest_box

tightest_box: (a boolean)
   computes tightest bounding box (overrided by reference_image if given)
   argument: `''--tightest-bounding-box''`

mutually_exclusive: reslice_by_header

reslice_by_header: (a boolean)
   Uses orientation matrix and origin encoded in reference image file header. Not typically used with additional transforms
   argument: `''--reslice-by-header''`

use_nearest: (a boolean)
   Use nearest neighbor interpolation
argument: `--use-NN`

use_bspline: (a boolean)
Use 3rd order B-Spline interpolation
argument: `--use-Bspline`

invert_affine: (a list of items which are an integer (int or long))
List of Affine transformations to invert. E.g.: [1,4,5] inverts the
1st, 4th, and 5th Affines found in transformation_series. Note that
indexing starts with 1 and does not include warp fields. Affine
transformations are distinguished from warp fields by the word
"affine" included in their filenames.

num_threads: (an integer (int or long), nipype default value: 1)
Number of ITK threads to use

args: (a unicode string)
Additional parameters to the command
argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a
value of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

output_image: (a pathlike object or string representing an existing
file)
Warped image

52.3.4 WarpTimeSeriesImageMultiTransform

Link to code
Wraps the executable command WarpTimeSeriesImageMultiTransform.
Warp a time-series from one space to another

Examples

```python
>>> from nipype.interfaces.ants import WarpTimeSeriesImageMultiTransform
>>> wtsimt = WarpTimeSeriesImageMultiTransform()
>>> wtsimt.inputs.input_image = 'resting.nii'
>>> wtsimt.inputs.reference_image = 'ants_deformed.nii.gz'
>>> wtsimt.inputs.transformation_series = ['ants_Warp.nii.gz', 'ants_Affine.txt']
>>> wtsimt.cmdline
'WarpTimeSeriesImageMultiTransform 4 resting.nii resting_wtsimt.nii -R ants_
→deformed.nii.gz ants_Warp.nii.gz ants_Affine.txt'
```

```python
>>> wtsimt = WarpTimeSeriesImageMultiTransform()
>>> wtsimt.inputs.input_image = 'resting.nii'
>>> wtsimt.inputs.reference_image = 'ants_deformed.nii.gz'
>>> wtsimt.inputs.transformation_series = ['ants_Warp.nii.gz', 'ants_Affine.txt']
>>> wtsimt.inputs.invert_affine = [1]  # # this will invert the 1st Affine file:
→ants_Affine.txt
>>> wtsimt.cmdline
'WarpTimeSeriesImageMultiTransform 4 resting.nii resting_wtsimt.nii -R ants_
→deformed.nii.gz ants_Warp.nii.gz -i ants_Affine.txt'
```

Inputs:
**Mandatory**

- **input_image**: (a pathlike object or string representing a file)
  - image to apply transformation to (generally a coregistered functional)
  - argument: `%s`

- **transformation_series**: (a list of items which are a pathlike object or string representing an existing file)
  - transformation file(s) to be applied
  - argument: `%s`

**Optional**

- **dimension**: (4 or 3, nipype default value: 4)
  - image dimension (3 or 4)
  - argument: `%d`, position: 1

- **out_postfix**: (a unicode string, nipype default value: _wtsimt)
  - Postfix that is prepended to all output files (default = _wtsimt)
  - argument: `%s`

- **reference_image**: (a pathlike object or string representing a file)
  - reference image space that you wish to warp INTO
  - argument: `'-R %s'`
  - mutually_exclusive: tightest_box

- **tightest_box**: (a boolean)
  - computes tightest bounding box (overrided by reference_image if given)
  - argument: `''--tightest-bounding-box''`
  - mutually_exclusive: reference_image

- **reslice_by_header**: (a boolean)
  - Uses orientation matrix and origin encoded in reference image file header. Not typically used with additional transforms
  - argument: `''--reslice-by-header''`

- **use_nearest**: (a boolean)
  - Use nearest neighbor interpolation
  - argument: `''--use-NN''`

- **use_bspline**: (a boolean)
  - Use 3rd order B-Spline interpolation
  - argument: `''--use-Bspline''`

- **invert_affine**: (a list of items which are an integer (int or long))
  - List of Affine transformations to invert. E.g.: [1,4,5] inverts the 1st, 4th, and 5th Affines found in transformation_series. Note that indexing starts with 1 and does not include warp fields. Affine transformations are distinguished from warp fields by the word "affine" included in their filenames.

- **num_threads**: (an integer (int or long), nipype default value: 1)
  - Number of ITK threads to use

- **args**: (a unicode string)
  - Additional parameters to the command
  - argument: `%s`

- **environ**: (a dictionary with keys which are a bytes or None or a value of class 'str', and with values which are a bytes or None or a value of class 'str', nipype default value: {})

**Outputs**

- **output_image**: (a pathlike object or string representing an existing file)
  - Warped image
52.4 interfaces.ants.segmentation

52.4.1 AntsJointFusion

Link to code
Wraps the executable command `antsJointFusion`.

Examples

```python
>>> from nipype.interfaces.ants import AntsJointFusion
>>> antsJointFusion = AntsJointFusion()
>>> antsJointFusion.inputs.out_label_fusion = 'ants_fusion_label_output.nii'
>>> antsJointFusion.inputs.atlas_image = ['rc1s1.nii', 'rc1s2.nii']
>>> antsJointFusion.inputs.atlas_segmentation_image = ['segmentation0.nii.gz']
>>> antsJointFusion.inputs.target_image = ['im1.nii']
>>> antsJointFusion.cmdline
"antsJointFusion -a 0.1 -g ['rc1s1.nii', 'rc1s2.nii'] -l segmentation0.nii.gz -b 2.0 -o ants_fusion_label_output.nii -s 3x3x3 -t ['im1.nii']"

>>> antsJointFusion.inputs.target_image = ['im1.nii', 'im2.nii']
>>> antsJointFusion.cmdline
"antsJointFusion -a 0.1 -g ['rc1s1.nii', 'rc1s2.nii'] -l segmentation0.nii.gz -b 2.0 -o ants_fusion_label_output.nii -s 3x3x3 -t ['im1.nii', 'im2.nii']"

>>> antsJointFusion.inputs.atlas_image = ['rc1s1.nii', 'rc1s2.nii', 'rc2s1.nii', 'rc2s2.nii']
>>> antsJointFusion.inputs.atlas_segmentation_image = ['segmentation0.nii.gz', 'segmentation1.nii.gz']
>>> antsJointFusion.cmdline
"antsJointFusion -a 0.1 -g ['rc1s1.nii', 'rc1s2.nii'] -g ['rc2s1.nii', 'rc2s2.nii'] -l segmentation0.nii.gz -l segmentation1.nii.gz -b 2.0 -o ants_fusion_label_output.nii -s 3x3x3 -t ['im1.nii', 'im2.nii']"

>>> antsJointFusion.inputs.dimension = 3
>>> antsJointFusion.inputs.alpha = 0.5
>>> antsJointFusion.inputs.beta = 1.0
>>> antsJointFusion.inputs.patch_radius = [3, 2, 1]
>>> antsJointFusion.inputs.search_radius = [3]
>>> antsJointFusion.cmdline
"antsJointFusion -a 0.5 -g ['rc1s1.nii', 'rc1s2.nii'] -g ['rc2s1.nii', 'rc2s2.nii'] -l segmentation0.nii.gz -l segmentation1.nii.gz -b 1.0 -d 3 -o ants_fusion_label_output.nii -p 3x2x1 -s 3 -t ['im1.nii', 'im2.nii']"

>>> antsJointFusion.inputs.search_radius = ['mask.nii']
>>> antsJointFusion.inputs.verbose = True
>>> antsJointFusion.inputs.exclusion_image = ['roi01.nii', 'roi02.nii']
>>> antsJointFusion.inputs.exclusion_image_label = ['1', '2']
>>> antsJointFusion.cmdline
"antsJointFusion -a 0.5 -g ['rc1s1.nii', 'rc1s2.nii'] -g ['rc2s1.nii', 'rc2s2.nii'] -l segmentation0.nii.gz -l segmentation1.nii.gz -b 1.0 -d 3 -e 1[roi01.nii] -e 2[roi02.nii] -o ants_fusion_label_output.nii -p 3x2x1 -s mask.nii -t ['im1.nii', 'im2.nii'] -v"

>>> antsJointFusion.inputs.out_label_fusion = 'ants_fusion_label_output.nii'
>>> antsJointFusion.inputs.out_intensity_fusion_name_format = 'ants_joint_fusion_intensity_%d.nii.gz'
```
(continues on next page)
>>> antsjointfusion.inputs.out_label_post_prob_name_format = 'ants_joint_fusion_
˓→posterior_%d.nii.gz'

>>> antsjointfusion.inputs.out_atlas_voting_weight_name_format = 'ants_joint_
˓→fusion_voting_weight_%d.nii.gz'

>>> antsjointfusion.cmdline
"antsJointFusion -a 0.5 -g ['rc1s1.nii', 'rc1s2.nii'] -g ['rc2s1.nii', 'rc2s2.nii 
˓→'] -l segmentation0.nii.gz -l segmentation1.nii.gz -b 1.0 -d 3 -e 1[roi01.nii] - 
˓→e 2[roi02.nii] -o [ants_fusion_label_output.nii, ants_joint_fusion_intensity_
˓→%d.nii.gz, ants_joint_fusion_posterior_%d.nii.gz, ants_joint_fusion_voting_
˓→weight_%d.nii.gz] -p 3x2x1 -s mask.nii -t ['im1.nii', 'im2.nii'] -v"

Inputs:

[Mandatory]

target_image: (a list of items which are a list of items which are a
pathlike object or string representing an existing file)
The target image (or multimodal target images) assumed to be aligned
common image domain.
argument: `--t %s`

atlas_image: (a list of items which are a list of items which are a
pathlike object or string representing an existing file)
The atlas image (or multimodal atlas images) assumed to be aligned
to a common image domain.
argument: `--g %s...`

atlas_segmentation_image: (a list of items which are a pathlike
object or string representing an existing file)
The atlas segmentation images. For performing label fusion the
number of specified segmentations should be identical to the number
of atlas image sets.
argument: `--l %s...`

[Optional]

dimension: (3 or 2 or 4)
This option forces the image to be treated as a specified-
dimensional image. If not specified, the program tries to infer the
dimensionality from the input image.
argument: `--d %d`

alpha: (a float, nipype default value: 0.1)
Regularization term added to matrix Mx for calculating the inverse.
Default = 0.1
argument: `--a %s`

beta: (a float, nipype default value: 2.0)
Exponent for mapping intensity difference to the joint error.
Default = 2.0
argument: `--b %s`

retain_label_posterior_images: (a boolean, nipype default value:
False)
Retain label posterior probability images. Requires atlas
segmentations to be specified. Default = false
argument: `--r`

retain_atlas_voting_images: (a boolean, nipype default value: False)
Retain atlas voting images. Default = false
argument: `--f`

constrain_nonnegative: (a boolean, nipype default value: False)
Constrain solution to non-negative weights.
argument: `--c`

(continues on next page)
patch_radius: (a list of items which are a value of class 'int')
  Patch radius for similarity measures. Default: 2x2x2
  argument: `\'-p %s`'
patch_metric: (\'PC\' or \'MSQ\')
  Metric to be used in determining the most similar neighborhood
  patch. Options include Pearson's correlation (PC) and mean squares
  (MSQ). Default = PC (Pearson correlation).
  argument: `\'-m %s`'
search_radius: (a list of from 1 to 3 items which are any value,
  nipype default value: [3, 3, 3])
  Search radius for similarity measures. Default = 3x3x3. One can also
  specify an image where the value at the voxel specifies the
  isotropic search radius at that voxel.
  argument: `\'-s %s`'
exclusion_image_label: (a list of items which are a unicode string)
  Specify a label for the exclusion region.
  argument: `\'-e %s`'
  requires: exclusion_image
exclusion_image: (a list of items which are a pathlike object or
  string representing an existing file)
  Specify an exclusion region for the given label.
mask_image: (a pathlike object or string representing an existing
  file)
  If a mask image is specified, fusion is only performed in the mask
  region.
  argument: `\'-x %s`'
out_label_fusion: (a pathlike object or string representing a file)
  The output label fusion image.
  argument: `\'%s`'
out_intensity_fusion_name_format: (a unicode string)
  Optional intensity fusion image file name format. (e.g.
  "antsJointFusionIntensity_%d.nii.gz")
out_label_post_prob_name_format: (a unicode string)
  Optional label posterior probability image file name format.
  requires: out_label_fusion, out_intensity_fusion_name_format
out_atlas_voting_weight_name_format: (a unicode string)
  Optional atlas voting weight image file name format.
  requires: out_label_fusion, out_intensity_fusion_name_format,
  out_label_post_prob_name_format
verbose: (a boolean)
  Verbose output.
  argument: `\'-v`'
num_threads: (an integer (int or long), nipype default value: 1)
  Number of ITK threads to use
args: (a unicode string)
  Additional parameters to the command
  argument: `\'%s`'
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
  Environment variables

Outputs:

out_label_fusion: (a pathlike object or string representing an
  existing file)
out_intensity_fusion_name_format: (a unicode string)
52.4.2 Atropos

Link to code
Wraps the executable command Atropos.
A finite mixture modeling (FMM) segmentation approach with possibilities for specifying prior constraints.
These prior constraints include the specification of a prior label image, prior probability images (one for each class), and/or an MRF prior to enforce spatial smoothing of the labels. Similar algorithms include FAST and SPM.

Examples

```python
>>> from nipype.interfaces.ants import Atropos
>>> at = Atropos()
>>> at.inputs.dimension = 3
>>> at.inputs.intensity_images = 'structural.nii'
>>> at.inputs.mask_image = 'mask.nii'
>>> at.inputs.initialization = 'PriorProbabilityImages'
>>> at.inputs.prior_probability_images = ['rc1s1.nii', 'rc1s2.nii']
>>> at.inputs.number_of_tissue_classes = 2
>>> at.inputs.prior_weighting = 0.8
>>> at.inputs.prior_probability_threshold = 0.000001
>>> at.inputs.likelihood_model = 'Gaussian'
>>> at.inputs.mrf_smoothing_factor = 0.2
>>> at.inputs.mrf_radius = [1, 1, 1]
>>> at.inputs.icm_use_synchronous_update = True
>>> at.inputs.maximum_number_of_icm_iterations = 1
>>> at.inputs.n_iterations = 5
>>> at.inputs.convergence_threshold = 0.000001
>>> at.inputs.posterior_formulation = 'Socrates'
>>> at.inputs.use_mixture_model_proportions = True
>>> at.inputs.save_posteriors = True
>>> at.cmdline
```

Inputs:

- **intensity_images**: (a list of items which are a pathlike object or string representing an existing file)
  - argument: `''--intensity-image %s...''`
- **mask_image**: (a pathlike object or string representing an existing file)
  - argument: `''--mask-image %s''`
- **initialization**: ('Random' or 'Otsu' or 'KMeans' or 'PriorProbabilityImages' or 'PriorLabelImage')
  - argument: `''%s''`
  - requires: number_of_tissue_classes
- **number_of_tissue_classes**: (an integer (int or long))
52.4.3 BrainExtraction

Link to code
Wraps the executable command `antsBrainExtraction.sh`.

**Examples**

```python
>>> from nipype.interfaces.ants.segmentation import BrainExtraction
>>> brainextraction = BrainExtraction()
>>> brainextraction.inputs.dimension = 3
>>> brainextraction.inputs.anatomical_image = 'T1.nii.gz'
>>> brainextraction.inputs.brain_template = 'study_template.nii.gz'
>>> brainextraction.inputs.brain_probability_mask =
   'ProbabilityMaskOfStudyTemplate.nii.gz'
>>> brainextraction.cmdline
'antsBrainExtraction.sh -a T1.nii.gz -m ProbabilityMaskOfStudyTemplate.nii.gz -e study_template.nii.gz -d 3 -s nii.gz -o highres001_'
```

**Inputs:**

- **anatomical_image:** (a pathlike object or string representing an existing file)
  - Structural image, typically T1. If more than one anatomical image is specified, subsequently specified images are used during the segmentation process. However, only the first image is used in the registration of priors. Our suggestion would be to specify the T1 as the first image. Anatomical template created using e.g. LPBA40 data set with `buildtemplateparallel.sh` in ANTs.
  - argument: `'-a %s'`

- **brain_template:** (a pathlike object or string representing an existing file)
  - Anatomical template created using e.g. LPBA40 data set with `buildtemplateparallel.sh` in ANTs.
  - argument: `'-e %s'`

- **brain_probability_mask:** (a pathlike object or string representing an existing file)
  - Brain probability mask created using e.g. LPBA40 data set which have brain masks defined, and warped to anatomical template and averaged resulting in a probability image.
  - argument: `'-m %s'`

- **dimension:** (3 or 2, nipype default value: 3)
  - image dimension (2 or 3)
  - argument: `'-d %d'`

- **out_prefix:** (a unicode string, nipype default value: `highres001_`)
  - Prefix that is prepended to all output files (default = `highres001_`)
  - argument: `'-o %s'`

- **extraction_registration_mask:** (a pathlike object or string representing an existing file)
  - Mask (defined in the template space) used during registration for brain extraction. To limit the metric computation to a specific region.
  - argument: `'-f %s'`

- **image_suffix:** (a unicode string, nipype default value: `nii.gz`)
  - any of standard ITK formats, `nii.gz` is default
  - argument: `'-s %s'`

- **use_random_seeding:** (0 or 1)
  - Use random number generated from system clock in Atropos (default = 52.4. interfaces.ants.segmentation 579)
1) argument: `"-u %d`'  
keep_temporary_files: (an integer (int or long))  
Keep brain extraction/segmentation warps, etc (default = 0).  
argument: `"-k %d`'  
use_floatingpoint_precision: (0 or 1)  
Use floating point precision in registrations (default = 0)  
argument: `"-q %d`'  

debug: (a boolean)  
If > 0, runs a faster version of the script. Only for testing.  
Implies -u 0. Requires single thread computation for complete  
reproducibility.  
argument: `"-z %d`'  
um_threads: (an integer (int or long), nipype default value: 1)  
Number of ITK threads to use  
args: (a unicode string)  
Additional parameters to the command  
argument: `"%s`'  
environ: (a dictionary with keys which are a bytes or None or a value  
of class 'str' and with values which are a bytes or None or a  
value of class 'str', nipype default value: {})  
Environment variables

Outputs:

BrainExtractionMask: (a pathlike object or string representing an  
existing file)  
brain extraction mask  
BrainExtractionBrain: (a pathlike object or string representing an  
existing file)  
brain extraction image  
BrainExtractionCSF: (a pathlike object or string representing an  
existing file)  
segmentation mask with only CSF  
BrainExtractionGM: (a pathlike object or string representing an  
existing file)  
segmentation mask with only grey matter  
BrainExtractionInitialAffine: (a pathlike object or string  
representing an existing file)  
BrainExtractionInitialAffineFixed: (a pathlike object or string  
representing an existing file)  
BrainExtractionInitialAffineMoving: (a pathlike object or string  
representing an existing file)  
BrainExtractionLaplacian: (a pathlike object or string representing an  
existing file)  
BrainExtractionPrior0GenericAffine: (a pathlike object or string  
representing an existing file)  
BrainExtractionPrior1InverseWarp: (a pathlike object or string  
representing an existing file)  
BrainExtractionPrior1Warp: (a pathlike object or string representing an  
existing file)  
BrainExtractionPriorWarped: (a pathlike object or string representing an  
existing file)  
BrainExtractionSegmentation: (a pathlike object or string  
representing an existing file)  
segmentation mask with CSF, GM, and WM  
BrainExtractionTemplateLaplacian: (a pathlike object or string  
representing an existing file)
representing an existing file)

BrainExtractionTmp: (a pathlike object or string representing an existing file)

BrainExtractionWM: (a pathlike object or string representing an existing file)

- segmentation mask with only white matter

N4Corrected0: (a pathlike object or string representing an existing file)

- N4 bias field corrected image

N4Truncated0: (a pathlike object or string representing an existing file)

52.4.4 CorticalThickness

Link to code

Wraps the executable command antsCorticalThickness.sh.

Examples

```python
>>> from nipype.interfaces.ants.segmentation import CorticalThickness

>>> corticalthickness = CorticalThickness()

>>> corticalthickness.inputs.dimension = 3

>>> corticalthickness.inputs.anatomical_image = 'T1.nii.gz'

>>> corticalthickness.inputs.brain_template = 'study_template.nii.gz'

>>> corticalthickness.inputs.brain_probability_mask = 'ProbabilityMaskOfStudyTemplate.nii.gz'

>>> corticalthickness.inputs.segmentation_priors =['BrainSegmentationPrior01.nii.gz', ...

... 'BrainSegmentationPrior02.nii.gz', ...

... 'BrainSegmentationPrior03.nii.gz', ...

... 'BrainSegmentationPrior04.nii.gz']

>>> corticalthickness.inputs.t1_registration_template = 'brain_study_template.nii.gz'

>>> corticalthickness.cmdline

'antsCorticalThickness.sh -a T1.nii.gz -m ProbabilityMaskOfStudyTemplate.nii.gz -e study_template.nii.gz -d 3 -s nii.gz -o antsCT -p nipype_priors/BrainSegmentationPrior%02d.nii.gz -t brain_study_template.nii.gz'
```

Inputs:

- [Mandatory]
  - anatomical_image: (a pathlike object or string representing an existing file)
    - Structural *intensity* image, typically T1. If more than one anatomical image is specified, subsequently specified images are used during the segmentation process. However, only the first image is used in the registration of priors. Our suggestion would be to specify the T1 as the first image.
    - argument: `'-%s'``
  - brain_template: (a pathlike object or string representing an existing file)
    - Anatomical *intensity* template (possibly created using a population data set with buildtemplateparallel.sh in ANTs). This template is
      (continues on next page)
*not* skull-stripped.

```
argument: ``-e %s``
```

brain_probability_mask: (a pathlike object or string representing an 
existing file)

```
brain probability mask in template space
argument: ``-m %s``
```

segmentation_priors: (a list of items which are a pathlike object or 
string representing an existing file)

```
argument: ``-p %s``
```

t1_registration_template: (a pathlike object or string representing 
an existing file)

Anatomical *intensity* template (assumed to be skull-stripped). A 
common case would be where this would be the same template as 
specified in the -e option which is not skull stripped.

```
argument: ``-t %s``
```

[Optional]

dimension: (3 or 2, nipype default value: 3)

```
image dimension (2 or 3)
argument: ``-d %d``
```

out_prefix: (a unicode string, nipype default value: antsCT_)

```
Prefix that is prepended to all output files (default = antsCT_)
argument: ``-o %s``
```

image_suffix: (a unicode string, nipype default value: nii.gz)

```
any of standard ITK formats, nii.gz is default
argument: ``-s %s``
```

extraction_registration_mask: (a pathlike object or string 
representing an existing file)

```
Mask (defined in the template space) used during registration for 
brain extraction.
argument: ``-f %s``
```

keep_temporary_files: (an integer (int or long))

```
Keep brain extraction/segmentation warps, etc (default = 0).
argument: ``-k %d``
```

max_iterations: (an integer (int or long))

```
ANTS registration max iterations (default = 100x100x70x20)
argument: ``-i %d``
```

prior_segmentation_weight: (a float)

```
Atropos spatial prior *probability* weight for the segmentation
argument: ``-w %f``
```

segmentation_iterations: (an integer (int or long))

```
N4 -> Atropos -> N4 iterations during segmentation (default = 3)
argument: ``-n %d``
```

posterior_formulation: (a unicode string)

```
Atropos posterior formulation and whether or not to use mixture 
model proportions. e.g 'Socrates[1]' (default) or 'Aristotle[1]'.
Choose the latter if you want use the distance priors (see also the 
-l option for label propagation control).
argument: ``-b %s``
```

use_floatingpoint_precision: (0 or 1)

```
Use floating point precision in registrations (default = 0)
argument: ``-j %d``
```

use_random_seeding: (0 or 1)

```
Use random number generated from system clock in Atropos (default = 1)
argument: ``-u %d``
```

b_spline_smoothing: (a boolean)
Use B-spline SyN for registrations and B-spline exponential mapping in DiReCT.

**Argument:** `-v`

**cortical_label_image:** (a pathlike object or string representing an existing file)
Cortical ROI labels to use as a prior for ATITH.

**label_propagation:** (a unicode string)
Incorporate a distance prior one the posterior formulation. Should be of the form 'label[lambda, boundaryProbability]' where label is a value of 1, 2, 3, ..., denoting label ID. The label probability for anything outside the current label = boundaryProbability \* exp(-lambda \* distanceFromBoundary) Intuitively, smaller lambda values will increase the spatial capture range of the distance prior. To apply to all label values, simply omit specifying the label, i.e. -l [lambda, boundaryProbability].

**Argument:** `-l %s`

**quick_registration:** (a boolean)
If = 1, use antsRegistrationSyNQuick.sh as the basis for registration during brain extraction, brain segmentation, and (optional) normalization to a template. Otherwise use antsRegistrationSyN.sh (default = 0).

**Argument:** `-q 1`

**debug:** (a boolean)
If > 0, runs a faster version of the script. Only for testing. Implies -u 0. Requires single thread computation for complete reproducibility.

**Argument:** `-z 1`

**num_threads:** (an integer (int or long), nipype default value: 1)
Number of ITK threads to use

**Args:** (a unicode string)
Additional parameters to the command

**Argument:** `%s`

**environ:** (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

**Outputs:**

**BrainExtractionMask:** (a pathlike object or string representing an existing file)
brain extraction mask

**ExtractedBrainN4:** (a pathlike object or string representing an existing file)
extracted brain from N4 image

**BrainSegmentation:** (a pathlike object or string representing an existing file)
brain segmentation image

**BrainSegmentationN4:** (a pathlike object or string representing an existing file)
N4 corrected image

**BrainSegmentationPosteriors:** (a list of items which are a pathlike object or string representing an existing file)
Posterior probability images

**CorticalThickness:** (a pathlike object or string representing an existing file)
cortical thickness file
TemplateToSubject1GenericAffine: (a pathlike object or string representing an existing file)
Template to subject affine

TemplateToSubject0Warp: (a pathlike object or string representing an existing file)
Template to subject warp

SubjectToTemplate1Warp: (a pathlike object or string representing an existing file)
Template to subject inverse warp

SubjectToTemplate0GenericAffine: (a pathlike object or string representing an existing file)
Template to subject inverse affine

SubjectToTemplateLogJacobian: (a pathlike object or string representing an existing file)
Template to subject log jacobian

CorticalThicknessNormedToTemplate: (a pathlike object or string representing an existing file)
Normalized cortical thickness

BrainVolumes: (a pathlike object or string representing an existing file)
Brain volumes as text

52.4.5 DenoiseImage

Link to code
Wraps the executable command DenoiseImage.

Examples

```python
>>> import copy
>>> from nipype.interfaces.ants import DenoiseImage

>>> denoise = DenoiseImage()
>>> denoise.inputs.dimension = 3
>>> denoise.inputs.input_image = 'im1.nii'
>>> denoise.cmdline
'DenoiseImage -d 3 -i im1.nii -n Gaussian -o im1_noise_corrected.nii -s 1'
```

```python
>>> denoise_2 = copy.deepcopy(denoise)
>>> denoise_2.inputs.output_image = 'output_corrected_image.nii.gz'
>>> denoise_2.inputs.noise_model = 'Rician'
>>> denoise_2.inputs.shrink_factor = 2
>>> denoise_2.cmdline
'DenoiseImage -d 3 -i im1.nii -n Rician -o output_corrected_image.nii.gz -s 2'
```

```python
>>> denoise_3 = DenoiseImage()
>>> denoise_3.inputs.input_image = 'im1.nii'
>>> denoise_3.inputs.save_noise = True
>>> denoise_3.cmdline
'DenoiseImage -i im1.nii -n Gaussian -o [ im1_noise_corrected.nii, im1_noise.nii
˓→] -s 1'
```

Inputs:

[Mandatory]
input_image: (a pathlike object or string representing an existing file)
A scalar image is expected as input for noise correction.

**save_noise**: (a boolean, nipype default value: False)
True if the estimated noise should be saved to file.

**mutually_exclusive**: noise_image

[Optional]

**dimension**: (2 or 3 or 4)
This option forces the image to be treated as a specified-dimensional image. If not specified, the program tries to infer the dimensionality from the input image.

**noise_model**: ('Gaussian' or 'Rician', nipype default value: Gaussian)
Employ a Rician or Gaussian noise model.

**shrink_factor**: (an integer (int or long), nipype default value: 1)
Running noise correction on large images can be time consuming. To lessen computation time, the input image can be resampled. The shrink factor, specified as a single integer, describes this resampling. Shrink factor = 1 is the default.

**output_image**: (a pathlike object or string representing a file)
The output consists of the noise corrected version of the input image.

**noise_image**: (a pathlike object or string representing a file)
Filename for the estimated noise.

**verbose**: (a boolean)
Verbose output.

**num_threads**: (an integer (int or long), nipype default value: 1)
Number of ITK threads to use

**args**: (a unicode string)
Additional parameters to the command

**environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

**Outputs**:

output_image: (a pathlike object or string representing an existing file)

noise_image: (a pathlike object or string representing a file)

### 52.4.6 JointFusion

Link to code

Wraps the executable command `jointfusion`.

**Examples**

```python
>>> from nipype.interfaces.ants import JointFusion
>>> at = JointFusion()
```
>>> at.inputs.dimension = 3
>>> at.inputs.modalities = 1
>>> at.inputs.method = 'Joint[0.1,2]'
>>> at.inputs.output_label_image = 'fusion_labelimage_output.nii'
>>> at.inputs.warped_intensity_images = ['im1.nii',
...   'im2.nii',
...   'im3.nii']
>>> at.inputs.warped_label_images = ['segmentation0.nii.gz',
...   'segmentation1.nii.gz',
...   'segmentation1.nii.gz']
>>> at.inputs.target_image = 'T1.nii'
>>> at.cmdline
'jointfusion 3 1 -m Joint[0.1,2] -tg T1.nii -g im1.nii -g im2.nii -g im3.nii -l segmentation0.nii.gz -l segmentation1.nii.gz -l segmentation1.nii.gz fusion_labelimage_output.nii'

>>> at.inputs.method = 'Joint'
>>> at.inputs.alpha = 0.5
>>> at.inputs.beta = 1
>>> at.inputs.patch_radius = [3,2,1]
>>> at.inputs.search_radius = [1,2,3]
>>> at.cmdline
'jointfusion 3 1 -m Joint[0.5,1] -rp 3x2x1 -rs 1x2x3 -tg T1.nii -g im1.nii -g im2. nii -g im3.nii -l segmentation0.nii.gz -l segmentation1.nii.gz -l segmentation1. nii.gz fusion_labelimage_output.nii'

Inputs:

[Mandatory]
dimension: (3 or 2 or 4, nipype default value: 3)
  image dimension (2, 3, or 4)
  argument: ``%d``, position: 0
modalities: (an integer (int or long))
  Number of modalities or features
  argument: ``%d``, position: 1
warped_intensity_images: (a list of items which are a pathlike object
  or string representing an existing file)
  Warped atlas images
  argument: ``-g %s...``
target_image: (a list of items which are a pathlike object or string
  representing an existing file)
  Target image(s)
  argument: ``-tg %s...``
warped_label_images: (a list of items which are a pathlike object or string
  representing an existing file)
  Warped atlas segmentations
  argument: ``-l %s...``
output_label_image: (a pathlike object or string representing a file)
  Output fusion label map image
  argument: ``%s``

[Optional]
method: (a unicode string, nipype default value: )
  Select voting method. Options: Joint (Joint Label Fusion). May be
  followed by optional parameters in brackets, e.g., -m Joint[0.1,2]
  argument: ``-m %s``
alpha: (a float, nipype default value: 0.0)
Regularization term added to matrix Mx for inverse
requires: method
beta: (an integer (int or long), nipype default value: 0)
   Exponent for mapping intensity difference to joint error
requires: method
patch_radius: (a list of items which are a value of class 'int')
   Patch radius for similarity measures, scalar or vector. Default:
   2x2x2
   argument: '-rp %s'
search_radius: (a list of items which are a value of class 'int')
   Local search radius. Default: 3x3x3
   argument: '-rs %s'
exclusion_region: (a pathlike object or string representing an
   existing file)
   Specify an exclusion region for the given label.
   argument: '-x %s'
atlas_group_id: (a list of items which are a value of class 'int')
   Assign a group ID for each atlas
   argument: '-gp %d...
atlas_group_weights: (a list of items which are a value of class
   'int')
   Assign the voting weights to each atlas group
   argument: '-gpw %d...
num_threads: (an integer (int or long), nipype default value: 1)
   Number of ITK threads to use
args: (a unicode string)
   Additional parameters to the command
   argument: %s
environ: (a dictionary with keys which are a bytes or None or a value
   of class 'str' and with values which are a bytes or None or a
   value of class 'str', nipype default value: {})
   Environment variables

Outputs:

output_label_image: (a pathlike object or string representing an
   existing file)

52.4.7 KellyKapowski

Link to code
Wraps the executable command KellyKapowski.
Nipype Interface to ANTs’ KellyKapowski, also known as DiReCT.
DiReCT is a registration based estimate of cortical thickness. It was published in S. R. Das, B. B. Avants, M.

Examples

```python
>>> from nipype.interfaces.ants.segmentation import KellyKapowski
>>> kk = KellyKapowski()
>>> kk.inputs.dimension = 3
>>> kk.inputs.segmentation_image = "segmentation0.nii.gz"
>>> kk.inputs.convergence = ":[45.0,0,10]"
>>> kk.inputs.thickness_prior_estimate = 10
>>> kk.cmdline
```

(continues on next page)
KellyKapowski --convergence "[45,0.0,10]" --output "[segmentation0_cortical_thickness.nii.gz,segmentation0_warped_white_matter.nii.gz]" --image-dimensionality 3 --gradient-step 0.025000 --maximum-number-of-invert-displacement-field-iterations 20 --number-of-integration-points 10 --segmentation-image "[segmentation0.nii.gz,2,3]" --smoothing-variance 1.000000 --smoothing-velocity-field-parameter 1.500000 --thickness-prior-estimate 10.000000

**Inputs:**

[Mandatory]
segmentation_image: (a pathlike object or string representing an existing file)
A segmentation image must be supplied labeling the gray and white matters. Default values = 2 and 3, respectively.
argument: ``--segmentation-image "%%s"``

[Optional]
dimension: (3 or 2, nipype default value: 3)
image dimension (2 or 3)
argument: ``--image-dimensionality %d``
gray_matter_label: (an integer (int or long), nipype default value: 2)
The label value for the gray matter label in the segmentation_image.
white_matter_label: (an integer (int or long), nipype default value: 3)
The label value for the white matter label in the segmentation_image.
gray_matter_prob_image: (a pathlike object or string representing an existing file)
In addition to the segmentation image, a gray matter probability image can be used. If no such image is supplied, one is created using the segmentation image and a variance of 1.0 mm.
argument: ``--gray-matter-probability-image "%%s"``
white_matter_prob_image: (a pathlike object or string representing an existing file)
In addition to the segmentation image, a white matter probability image can be used. If no such image is supplied, one is created using the segmentation image and a variance of 1.0 mm.
argument: ``--white-matter-probability-image "%%s"``
convergence: (a unicode string, nipype default value: )
Convergence is determined by fitting a line to the normalized energy profile of the last N iterations (where N is specified by the window size) and determining the slope which is then compared with the convergence threshold.
argument: ``--convergence "%%s"``
thickness_prior_estimate: (a float, nipype default value: 10)
Provides a prior constraint on the final thickness measurement in mm.
argument: ``--thickness-prior-estimate %f``
thickness_prior_image: (a pathlike object or string representing an existing file)
An image containing spatially varying prior thickness values.
argument: ``--thickness-prior-image "%%s"``
gradient_step: (a float, nipype default value: 0.025)
Gradient step size for the optimization.
argument: ``--gradient-step %f``

(continues on next page)
smoothing_variance: (a float, nipype default value: 1.0)
    Defines the Gaussian smoothing of the hit and total images.
    argument: `--smoothing-variance %f`

smoothing_velocity_field: (a float, nipype default value: 1.5)
    Defines the Gaussian smoothing of the velocity field (default = 1.5). If the b-spline smoothing option is chosen, then this defines
    the isotropic mesh spacing for the smoothing spline (default = 15).
    argument: `--smoothing-velocity-field-parameter %f`

use_bspline_smoothing: (a boolean)
    Sets the option for B-spline smoothing of the velocity field.
    argument: `--use-bspline-smoothing 1`

number_integration_points: (an integer (int or long), nipype default
    value: 10)
    Number of compositions of the diffeomorphism per iteration.
    argument: `--number-of-integration-points %d`

max_invert_displacement_field_iters: (an integer (int or long),
    nipype default value: 20)
    Maximum number of iterations for estimating the invertdisplacement
    field.
    argument: `--maximum-number-of-invert-displacement-field-iterations
    %d`

cortical_thickness: (a pathlike object or string representing a file)
    Filename for the cortical thickness.
    argument: `--output "%s"`

warped_white_matter: (a pathlike object or string representing a
    file)
    Filename for the warped white matter image.

num_threads: (an integer (int or long), nipype default value: 1)
    Number of ITK threads to use.

args: (a unicode string)
    Additional parameters to the command
    argument: `"%s"

environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipipe default value: {})
    Environment variables

Outputs:

cortical_thickness: (a pathlike object or string representing a file)
    A thickness map defined in the segmented gray matter.

warped_white_matter: (a pathlike object or string representing a
    file)
    A warped white matter image.

References:

None

52.4.8 LaplacianThickness

Link to code
Wraps the executable command LaplacianThickness.
Calculates the cortical thickness from an anatomical image
Examples

```python
>>> from nipype.interfaces.ants import LaplacianThickness
>>> cort_thick = LaplacianThickness()
>>> cort_thick.inputs.input_wm = 'white_matter.nii.gz'
>>> cort_thick.inputs.input_gm = 'gray_matter.nii.gz'
```n
```
LaplacianThickness white_matter.nii.gz gray_matter.nii.gz white_matter_thickness.nii.gz
```n
```python
>>> cort_thick.inputs.output_image = 'output_thickness.nii.gz'
>>> cort_thick.cmdline
LaplacianThickness white_matter.nii.gz gray_matter.nii.gz output_thickness.nii.gz
```n
```
Inputs:

[Mandatory]
input_wm: (a pathlike object or string representing a file)
  white matter segmentation image
  argument: ``%s``, position: 1
input_gm: (a pathlike object or string representing a file)
  gray matter segmentation image
  argument: ``%s``, position: 2

[Optional]
output_image: (a pathlike object or string representing a file)
  name of output file
  argument: ``%s``, position: 3
smooth_param: (a float)
  Sigma of the Laplacian Recursive Image Filter (defaults to 1)
  argument: ``%s``, position: 4
prior_thickness: (a float)
  Prior thickness (defaults to 500)
  argument: ``%s``, position: 5
  requires: smooth_param
dT: (a float)
  Time delta used during integration (defaults to 0.01)
  argument: ``%s``, position: 6
  requires: prior_thickness
sulcus_prior: (a float)
  Positive floating point number for sulcus prior. Authors said that
  0.15 might be a reasonable value
  argument: ``%s``, position: 7
  requires: dT
tolerance: (a float)
  Tolerance to reach during optimization (defaults to 0.001)
  argument: ``%s``, position: 8
  requires: sulcus_prior
num_threads: (an integer (int or long), nipype default value: 1)
  Number of ITK threads to use
args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
  Environment variables
```
Outputs:

output_image: (a pathlike object or string representing an existing file)
  Cortical thickness

52.4.9 N4BiasFieldCorrection

Link to code
Wraps the executable command N4BiasFieldCorrection. Bias field correction.
N4 is a variant of the popular N3 (nonparametric nonuniform normalization) retrospective bias correction algorithm. Based on the assumption that the corruption of the low frequency bias field can be modeled as a convolution of the intensity histogram by a Gaussian, the basic algorithmic protocol is to iterate between deconvolving the intensity histogram by a Gaussian, remapping the intensities, and then spatially smoothing this result by a B-spline modeling of the bias field itself. The modifications from and improvements obtained over the original N3 algorithm are described in [Tustison2010].

Examples

```python
>>> import copy
>>> from nipype.interfaces.ants import N4BiasFieldCorrection

>>> n4 = N4BiasFieldCorrection()
>>> n4.inputs.dimension = 3
>>> n4.inputs.input_image = 'structural.nii'
>>> n4.inputs.bspline_fitting_distance = 300
>>> n4.inputs.shrink_factor = 3
>>> n4.inputs.n_iterations = [50, 50, 30, 20]
>>> n4.cmdline
  'N4BiasFieldCorrection --bspline-fitting [300] -d 3 --input-image structural.
   nii --convergence [50x50x30x20] --output structural_corrected.nii --shrink-
   factor 3'

>>> n4_2 = copy.deepcopy(n4)
>>> n4_2.inputs.convergence_threshold = 1e-6
>>> n4_2.cmdline
  'N4BiasFieldCorrection --bspline-fitting [300] -d 3 --input-image structural.
   nii --convergence [50x50x30x20, 1e-06] --output structural_corrected.nii --
   shrink-factor 3'

>>> n4_3 = copy.deepcopy(n4_2)
>>> n4_3.inputs.bspline_order = 5
>>> n4_3.cmdline
  'N4BiasFieldCorrection --bspline-fitting [300, 5] -d 3 --input-image structural.
   nii --convergence [50x50x30x20, 1e-06] --output structural_corrected.nii --
   shrink-factor 3'

>>> n4_4 = N4BiasFieldCorrection()
>>> n4_4.inputs.input_image = 'structural.nii'
>>> n4_4.inputs.save_bias = True
>>> n4_4.inputs.dimension = 3
>>> n4_4.cmdline
  'N4BiasFieldCorrection -d 3 --input-image structural.nii --output [structural_
   corrected.nii, structural_bias.nii]'
```

Inputs:
[Mandatory]
input_image: (a pathlike object or string representing a file)
    input for bias correction. Negative values or values close to zero
    should be processed prior to correction
    argument: ``--input-image %s``
save_bias: (a boolean, nipype default value: False)
    True if the estimated bias should be saved to file.
    mutually_exclusive: bias_image
copy_header: (a boolean, nipipe default value: False)
    copy headers of the original image into the output (corrected) file

[Optional]
dimension: (3 or 2 or 4, nipype default value: 3)
    image dimension (2, 3 or 4)
    argument: ``-d %d``
mask_image: (a pathlike object or string representing a file)
    image to specify region to perform final bias correction in
    argument: ``--mask-image %s``
weight_image: (a pathlike object or string representing a file)
    image for relative weighting (e.g. probability map of the white
    matter) of voxels during the B-spline fitting.
    argument: ``--weight-image %s``
output_image: (a unicode string)
    output file name
    argument: ``--output %s``
bspline_fitting_distance: (a float)
    argument: ``--bspline-fitting %s``
bspline_order: (an integer (int or long))
    requires: bspline_fitting_distance
shrink_factor: (an integer (int or long))
    argument: ``--shrink-factor %d``
n_iterations: (a list of items which are an integer (int or long))
    argument: ``--convergence %s``
convergence_threshold: (a float)
    requires: n_iterations
bias_image: (a pathlike object or string representing a file)
    Filename for the estimated bias.
rescale_intensities: (a boolean, nipipe default value: False)
    [NOTE: Only ANTs>=2.1.0]
    At each iteration, a new intensity mapping is calculated and applied
    but there
    is nothing which constrains the new intensity range to be within
    certain values.
    The result is that the range can "drift" from the original at each
    iteration.
    This option rescales to the [min,max] range of the original image
    intensities
    within the user-specified mask.
    argument: ``--r``
num_threads: (an integer (int or long), nipype default value: 1)
    Number of ITK threads to use
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})
    Environment variables
Outpus:

- output_image: (a pathlike object or string representing an existing file)
  - Warped image
- bias_image: (a pathlike object or string representing an existing file)
  - Estimated bias

## 52.5 interfaces.ants.utils

### 52.5.1 AffineInitializer

**Link to code**

Wraps the executable command `antsAffineInitializer`. Initialize an affine transform (as in `antsBrainExtraction.sh`)

```python
>>> from nipype.interfaces.ants import AffineInitializer
>>> init = AffineInitializer()
>>> init.inputs.fixed_image = 'fixed1.nii'
>>> init.inputs.moving_image = 'moving1.nii'
>>> init.cmdline
'antsAffineInitializer 3 fixed1.nii moving1.nii transform.mat 15.000000 0.100000 → 0 10'
```

**Inputs:**

- **fixed_image:** (a pathlike object or string representing an existing file)
  - reference image
  - argument: ```%s```, position: 1
- **moving_image:** (a pathlike object or string representing an existing file)
  - moving image
  - argument: ```%s```, position: 2
- **dimension:** (3 or 2, nipype default value: 3)
  - dimension
  - argument: ```%s```, position: 0
- **out_file:** (a pathlike object or string representing a file, nipype default value: `transform.mat`)
  - output transform file
  - argument: ```%s```, position: 3
- **search_factor:** (a float, nipype default value: 15.0)
  - increments (degrees) for affine search
  - argument: ```%f```, position: 4
- **radian_fraction:** (0.0 <= a floating point number <= 1.0, nipype default value: 0.1)
  - search this arc +/- principal axes
  - argument: ```%f```, position: 5
- **principal_axes:** (a boolean, nipype default value: False)
  - whether the rotation is searched around an initial principal axis alignment.
  - argument: ```%d```, position: 6
- **local_search:** (an integer (int or long), nipype default value: 10)
  - (continues on next page)
determines if a local optimization is run at each search point for
the set number of iterations
argument: ``%d``, position: 7
num_threads: (an integer (int or long), nipype default value: 1)
  Number of ITK threads to use
args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
  Environment variables

Outputs:

out_file: (a pathlike object or string representing a file)
  output transform file

52.5.2 AverageAffineTransform

Link to code
Wraps the executable command AverageAffineTransform.

Examples

```python
>>> from nipype.interfaces.ants import AverageAffineTransform
>>> avg = AverageAffineTransform()
>>> avg.inputs.dimension = 3
>>> avg.inputs.transforms = ['trans.mat', 'func_to_struct.mat']
>>> avg.inputs.output_affine_transform = 'MYtemplatewarp.mat'
>>> avg.cmdline
'AverageAffineTransform 3 MYtemplatewarp.mat trans.mat func_to_struct.mat'
```

Inputs:

[Mandatory]
dimension: (3 or 2)
  image dimension (2 or 3)
  argument: ``%d``, position: 0
output_affine_transform: (a pathlike object or string representing a
  file)
  Outputfname.txt: the name of the resulting transform.
  argument: ``%s``, position: 1
transforms: (a list of items which are a pathlike object or string
  representing an existing file)
  transforms to average
  argument: ``%s``, position: 3

[Optional]
um_threads: (an integer (int or long), nipype default value: 1)
  Number of ITK threads to use
args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
Environment variables

affine_transform: (a pathlike object or string representing an existing file)
average transform file

52.5.3 AverageImages

Link to code
Wraps the executable command AverageImages.

Examples

```python
>>> from nipype.interfaces.ants import AverageImages
>>> avg = AverageImages()
>>> avg.inputs.dimension = 3
>>> avg.inputs.output_average_image = "average.nii.gz"
>>> avg.inputs.normalize = True
>>> avg.inputs.images = ["rcls1.nii", 'rcls1.nii']
>>> avg.cmdline
'AverageImages 3 average.nii.gz 1 rcls1.nii rcls1.nii'
```

Inputs:

[Mandatory]
dimension: (3 or 2)
  image dimension (2 or 3)
  argument: `\'%d\'`, position: 0
normalize: (a boolean)
  Normalize: if true, the 2nd image is divided by its mean. This will select the largest image to average into.
  argument: `\'%d\'`, position: 2
images: (a list of items which are a pathlike object or string representing an existing file)
  image to apply transformation to (generally a coregistered functional)
  argument: `\'%s\'`, position: 3

[Optional]
output_average_image: (a pathlike object or string representing a file, nipype default value: average.nii)
  the name of the resulting image.
  argument: `\'%s\'`, position: 1
num_threads: (an integer (int or long), nipype default value: 1)
  Number of ITK threads to use
args: (a unicode string)
  Additional parameters to the command
  argument: `\'%s\`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  Environment variables

Outputs:
output_average_image: (a pathlike object or string representing an existing file)
    average image file

52.5.4 ComposeMultiTransform

Link to code
Wraps the executable command ComposeMultiTransform. Take a set of transformations and convert them to a single transformation matrix/warpfield.

Examples

```python
>>> from nipype.interfaces.ants import ComposeMultiTransform
>>> compose_transform = ComposeMultiTransform()
>>> compose_transform.inputs.dimension = 3
>>> compose_transform.inputs.transforms = ['struct_to_template.mat', 'func_to_struct.mat']
>>> compose_transform.cmdline
'ComposeMultiTransform 3 struct_to_template_composed.mat struct_to_template.mat
→func_to_struct.mat'
```

Inputs:

[Mandatory]
transforms: (a list of items which are a pathlike object or string representing an existing file)
    transforms to average
    argument: `"%s"`, position: 3

[Optional]
dimension: (3 or 2, nipype default value: 3)
    image dimension (2 or 3)
    argument: `"%d"`, position: 0
output_transform: (a pathlike object or string representing a file)
    the name of the resulting transform.
    argument: `"%s"`, position: 1
reference_image: (a pathlike object or string representing a file)
    Reference image (only necessary when output is warpfield)
    argument: `"%s"`, position: 2
num_threads: (an integer (int or long), nipype default value: 1)
    Number of ITK threads to use
args: (a unicode string)
    Additional parameters to the command
    argument: `"%s"`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

output_transform: (a pathlike object or string representing an existing file)
    Composed transform file
52.5.5 CreateJacobianDeterminantImage

Link to code
Wraps the executable command CreateJacobianDeterminantImage.

Examples

```python
def from nipype.interfaces.ants import CreateJacobianDeterminantImage() >>> jacobian = CreateJacobianDeterminantImage() >>> jacobian.inputs.imageDimension = 3 >>> jacobian.inputs.deformationField = 'ants_Warp.nii.gz' >>> jacobian.inputs.outputImage = 'out_name.nii.gz' >>> jacobian.cmdline 'CreateJacobianDeterminantImage 3 ants_Warp.nii.gz out_name.nii.gz'
```

Inputs:

- **imageDimension**: (3 or 2)
  - image dimension (2 or 3)
  - argument: `\`%d\``, position: 0
- **deformationField**: (a pathlike object or string representing an existing file)
  - deformation transformation file
  - argument: `\`%s\``, position: 1
- **outputImage**: (a pathlike object or string representing a file)
  - output filename
  - argument: `\`%s\``, position: 2

Optional:

- **doLogJacobian**: (0 or 1)
  - return the log jacobian
  - argument: `\`%d\``, position: 3
- **useGeometric**: (0 or 1)
  - return the geometric jacobian
  - argument: `\`%d\``, position: 4
- **num_threads**: (an integer (int or long), nipype default value: 1)
  - Number of ITK threads to use
- **args**: (a unicode string)
  - Additional parameters to the command
  - argument: `\`%s\`
- **environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})  
  - Environment variables

Outputs:

- **jacobian_image**: (a pathlike object or string representing an existing file)
  - jacobian image

52.5.6 LabelGeometry

Link to code
Wraps the executable command LabelGeometryMeasures.
Extracts geometry measures using a label file and an optional image file
Examples

```python
>>> from nipype.interfaces.ants import LabelGeometry
>>> label_extract = LabelGeometry()
>>> label_extract.inputs.dimension = 3
>>> label_extract.inputs.label_image = 'atlas.nii.gz'
>>> label_extract.cmdline
'LabelGeometryMeasures 3 atlas.nii.gz [] atlas.csv'
>>> label_extract.inputs.intensity_image = 'ants_Warp.nii.gz'
>>> label_extract.cmdline
'LabelGeometryMeasures 3 atlas.nii.gz ants_Warp.nii.gz atlas.csv'
```

Inputs:

- **label_image**: (a pathlike object or string representing a file)
  - label image to use for extracting geometry measures
  - argument: `''%s''`, position: 1
- **intensity_image**: (a pathlike object or string representing an existing file, nipype default value: `[]`)
  - Intensity image to extract values from. This is an optional input
  - argument: `''%s''`, position: 2
- **dimension**: (3 or 2, nipype default value: 3)
  - image dimension (2 or 3)
  - argument: `''%d''`, position: 0
- **output_file**: (a unicode string)
  - name of output file
  - argument: `''%s''`, position: 3
- **num_threads**: (an integer (int or long), nipype default value: 1)
  - Number of ITK threads to use
- **args**: (a unicode string)
  - Additional parameters to the command
  - argument: `''%s''`
- **environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: `{}`)
  - Environment variables

Outputs:

- **output_file**: (a pathlike object or string representing an existing file)
  - CSV file of geometry measures

52.5.7 MultiplyImages

Link to code

Wraps the executable command MultiplyImages.

Examples

```python
>>> from nipype.interfaces.ants import MultiplyImages
>>> test = MultiplyImages()
```
>>> test.inputs.dimension = 3
>>> test.inputs.first_input = 'moving2.nii'
>>> test.inputs.second_input = 0.25
>>> test.inputs.output_product_image = "out.nii"
>>> test.cmdline
'MultiplyImages 3 moving2.nii 0.25 out.nii'

Inputs:

[Mandatory]
dimension: (3 or 2)
  image dimension (2 or 3)
  argument: `\%d`, position: 0

first_input: (a pathlike object or string representing an existing
  file)
  image 1
  argument: `\%s`, position: 1

second_input: (a pathlike object or string representing an existing
  file or a float)
  image 2 or multiplication weight
  argument: `\%s`, position: 2

output_product_image: (a pathlike object or string representing a
  file)
  Outputfname.nii.gz: the name of the resulting image.
  argument: `\%s`, position: 3

[Optional]

num_threads: (an integer (int or long), nipype default value: 1)
  Number of ITK threads to use

args: (a unicode string)
  Additional parameters to the command
  argument: `\%s`

environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str', with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
  Environment variables

Outputs:

output_product_image: (a pathlike object or string representing an
  existing file)
  average image file

52.6 interfaces.ants.visualization

52.6.1 ConvertScalarImageToRGB

Link to code
Wraps the executable command ConvertScalarImageToRGB.

Examples

```python
>>> from nipype.interfaces.ants.visualization import ConvertScalarImageToRGB
>>> converter = ConvertScalarImageToRGB()
>>> converter.inputs.dimension = 3
```
```python
>>> converter.inputs.input_image = 'T1.nii.gz'
>>> converter.inputs.colormap = 'jet'
>>> converter.inputs.minimum_input = 0
>>> converter.inputs.maximum_input = 6
>>> converter.cmdline
'ConvertScalarImageToRGB 3 T1.nii.gz rgb.nii.gz none jet none 0 6 0 255'
```

### Inputs:

<table>
<thead>
<tr>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension</td>
<td>(3 or 2, nipype default value: 3) Main input is a 3-D grayscale image.</td>
</tr>
<tr>
<td>input_image</td>
<td>(a pathlike object or string representing an existing file) Possible colormaps: grey, red, green, blue, copper, jet, hsv, spring, summer, autumn, winter, hot, cool, overunder, custom.</td>
</tr>
<tr>
<td>colormap</td>
<td>(a unicode string, nipype default value: ) Possible colormaps: grey, red, green, blue, copper, jet, hsv, spring, summer, autumn, winter, hot, cool, overunder, custom.</td>
</tr>
<tr>
<td>minimum_input</td>
<td>(an integer (int or long)) Minimum input</td>
</tr>
<tr>
<td>maximum_input</td>
<td>(an integer (int or long)) Maximum input</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>output_image</td>
<td>(a unicode string, nipype default value: rgb.nii.gz) RGB output image</td>
</tr>
<tr>
<td>mask_image</td>
<td>(a pathlike object or string representing an existing file, nipype default value: none) Mask image</td>
</tr>
<tr>
<td>custom_color_map_file</td>
<td>(a unicode string, nipype default value: none) Custom color map file</td>
</tr>
<tr>
<td>minimum_RGB_output</td>
<td>(an integer (int or long), nipype default value: 0) Minimum output RGB image</td>
</tr>
<tr>
<td>maximum_RGB_output</td>
<td>(an integer (int or long), nipype default value: 255) Maximum output RGB image</td>
</tr>
<tr>
<td>num_threads</td>
<td>(an integer (int or long), nipype default value: 1) Number of ITK threads to use</td>
</tr>
<tr>
<td>args</td>
<td>(a unicode string) Additional parameters to the command</td>
</tr>
<tr>
<td>environ</td>
<td>(a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables</td>
</tr>
</tbody>
</table>

### Outputs:
output_image: (a pathlike object or string representing an existing file)
converted RGB image

52.6.2 CreateTiledMosaic

Link to code
Wraps the executable command CreateTiledMosaic.
The program CreateTiledMosaic in conjunction with ConvertScalarImageToRGB provides useful functionality for common image analysis tasks. The basic usage of CreateTiledMosaic is to tile a 3-D image volume slice-wise into a 2-D image.

Examples

```python
>>> from nipype.interfaces.ants.visualization import CreateTiledMosaic
>>> mosaic_slicer = CreateTiledMosaic()
>>> mosaic_slicer.inputs.input_image = 'T1.nii.gz'
>>> mosaic_slicer.inputs.rgb_image = 'rgb.nii.gz'
>>> mosaic_slicer.inputs.mask_image = 'mask.nii.gz'
>>> mosaic_slicer.inputs.output_image = 'output.png'
>>> mosaic_slicer.inputs.alpha_value = 0.5
>>> mosaic_slicer.inputs.direction = 2
>>> mosaic_slicer.inputs.pad_or_crop = '[-15x -50 , -15x -30 ,0]'
>>> mosaic_slicer.inputs.slices = '[2 ,100 ,160]'
>>> mosaic_slicer.cmdline
'CreateTiledMosaic -a 0.50 -d 2 -i T1.nii.gz -x mask.nii.gz -o output.png -p [-15x -50 , -15x -30 ,0] -r rgb.nii.gz -s [2 ,100 ,160]'
```

Inputs:

[Mandatory]
input_image: (a pathlike object or string representing an existing file)
Main input is a 3-D grayscale image.
argument: ``(\'\`-i %s\'")
rgb_image: (a pathlike object or string representing an existing file)
An optional Rgb image can be added as an overlay. It must have the same image geometry as the input grayscale image.
argument: ``(\`-r %s\'")

[Optional]
mask_image: (a pathlike object or string representing an existing file)
Specifies the ROI of the RGB voxels used.
argument: ``(\`-x %s\'")
alpha_value: (a float)
If an Rgb image is provided, render the overlay using the specified alpha parameter.
argument: ``(\`-a %.2f\'")
output_image: (a unicode string, nipype default value: output.png)
The output consists of the tiled mosaic image.
argument: ``(\`-o %s\'")
tile_geometry: (a unicode string)
The tile geometry specifies the number of rows and columns in the output image. For example, if the user specifies "5x10", then 5 rows

(continues on next page)
by 10 columns of slices are rendered. If \( R < 0 \) and \( C > 0 \) (or vice versa), the negative value is selected based on direction.

**argument:** `\'-t \$s\'`

**direction:** (an integer \((\text{int or long})\))

Specifies the direction of the slices. If no direction is specified, the direction with the coarsest spacing is chosen.

**argument:** `\'-d \$d\'`

**pad_or_crop:** (a unicode string)

- Argument passed to `-p` flag: `[padVoxelWidth, <constantValue=0>] [lowerPadding[0]xlowerPadding[1], upperPadding[0]xupperPadding[1], constantValue]`

The user can specify whether to pad or crop a specified voxel-width boundary of each individual slice. For this program, cropping is simply padding with negative voxel-widths. If one pads (+), the user can also specify a constant pad value (default = 0). If a mask is specified, the user can use the mask to define the region, by using the keyword "mask" plus an offset, e.g. "-p mask+3".

**argument:** `\'-p \$s\'`

**slices:** (a unicode string)

- Number of slices to increment: `Slice1xSlice2xSlice3[numberOfSlicesToIncrement, <minSlice=0>, <maxSlice=lastSlice>]`

**argument:** `\'-s \$s\'`

**flip_slice:** (a unicode string)

- `flipXxflipY`

**permute_axes:** (a boolean)

- `doPermute`

**num_threads:** (an integer \((\text{int or long})\), nipype default value: 1)

- Number of ITK threads to use

**args:** (a unicode string)

- Additional parameters to the command

**environ:** (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

Environment variables

**Outputs:**

**output_image:** (a pathlike object or string representing an existing file)

- `image file`
53.1 interfaces.base.core

53.1.1 BaseInterface

Link to code
Implements common interface functionality.

Implements

- Initializes inputs/outputs from input_spec/output_spec
- Provides help based on input_spec and output_spec
- Checks for mandatory inputs before running an interface
- Runs an interface and returns results
- Determines which inputs should be copied or linked to cwd

This class does not implement aggregate_outputs, input_spec or output_spec. These should be defined by derived classes.
This class cannot be instantiated.

Relevant Interface attributes

input_spec points to the traited class for the inputs output_spec points to the traited class for the outputs
_redirect_x should be set to True when the interface requires connecting to a $DISPLAY (default is False).

resource_monitor if False prevents resource-monitoring this interface, if True monitoring will be enabled IFF the general Nipype config is set on (resource_monitor = true).

Inputs:

None

Outputs:

None

53.1.2 CommandLine

Link to code
Implements functionality to interact with command line programs class must be instantiated with a command argument

**Parameters**

**command** [string] define base immutable *command* you wish to run

**args** [string, optional] optional arguments passed to base *command*

**Examples**

```python
>>> import pprint
>>> from nipype.interfaces.base import CommandLine

>>> cli = CommandLine(command='ls', environ={'DISPLAY': ':1'})
>>> cli.inputs.args = '-al'

>>> cli.cmdline
'ls -al'

# Use get_traitsfree() to check all inputs set
>>> pprint.pprint(cli.inputs.get_traitsfree())  # doctest: {'args': '-al', 'environ': {'DISPLAY': ':1'}}

>>> cli.inputs.get_hashval()[0][0]
('args', '-al')

>>> cli.inputs.get_hashval()[1]
'11c37f97649cd61627f4afe5136af8c0'
```

**Inputs:**

```
[Optional]
args: (a unicode string)
    Additional parameters to the command argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})  

Environment variables
```

**Outputs:**

```
None
```

### 53.1.3 LibraryBaseInterface

**Link to code**

Inputs:

```
None
```

Outputs:

```
None
```

### 53.1.4 MpiCommandLine

**Link to code**

Implements functionality to interact with command line programs that can be run with MPI (i.e. using `mpiexec`).
Examples

```python
>>> from nipype.interfaces.base import MpiCommandLine
>>> mpi_cli = MpiCommandLine(command='my_mpi_prog')
>>> mpi_cli.inputs.args = '-v'
>>> mpi_cli.cmdline
'my_mpi_prog -v'

>>> mpi_cli.inputs.use_mpi = True
>>> mpi_cli.inputs.n_procs = 8
>>> mpi_cli.cmdline
'mpiexec -n 8 my_mpi_prog -v'
```

Inputs:

- `use_mpi`: (a boolean, nipype default value: False)
  Whether or not to run the command with mpiexec
- `n_procs`: (an integer (int or long))
  Num processors to specify to mpiexec. Do not specify if this is
  managed externally (e.g. through SGE)
- `args`: (a unicode string)
  Additional parameters to the command
- `environ`: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})

Outputs:

None

### 53.1.5 SEMLikeCommandLine

Link to code

In SEM derived interface all outputs have corresponding inputs. However, some SEM commands create outputs
that are not defined in the XML. In those cases one has to create a subclass of the autogenerated one and overload
the `_list_outputs` method. `_outputs_from_inputs` should still be used but only for the reduced (by excluding those
that do not have corresponding inputs list of outputs).

Inputs:

- `args`: (a unicode string)
  Additional parameters to the command
- `environ`: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})

Outputs:

None

### 53.1.6 SimpleInterface

Link to code
An interface pattern that allows outputs to be set in a dictionary called `_results` that is automatically interpreted by `_list_outputs()` to find the outputs.

When implementing `_run_interface`, set outputs with:

```python
self._results[out_name] = out_value
```

This can be a way to upgrade a `Function` interface to do type checking.

**Examples**

```python
>>> from nipype.interfaces.base import (
...     SimpleInterface, BaseInterfaceInputSpec, TraitedSpec)
... 
... def double(x):
...     return 2 * x
... 
... class DoubleInputSpec(BaseInterfaceInputSpec):
...     x = traits.Float(mandatory=True)
... 
... class DoubleOutputSpec(TraitedSpec):
...     doubled = traits.Float()
... 
... class Double(SimpleInterface):
...     input_spec = DoubleInputSpec
...     output_spec = DoubleOutputSpec
... 
...     def _run_interface(self, runtime):
...         self._results['doubled'] = double(self.inputs.x)
...         return runtime
...

>>> dbl = Double()
>>> dbl.inputs.x = 2
>>> dbl.run().outputs.doubled
4.0
```

**53.1.7 StdOutCommandLine**

**Inputs:**

None

**Outputs:**

None
53.2 interfaces.base.specs

53.2.1 get_filecopy_info()

Link to code
Provides information about file inputs to copy or link to cwd. Necessary for pipeline operation

53.3 interfaces.base.support

53.3.1 format_help()

Link to code
Prints help text of a Nipype interface

```python
>>> from nipype.interfaces.afni import GCOR
>>> GCOR.help()  # doctest: +ELLIPSIS, +NORMALIZE_WHITESPACE
Wraps the executable command `@compute_gcor`.  
Computes the average correlation between every voxel 
and ever other voxel, over any give mask.  
For complete details, ...
```

53.3.2 get_trait_desc()

Link to code
Parses a HasTraits object into a nipype documentation string

53.4 interfaces.base.traits_extension

53.4.1 has_metadata()

Link to code
Checks if a given trait has a metadata (and optionally if it is set to particular value)

53.4.2 isdefined()

Link to code

53.4.3 rebase_path_traits()

Link to code
Rebase a BasePath-derived trait given an interface spec.

53.4.4 resolve_path_traits()

Link to code
Resolve a BasePath-derived trait given an interface spec.
54.1 interfaces.brainsuite.brainsuite

54.1.1 BDP

Link to code

Wraps the executable command bdp.sh.

BrainSuite Diffusion Pipeline (BDP) enables fusion of diffusion and structural MRI information for advanced image and connectivity analysis. It provides various methods for distortion correction, co-registration, diffusion modeling (DTI and ODF) and basic ROI-wise statistic. BDP is a flexible and diverse tool which supports wide variety of diffusion datasets. For more information, please see: http://brainsuite.org/processing/diffusion/

Examples

```python
>>> from nipype.interfaces import brainsuite

>>> bdp = brainsuite.BDP()

>>> bdp.inputs.bfcFile = '/directory/subdir/prefix.bfc.nii.gz'

>>> bdp.inputs.inputDiffusionData = '/directory/subdir/prefix.dwi.nii.gz'

>>> bdp.inputs.BVecBValPair = ['/directory/subdir/prefix.dwi.bvec', '/directory/subdir/prefix.dwi.bval']

>>> results = bdp.run() #doctest: +SKIP
```

Inputs:

- bfcFile: (a pathlike object or string representing a file)
  Specify absolute path to file produced by bfc. By default, bfc produces the file in the format: prefix.bfc.nii.gz
  argument: `%s` position: 0 mutually_exclusive: noStructuralRegistration

- noStructuralRegistration: (a boolean)
  Allows BDP to work without any structural input. This can useful when one is only interested in diffusion modelling part of BDP. With this flag only fieldmap-based distortion correction is supported.
  outPrefix can be used to specify fileprefix of the output filenames.
  Change dwiMask to define region of interest for diffusion modelling.
argument: `--no-structural-registration`\), position: 0

mutually_exclusive: bfcFile

inputDiffusionData: (a path like object or string representing a file)

  Specifies the absolute path and filename of the input diffusion data
  in 4D NIfTI-1 format. The flag must be followed by the filename.
  Only NIfTI-1 files with extension .nii or .nii.gz are supported.
  Furthermore, either bMatrixFile, or a combination of both bValueFile
  and diffusionGradientFile must be used to provide the necessary
  b-matrices/b-values and gradient vectors.

argument: `--nii %s`, position: -2

bMatrixFile: (a path like object or string representing a file)

  Specifies the absolute path and filename of the file containing
  b-matrices for diffusion-weighted scans. The flag must be followed
  by the filename. This file must be a plain text file containing 3x3
  matrices for each diffusion encoding direction. It should contain
  zero matrices corresponding to b=0 images. This file usually has
  ".bmat" as its extension, and can be used to provide BDP with the
  more-accurate b-matrices as saved by some proprietary scanners. The
  b-matrices specified by the file must be in the voxel coordinates of
  the input diffusion weighted image (NIfTI file). In case b-matrices
  are not known/calculated, bvec and .bval files can be used instead
  (see diffusionGradientFile and bValueFile).

argument: `--bmat %s`, position: -1

mutually_exclusive: BVecBValPair

BVecBValPair: (a list of from 2 to 2 items which are a unicode
  string)

  Must input a list containing first the BVector file, then the BValue
  file (both must be absolute paths)

Example: bdp.inputs.BVecBValPair =
  ['/directory/subdir/prefix.dwi.bvec',
   '/directory/subdir/prefix.dwi.bval'] The first item in the list
  specifies the filename of the file containing b-values for the
  diffusion scan. The b-value file must be a plain-text file and
  usually has an extension of .bval

The second item in the list specifies the filename of the file
containing the diffusion gradient directions (specified in the voxel
coordinates of the input diffusion-weighted image)The b-vectors file
must be a plain text file and usually has an extension of .bvec

argument: `--bvec %s --bval %s`, position: -1

mutually_exclusive: bMatrixFile

[Optional]

dataSinkDelay: (a list of items which are a unicode string)

  For use in parallel processing workflows including Brainsuite
  Cortical Surface Extraction sequence. Connect datasink out_file to
dataSinkDelay to delay execution of BDP until dataSink has finished
sinking outputs. In particular, BDP may be run after BFC has
finished. For more information see

  http://brainsuite.org/processing/diffusion/pipeline/

argument: `%s`

phaseEncodingDirection: ('x' or 'x-' or 'y' or 'y-' or 'z' or 'z-')

  Specifies the phase-encoding direction of the EPI (diffusion)
  images. It is same as the dominant direction of distortion in the
  images. This information is used to constrain the distortion
  correction along the specified direction. Directions are represented
  by any one of x, x-, y, y-, z or z-. "x" direction increases towards
  the right side of the subject, while "x-" increases towards the left

(continues on next page)
side of the subject. Similarly, "$y$" and "$y-$" are along the anterior-
posterior direction of the subject, and "$z$" & "$z-$" are along the
inferior-superior direction. When this flag is not used, BDP uses
"y" as the default phase-encoding direction.

argument: `--dir=%s`

echoSpacing: (a float)

Sets the echo spacing to t seconds, which is used for fieldmap-based
distortion correction. This flag is required when using
fieldmapCorrection

argument: `--echo-spacing=%f`

bValRatioThreshold: (a float)

Sets a threshold which is used to determine b=0 images. When there
are no diffusion weighted image with b-value of zero, then BDP tries
to use diffusion weighted images with a low b-value in place of b=0
image. The diffusion images with minimum b-value is used as b=0
image only if the ratio of the maximum and minimum b-value is more
than the specified threshold. A lower value of threshold will allow
diffusion images with higher b-value to be used as b=0 image. The
default value of this threshold is set to 45, if this trait is not
set.

argument: `--bval-ratio-threshold %f`

estimateTensors: (a boolean)

Estimates diffusion tensors using a weighted log-linear estimation
and saves derived diffusion tensor parameters (FA, MD, axial,
radial, L2, L3). This is the default behavior if no diffusion
modeling flags are specified. The estimated diffusion tensors can be
visualized by loading the saved *.eig.nii.gz file in BrainSuite. BDP
reports diffusivity (MD, axial, radial, L2 and L3) in a unit which
is reciprocal inverse of the unit of input b-value.

argument: `--tensors`

estimateODF_FRACT: (a boolean)

Estimates ODFs using the Funk-Radon and Cosine Transformation
(FRACT). The outputs are saved in a separate directory with name
"FRACT" and the ODFs can be visualized by loading the saved ".odf"
file in BrainSuite.

argument: `--FRACT`

estimateODF_FRT: (a boolean)

Estimates ODFs using Funk-Radon Transformation (FRT). The
coefficient maps for ODFs are saved in a separate directory with
name "FRT" and the ODFs can be visualized by loading the saved ".odf"
file in BrainSuite. The derived generalized-FA (GFA) maps are
also saved in the output directory.

argument: `--FRT`

estimateODF_3DShore: (a float)

Estimates ODFs using 3Dshore. Pass in diffusion time, in ms

argument: `--3dshore --diffusion_time_ms %f`

odfLambda: (a boolean)

Sets the regularization parameter, lambda, of the Laplace-Beltrami
operator while estimating ODFs. The default value is set to 0.006 .
This can be used to set the appropriate regularization for the input
diffusion data.

argument: `--odf-lambda <L>`

t1Mask: (a pathlike object or string representing a file)

Specifies the filename of the brain-mask file for input T1-weighted
image. This mask can be same as the brain mask generated during
BrainSuite extraction sequence. For best results, the mask should
not include any extra-meningeal tissues from T1-weighted image. The
mask must be in the same coordinates as input T1-weighted image (i.e. should overlay correctly with input <fileprefix>.bfc.nii.gz file in BrainSuite). This mask is used for co-registration and defining brain boundary for statistics computation. The mask can be generated and/or edited in BrainSuite. In case outputDiffusionCoordinates is also used, this mask is first transformed to diffusion coordinate and the transformed mask is used for defining brain boundary in diffusion coordinates. When t1Mask is not set, BDP will try to use fileprefix>.mask.nii.gz as brain-mask. If <fileprefix>.mask.nii.gz is not found, then BDP will use the input <fileprefix>.bfc.nii.gz itself as mask (i.e. all non-zero voxels in <fileprefix>.bfc.nii.gz is assumed to constitute brain mask).

argument: '--t1-mask %s'

dwiMask: (a pathlike object or string representing a file)
Specifies the filename of the brain-mask file for diffusion data. This mask is used only for co-registration purposes and can affect overall quality of co-registration (see t1Mask for definition of brain mask for statistics computation). The mask must be a 3D volume and should be in the same coordinates as input Diffusion file/data (i.e. should overlay correctly with input diffusion data in BrainSuite). For best results, the mask should include only brain voxels (CSF voxels around brain is also acceptable). When this flag is not used, BDP will generate a pseudo mask using first b=0 image volume and would save it as fileprefix>.dwi.RSA.mask.nii.gz. In case co-registration is not accurate with automatically generated pseudo mask, BDP should be re-run with a refined diffusion mask. The mask can be generated and/or edited in BrainSuite.

argument: '--dwi-mask %s'

rigidRegMeasure: ('MI' or 'INVERSION' or 'BDP')
Defines the similarity measure to be used for rigid registration. Possible measures are "MI", "INVERSION" and "BDP". MI measure uses normalized mutual information based cost function. INVERSION measure uses simpler cost function based on sum of squared difference by exploiting the approximate inverse-contrast relationship in T1- and T2-weighted images. BDP measure combines MI and INVERSION. It starts with INVERSION measure and refines the result with MI measure. BDP is the default measure when this trait is not set.

argument: '--rigid-reg-measure %s'

dcorrRegMeasure: ('MI' or 'INVERSION-EPI' or 'INVERSION-T1' or 'INVERSION-BOTH' or 'BDP')
Defines the method for registration-based distortion correction. Possible methods are "MI", "INVERSION-EPI", "INVERSION-T1", INVERSION-BOTH", and "BDP". MI method uses normalized mutual information based cost-function while estimating the distortion field. INVERSION-based method uses simpler cost function based on sum of squared difference by exploiting the known approximate contrast relationship in T1- and T2-weighted images. T2-weighted EPI is inverted when INVERSION-EPI is used; T1-image is inverted when INVERSION-T1 is used; and both are inverted when INVERSION-BOTH is used. BDP method add the MI-based refinement after the correction using INVERSION-BOTH method. BDP is the default method when this trait is not set.

argument: '--dcorr-reg-method %s'

dcorrWeight: (a float)
Sets the (scalar) weighting parameter for regularization penalty in registration-based distortion correction. Set this trait to a
single, non-negative number which specifies the weight. A large
regularization weight encourages smoother distortion field at the
cost of low measure of image similarity after distortion correction.
On the other hand, a smaller regularization weight can result into
higher measure of image similarity but with unrealistic and unsmooth
distortion field. A weight of 0.5 would reduce the penalty to half
of the default regularization penalty (By default, this weight is
set to 1.0). Similarly, a weight of 2.0 would increase the penalty
to twice of the default penalty.

argument: `--dcorr-regularization-wt %f`

skipDistortionCorr: (a boolean)
Skips distortion correction completely and performs only a rigid
registration of diffusion and T1-weighted image. This can be useful
when the input diffusion images do not have any distortion or they
have been corrected for distortion.

argument: `--no-distortion-correction`

skipNonuniformityCorr: (a boolean)
Skips intensity non-uniformity correction in b=0 image for
registration-based distortion correction. The intensity non-
uniformity correction does not affect any diffusion modeling.

argument: `--no-nonuniformity-correction`

skipIntensityCorr: (a boolean)
Disables intensity correction when performing distortion correction.
Intensity correction can change the noise distribution in the
corrected image, but it does not affect estimated diffusion
parameters like FA, etc.

argument: `--no-intensity-correction`

mutually_exclusive: fieldmapCorrectionMethod

defines the distortion correction method while using fieldmap.
Possible methods are "pixelshift" and "leastsq". leastsq is the
default method when this flag is not used. Pixel-shift (pixelshift)
method uses image interpolation to un-distort the distorted
diffusion images. Least squares (leastsq) method uses a physical
model of distortion which is more accurate (and more computationally
expensive) than pixel-shift method.
argument: `'--fieldmap-correction-method %s'`

**mutually_exclusive: skipIntensityCorr**

ignoreFieldmapFOV: (a boolean)

Supresses the error generated by an insufficient field of view of the input fieldmap and continues with the processing. It is useful only when used with fieldmap-based distortion correction. See fieldmap-correction for a detailed explanation.

argument: `'--ignore-fieldmap-fov'`

fieldmapSmooth: (a float)

Applies 3D Gaussian smoothing with a standard deviation of S millimeters (mm) to the input fieldmap before applying distortion correction. This trait is only useful with fieldmapCorrection. Skip this trait for no smoothing.

argument: `'--fieldmap-smooth3=%f'`

transformDiffusionVolume: (a pathlike object or string representing a file)

This flag allows to define custom volumes in diffusion coordinate which would be transformed into T1 coordinate in a rigid fashion. The flag must be followed by the name of either a NIfTI file or of a folder that contains one or more NIfTI files. All of the files must be in diffusion coordinate, i.e. the files should overlay correctly with the diffusion scan in BrainSuite. Only NIfTI files with an extension of .nii or .nii.gz are supported. The transformed files are written to the output directory with suffix ".T1_coord" in the filename and will not be corrected for distortion, if any. The trait transformInterpolation can be used to define the type of interpolation that would be used (default is set to linear). If you are attempting to transform a label file or mask file, use "nearest" interpolation method with transformInterpolation. See also transformT1Volume and transformInterpolation.

argument: `'--transform-diffusion-volume %s'`

transformT1Volume: (a pathlike object or string representing a file)

Same as transformDiffusionVolume except that files specified must be in T1 coordinate, i.e. the files should overlay correctly with the input <fileprefix>.bfc.nii.gz files in BrainSuite. BDP transforms these data/images from T1 coordinate to diffusion coordinate. The transformed files are written to the output directory with suffix ".D_coord" in the filename. See also transformDiffusionVolume and transformInterpolation.

argument: `'--transform-t1-volume %s'`

transformInterpolation: ("linear" or 'nearest' or 'cubic' or 'spline')

Defines the type of interpolation method which would be used while transforming volumes defined by transformT1Volume and transformDiffusionVolume. Possible methods are "linear", "nearest", "cubic" and "spline". By default, "linear" interpolation is used.

argument: `'--transform-interpolation %s'`

transformT1Surface: (a pathlike object or string representing a file)

Similar to transformT1Volume, except that this flag allows transforming surfaces (instead of volumes) in T1 coordinate into diffusion coordinate in a rigid fashion. The flag must be followed by the name of either a .dfs file or of a folder that contains one or more dfs files. All of the files must be in T1 coordinate, i.e. the files should overlay correctly with the T1-weighted scan in BrainSuite. The transformed files are written to the output directory with suffix D_coord" in the filename.

argument: `'--transform-t1-surface %s'`
transformDiffusionSurface: (a pathlike object or string representing a file)
   Same as transformT1Volume, except that the .dfs files specified must be in diffusion coordinate, i.e. the surface files should overlay correctly with the diffusion scan in BrainSuite. The transformed files are written to the output directory with suffix ".T1_coord" in the filename. See also transformT1Volume.
   argument: '--transform-diffusion-surface %s'

transformDataOnly: (a boolean)
   Skip all of the processing (co-registration, distortion correction and tensor/ODF estimation) and directly start transformation of defined custom volumes, mask and labels (using transformT1Volume, transformDiffusionVolume, transformT1Surface, transformDiffusionSurface, customDiffusionLabel, customT1Label). This flag is useful when BDP was previously run on a subject (or <fileprefix>) and some more data (volumes, mask or labels) need to be transformed across the T1-diffusion coordinate spaces. This assumes that all the necessary files were generated earlier and all of the other flags MUST be used in the same way as they were in the initial BDP run that processed the data.
   argument: '--transform-data-only'

generateStats: (a boolean)
   Generate ROI-wise statistics of estimated diffusion tensor parameters. Units of the reported statistics are same as that of the estimated tensor parameters (see estimateTensors). Mean, variance, and voxel counts of white matter (WM), grey matter (GM), and both WM and GM combined are written for each estimated parameter in a separate comma-separated value csv) file. BDP uses the ROI labels generated by Surface-Volume Registration (SVReg) in the BrainSuite extraction sequence. Specifically, it looks for labels saved in either fileprefix>.svreg.corr.label.nii.gz or <fileprefix>.svreg.label.nii.gz. In case both files are present, only the first file is used. Also see customDiffusionLabel and customT1Label for specifying your own ROIs. It is also possible to forgo computing the SVReg ROI-wise statistics and only compute stats with custom labels if SVReg label is missing. BDP also transfers (and saves) the label/mask files to appropriate coordinates before computing statistics. Also see outputDiffusionCoordinates for outputs in diffusion coordinate and forcePartialROIStats for an important note about field of view of diffusion and T1-weighted scans.
   argument: '--generate-stats'

onlyStats: (a boolean)
   Skip all of the processing (co-registration, distortion correction and tensor/ODF estimation) and directly start computation of statistics. This flag is useful when BDP was previously run on a subject (or fileprefix>) and statistics need to be (re-)computed later. This assumes that all the necessary files were generated earlier. All of the other flags MUST be used in the same way as they were in the initial BDP run that processed the data.
   argument: '--generate-only-stats'

forcePartialROIStats: (a boolean)
   The field of view (FOV) of the diffusion and T1-weighted scans may differ significantly in some situations. This may result in partial acquisitions of some ROIs in the diffusion scan. By default, BDP does not compute statistics for partially acquired ROIs and shows warnings. This flag forces computation of statistics for all ROIs,
including those which are partially acquired. When this flag is used, number of missing voxels are also reported for each ROI in statistics files. Number of missing voxels are reported in the same coordinate system as the statistics file.

argument: `--force-partial-roi-stats`

customDiffusionLabel: (a pathlike object or string representing a file)

BDP supports custom ROIs in addition to those generated by BrainSuite SVReg for ROI-wise statistics calculation. The flag must be followed by the name of either a file (custom ROI file) or of a folder that contains one or more ROI files. All of the files must be in diffusion coordinate, i.e. the label files should overlay correctly with the diffusion scan in BrainSuite. These input label files are also transferred (and saved) to T1 coordinate for statistics in T1 coordinate. BDP uses nearest-neighborhood interpolation for this transformation. Only NIfTI files, with an extension of .nii or .nii.gz are supported. In order to avoid confusion with other ROI IDs in the statistic files, a 5-digit ROI ID is generated for each custom label found and the mapping of ID to label file is saved in the file fileprefix>.BDP_ROI_MAP.xml. Custom label files can also be generated by using the label painter tool in BrainSuite. See also customLabelXML

argument: `--custom-diffusion-label %s`

customT1Label: (a pathlike object or string representing a file)

Same as customDiffusionLabel except that the label files specified must be in T1 coordinate, i.e. the label files should overlay correctly with the T1-weighted scan in BrainSuite. If the trait outputDiffusionCoordinates is also used then these input label files are also transferred (and saved) to diffusion coordinate for statistics in diffusion coordinate. BDP uses nearest-neighborhood interpolation for this transformation. See also customLabelXML.

argument: `--custom-t1-label %s`

customLabelXML: (a pathlike object or string representing a file)

BrainSuite saves a description of the SVReg labels (ROI name, ID, color, and description) in an .xml file brainsuite_labeldescription.xml. BDP uses the ROI ID{s} from this xml file to report statistics. This flag allows for the use of a custom label description xml file. The flag must be followed by an xml filename. This can be useful when you want to limit the ROIs for which you compute statistics. You can also use custom xml files to name your own ROIs (assign ID{s}) for custom labels. BrainSuite can save a label description in .xml format after using the label painter tool to create a ROI label. The xml file MUST be in the same format as BrainSuite's label description file (see brainsuite_labeldescription.xml for an example). When this flag is used, NO 5-digit ROI ID is generated for custom label files and NO Statistics will be calculated for ROIs not identified in the custom xml file. See also customDiffusionLabel and customT1Label.

argument: `--custom-label-xml %s`

outputSubdir: (a unicode string)

By default, BDP writes out all the output (and intermediate) files in the same directory (or folder) as the BFC file. This flag allows to specify a sub-directory name in which output (and intermediate) files would be written. BDP will create the sub-directory in the same directory as BFC file. <directory_name> should be the name of the sub-directory without any path. This can be useful to organize all outputs generated by BDP in a separate sub-directory.
outputDiffusionCoordinates: (a boolean)
  Enables estimation of diffusion tensors and/or ODFs (and statistics if applicable) in the native diffusion coordinate in addition to the default T1-coordinate. All native diffusion coordinate files are saved in a separate folder named "diffusion_coord_outputs". In case statistics computation is required, it will also transform/save all label/mask files required to diffusion coordinate (see generateStats for details).
  argument: `--output-diffusion-coordinate`

flagConfigFile: (a pathlike object or string representing a file)
  Uses the defined file to specify BDP flags which can be useful for batch processing. A flag configuration file is a plain text file which can contain any number of BDP's optional flags (and their parameters) separated by whitespace. Everything coming after # until end-of-line is treated as comment and is ignored. If a flag is defined in configuration file and is also specified in the command used to run BDP, then the later get preference and overrides the definition in configuration file.
  argument: `--flag-conf-file %s`

outPrefix: (a unicode string)
  Specifies output fileprefix when noStructuralRegistration is used. The fileprefix can not start with a dash (-) and should be a simple string reflecting the absolute path to desired location, along with outPrefix. When this flag is not specified (and noStructuralRegistration is used) then the output files have same file-base as the input diffusion file. This trait is ignored when noStructuralRegistration is not used.
  argument: `--output-fileprefix %s`

threads: (an integer (int or long))
  Sets the number of parallel process threads which can be used for computations to N, where N must be an integer. Default value of N is argument: `--threads=%d`

lowMemory: (a boolean)
  Activates low-memory mode. This will run the registration-based distortion correction at a lower resolution, which could result in a less-accurate correction. This should only be used when no other alternative is available.
  argument: `--low-memory`

ignoreMemory: (a boolean)
  Deactivates the inbuilt memory checks and forces BDP to run registration-based distortion correction at its default resolution even on machines with a low amount of memory. This may result in an out-of-memory error when BDP cannot allocate sufficient memory.
  argument: `--ignore-memory`

args: (a unicode string)
  Additional parameters to the command argument: `%%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: { })
  Environment variables

Outputs:

None
54.1.2 Bfc

Link to code
Wraps the executable command bfc. bias field corrector (BFC) This program corrects gain variation in T1-weighted MRI.
http://brainsuite.org/processing/surfaceextraction/bfc/

Examples

```python
>>> from nipype.interfaces import brainsuite
>>> from nipype.testing import example_data

>>> bfc = brainsuite.Bfc()

>>> bfc.inputs.inputMRIFile = example_data('structural.nii')

>>> bfc.inputs.inputMaskFile = example_data('mask.nii')

>>> results = bfc.run()  # doctest: +SKIP
```

Inputs:

<table>
<thead>
<tr>
<th>Mandatory</th>
<th>inputMRIFile: (a pathlike object or string representing a file)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>input skull-stripped MRI volume</td>
</tr>
<tr>
<td></td>
<td>argument: <code>-i %s</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optional</th>
<th>inputMaskFile: (a pathlike object or string representing a file)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mask file</td>
</tr>
<tr>
<td></td>
<td>argument: <code>-m %s</code></td>
</tr>
</tbody>
</table>

outputMRIVolume: (a pathlike object or string representing a file)
output bias-corrected MRI volume. If unspecified, output file name will be auto generated.
argument: ``-o %s``

outputBiasField: (a pathlike object or string representing a file)
save bias field estimate
argument: ``--bias %s``

outputMaskedBiasField: (a pathlike object or string representing a file)
save bias field estimate (masked)
argument: ``--maskedbias %s``

histogramRadius: (an integer (int or long))
histogram radius (voxels)
argument: ``-r %d``

biasEstimateSpacing: (an integer (int or long))
bias sample spacing (voxels)
argument: ``-s %d``

controlPointSpacing: (an integer (int or long))
control point spacing (voxels)
argument: ``-c %d``

splineLambda: (a float)
spline stiffness weighting parameter
argument: ``-w %f``

histogramType: ("ellipse" or 'block')
Options for type of histogram
ellipse: use ellipsoid for ROI histogram
block :use block for ROI histogram
argument: ``%s``

iterativeMode: (a boolean)
iterative mode (overrides -r, -s, -c, -w settings)
argument: ``--iterate``

(continues on next page)
correctionScheduleFile: (a pathlike object or string representing a file)
    list of parameters
    argument: `--schedule %s`

biasFieldEstimatesOutputPrefix: (a unicode string)
    save iterative bias field estimates as <prefix>.n.field.nii.gz
    argument: `--biasprefix %s`

correctedImagesOutputPrefix: (a unicode string)
    save iterative corrected images as <prefix>.n.bfc.nii.gz
    argument: `--prefix %s`

correctWholeVolume: (a boolean)
    apply correction field to entire volume
    argument: `--extrapolate`

minBias: (a float, nipype default value: 0.5)
    minimum allowed bias value
    argument: `--L %f`

maxBias: (a float, nipype default value: 1.5)
    maximum allowed bias value
    argument: `--U %f`

biasRange: ('low' or 'medium' or 'high')
    Preset options for bias_model
    low: small bias model [0.95,1.05]
    medium: medium bias model [0.90,1.10]
    high: high bias model [0.80,1.20]
    argument: `%s`

intermediate_file_type: ('analyze' or 'nifti' or 'gzippedAnalyze' or 'gzippedNifti')
    Options for the format in which intermediate files are generated
    argument: `%s`

convergenceThreshold: (a float)
    convergence threshold
    argument: `--eps %f`

biasEstimateConvergenceThreshold: (a float)
    bias estimate convergence threshold (values > 0.1 disable)
    argument: `--beps %f`

verbosityLevel: (an integer (int or long))
    verbosity level (0=silent)
    argument: `--v %d`

timer: (a boolean)
    display timing information
    argument: `--timer`

args: (a unicode string)
    Additional parameters to the command
    argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

Environment variables

Outputs:

outputMRIVolume: (a pathlike object or string representing a file)
    path/name of output file

outputBiasField: (a pathlike object or string representing a file)
    path/name of bias field output file

outputMaskedBiasField: (a pathlike object or string representing a file)

(continues on next page)
54.1.3 Bse

Link to code
Wraps the executable command bse, brain surface extractor (BSE) This program performs automated skull and scalp removal on T1-weighted MRI volumes.
http://brainsuite.org/processing/surfaceextraction/bse/

Examples

```python
>>> from nipype.interfaces import brainsuite
>>> from nipype.testing import example_data

bse = brainsuite.Bse()

bse.inputs.inputMRIFile = example_data('structural.nii')

results = bse.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
inputMRIFile: (a pathlike object or string representing a file)
input MRI volume
argument: ```-i %s```

[Optional]
outputMRIVolume: (a pathlike object or string representing a file)
output brain-masked MRI volume. If unspecified, output file name will be auto generated.
argument: ```-o %s```
outputMaskFile: (a pathlike object or string representing a file)
save smooth brain mask. If unspecified, output file name will be auto generated.
argument: ```--mask %s```
diffusionConstant: (a float, nipype default value: 25)
diffusion constant
argument: ```-d %f```
diffusionIterations: (an integer (int or long), nipype default value: 3)
diffusion iterations
argument: ```-n %d```
edgeDetectionConstant: (a float, nipype default value: 0.64)
diffusion constant
argument: ```-s %f```
radius: (a float, nipype default value: 1)
radius of erosion/dilation filter
argument: ```-r %f```
dilateFinalMask: (a boolean, nipype default value: True)
dilate final mask
argument: ```-p```
trim: (a boolean, nipype default value: True)
trim brainstem
argument: `--trim`
outputDiffusionFilter: (a pathlike object or string representing a file)
diffusion filter output
argument: `--adf %s`
outputEdgeMap: (a pathlike object or string representing a file)
edge map output
argument: `--edge %s`
outputDetailedBrainMask: (a pathlike object or string representing a file)
save detailed brain mask
argument: `--hires %s`
outputCortexFile: (a pathlike object or string representing a file)
cortex file
argument: `--cortex %s`
verbosityLevel: (a float, nipype default value: 1)
verbosity level (0=silent)
argument: `--v %f`
noRotate: (a boolean)
retain original orientation(default behavior will auto-rotate input NII files to LPI orientation)
argument: `--norotate`
timer: (a boolean)
show timing
argument: `--timer`
args: (a unicode string)
Additional parameters to the command
argument: `%%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}))
Environment variables

Outputs:

outputMRIVolume: (a pathlike object or string representing a file)
path/name of brain-masked MRI volume
outputMaskFile: (a pathlike object or string representing a file)
path/name of smooth brain mask
outputDiffusionFilter: (a pathlike object or string representing a file)
path/name of diffusion filter output
outputEdgeMap: (a pathlike object or string representing a file)
path/name of edge map output
outputDetailedBrainMask: (a pathlike object or string representing a file)
path/name of detailed brain mask
outputCortexFile: (a pathlike object or string representing a file)
path/name of cortex file

54.1.4 Cerebro

Link to code
Wraps the executable command cerebro.
Cerebrum/cerebellum labeling tool This program performs automated labeling of cerebellum and cerebrum in T1 MRI. Input MRI should be skull-stripped or a brain-only mask should be provided.
http://brainsuite.org/processing/surfaceextraction/cerebrum/
Examples

```python
>>> from nipype.interfaces import brainsuite
>>> from nipype.testing import example_data

>>> cerebro = brainsuite.Cerebro()

>>> cerebro.inputs.inputMRIFile = example_data('structural.nii')

>>> cerebro.inputs.inputAtlasMRIFile = 'atlasMRIVolume.img'

>>> cerebro.inputs.inputAtlasLabelFile = 'atlasLabels.img'

>>> cerebro.inputs.inputBrainMaskFile = example_data('mask.nii')

>>> results = cerebro.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
inputMRIFile: (a pathlike object or string representing a file)
  input 3D MRI volume
  argument: `\'-i %s\'`

inputAtlasMRIFile: (a pathlike object or string representing a file)
  atlas MRI volume
  argument: `\'-atlas %s\'`

inputAtlasLabelFile: (a pathlike object or string representing a file)
  atlas labeling
  argument: `\'-atlaslabels %s\'`

[Optional]
inputBrainMaskFile: (a pathlike object or string representing a file)
  brain mask file
  argument: `\'-m %s\'`

outputCerebrumMaskFile: (a pathlike object or string representing a file)
  output cerebrum mask volume. If unspecified, output file name will be auto generated.
  argument: `\'-o %s\'`

outputLabelVolumeFile: (a pathlike object or string representing a file)
  output labeled hemisphere/cerebrum volume. If unspecified, output file name will be auto generated.
  argument: `\'-l %s\'`

costFunction: (an integer (int or long), nipype default value: 2)
  0, 1, 2
  argument: `\'-c %d\'`

useCentroids: (a boolean)
  use centroids of data to initialize position
  argument: `\'-centroids\'`

outputAffineTransformFile: (a pathlike object or string representing a file)
  save affine transform to file.
  argument: `\'-air %s\'`

outputWarpTransformFile: (a pathlike object or string representing a file)
  save warp transform to file.
  argument: `\'-warp %s\'`

verbosity: (an integer (int or long))
  verbosity level (0=silent)
  argument: `\'-v %d\'`

linearConvergence: (a float)
  linear convergence
54.1.5 Cortex

**Link to code**

Wraps the executable command `cortex`.  
This program produces a cortical mask using tissue fraction estimates and a co-registered cerebellum/hemisphere mask.  
http://brainsuite.org/processing/surfaceextraction/cortex/

**Examples**

```python
>>> from nipype.interfaces import brainsuite
>>> from nipype.testing import example_data
>>> cortex = brainsuite.Cortex()
>>> cortex.inputs.inputHemisphereLabelFile = example_data('mask.nii')
>>> cortex.inputs.inputTissueFractionFile = example_data('tissues.nii.gz')
>>> results = cortex.run()  # doctest: +SKIP
```

**Inputs:**
inputHemisphereLabelFile: (a pathlike object or string representing a file)
  hemisphere / lobe label volume
  argument: ```-h %s```
inputTissueFractionFile: (a pathlike object or string representing a file)
  tissue fraction file (32-bit float)
  argument: ```-f %s```

outputCerebrumMask: (a pathlike object or string representing a file)
  output structure mask. If unspecified, output file name will be auto generated.
  argument: ```-o %s```

tissueFractionThreshold: (a float, nipype default value: 50.0)
  tissue fraction threshold (percentage)
  argument: ```-p %f```
computeWGBoundary: (a boolean, nipype default value: True)
  compute WM/GM boundary
  argument: ```-w```
computeGCBoundary: (a boolean)
  compute GM/CSF boundary
  argument: ```-g```
includeAllSubcorticalAreas: (a boolean, nipype default value: True)
  include all subcortical areas in WM mask
  argument: ```-a```
verbosity: (an integer (int or long))
  verbosity level
  argument: ```-v %d```
timer: (a boolean)
  timing function
  argument: ```--timer```
args: (a unicode string)
  Additional parameters to the command
  argument: ```%s```
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  Environment variables

Outputs:

outputCerebrumMask: (a pathlike object or string representing a file)
  path/name of cerebrum mask

54.1.6 Dewisp

Link to code
Wraps the executable command `dewisp`.
dewisp removes wispy tendril structures from cortex model binary masks. It does so based on graph theoretic analysis of connected components, similar to TCA. Each branch of the structure graph is analyzed to determine pinch points that indicate a likely error in segmentation that attaches noise to the image. The pinch threshold determines how many voxels the cross-section can be before it is considered part of the image.
http://brainsuite.org/processing/surfaceextraction/dewisp/
Examples

```python
>>> from nipype.interfaces import brainsuite
>>> from nipype.testing import example_data

>>> dewisp = brainsuite.Dewisp()

>>> dewisp.inputs.inputMaskFile = example_data('mask.nii')

>>> results = dewisp.run()  # doctest: +SKIP
```

Inputs:

- **inputMaskFile**: (a pathlike object or string representing a file)
  - input file
  - argument: `''-i %s''`

- **outputMaskFile**: (a pathlike object or string representing a file)
  - output file. If unspecified, output file name will be auto generated.
  - argument: `''-o %s''`

- **verbosity**: (an integer (int or long))
  - verbosity
  - argument: `''-v %d''`

- **sizeThreshold**: (an integer (int or long))
  - size threshold
  - argument: `''-t %d''`

- **maximumIterations**: (an integer (int or long))
  - maximum number of iterations
  - argument: `''-n %d''`

- **timer**: (a boolean)
  - time processing
  - argument: `''--timer''`

- **args**: (a unicode string)
  - Additional parameters to the command
  - argument: `''%s''`

- **environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

- **outputMaskFile**: (a pathlike object or string representing a file)
  - path/name of mask file

54.1.7 Dfs

Link to code

Wraps the executable command dfs.

Surface Generator Generates mesh surfaces using an isosurface algorithm.

http://brainsuite.org/processing/surfaceextraction/inner-cortical-surface/

Examples

```python
>>> from nipype.interfaces import brainsuite
>>> from nipype.testing import example_data

>>> dfs = brainsuite.Dfs()
```

(continues on next page)
>>> dfs.inputs.inputVolumeFile = example_data('structural.nii')
>>> results = dfs.run() #doctest: +SKIP

Inputs:

[Mandatory]
inputVolumeFile: (a pathlike object or string representing a file)
  input 3D volume
  argument: '``-i %s``'

[Optional]
outputSurfaceFile: (a pathlike object or string representing a file)
  output surface mesh file. If unspecified, output file name will be
  auto generated.
  argument: '``-o %s``'
inputShadingVolume: (a pathlike object or string representing a file)
  shade surface model with data from image volume
  argument: '``-c %s``'
smoothingIterations: (an integer (int or long), nipype default value: 10)
  number of smoothing iterations
  argument: '``-n %d``'
smoothingConstant: (a float, nipype default value: 0.5)
  smoothing constant
  argument: '``-a %f``'
curvatureWeighting: (a float, nipype default value: 5.0)
  curvature weighting
  argument: '``-w %f``'
scalingPercentile: (a float)
  scaling percentile
  argument: '``-f %f``'
nonZeroTessellation: (a boolean)
  tessellate non-zero voxels
  argument: '``-nz``'
  mutually_exclusive: nonZeroTessellation, specialTessellation
tessellationThreshold: (a float)
  To be used with specialTessellation. Set this value first, then set
  specialTessellation value.
  Usage: tessellate voxels greater_than, less_than, or equal_to
  <tessellationThreshold>
  argument: '``-%f``'
specialTessellation: ('greater_than' or 'less_than' or 'equal_to')
  To avoid throwing a UserWarning, set tessellationThreshold first.
  Then set this attribute.
  Usage: tessellate voxels greater_than, less_than, or equal_to
  <tessellationThreshold>
  argument: '``-%s``, position: -1
  mutually_exclusive: nonZeroTessellation, specialTessellation
  requires: tessellationThreshold
zeroPadFlag: (a boolean)
  zero-pad volume (avoids clipping at edges)
  argument: '``-z``'
noNormalsFlag: (a boolean)
  do not compute vertex normals
  argument: '``--nonormals``'
postSmoothFlag: (a boolean)
  smooth vertices after coloring
**54.1.8 Hemisplit**

Wraps the executable command hemisplit. Hemisphere splitter Splits a surface object into two separate surfaces given an input label volume. Each vertex is labeled left or right based on the labels being odd (left) or even (right). The largest contour on the split surface is then found and used as the separation between left and right.

**Examples**

```python
>>> from nipype.interfaces import brainsuite
>>> from nipype.testing import example_data

>>> hemisplit = brainsuite.Hemisplit()

>>> hemisplit.inputs.inputSurfaceFile = 'input_surf.dfs'

>>> hemisplit.inputs.inputHemisphereLabelFile = 'label.nii'

>>> hemisplit.inputs.pialSurfaceFile = 'pial.dfs'

>>> results = hemisplit.run()  # doctest: +SKIP
```

**Inputs:**

- **inputSurfaceFile**: (a pathlike object or string representing a file)
  - Input surface
  - argument: `''-i %s'`
- **inputHemisphereLabelFile**: (a pathlike object or string representing a file)
  - Input hemisphere label volume
  - argument: `''-l %s'`
- **outputLeftHemisphere**: (a pathlike object or string representing a file)
  - Output surface file, left hemisphere. If unspecified, output file name will be auto generated.
  - argument: `''--left %s'`
- **outputRightHemisphere**: (a pathlike object or string representing a file)
  - Output surface file, right hemisphere. If unspecified, output file name will be auto generated.
  - argument: `''--right %s'`

**Outputs:**

- **outputSurfaceFile**: (a pathlike object or string representing a file)
  - Path/name of surface file
output surface file, right hemisphere. If unspecified, output file name will be auto generated.
argument: `--right %s`
pialSurfaceFile: (a pathlike object or string representing a file)
pial surface file -- must have same geometry as input surface
argument: `-p %s`
outputLeftPialHemisphere: (a pathlike object or string representing a file)
output pial surface file, left hemisphere. If unspecified, output file name will be auto generated.
argument: `--pl %s`
outputRightPialHemisphere: (a pathlike object or string representing a file)
output pial surface file, right hemisphere. If unspecified, output file name will be auto generated.
argument: `--pr %s`
verbosity: (an integer (int or long))
verbosity (0 = silent)
argument: `--v %d`
timer: (a boolean)
timing function
argument: `--timer`
args: (a unicode string)
Additional parameters to the command
argument: `-%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

outputLeftHemisphere: (a pathlike object or string representing a file)
path/name of left hemisphere
outputRightHemisphere: (a pathlike object or string representing a file)
path/name of right hemisphere
outputLeftPialHemisphere: (a pathlike object or string representing a file)
path/name of left pial hemisphere
outputRightPialHemisphere: (a pathlike object or string representing a file)
path/name of right pial hemisphere

54.1.9 Pialmesh

Link to code
Wraps the executable command `pialmesh`.
pialmesh computes a pial surface model using an inner WM/GM mesh and a tissue fraction map.
http://brainsuite.org/processing/surfaceextraction/pial/
Examples

```python
>>> from nipype.interfaces import brainsuite
>>> from nipype.testing import example_data

>>> pialmesh = brainsuite.Pialmesh()

>>> pialmesh.inputs.inputSurfaceFile = 'input_mesh.dfs'

>>> pialmesh.inputs.inputTissueFractionFile = 'frac_file.nii.gz'

>>> pialmesh.inputs.inputMaskFile = example_data('mask.nii')

>>> results = pialmesh.run()  # doctest: +SKIP
```

Inputs:

```yaml
[Mandatory]
inputSurfaceFile: (a pathlike object or string representing a file)
    input file
    argument: '-i %s'

inputTissueFractionFile: (a pathlike object or string representing a file)
    floating point (32) tissue fraction image
    argument: '-f %s'

inputMaskFile: (a pathlike object or string representing a file)
    restrict growth to mask file region
    argument: '-m %s'

[Optional]
outputSurfaceFile: (a pathlike object or string representing a file)
    output file. If unspecified, output file name will be auto generated.
    argument: '-o %s'

verbosity: (an integer (int or long))
    verbosity
    argument: '-v %d'

numIterations: (an integer (int or long), nipype default value: 100)
    number of iterations
    argument: '-n %d'

searchRadius: (a float, nipype default value: 1)
    search radius
    argument: '-r %f'

stepSize: (a float, nipype default value: 0.4)
    step size
    argument: '-s %f'

maxThickness: (a float, nipype default value: 20)
    maximum allowed tissue thickness
    argument: '--max %f'

tissueThreshold: (a float, nipype default value: 1.05)
    tissue threshold
    argument: '-t %f'

outputInterval: (an integer (int or long), nipype default value: 10)
    output interval
    argument: '--interval %d'

exportPrefix: (a unicode string)
    prefix for exporting surfaces if interval is set
    argument: '--prefix %s'

laplacianSmoothing: (a float, nipype default value: 0.025)
    apply Laplacian smoothing
    argument: '--smooth %f'

timer: (a boolean)
    show timing
```

(continues on next page)
54.1.10 Pvc

Link to code
Wraps the executable command pvc.
partial volume classifier (PVC) tool. This program performs voxel-wise tissue classification T1-weighted MRI. Image should be skull-stripped and bias-corrected before tissue classification.
http://brainsuite.org/processing/surfaceextraction/pvc/

Examples

```python
>>> from nipype.interfaces import brainsuite
>>> from nipype.testing import example_data
>>> pvc = brainsuite.Pvc()
>>> pvc.inputs.inputMRIFile = example_data('structural.nii')
>>> pvc.inputs.inputMaskFile = example_data('mask.nii')
>>> results = pvc.run() #doctest: +SKIP
```

Inputs:

[Mandatory]
inputMRIFile: (a pathlike object or string representing a file)
MRI file
argument: ``--i %s``

[Optional]
inputMaskFile: (a pathlike object or string representing a file)
brain mask file
argument: ``--m %s``
outputLabelFile: (a pathlike object or string representing a file)
output label file. If unspecified, output file name will be auto generated.
argument: ``--o %s``
outputTissueFractionFile: (a pathlike object or string representing a file)
file)
output tissue fraction file
argument: ``-f %s``
spatialPrior: (a float)
spatial prior strength
argument: ``-l %f``
verbosity: (an integer (int or long))
verbosity level (0 = silent)
argument: ``-v %d``
threeClassFlag: (a boolean)
use a three-class (CSF=0,GM=1,WM=2) labeling
argument: ``-3``
timer: (a boolean)
time processing
argument: ``--timer``
args: (a unicode string)
Additional parameters to the command
argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

outputLabelFile: (a pathlike object or string representing a file)
path/name of label file
outputTissueFractionFile: (a pathlike object or string representing a
file)
path/name of tissue fraction file

54.1.11 SVReg

Link to code
Wraps the executable command svreg.sh.
surface and volume registration (svreg) This program registers a subject’s BrainSuite-processed volume and
surfaces to an atlas, allowing for automatic labelling of volume and surface ROIs.
For more information, please see: http://brainsuite.org/processing/svreg/usage/

Examples

>>> from nipype.interfaces import brainsuite
>>> svreg = brainsuite.SVReg()
>>> svreg.inputs.subjectFilePrefix = 'home/user/btestsubject/testsubject'
>>> svreg.inputs.refineOutputs = True
>>> svreg.inputs.skipToVolumeReg = False
>>> svreg.inputs. keepIntermediates = True
>>> svreg.inputs. verbosity2 = True
>>> svreg.inputs.displayTimestamps = True
>>> svreg.inputs.useSingleThreading = True
>>> results = svreg.run() #doctest: +SKIP

Inputs:

[Mandatory]
subjectFilePrefix: (a unicode string)
Absolute path and filename prefix of the subjects output from BrainSuite Cortical Surface Extraction Sequence argument: ```'%s'```, position: 0

[Optional]

dataSinkDelay: (a list of items which are a unicode string)

Connect datasink out_file to dataSinkDelay to delay execution of SVReg until dataSink has finished sinking CSE outputs. For use with parallel processing workflows including Brainsuites Cortical Surface Extraction sequence (SVReg requires certain files from Brainsuite CSE, which must all be in the pathway specified by subjectFilePrefix. see http://brainsuite.org/processing/svreg/usage/ for list of required inputs argument: ```'%s'```

atlasFilePrefix: (a unicode string)

Optional: Absolute Path and filename prefix of atlas files and labels to which the subject will be registered. If unspecified, SVReg will use its own included atlas files argument: ```'%s'```, position: 1

iterations: (an integer (int or long))

Assigns a number of iterations in the intensity registration step. If unspecified, performs 100 iterations argument: ```'-%d'```

refineOutputs: (a boolean)

Refine outputs at the expense of more processing time. argument: ```'-r'```

skipToVolumeReg: (a boolean)

If surface registration was already performed at an earlier time and the user would not like to redo this step, then this flag may be used to skip ahead to the volumetric registration. Necessary input files will need to be present in the input directory called by the command. argument: ```'-s'```

skipToIntensityReg: (a boolean)

If the p-harmonic volumetric registration was already performed at an earlier time and the user would not like to redo this step, then this flag may be used to skip ahead to the intensity registration and label transfer step. argument: ```'-p'```

useManualMaskFile: (a boolean)

Can call a manually edited cerebrum mask to limit boundaries. Will use file: subbasename.cerebrum.mask.nii.gz Make sure to correctly replace your manually edited mask file in your input folder with the correct subbasename. argument: ```'-cbm'```

curveMatchingInstructions: (a unicode string)

Used to take control of the curve matching process between the atlas and subject. One can specify the name of the .dfc file <sulname.dfc> and the sulcal numbers <#sul> to be used as constraints. example: curveMatchingInstructions = "subbasename.right.dfc 1 2 20" argument: ```'-cur %s'```

useCerebrumMask: (a boolean)

The cerebrum mask <subbasename.cerebrum.mask.nii.gz> will be used for masking the final labels instead of the default pial surface mask. Every voxel will be labeled within the cerebrum mask regardless of the boundaries of the pial surface. argument: ```'-C'```

(continues on next page)
pialSurfaceMaskDilation: (an integer (int or long))
   Cortical volume labels found in file output
   subbasename.svreg.label.nii.gz find its boundaries by using the pial
   surface then dilating by 1 voxel. Use this flag in order to control
   the number of pial surface mask dilation. (ie. -D 0 will assign no
   voxel dilation)
   argument: `'-D %d'`
keepIntermediates: (a boolean)
   Keep the intermediate files after the svreg sequence is complete.
   argument: `'-k'`
verbosity0: (a boolean)
   no messages will be reported
   argument: `'-v0'`
   mutually_exclusive: verbosity0, verbosity1, verbosity2
verbosity1: (a boolean)
   messages will be reported but not the iteration-wise detailed
   messages
   argument: `'-v1'`
   mutually_exclusive: verbosity0, verbosity1, verbosity2
verbosity2: (a boolean)
   all the messages, including per-iteration, will be displayed
   argument: `'-v2'`
   mutually_exclusive: verbosity0, verbosity1, verbosity2
shortMessages: (a boolean)
   Short messages instead of detailed messages
   argument: `'-gui'`
displayModuleName: (a boolean)
   Module name will be displayed in the messages
   argument: `'-m'`
displayTimestamps: (a boolean)
   Timestamps will be displayed in the messages
   argument: `'-t'`
skipVolumetricProcessing: (a boolean)
   Only surface registration and labeling will be performed. Volumetric
   processing will be skipped.
   argument: `'-S'`
useMultiThreading: (a boolean)
   If multiple CPUs are present on the system, the code will try to use
   multithreading to make the execution fast.
   argument: `'-P'`
useSingleThreading: (a boolean)
   Use single threaded mode.
   argument: `'-U'`
args: (a unicode string)
   Additional parameters to the command
   argument: `'%s'`
environ: (a dictionary with keys which are a bytes or None or a value
   of class 'str' and with values which are a bytes or None or a
   value of class 'str', nipype default value: {})
   Environment variables

Outputs:

None
54.1.12 Scrubmask

Link to code
Wraps the executable command scrubmask.
ScrubMask tool scrubmask filters binary masks to trim loosely connected voxels that may result from segmentation errors and produce bumps on tessellated surfaces.
http://brainsuite.org/processing/surfaceextraction/scrubmask/

Examples

```python
>>> from nipype.interfaces import brainsuite
>>> from nipype.testing import example_data

>>> scrubmask = brainsuite.Scrubmask()
>>> scrubmask.inputs.inputMaskFile = example_data('mask.nii')

>>> results = scrubmask.run()  #doctest: +SKIP
```

Inputs:

[Mandatory]
inputMaskFile: (a pathlike object or string representing a file)
  input structure mask file
  argument: `'-i %s'`

[Optional]
outputMaskFile: (a pathlike object or string representing a file)
  output structure mask file. If unspecified, output file name will be auto generated.
  argument: `'-o %s'`
backgroundFillThreshold: (an integer (int or long), nipype default value: 2)
  background fill threshold
  argument: `'-b %d'`
foregroundTrimThreshold: (an integer (int or long), nipype default value: 0)
  foreground trim threshold
  argument: `'-f %d'`
numberIterations: (an integer (int or long))
  number of iterations
  argument: `'-n %d'`
verbosity: (an integer (int or long))
  verbosity (0=silent)
  argument: `'-v %d'`
timer: (a boolean)
  timing function
  argument: `''--timer''`
args: (a unicode string)
  Additional parameters to the command
  argument: `''%s''`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  Environment variables

Outputs:

outputMaskFile: (a pathlike object or string representing a file)
  path/name of mask file
## 54.1.13 Skullfinder

Link to code
Wraps the executable command `skullfinder`. Skull and scalp segmentation algorithm.

### Examples

```python
>>> from nipype.interfaces import brainsuite
>>> from nipype.testing import example_data

>>> skullfinder = brainsuite.Skullfinder()

>>> skullfinder.inputs.inputMRIFile = example_data('structural.nii')

>>> skullfinder.inputs.inputMaskFile = example_data('mask.nii')

>>> results = skullfinder.run()  #doctest: +SKIP
```

### Inputs:

- **inputMRIFile**: (a pathlike object or string representing a file)
  - input file
  - argument: `--i %s`

- **inputMaskFile**: (a pathlike object or string representing a file)
  - A brain mask file, 8-bit image (0=non-brain, 255=brain)
  - argument: `--m %s`

- **outputLabelFile**: (a pathlike object or string representing a file)
  - output multi-colored label volume segmenting brain, scalp, inner skull & outer skull If unspecified, output file name will be auto generated.
  - argument: `--o %s`

- **verbosity**: (an integer (int or long))
  - verbosity
  - argument: `--v %d`

- **lowerThreshold**: (an integer (int or long))
  - Lower threshold for segmentation
  - argument: `--l %d`

- **upperThreshold**: (an integer (int or long))
  - Upper threshold for segmentation
  - argument: `--u %d`

- **surfaceFilePrefix**: (a unicode string)
  - if specified, generate surface files for brain, skull, and scalp
  - argument: `--s %s`

- **bgLabelValue**: (an integer (int or long))
  - background label value (0-255)
  - argument: `--bglabel %d`

- **scalpLabelValue**: (an integer (int or long))
  - scalp label value (0-255)
  - argument: `--scalplabel %d`

- **skullLabelValue**: (an integer (int or long))
  - skull label value (0-255)
  - argument: `--skulllabel %d`

- **spaceLabelValue**: (an integer (int or long))
  - space label value (0-255)
  - argument: `--spacelabel %d`

- **brainLabelValue**: (an integer (int or long))
  - brain label value (0-255)
  - argument: `--brainlabel %d`

(continues on next page)
**performFinalOpening**: (a boolean)
perform a final opening operation on the scalp mask
argument: `--finalOpening`

**args**: (a unicode string)
Additional parameters to the command
argument: `%-s`

**environ**: (a dictionary with keys which are a bytes or None or a value of class `str` and with values which are a bytes or None or a value of class `str`, nipype default value: `{}`)
Environment variables

### Outputs:

**outputLabelFile**: (a pathlike object or string representing a file)
path/name of label file

## 54.1.14 Tca

**Link to code**
Wraps the executable command tca.

**topological correction algorithm (TCA)** This program removes topological handles from a binary object.
http://brainsuite.org/processing/surfaceextraction/tca/

### Examples

```python
>>> from nipype.interfaces import brainsuite
>>> from nipype.testing import example_data

>>> tca = brainsuite.Tca()

>>> tca.inputs.inputMaskFile = example_data('mask.nii')

>>> results = tca.run()  # doctest: +SKIP
```

### Inputs:

**[Mandatory]**

**inputMaskFile**: (a pathlike object or string representing a file)
input mask volume
argument: `--i %s`

**[Optional]**

**outputMaskFile**: (a pathlike object or string representing a file)
output mask volume. If unspecified, output file name will be automatically generated.
argument: `--o %s`

**minCorrectionSize**: (an integer (int or long), nipype default value: 2500)
maximum correction size
argument: `--m %d`

**maxCorrectionSize**: (an integer (int or long))
minimum correction size
argument: `--n %d`

**foregroundDelta**: (an integer (int or long), nipype default value: 20)
foreground delta
argument: `--delta %d`

**verbosity**: (an integer (int or long))
verbosity (0 = quiet)
argument: `--v %d`
timer: (a boolean)
   timing function
   argument: `''--timer''`
args: (a unicode string)
   Additional parameters to the command
   argument: `''%s''`
environ: (a dictionary with keys which are a bytes or None or a value
   of class 'str' and with values which are a bytes or None or a
   value of class 'str', nipype default value: { })
Environment variables

Outputs:

outputMaskFile: (a pathlike object or string representing a file)
   path/name of mask file

54.1.15 ThicknessPVC

Link to code

ThicknessPVC computes cortical thickness using partial tissue fractions. This thickness measure is then trans-
ferred to the atlas surface to facilitate population studies. It also stores the computed thickness into separate
hemisphere files and subject thickness mapped to the atlas hemisphere surfaces. ThicknessPVC is not run
through the main SVReg sequence, and should be used after executing the BrainSuite and SVReg sequence. For
more information, please see:
http://brainsuite.org/processing/svreg/svreg_modules/

Examples

>>> from nipype.interfaces import brainsuite
>>> thicknessPVC = brainsuite.ThicknessPVC()
>>> thicknessPVC.inputs.subjectFilePrefix = 'home/user/btestsubject/testsubject'
>>> results = thicknessPVC.run() #doctest: +SKIP

Inputs:

[Mandatory]
subjectFilePrefix: (a unicode string)
   Absolute path and filename prefix of the subject data
   argument: `''%s''`

[Optional]
args: (a unicode string)
   Additional parameters to the command
   argument: `''%s''`
environ: (a dictionary with keys which are a bytes or None or a value
   of class 'str' and with values which are a bytes or None or a
   value of class 'str', nipype default value: { })
Environment variables

Outputs:

None
54.1.16 `getFileName()`

Link to code

54.1.17 `l_outputs()`

Link to code
55.1 interfaces.camino.calib

55.1.1 SFLUTGen

Link to code

Wraps the executable command `sflutgen`. Generates PICo lookup tables (LUT) for multi-fibre methods such as PASMRI and Q-Ball. SFLUTGen creates the lookup tables for the generalized multi-fibre implementation of the PICo tractography algorithm. The outputs of this utility are either surface or line coefficients up to a given order. The calibration can be performed for different distributions, such as the Bingham and Watson distributions. This utility uses calibration data generated from SFPICOCalibData and peak information created by SFPeaks. The utility outputs two lut’s, *_oneFibreSurfaceCoeffs.Bdouble and *_twoFibreSurfaceCoeffs.Bdouble. Each of these files contains big-endian doubles as standard. The format of the output is:

```
dimensions   (1 for Watson, 2 for Bingham)
order        (the order of the polynomial)
coefficient_1
coefficient_2
~~~
coefficient_N
```

In the case of the Watson, there is a single set of coefficients, which are ordered:

```
constant, x, x^2, ..., x^order.
```

In the case of the Bingham, there are two sets of coefficients (one for each surface), ordered so that:

```
for j = 1 to order
  for k = 1 to order
    coeff_i = x^j * y^k
  where j+k < order
```

Example

To create a calibration dataset using the default settings
>>> import nipype.interfaces.camino as cam
>>> lutgen = cam.SFLUTGen()
>>> lutgen.inputs.in_file = 'QSH_peaks.Bdouble'
>>> lutgen.inputs.info_file = 'PICO_calib.info'
>>> lutgen.run()  # doctest: +SKIP

Inputs:

[Mandatory]

in_file: (a pathlike object or string representing an existing file)
Voxel-order data of the spherical functions peaks.
argument: ``--inputfile %s``

info_file: (a pathlike object or string representing a file)
The Info file that corresponds to the calibration datafile used in
the reconstruction.
argument: ``--infofile %s``

[Optional]

outputstem: (a unicode string, nipype default value: LUT)
Define the name of the generated luts. The form of the filenames
will be [outputstem]_oneFibreSurfaceCoeffs.Bdouble and
[outputstem]_twoFibreSurfaceCoeffs.Bdouble
argument: ``--outputstem %s``

pdf: (``bingham`` or ``watson``, nipype default value: bingham)
Sets the distribution to use for the calibration. The default is the
Bingham distribution, which allows elliptical probability density
contours. Currently supported options are: bingham - The Bingham
distribution, which allows elliptical probability density contours.
watson - The Watson distribution. This distribution is rotationally
symmetric.
argument: ``--pdf %s``

binincsize: (an integer (int or long))
Sets the size of the bins. In the case of 2D histograms such as the
Bingham, the bins are always square. Default is 1.
argument: ``--binincsize %d``

minvectsperbin: (an integer (int or long))
Specifies the minimum number of fibre-orientation estimates a bin
must contain before it is used in the lut line/surface generation.
Default is 50. If you get the error "no fibre-orientation estimates
in histogram!", the calibration data set is too small to get enough
samples in any of the histogram bins. You can decrease the minimum
number per bin to get things running in quick tests, but the sta-
tistics will not be reliable and for serious applications, you need
to increase the size of the calibration data set until the error
goes.
argument: ``--minvectsperbin %d``

directmap: (a boolean)
Use direct mapping between the eigenvalues and the distribution
parameters instead of the log of the eigenvalues.
argument: ``--directmap``

order: (an integer (int or long))
The order of the polynomial fitting the surface. Order 1 is linear.
Order 2 (default) is quadratic.
argument: ``--order %d``

out_file: (a pathlike object or string representing a file)
argument: ``>%s``, position: -1

args: (a unicode string)
Additional parameters to the command

(continues on next page)
argument: `'\%s'`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

```
lut_one_fibre: (a pathlike object or string representing an existing file)
   PICo lut for one-fibre model
lut_two_fibres: (a pathlike object or string representing an existing file)
   PICo lut for two-fibre model
```

### 55.1.2 SFPICOCalibData

**Link to code**

Wraps the executable command `sfpicocalibdata`.

Generates Spherical Function PICo Calibration Data.

SFPICOCalibData creates synthetic data for use with SFLUTGen. The synthetic data is generated using a mixture of gaussians, in the same way datasynth generates data. Each voxel of data models a slightly different fibre configuration (varying FA and fibre-crossings) and undergoes a random rotation to help account for any directional bias in the chosen acquisition scheme. A second file, which stores information about the datafile, is generated along with the datafile.

**Example 1**

To create a calibration dataset using the default settings

```
>>> import nipype.interfaces.camino as cam
>>> calib = cam.SFPICOCalibData()
>>> calib.inputs.scheme_file = 'A.scheme'
>>> calib.inputs.snr = 20
>>> calib.inputs.info_file = 'PICO_calib.info'
>>> calib.run()  # doctest: +SKIP
```

The default settings create a large dataset (249,231 voxels), of which 3401 voxels contain a single fibre population per voxel and the rest of the voxels contain two fibre-populations. The amount of data produced can be varied by specifying the ranges and steps of the parameters for both the one and two fibre datasets used.

**Example 2**

To create a custom calibration dataset

```
>>> import nipype.interfaces.camino as cam
>>> calib = cam.SFPICOCalibData()
>>> calib.inputs.scheme_file = 'A.scheme'
>>> calib.inputs.snr = 20
>>> calib.inputs.info_file = 'PICO_calib.info'
>>> calib.inputs.twodtfarange = [0.3, 0.9]
>>> calib.inputs.twodtfastep = 0.02
>>> calib.inputs.twodtanglerange = [0, 0.785]
>>> calib.inputs.twodtanglestep = 0.03925
>>> calib.inputs.twodtmixmax = 0.8
>>> calib.inputs.twodtmixstep = 0.1
>>> calib.run()  # doctest: +SKIP
```
This would provide 76,313 voxels of synthetic data, where 3401 voxels simulate the one fibre cases and 72,912 voxels simulate the various two fibre cases. However, care should be taken to ensure that enough data is generated for calculating the LUT. # doctest: +SKIP

Inputs:

[Optional]

```
snr: (a float)
  Specifies the signal-to-noise ratio of the non-diffusion-weighted
  measurements to use in simulations.
  argument: `'-snr %f'``

```
trace: (a float)
  Trace of the diffusion tensor(s) used in the test function.
  argument: `'-trace %f'``

```
twodtfarange: (a list of from 2 to 2 items which are a float)
  Minimum and maximum FA for the two tensor synthetic data.
  argument: `'-twodtfarange %s'``

```
twodtfastep: (a float)
  FA step size controlling how many steps there are between the
  minimum and maximum FA settings.
  argument: `'-twodtfastep %f'``

```
twodtanglerange: (a list of from 2 to 2 items which are a float)
  Minimum and maximum crossing angles between the two fibres.
  argument: `'-twodtanglerange %s'``

```
twodtanglestep: (a float)
  Angle step size controlling how many steps there are between the
  minimum and maximum crossing angles for the two tensor cases.
  argument: `'-twodtanglestep %f'``

```
twodtmixmax: (a float)
  Mixing parameter controlling the proportion of one fibre population
  to the other. The minimum mixing parameter is (1 - twodtmixmax).
  argument: `'-twodtmixmax %f'``

```
twodtmixstep: (a float)
  Mixing parameter step size for the two tensor cases. Specify how
  many mixing parameter increments to use.
  argument: `'-twodtmixstep %f'``

```
seed: (a float)
  Specifies the random seed to use for noise generation in simulation
  trials.
  argument: `'-seed %f'``

```
out_file: (a pathlike object or string representing a file)
  argument: `'> %s', position: -1``
55.2 interfaces.camino.connectivity

55.2.1 Conmat

Link to code
Wraps the executable command `conmat`.

Creates a connectivity matrix using a 3D label image (the target image) and a set of streamlines. The connectivity matrix records how many streamlines connect each pair of targets, and optionally the mean tractwise statistic (e.g., tract-averaged FA, or length).

The output is a comma-separated variable file or files. The first row of the output matrix is label names. Label names may be defined by the user, otherwise they are assigned based on label intensity.

Starting from the seed point, we move along the streamline until we find a point in a labeled region. This is done in both directions from the seed point. Streamlines are counted if they connect two target regions, one on either side of the seed point. Only the labeled region closest to the seed is counted, for example if the input contains two streamlines:

1: A-----B------SEED---C
2: A--------SEED-----------

Then the output would be:

A, B, C
0, 0, 0
0, 0, 1
0, 1, 0

There are zero connections to A because in streamline 1, the connection to B is closer to the seed than the connection to A, and in streamline 2 there is no region reached in the other direction.

The connected target regions can have the same label, as long as the seed point is outside of the labeled region and both ends connect to the same label (which may be in different locations). Therefore this is allowed:

A------SEED-------A

Such fibers will add to the diagonal elements of the matrix. To remove these entries, run `procstreamlines` with `-endpointfile` before running `conmat`.

If the seed point is inside a labeled region, it counts as one end of the connection. So

----[SEED inside A]--------B

counts as a connection between A and B, while
counts as a connection between A and C, because C is closer to the seed point. In all cases, distance to the seed point is defined along the streamline path.

**Example 1**

To create a standard connectivity matrix based on streamline counts.

```python
>>> import nipype.interfaces.camino as cam
>>> conmat = cam.Conmat()
>>> conmat.inputs.in_file = 'tracts.Bdouble'
>>> conmat.inputs.target_file = 'atlas.nii.gz'
>>> conmat.run()  # doctest: +SKIP
```

**Example 1**

To create a standard connectivity matrix and mean tractwise FA statistics.

```python
>>> import nipype.interfaces.camino as cam
>>> conmat = cam.Conmat()
>>> conmat.inputs.in_file = 'tracts.Bdouble'
>>> conmat.inputs.target_file = 'atlas.nii.gz'
>>> conmat.inputs.scalar_file = 'fa.nii.gz'
>>> conmat.tract_stat = 'mean'
>>> conmat.run()  # doctest: +SKIP
```

**Inputs:**

[Mandatory]

- `in_file`: (a pathlike object or string representing an existing file)
  Streamlines as generated by the Track interface
  argument: `''-inputfile %s''`

- `target_file`: (a pathlike object or string representing an existing file)
  An image containing targets, as used in ProcStreamlines interface.
  argument: `''-targetfile %s''`

[Optional]

- `scalar_file`: (a pathlike object or string representing an existing file)
  Optional scalar file for computing tract-based statistics. Must be in the same space as the target file.
  argument: `''-scalarfile %s''`

- `targetname_file`: (a pathlike object or string representing an existing file)
  Optional names of targets. This file should contain one entry per line, with the target intensity followed by the name, separated by white space. For example: 1 some_brain_region 2 some_other_region These names will be used in the output. The names themselves should not contain spaces or commas. The labels may be in any order but the output matrices will be ordered by label intensity.
  argument: `''-targetnamefile %s''`

- `tract_stat`: (`'mean'` or `'min'` or `'max'` or `'sum'` or `'median'` or `'var'`)
  Tract statistic to use. See TractStats for other options.
  argument: `''-tractstat %s''`

mutually_exclusive: `tract_prop`

(continues on next page)
requires: scalar_file
tract_prop: ('length' or 'endpointsep')
   Tract property average to compute in the connectivity matrix. See
   TractStats for details.
   argument: ``--tractstat %s``
   mutually exclusive: tract_stat
output_root: (a pathlike object or string representing a file)
   filename root prepended onto the names of the output files. The
   extension will be determined from the input.
   argument: ``--outputroot %s``
args: (a unicode string)
   Additional parameters to the command
   argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
   of class 'str' and with values which are a bytes or None or a
   value of class 'str', nipype default value: {})
   Environment variables

Outputs:

conmat_sc: (a pathlike object or string representing an existing
           file)
   Connectivity matrix in CSV file.
conmat_ts: (a pathlike object or string representing a file)
   Tract statistics in CSV file.

55.3 interfaces.camino.convert

55.3.1 AnalyzeHeader

Link to code
Wraps the executable command analyzeheader.
Create or read an Analyze 7.5 header file.
Analyze image header, provides support for the most common header fields. Some fields, such as patient_id, are
not currently supported. The program allows three nonstandard options: the field image_dimension.funused1
is the image scale. The intensity of each pixel in the associated .img file is (image value from file) * scale.
Also, the origin of the Talairach coordinates (midline of the anterior commissure) are encoded in the field
data_history.originator. These changes are included for compatibility with SPM.
All headers written with this program are big endian by default.

Example

```
>>> import nipype.interfaces.camino as cmon
>>> hdr = cmon.AnalyzeHeader()
>>> hdr.inputs.in_file = 'tensor_fitted_data.Bdouble'
>>> hdr.inputs.scheme_file = 'A.scheme'
>>> hdr.inputs.data_dims = [256,256,256]
>>> hdr.inputs.voxel_dims = [1,1,1]
>>> hdr.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
   Tensor-fitted data filename

(continues on next page)
 nipype Documentation, Release 1.2.1

(continued from previous page)

argument: `"< %s"`, position: 1
datatype: ('byte' or 'char' or '[u]short' or '[u]int' or 'float' or 'complex' or 'double')
The char datatype is 8 bit (not the 16 bit char of Java), as specified by the Analyze 7.5 standard. The byte, ushort and uint types are not part of the Analyze specification but are supported by SPM.
argument: `"-datatype %s"

[Optional]
scheme_file: (a pathlike object or string representing an existing file)
Camino scheme file (b values / vectors, see camino.fsl2scheme)
argument: `"%s"`, position: 2
readheader: (a pathlike object or string representing an existing file)
Reads header information from file and prints to stdout. If this option is not specified, then the program writes a header based on the other arguments.
argument: `"-readheader %s"`, position: 3
printimagedims: (a pathlike object or string representing an existing file)
Prints image data and voxel dimensions as Camino arguments and exits.
argument: `"-printimagedims %s"`, position: 3
printprogargs: (a pathlike object or string representing an existing file)
Prints data dimension (and type, if relevant) arguments for a specific Camino program, where prog is one of shredder, scanner2voxel, vcthreshselect, pdview, track.
argument: `"-printprogargs %s"`, position: 3
printintelbyteorder: (a pathlike object or string representing an existing file)
Prints 1 if the header is little-endian, 0 otherwise.
argument: `"-printintelbyteorder %s"`, position: 3
printbigendian: (a pathlike object or string representing an existing file)
Prints 1 if the header is big-endian, 0 otherwise.
argument: `"-printbigendian %s"`, position: 3
initfromheader: (a pathlike object or string representing an existing file)
Reads header information from file and initializes a new header with the values read from the file. You may replace any combination of fields in the new header by specifying subsequent options.
argument: `"-initfromheader %s"`, position: 3
data_dims: (a list of from 3 to 3 items which are an integer (int or long))
data dimensions in voxels
argument: `"-datadims %s"
voxel_dims: (a list of from 3 to 3 items which are a float)
voxel dimensions in mm
argument: `"-voxeldims %s"
centre: (a list of from 3 to 3 items which are an integer (int or long))
Voxel specifying origin of Talairach coordinate system for SPM, default [0 0 0].
argument: `"-centre %s"

(continues on next page)
picoseed: (a list of from 3 to 3 items which are an integer (int or long))
  Voxel specifying the seed (for PICo maps), default [0 0 0].
  argument: `--picoseed %s`

nimages: (an integer (int or long))
  Number of images in the img file. Default 1.
  argument: `--nimages %d`

offset: (an integer (int or long))
  According to the Analyze 7.5 standard, this is the byte offset in
  the .img file at which voxels start. This value can be negative to
  specify that the absolute value is applied for every image in the
  file.
  argument: `--offset %d`

greylevels: (a list of from 2 to 2 items which are an integer (int or long))
  Minimum and maximum greylevels. Stored as shorts in the header.
  argument: `--gl %s`

scaleslope: (a float)
  Intensities in the image are scaled by this factor by SPM and
  MRICro. Default is 1.0.
  argument: `--scaleslope %d`

scaleinter: (a float)
  Constant to add to the image intensities. Used by SPM and MRICro.
  argument: `--scaleinter %d`

description: (a string)
  Short description - No spaces, max length 79 bytes. Will be null
  terminated automatically.
  argument: `--description %s`

intelbyteorder: (a boolean)
  Write header in intel byte order (little-endian).
  argument: `--intelbyteorder`

networkbyteorder: (a boolean)
  Write header in network byte order (big-endian). This is the default
  for new headers.
  argument: `--networkbyteorder`

out_file: (a pathlike object or string representing a file)
  argument: `>%s`, position: -1

args: (a unicode string)
  Additional parameters to the command
  argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
  Environment variables

header: (a pathlike object or string representing an existing file)
  Analyze header

55.3.2 DT2NIfTI

Link to code
Wraps the executable command dt2nii.
Converts camino tensor data to NIfTI format
Reads Camino diffusion tensors, and converts them to NIFTI format as three .nii files.

Inputs:
[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  tract file
  argument: ``-inputfile %s``, position: 1
header_file: (a pathlike object or string representing an existing file)
  A Nifti .nii or .hdr file containing the header information
  argument: ``-header %s``, position: 3

[Optional]
output_root: (a pathlike object or string representing a file)
  filename root prepended onto the names of three output files.
  argument: ``-outputroot %s``, position: 2
args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

Outputs:

- **dt**: (a pathlike object or string representing an existing file)
  diffusion tensors in NIfTI format
- **exitcode**: (a pathlike object or string representing an existing file)
  exit codes from Camino reconstruction in NIfTI format
- **lns0**: (a pathlike object or string representing an existing file)
  estimated lns0 from Camino reconstruction in NIfTI format

### 55.3.3 Image2Voxel

Link to code
Wraps the executable command `image2voxel`. Converts Analyze / NIFTI / MHA files to voxel order. Converts scanner-order data in a supported image format to voxel-order data. Either takes a 4D file (all measurements in single image) or a list of 3D images.

#### Examples

```python
>>> import nipype.interfaces.camino as cmon
>>> img2vox = cmon.Image2Voxel()
>>> img2vox.inputs.in_file = '4d_dwi.nii'
>>> img2vox.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  4d image file
  argument: ``-4dimage %s``, position: 1

[Optional]
out_type: ('float' or 'char' or 'short' or 'int' or 'long' or 'double', nipype default value: float)
  "i.e. Bfloat". Can be "char", "short", "int", "long", "float" or (continues on next page)
"double"
argument: `'-outputdatatype %s'`, position: 2

out_file: (a pathlike object or string representing a file)
argument: `'> %s'`, position: -1

args: (a unicode string)
Additional parameters to the command
argument: `'%s'

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

voxel_order: (a pathlike object or string representing an existing
file)
path/name of 4D volume in voxel order

55.3.4 NIfTIDT2Camino

Link to code
Wraps the executable command `niftidt2camino`.
Converts NIFTI-1 diffusion tensors to Camino format. The program reads the NIFTI header but does not apply
any spatial transformations to the data. The NIFTI intensity scaling parameters are applied.
The output is the tensors in Camino voxel ordering: [exit, ln(S0), dxx, dxy, dxz, dyy, dyz, dzz].
The exit code is set to 0 unless a background mask is supplied, in which case the code is 0 in brain voxels and
-1 in background voxels.
The value of ln(S0) in the output is taken from a file if one is supplied, otherwise it is set to 0.
NOTE FOR FSL USERS - FSL's dtifit can output NIFTI tensors, but they are not stored in the usual way (which
is using NIFTI_INTENT_SYMMATRIX). FSL's tensors follow the ITK / VTK “upper-triangular” convention,
so you will need to use the -uppertriangular option to convert these correctly.

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
A NIFTI-1 dataset containing diffusion tensors. The tensors are
assumed to be in lower-triangular order as specified by the NIFTI
standard for the storage of symmetric matrices. This file should be
either a .nii or a .hdr file.
argument: `'-inputfile %s'`, position: 1

[Optional]
s0_file: (a pathlike object or string representing an existing file)
File containing the unweighted signal for each voxel, may be a raw
binary file (specify type with -inputdatatype) or a supported image
file.
argument: `'-s0 %s'

lns0_file: (a pathlike object or string representing an existing
file)
File containing the log of the unweighted signal for each voxel, may
be a raw binary file (specify type with -inputdatatype) or a
supported image file.
argument: `'-lns0 %s'

bgmask: (a pathlike object or string representing an existing file)
Binary valued brain / background segmentation, may be a raw binary
file (specify type with -maskdatatype) or a supported image file.
argument: ``-bgmask %s``
scaleslope: (a float)
   A value v in the diffusion tensor is scaled to v * s + i. This is
   applied after any scaling specified by the input image. Default is
   1.0.
   argument: ``-scaleslope %s``
scaleinter: (a float)
   A value v in the diffusion tensor is scaled to v * s + i. This is
   applied after any scaling specified by the input image. Default is
   0.0.
   argument: ``-scaleinter %s``
uppertriangular: (a boolean)
   Specifies input in upper-triangular (VTK style) order.
   argument: ``-uppertriangular %s``
out_file: (a pathlike object or string representing a file)
   argument: ``%s``, position: -1
args: (a unicode string)
   Additional parameters to the command
   argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
   of class 'str' and with values which are a bytes or None or a value
   of class 'str', nipype default value: {}) Environment variables

Outputs:

out_file: (a pathlike object or string representing a file)
   diffusion tensors data in Camino format

55.3.5 ProcStreamlines

Link to code
Wraps the executable command procstreamlines.
Process streamline data
This program does post-processing of streamline output from track. It can either output streamlines or connection probability maps.


Examples

```python
>>> import nipype.interfaces.camino as cmon
>>> proc = cmon.ProcStreamlines()
>>> proc.inputs.in_file = 'tract_data.Bfloat'
>>> proc.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
   data file
   argument: ``-inputfile %s``, position: 1

[Optional]
inputmodel: ('raw' or 'voxels', nipype default value: raw)
   input model type (raw or voxels)
   argument: ``-inputmodel %s``

(continues on next page)
maxtractpoints: (an integer (int or long))
    maximum number of tract points
    argument: `--maxtractpoints %d`
mintractpoints: (an integer (int or long))
    minimum number of tract points
    argument: `--mintractpoints %d`
maxtractlength: (an integer (int or long))
    maximum length of tracts
    argument: `--maxtractlength %d`
mintractlength: (an integer (int or long))
    minimum length of tracts
    argument: `--mintractlength %d`
datadims: (a list of from 3 to 3 items which are an integer (int or long))
    data dimensions in voxels
    argument: `--datadims %s`
voxeldims: (a list of from 3 to 3 items which are an integer (int or long))
    voxel dimensions in mm
    argument: `--voxeldims %s`
seedpointmm: (a list of from 3 to 3 items which are an integer (int or long))
    The coordinates of a single seed point for tractography in mm
    argument: `--seedpointmm %s`
seedpointvox: (a list of from 3 to 3 items which are an integer (int or long))
    The coordinates of a single seed point for tractography in voxels
    argument: `--seedpointvox %s`
seedfile: (a pathlike object or string representing a file)
    Image Containing Seed Points
    argument: `--seedfile %s`
regionindex: (an integer (int or long))
    index of specific region to process
    argument: `--regionindex %d`
iterations: (a float)
    Number of streamlines generated for each seed. Not required when
    outputting streamlines, but needed to create PICo images. The
    default is 1 if the output is streamlines, and 5000 if the output is
    connection probability images.
    argument: `--iterations %d`
targetfile: (a pathlike object or string representing a file)
    Image containing target volumes.
    argument: `--targetfile %s`
allowmultitargets: (a boolean)
    Allows streamlines to connect to multiple target volumes.
    argument: `--allowmultitargets`
directional: (a list of from 3 to 3 items which are an integer (int or long))
    Splits the streamlines at the seed point and computes separate
    connection probabilities for each segment. Streamline segments are
    grouped according to their dot product with the vector (X, Y, Z).
    The ideal vector will be tangential to the streamline trajectory at
    the seed, such that the streamline projects from the seed along (X,
    Y, Z) and -(X, Y, Z). However, it is only necessary for the
    streamline trajectory to not be orthogonal to (X, Y, Z).
    argument: `--directional %s`
waypointfile: (a pathlike object or string representing a file)
Image containing waypoints. Waypoints are defined as regions of the image with the same intensity, where 0 is background and any value > 0 is a waypoint.

argument: `--waypointfile %s`

truncateloops: (a boolean)
This option allows streamlines to enter a waypoint exactly once. After the streamline leaves the waypoint, it is truncated upon a second entry to the waypoint.

argument: `--truncateloops`

discardloops: (a boolean)
This option allows streamlines to enter a waypoint exactly once. After the streamline leaves the waypoint, the entire streamline is discarded upon a second entry to the waypoint.

argument: `--discardloops`

exclusionfile: (a pathlike object or string representing a file)
Image containing exclusion ROIs. This should be an Analyze 7.5 header / image file.hdr and file.img.

argument: `--exclusionfile %s`

truncateinexclusion: (a boolean)
Retain segments of a streamline before entry to an exclusion ROI.

argument: `--truncateinexclusion`

endpointfile: (a pathlike object or string representing a file)
Image containing endpoint ROIs. This should be an Analyze 7.5 header / image file.hdr and file.img.

argument: `--endpointfile %s`

resamplestepsize: (a float)
Each point on a streamline is tested for entry into target, exclusion or waypoint volumes. If the length between points on a tract is not much smaller than the voxel length, then streamlines may pass through part of a voxel without being counted. To avoid this, the program resamples streamlines such that the step size is one tenth of the smallest voxel dimension in the image. This increases the size of raw or oogl streamline output and incurs some performance penalty. The resample resolution can be controlled with this option or disabled altogether by passing a negative step size or by passing the `--noresample` option.

argument: `--resamplestepsize %d`

noresample: (a boolean)
Disables resampling of input streamlines. Resampling is automatically disabled if the input model is voxels.

argument: `--noresample`

outputtracts: (a boolean)
Output streamlines in raw binary format.

argument: `--outputtracts`

outputroot: (a pathlike object or string representing a file)
Prepended onto all output file names.

argument: `--outputroot %s`

gzip: (a boolean)
Save the output image in gzip format.

argument: `--gzip`

outputcp: (a boolean)
Output the connection probability map (Analyze image, float)

argument: `--outputcp`

requires: outputroot, seedfile

outputsc: (a boolean)
Output the connection probability map (raw streamlines, int)

argument: `--outputsc`
requires: outputroot, seedfile
outputacm: (a boolean)
    output all tracts in a single connection probability map (Analyze image)
    argument: `\"-outputacm\"`
    requires: outputroot, seedfile
outputcbs: (a boolean)
    outputs connectivity-based segmentation maps; requires target
    outputfile
    argument: `\"-outputcbs\"`
    requires: outputroot, targetfile, seedfile
out_file: (a pathlike object or string representing a file)
    argument: `\"> %s\``), position: -1
args: (a unicode string)
    Additional parameters to the command
    argument: `\"%s\``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {}) Environment variables

Outputs:

proc: (a pathlike object or string representing an existing file)
    Processed Streamlines
outputroot_files: (a list of items which are a pathlike object or
    string representing an existing file)

55.3.6 Shredder

Link to code
Wraps the executable command shredder.
Extracts periodic chunks from a data stream.
Shredder makes an initial offset of offset bytes. It then reads and outputs chunksize bytes, skips space bytes, and repeats until there is no more input.
If the chunksize is negative, chunks of size chunksize are read and the byte ordering of each chunk is reversed.
The whole chunk will be reversed, so the chunk must be the same size as the data type, otherwise the order of the values in the chunk, as well as their endianness, will be reversed.

Examples

```python
>>> import nipype.interfaces.camino as cam
>>> shred = cam.Shredder()
>>> shred.inputs.in_file = 'SubjectA.Bfloat'
>>> shred.inputs.offset = 0
>>> shred.inputs.chunksize = 1
>>> shred.inputs.space = 2
>>> shred.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
    raw binary data file
    argument: `\"< %s\``), position: -2

(continues on next page)
55.3.7 TractShredder

Link to code

Wraps the executable command tractshredder. Extracts bunches of streamlines.

tractshredder works in a similar way to shredder, but processes streamlines instead of scalar data. The input is raw streamlines, in the format produced by track or procstreamlines.

The program first makes an initial offset of offset tracts. It then reads and outputs a group of bunchsize tracts, skips space tracts, and repeats until there is no more input.

Examples

```python
>>> import nipype.interfaces.camino as cmon
>>> shred = cmon.TractShredder()
>>> shred.inputs.in_file = 'tract_data.Bfloat'
>>> shred.inputs.offset = 0
>>> shred.inputs.bunchsize = 1
>>> shred.inputs.space = 2
>>> shred.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]

in_file: (a pathlike object or string representing an existing file)

  tract file
  argument: `'< %s', position: -2

[Optional]

offset: (an integer (int or long))

  initial offset of offset tracts
  argument: `''%d'`, position: 1

chunksize: (an integer (int or long))

  reads and outputs a chunk of chunksize bytes
  argument: `''%d'`, position: 2

space: (an integer (int or long))

  skips space tracts
  argument: `''%d'`, position: 3

out_file: (a pathlike object or string representing a file)

  argument: `''> %s'`, position: -1

args: (a unicode string)

  Additional parameters to the command
  argument: `''%s'`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

Environment variables

Outputs:

shredded: (a pathlike object or string representing an existing file)

  Shredded binary data file
argument: ``%d``, position: 1
bunchsize: (an integer (int or long))
reads and outputs a group of bunchsize tracts
argument: ``%d``, position: 2
space: (an integer (int or long))
skips space tracts
argument: ``%d``, position: 3
out_file: (a pathlike object or string representing a file)
argument: ``> %s``, position: -1
args: (a unicode string)
Additional parameters to the command
argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {}))
Environment variables

Outputs:

shredded: (a pathlike object or string representing an existing file)
  Shredded tract file

55.3.8 VtkStreamlines

Link to code
Wraps the executable command vtkstreamlines.
Use vtkstreamlines to convert raw or voxel format streamlines to VTK polydata

Examples

```python
>>> import nipype.interfaces.camino as cmon
>>> vtk = cmon.VtkStreamlines()
>>> vtk.inputs.in_file = 'tract_data.Bfloat'
>>> vtk.inputs.voxeldims = [1,1,1]
>>> vtk.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  data file
  argument: ``< %s``, position: -2

[Optional]
inputmodel: ('raw' or 'voxels', nipype default value: raw)
  input model type (raw or voxels)
  argument: ``-inputmodel %s``
voxeldims: (a list of from 3 to 3 items which are an integer (int or
  long))
  voxel dimensions in mm
  argument: ``-voxeldims %s``, position: 4
seed_file: (a pathlike object or string representing a file)
  image containing seed points
  argument: ``-seedfile %s``, position: 1
target_file: (a pathlike object or string representing a file)
  image containing integer-valued target regions

(continues on next page)
argumen: `--targetfile %s`, position: 2
scalar_file: (a pathlike object or string representing a file)
   image that is in the same physical space as the tracts
   argument: `--scalarfile %s`, position: 3
colourorient: (a boolean)
   Each point on the streamline is coloured by the local orientation.
   argument: `--colourorient`
interpolatescalars: (a boolean)
   the scalar value at each point on the streamline is calculated by trilinear interpolation
   argument: `--interpolatescalars`
interpolate: (a boolean)
   the scalar value at each point on the streamline is calculated by trilinear interpolation
   argument: `--interpolate`
out_file: (a pathlike object or string representing a file)
   argument: `>` %s`, position: -1
args: (a unicode string)
   Additional parameters to the command
   argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

vtk: (a pathlike object or string representing an existing file)
   Streamlines in VTK format

55.4 interfaces.camino.dti

55.4.1 ComputeEigensystem

Link to code
Wraps the executable command dteig.
Computes the eigensystem from tensor fitted data.
Reads diffusion tensor (single, two-tensor, three-tensor or multitensor) data from the standard input, computes the eigenvalues and eigenvectors of each tensor and outputs the results to the standard output. For multiple-tensor data the program outputs the eigensystem of each tensor. For each tensor the program outputs: {l_1, e_11, e_12, e_13, l_2, e_21, e_22, e_23, l_3, e_31, e_32, e_33}, where l_1 >= l_2 >= l_3 and e_i = (e_i1, e_i2, e_i3) is the eigenvector with eigenvalue l_i. For three-tensor data, for example, the output contains thirty-six values per voxel.

Example

```python
>>> import nipype.interfaces.camino as cmon
>>> dteig = cmon.ComputeEigensystem()
>>> dteig.inputs.in_file = 'tensor_fitted_data.Bdouble'
>>> dteig.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
Tensor-fitted data filename
argument: `"< %s"`, position: 1

[Optional]
inputmodel: ('dt' or 'multitensor')
  Specifies the model that the input data contains parameters for.
  Possible model types are: "dt" (diffusion-tensor data) and "multitensor"
  argument: `"-inputmodel %s"
maxcomponents: (an integer (int or long))
  The maximum number of tensor components in a voxel of the input data.
  argument: `"-maxcomponents %d"
inputdatatype: ('double' or 'float' or 'long' or 'int' or 'short' or 'char', nipype default value: double)
  Specifies the data type of the input data. The data type can be any of the following strings: "char", "short", "int", "long", "float" or "double". Default is double data type
  argument: `"-inputdatatype %s"
outputdatatype: ('double' or 'float' or 'long' or 'int' or 'short' or 'char', nipype default value: double)
  Specifies the data type of the output data. The data type can be any of the following strings: "char", "short", "int", "long", "float" or "double". Default is double data type
  argument: `"-outputdatatype %s"
out_file: (a pathlike object or string representing a file)
  argument: `"> %s"`, position: -1
args: (a unicode string)
  Additional parameters to the command
  argument: `"%s"
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: { })
  Environment variables

Outputs:

eigen: (a pathlike object or string representing an existing file)
  Trace of the diffusion tensor

### 55.4.2 ComputeFractionalAnisotropy

**Link to code**

Wraps the executable command `fa`.

Computes the fractional anisotropy of tensors.

Reads diffusion tensor (single, two-tensor or three-tensor) data from the standard input, computes the fractional anisotropy (FA) of each tensor and outputs the results to the standard output. For multiple-tensor data the program outputs the FA of each tensor, so for three-tensor data, for example, the output contains three fractional anisotropy values per voxel.

**Example**

```python
>>> import nipype.interfaces.camino as cmon
>>> fa = cmon.ComputeFractionalAnisotropy()
>>> fa.inputs.in_file = 'tensor_fitted_data.Bdouble'
```

(continues on next page)
>>> fa.inputs.scheme_file = 'A.scheme'
>>> fa.run() # doctest: +SKIP

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  Tensor-fitted data filename
  argument: ``< %s`` , position: 1

[Optional]
scheme_file: (a pathlike object or string representing an existing file)
  Camino scheme file (b values / vectors, see camino.fsl2scheme)
  argument: ``%s`` , position: 2

inputmodel: ('dt' or 'twotensor' or 'threetensor' or 'multitensor')
  Specifies the model that the input tensor data contains parameters for.
  Possible model types are: "dt" (diffusion-tensor data),
  "twotensor" (two-tensor data), "threetensor" (three-tensor data).
  By default, the program assumes that the input data contains a single
  diffusion tensor in each voxel.
  argument: ``-inputmodel %s``

inputdatatype: ('char' or 'short' or 'int' or 'long' or 'float' or 'double')
  Specifies the data type of the input file. The data type can be any
  of the following strings: "char", "short", "int", "long", "float" or
  "double".
  argument: ``-inputdatatype %s``

outputdatatype: ('char' or 'short' or 'int' or 'long' or 'float' or 'double')
  Specifies the data type of the output data. The data type can be any
  of the following strings: "char", "short", "int", "long", "float" or
  "double".
  argument: ``-outputdatatype %s``

out_file: (a pathlike object or string representing a file)
  argument: ``> %s`` , position: -1

args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
  Environment variables

Outputs:

fa: (a pathlike object or string representing an existing file)
  Fractional Anisotropy Map

55.4.3 ComputeMeanDiffusivity

Link to code
Wraps the executable command md.
Computes the mean diffusivity (trace/3) from diffusion tensors.
Example

```python
>>> import nipype.interfaces.camino as cmon
>>> md = cmon.ComputeMeanDiffusivity()
>>> md.inputs.in_file = 'tensor_fitted_data.Bdouble'
>>> md.inputs.scheme_file = 'A.scheme'
>>> md.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]

in_file: (a pathlike object or string representing an existing file)
  Tensor-fitted data filename
  argument: `''< %s''`, position: 1

[Optional]

scheme_file: (a pathlike object or string representing an existing file)
  Camino scheme file (b values / vectors, see camino.fsl2scheme)
  argument: `''%s''`, position: 2

out_file: (a pathlike object or string representing a file)
  argument: `''> %s''`, position: -1

inputmodel: ('dt' or 'twotensor' or 'threetensor')
  Specifies the model that the input tensor data contains parameters for.
  Possible model types are: "dt" (diffusion-tensor data), "twotensor"
  (two-tensor data), "threetensor" (three-tensor data). By default,
  the program assumes that the input data contains a single
  diffusion tensor in each voxel.
  argument: `''-inputmodel %s''`

inputdatatype: ('char' or 'short' or 'int' or 'long' or 'float' or 'double')
  Specifies the data type of the input file. The data type can be any
  of the following strings: "char", "short", "int", "long", "float" or
  "double".
  argument: `''-inputdatatype %s''`

outputdatatype: ('char' or 'short' or 'int' or 'long' or 'float' or 'double')
  Specifies the data type of the output data. The data type can be any
  of the following strings: "char", "short", "int", "long", "float" or
  "double".
  argument: `''-outputdatatype %s''`

args: (a unicode string)
  Additional parameters to the command
  argument: `''%s''`

environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
  Environment variables

Outputs:

md: (a pathlike object or string representing an existing file)
  Mean Diffusivity Map

55.4.4 ComputeTensorTrace

Link to code
Wraps the executable command trd. Computes the trace of tensors.
Reads diffusion tensor (single, two-tensor or three-tensor) data from the standard input, computes the trace of each tensor, i.e., three times the mean diffusivity, and outputs the results to the standard output. For multiple-tensor data the program outputs the trace of each tensor, so for three-tensor data, for example, the output contains three values per voxel. Divide the output by three to get the mean diffusivity.

Example

```python
>>> import nipype.interfaces.camino as cmon
>>> trace = cmon.ComputeTensorTrace()
>>> trace.inputs.in_file = 'tensor_fitted_data.Bdouble'
>>> trace.inputs.scheme_file = 'A.scheme'
>>> trace.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
- `in_file`: (a pathlike object or string representing an existing file)
  - Tensor-fitted data filename
    - argument: `'' < %s''`, position: 1

[Optional]
- `scheme_file`: (a pathlike object or string representing an existing file)
  - Camino scheme file (b values / vectors, see camino.fsl2scheme)
    - argument: `'' %s''`, position: 2
- `inputmodel`: ('dt' or 'twotensor' or 'threetensor' or 'multitensor')
  - Specifies the model that the input tensor data contains parameters for. Possible model types are: "dt" (diffusion-tensor data), "twotensor" (two-tensor data), "threetensor" (three-tensor data). By default, the program assumes that the input data contains a single diffusion tensor in each voxel.
    - argument: `''-inputmodel %s''`
- `inputdatatype`: ('char' or 'short' or 'int' or 'long' or 'float' or 'double')
  - Specifies the data type of the input file. The data type can be any of the following strings: "char", "short", "int", "long", "float" or "double".
    - argument: `''-inputdatatype %s''`
- `outputdatatype`: ('char' or 'short' or 'int' or 'long' or 'float' or 'double')
  - Specifies the data type of the output data. The data type can be any of the following strings: "char", "short", "int", "long", "float" or "double".
    - argument: `''-outputdatatype %s''`
- `out_file`: (a pathlike object or string representing a file)
  - argument: `'' > %s''`, position: -1
- `args`: (a unicode string)
  - Additional parameters to the command
    - argument: `'' %s''`
- `environ`: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
trace: (a pathlike object or string representing an existing file)
  Trace of the diffusion tensor

55.4.5 DTIFit

Link to code
Wraps the executable command dtfit.
Reads diffusion MRI data, acquired using the acquisition scheme detailed in the scheme file, from the data file. Use non-linear fitting instead of the default linear regression to the log measurements. The data file stores the diffusion MRI data in voxel order with the measurements stored in big-endian format and ordered as in the scheme file. The default input data type is four-byte float. The default output data type is eight-byte double. See modelfit and camino for the format of the data file and scheme file. The program fits the diffusion tensor to each voxel and outputs the results, in voxel order and as big-endian eight-byte doubles, to the standard output. The program outputs eight values in each voxel: [exit code, ln(S(0)), D_xx, D_xy, D_xz, D_yy, D_yz, D_zz]. An exit code of zero indicates no problems. For a list of other exit codes, see modelfit(1). The entry S(0) is an estimate of the signal at q=0.

Example

```python
>>> import nipype.interfaces.camino as cmon
>>> fit = cmon.DTIFit()
>>> fit.inputs.scheme_file = 'A.scheme'
>>> fit.inputs.in_file = 'tensor_fitted_data.Bdouble'
>>> fit.run()
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  voxel-order data filename
  argument: ``%s``, position: 1
scheme_file: (a pathlike object or string representing an existing file)
  Camino scheme file (b values / vectors, see camino.fsl2scheme)
  argument: ``%s``, position: 2

[Optional]
bgmask: (a pathlike object or string representing an existing file)
  Provides the name of a file containing a background mask computed using, for example, FSL bet2 program. The mask file contains zero in background voxels and non-zero in foreground.
  argument: ``-bgmask %s``
non_linear: (a boolean)
  Use non-linear fitting instead of the default linear regression to the log measurements.
  argument: ``-nonlinear``
out_file: (a pathlike object or string representing a file)
  argument: ``> %s``, position: -1
args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``
enviren: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
55.4.6 DTLUTGen

Link to code

Wraps the executable command dtlutgen.

Calibrates the PDFs for PICo probabilistic tractography.

This program needs to be run once for every acquisition scheme. It outputs a lookup table that is used by the dtpicoparams program to find PICo PDF parameters for an image. The default single tensor LUT contains parameters of the Bingham distribution and is generated by supplying a scheme file and an estimated signal to noise in white matter regions of the (q=0) image. The default inversion is linear (inversion index 1).

Advanced users can control several options, including the extent and resolution of the LUT, the inversion index, and the type of PDF. See dtlutgen(1) for details.

Example

```python
>>> import nipype.interfaces.camino as cmon
>>> dtl = cmon.DTLUTGen()
>>> dtl.inputs.snr = 16
>>> dtl.inputs.scheme_file = 'A.scheme'
>>> dtl.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]

scheme_file: (a pathlike object or string representing a file)

- The scheme file of the images to be processed using this LUT.
  - argument: `"-schemefile %s"`, position: 2

[Optional]

lrange: (a list of from 2 to 2 items which are a float)

- Index to one-tensor LUTs. This is the ratio L1/L3 and L2 / L3. The LUT is square, with half the values calculated (because L2 / L3 cannot be less than L1 / L3 by definition). The minimum must be >= 1.
  - For comparison, a ratio L1 / L3 = 10 with L2 / L3 = 1 corresponds to an FA of 0.891, and L1 / L3 = 15 with L2 / L3 = 1 corresponds to an FA of 0.929. The default range is 1 to 10.
  - argument: `"-lrange %s"`, position: 1

frange: (a list of from 2 to 2 items which are a float)

- Index to two-tensor LUTs. This is the fractional anisotropy of the two tensors. The default is 0.3 to 0.94
  - argument: `"-frange %s"`, position: 1

step: (a float)

- Distance between points in the LUT. For example, if lrange is 1 to 10 and the step is 0.1, LUT entries will be computed at L1 / L3 = 1, 1.1, 1.2 ... 10.0 and at L2 / L3 = 1.0, 1.1 ... L1 / L3. For single tensor LUTs, the default step is 0.2, for two-tensor LUTs it is 0.02.
  - argument: `"-step %f"

samples: (an integer (int or long))

- The number of synthetic measurements to generate at each point in the LUT. The default is 2000.
  - argument: `"-samples %d"

snr: (a float)
The signal to noise ratio of the unweighted \((q = 0)\) measurements. This should match the SNR (in white matter) of the images that the LUTs are used with.

argument: `''-snr %f``

**bingham**: (a boolean)
Compute a LUT for the Bingham PDF. This is the default.
argument: `''-bingham``

**acg**: (a boolean)
Compute a LUT for the ACG PDF.
argument: `''-acg``

**watson**: (a boolean)
Compute a LUT for the Watson PDF.
argument: `''-watson``

**inversion**: (an integer \((\text{int or long})\))
Index of the inversion to use. The default is 1 (linear single tensor inversion).
argument: `''-inversion %d``

**trace**: (a float)
Trace of the diffusion tensor(s) used in the test function in the LUT generation. The default is \(2100\times 10^{-12} \text{ m}^2 \text{ s}^{-1}\).
argument: `''-trace %G``

**out_file**: (a pathlike object or string representing a file)
argument: `''> %s``, position: -1

**args**: (a unicode string)
Additional parameters to the command
argument: `''%s``

**environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: `{}`)
Environment variables

### Outputs:

**dtLUT**: (a pathlike object or string representing an existing file)
Lookup Table

#### 55.4.7 DTMetric

**Link to code**

Wraps the executable command `dtshape`.

Computes tensor metric statistics based on the eigenvalues \(l_1 \geq l_2 \geq l_3\) typically obtained from `ComputeEigensystem`.

The full list of statistics is:

- \(<c_l> = (l_1 - l_2) / l_1\), a measure of linearity
- \(<c_p> = (l_2 - l_3) / l_1\), a measure of planarity
- \(<c_s> = l_3 / l_1\), a measure of isotropy with: \(c_l + c_p + c_s = 1\)
- \(<l_1> = \) first eigenvalue
- \(<l_2> = \) second eigenvalue
- \(<l_3> = \) third eigenvalue
- \(<tr> = l_1 + l_2 + l_3\)
- \(<md> = tr / 3\)
- \(<rd> = (l_2 + l_3) / 2\)
- \(<fa> = \) fractional anisotropy. (Basser et al, J Magn Reson B 1996)
- \(<ra> = \) relative anisotropy (Basser et al, J Magn Reson B 1996)
- \(<2dfa> = 2D FA of the two minor eigenvalues \(l_2 \) and \(l_3\) i.e. \(\sqrt{2 \times ([l_2 - <l>]^2 + [l_3 - <l>]^2) / ([l_2]^2 + [l_3]^2)}\) with: \(<l> = (l_2 + l_3) / 2\)
Example

Compute the CP planar metric as float data type.

```python
>>> import nipype.interfaces.camino as cam

>>> dtmetric = cam.DTMetric()

>>> dtmetric.inputs.eigen_data = 'dteig.Bdouble'

>>> dtmetric.inputs.metric = 'cp'

>>> dtmetric.inputs.outputdatatype = 'float'

>>> dtmetric.run()  # doctest: +SKIP
```

**Inputs:**

[Mandatory]

- **eigen_data:** (a pathlike object or string representing an existing file)
  - voxel-order data filename
  - argument: ``-inputfile %s``
- **metric:** ('fa' or 'md' or 'rd' or 'll' or 'l2' or 'l3' or 'tr' or 'ra'
  or '2dfa' or 'cl' or 'cp' or 'cs')
  - Specifies the metric to compute. Possible choices are: "fa", "md",
  "rd", "ll", "l2", "l3", "tr", "ra", "2dfa", "cl", "cp" or "cs".
  - argument: ``-stat %s``

[Optional]

- **inputdatatype:** ('double' or 'float' or 'long' or 'int' or 'short' or
  'char', nipype default value: double)
  - Specifies the data type of the input data. The data type can be any
  of the following strings: "char", "short", "int", "long", "float" or
  "double". Default is double data type
  - argument: ``-inputdatatype %s``
- **outputdatatype:** ('double' or 'float' or 'long' or 'int' or 'short' or
  'char', nipype default value: double)
  - Specifies the data type of the output data. The data type can be any
  of the following strings: "char", "short", "int", "long", "float" or
  "double". Default is double data type
  - argument: ``-outputdatatype %s``
- **data_header:** (a pathlike object or string representing an existing file)
  - A Nifti .nii or .nii.gz file containing the header information.
  Usually this will be the header of the raw data file from which the
  diffusion tensors were reconstructed.
  - argument: ``-header %s``
- **outputfile:** (a pathlike object or string representing a file)
  - Output name. Output will be a .nii.gz file if data_header is
    provided and in voxel order with outputdatatype datatype (default: double)
    otherwise.
  - argument: ``-outputfile %s``
- **args:** (a unicode string)
  - Additional parameters to the command
  - argument: ``%s``
- **environ:** (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})  
  - Environment variables

**Outputs:**

- **metric_stats:** (a pathlike object or string representing an existing file)
55.4.8 ModelFit

Link to code

Wraps the executable command modelfit. Fits models of the spin-displacement density to diffusion MRI measurements.

This is an interface to various model fitting routines for diffusion MRI data that fit models of the spin-displacement density function. In particular, it will fit the diffusion tensor to a set of measurements as well as various other models including two or three-tensor models. The program can read input data from a file or can generate synthetic data using various test functions for testing and simulations.

Example

```python
>>> import nipype.interfaces.camino as cmon
>>> fit = cmon.ModelFit()
>>> fit.model = 'dt'
>>> fit.inputs.scheme_file = 'A.scheme'
>>> fit.inputs.in_file = 'tensor_fitted_data.Bdouble'
>>> fit.run()  # doctest: +SKIP
```

Inputs:

- `model`: ('dt' or 'restore' or 'aldt' or 'nldt_pos' or 'nldt' or 'ldt_wtd' or 'adc' or 'ball_stick' or 'cylcyl dt' or 'cylcyl restore' or 'cylcyl aldlt' or 'cylcyl nldt_pos' or 'cylcyl nldt' or 'cylcyl ldt_wtd' or 'cylcyl adc' or 'cylcyl ball_stick' or 'cylcyl_eq dt' or 'cylcyl_eq restore' or 'cylcyl_eq aldlt' or 'cylcyl_eq nldt_pos' or 'cylcyl_eq nldt' or 'cylcyl_eq ldt_wtd' or 'cylcyl_eq adc' or 'cylcyl_eq ball_stick' or 'pospos dt' or 'pospos restore' or 'pospos aldlt' or 'pospos nldt_pos' or 'pospos nldt' or 'pospos ldt_wtd' or 'pospos adc' or 'pospos ball_stick' or 'pospos_eq dt' or 'pospos_eq restore' or 'pospos_eq aldlt' or 'pospos_eq nldt_pos' or 'pospos_eq nldt' or 'pospos_eq ldt_wtd' or 'pospos_eq adc' or 'pospos_eq ball_stick' or 'poscyl dt' or 'poscyl restore' or 'poscyl aldlt' or 'poscyl nldt_pos' or 'poscyl nldt' or 'poscyl ldt_wtd' or 'poscyl adc' or 'poscyl ball_stick' or 'poscyl_eq dt' or 'poscyl_eq restore' or 'poscyl_eq aldlt' or 'poscyl_eq nldt_pos' or 'poscyl_eq nldt' or 'poscyl_eq ldt_wtd' or 'poscyl_eq adc' or 'poscyl_eq ball_stick' or 'cylcylcyl dt' or 'cylcylcyl restore' or 'cylcylcyl aldlt' or 'cylcylcyl nldt_pos' or 'cylcylcyl nldt' or 'cylcylcyl ldt_wtd' or 'cylcylcyl adc' or 'cylcylcyl ball_stick' or 'cylcylcyl_eq dt' or 'cylcylcyl_eq restore' or 'cylcylcyl_eq aldlt' or 'cylcylcyl_eq nldt_pos' or 'cylcylcyl_eq nldt' or 'cylcylcyl_eq ldt_wtd' or 'cylcylcyl_eq adc' or 'cylcylcyl_eq ball_stick' or 'pospospos dt' or 'pospospos restore' or 'pospospos aldlt' or 'pospospos nldt_pos' or 'pospospos nldt' or 'pospospos ldt_wtd' or 'pospospos adc' or 'pospospos ball_stick' or 'pospospos_eq dt' or 'pospospos_eq restore' or 'pospospos_eq aldlt' or 'pospospos_eq nldt_pos' or 'pospospos_eq nldt' or 'pospospos_eq ldt_wtd' or 'pospospos_eq adc' or 'pospospos_eq ball_stick' or 'posposcyl dt' or 'posposcyl restore' or 'posposcyl aldlt' or 'posposcyl nldt_pos' or 'posposcyl nldt' or 'posposcyl ldt_wtd' or 'posposcyl adc' or 'posposcyl ball_stick' or 'posposcyl_eq dt' or 'posposcyl_eq restore' or 'posposcyl_eq aldlt' or 'posposcyl_eq nldt_pos' or 'posposcyl_eq nldt' or 'posposcyl_eq ldt_wtd' or 'posposcyl_eq adc' or 'posposcyl_eq ball_stick' or 'posposcylcyl dt' or 'posposcylcyl restore' or 'posposcylcyl aldlt' or 'posposcylcyl nldt_pos' or 'posposcylcyl nldt' or 'posposcylcyl ldt_wtd' or 'posposcylcyl adc' or 'posposcylcyl ball_stick' or 'posposcylcyl_eq dt' or 'posposcylcyl_eq restore' or 'posposcylcyl_eq aldlt' or 'posposcylcyl_eq nldt_pos' or 'posposcylcyl_eq nldt' or 'posposcylcyl_eq ldt_wtd' or 'posposcylcyl_eq adc' or 'posposcylcyl_eq ball_stick'
```

(continues on next page)
Specifies the model to be fit to the data.

Argument: ``-model %s``

In_file: (a pathlike object or string representing an existing file)

Voxel-order data filename

Argument: ``-inputfile %s``

Scheme_file: (a pathlike object or string representing an existing file)

Camino scheme file (b values / vectors, see camino.fsl2scheme)

Argument: ``-schemefile %s``

[Optional]

Inputdatatype: ('float' or 'char' or 'short' or 'int' or 'long' or 'double')

Specifies the data type of the input file: "char", "short", "int", "long", "float" or "double". The input file must have BIG-ENDIAN ordering. By default, the input type is "float".

Argument: ``-inputdatatype %s``

Outputfile: (a pathlike object or string representing a file)

Filename of the output file.

Argument: ``-outputfile %s``

Outlier: (a pathlike object or string representing an existing file)

Specifies the name of the file to contain the outlier map generated by the RESTORE algorithm.

Argument: ``-outliermap %s``

Noisemap: (a pathlike object or string representing an existing file)

Specifies the name of the file to contain the estimated noise variance on the diffusion-weighted signal, generated by a weighted tensor fit. The data type of this file is big-endian double.

Argument: ``-noisemap %s``

Residualmap: (a pathlike object or string representing an existing file)

Specifies the name of the file to contain the weighted residual errors after computing a weighted linear tensor fit. One value is produced per measurement, in voxel order. The data type of this file is big-endian double. Images of the residuals for each measurement can be extracted with shredder.

Argument: ``-residualmap %s``

Sigma: (a float)

Specifies the standard deviation of the noise in the data. Required by the RESTORE algorithm.

Argument: ``-sigma %G``

Bgthresh: (a float)

Sets a threshold on the average q=0 measurement to separate foreground and background. The program does not process background voxels, but outputs the same number of values in background voxels and foreground voxels. Each value is zero in background voxels apart.
from the exit code which is -1.

argument: ``-bgthresh %G``

bgmask: (a pathlike object or string representing an existing file)

Provides the name of a file containing a background mask computed using, for example, FSL's bet2 program. The mask file contains zero in background voxels and non-zero in foreground.

argument: ``-bgmask %s``

cfthresh: (a float)

Sets a threshold on the average q=0 measurement to determine which voxels are CSF. This program does not treat CSF voxels any different to other voxels.

argument: ``-csfthresh %G``

fixedmodq: (a list of from 4 to 4 items which are a float)

Specifies <M> <N> <Q> <tau> a spherical acquisition scheme with M measurements with q=0 and N measurements with |q|=Q and diffusion time tau. The N measurements with |q|=Q have unique directions. The program reads in the directions from the files in directory PointSets.

argument: ``-fixedmod %s``

fixedbvalue: (a list of from 3 to 3 items which are a float)

As above, but specifies <M> <N> <b>. The resulting scheme is the same whether you specify b directly or indirectly using -fixedmodq.

argument: ``-fixedbvalue %s``

tau: (a float)

Sets the diffusion time separately. This overrides the diffusion time specified in a scheme file or by a scheme index for both the acquisition scheme and in the data synthesis.

argument: ``-tau %G``

out_file: (a pathlike object or string representing a file)

argument: ``> %s``, position: -1

args: (a unicode string)

Additional parameters to the command

argument:``%s``

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

Environment variables

Outputs:

fitted_data: (a pathlike object or string representing an existing file)

output file of 4D volume in voxel order

55.4.9 PicoPDFs

Link to code

Wraps the executable command picopdfs. Constructs a spherical PDF in each voxel for probabilistic tractography.

Example

```python
>>> import nipype.interfaces.camino as cmon
>>> pdf = cmon.PicoPDFs()
>>> pdf.inputs.inputmodel = 'dt'
```
>>> pdf.inputs.luts = ['lut_file']
>>> pdf.inputs.in_file = 'voxel-order_data.Bfloat'
>>> pdf.run()  # doctest: +SKIP

Inputs:

[Mandatory]

in_file: (a pathlike object or string representing an existing file)
  voxel-order data filename
  argument: ``< %s``
  position: 1

luts: (a list of items which are a pathlike object or string
  representing an existing file)
  Files containing the lookup tables. For tensor data, one lut must be
  specified for each type of inversion used in the image (one-tensor,
  two-tensor, three-tensor). For pds, the number of LUTs must match
  -numpds (it is acceptable to use the same LUT several times - see
  example, above). These LUTs may be generated with dtlutgen.
  argument: ``-luts %s``

[Optional]

inputmodel: ('dt' or 'multitensor' or 'pds', nipype default value: dt)
  input model type
  argument: ``-inputmodel %s``
  position: 2

pdf: ('bingham' or 'watson' or 'acg', nipype default value: bingham)
  Specifies the PDF to use. There are three choices:
  watson - The Watson distribution. This distribution is rotationally
  symmetric.
  bingham - The Bingham distribution, which allows
  elliptical probability density contours.
  acg - The Angular Central Gaussian distribution, which also allows
  elliptical probability density contours.
  argument: ``-pdf %s``
  position: 4

directmap: (a boolean)
  Only applicable when using pds as the inputmodel. Use direct mapping
  between the eigenvalues and the distribution parameters instead of
  the log of the eigenvalues.
  argument: ``-directmap``

maxcomponents: (an integer (int or long))
  The maximum number of tensor components in a voxel (default 2) for
  multitensor data. Currently, only the default is supported, but
  future releases may allow the input of three-tensor data using this
  option.
  argument: ``-maxcomponents %d``

numpds: (an integer (int or long))
  The maximum number of PDs in a voxel (default 3) for PD data. This
  option determines the size of the input and output voxels. This means
  that the data file may be large enough to accommodate three or more
  PDs, but does not mean that any of the voxels are classified as
  containing three or more PDs.
  argument: ``-numpds %d``

out_file: (a pathlike object or string representing a file)
  argument: ``> %s``
  position: -1

args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
Environment variables

<table>
<thead>
<tr>
<th>Outputs:</th>
</tr>
</thead>
<tbody>
<tr>
<td>pdfs: (a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>path/name of 4D volume in voxel order</td>
</tr>
</tbody>
</table>

### 55.4.10 Track

**Link to code**

Wraps the executable command `track`.

Performs tractography using one of the following models: `dt`, `multitensor`, `pds`, `pico`, `bootstrap`, `ballstick`, `bayesdirac`.

**Example**

```python
>>> import nipype.interfaces.camino as cmon
>>> track = cmon.Track()
>>> track.inputs.inputmodel = 'dt'
>>> track.inputs.in_file = 'data.Bfloat'
>>> track.inputs.seed_file = 'seed_mask.nii'
>>> track.run()  # doctest: +SKIP
```

**Inputs:**

- `in_file`: (a pathlike object or string representing an existing file)
  - input data file
  - argument: `'-inputfile %s'`, position: 1
- `seed_file`: (a pathlike object or string representing an existing file)
  - seed file
  - argument: `'-seedfile %s'`, position: 2
- `inputmodel`: (`'dt'` or `'multitensor'` or `'sfpeak'` or `'pico'` or `repbs_dt` or `repbs_multitensor` or `ballstick` or `wildbs_dt` or `bayesdirac` or `bayesdirac_dt` or `bedpostx_dyad` or `bedpostx`,
  - nipype default value: `dt`)
  - input model type
  - argument: `'-inputmodel %s'`
- `tracker`: (`'fact'` or `'euler'` or `'rk4'`, nipype default value: `fact`)
  - The tracking algorithm controls streamlines are generated from the data. The choices are: - FACT, which follows the local fibre orientation in each voxel. No interpolation is used.- EULER, which uses a fixed step size along the local fibre orientation. With nearest-neighbour interpolation, this method may be very similar to FACT, except that the step size is fixed, whereas FACT steps extend to the boundary of the next voxel (distance variable depending on the entry and exit points to the voxel).- RK4: Fourth-order Runge-Kutta method. The step size is fixed, however the eventual direction of the step is determined by taking and averaging a series of partial steps.
  - argument: `'-tracker %s'`
- `interpolator`: (`'nn'` or `'prob_nn'` or `'linear'`)
  - The interpolation algorithm determines how the fiber orientation(s) are defined at a given continuous point within the input image.
Interpolators are only used when the tracking algorithm is not FACT. The choices are:
- NN: Nearest-neighbour interpolation, just uses the local voxel data directly.
- PROB_NN: Probabilistic nearest-neighbor interpolation, similar to the method proposed by Behrens et al [Magnetic Resonance in Medicine, 50:1077-1088, 2003]. The data is not interpolated, but at each point we randomly choose one of the 8 voxels surrounding a point. The probability of choosing a particular voxel is based on how close the point is to the centre of that voxel.
- LINEAR: Linear interpolation of the vector field containing the principal directions at each point.

```
argument: ``--interpolator %s``
```

stepsize: (a float)
Step size for EULER and RK4 tracking. The default is 1mm.

```
argument: ``--stepsize %f``
```

inputdatatype: ('float' or 'double')
input file type

```
argument: ``--inputdatatype %s``
```

gzip: (a boolean)
save the output image in gzip format

```
argument: ``--gzip``
```

maxcomponents: (an integer (int or long))
The maximum number of tensor components in a voxel. This determines the size of the input file and does not say anything about the voxel classification. The default is 2 if the input model is multitensor and 1 if the input model is dt.

```
argument: ``--maxcomponents %d``
```

numpds: (an integer (int or long))
The maximum number of PDs in a voxel for input models sfpeak and pico. The default is 3 for input model sfpeak and 1 for input model pico. This option determines the size of the voxels in the input file and does not affect tracking. For tensor data, use the -maxcomponents option.

```
argument: ``--numpds %d``
```

data_dims: (a list of from 3 to 3 items which are an integer (int or long))
data dimensions in voxels

```
argument: ``--datadims %s``
```

voxel_dims: (a list of from 3 to 3 items which are a float)
voxel dimensions in mm

```
argument: ``--voxeldims %s``
```

iptresh: (a float)
Curvature threshold for tracking, expressed as the minimum dot product between two streamline orientations calculated over the length of a voxel. If the dot product between the previous and current directions is less than this threshold, then the streamline terminates. The default setting will terminate fibres that curve by more than 80 degrees. Set this to -1.0 to disable curvature checking completely.

```
argument: ``--iptresh %f``
```

curvethresh: (a float)
Curvature threshold for tracking, expressed as the maximum angle (in degrees) between between two streamline orientations calculated over the length of a voxel. If the angle is greater than this, then the streamline terminates.

```
argument: ``--curvethresh %f``
```

curveinterval: (a float)
Interval over which the curvature threshold should be evaluated, in
mm. The default is 5mm. When using the default curvature threshold
of 90 degrees, this means that streamlines will terminate if they
curve by more than 90 degrees over a path length of 5mm.
argument: `--curveinterval %f`
requires: curvethresh

anisthresh: (a float)
Terminate fibres that enter a voxel with lower anisotropy than the
threshold.
argument: `--anisthresh %f`

anisfile: (a pathlike object or string representing an existing file)
File containing the anisotropy map. This is required to apply an
anisotropy threshold with non tensor data. If the map is supplied it
is always used, even in tensor data.
argument: `--anisfile %s`

outputtracts: ('float' or 'double' or 'oogl')
output tract file type
argument: `--outputtracts %s`

out_file: (a pathlike object or string representing a file)
output data file
argument: `--outputfile %s`, position: -1

output_root: (a pathlike object or string representing a file)
root directory for output
argument: `--outputroot %s`, position: -1

args: (a unicode string)
Additional parameters to the command
argument: `--%s`

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

tracked: (a pathlike object or string representing an existing file)
output file containing reconstructed tracts

55.4.11 TrackBallStick

Link to code
Wraps the executable command track.
Performs streamline tractography using ball-stick fitted data

Example

```python
>>> import nipype.interfaces.camino as cmon
>>> track = cmon.TrackBallStick()
>>> track.inputs.in_file = 'ballstickfit_data.Bfloat'
>>> track.inputs.seed_file = 'seed_mask.nii'
>>> track.run() # doctest: +SKIP
```

Inputs:

[Optional]
in_file: (a pathlike object or string representing an existing file)
input data file

(continues on next page)
nipype Documentation, Release 1.2.1

(continued from previous page)

argument: `'-inputfile %s'`, position: 1

seed_file: (a pathlike object or string representing an existing file)

seed file

argument: `'-seedfile %s'`, position: 2

inputmodel: (`'dt'` or `'multitensor'` or `'sfpeak'` or `'pico'` or
`'repbs_dt'` or `'repbs_multitensor'` or `'ballstick'` or `'wildbs_dt'` or
`'bayesdirac'` or `'bayesdirac_dt'` or `'bedpostx_dyad'` or `'bedpostx'`,
nipype default value: `dt`)

input model type

argument: `'-inputmodel %s'`

tracker: (`'fact'` or `'euler'` or `'rk4'`, nipype default value: `fact`)

The tracking algorithm controls streamlines are generated from the data. The choices are: - FACT, which follows the local fibre orientation in each voxel. No interpolation is used.- EULER, which uses a fixed step size along the local fibre orientation. With nearest-neighbour interpolation, this method may be very similar to FACT, except that the step size is fixed, whereas FACT steps extend to the boundary of the next voxel (distance variable depending on the entry and exit points to the voxel).- RK4: Fourth-order Runge-Kutta method. The step size is fixed, however the eventual direction of the step is determined by taking and averaging a series of partial steps.

argument: `'-tracker %s'`

interpolator: (`'nn'` or `'prob_nn'` or `'linear'`)

The interpolation algorithm determines how the fiber orientation(s) are defined at a given continuous point within the input image. Interpolators are only used when the tracking algorithm is not FACT. The choices are: - NN: Nearest-neighbour interpolation, just uses the local voxel data directly.- PROB_NN: Probabilistic nearest-neighbor interpolation, similar to the method pro- posed by Behrens et al [Magnetic Resonance in Medicine, 50:1077-1088, 2003]. The data is not interpolated, but at each point we randomly choose one of the 8 voxels sur- rounding a point. The probability of choosing a particular voxel is based on how close the point is to the centre of that voxel.- LINEAR: Linear interpolation of the vector field containing the principal directions at each point.

argument: `'-interpolator %s'`

stepsize: (a float)

Step size for EULER and RK4 tracking. The default is 1mm.

argument: `'-stepsize %f'`

requires: tracker

inputdatatype: (`'float'` or `'double'`)

input file type

argument: `'-inputdatatype %s'`

gzip: (a boolean)

save the output image in gzip format

argument: `'-gzip'`

maxcomponents: (an integer (int or long))

The maximum number of tensor components in a voxel. This determines the size of the input file and does not say anything about the voxel classification. The default is 2 if the input model is multitensor and 1 if the input model is `dt`.

argument: `'-maxcomponents %d'`

numpds: (an integer (int or long))

The maximum number of PDs in a voxel for input models `sfpeak` and `pico`. The default is 3 for input model `sfpeak` and 1 for input model.
pico. This option determines the size of the voxels in the input file and does not affect tracking. For tensor data, use the -maxcomponents option.

- `data_dims`: (a list of from 3 to 3 items which are an integer (int or long))
  - data dimensions in voxels
    - argument: `'-datadims %s'`

- `voxel_dims`: (a list of from 3 to 3 items which are a float)
  - voxel dimensions in mm
    - argument: `'-voxeldims %s'`

- `ipthresh`: (a float)
  - Curvature threshold for tracking, expressed as the minimum dot product between two streamline orientations calculated over the length of a voxel. If the dot product between the previous and current directions is less than this threshold, then the streamline terminates. The default setting will terminate fibres that curve by more than 80 degrees. Set this to -1.0 to disable curvature checking completely.
    - argument: `'-ipthresh %f'`

- `curvethresh`: (a float)
  - Curvature threshold for tracking, expressed as the maximum angle (in degrees) between between two streamline orientations calculated over the length of a voxel. If the angle is greater than this, then the streamline terminates.
    - argument: `'-curvethresh %f'`

- `curveinterval`: (a float)
  - Interval over which the curvature threshold should be evaluated, in mm. The default is 5mm. When using the default curvature threshold of 90 degrees, this means that streamlines will terminate if they curve by more than 90 degrees over a path length of 5mm.
    - argument: `'-curveinterval %f'`
    - requires: curvethresh

- `anisthresh`: (a float)
  - Terminate fibres that enter a voxel with lower anisotropy than the threshold.
    - argument: `'-anisthresh %f'`

- `anisfile`: (a pathlike object or string representing an existing file)
  - File containing the anisotropy map. This is required to apply an anisotropy threshold with non tensor data. If the map is supplied it is always used, even in tensor data.
    - argument: `'-anisfile %s'`

- `outputtracts`: ('float' or 'double' or 'oogl')
  - output tract file type
    - argument: `'-outputtracts %s'`

- `out_file`: (a pathlike object or string representing a file)
  - output data file
    - argument: `'-outputfile %s'`, position: -1

- `output_root`: (a pathlike object or string representing a file)
  - root directory for output
    - argument: `'-outputroot %s'`, position: -1

- `args`: (a unicode string)
  - Additional parameters to the command
    - argument: `'%s'`

- `environ`: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

tracked: (a pathlike object or string representing an existing file)
output file containing reconstructed tracts

55.4.12 TrackBayesDirac

Link to code
Wraps the executable command track.
Performs streamline tractography using a Bayesian tracking with Dirac priors

Example

```python
>>> import nipype.interfaces.camino as cmon
>>> track = cmon.TrackBayesDirac()
>>> track.inputs.in_file = 'tensor_fitted_data.Bdouble'
>>> track.inputs.seed_file = 'seed_mask.nii'
>>> track.inputs.scheme_file = 'bvecs.scheme'
>>> track.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
scheme_file: (a pathlike object or string representing an existing file)
The scheme file corresponding to the data being processed.
argument: `'-schemefile %s'`

[Optional]
iterations: (an integer (int or long))
Number of streamlines to generate at each seed point. The default is 5000.
argument: `'-iterations %d'`
pdf: ('bingham' or 'watson' or 'acg')
Specifies the model for PICo priors (not the curvature priors). The default is "bingham".
argument: `'-pdf %s'`
pointset: (an integer (int or long))
Index to the point set to use for Bayesian likelihood calculation. The index specifies a set of evenly distributed points on the unit sphere, where each point x defines two possible step directions (x or -x) for the streamline path. A larger number indexes a larger point set, which gives higher angular resolution at the expense of computation time. The default is index 1, which gives 1922 points, index 0 gives 1082 points, index 2 gives 3002 points.
argument: `'-pointset %s'`
datamodel: ('cylsymmdt' or 'ballstick')
Model of the data for Bayesian tracking. The default model is "cylsymmdt", a diffusion tensor with cylindrical symmetry about e_1, ie L1 >= L_2 = L_3. The other model is "ballstick", the partial volume model (see ballstickfit).
argument: `'-datamodel %s'`
curvetriork: (a float)
Concentration parameter for the prior distribution on fibre
(continues on next page)
orientations given the fibre orientation at the previous step. Larger values of k make curvature less likely.

argument: ``-curvepriork %G``

curvepriorg: (a float)
  Concentration parameter for the prior distribution on fibre orientations given the fibre orientation at the previous step. Larger values of g make curvature less likely.

argument: ``-curvepriorg %G``

extpriorfile: (a pathlike object or string representing an existing file)
  Path to a PICo image produced by picopdfs. The PDF in each voxel is used as a prior for the fibre orientation in Bayesian tracking. The prior image must be in the same space as the diffusion data.

argument: ``-extpriorfile %s``

extpriordatatype: ('float' or 'double')
  Datatype of the prior image. The default is "double".

argument: ``-extpriordatatype %s``

in_file: (a pathlike object or string representing an existing file)
  input data file

argument: ``-inputfile %s``, position: 1

seed_file: (a pathlike object or string representing an existing file)
  seed file

argument: ``-seedfile %s``, position: 2

inputmodel: ('dt' or 'multitensor' or 'sfpeak' or 'pico' or 'repbs_dt' or 'repbs_multitensor' or 'ballstick' or 'wildbs_dt' or 'bayesdirac' or 'bayesdirac_dt' or 'bedpostx_dyad' or 'bedpostx', nipype default value: dt)
  input model type

argument: ``-inputmodel %s``

tracker: ('fact' or 'euler' or 'rk4', nipype default value: fact)
  The tracking algorithm controls streamlines are generated from the data. The choices are: - FACT, which follows the local fibre orientation in each voxel. No interpolation is used.- EULER, which uses a fixed step size along the local fibre orientation. With nearest-neighbour interpolation, this method may be very similar to FACT, except that the step size is fixed, whereas FACT steps extend to the boundary of the next voxel (distance variable depending on the entry and exit points to the voxel).- RK4: Fourth-order Runge-Kutta method. The step size is fixed, however the eventual direction of the step is determined by taking and averaging a series of partial steps.

argument: ``-tracker %s``

interpolator: ('nn' or 'prob_nn' or 'linear')
  The interpolation algorithm determines how the fiber orientation(s) are defined at a given continuous point within the input image. Interpolators are only used when the tracking algorithm is not FACT. The choices are: - NN: Nearest-neighbour interpolation, just uses the local voxel data directly.- PROB_NN: Probabilistic nearest-neighbor interpolation, similar to the method proposed by Behrens et al [Magnetic Resonance in Medicine, 50:1077-1088, 2003]. The data is not interpolated, but at each point we randomly choose one of the 8 voxels surrounding a point. The probability of choosing a particular voxel is based on how close the point is to the centre of that voxel.- LINEAR: Linear interpolation of the vector field containing the principal directions at each point.

argument: ``-interpolator %s``
stepsize: (a float)
    Step size for EULER and RK4 tracking. The default is 1mm.
    argument: ``-stepsize %f``
    requires: tracker

inputdatatype: ('float' or 'double')
    input file type
    argument: ``-inputdatatype %s``

gzip: (a boolean)
    save the output image in gzip format
    argument: ``-gzip``

maxcomponents: (an integer (int or long))
    The maximum number of tensor components in a voxel. This determines
    the size of the input file and does not say anything about the voxel
    classification. The default is 2 if the input model is multitensor
    and 1 if the input model is dt.
    argument: ``-maxcomponents %d``

numpds: (an integer (int or long))
    The maximum number of PDs in a voxel for input models sfpeak and
    pico. The default is 3 for input model sfpeak and 1 for input model
    pico. This option determines the size of the voxels in the input
    file and does not affect tracking. For tensor data, use the
    -maxcomponents option.
    argument: ``-numpds %d``

data_dims: (a list of from 3 to 3 items which are an integer (int or
    long))
    data dimensions in voxels
    argument: ``-datadims %s``

voxel_dims: (a list of from 3 to 3 items which are a float)
    voxel dimensions in mm
    argument: ``-voxeldims %s``

ipthresh: (a float)
    Curvature threshold for tracking, expressed as the minimum dot
    product between two streamline orientations calculated over the
    length of a voxel. If the dot product between the previous and
    current directions is less than this threshold, then the streamline
    terminates. The default setting will terminate fibres that curve by
    more than 80 degrees. Set this to -1.0 to disable curvature checking
    completely.
    argument: ``-ipthresh %f``

curvethresh: (a float)
    Curvature threshold for tracking, expressed as the maximum angle (in
    degrees) between between two streamline orientations calculated over
    the length of a voxel. If the angle is greater than this, then the
    streamline terminates.
    argument: ``-curvethresh %f``

curveinterval: (a float)
    Interval over which the curvature threshold should be evaluated, in
    mm. The default is 5mm. When using the default curvature threshold
    of 90 degrees, this means that streamlines will terminate if they
    curve by more than 90 degrees over a path length of 5mm.
    argument: ``-curveinterval %f``
    requires: curvethresh

anisthresh: (a float)
    Terminate fibres that enter a voxel with lower anisotropy than the
    threshold.
    argument: ``-anisthresh %f``

anisfile: (a pathlike object or string representing an existing file)
File containing the anisotropy map. This is required to apply an anisotropy threshold with non tensor data. If the map is supplied it is always used, even in tensor data.

**argument:** ``--anisfile %s``

**outputtracts:** ('float' or 'double' or 'oogl')

**output tract file type**

**argument:** ``--outputtracts %s``

**out_file:** (a pathlike object or string representing a file)

**output data file**

**argument:** ``--outputfile %s``, position: -1

**output_root:** (a pathlike object or string representing a file)

**root directory for output**

**argument:** ``--outputroot %s``, position: -1

**args:** (a unicode string)

**Additional parameters to the command**

**argument:** ``%s``

**environ:** (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})""

---

**Output variables**

---

**tracked:** (a pathlike object or string representing an existing file)

**output file containing reconstructed tracts**

---

### 55.4.13 TrackBedpostxDeter

**Link to code**

Wraps the executable command track.

Data from FSL’s bedpostx can be imported into Camino for deterministic tracking. (Use TrackBedpostxProba for bedpostx probabilistic tractography.)

The tracking is based on the vector images dyads1.nii.gz, . . . , dyadsN.nii.gz, where there are a maximum of N compartments (corresponding to each fiber population) in each voxel.

It also uses the N images mean_f1samples.nii.gz, . . . , mean_fNsamples.nii.gz, normalized such that the sum of all compartments is 1. Compartments where the mean_f is less than a threshold are discarded and not used for tracking. The default value is 0.01. This can be changed with the min_vol_frac option.

**Example**

```
>>> import nipype.interfaces.camino as cam
>>> track = cam.TrackBedpostxDeter()
>>> track.inputs.bedpostxdir = 'bedpostxout'
>>> track.inputs.seed_file = 'seed_mask.nii'
>>> track.run()  # doctest: +SKIP
```

---

**Inputs:**

[Mandatory]

**bedpostxdir:** (a pathlike object or string representing an existing directory)

Directory containing bedpostx output

**argument:** ``--bedpostxdir %s``

---

[Optional]
<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-bedpostxminf %d</code></td>
<td><code>in_file: (a pathlike object or string representing an existing file)</code></td>
</tr>
<tr>
<td><code>-inputfile %s</code>, position: 1</td>
<td>input data file</td>
</tr>
<tr>
<td><code>-seedfile %s</code>, position: 2</td>
<td><code>seed_file: (a pathlike object or string representing an existing file)</code></td>
</tr>
<tr>
<td><code>-inputmodel %s</code></td>
<td><code>inputmodel: ('dt' or 'multitensor' or 'sfpeak' or 'pico' or 'repbs_dt' or 'repbs_multitensor' or 'ballstick' or 'wildbs_dt' or 'bayesdirac' or 'bayesdirac_dt' or 'bedpostx_dyad' or 'bedpostx', nipype default value: dt)</code></td>
</tr>
<tr>
<td><code>-tracker %s</code></td>
<td><code>tracker: ('fact' or 'euler' or 'rk4', nipype default value: fact)</code></td>
</tr>
<tr>
<td><code>-interpolator %s</code></td>
<td><code>interpolator: ('nn' or 'prob_nn' or 'linear')</code></td>
</tr>
<tr>
<td><code>-stepsize %f</code></td>
<td><code>stepsize: (a float)</code></td>
</tr>
<tr>
<td><code>-inputdatatype %s</code></td>
<td><code>inputdatatype: ('float' or 'double')</code></td>
</tr>
<tr>
<td><code>-gzip</code></td>
<td><code>gzip: (a boolean)</code></td>
</tr>
<tr>
<td><code>-maxcomponents %d</code></td>
<td><code>maxcomponents: (an integer (int or long))</code></td>
</tr>
</tbody>
</table>

- bedpostxminf: Zeros out compartments in bedpostx data with a mean volume fraction f of less than min_vol_frac. The default is 0.01.
- in_file: Input data file
- seed_file: Seed file
- inputmodel: Input model type
- tracker: The tracking algorithm controls streamlines generated from the data. The choices are:
  - FACT: Follows the local fibre orientation in each voxel. No interpolation is used.
  - EULER: Uses a fixed step size along the local fibre orientation. With nearest-neighbour interpolation, this method may be very similar to FACT, except that the step size is fixed, whereas FACT steps extend to the boundary of the next voxel (distance variable depending on the entry and exit points to the voxel).
  - RK4: Fourth-order Runge-Kutta method. The step size is fixed, however the eventual direction of the step is determined by taking and averaging a series of partial steps.
- interpolator: The interpolation algorithm determines how the fiber orientation(s) are defined at a given continuous point within the input image. Interpolators are only used when the tracking algorithm is not FACT. The choices are:
  - NN: Nearest-neighbour interpolation, just uses the local voxel data directly.
  - PROB_NN: Probabilistic nearest-neighbor interpolation, similar to the method proposed by Behrens et al [Magnetic Resonance in Medicine, 50:1077-1088, 2003]. The data is not interpolated, but at each point we randomly choose one of the 8 voxels surrounding a point. The probability of choosing a particular voxel is based on how close the point is to the centre of that voxel.
  - LINEAR: Linear interpolation of the vector field containing the principal directions at each point.
- stepsize: Step size for EULER and RK4 tracking. The default is 1mm.
- gzip: Save the output image in gzip format.
classification. The default is 2 if the input model is multitensor
and 1 if the input model is dt.
argument: `--maxcomponents %d`

numpds: (an integer (int or long))
The maximum number of PDs in a voxel for input models sfpeak and
pico. The default is 3 for input model sfpeak and 1 for input model
pico. This option determines the size of the voxels in the input
file and does not affect tracking. For tensor data, use the
--maxcomponents option.
argument: `--numpds %d`
data_dims: (a list of from 3 to 3 items which are an integer (int or
long))
data dimensions in voxels
argument: `--datadims %s`
voxel_dims: (a list of from 3 to 3 items which are a float)
voxel dimensions in mm
argument: `--voxeldims %s`
ipthresh: (a float)
Curvature threshold for tracking, expressed as the minimum dot
product between two streamline orientations calculated over the
length of a voxel. If the dot product between the previous and
current directions is less than this threshold, then the streamline
terminates. The default setting will terminate fibres that curve by
more than 80 degrees. Set this to -1.0 to disable curvature checking
completely.
argument: `--ipthresh %f`
curvethresh: (a float)
Curvature threshold for tracking, expressed as the maximum angle (in
degrees) between between two streamline orientations calculated over
the length of a voxel. If the angle is greater than this, then the
streamline terminates.
argument: `--curvethresh %f`
curveinterval: (a float)
Interval over which the curvature threshold should be evaluated, in
mm. The default is 5mm. When using the default curvature threshold
of 90 degrees, this means that streamlines will terminate if they
curve by more than 90 degrees over a path length of 5mm.
argument: `--curveinterval %f`
requires: curvethresh
anisthresh: (a float)
Terminate fibres that enter a voxel with lower anisotropy than the
threshold.
argument: `--anisthresh %f`
anisfile: (a pathlike object or string representing an existing file)
File containing the anisotropy map. This is required to apply an
anisotropy threshold with non tensor data. If the map issupplied it
is always used, even in tensor data.
argument: `--anisfile %s`
outputtracts: ("float" or "double" or "oogl")
output tract file type
argument: `--outputtracts %s`
out_file: (a pathlike object or string representing a file)
output data file
argument: `--outputfile %s`, position: -1
output_root: (a pathlike object or string representing a file)
root directory for output
argument: `--outputroot %s`, position: -1

(continues on previous page)
55.4.14 TrackBedpostxProba

Link to code
Wraps the executable command track.
Data from FSL’s bedpostx can be imported into Camino for probabilistic tracking. (Use TrackBedpostxDeter for bedpostx deterministic tractography.)
The tracking uses the files merged_th1samples.nii.gz, merged_ph1samples.nii.gz, ... ,
merged_thNsamples.nii.gz, merged_phNsamples.nii.gz where there are a maximum of N compartments
(corresponding to each fiber population) in each voxel. These images contain M samples of theta and phi, the
polar coordinates describing the "stick" for each compartment. At each iteration, a random number X between
1 and M is drawn and the Xth samples of theta and phi become the principal directions in the voxel.
It also uses the N images mean_f1samples.nii.gz, ..., mean_fNsamples.nii.gz, normalized such that the sum of
all compartments is 1. Compartments where the mean_f is less than a threshold are discarded and not used for
tracking. The default value is 0.01. This can be changed with the min_vol_frac option.

Example

```python
>>> import nipype.interfaces.camino as cam
>>> track = cam.TrackBedpostxProba()
>>> track.inputs.bedpostxdir = 'bedpostxout'
>>> track.inputs.seed_file = 'seed_mask.nii'
>>> track.inputs.iterations = 100
>>> track.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
bedpostxdir: (a pathlike object or string representing an existing
directory)
   Directory containing bedpostx output
   argument: `-bedpostxdir %s`

[Optional]
min_vol_frac: (a float)
   Zeros out compartments in bedpostx data with a mean volume fraction
   f of less than min_vol_frac. The default is 0.01.
   argument: `-bedpostxminf %d`
iterations: (an integer (int or long))
   Number of streamlines to generate at each seed point. The default is
   1.
   argument: `-iterations %d`
in_file: (a pathlike object or string representing an existing file)
   argument: `-%s`
**input data file**
argument: `''-inputfile %s''`, position: 1

**seed file**
argument: `''-seedfile %s''`, position: 2

**inputmodel**
argument: `''-inputmodel %s''`

**tracker**
argument: `''-tracker %s''`

**interpolator**
argument: `''-interpolator %s''`

**stepsize**
argument: `''-stepsize %f''`

**inputdatatype**
argument: `''-inputdatatype %s''`

**gzip**
argument: `''-gzip''`

**maxcomponents**
argument: `''-maxcomponents %d''`

**numpds**
argument: `''-numpds %d''`

---

The tracking algorithm controls streamlines are generated from the data. The choices are: - FACT, which follows the local fibre orientation in each voxel. No interpolation is used.- EULER, which uses a fixed step size along the local fibre orientation. With nearest-neighbour interpolation, this method may be very similar to FACT, except that the step size is fixed, whereas FACT steps extend to the boundary of the next voxel (distance variable depending on the entry and exit points to the voxel).- RK4: Fourth-order Runge-Kutta method. The step size is fixed, however the eventual direction of the step is determined by taking and averaging a series of partial steps.

The interpolation algorithm determines how the fiber orientation(s) are defined at a given continuous point within the input image. Interpolators are only used when the tracking algorithm is not FACT. The choices are: - NN: Nearest-neighbour interpolation, just uses the local voxel data directly.- PROB_NN: Probabilistic nearest-neighbor interpolation, similar to the method pro- posed by Behrens et al [Magnetic Resonance in Medicine, 50:1077-1088, 2003]. The data is not interpolated, but at each point we randomly choose one of the 8 voxels sur- rounding a point. The probability of choosing a particular voxel is based on how close the point is to the centre of that voxel.- LINEAR: Linear interpolation of the vector field containing the principal directions at each point.

The maximum number of tensor components in a voxel. This determines the size of the input file and does not say anything about the voxel classification. The default is 2 if the input model is multitensor and 1 if the input model is dt.

The maximum number of PDs in a voxel for input models sfpeak and
pico. The default is 3 for input model sfpeak and 1 for input model pico. This option determines the size of the voxels in the input file and does not affect tracking. For tensor data, use the -maxcomponents option.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_dims</td>
<td>A list of from 3 to 3 items which are an integer</td>
</tr>
<tr>
<td>voxel_dims</td>
<td>A list of from 3 to 3 items which are a float</td>
</tr>
<tr>
<td>ipthresh</td>
<td>A float</td>
</tr>
<tr>
<td>curvethresh</td>
<td>A float</td>
</tr>
<tr>
<td>curveinterval</td>
<td>A float</td>
</tr>
<tr>
<td>anisthresh</td>
<td>A float</td>
</tr>
<tr>
<td>anisfile</td>
<td>A pathlike object or string representing an existing file</td>
</tr>
<tr>
<td>output_tracts</td>
<td>('float' or 'double' or 'oogl')</td>
</tr>
<tr>
<td>out_file</td>
<td>A pathlike object or string representing a file</td>
</tr>
<tr>
<td>output_root</td>
<td>A pathlike object or string representing a file</td>
</tr>
<tr>
<td>args</td>
<td>A unicode string</td>
</tr>
<tr>
<td>environ</td>
<td>A dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a</td>
</tr>
</tbody>
</table>

Curvature threshold for tracking, expressed as the minimum dot product between two streamline orientations calculated over the length of a voxel. If the dot product between the previous and current directions is less than this threshold, then the streamline terminates. The default setting will terminate fibres that curve by more than 80 degrees. Set this to -1.0 to disable curvature checking completely.

Argument: ``-ipthresh %f``

Curvature threshold for tracking, expressed as the maximum angle (in degrees) between between two streamline orientations calculated over the length of a voxel. If the angle is greater than this, then the streamline terminates.

Argument: ``-curvethresh %f``

Interval over which the curvature threshold should be evaluated, in mm. The default is 5mm. When using the default curvature threshold of 90 degrees, this means that streamlines will terminate if they curve by more than 90 degrees over a path length of 5mm.

Argument: ``-curveinterval %f``

Terminate fibres that enter a voxel with lower anisotropy than the threshold.

Argument: ``-anisthresh %f``

File containing the anisotropy map. This is required to apply an anisotropy threshold with non tensor data. If the map is supplied it is always used, even in tensor data.

Argument: ``-anisfile %s``

output_tracts: ('float' or 'double' or 'oogl')

output file type

Argument: ``-outputtracts %s``

output_tract file type

Argument: ``-outputfile %s``

output data file

Argument: ``-outputroot %s``

root directory for output

Argument: ``-outputroot %s``

root directory for output

args: (a unicode string)

Additional parameters to the command

Argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a
55.4.15 TrackBootstrap

Link to code
Wraps the executable command track.
Performs bootstrap streamline tractography using multiple scans of the same subject

Example

```python
>>> import nipype.interfaces.camino as cmon
>>> track = cmon.TrackBootstrap()
>>> track.inputs.inputmodel='repbs_dt'
>>> track.inputs.scheme_file = 'bvecs.scheme'
>>> track.inputs.bsdatafiles = ['fitted_data1.Bfloat', 'fitted_data2.Bfloat']
>>> track.inputs.seed_file = 'seed_mask.nii'
>>> track.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
scheme_file: (a pathlike object or string representing an existing file)
The scheme file corresponding to the data being processed.
argument: `'--schemefile %s'`
bsdatafiles: (a list of items which are a pathlike object or string representing an existing file)
Specifies files containing raw data for repetition bootstrapping.
Use `--inputfile` for wild bootstrap data.
argument: `'--bsdatafile %s'`

[Optional]
iterations: (an integer (int or long))
Number of streamlines to generate at each seed point.
argument: `'--iterations %d'`
inversion: (an integer (int or long))
Tensor reconstruction algorithm for repetition bootstrapping.
Default is 1 (linear reconstruction, single tensor).
argument: `'--inversion %s'`
bgmask: (a pathlike object or string representing an existing file)
Provides the name of a file containing a background mask computed using, for example, FSL's bet2 program. The mask file contains zero in background voxels and non-zero in foreground.
argument: `'--bgmask %s'`
in_file: (a pathlike object or string representing an existing file)
input data file
argument: `'--inputfile %s'`, position: 1
seed_file: (a pathlike object or string representing an existing file)
seed file

Outputs:

tracked: (a pathlike object or string representing an existing file)
output file containing reconstructed tracts
argument: ``-seedfile %s``
inputmodel: ('dt' or 'multitensor' or 'sfpeak' or 'pico' or
'repbs_dt' or 'repbs_multitensor' or 'ballstick' or 'wildbs_dt' or
'bayesdirac' or 'bayesdirac_dt' or 'bedpostx_dyad' or 'bedpostx',
nipype default value: dt)
input model type
argument: ``-inputmodel %s``
tracker: ('fact' or 'euler' or 'rk4', nipype default value: fact)
The tracking algorithm controls streamlines are generated from the
data. The choices are: - FACT, which follows the local fibre
orientation in each voxel. No interpolation is used.- EULER, which
uses a fixed step size along the local fibre orientation. With
nearest-neighbour interpolation, this method may be very similar to
FACT, whereas FACT steps extend to the boundary of the next voxel (distance variable depending on
the entry and exit points to the voxel).- RK4: Fourth-order Runge-
Kutta method. The step size is fixed, however the eventual direction of
the step is determined by taking and averaging a series of
partial steps.
argument: ``-tracker %s``
interpolator: ('nn' or 'prob_nn' or 'linear')
The interpolation algorithm determines how the fiber orientation(s)
are defined at a given continuous point within the input image.
Interpolator are only used when the tracking algorithm is not FACT.
The choices are: - NN: Nearest-neighbour interpolation, just uses
the local voxel data directly.- PROB_NN: Probabilistic nearest-
neighbor interpolation, similar to the method pro- posed by Behrens
et al [Magnetic Resonance in Medicine, 50:1077-1088, 2003]. The data
is not interpolated, but at each point we randomly choose one of the
8 voxels sur- rounding a point. The probability of choosing a
particular voxel is based on how close the point is to the centre of
that voxel.- LINEAR: Linear interpolation of the vector field
containing the principal directions at each point.
argument: ``-interpolator %s``
stepsize: (a float)
Step size for EULER and RK4 tracking. The default is 1mm.
argument: ``-stepsize %f``
requires: tracker
inputdatatype: ('float' or 'double')
input file type
argument: ``-inputdatatype %s``
gzip: (a boolean)
save the output image in gzip format
argument: ``-gzip``
maxcomponents: (an integer (int or long))
The maximum number of tensor components in a voxel. This determines
the size of the input file and does not say anything about the voxel
classification. The default is 2 if the input model is multitensor
and 1 if the input model is dt.
argument: ``-maxcomponents %d``
numpds: (an integer (int or long))
The maximum number of PDs in a voxel for input models sfpeak and
pico. The default is 3 for input model sfpeak and 1 for input model
pico. This option determines the size of the voxels in the input
file and does not affect tracking. For tensor data, use the
-maxcomponents option.
argument: ``-numpds %d``
data_dims: (a list of from 3 to 3 items which are an integer (int or long))
  data dimensions in voxels
  argument: `--datadims %s`

voxel_dims: (a list of from 3 to 3 items which are a float)
  voxel dimensions in mm
  argument: `--voxeldims %s`

ipthresh: (a float)
  Curvature threshold for tracking, expressed as the minimum dot
  product between two streamline orientations calculated over the
  length of a voxel. If the dot product between the previous and
  current directions is less than this threshold, then the streamline
  terminates. The default setting will terminate fibres that curve by
  more than 80 degrees. Set this to -1.0 to disable curvature checking
  completely.
  argument: `--ipthresh %f`

curvethresh: (a float)
  Curvature threshold for tracking, expressed as the maximum angle (in
  degrees) between between two streamline orientations calculated over
  the length of a voxel. If the angle is greater than this, then the
  streamline terminates.
  argument: `--curvethresh %f`

curveinterval: (a float)
  Interval over which the curvature threshold should be evaluated, in
  mm. The default is 5mm. When using the default curvature threshold
  of 90 degrees, this means that streamlines will terminate if they
  curve by more than 90 degrees over a path length of 5mm.
  argument: `--curveinterval %f`
  requires: curvethresh

anisthresh: (a float)
  Terminate fibres that enter a voxel with lower anisotropy than the
  threshold.
  argument: `--anisthresh %f`

anisfile: (a pathlike object or string representing an existing file)
  File containing the anisotropy map. This is required to apply an
  anisotropy threshold with non tensor data. If the map issupplied it
  is always used, even in tensor data.
  argument: `--anisfile %s`

outputtracts: ('float' or 'double' or 'oogl')
  output tract file type
  argument: `--outputtracts %s`

out_file: (a pathlike object or string representing a file)
  output data file
  argument: `--outputfile %s`, position: -1

output_root: (a pathlike object or string representing a file)
  root directory for output
  argument: `--outputroot %s`, position: -1

args: (a unicode string)
  Additional parameters to the command
  argument: `%%s`

environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a value
  of class 'str', nipype default value: {})
55.4.16 TrackDT

Link to code
Wraps the executable command track.
Performs streamline tractography using tensor data

Example

```python
>>> import nipype.interfaces.camino as cmon
>>> track = cmon.TrackDT()
>>> track.inputs.in_file = 'tensor_fitted_data.Bdouble'
>>> track.inputs.seed_file = 'seed_mask.nii'
>>> track.run() # doctest: +SKIP
```

Inputs:

- **in_file**: (a pathlike object or string representing an existing file)
  - input data file
    - argument: `''-inputfile %s''`, position: 1
  
- **seed_file**: (a pathlike object or string representing an existing file)
  - seed file
    - argument: `''-seedfile %s''`, position: 2
  
- **inputmodel**: ('dt' or 'multitensor' or 'sfpeak' or 'pico' or 'repbs_dt' or 'repbs_multitensor' or 'ballstick' or 'wildbs_dt' or 'bayesdirac' or 'bayesdirac_dt' or 'bedpostx_dyad' or 'bedpostx', nipype default value: dt)
  - input model type
    - argument: `''-inputmodel %s''`
  
- **tracker**: ('fact' or 'euler' or 'rk4', nipype default value: fact)
  - The tracking algorithm controls streamlines are generated from the data. The choices are:
    - FACT, which follows the local fibre orientation in each voxel. No interpolation is used.
    - EULER, which uses a fixed step size along the local fibre orientation. With nearest-neighbour interpolation, this method may be very similar to FACT, except that the step size is fixed, whereas FACT steps extend to the boundary of the next voxel (distance variable depending on the entry and exit points to the voxel).
    - RK4: Fourth-order Runge-Kutta method. The step size is fixed, however the eventual direction of the step is determined by taking and averaging a series of partial steps.

- **interpolator**: ('nn' or 'prob_nn' or 'linear')
  - The interpolation algorithm determines how the fiber orientation(s) are defined at a given continuous point within the input image. Interpolators are only used when the tracking algorithm is not FACT. The choices are:
    - NN: Nearest-neighbour interpolation, just uses the local voxel data directly.
    - PROB_NN: Probabilistic nearest-neighbor interpolation, similar to the method proposed by Behrens et al [Magnetic Resonance in Medicine, 50:1077-1088, 2003]. The data is not interpolated, but at each point we randomly choose one of the 8 voxels surrounding a point. The probability of choosing a
particular voxel is based on how close the point is to the centre of that voxel.- LINEAR: Linear interpolation of the vector field containing the principal directions at each point.

**argument:** `--interpolator %s`

**stepsize:** (a float)
Step size for EULER and RK4 tracking. The default is 1mm.

**argument:** `--stepsize %f`

**requires:** tracker

**inputdatatype:** ('float' or 'double')
Input file type

**argument:** `--inputdatatype %s`

**gzip:** (a boolean)
Save the output image in gzip format

**argument:** `--gzip`

**maxcomponents:** (an integer (int or long))
The maximum number of tensor components in a voxel. This determines the size of the input file and does not say anything about the voxel classification. The default is 2 if the input model is multitensor and 1 if the input model is dt.

**argument:** `--maxcomponents %d`

**numpds:** (an integer (int or long))
The maximum number of PDs in a voxel for input models sfpeak and pico. The default is 3 for input model sfpeak and 1 for input model pico. This option determines the size of the voxels in the input file and does not affect tracking. For tensor data, use the `-maxcomponents` option.

**argument:** `--numpds %d`

**data_dims:** (a list of from 3 to 3 items which are an integer (int or long))
Data dimensions in voxels

**argument:** `--datadims %s`

**voxel_dims:** (a list of from 3 to 3 items which are a float)
Voxel dimensions in mm

**argument:** `--voxeldims %s`

**ipthresh:** (a float)
Curvature threshold for tracking, expressed as the minimum dot product between two streamline orientations calculated over the length of a voxel. If the dot product between the previous and current directions is less than this threshold, then the streamline terminates. The default setting will terminate fibres that curve by more than 80 degrees. Set this to -1.0 to disable curvature checking completely.

**argument:** `--ipthresh %f`

**curvethresh:** (a float)
Curvature threshold for tracking, expressed as the maximum angle (in degrees) between between two streamline orientations calculated over the length of a voxel. If the angle is greater than this, then the streamline terminates.

**argument:** `--curvethresh %f`

**curveinterval:** (a float)
Interval over which the curvature threshold should be evaluated, in mm. The default is 5mm. When using the default curvature threshold of 90 degrees, this means that streamlines will terminate if they curve by more than 90 degrees over a path length of 5mm.

**argument:** `--curveinterval %f`

**requires:** curvethresh

**anisthresh:** (a float)
Terminate fibres that enter a voxel with lower anisotropy than the threshold.
argument: `--anisthresh %f`

anisfile: (a pathlike object or string representing an existing file)
File containing the anisotropy map. This is required to apply an anisotropy threshold with non tensor data. If the map is supplied it is always used, even in tensor data.
argument: `--anisfile %s`

outputtracts: ('float' or 'double' or 'oogl')
output tract file type
argument: `--outputtracts %s`

out_file: (a pathlike object or string representing a file)
output data file
argument: `--outputfile %s`, position: -1

output_root: (a pathlike object or string representing a file)
root directory for output
argument: `--outputroot %s`, position: -1

args: (a unicode string)
Additional parameters to the command
argument: `--%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

tracked: (a pathlike object or string representing an existing file)
output file containing reconstructed tracts

55.4.17 TrackPICo

Link to code
Wraps the executable command track.
Performs streamline tractography using the Probabilistic Index of Connectivity (PICO) algorithm

Example

```python
>>> import nipype.interfaces.camino as cmon
>>> track = cmon.TrackPICo()
>>> track.inputs.in_file = 'pdfs.Bfloat'
>>> track.inputs.seed_file = 'seed_mask.nii'
>>> track.run()  # doctest: +SKIP
```

Inputs:

[Optional]
pdf: ('bingham' or 'watson' or 'acg')
Specifies the model for PICO parameters. The default is "bingham.
argument: `--pdf %s`

iterations: (an integer (int or long))
Number of streamlines to generate at each seed point. The default is 5000.
argument: `--iterations %d`

in_file: (a pathlike object or string representing an existing file)
input data file

(continues on next page)
argument: ```-inputfile %s```, position: 1

seed file: (a pathlike object or string representing an existing file)

seed file

argument: ```-seedfile %s```, position: 2

inputmodel: ("dt" or "multitensor" or "sfpeak" or "pico" or "repbs_dt" or "repbs_multitensor" or "ballstick" or "wildbs_dt" or "bayesdirac" or "bayesdirac_dt" or "bedpostx_dyad" or "bedpostx", nipype default value: dt)

input model type

argument: ```-inputmodel %s```

tracker: ("fact" or "euler" or "rk4", nipype default value: fact)

The tracking algorithm controls streamlines are generated from the data. The choices are: - FACT, which follows the local fibre orientation in each voxel. No interpolation is used.- EULER, which uses a fixed step size along the local fibre orientation. With nearest-neighbour interpolation, this method may be very similar to FACT, except that the step size is fixed, whereas FACT steps extend to the boundary of the next voxel (distance variable depending on the entry and exit points to the voxel).- RK4: Fourth-order Runge-Kutta method. The step size is fixed, however the eventual direction of the step is determined by taking and averaging a series of partial steps.

argument: ```-tracker %s```

interpolator: ("nn" or "prob_nn" or "linear")

The interpolation algorithm determines how the fiber orientation(s) are defined at a given continuous point within the input image. Interpolators are only used when the tracking algorithm is not FACT. The choices are: - NN: Nearest-neighbour interpolation, just uses the local voxel data directly.- PROB_NN: Probabilistic nearest-neighbor interpolation, similar to the method pro- posed by Behrens et al [Magnetic Resonance in Medicine, 50:1077-1088, 2003]. The data is not interpolated, but at each point we randomly choose one of the 8 voxels sur- rounding a point. The probability of choosing a particular voxel is based on how close the point is to the centre of that voxel.- LINEAR: Linear interpolation of the vector field containing the principal directions at each point.

argument: ```-interpolator %s```

stepsize: (a float)

Step size for EULER and RK4 tracking. The default is 1mm.

argument: ```-stepsize %f```

requires: tracker

inputdatatype: ("float" or "double")

input file type

argument: ```-inputdatatype %s```

gzip: (a boolean)

save the output image in gzip format

argument: ```-gzip```

maxcomponents: (an integer (int or long))

The maximum number of tensor components in a voxel. This determines the size of the input file and does not say anything about the voxel classification. The default is 2 if the input model is multitensor and 1 if the input model is dt.

argument: ```-maxcomponents %d```

numpds: (an integer (int or long))

The maximum number of PDs in a voxel for input models sfpeak and pico. The default is 3 for input model sfpeak and 1 for input model...
pico. This option determines the size of the voxels in the input file and does not affect tracking. For tensor data, use the -maxcomponents option.

argument: ``-numpds %d``

data_dims: (a list of from 3 to 3 items which are an integer (int or long))
data dimensions in voxels
argument: ``-datadims %s``

voxel_dims: (a list of from 3 to 3 items which are a float)
voxel dimensions in mm
argument: ``-voxeldims %s``

ipthresh: (a float)
Curvature threshold for tracking, expressed as the minimum dot product between two streamline orientations calculated over the length of a voxel. If the dot product between the previous and current directions is less than this threshold, then the streamline terminates. The default setting will terminate fibres that curve by more than 80 degrees. Set this to -1.0 to disable curvature checking completely.

argument: ``-ipthresh %f``

curvethresh: (a float)
Curvature threshold for tracking, expressed as the maximum angle (in degrees) between between two streamline orientations calculated over the length of a voxel. If the angle is greater than this, then the streamline terminates.

argument: ``-curvethresh %f``

curveinterval: (a float)
Interval over which the curvature threshold should be evaluated, in mm. The default is 5mm. When using the default curvature threshold of 90 degrees, this means that streamlines will terminate if they curve by more than 90 degrees over a path length of 5mm.

argument: ``-curveinterval %f``

requires: curvethresh

anisthresh: (a float)
Terminate fibres that enter a voxel with lower anisotropy than the threshold.

argument: ``-anisthresh %f``

anisfile: (a pathlike object or string representing an existing file)
File containing the anisotropy map. This is required to apply an anisotropy threshold with non tensor data. If the map is supplied it is always used, even in tensor data.

argument: ``-anisfile %s``

outputtracts: ('float' or 'double' or 'oogl')
output tract file type

argument: ``-outputtracts %s``

out_file: (a pathlike object or string representing a file)
output data file

argument: ``-outputfile %s``, position: -1

output_root: (a pathlike object or string representing a file)
root directory for output

argument: ``-outputroot %s``, position: -1

args: (a unicode string)
Additional parameters to the command

argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

tracked: (a pathlike object or string representing an existing file)
output file containing reconstructed tracts

55.5 interfaces.camino.odf

55.5.1 LinRecon

Link to code
Wraps the executable command linrecon.
Runs a linear transformation in each voxel.
Reads a linear transformation from the matrix file assuming the imaging scheme specified in the scheme file.
Performs the linear transformation on the data in every voxel and outputs the result to the standard output. The output in every voxel is actually:

\[ [\text{exit code}, \ln(S(0)), p_1, \ldots, p_R] \]

where \( p_1, \ldots, p_R \) are the parameters of the reconstruction. Possible exit codes are:

- 0. No problems.
- 6. Bad data replaced by substitution of zero.

The matrix must be \( R \) by \( N+M \) where \( N+M \) is the number of measurements and \( R \) is the number of parameters of the reconstruction. The matrix file contains binary double-precision floats. The matrix elements are stored row by row.

Example

First run QBallMX and create a linear transform matrix using Spherical Harmonics (sh).

```python
>>> import nipype.interfaces.camino as cam
>>> qballmx = cam.QBallMX()
>>> qballmx.inputs.scheme_file = 'A.scheme'
>>> qballmx.inputs.basistype = 'sh'
>>> qballmx.inputs.order = 4
>>> qballmx.run()  # doctest: +SKIP
```

Then run it over each voxel using LinRecon

```python
>>> qballcoeffs = cam.LinRecon()
>>> qballcoeffs.inputs.in_file = 'SubjectA.Bfloat'
>>> qballcoeffs.inputs.scheme_file = 'A.scheme'
>>> qballcoeffs.inputs.qball_mat = 'A_qmat.Bdouble'
>>> qballcoeffs.inputs.normalize = True
>>> qballcoeffs.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
voxel-order data filename
argument: `\$s\`, position: 1
scheme_file: (a pathlike object or string representing an existing file)
  Specifies the scheme file for the diffusion MRI data
55.5.2 MESD

Link to code

Wraps the executable command mesd.

MESD is a general program for maximum entropy spherical deconvolution. It also runs PASMRI, which is a special case of spherical deconvolution. The input data must be in voxel order. The format of the output in each voxel is: { exitcode, ln(A^*star(0)), lambda_0, lambda_1, ..., lambda_N }
The exitcode contains the results of three tests. The first test thresholds the maximum relative error between the numerical integrals computed at convergence and those computed using a larger test point set; if the error is greater than a threshold the exitcode is increased from zero to one as a warning; if it is greater than a larger threshold the exitcode is increased to two to suggest failure. The second test thresholds the predicted error in numerical integrals computed using the test point set; if the predicted error is greater than a threshold the exitcode is increased by 10. The third test thresholds the RMS error between the measurements and their predictions from the fitted deconvolution; if the errors are greater than a threshold, the exit code is increased by 100. An exitcode of 112 means that all three tests were failed and the result is likely to be unreliable. If all is well the exitcode is zero. Results are often still reliable even if one or two of the tests are failed.

Other possible exitcodes are:

- 5 - The optimization failed to converge
- -1 - Background
- -100 - Something wrong in the MRI data, e.g. negative or zero measurements, so that the optimization could not run.

The standard MESD implementation is computationally demanding, particularly as the number of measurements
increases (computation is approximately $O(N^2)$, where $N$ is the number of measurements). There are two ways
to obtain significant computational speed-up:
i) Turn off error checks and use a small point set for computing numerical integrals in the algorithm by adding
the flag -fastmesd. Sakaie CDMRI 2008 shows that using the smallest point set (-basepointset 0) with no
error checks usually has only a minor effect on the output of the algorithm, but provides a major reduction in
computation time. You can increase the point set size using -basepointset with an argument higher than 0, which
may produce better results in some voxels, but will increase computation time, which approximately doubles
every time the point set index increases by 1.

ii) Reduce the complexity of the maximum entropy encoding using -mepointset $<X>$. By default $<X> = N$,
the number of measurements, and is the number of parameters in the max. ent. representation of the output
function, ie the number of lambda parameters, as described in Jansons and Alexander Inverse Problems 2003.
However, we can represent the function using less components and $<X>$ here specifies the number of lambda
parameters. To obtain speed-up, set $<X> < N$; complexity become $O(<X>^2)$ rather than $O(N^2)$. Note that
$<X>$ must be chosen so that the camino/PointSets directory contains a point set with that number of elements.
When -mepointset decreases, the numerical integration checks make less and less of a difference and smaller
point sets for numerical integration (see -basepointset) become adequate. So when $<X>$ is low -fastmesd is
worth using to get even more speed-up.
The choice of $<X>$ is a parameter of the technique. Too low and you lose angular resolution; too high and you
see no computational benefit and may even suffer from overfitting. Empirically, we have found that $<X>$=16
often gives good results and good speed up, but it is worth trying a few values a comparing performance. The
reduced encoding is described in the following ISMRM abstract: Sweet and Alexander “Reduced Encoding
Persistent Angular Structure” 572 ISMRM 2010.

Example

Run MESD on every voxel of the data file SubjectA.Bfloat using the PASMRA kernel.

```python
>>> import nipype.interfaces.camino as cam
>>> mesd = cam.MESD()
>>> mesd.inputs.in_file = 'SubjectA.Bfloat'
>>> mesd.inputs.scheme_file = 'A.scheme'
>>> mesd.inputs.inverter = 'PAS'
>>> mesd.inputs.inverter_param = 1.4
>>> mesd.run() # doctest: +SKIP
```

Inputs:

[Mandatory]

- **in_file**: (a pathlike object or string representing an existing file)
  - voxel-order data filename
  - argument: `"-inputfile %s"`, position: 1

- **inverter**: ('SPIKE' or 'PAS')
  - The inversion index specifies the type of inversion to perform on
  the data. The currently available choices are: Inverter name | Inverter parameters--------------------------|--------------------------SPIKE | bd
  - (b-value x diffusivity along the fibre.) PAS | r
  - argument: `"-filter %s"`, position: 2

- **inverter_param**: (a float)
  - Parameter associated with the inverter. Cf. inverter description
  for more information.
  - argument: `"%f"`, position: 3

- **scheme_file**: (a pathlike object or string representing an existing file)
  - Specifies the scheme file for the diffusion MRI data
  - argument: `"-schemefile %s"`

[Optional]
fastmesd: (a boolean)
    Turns off numerical integration checks and fixes the integration point set size at that of the index specified by -basepointset.
    argument: `--fastmesd`
    requires: mepointset
mepointset: (an integer (int or long))
    Use a set of directions other than those in the scheme file for the deconvolution kernel. The number refers to the number of directions on the unit sphere. For example, `--mepointset 54` uses the directions in "camino/PointSets/Elec054.txt".
    argument: `--mepointset %d`
bgmask: (a pathlike object or string representing an existing file)
    background mask
    argument: `--bgmask %s`
inputdatatype: ('float' or 'char' or 'short' or 'int' or 'long' or 'double')
    Specifies the data type of the input file: "char", "short", "int", "long", "float" or "double". The input file must have BIG-ENDIAN ordering. By default, the input type is "float".
    argument: `--inputdatatype %s`
out_file: (a pathlike object or string representing a file)
    argument: `> %s`, position: -1
args: (a unicode string)
    Additional parameters to the command
    argument: `%%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
    Environment variables

Outputs:

mesd_data: (a pathlike object or string representing an existing file)
    MESD data

55.5.3 QBallMX

Link to code
Wraps the executable command qballmx.
Generates a reconstruction matrix for Q-Ball. Used in LinRecon with the same scheme file to reconstruct data.

Example 1
To create a linear transform matrix using Spherical Harmonics (sh).

```python
>>> import nipype.interfaces.camino as cam
>>> qballmx = cam.QBallMX()
>>> qballmx.inputs.scheme_file = 'A.scheme'
>>> qballmx.inputs.basistype = 'sh'
>>> qballmx.inputs.order = 6
>>> qballmx.run()  # doctest: +SKIP
```

Example 2
To create a linear transform matrix using Radial Basis Functions (rbf). This command uses the default setting of rbf sigma = 0.2618 (15 degrees), data smoothing sigma = 0.1309 (7.5 degrees), rbf pointset 246
The linear transform matrix from any of these two examples can then be run over each voxel using LinRecon:

```python
>>> qballcoeffs = cam.LinRecon()
>>> qballcoeffs.inputs.in_file = 'SubjectA.Bfloat'
>>> qballcoeffs.inputs.scheme_file = 'A.scheme'
>>> qballcoeffs.inputs.qball_mat = 'A_qmat.Bdouble'
>>> qballcoeffs.inputs.normalize = True
>>> qballcoeffs.inputs.bgmask = 'brain_mask.nii'
>>> qballcoeffs.run()  # doctest: +SKIP
```

**Inputs:**

[Mandatory]

- `scheme_file`: (a pathlike object or string representing an existing file)

  Specifies the scheme file for the diffusion MRI data

  argument: ``-schemefile %s``

[Optional]

- `basistype`: ('rbf' or 'sh', nipype default value: rbf)

  Basis function type. "rbf" to use radial basis functions "sh" to use spherical harmonics

  argument: ``-basistype %s``

- `order`: (an integer (int or long))

  Specific to sh. Maximum order of the spherical harmonic series.

  Default is 4.

  argument: ``-order %d``

- `rbfpointset`: (an integer (int or long))

  Specific to rbf. Sets the number of radial basis functions to use.

  The value specified must be present in the Pointsets directory. The default value is 246.

  argument: ``-rbfpointset %d``

- `rbfsigma`: (a float)

  Specific to rbf. Sets the width of the interpolating basis functions. The default value is 0.2618 (15 degrees).

  argument: ``-rbfsigma %f``

- `smoothingsigma`: (a float)

  Specific to rbf. Sets the width of the smoothing basis functions.

  The default value is 0.1309 (7.5 degrees).

  argument: ``-smoothingsigma %f``

- `out_file`: (a pathlike object or string representing a file)

  argument: ``> %s``, position: -1

- `args`: (a unicode string)

  Additional parameters to the command

  argument: ``%s``

- `environ`: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

  Environment variables

**Outputs:**

- `qmat`: (a pathlike object or string representing an existing file)

  Q-Ball reconstruction matrix
55.5.4 SFPeaks

Link to code
Wraps the executable command `sfpeaks`.
Finds the peaks of spherical functions.
This utility reads coefficients of the spherical functions and outputs a list of peak directions of the function. It computes the value of the function at each of a set of sample points. Then it finds local maxima by finding all points at which the function is larger than for any other point within a fixed search radius (the default is 0.4). The utility then uses Powell’s algorithm to optimize the position of each local maximum. Finally the utility removes duplicates and tiny peaks with function value smaller than some threshold, which is the mean of the function plus some number of standard deviations. By default the program checks for con-sistency with a second set of starting points, but skips the optimization step. To speed up execution, you can turn off the con-sistency check by setting the noconsistencycheck flag to True.

By default, the utility constructs a set of sample points by randomly rotating a unit icosahedron repeatedly (the default is 1000 times, which produces a set of 6000 points) and concatenating the lists of vertices. The ‘pointset = <index>’ attribute can tell the utility to use an evenly distributed set of points (index 0 gives 1082 points, 1 gives 1922, 2 gives 4322, 3 gives 8672, 4 gives 15872, 5 gives 32762, 6 gives 72032), which is quicker, because you can get away with fewer points. We estimate that you can use a factor of 2.5 less evenly distributed points than randomly distributed points and still expect similar performance levels.

The output for each voxel is:
• exitcode (inherited from the input data).
• ln(A(0))
• number of peaks found.
• flag for consistency with a repeated run (number of directions is the same and the directions are the same to within a threshold.)
• mean(f).
• std(f).
• direction 1 (x, y, z, f, H00, H01, H10, H11).
• direction 2 (x, y, z, f, H00, H01, H10, H11).
• direction 3 (x, y, z, f, H00, H01, H10, H11).

H is the Hessian of f at the peak. It is the matrix:

\[
\begin{bmatrix}
\frac{d^2f}{ds^2} & \frac{d^2f}{dsdt} \\
\frac{d^2f}{dt^2} & \frac{d^2f}{dt^2}
\end{bmatrix}
= \begin{bmatrix}
H00 & H01 \\
H10 & H11
\end{bmatrix}
\]

where s and t are orthogonal coordinates local to the peak.
By default the maximum number of peak directions output in each voxel is three. If less than three directions are found, zeros are output for later directions. The peaks are ordered by the value of the function at the peak. If more than the maximum number of directions are found only the strongest ones are output. The maximum number can be changed setting the ‘numpds’ attribute.

The utility can read various kinds of spherical function, but must be told what kind of function is input using the ‘inputmodel’ attribute. The description of the ‘inputmodel’ attribute lists additional information required by SFPeaks for each input model.

Example
First run QBallMX and create a linear transform matrix using Spherical Harmonics (sh).

```python
>>> import nipype.interfaces.camino as cam
>>> sf_peaks = cam.SFPeaks()
>>> sf_peaks.inputs.in_file = 'A_recon_params.Bdouble'
>>> sf_peaks.inputs.inputmodel = 'sh'
>>> sf_peaks.inputs.order = 4
>>> sf_peaks.inputs.density = 100
```
>>> sf_peaks.inputs.searchradius = 1.0
>>> sf_peaks.run()  # doctest: +SKIP

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  Voxel-order data of spherical functions
  argument: '-inputfile %s'
inputmodel: ('sh' or 'maxent' or 'rbf')
  Type of functions input via in_file. Currently supported options
  are: sh - Spherical harmonic series. Specify the maximum order of
  the SH series with the "order" attribute if different from the
  default of 4. maxent - Maximum entropy representations output by
  MESD. The reconstruction directions input to MESD must be specified.
  By default this is the same set of gradient directions (excluding
  zero gradients) in the scheme file, so specify the "schemefile"
  attribute unless the "mepointset" attribute was set in MESD. rbf -
  Sums of radial basis functions. Specify the pointset with the
  attribute "rbfpointset" if different from the default. See QBallMx.
  argument: '-inputmodel %s'

[Optional]
order: (an integer (int or long))
  Specific to sh. Maximum order of the spherical harmonic series.
  argument: '-order %d'
scheme_file: (a pathlike object or string representing an existing
  file)
  Specific to maxent. Specifies the scheme file.
  argument: '%s'
rbfpointset: (an integer (int or long))
  Specific to rbf. Sets the number of radial basis functions to use.
  The value specified must be present in the Pointsets directory. The
  default value is 246.
  argument: '-rbfpointset %d'
mepointset: (an integer (int or long))
  Use a set of directions other than those in the scheme file for the
  deconvolution kernel. The number refers to the number of directions
  on the unit sphere. For example, "mepointset = 54" uses the
  directions in "camino/PointSets/Elec054.txt" Use this option only if
  you told MESD to use a custom set of directions with the same
  option. Otherwise, specify the scheme file with the "schemefile"
  attribute.
  argument: '-mepointset %d'
numpds: (an integer (int or long))
  The largest number of peak directions to output in each voxel.
  argument: '-numpds %d'
noconsistencycheck: (a boolean)
  Turns off the consistency check. The output shows all consistencies
  as true.
  argument: '-noconsistencycheck'
searchradius: (a float)
  The search radius in the peak finding algorithm. The default is 0.4
  (cf. "density")
  argument: '-searchradius %f'
density: (an integer (int or long))
  The number of randomly rotated icosahedra to use in constructing the
(continues on next page)
set of points for random sampling in the peak finding algorithm. Default is 1000, which works well for very spiky maxent functions. For other types of function, it is reasonable to set the density much lower and increase the search radius slightly, which speeds up the computation.

argument: `--density %d`

pointset: (an integer (int or long))
To sample using an evenly distributed set of points instead. The integer can be 0, 1, ..., 7. Index 0 gives 1082 points, 1 gives 1922, 2 gives 3002, 3 gives 4322, 4 gives 5882, 5 gives 8672, 6 gives 12002, 7 gives 15872.

argument: `--pointset %d`

pdthresh: (a float)
Base threshold on the actual peak direction strength divided by the mean of the function. The default is 1.0 (the peak must be equal or greater than the mean).

argument: `--pdthresh %f`

stdsfrommean: (a float)
This is the number of standard deviations of the function to be added to the "pdthresh" attribute in the peak directions pruning.

argument: `--stdsfrommean %f`

out_file: (a pathlike object or string representing a file)

argument: `>` %s`, position: -1

args: (a unicode string)
Additional parameters to the command

argument: `"%s"`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environmental variables

Outputs:

peaks: (a pathlike object or string representing an existing file)
Peaks of the spherical functions.

55.6 interfaces.camino.utils

55.6.1 ImageStats

Link to code
Wraps the executable command `imagestats`.
This program computes voxelwise statistics on a series of 3D images. The images must be in the same space; the operation is performed voxelwise and one output is produced per voxel.

Examples

```python
>>> import nipype.interfaces.camino as cam
>>> imstats = cam.ImageStats()
>>> imstats.inputs.in_files = ['im1.nii','im2.nii','im3.nii']
>>> imstats.inputs.stat = 'max'
>>> imstats.run() # doctest: +SKIP
```

Inputs:
**[Mandatory]**

in_files: (a list of items which are a pathlike object or string representing an existing file)
- List of images to process. They must be in the same space and have the same dimensions.
  - argument: `--images %s`, position: -1

stat: ('min' or 'max' or 'mean' or 'median' or 'sum' or 'std' or 'var')
- The statistic to compute.
  - argument: `--stat %s`

output_root: (a pathlike object or string representing a file)
- Filename root prepended onto the names of the output files. The extension will be determined from the input.
  - argument: `--outputroot %s`

**[Optional]**

out_type: ('float' or 'char' or 'short' or 'int' or 'long' or 'double', nipype default value: float)
- A Camino data type string, default is "float". Type must be signed.
  - argument: `--outputdatatype %s`

args: (a unicode string)
- Additional parameters to the command
  - argument: `%%%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
- Environment variables

**Outputs:**

out_file: (a pathlike object or string representing an existing file)
- Path of the file computed with the statistic chosen
56.1 interfaces.camino2trackvis.convert

56.1.1 Camino2Trackvis

Link to code
Wraps the executable command camino_to_trackvis.
Wraps camino_to_trackvis from Camino-Trackvis
Convert files from camino .Bfloat format to trackvis .trk format.

Example

```python
>>> import nipype.interfaces.camino2trackvis as cam2trk
>>> c2t = cam2trk.Camino2Trackvis()
>>> c2t.inputs.in_file = 'data.Bfloat'
>>> c2t.inputs.out_file = 'streamlines.trk'
>>> c2t.inputs.min_length = 30
>>> c2t.inputs.data_dims = [128, 104, 64]
>>> c2t.inputs.voxel_dims = [2.0, 2.0, 2.0]
>>> c2t.inputs.voxel_order = 'LAS'
>>> c2t.run() # doctest: +SKIP
```

Inputs:

[Mandatory]

in_file: (a pathlike object or string representing an existing file)
The input .Bfloat (camino) file.
argument: `''-i %s''`, position: 1

data_dims: (a list of from 3 to 3 items which are an integer (int or long))
Three comma-separated integers giving the number of voxels along each dimension of the source scans.
argument: `''-d %s''`, position: 4

voxel_dims: (a list of from 3 to 3 items which are a float)
Three comma-separated numbers giving the size of each voxel in mm.
argument: `''-x %s''`, position: 5

voxel_order: (a pathlike object or string representing a file)
Set the order in which various directions were stored. Specify with three letters consisting of one each from the pairs LR, AP, and SI. These stand for Left-Right, Anterior-Posterior, and Superior-Inferior. Whichever is specified in each position will be the direction of increasing order. Read coordinate system from a NIfTI file.

argument: `--voxel-order %s`, position: 6

[Optional]
out_file: (a pathlike object or string representing a file)
The filename to which to write the .trk (trackvis) file.
argument: `%o %s` , position: 2

min_length: (a float)
The minimum length of tracts to output
argument: `%1 %d` , position: 3

nifti_file: (a pathlike object or string representing an existing file)
Read coordinate system from a NIfTI file.
argument: `--nifti %s` , position: 7

args: (a unicode string)
Additional parameters to the command
argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

trackvis: (a pathlike object or string representing an existing file)
The filename to which to write the .trk (trackvis) file.

56.1.2 Trackvis2Camino

Link to code
Wraps the executable command trackvis_to_camino.

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
The input .trk (trackvis) file.
argument: `%i %s` , position: 1

[Optional]
out_file: (a pathlike object or string representing a file)
The filename to which to write the .Bfloat (camino).
argument: `%o %s` , position: 2

append_file: (a pathlike object or string representing an existing file)
A file to which the append the .Bfloat data.
argument: `%a %s` , position: 2

args: (a unicode string)
Additional parameters to the command
argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a
Outputs:

`camino`: (a pathlike `object` or `string` representing an existing file)

The filename to which to write the `.Bfloat` (`camino`).
57.1 interfaces.cmtk.base

57.1.1 CFFBaseInterface

Link to code

**Inputs:**

None

**Outputs:**

None

57.2 interfaces.cmtk.cmtk

57.2.1 CreateMatrix

Link to code

Performs connectivity mapping and outputs the result as a NetworkX graph and a Matlab matrix

**Example**

```python
>>> import nipype.interfaces.cmtk as cmtk
>>> conmap = cmtk.CreateMatrix()
>>> conmap.roi_file = 'fsLUT_aparc+aseg.nii'
>>> conmap.tract_file = 'fibers.trk'
>>> conmap.run()    # doctest: +SKIP
```

**Inputs:**

[Mandatory]

- **roi_file**: (a pathlike object or string representing an existing file)
  - Freesurfer aparc+aseg file
- **tract_file**: (a pathlike object or string representing an existing file)

(continues on next page)
Trackvis tract file
resolution_network_file: (a pathlike object or string representing an
existing file)
Parcellation files from Connectome Mapping Toolkit

[Optional]
count_region_intersections: (a boolean, nipype default value: False)
Counts all of the fiber-region traversals in the connectivity matrix
(requires significantly more computational time)
out_matrix_file: (a pathlike object or string representing a file)
NetworkX graph describing the connectivity
out_matrix_mat_file: (a pathlike object or string representing a
file, nipype default value: cmatrix.mat)
Matlab matrix describing the connectivity
out_mean_fiber_length_matrix_mat_file: (a pathlike object or string
representing a file)
Matlab matrix describing the mean fiber lengths between each node.
out_median_fiber_length_matrix_mat_file: (a pathlike object or string
representing a file)
Matlab matrix describing the median fiber lengths between each node.
out_fiber_length_std_matrix_mat_file: (a pathlike object or string
representing a file)
Matlab matrix describing the deviation in fiber lengths connecting
each node.
out_intersection_matrix_mat_file: (a pathlike object or string
representing a file)
Matlab connectivity matrix if all region/fiber intersections are
counted.
out_endpoint_array_name: (a pathlike object or string representing a
file)
Name for the generated endpoint arrays

Outputs:

matrix_file: (a pathlike object or string representing an existing
file)
NetworkX graph describing the connectivity
intersection_matrix_file: (a pathlike object or string representing an
existing file)
NetworkX graph describing the connectivity
matrix_files: (a list of items which are a pathlike object or string
representing an existing file)
matrix_mat_files: (a list of items which are a pathlike object or
string representing an existing file)
matrix_mat_file: (a pathlike object or string representing an
existing file)
Matlab matrix describing the connectivity
intersection_matrix_mat_file: (a pathlike object or string
representing an existing file)
Matlab matrix describing the mean fiber lengths between each node.
mean_fiber_length_matrix_mat_file: (a pathlike object or string
representing an existing file)
Matlab matrix describing the mean fiber lengths between each node.
median_fiber_length_matrix_mat_file: (a pathlike object or string
representing an existing file)
Matlab matrix describing the median fiber lengths between each node.
fiber_length_std_matrix_mat_file: (a pathlike object or string
representing an existing file)
Matlab matrix describing the deviation in fiber lengths connecting
each node.
representing an existing file)
Matlab matrix describing the deviation in fiber lengths connecting each node.

endpoint_file: (a pathlike object or string representing an existing file)
Saved Numpy array with the endpoints of each fiber

endpoint_file_mm: (a pathlike object or string representing an existing file)
Saved Numpy array with the endpoints of each fiber (in millimeters)

fiber_length_file: (a pathlike object or string representing an existing file)
Saved Numpy array with the lengths of each fiber

fiber_label_file: (a pathlike object or string representing an existing file)
Saved Numpy array with the labels for each fiber

fiber_labels_noorphans: (a pathlike object or string representing an existing file)
Saved Numpy array with the labels for each non-orphan fiber

filtered_tractography: (a pathlike object or string representing an existing file)
TrackVis file containing only those fibers originate in one and terminate in another region

filtered_tractography_by_intersections: (a pathlike object or string representing an existing file)
TrackVis file containing all fibers which connect two regions

filtered_tractographies: (a list of items which are a pathlike object or string representing an existing file)

stats_file: (a pathlike object or string representing an existing file)
Saved Matlab .mat file with the number of fibers saved at each stage

57.2.2 CreateNodes

Link to code
Generates a NetworkX graph containing nodes at the centroid of each region in the input ROI file. Node data is added from the resolution network file.

Example

```python
>>> import nipype.interfaces.cmtk as cmtk
>>> mknode = cmtk.CreateNodes()
>>> mknode.inputs.roi_file = 'ROI_scale500.nii.gz'
>>> mknode.run() # doctest: +SKIP
```

Inputs:

- [Mandatory]
  - roi_file: (a pathlike object or string representing an existing file)
    Region of interest file
  - resolution_network_file: (a pathlike object or string representing an existing file)
    Parcellation file from Connectome Mapping Toolkit

- [Optional]
  - out_filename: (a pathlike object or string representing a file,
nipype default value: nodenetwork.pck
Output gpickled network with the nodes defined.

Outputs:

node_network: (a pathlike object or string representing a file)
Output gpickled network with the nodes defined.

57.2.3 ROIGen

Link to code
Generates a ROI file for connectivity mapping and a dictionary file containing relevant node information

Example

```python
>>> import nipype.interfaces.cmtk as cmtk
>>> rg = cmtk.ROIGen()
>>> rg.inputs.aparc_aseg_file = 'aparc+aseg.nii'
>>> rg.inputs.use_freesurfer_LUT = True
>>> rg.inputs.freesurfer_dir = '/usr/local/freesurfer'
>>> rg.run() # doctest: +SKIP

The label dictionary is written to disk using Pickle. Resulting data can be loaded using:

```python
>>> file = open("FreeSurferColorLUT_adapted_aparc+aseg_out.pck", "r")
>>> labelDict = pickle.load(file) # doctest: +SKIP

Inputs:

[Mandatory]
aparc_aseg_file: (a pathlike object or string representing an existing file)
    Freesurfer aparc+aseg file

[Optional]
LUT_file: (a pathlike object or string representing an existing file)
    Custom lookup table (cf. FreeSurferColorLUT.txt)
    mutually_exclusive: use_freesurfer_LUT
use_freesurfer_LUT: (a boolean)
    Boolean value; Set to True to use default Freesurfer LUT, False for custom LUT
    mutually_exclusive: LUT_file
freesurfer_dir: (a pathlike object or string representing a directory)
    Freesurfer main directory
    requires: use_freesurfer_LUT
out_roi_file: (a pathlike object or string representing a file)
    Region of Interest file for connectivity mapping
out_dict_file: (a pathlike object or string representing a file)
    Label dictionary saved in Pickle format

Outputs:

roi_file: (a pathlike object or string representing a file)
    Region of Interest file for connectivity mapping

(continues on next page)
dict_file: (a pathlike object or string representing a file)
   Label dictionary saved in Pickle format

57.2.4 cmat()

   Link to code
   Create the connection matrix for each resolution using fibers and ROIs.

57.2.5 create_allpoints_cmat()

   Link to code
   Create the intersection arrays for each fiber

57.2.6 create_endpoints_array()

   Link to code
   Create the endpoints arrays for each fiber
   Parameters
   ~~~~~~~~~~~
   fib: the fibers data
   voxelSize: 3-tuple containing the voxel size of the ROI image
   Returns
   ~~~~~~~
   (endpoints: matrix of size [#fibers, 2, 3] containing for each fiber the index of its first and last point in the voxelSize volume
   endpointsmm): endpoints in millimeter coordinates

57.2.7 create_nodes()

   Link to code

57.2.8 get_connectivity_matrix()

   Link to code

57.2.9 get_rois_crossed()

   Link to code

57.2.10 length()

   Link to code
   Euclidean length of track line

   Parameters
   ~~~~~~~
   xyz [array-like shape (N,3)] array representing x,y,z of N points in a track
   along [bool, optional] If True, return array giving cumulative length along track, otherwise (default) return scalar giving total length.

   Returns
   ~~~~
   L [scalar or array shape (N-1,)] scalar in case of along == False, giving total length, array if along == True, giving cumulative lengths.

   Examples
>>> xyz = np.array([[1,1,1],[2,3,4],[0,0,0]])
>>> expected_lens = np.sqrt([1+2**2+3**2, 2**2+3**2+4**2])
>>> length(xyz) == expected_lens.sum()
True
>>> len_along = length(xyz, along=True)
>>> np.allclose(len_along, expected_lens.cumsum())
True
>>> length([])
~
>>> length([[1, 2, 3]])
~
>>> length([], along=True)
array([0])

57.2.11 save_fibers()
Link to code
Stores a new trackvis file fname using only given indices

57.3 interfaces.cmtk.convert

57.3.1 CFFConverter
Link to code
Creates a Connectome File Format (CFF) file from input networks, surfaces, volumes, tracts, etcetera.

Example

```python
>>> import nipype.interfaces.cmtk as cmtk
>>> cvt = cmtk.CFFConverter()
>>> cvt.inputs.title = 'subject 1'
>>> cvt.inputs.gifti_surfaces = ['lh.pial_converted.gii', 'rh.pial_converted.gii']
>>> cvt.inputs.tract_files = ['streamlines.trk']
>>> cvt.inputs.gpickled_networks = ['network0.gpickle']
>>> cvt.run()  # doctest: +SKIP
```

Inputs:

[Optional]

- graphml_networks: (a list of items which are a pathlike object or string representing an existing file)
  list of graphML networks
- gpickled_networks: (a list of items which are a pathlike object or string representing an existing file)
  list of gpickled Networkx graphs
- gifti_surfaces: (a list of items which are a pathlike object or string representing an existing file)
  list of GIFTI surfaces
- gifti_labels: (a list of items which are a pathlike object or string representing an existing file)
  list of GIFTI labels
- nifti_volumes: (a list of items which are a pathlike object or string representing an existing file)
  list of NIFTI volumes
- tract_files: (a list of items which are a pathlike object or string representing an existing file)
  list of tract files

(continues on next page)
representing an existing file)
list of Trackvis fiber files
timeseries_files: (a list of items which are a pathlike object or string representing an existing file)
list of HDF5 timeseries files
script_files: (a list of items which are a pathlike object or string representing an existing file)
list of script files to include
data_files: (a list of items which are a pathlike object or string representing an existing file)
list of external data files (i.e. Numpy, HD5, XML)
title: (a unicode string)
  Connectome Title
creator: (a unicode string)
  Creator
eemail: (a unicode string)
  Email address
publisher: (a unicode string)
  Publisher
license: (a unicode string)
  License
rights: (a unicode string)
  Rights
references: (a unicode string)
  References
relation: (a unicode string)
  Relation
species: (a unicode string, nipype default value: Homo sapiens)
  Species
description: (a unicode string, nipype default value: Created with the Nipype CFF converter)
  Description
out_file: (a pathlike object or string representing a file, nipype default value: connectome.cff)
  Output connectome file

Outputs:

connectome_file: (a pathlike object or string representing an existing file)
  Output connectome file

57.3.2 MergeCNetworks

Link to code
Merges networks from multiple CFF files into one new CFF file.

Example

```python
>>> import nipype.interfaces.cmtk as cmtk
>>> mrg = cmtk.MergeCNetworks()
>>> mrg.inputs.in_files = ['subj1.cff','subj2.cff']
>>> mrg.run() # doctest: +SKIP
```

Inputs:
57.4 interfaces.cmtk.nbs

57.4.1 NetworkBasedStatistic

Link to code
Calculates and outputs the average network given a set of input NetworkX gpickle files
For documentation of Network-based statistic parameters:

Example

```python
>>> import nipype.interfaces.cmtk as cmtk
>>> nbs = cmtk.NetworkBasedStatistic()
>>> nbs.inputs.in_group1 = ['subj1.pck', 'subj2.pck']  # doctest: +SKIP
>>> nbs.inputs.in_group2 = ['pat1.pck', 'pat2.pck']  # doctest: +SKIP
>>> nbs.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
- **in_group1**: (a list of items which are a pathlike object or string representing an existing file)
  - Networks for the first group of subjects
- **in_group2**: (a list of items which are a pathlike object or string representing an existing file)
  - Networks for the second group of subjects

[Optional]
- **node_position_network**: (a pathlike object or string representing a file)
  - An optional network used to position the nodes for the output networks
- **number_of_permutations**: (an integer (int or long), nipype default value: 1000)
  - Number of permutations to perform
- **threshold**: (a float, nipype default value: 3)
  - T-statistic threshold
- **t_tail**: ('left' or 'right' or 'both', nipype default value: left)
  - Can be one of "left", "right", or "both"
57.4.2 ntwks_to_matrices()

Link to code

57.5 interfaces.cmtk.nx

57.5.1 AverageNetworks

Link to code
Calculates and outputs the average network given a set of input NetworkX gpickle files
This interface will only keep an edge in the averaged network if that edge is present in at least half of the input networks.

Example

```python
>>> import nipype.interfaces.cmtk as cmtk
>>> avg = cmtk.AverageNetworks()
>>> avg.inputs.in_files = ['subj1.pck', 'subj2.pck']
>>> avg.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
in_files: (a list of items which are a pathlike object or string representing an existing file)
    Networks for a group of subjects

[Optional]
resolution_network_file: (a pathlike object or string representing an existing file)
    Parcellation files from Connectome Mapping Toolkit. This is not

Outputs:

nbs_network: (a pathlike object or string representing an existing file)
    Output network with edges identified by the NBS
nbs_pval_network: (a pathlike object or string representing an existing file)
    Output network with p-values to weight the edges identified by the NBS
network_files: (a list of items which are a pathlike object or string representing an existing file)
    Output network with edges identified by the NBS
necessary, but if included, the interface will output the statistical maps as networkx graphs.
group_id: (a unicode string, nipype default value: group1)
   ID for group
out_gpickled_groupavg: (a pathlike object or string representing a file)
   Average network saved as a NetworkX .pck
out_gexf_groupavg: (a pathlike object or string representing a file)
   Average network saved as a .gexf file

Outputs:
gpickled_groupavg: (a pathlike object or string representing a file)
   Average network saved as a NetworkX .pck
gexf_groupavg: (a pathlike object or string representing a file)
   Average network saved as a .gexf file
matlab_groupavgs: (a list of items which are a pathlike object or string representing a file)

57.5.2 NetworkXMetrics

Link to code
Calculates and outputs NetworkX-based measures for an input network

Example

```python
>>> import nipype.interfaces.cmtk as cmtk
>>> nxmetrics = cmtk.NetworkXMetrics()
>>> nxmetrics.inputs.in_file = 'subj1.pck'
>>> nxmetrics.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
   Input network

[Optional]
out_k_core: (a pathlike object or string representing a file, nipype default value: k_core)
   Computed k-core network stored as a NetworkX pickle.
out_k_shell: (a pathlike object or string representing a file, nipype default value: k_shell)
   Computed k-shell network stored as a NetworkX pickle.
out_k_crust: (a pathlike object or string representing a file, nipype default value: k_crust)
   Computed k-crust network stored as a NetworkX pickle.
compute_clique_related_measures: (a boolean, nipype default value: True)
   Some network metrics can be calculated while considering only a binarized version of the graph
compute_clique_related_measures: (a boolean, nipype default value: False)
   Computing clique-related measures (e.g. node clique number) can be very time consuming
out_global_metrics_matlab: (a pathlike object or string representing a file)
Output node metrics in MATLAB .mat format
out_node_metrics_matlab: (a pathlike object or string representing a file)

Output edge metrics in MATLAB .mat format
out_edge_metrics_matlab: (a pathlike object or string representing a file)

Output edge metrics in MATLAB .mat format
out_pickled_extra_measures: (a pathlike object or string representing a file, nipype default value: extra_measures)

Network measures for group 1 that return dictionaries stored as a Pickle.

Outputs:

gpicked_network_files: (a list of items which are a pathlike object or string representing a file)
matlab_matrix_files: (a list of items which are a pathlike object or string representing a file)
global_measures_matlab: (a pathlike object or string representing a file)

Output global metrics in MATLAB .mat format
node_measures_matlab: (a pathlike object or string representing a file)
edge_measures_matlab: (a pathlike object or string representing a file)

Output edge metrics in MATLAB .mat format
node_measure_networks: (a list of items which are a pathlike object or string representing a file)
edge_measure_networks: (a list of items which are a pathlike object or string representing a file)
k_networks: (a list of items which are a pathlike object or string representing a file)
k_core: (a pathlike object or string representing a file)
    Computed k-core network stored as a NetworkX pickle.
k_shell: (a pathlike object or string representing a file)
    Computed k-shell network stored as a NetworkX pickle.
k_crust: (a pathlike object or string representing a file)
    Computed k-crust network stored as a NetworkX pickle.
pickled_extra_measures: (a pathlike object or string representing a file)
    Network measures for the group that return dictionaries, stored as a Pickle.
matlab_dict_measures: (a list of items which are a pathlike object or string representing a file)

57.5.3 add_dicts_by_key()

Link to code
Combines two dictionaries and adds the values for those keys that are shared

57.5.4 add_edge_data()

Link to code
57.5.5 add_node_data()
Link to code

57.5.6 average_networks()
Link to code
Sums the edges of input networks and divides by the number of networks Writes the average network as .pck and .gexf and returns the name of the written networks

57.5.7 compute_dict_measures()
Link to code
Returns a dictionary

57.5.8 compute_edge_measures()
Link to code
These return edge-based measures

57.5.9 compute_network_measures()
Link to code

57.5.10 compute_node_measures()
Link to code
These return node-based measures

57.5.11 compute_singlevalued_measures()
Link to code
Returns a single value per network

57.5.12 fix_keys_for_gexf()
Link to code
GEXF Networks can be read in Gephi, however, the keys for the node and edge IDs must be converted to strings

57.5.13 read_unknown_ntwk()
Link to code

57.5.14 remove_all_edges()
Link to code

57.6 interfaces.cmtk.parcellation

57.6.1 Parcellate
Link to code
Subdivides segmented ROI file into smaller subregions
This interface implements the same procedure as in the ConnectomeMapper’s parcellation stage (cmp/stages/parcellation/maskcreation.py) for a single parcellation scheme (e.g. ‘scale500’).
Example

```python
>>> import nipype.interfaces.cmtk as cmtk
>>> parcellate = cmtk.Parcellate()
>>> parcellate.inputs.freesurfer_dir = '.'
>>> parcellate.inputs.subjects_dir = '.'
>>> parcellate.inputs.subject_id = 'subj1'
>>> parcellate.inputs.dilation = True
>>> parcellate.inputs.parcellation_name = 'scale500'
>>> parcellate.run()  # doctest: +SKIP
```

Inputs:

- **subject_id**: (a string) Subject ID
- **parcellation_name**: ('scale33' or 'scale60' or 'scale125' or 'scale250' or 'scale500', nipype default value: scale500)
- **freesurfer_dir**: (a pathlike object or string representing an existing directory) Freesurfer main directory
- **subjects_dir**: (a pathlike object or string representing an existing directory) Freesurfer subjects directory
- **out_roi_file**: (a pathlike object or string representing a file) Region of Interest file for connectivity mapping
- **dilation**: (a boolean, nipype default value: False) Dilate cortical parcels? Useful for fMRI connectivity

Outputs:

- **roi_file**: (a pathlike object or string representing an existing file) Region of Interest file for connectivity mapping
- **roiv_file**: (a pathlike object or string representing a file) Region of Interest file for fMRI connectivity mapping
- **white_matter_mask_file**: (a pathlike object or string representing an existing file) White matter mask file
- **cc_unknown_file**: (a pathlike object or string representing an existing file) Image file with regions labelled as unknown cortical structures
- **ribbon_file**: (a pathlike object or string representing an existing file) Image file detailing the cortical ribbon
- **aseg_file**: (a pathlike object or string representing an existing file) Automated segmentation file converted from Freesurfer "subjects" directory
- **roi_file_in_structural_space**: (a pathlike object or string representing an existing file) ROI image resliced to the dimensions of the original structural image
- **dilated_roi_file_in_structural_space**: (a pathlike object or string representing a file) Dilated ROI image resliced to the dimensions of the original structural image
57.6.2 create_annot_label()
Link to code

57.6.3 create_roi()
Link to code
Creates the ROI_%s.nii.gz files using the given parcellation information from networks. Iteratively create volume.

57.6.4 create_wm_mask()
Link to code

57.6.5 crop_and_move_datasets()
Link to code

57.6.6 extract()
Link to code
Extract voxel neighbourhood Parameters ~~~~~~~~~~~ Z: the original data shape: tuple containing neighbourhood dimensions position: tuple containing central point indexes fill: value for the padding of Z Returns ~~~~~~~ R: the neighbourhood of the specified point in Z
58.1 interfaces.diffusion_toolkit.dti

58.1.1 DTIRecon

Link to code

Wraps the executable command dti_recon.

Use dti_recon to generate tensors and other maps

Inputs:

[Mandatory]
DVI: (a pathlike object or string representing an existing file)
   Input diffusion volume
   argument: `-%s`, position: 1
bvecs: (a pathlike object or string representing an existing file)
   b vectors file
   argument: `--gm %s`
bvals: (a pathlike object or string representing an existing file)
   b values file

[Optional]
out_prefix: (a unicode string, nipype default value: dti)
   Output file prefix
   argument: `-%s`, position: 2
output_type: (`nii' or 'analyze' or 'nil' or 'nii.gz', nipype default
   value: nii)
   output file type
   argument: `--ot %s`
n_averages: (an integer (int or long))
   Number of averages
   argument: `--nex %s`
image_orientation_vectors: (a list of from 6 to 6 items which are a
   float)
   specify image orientation vectors. if just one argument given,
   will treat it as filename and read the orientation vectors from
   the file. if 6 arguments are given, will treat them as 6 float
   numbers and construct the 1st and 2nd vector and calculate the 3rd

(continues on next page)
one automatically.
this information will be used to determine image orientation,
as well as to adjust gradient vectors with oblique angle when
argument: `--iop %f`

**oblique_correction** (a boolean)
when oblique angle(s) applied, some SIEMENS dti protocols do not
adjust gradient accordingly, thus it requires adjustment for
correct diffusion tensor calculation
argument: `--oc`

**b0_threshold** (a float)
program will use b0 image with the given threshold to mask out high
background of fa/adc maps. by default it will calculate threshold
automatically. but if it failed, you need to set it manually.
argument: `--b0_th`

**args** (a unicode string)
Additional parameters to the command
argument: `--%s`

**environ** (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

**Outputs:**

ADC: (a pathlike object or string representing an existing file)
B0: (a pathlike object or string representing an existing file)
L1: (a pathlike object or string representing an existing file)
L2: (a pathlike object or string representing an existing file)
L3: (a pathlike object or string representing an existing file)
ex: (a pathlike object or string representing an existing file)
FA: (a pathlike object or string representing an existing file)
FA_color: (a pathlike object or string representing an existing file)
tensor: (a pathlike object or string representing an existing file)
V1: (a pathlike object or string representing an existing file)
V2: (a pathlike object or string representing an existing file)
V3: (a pathlike object or string representing an existing file)

### 58.1.2 DTITracker

**Link to code**
Wraps the executable command `dti_tracker`.

**Inputs:**

[Mandatory]
**mask1_file** (a pathlike object or string representing a file)
first mask image
argument: `--m %s`, position: 2

[Optional]
**tensor_file** (a pathlike object or string representing an existing
file)
reconstructed tensor file
**input_type**: ('nii' or 'analyze' or 'ni1' or 'nii.gz')
input and output file type. accepted values are:
analyze -> analyze format 7.5

(continues on next page)
ni1 -> nifti format saved in separate .hdr and .img file
nii -> nifti format with one .nii file
nii.gz -> nifti format with compression
default type is 'nii'
argument: `--it %s`
tracking_method: ('fact' or 'rk2' or 't1' or 'sl')
fact -> use FACT method for tracking. this is the default method.
rk2 -> use 2nd order runge-kutta method for tracking.
t1 -> use tensorline method for tracking.
s1 -> use interpolated streamline method with fixed step-length
argument: `--%s`
step_length: (a float)
set step length, in the unit of minimum voxel size.
default value is 0.5 for interpolated streamline method
and 0.1 for other methods
argument: `--l %f`
age_angle_threshold: (a float)
set angle threshold. default value is 35 degree
argument: `--at %f`
age_angle_threshold_weight: (a float)
set angle threshold weighting factor. weighting will be be applied
on top of the angle_threshold
argument: `--atw %f`
random_seed: (an integer (int or long))
use random location in a voxel instead of the center of the voxel to
seed. can also define number of seed per voxel. default is 1
argument: `--rseed %d`
invert_x: (a boolean)
invert x component of the vector
argument: `--ix`
invert_y: (a boolean)
invert y component of the vector
argument: `--iy`
invert_z: (a boolean)
invert z component of the vector
argument: `--iz`
swap_xy: (a boolean)
swap x & y vectors while tracking
argument: `--sxy`
swap_yz: (a boolean)
swap y & z vectors while tracking
argument: `--syz`
swap_zx: (a boolean)
swap x & z vectors while tracking
argument: `--szx`
mask1_threshold: (a float)
threshold value for the first mask image, if not given, the program
will try automatically find the threshold
mask2_file: (a pathlike object or string representing a file)
second mask image
argument: `--m2 %s`, position: 4
mask2_threshold: (a float)
threshold value for the second mask image, if not given, the program
will try automatically find the threshold
input_data_prefix: (a unicode string, nipype default value: dti)
for internal naming use only
argument: `%s`, position: 0

(continues on next page)
output_file: (a pathlike object or string representing an existing file, nipype default value: tracks.trk)
    argument: `-%s`, position: 1
output_mask: (a pathlike object or string representing a file)
    output a binary mask file in analyze format
    argument: `-%om %s`
primary_vector: ('v2' or 'v3')
    which vector to use for fibre tracking: v2 or v3. If not set use v1
    argument: `-%s`
args: (a unicode string)
    Additional parameters to the command
    argument: `-%s`
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables

Outputs:

track_file: (a pathlike object or string representing an existing file)
mask_file: (a pathlike object or string representing an existing file)

58.2 interfaces.diffusion_toolkit.odf

58.2.1 HARDIMat

Link to code
Wraps the executable command hardi_mat.
Use hardi_mat to calculate a reconstruction matrix from a gradient table

Inputs:

[Mandatory]
bvecs: (a pathlike object or string representing an existing file)
    b vectors file
    argument: `-%s`, position: 1
bvals: (a pathlike object or string representing an existing file)
    b values file

[Optional]
out_file: (a pathlike object or string representing a file, nipype
    default value: recon_mat.dat)
    output matrix file
    argument: `-%s`, position: 2
order: (an integer (int or long))
    maximum order of spherical harmonics. must be even number. default
    is 4
    argument: `-%order %s`
odf_file: (a pathlike object or string representing an existing file)
    filename that contains the reconstruction points on a HEMI-sphere.
    use the pre-set 181 points by default
    argument: `-%odf %s`
reference_file: (a pathlike object or string representing an existing file)
provide a dicom or nifti image as the reference for the program to figure out the image orientation information. If no such info was found in the given image header, the next 5 options -info, etc., will be used if provided. If image orientation info can be found in the given reference, all other 5 image orientation options will be IGNORED.

Argument: `-%ref %s` 

Image info: (a pathlike object or string representing an existing file) 
Specify image information file. The image info file is generated from original dicom image by diff_unpack program and contains image orientation and other information needed for reconstruction and tracking. By default will look into the image folder for .info file.

Argument: `-%info %s` 

Image orientation vectors: (a list of from 6 to 6 items which are a float) 
Specify image orientation vectors. If just one argument given, will treat it as filename and read the orientation vectors from the file. If 6 arguments are given, will treat them as 6 float numbers and construct the 1st and 2nd vector and calculate the 3rd one automatically. This information will be used to determine image orientation, as well as to adjust gradient vectors with oblique angle when.

Argument: `-%iop %f` 

Oblique correction: (a boolean) 
When oblique angle(s) applied, some SIEMENS dti protocols do not adjust gradient accordingly, thus it requires adjustment for correct diffusion tensor calculation.

Argument: `-%oc` 

Args: (a unicode string) 
Additional parameters to the command.

Argument: `-%s` 

Environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) 
Environment variables

Outputs:

Out file: (a pathlike object or string representing an existing file) 
Output matrix file

58.2.2 ODFRecon

Link to code

Wraps the executable command odf_recon.

Use odf_recon to generate tensors and other maps.

Inputs:

[Mandatory]

DWI: (a pathlike object or string representing an existing file) 
Input raw data

Argument: `-%s`, position: 1

n_directions: (an integer (int or long)) 
Number of directions
argument: ``%s``, position: 2

n_output_directions: (an integer (int or long))
Number of output directions
argument: ``%s``, position: 3

matrix: (a pathlike object or string representing an existing file)
use given file as reconstruction matrix.
argument: ``-mat %s``

n_b0: (an integer (int or long))
number of b0 scans. by default the program gets this information
from the number of directions and number of volumes in
the raw data. useful when dealing with incomplete raw
data set or only using part of raw data set to reconstruct
argument: ``-b0 %s``

[Optional]

out_prefix: (a unicode string, nipype default value: odf)
Output file prefix
argument: ``%s``, position: 4

output_type: ('nii' or 'analyze' or 'nil' or 'nii.gz', nipype default
value: nii)
output file type
argument: ``-ot %s``

sharpness: (a float)
smooth or sharpen the raw data. factor > 0 is smoothing.
factor < 0 is sharpening. default value is 0
NOTE: this option applies to DSI study only
argument: ``-s %f``

filter: (a boolean)
apply a filter (e.g. high pass) to the raw image
argument: ``-f``

subtract_background: (a boolean)
subtract the background value before reconstruction
argument: ``-bg``

dsi: (a boolean)
indicates that the data is dsi
argument: ``-dsi``

output_entropy: (a boolean)
output entropy map
argument: ``-oe``

image_orientation_vectors: (a list of from 6 to 6 items which are a
float)
specify image orientation vectors. if just one argument given,
will treat it as filename and read the orientation vectors from
the file. if 6 arguments are given, will treat them as 6 float
numbers and construct the 1st and 2nd vector and calculate the 3rd
one automatically.
this information will be used to determine image orientation,
as well as to adjust gradient vectors with oblique angle when
argument: ``-iop %f``

oblique_correction: (a boolean)
when oblique angle(s) applied, some SIEMENS dti protocols do not
adjust gradient accordingly, thus it requires adjustment for
correct
diffusion tensor calculation
argument: ``-oc``

args: (a unicode string)
Additional parameters to the command
(continues on next page)


58.2. interfaces.diffusion_toolkit.odf

58.2.3 ODFTracker

Link to code

Wraps the executable command `odf_tracker`. Use `odf_tracker` to generate track file

Inputs:

[Mandatory]
max: (a pathlike object or string representing an existing file)
ODF: (a pathlike object or string representing an existing file)
mask1_file: (a pathlike object or string representing a file)
  first mask image
  argument: `''-m %s''`, position: 2

[Optional]
input_data_prefix: (a unicode string, nipype default value: odf)
  recon data prefix
  argument: `''%s''`, position: 0
out_file: (a pathlike object or string representing a file, nipype
default value: tracks.trk)
  output track file
  argument: `''%s''`, position: 1
input_output_type: ("nii" or 'analyze' or 'ni1' or 'nii.gz', nipype
default value: nii)
  input and output file type
  argument: `''-it %s''`
runge_kutta2: (a boolean)
  use 2nd order runge-kutta method for tracking.
  default tracking method is non-interpolate streamline
  argument: `''-rk2''`
step_length: (a float)
  set step length, in the unit of minimum voxel size.
  default value is 0.1.
  argument: `''-l %f''`
angle_threshold: (a float)
  set angle threshold. default value is 35 degree for
  default tracking method and 25 for rk2
  argument: `''-at %f''`
random_seed: (an integer (int or long))
  use random location in a voxel instead of the center of the voxel
  to seed. can also define number of seed per voxel. default is 1
  argument: `''-rseed %s''`
invert_x: (a boolean)
invert x component of the vector
argument: `'-ix'`

invert_y: (a boolean)
invert y component of the vector
argument: `'-iy'`

invert_z: (a boolean)
invert z component of the vector
argument: `'-iz'`

swap_xy: (a boolean)
swap x and y vectors while tracking
argument: `'-sxy'`

swap_yz: (a boolean)
swap y and z vectors while tracking
argument: `'-syz'`

swap_zx: (a boolean)
swap x and z vectors while tracking
argument: `'-szx'`

disc: (a boolean)
use disc tracking
argument: `'-disc'`

mask1_threshold: (a float)
threshold value for the first mask image, if not given, the program will try automatically find the threshold

mask2_file: (a pathlike object or string representing a file)
second mask image
argument: `'-m2 %s'`, position: 4

mask2_threshold: (a float)
threshold value for the second mask image, if not given, the program will try automatically find the threshold

limit: (an integer (int or long))
in some special case, such as heart data, some track may go into infinite circle and take long time to stop. this option allows setting a limit for the longest tracking steps (voxels)
argument: `'-limit %d'`

dsi: (a boolean)
specify the input odf data is dsi. because dsi recon uses fixed pre-calculated matrix, some special orientation patch needs to be applied to keep dti/dsi/q-ball consistent.
argument: `'-dsi'`

image_orientation_vectors: (a list of from 6 to 6 items which are a float)
specify image orientation vectors. if just one argument given, will treat it as filename and read the orientation vectors from the file. if 6 arguments are given, will treat them as 6 float numbers and construct the 1st and 2nd vector and calculate the 3rd one automatically.
this information will be used to determine image orientation, as well as to adjust gradient vectors with oblique angle when argument: `'-iop %f'`

slice_order: (an integer (int or long))
set the slice order. 1 means normal, -1 means reversed. default value is 1
argument: `'-sorder %d'`

voxel_order: ("RAS" or "RPS" or "RAI" or "RPI" or "LAI" or "LAS" or "LPS" or "LPI")
specify the voxel order in RL/AP/IS (human brain) reference. must be
3 letters with no space in between. for example, RAS means the voxel row is from L->R, the column is from P->A and the slice order is from I->S. by default voxel order is determined by the image orientation (but NOT guaranteed to be correct because of various standards). for example, siemens axial image is LPS, coronal image is LIP and sagittal image is PIL. this information also is NOT needed for tracking but will be saved in the track file and is essential for track display to map onto the right coordinates argument: ``-vorder %s``
args: (a unicode string) Additional parameters to the command argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

### Outputs:

| track_file: (a pathlike object or string representing an existing file) |
| ... output track file |

#### 58.3 interfaces.diffusion_toolkit.postproc

#### 58.3.1 SplineFilter

Link to code
Wraps the executable command spline_filter. Smoothes TrackVis track files with a B-Spline filter. Helps remove redundant track points and segments (thus reducing the size of the track file) and also make tracks nicely smoothed. It will NOT change the quality of the tracks or lose any original information.

**Example**

```python
>>> import nipype.interfaces.diffusion_toolkit as dtk
>>> filt = dtk.SplineFilter()
>>> filt.inputs.track_file = 'tracks.trk'
>>> filt.inputs.step_length = 0.5
>>> filt.run() # doctest: +SKIP
```

**Inputs:**

[Mandatory]

| track_file: (a pathlike object or string representing an existing file) |
| ... file containing tracks to be filtered |
| ... argument: ``%s`` |
| step_length: (a float) |
| ... in the unit of minimum voxel size |
| ... argument: ``%f`` |

[Optional]

(continues on next page)
output_file: (a pathlike object or string representing a file, nipype
default value: spline_tracks.trk)
target file for smoothed tracks
argument: ``%s``
position: 2
args: (a unicode string)
Additional parameters to the command
argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: { })
Environment variables

Outputs:

smoothed_track_file: (a pathlike object or string representing an
existing file)

58.3.2 TrackMerge

Link to code
Wraps the executable command track_merge.
Merges several TrackVis track files into a single track file.
An id type property tag is added to each track in the newly merged file, with each unique id representing where
the track was originally from. When the merged file is loaded in TrackVis, a property filter will show up in
Track Property panel. Users can adjust that to distinguish and sub-group tracks by its id (origin).

Example

```python
>>> import nipype.interfaces.diffusion_toolkit as dtk
>>> mrg = dtk.TrackMerge()
>>> mrg.inputs.track_files = ['track1.trk','track2.trk']
>>> mrg.run()
# doctest: +SKIP
```

Inputs:

[Mandatory]
track_files: (a list of items which are a pathlike object or string
representing an existing file)
file containing tracks to be filtered
argument: ``%s...``
position: 0

[Optional]
output_file: (a pathlike object or string representing a file, nipype
default value: merged_tracks.trk)
target file for merged tracks
argument: ``%s``
position: -1
args: (a unicode string)
Additional parameters to the command
argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: { })
Environment variables

Outputs:
track_file: (a pathlike object or string representing an existing file)
59.1 interfaces.dipy.anisotropic_power

59.1.1 APMQball

Link to code
Calculates the anisotropic power map

Example

```python
>>> import nipype.interfaces.dipy as dipy
>>> apm = dipy.APMQball()
>>> apm.inputs.in_file = 'diffusion.nii'
>>> apm.inputs.in_bvec = 'bvecs'
>>> apm.inputs.in_bval = 'bvals'
>>> apm.run() # doctest: +SKIP
```

Inputs:

- `in_file`: (a pathlike object or string representing an existing file)  
  input diffusion data
- `in_bval`: (a pathlike object or string representing an existing file)  
  input b-values table
- `in_bvec`: (a pathlike object or string representing an existing file)  
  input b-vectors table
- `mask_file`: (a pathlike object or string representing an existing file)  
  An optional brain mask
- `b0_thres`: (an integer (int or long), nipype default value: 700)  
  b0 threshold
- `out_prefix`: (a unicode string)  
  output prefix for file names

Outputs:
59.2 interfaces.dipy.base

59.2.1 DipyBaseInterface

Link to code
A base interface for py:mod:dipy computations

Inputs:

None

Outputs:

None

59.2.2 DipyDiffusionInterface

Link to code
A base interface for py:mod:dipy computations

Inputs:

[Mandatory]
- in_file: (a pathlike object or string representing an existing file)
  input diffusion data
- in_bval: (a pathlike object or string representing an existing file)
  input b-values table
- in_bvec: (a pathlike object or string representing an existing file)
  input b-vectors table

[Optional]
- b0_thres: (an integer (int or long), nipype default value: 700)
  b0 threshold
- out_prefix: (a unicode string)
  output prefix for file names

Outputs:

None

59.2.3 convert_to_traits_type()

Link to code
Convert DIPY type to Traits type.

59.2.4 create_interface_specs()

Link to code
Create IN/Out interface specifications dynamically.

Parameters

class_name: str  The future class name(e.g. (MyClassInSpec))
params: list of tuple  dipy argument list
BaseClass: TraitedSpec object  parent class
59.2.5 dipy_to_nipype_interface()

Link to code
Construct a class in order to respect nipype interface specifications. This convenient class factory convert a DIPY Workflow to a nipype interface.

Parameters

cls_name: string new class name
dipy_flow: Workflow class type. It should be any children class of dipy.workflows.workflow.Workflow

BaseClass: object nipype instance object

Returns

newclass: object new nipype interface specification class

59.2.6 get_dipy_workflows()

Link to code
Search for DIPY workflow class.

Parameters

module [object] module object

Returns

l_wkflw [list of tuple] This a list of tuple containing 2 elements: Workflow name, Workflow class obj

Examples

```python
>>> from dipy.workflows import align  # doctest: +SKIP
>>> get_dipy_workflows(align)  # doctest: +SKIP
```

59.3 interfaces.dipy.preprocess

59.3.1 Denoise

Link to code

Example

```python
>>> import nipype.interfaces.dipy as dipy
>>> denoise = dipy.Denoise()
>>> denoise.inputs.in_file = 'diffusion.nii'
>>> denoise.run()  # doctest: +SKIP
```

Inputs:
### [Mandatory]

**in_file**: (a pathlike object or string representing an existing file)

The input 4D diffusion-weighted image file

**noise_model**: (‘rician’ or ‘gaussian’, nipype default value: rician)

noise distribution model

### [Optional]

**in_mask**: (a pathlike object or string representing an existing file)

brain mask

**signal_mask**: (a pathlike object or string representing an existing file)

mask in which the mean signal will be computed

**noise_mask**: (a pathlike object or string representing an existing file)

mask in which the standard deviation of noise will be computed

**patch_radius**: (an integer (int or long), nipype default value: 1)

patch radius

**block_radius**: (an integer (int or long), nipype default value: 5)

block radius

**snr**: (a float)

manually set an SNR

### Outputs:

**out_file**: (a pathlike object or string representing an existing file)

---

### 59.3.2 Resample

Link to code

An interface to reslicing diffusion datasets. See [http://nipy.org/dipy/examples_built/reslice_datasets.html#example-reslice-datasets](http://nipy.org/dipy/examples_built/reslice_datasets.html#example-reslice-datasets).

#### Example

```python
>>> import nipype.interfaces.dipy as dipy
dipy
>>> reslice = dipy.Resample()
>>> reslice.inputs.in_file = 'diffusion.nii'
>>> reslice.run() # doctest: +SKIP
```

### Inputs:

#### [Mandatory]

**in_file**: (a pathlike object or string representing an existing file)

The input 4D diffusion-weighted image file

**interp**: (an integer (int or long), nipype default value: 1)

order of the interpolator (0 = nearest, 1 = linear, etc.

### [Optional]

**vox_size**: (a tuple of the form: (a float, a float, a float))

specify the new voxel zooms. If no vox_size is set, then isotropic regridding will be performed, with spacing equal to the smallest current zoom.

### Outputs:

**out_file**: (a pathlike object or string representing an existing file)
59.3.3 nlmeans_proxy()

Link to code
Uses non-local means to denoise 4D datasets

59.3.4 resample_proxy()

Link to code
Performs regridding of an image to set isotropic voxel sizes using dipy.

59.4 interfaces.dipy.reconstruction

59.4.1 CSD

Link to code
Uses CSD [Tournier2007] to generate the fODF of DWIs. The interface uses dipy, as explained in dipy’s CSD example.

Example

```python
>>> from nipype.interfaces import dipy as ndp
>>> csd = ndp.CSD()
>>> csd.inputs.in_file = '4d_dwi.nii'
>>> csd.inputs.in_bval = 'bvals'
>>> csd.inputs.in_bvec = 'bvecs'
>>> res = csd.run() # doctest: +SKIP
```

Inputs:

<table>
<thead>
<tr>
<th>Mandatory</th>
</tr>
</thead>
</table>
in_file: (a pathlike object or string representing an existing file)
in_bval: (a pathlike object or string representing an existing file)
in_bvec: (a pathlike object or string representing an existing file)

<table>
<thead>
<tr>
<th>Optional</th>
</tr>
</thead>
</table>
in_mask: (a pathlike object or string representing an existing file)
response: (a pathlike object or string representing an existing file)
sh_order: (an integer (int or long), nipype default value: 8)
save_fods: (a boolean, nipype default value: True)
out_fods: (a pathlike object or string representing a file)
b0_thres: (an integer (int or long), nipype default value: 700)
out_prefix: (a unicode string)

Outputs:

|---|
model: (a pathlike object or string representing a file)
Python pickled object of the CSD model fitted.
out_fods: (a pathlike object or string representing a file)
    fODFs output file name

59.4.2 EstimateResponseSH

Link to code
Uses dipy to compute the single fiber response to be used in spherical deconvolution methods, in a similar way to MRTrix’s command estimate_response.

Example

```python
>>> from nipype.interfaces import dipy as ndp
>>> dti = ndp.EstimateResponseSH()
>>> dti.inputs.in_file = '4d_dwi.nii'
>>> dti.inputs.in_bval = 'bvals'
>>> dti.inputs.in_bvec = 'bvecs'
>>> dti.inputs.in_evals = 'dwi_evals.nii'
>>> res = dti.run() # doctest: +SKIP
```

Inputs:

[Mandatory]

- in_evals: (a pathlike object or string representing an existing file)
  - input eigenvalues file
- in_file: (a pathlike object or string representing an existing file)
  - input diffusion data
- in_bval: (a pathlike object or string representing an existing file)
  - input b-values table
- in_bvec: (a pathlike object or string representing an existing file)
  - input b-vectors table

[Optional]

- in_mask: (a pathlike object or string representing an existing file)
  - input mask in which we find single fibers
- fa_thresh: (a float, nipype default value: 0.7)
  - FA threshold
- roi_radius: (an integer (int or long), nipype default value: 10)
  - ROI radius to be used in auto_response
- auto: (a boolean)
  - use the auto_response estimator from dipy
- recursive: (a boolean)
  - use the recursive response estimator from dipy
  - mutually_exclusive: recursive
- response: (a pathlike object or string representing a file, nipype default value: response.txt)
  - the output response file
- out_mask: (a pathlike object or string representing a file, nipype default value: wm_mask.nii.gz)
  - computed wm mask
- b0_thres: (an integer (int or long), nipype default value: 700)
  - b0 threshold
- out_prefix: (a unicode string)
  - output prefix for file names

Outputs:
response: (a pathlike object or string representing an existing file)
    the response file
out_mask: (a pathlike object or string representing an existing file)
    output wm mask

59.4.3 RESTORE

Link to code

Uses RESTORE [Chang2005] to perform DTI fitting with outlier detection. The interface uses dipy, as explained in dipy's documentation.

Example

```python
>>> from nipype.interfaces import dipy as ndp
>>> dti = ndp.RESTORE()
>>> dti.inputs.in_file = '4d_dwi.nii'
>>> dti.inputs.in_bval = 'bvals'
>>> dti.inputs.in_bvec = 'bvecs'
>>> res = dti.run()  # doctest: +SKIP
```

Inputs:

|Mandatory|
in_file: (a pathlike object or string representing an existing file)
    input diffusion data
in_bval: (a pathlike object or string representing an existing file)
    input b-values table
in_bvec: (a pathlike object or string representing an existing file)
    input b-vectors table

|Optional|
in_mask: (a pathlike object or string representing an existing file)
    input mask in which compute tensors
noise_mask: (a pathlike object or string representing an existing file)
    input mask in which compute noise variance
b0_thres: (an integer (int or long), nipype default value: 700)
    b0 threshold
out_prefix: (a unicode string)
    output prefix for file names

Outputs:

|fa: (a pathlike object or string representing a file)
    output fractional anisotropy (FA) map computed from the fitted DTI
md: (a pathlike object or string representing a file)
    output mean diffusivity (MD) map computed from the fitted DTI
rd: (a pathlike object or string representing a file)
    output radial diffusivity (RD) map computed from the fitted DTI
mode: (a pathlike object or string representing a file)
    output mode (MO) map computed from the fitted DTI
trace: (a pathlike object or string representing a file)
    output the tensor trace map computed from the fitted DTI
evals: (a pathlike object or string representing a file)
    output the eigenvalues of the fitted DTI
evecs: (a pathlike object or string representing a file)
    output the eigenvectors of the fitted DTI
59.5 interfaces.dipy.simulate

59.5.1 SimulateMultiTensor

Link to code
Interface to MultiTensor model simulator in dipy http://nipy.org/dipy/examples_built/simulate_multi_tensor.html

Example

```python
>>> import nipype.interfaces.dipy as dipy
>>> sim = dipy.SimulateMultiTensor()
>>> sim.inputs.in_dirs = ['fdir00.nii', 'fdir01.nii']
>>> sim.inputs.in_frac = ['ffra00.nii', 'ffra01.nii']
>>> sim.inputs.in_vfms = ['tpm_00.nii.gz', 'tpm_01.nii.gz', ...
   'tpm_02.nii.gz']
>>> sim.inputs.baseline = 'b0.nii'
>>> sim.inputs.in_bvec = 'bvecs'
>>> sim.inputs.in_bval = 'bvals'
>>> sim.run() # doctest: +SKIP
```

Inputs:

[Mandatory]

- **in_dirs**: (a list of items which are a pathlike object or string representing an existing file)
  - list of fibers (principal directions)
- **in_frac**: (a list of items which are a pathlike object or string representing an existing file)
  - volume fraction of each fiber
- **in_vfms**: (a list of items which are a pathlike object or string representing an existing file)
  - volume fractions of isotropic compartments
- **baseline**: (a pathlike object or string representing an existing file)
  - baseline T2 signal

[Optional]

- **in_mask**: (a pathlike object or string representing an existing file)
  - mask to simulate data
- **diff_iso**: (a list of items which are a float, nipype default value: [0.003, 0.00096, 0.00068])
  - Diffusivity of isotropic compartments
- **diff_sf**: (a tuple of the form: (a float, a float, a float), nipype default value: (0.0017, 0.0002, 0.0002))
  - Single fiber tensor
- **n_proc**: (an integer (int or long), nipype default value: 0)
  - number of processes
- **gradients**: (a pathlike object or string representing an existing file)
  - gradients file
- **in_bvec**: (a pathlike object or string representing an existing file)
  - input bvecs file
- **in_bval**: (a pathlike object or string representing an existing file)
  - input bvals file
- **num_dirs**: (an integer (int or long), nipype default value: 32)
  - number of gradient directions (when table is automatically generated)
bvalues: (a list of items which are an integer (int or long), nipype default value: [1000, 3000]) list of b-values (when table is automatically generated)

out_file: (a pathlike object or string representing a file, nipype default value: sim_dwi.nii.gz) output file with fractions to be simulated

out_mask: (a pathlike object or string representing a file, nipype default value: sim_msk.nii.gz) file with the mask simulated

out_bvec: (a pathlike object or string representing a file, nipype default value: bvec.sim) simulated b vectors

out_bval: (a pathlike object or string representing a file, nipype default value: bval.sim) simulated b values

snr: (an integer (int or long), nipype default value: 0) signal-to-noise ratio (dB)

Outputs:

out_file: (a pathlike object or string representing an existing file) simulated DWIs

out_mask: (a pathlike object or string representing an existing file) mask file

out_bvec: (a pathlike object or string representing an existing file) simulated b vectors

out_bval: (a pathlike object or string representing an existing file) simulated b values

59.6 interfaces.dipy.tensors

59.6.1 DTI

Link to code
Calculates the diffusion tensor model parameters

Example

```python
>>> import nipype.interfaces.dipy as dipy
>>> dti = dipy.DTI()
>>> dti.inputs.in_file = 'diffusion.nii'
>>> dti.inputs.in_bvec = 'bvecs'
>>> dti.inputs.in_bval = 'bvals'
>>> dti.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]

in_file: (a pathlike object or string representing an existing file) input diffusion data

in_bval: (a pathlike object or string representing an existing file) input b-values table

in_bvec: (a pathlike object or string representing an existing file) input b-vectors table
[Optional]

**mask_file**: (a pathlike object or string representing an existing file)
   An optional white matter mask

**b0_thres**: (an integer (int or long), nipype default value: 700)
   b0 threshold

**out_prefix**: (a unicode string)
   output prefix for file names

**Outputs:**

**out_file**: (a pathlike object or string representing an existing file)
**fa_file**: (a pathlike object or string representing an existing file)
**md_file**: (a pathlike object or string representing an existing file)
**rd_file**: (a pathlike object or string representing an existing file)
**ad_file**: (a pathlike object or string representing an existing file)
**color_fa_file**: (a pathlike object or string representing an existing file)

### 59.6.2 TensorMode

**Link to code**

Creates a map of the mode of the diffusion tensors given a set of diffusion-weighted images, as well as their associated b-values and b-vectors. Fits the diffusion tensors and calculates tensor mode with Dipy.

**Example**

```python
def example():
    mode = dipy.TensorMode()
    mode.inputs.in_file = 'diffusion.nii'
    mode.inputs.in_bvec = 'bvecs'
    mode.inputs.in_bval = 'bvals'
    mode.run()  # doctest: +SKIP
```

**Inputs:**

[Mandatory]

**in_file**: (a pathlike object or string representing an existing file)
   input diffusion data

**in_bval**: (a pathlike object or string representing an existing file)
   input b-values table

**in_bvec**: (a pathlike object or string representing an existing file)
   input b-vectors table

[Optional]

**mask_file**: (a pathlike object or string representing an existing file)
   An optional white matter mask

**b0_thres**: (an integer (int or long), nipype default value: 700)
   b0 threshold

**out_prefix**: (a unicode string)
   output prefix for file names

**Outputs:**

**out_file**: (a pathlike object or string representing an existing file)
59.7 interfaces.dipy.tracks

59.7.1 StreamlineTractography

Link to code
Streamline tractography using EuDX [Garyfallidis12].

Example

```python
>>> from nipype.interfaces import dipy as ndp
>>> track = ndp.StreamlineTractography()
>>> track.inputs.in_file = '4d_dwi.nii'
>>> track.inputs.in_model = 'model.pklz'
>>> track.inputs.tracking_mask = 'dilated_wm_mask.nii'
>>> res = track.run() # doctest: +SKIP
```

Inputs:

**[Mandatory]**
- `in_file`: (a pathlike object or string representing an existing file)
  - input diffusion data
- `gfa_thresh`: (a float, nipype default value: 0.2)
  - GFA threshold to compute tracking mask
- `peak_threshold`: (a float, nipype default value: 0.5)
  - threshold to consider peaks from model
- `min_angle`: (a float, nipype default value: 25.0)
  - minimum separation angle
- `multiprocess`: (a boolean, nipype default value: True)
  - use multiprocessing
- `save_seeds`: (a boolean, nipype default value: False)
  - save seeding voxels coordinates
- `num_seeds`: (an integer (int or long), nipype default value: 10000)
  - desired number of tracks in tractography

**[Optional]**
- `in_model`: (a pathlike object or string representing an existing file)
  - input f/d-ODF model extracted from.
- `tracking_mask`: (a pathlike object or string representing an existing file)
  - input mask within which perform tracking
- `seed_mask`: (a pathlike object or string representing an existing file)
  - input mask within which perform seeding
- `in_peaks`: (a pathlike object or string representing an existing file)
  - peaks computed from the odf
- `seed_coord`: (a pathlike object or string representing an existing file)
  - file containing the list of seed voxel coordinates (N,3)
- `out_prefix`: (a unicode string)
  - output prefix for file names

Outputs:

- `tracks`: (a pathlike object or string representing a file)
  - TrackVis file containing extracted streamlines
- `gfa`: (a pathlike object or string representing a file)
  - The resulting GFA (generalized FA) computed using the peaks of the ODF

(continues on next page)
odf_peaks: (a pathlike object or string representing a file)
    peaks computed from the odf
out_seeds: (a pathlike object or string representing a file)
    file containing the (N,3) `voxel` coordinates used in seeding.

## 59.7.2 TrackDensityMap

**Link to code**

Creates a tract density image from a TrackVis track file using functions from dipy

**Example**

```python
>>> import nipype.interfaces.dipy as dipy
>>> trk2tdi = dipy.TrackDensityMap()
>>> trk2tdi.inputs.in_file = 'converted.trk'
>>> trk2tdi.run()  # doctest: +SKIP
```

**Inputs:**

- `in_file`: (a pathlike object or string representing an existing file)
  - The input TrackVis track file

- `reference`: (a pathlike object or string representing an existing file)
  - A reference file to define RAS coordinates space

- `points_space`: (`'rasmm'` or `'voxel'` or `None`, nipype default value: rasmm)
  - Coordinates of trk file

- `voxel_dims`: (a list of from 3 to 3 items which are a float)
  - The size of each voxel in mm.

- `data_dims`: (a list of from 3 to 3 items which are an integer (int or long))
  - The size of the image in voxels.

- `out_filename`: (a pathlike object or string representing a file, nipype default value: tdi.nii)
  - The output filename for the tracks in TrackVis (.trk) format

**Outputs:**

- `out_file`: (a pathlike object or string representing an existing file)
60.1 interfaces.dtitk.base

60.1.1 CommandLineDtitk

Link to code

Inputs:

```
[Optional]
args: (a unicode string)
    Additional parameters to the command
    argument: `\%s`
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})
Environment variables
```

Outputs:

None

60.2 interfaces.dtitk.registration

60.2.1 AffScalarVol

Link to code

Wraps the executable command affineScalarVolume. Applies affine transform to a scalar volume

Example

```python
>>> from nipype.interfaces import dtitk
>>> node = dtitk.AffScalarVol()
>>> node.inputs.in_file = 'im1.nii'
>>> node.inputs.transform = 'im_affine.aff'
```
```python
>>> node.cmdline
'affineScalarVolume -in im1.nii -interp 0 -out im1_affxfmd.nii -trans im_affine.aff'
>>> node.run()  # doctest: +SKIP
```

### Inputs:

[Mandatory]
- **in_file**: (a pathlike object or string representing an existing file)
  - moving scalar volume
  - argument: `'--in %s'`

[Optional]
- **out_file**: (a pathlike object or string representing a file)
  - output filename
  - argument: `'--out %s'`
- **transform**: (a pathlike object or string representing an existing file)
  - transform to apply: specify an input transformation file; parameters input will be ignored
  - argument: `'--trans %s'`
  - mutually_exclusive: target, translation, euler, deformation
- **interpolation**: ('trilinear' or 'NN', nipype default value: trilinear)
  - trilinear or nearest neighbor interpolation
  - argument: `'--interp %s'`
- **target**: (a pathlike object or string representing an existing file)
  - output volume specification read from the target volume if specified
  - argument: `'--target %s'`
  - mutually_exclusive: transform
- **translation**: (a tuple of the form: (a float, a float, a float))
  - translation \( (x, y, z) \) in mm
  - argument: `'--translation %g %g %g'`
  - mutually_exclusive: transform
- **euler**: (a tuple of the form: (a float, a float, a float))
  - \( (\theta, \phi, \psi) \) in degrees
  - argument: `'--euler %g %g %g'`
  - mutually_exclusive: transform
- **deformation**: (a tuple of the form: (a float, a float, a float, a float, a float, a float))
  - \( (xx, yy, zz, xy, yz, xz) \)
  - argument: `'--deformation %g %g %g %g %g %g'`
  - mutually_exclusive: transform
- **args**: (a unicode string)
  - Additional parameters to the command
  - argument: `'--%s'`
- **environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  - Environment variables

### Outputs:

- **out_file**: (a pathlike object or string representing an existing file)
  - moved volume

### 60.2.2 AffSymTensor3DVol

Link to code
Wraps the executable command `affineSymTensor3DVolume`. Applies affine transform to a tensor volume.

**Example**

```python
>>> from nipype.interfaces import dtitk
>>> node = dtitk.AffSymTensor3DVol()
>>> node.inputs.in_file = 'im1.nii'
>>> node.inputs.transform = 'im_affine.aff'
>>> node.cmdline
'affineSymTensor3DVolume -in im1.nii -interp LEI -out im1_affxfmd.nii -reorient PPD -trans im_affine.aff'
>>> node.run() # doctest: +SKIP
```

**Inputs:**

[Mandatory]
- `in_file`: (a pathlike object or string representing an existing file)
  - moving tensor volume
  - argument: `''-in %s''`

[Optional]
- `out_file`: (a pathlike object or string representing a file)
  - output filename
  - argument: `''-out %s''`
- `transform`: (a pathlike object or string representing an existing file)
  - transform to apply: specify an input transformation file; parameters input will be ignored
  - argument: `''-trans %s''`
  - mutually_exclusive: target, translation, euler, deformation
- `interpolation`: ('LEI' or 'EI', nipype default value: LEI)
  - Log Euclidean/Euclidean Interpolation
  - argument: `''-interp %s''`
- `reorient`: ('PPD' or 'NO' or 'FS', nipype default value: PPD)
  - Reorientation strategy: preservation of principal direction, no reorientation, or finite strain
  - argument: `''-reorient %s''`
- `target`: (a pathlike object or string representing an existing file)
  - output volume specification read from the target volume if specified
  - argument: `''-target %s''`
  - mutually_exclusive: transform
- `translation`: (a tuple of the form: (a float, a float, a float))
  - translation (x,y,z) in mm
  - argument: `''-translation %g %g %g''`
  - mutually_exclusive: transform
- `euler`: (a tuple of the form: (a float, a float, a float))
  - (theta, phi, psi) in degrees
  - argument: `''-euler %g %g %g''`
  - mutually_exclusive: transform
- `deformation`: (a tuple of the form: (a float, a float, a float, a float, a float, a float))
  - (xx,yy,zz,xy,yz,xz)
  - argument: `''-deformation %g %g %g %g %g %g''`
  - mutually_exclusive: transform
- `args`: (a unicode string)
  - Additional parameters to the command
  - argument: `''%s''`

(continues on next page)
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)

60.2.3 Affine

Link to code
Wraps the executable command dti_affine_reg.
Performs affine registration between two tensor volumes

Example

```python
>>> from nipype.interfaces import dtitk
>>> node = dtitk.Affine()
>>> node.inputs.fixed_file = 'im1.nii'
>>> node.inputs.moving_file = 'im2.nii'
>>> node.inputs.similarity_metric = 'EDS'
>>> node.inputs.sampling_xyz = (4,4,4)
>>> node.inputs.ftol = 0.01
>>> node.inputs.initialize_xfm = 'im_affine.aff'
>>> node.cmdline
'dti_affine_reg im1.nii im2.nii EDS 4 4 4 0.01 im_affine.aff'
>>> node.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
fixed_file: (a pathlike object or string representing an existing file)
fixed tensor volume argument: "%s", position: 0
moving_file: (a pathlike object or string representing an existing file)
moving tensor volume argument: "%s", position: 1
similarity_metric: ('EDS' or 'GDS' or 'DDS' or 'NMI', nipype default value: EDS)
similarity metric argument: "%s", position: 2
sampling_xyz: (a tuple of the form: (a value of class 'int', a value of class 'int'), nipype default value: (4, 4, 4))
dist between samp points (mm) (x,y,z) argument: "%g %g %g", position: 3
ftol: (a float, nipype default value: 0.01)
cost function tolerance argument: "%g", position: 4

[Optional]
initialize_xfm: (a pathlike object or string representing an existing file)
60.2.4 AffineTask

**Link to code**

Wraps the executable command `dti_affine_reg`.

**Inputs:**

- **fixed_file**: (a pathlike object or string representing an existing file)
  - fixed tensor volume
  - argument: `'\%s'`, position: 0
- **moving_file**: (a pathlike object or string representing an existing file)
  - moving tensor volume
  - argument: `'\%s'`, position: 1
- **similarity_metric**: (`'EDS'` or `'GDS'` or `'DDS'` or `'NMI'`, nipype default value: `EDS`)
  - similarity metric
  - argument: `'\%s'`, position: 2
- **sampling_xyz**: (a tuple of the form: (a value of class 'int', a value of class 'int'), nipype default value: `(4, 4, 4)`)
  - dist between samp points (mm) (x,y,z)
  - argument: `'\%g %g %g'`, position: 3
- **ftol**: (a float, nipype default value: 0.01)
  - cost function tolerance
  - argument: `'\%g'`, position: 4

- **initialize_xfm**: (a pathlike object or string representing an existing file)
  - Initialize w/DTITK-FORMATaffine
  - argument: `'\%s'`, position: 5
  - args: (a unicode string)
  - Additional parameters to the command
  - argument: `'\%s'`
  - environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: `{}`)
  - Environment variables

**Outputs:**

- **out_file**: (a pathlike object or string representing an existing file)
- **out_file_xfm**: (a pathlike object or string representing an existing file)
60.2.5 ComposeXfm

Link to code
Wraps the executable command dfRightComposeAffine.
Combines diffeomorphic and affine transforms

Example

```python
>>> from nipype.interfaces import dtitk
>>> node = dtitk.ComposeXfm()
>>> node.inputs.in_df = 'im_warp.df.nii'
>>> node.inputs.in_aff= 'im_affine.aff'
>>> node.cmdline
'dfRightComposeAffine -aff im_affine.aff -df im_warp.df.nii -out
im_warp_affdf.df.nii'
>>> node.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
in_df: (a pathlike object or string representing an existing file)
  diffeomorphic warp file
  argument: ‘-df %s’
in_aff: (a pathlike object or string representing an existing file)
  affine transform file
  argument: ‘-aff %s’

[Optional]
out_file: (a pathlike object or string representing a file)
  output path
  argument: ‘-out %s’
args: (a unicode string)
  Additional parameters to the command
  argument: ‘%s’
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
  Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)

60.2.6 ComposeXfmTask

Link to code
Wraps the executable command dfRightComposeAffine.
argument: ``-df %s``
in_aff: (a pathlike object or string representing an existing file)
affine transform file
argument: ``-aff %s``

[Optional]
out_file: (a pathlike object or string representing a file)
output path
argument: ``-out %s``
args: (a unicode string)
Additional parameters to the command
argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environ variables

Outputs:

out_file: (a pathlike object or string representing an existing file)

60.2.7 Diffeo

Link to code
Wraps the executable command `dti_diffeomorphic_reg`. Performs diffeomorphic registration between two tensor volumes

Example

```python
>>> from nipype.interfaces import dtitk
>>> node = dtitk.Diffeo()
>>> node.inputs.fixed_file = 'im1.nii'
>>> node.inputs.moving_file = 'im2.nii'
>>> node.inputs.mask_file = 'mask.nii'
>>> node.inputs.legacy = 1
>>> node.inputs.n_iters = 6
>>> node.inputs.ftol = 0.002
>>> node.cmdline
'dti_diffeomorphic_reg im1.nii im2.nii mask.nii 1 6 0.002'
>>> node.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
legacy: (1, nipype default value: 1)
  legacy parameter; always set to 1
  argument: ``-%d``, position: 3
n_iters: (an integer (int or long), nipype default value: 6)
  number of iterations
  argument: ``-%d``, position: 4
ftol: (a float, nipype default value: 0.002)
  iteration for the optimization to stop
  argument: ``-%g``, position: 5

[Optional]
fixed_file: (a pathlike object or string representing an existing
DiffeoScalarVol

Link to code
Wraps the executable command deformationScalarVolume.
Applies diffeomorphic transform to a scalar volume

Example

```python
>>> from nipype.interfaces import dtitk
>>> node = dtitk.DiffeoScalarVol()
>>> node.inputs.in_file = 'im1.nii'
>>> node.inputs.transform = 'im_warp.df.nii'
>>> node.cmdline
'deformationScalarVolume -in im1.nii -interp 0 -out im1_diffeoxfmd.nii
-trans im_warp.df.nii'
>>> node.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  moving scalar volume
  argument: ``-in %s``
transform: (a pathlike object or string representing an existing file)
  transform to apply
  argument: ``-trans %s``

[Optional]
out_file: (a pathlike object or string representing a file)
  output filename
argument: `"-out %s"`

target: (a pathlike object or string representing an existing file)
output volume specification read from the target volume if specified
argument: `"-target %s"
mutually_exclusive: voxel_size

voxel_size: (a tuple of the form: (a float, a float, a float))
xyz voxel size (superseded by target)
argument: `"-vsize %g %g %g"
mutually_exclusive: target

flip: (a tuple of the form: (an integer (int or long), an integer
(int or long), an integer (int or long)))
argument: `"-flip %d %d %d"

resampling_type: ('backward' or 'forward')
use backward or forward resampling
argument: `"-type %s"

interpolation: ('trilinear' or 'NN', nipype default value: trilinear)
trilinear, or nearest neighbor
argument: `"-interp %s"

args: (a unicode string)
Additional parameters to the command
argument: `"%s"

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})

Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
moved volume

60.2.9 DiffeoSymTensor3DVol

Link to code
Wraps the executable command deformationSymTensor3DVolume.
Applies diffeomorphic transform to a tensor volume

Example

```python
>>> from nipype.interfaces import dtitk
>>> node = dtitk.DiffeoSymTensor3DVol()
>>> node.inputs.in_file = 'im1.nii'
>>> node.inputs.transform = 'im_warp.df.nii'
>>> node.cmdline
'deformationSymTensor3DVolume -df FD -in im1.nii -interp LEI -out
im1_diffeoXfmd.nii -reorient PPD -trans im_warp.df.nii'
>>> node.run() # doctest: +SKIP

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
moving tensor volume
argument: `"-in %s"

transform: (a pathlike object or string representing an existing file)
transform to apply
  argument: `'-trans %s`'  

[Optional]
out_file: (a pathlike object or string representing a file)
  output filename
  argument: `'-out %s`'

df: (a unicode string, nipype default value: FD)
  argument: `'-df %s`'

interpolation: ('LEI' or 'EI', nipype default value: LEI)
  Log Euclidean/Euclidean Interpolation
  argument: `'-interp %s`'

reorient: ('PPD' or 'FS', nipype default value: PPD)
  Reorientation strategy: preservation of principal direction or
  finite strain
  argument: `'-reorient %s`'

target: (a pathlike object or string representing an existing file)
  output volume specification read from the target volume if specified
  argument: `'-target %s`'

mutually_exclusive: voxel_size

voxel_size: (a tuple of the form: (a float, a float, a float))
  xyz voxel size (superseded by target)
  argument: `'-vsize %g %g %g`'

mutually_exclusive: target

flip: (a tuple of the form: (an integer (int or long), an integer
  (int or long), an integer (int or long)))
  argument: `'-flip %d %d %d`'

resampling_type: ('backward' or 'forward')
  use backward or forward resampling
  argument: `'-type %s`'

args: (a unicode string)
  Additional parameters to the command
  argument: `'%s`'

environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
iteration for the optimization to stop
argument: `"%g"`, position: 5

[Optional]
fixed_file: (a pathlike object or string representing an existing file)
fixed tensor volume
argument: `"%s"`, position: 0
moving_file: (a pathlike object or string representing an existing file)
moving tensor volume
argument: `"%s"`, position: 1
mask_file: (a pathlike object or string representing an existing file)
mask
argument: `"%s"`, position: 2
args: (a unicode string)
Additional parameters to the command
argument: `"%s"

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
out_file_xfm: (a pathlike object or string representing an existing file)

60.2.11 Rigid

Link to code
Wraps the executable command dti_rigid_reg.
Performs rigid registration between two tensor volumes

Example

```python
>>> from nipype.interfaces import dtitk
>>> node = dtitk.Rigid()
>>> node.inputs.fixed_file = 'im1.nii'
>>> node.inputs.moving_file = 'im2.nii'
>>> node.inputs.similarity_metric = 'EDS'
>>> node.inputs.sampling_xyz = (4,4,4)
>>> node.inputs.ftol = 0.01
>>> node.cmdline
'dti_rigid_reg im1.nii im2.nii EDS 4 4 4 0.01'
>>> node.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
fixed_file: (a pathlike object or string representing an existing file)
fixed tensor volume
argument: `"%s"`, position: 0
moving_file: (a pathlike object or string representing an existing file)

  moving tensor volume
  argument: "%s", position: 1

similarity_metric: ('EDS' or 'GDS' or 'DDS' or 'NMI', nipype default value: EDS)

  similarity metric
  argument: "%s", position: 2

sampling_xyz: (a tuple of the form: (a value of class 'int', a value of class 'int'), nipype default value: (4, 4, 4))

  dist between samp points (mm) (x,y,z)
  argument: "%g %g %g", position: 3

ftol: (a float, nipype default value: 0.01)

  cost function tolerance
  argument: "%g", position: 4

[Optional]

initialize_xfm: (a pathlike object or string representing an existing file)

  Initialize w/DTITK-FORMATaffine
  argument: "%s", position: 5

args: (a unicode string)

  Additional parameters to the command
  argument: "%s"

eviron: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

  Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)

out_file_xfm: (a pathlike object or string representing an existing file)

60.2.12 RigidTask

Link to code

Wraps the executable command dti_rigid_reg.

Inputs:

[Optional]

fixed_file: (a pathlike object or string representing an existing file)

  fixed tensor volume
  argument: "%s", position: 0

moving_file: (a pathlike object or string representing an existing file)

  moving tensor volume
  argument: "%s", position: 1

similarity_metric: ('EDS' or 'GDS' or 'DDS' or 'NMI', nipype default value: EDS)

  similarity metric
  argument: "%s", position: 2

sampling_xyz: (a tuple of the form: (a value of class 'int', a value of class 'int'), nipype default value: (4, 4, 4))

  dist between samp points (mm) (x,y,z)
  argument: "%g %g %g", position: 3

ftol: (a float, nipype default value: 0.01)

  cost function tolerance
  argument: "%g", position: 4

[Optional]

initialize_xfm: (a pathlike object or string representing an existing file)

  Initialize w/DTITK-FORMATaffine
  argument: "%s", position: 5

args: (a unicode string)

  Additional parameters to the command
  argument: "%s"

eviron: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipipe default value: {})

  Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)

out_file_xfm: (a pathlike object or string representing an existing file)
of class 'int', a value of class 'int'), nipype default value: (4, 4, 4))

dist between samp points (mm) (x,y,z)
argument: "%g %g %g", position: 3

ftol: (a float, nipype default value: 0.01)
cost function tolerance
argument: "%g", position: 4

[Optional]
initialize_xfm: (a pathlike object or string representing an existing file)
  Initialize w/DTITK-FORMATaffine
argument: "%s", position: 5

args: (a unicode string)
  Additional parameters to the command
argument: "%s"
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

60.2.13 affScalarVolTask

Link to code
Wraps the executable command affineScalarVolume.

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  moving scalar volume
argument: "-in %s"

[Optional]
out_file: (a pathlike object or string representing a file)
  output filename
argument: "-out %s"
transform: (a pathlike object or string representing an existing file)
  transform to apply: specify an input transformation file; parameters input will be ignored
argument: "-trans %s"
  mutually_exclusive: target, translation, euler, deformation
interpolation: ('trilinear' or 'NN', nipype default value: trilinear)
  trilinear or nearest neighbor interpolation
argument: "-interp %s"
target: (a pathlike object or string representing an existing file)
  output volume specification read from the target volume if specified
argument: "-target %s"
  mutually_exclusive: transform
translation: (a tuple of the form: (a float, a float, a float))
translation (x, y, z) in mm
argument: ``-translation %g %g %g``
mutually_exclusive: transform
euler: (a tuple of the form: (a float, a float, a float))
(theta, phi, psi) in degrees
argument: ``-euler %g %g %g``
mutually_exclusive: transform
defformation: (a tuple of the form: (a float, a float, a float, a float, a float, a float))
(xx, yy, zz, xy, yz, xz)
argument: ``-deformation %g %g %g %g %g %g``
mutually_exclusive: transform
args: (a unicode string)
Additional parameters to the command
argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
moved volume

60.2.14 affSymTensor3DVolTask

Link to code
Wraps the executable command affineSymTensor3DVolume.

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
moving tensor volume
argument: ``-in %s``

[Optional]
out_file: (a pathlike object or string representing a file)
output filename
argument: ``-out %s``
transform: (a pathlike object or string representing an existing file)
transform to apply: specify an input transformation file; parameters input will be ignored
argument: ``-trans %s``
mutually_exclusive: target, translation, euler, deformation
interpolation: ('LEI' or 'EI', nipype default value: LEI)
Log Euclidean/Euclidean Interpolation
argument: ``-interp %s``
reorient: ('PPD' or 'NO' or 'FS', nipype default value: PPD)
Reorientation strategy: preservation of principal direction, no reorientation, or finite strain
argument: ``-reorient %s``
target: (a pathlike object or string representing an existing file)
output volume specification read from the target volume if specified
argument: ``-target %s``
mutually_exclusive: transform
translation: (a tuple of the form: (a float, a float, a float))
  translation (x,y,z) in mm
  argument: `--translation %g %g %g`
mutually_exclusive: transform
euler: (a tuple of the form: (a float, a float, a float))
  (theta, phi, psi) in degrees
  argument: `--euler %g %g %g`
mutually_exclusive: transform
defection: (a tuple of the form: (a float, a float, a float, a float, a float, a float))
  (xx,yy,zz,xy,yz,xz)
  argument: `--deformation %g %g %g %g %g %g`
mutually_exclusive: transform
args: (a unicode string)
  Additional parameters to the command
  argument: `--%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)

60.2.15 diffeoScalarVolTask

Link to code
Wraps the executable command deformationScalarVolume. Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  moving scalar volume
  argument: `--in %s`
transform: (a pathlike object or string representing an existing file)
  transform to apply
  argument: `--trans %s`

[Optional]
out_file: (a pathlike object or string representing a file)
  output filename
  argument: `--out %s`
target: (a pathlike object or string representing an existing file)
  output volume specification read from the target volume if specified
  argument: `--target %s`
  mutually_exclusive: voxel_size
voxel_size: (a tuple of the form: (a float, a float, a float))
  xyz voxel size (superseded by target)
  argument: `--vsize %g %g %g`
  mutually_exclusive: target
flip: (a tuple of the form: (an integer (int or long), an integer (int or long), an integer (int or long)))
  argument: `--flip %d %d %d`

(continues on next page)
resampling_type: ('backward' or 'forward')
use backward or forward resampling
argument: `'-type %s`'
interpolation: ('trilinear' or 'NN', nipype default value: trilinear)
trilinear, or nearest neighbor
argument: `'-interp %s`'
args: (a unicode string)
Additional parameters to the command
argument: `'%s`'
environ: (a dictionary with keys which are a bytes or None or a value
of class `str` and with values which are a bytes or None or a
value of class `str`, nipipe default value: {})
Environment variables

Outputs:
out_file: (a pathlike object or string representing an existing file)
moved volume

60.2.16 diffeoSymTensor3DVolTask

Link to code
Wraps the executable command deformationSymTensor3DVolume.

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  moving tensor volume
  argument: `'-in %s`'
transform: (a pathlike object or string representing an existing file)
  transform to apply
  argument: `'-trans %s`'

[Optional]
out_file: (a pathlike object or string representing a file)
  output filename
  argument: `'-out %s`'
df: (a unicode string, nipype default value: FD)
  argument: `'-df %s`'
interpolation: ('LEI' or 'EI', nipipe default value: LEI)
  Log Euclidean/Euclidean Interpolation
  argument: `'-interp %s`'
reorient: ('PPD' or 'FS', nipipe default value: PPD)
  Reorientation strategy: preservation of principal direction or
  finite strain
  argument: `'-reorient %s`'
target: (a pathlike object or string representing an existing file)
  output volume specification read from the target volume if specified
  argument: `'-target %s`
  mutually_exclusive: voxel_size
voxel_size: (a tuple of the form: (a float, a float, a float))
  xyz voxel size (superseded by target)
  argument: `'-vsize %g %g %g`
  mutually_exclusive: target
flip: (a tuple of the form: (an integer (int or long), an integer)
(int or long), an integer (int or long)))

argument: ``-flip %d %d %d``

resampling_type: ('backward' or 'forward')

use backward or forward resampling
argument: ``-type %s``

args: (a unicode string)

Additional parameters to the command
argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value

class 'str' and with values which are a bytes or None or a

class 'str', nipype default value: {}))

Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)

60.3 interfaces.dtitk.utils

60.3.1 BinThresh

Link to code

Wraps the executable command BinaryThresholdImageFilter.

Binarizes an image

Example

```python
>>> from nipype.interfaces import dtitk
>>> node = dtitk.BinThresh()
>>> node.inputs.in_file = 'im1.nii'
>>> node.inputs.lower_bound = 0
>>> node.inputs.upper_bound = 100
>>> node.inputs.inside_value = 1
>>> node.inputs.outside_value = 0
>>> node.cmdline
'BinaryThresholdImageFilter im1.nii im1_thrbin.nii 0 100 1 0'
>>> node.run() # doctest: +SKIP
```

Inputs:

[Mandatory]

in_file: (a pathlike object or string representing an existing file)

Image to threshold/binarize

argument: ``%s``

lower_bound: (a float, nipype default value: 0.01)

lower bound of binarization range

argument: ``%g``

upper_bound: (a float, nipype default value: 100)

upper bound of binarization range

argument: ``%g``

inside_value: (a float, nipype default value: 1)

value for voxels in binarization range

argument: ``%g``

outside_value: (a float, nipype default value: 0)

value for voxels outside of binarization range
Outputs:

out_file: (a pathlike object or string representing an existing file)

60.3.2 BinThreshTask

Link to code

Wraps the executable command BinaryThresholdImageFilter.

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  Image to threshold/binarize
  argument: `\%s`, position: 0
lower_bound: (a float, nipype default value: 0.01)
  lower bound of binarization range
  argument: `\%g`, position: 2
upper_bound: (a float, nipype default value: 100)
  upper bound of binarization range
  argument: `\%g`, position: 3
inside_value: (a float, nipype default value: 1)
  value for voxels in binarization range
  argument: `\%g`, position: 4
outside_value: (a float, nipype default value: 0)
  value for voxels outside of binarization range
  argument: `\%g`, position: 5

[Optional]
out_file: (a pathlike object or string representing a file)
  output path
  argument: `\%s`, position: 1
args: (a unicode string)
  Additional parameters to the command
  argument: `\%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
60.3.3 SVAdjustVoxSp

Link to code
Wraps the executable command SVAdjustVoxelSpace.
Adjusts the voxel space of a scalar volume

```python
>>> from nipype.interfaces import dtitk
>>> node = dtitk.SVAdjustVoxSp()
>>> node.inputs.in_file = 'im1.nii'
>>> node.inputs.target_file = 'im2.nii'
>>> node.cmdline
'SVAdjustVoxelSpace -in im1.nii -out im1_avs.nii -target im2.nii'
>>> node.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
scalar volume to modify
argument: `'-in %s'`

[Optional]
out_file: (a pathlike object or string representing a file)
output path
argument: `'-out %s'`
target_file: (a pathlike object or string representing a file)
target volume to match
argument: `'-target %s'`
mutually_exclusive: voxel_size, origin
voxel_size: (a tuple of the form: (a float, a float, a float))
xyz voxel size (superseded by target)
argument: `'-vsize %g %g %g'`
mutually_exclusive: target_file
origin: (a tuple of the form: (a float, a float, a float))
xyz origin (superseded by target)
argument: `'-origin %g %g %g'`
mutually_exclusive: target_file
args: (a unicode string)
Additional parameters to the command
argument: `'%s'`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)

60.3.4 SVAdjustVoxSpTask

Link to code
Wraps the executable command SVAdjustVoxelSpace.

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
scalar volume to modify

(continues on next page)
60.3.5 SVResample

Link to code

Wraps the executable command SVResample. Resamples a scalar volume

```python
>>> from nipype.interfaces import dtitk
>>> node = dtitk.SVResample()
>>> node.inputs.in_file = 'im1.nii'
>>> node.inputs.target_file = 'im2.nii'
>>> node.cmdline
'SVResample -in im1.nii -out im1_resampled.nii -target im2.nii'
>>> node.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file) image to resample
  argument:``-in %s``

[Optional]
out_file: (a pathlike object or string representing a file) output path
  argument:``-out %s``

target_file: (a pathlike object or string representing a file) specs read from the target volume
  argument:``-target %s`` mutually_exclusive: voxel_size, origin

voxel_size: (a tuple of the form: (a float, a float, a float)) xyz voxel size (superseded by target)
  argument:``-vsize %g %g %g`` mutually_exclusive: target_file

origin: (a tuple of the form: (a float, a float, a float)) xyz origin (superseded by target)
  argument:``-origin %g %g %g`` mutually_exclusive: target_file

args: (a unicode string) Additional parameters to the command
  argument:``%s``

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
argument: `\'-target %s``
mutually_exclusive: array_size, voxel_size, origin
align: ('center' or 'origin')
how to align output volume to input volume
argument: `\'-align %s``
array_size: (a tuple of the form: (an integer (int or long), an integer (int or long), an integer (int or long)))
resampled array size
argument: `\'-size %d %d %d``
mutually_exclusive: target_file
voxel_size: (a tuple of the form: (a float, a float, a float))
resampled voxel size
argument: `\'-vsize %g %g %g``
mutually_exclusive: target_file
origin: (a tuple of the form: (a float, a float, a float))
xyz origin
argument: `\'-origin %g %g %g``
mutually_exclusive: target_file
args: (a unicode string)
Additional parameters to the command
argument: `\'%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

 Outputs: 

 out_file: (a pathlike object or string representing an existing file)

 60.3.6 SVResampleTask

 Link to code
Wraps the executable command SVResample.

 Inputs: 

 [Mandatory]
in_file: (a pathlike object or string representing an existing file)
  image to resample
  argument: `\'-in %s``

 [Optional]
out_file: (a pathlike object or string representing a file)
  output path
  argument: `\'-out %s``
target_file: (a pathlike object or string representing a file)
  specs read from the target volume
  argument: `\'-target %s``
mutually_exclusive: array_size, voxel_size, origin
align: ('center' or 'origin')
how to align output volume to input volume
argument: `\'-align %s``
array_size: (a tuple of the form: (an integer (int or long), an integer (int or long), an integer (int or long)))
resampled array size
argument: `\'-size %d %d %d``
mutually_exclusive: target_file
voxel_size: (a tuple of the form: (a float, a float, a float))
  resampled voxel size
  argument: ``-vsize %g %g %g``
  mutually_exclusive: target_file
origin: (a tuple of the form: (a float, a float, a float))
  xyz origin
  argument: ``-origin %g %g %g``
  mutually_exclusive: target_file
args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  Environment variables

Outputs:
out_file: (a pathlike object or string representing an existing file)

60.3.7 TVAdjustOriginTask

Link to code
Wraps the executable command TVAdjustVoxelspace.

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  tensor volume to modify
  argument: ``-in %s``

[Optional]
out_file: (a pathlike object or string representing a file)
  output path
  argument: ``-out %s``
target_file: (a pathlike object or string representing a file)
  target volume to match
  argument: ``-target %s``
  mutually_exclusive: voxel_size, origin
voxel_size: (a tuple of the form: (a float, a float, a float))
  xyz voxel size (superseded by target)
  argument: ``-vsize %g %g %g``
  mutually_exclusive: target_file
origin: (a tuple of the form: (a float, a float, a float))
  xyz origin (superseded by target)
  argument: ``-origin %g %g %g``
  mutually_exclusive: target_file
args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  Environment variables

Outputs:
60.3.8 TVAdjustVoxSp

Link to code
Wraps the executable command TVAdjustVoxelspace.
  Adjusts the voxel space of a tensor volume

Example

```python
>>> from nipype.interfaces import dtitk
>>> node = dtitk.TVAdjustVoxSp()
>>> node.inputs.in_file = 'im1.nii'
>>> node.inputs.target_file = 'im2.nii'
>>> node.cmdline
'TVAdjustVoxelspace -in im1.nii -out im1_avs.nii -target im2.nii'
>>> node.run() # doctest: +SKIP
```

Inputs:

|Mandatory|
in_file: (a pathlike object or string representing an existing file)
  tensor volume to modify
  argument: `''-in %s''`

[Optional]
out_file: (a pathlike object or string representing a file)
  output path
  argument: `''-out %s''`

target_file: (a pathlike object or string representing a file)
  target volume to match
  argument: `''-target %s''`
  mutually_exclusive: voxel_size, origin

voxel_size: (a tuple of the form: (a float, a float, a float))
  xyz voxel size (superseded by target)
  argument: `''-vsize %g %g %g''`
  mutually_exclusive: target_file

origin: (a tuple of the form: (a float, a float, a float))
  xyz origin (superseded by target)
  argument: `''-origin %g %g %g''`
  mutually_exclusive: target_file

args: (a unicode string)
  Additional parameters to the command
  argument: `''%s''`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)

60.3.9 TVAdjustVoxSpTask

Link to code
Wraps the executable command TVAdjustVoxelspace.
## Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
    tensor volume to modify
    argument: ``-in %s``

[Optional]
out_file: (a pathlike object or string representing a file)
    output path
    argument: ``-out %s``
target_file: (a pathlike object or string representing a file)
    target volume to match
    argument: ``-target %s``
    mutually_exclusive: voxel_size, origin

voxel_size: (a tuple of the form: (a float, a float, a float))
    xyz voxel size (superseded by target)
    argument: ``-vsize %g %g %g``
    mutually_exclusive: target_file

origin: (a tuple of the form: (a float, a float, a float))
    xyz origin (superseded by target)
    argument: ``-origin %g %g %g``
    mutually_exclusive: target_file

args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
    Environment variables

## Outputs:

out_file: (a pathlike object or string representing an existing file)

### 60.3.10 TVResample

Link to code

Wraps the executable command TVResample.

Resamples a tensor volume

```python
>>> from nipype.interfaces import dtitk
>>> node = dtitk.TVResample()
>>> node.inputs.in_file = 'im1.nii'
>>> node.inputs.target_file = 'im2.nii'
>>> node.cmdline
'TVResample -in im1.nii -out im1_resampled.nii -target im2.nii'
>>> node.run() # doctest: +SKIP
```

## Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
    tensor volume to resample
    argument: ``-in %s``

[Optional]
out_file: (a pathlike object or string representing a file)
output path
argument: `\'\'-out %s`\`
target_file: (a pathlike object or string representing a file)
specs read from the target volume
argument: `\'\'-target %s`\`
mutually_exclusive: array_size, voxel_size, origin
align: (\'center\' or \'origin\')
how to align output volume to input volume
argument: `\'\'-align %s`\`
interpolation: (\'LEI\' or \'EI\')
Log Euclidean Euclidean Interpolation
argument: `\'\'-interp %s`\`
array_size: (a tuple of the form: (an integer (int or long), an integer (int or long), an integer (int or long)))
resampled array size
argument: `\'\'-size %d %d %d`\`
mutually_exclusive: target_file
voxel_size: (a tuple of the form: (a float, a float, a float))
resampled voxel size
argument: `\'\'-vsize %g %g %g`\`
mutually_exclusive: target_file
origin: (a tuple of the form: (a float, a float, a float))
xyz origin
argument: `\'\'-origin %g %g %g`\`
mutually_exclusive: target_file
args: (a unicode string)
Additional parameters to the command
argument: `\'\%-s`\`
environ: (a dictionary with keys which are a bytes or None or a value of class \'str\' and with values which are a bytes or None or a value of class \'str\', nipype default value: `{}`)
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)

60.3.11 TVResampleTask

Link to code
Wraps the executable command TVResample.

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
tensor volume to resample
argument: `\'\'-in %s`\`

[Optional]
out_file: (a pathlike object or string representing a file)
output path
argument: `\'\'-out %s`\`
target_file: (a pathlike object or string representing a file)
specs read from the target volume
argument: `\'\'-target %s`\`
mutually_exclusive: array_size, voxel_size, origin
align: ('center' or 'origin')
how to align output volume to input volume
argument: `'-align %s'`
interpolation: ('LEI' or 'EI')
Log Euclidean Euclidean Interpolation
argument: `'-interp %s'`
array_size: (a tuple of the form: (an integer (int or long), an integer (int or long), an integer (int or long)))
resampled array size
argument: `'-size %d %d %d'
mutually_exclusive: target_file
voxel_size: (a tuple of the form: (a float, a float, a float))
resampled voxel size
argument: `'-vsize %g %g %g'
mutually_exclusive: target_file
origin: (a tuple of the form: (a float, a float, a float))
xyz origin
argument: `'-origin %g %g %g'
mutually_exclusive: target_file
args: (a unicode string)
Additional parameters to the command
argument: `'%s'
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)

**60.3.12 TVtool**

Link to code
Wraps the executable command TVtool.
Calculates a tensor metric volume from a tensor volume

```
>>> from nipype.interfaces import dtitk
>>> node = dtitk.TVtool()
>>> node.inputs.in_file = 'im1.nii'
>>> node.inputs.in_flag = 'fa'
>>> node.cmdline
'TVtool -in im1.nii -fa -out im1_fa.nii'
>>> node.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
scalar volume to resample
argument: `'-in %s'`

[Optional]
in_flag: ('fa' or 'tr' or 'ad' or 'rd' or 'pd' or 'rgb')
argument: `'-%s'`
out_file: (a pathlike object or string representing a file)
argument: `'-out %s'`
args: (a unicode string)
    Additional parameters to the command
    argument: `\%s`  
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: `{}`)
    Environment variables

Outputs:

out_file: (a pathlike object or string representing a file)

### 60.3.13 TVtoolTask

**Link to code**
Wraps the executable command TVtool.

**Inputs:**

[Mandatory]

in_file: (a pathlike object or string representing an existing file)
    scalar volume to resample
    argument: `\-in \%s`

[Optional]

in_flag: ('fa' or 'tr' or 'ad' or 'rd' or 'pd' or 'rgb')
    argument: `\-%s`

out_file: (a pathlike object or string representing a file)
    argument: `\-out \%s`

args: (a unicode string)
    Additional parameters to the command
    argument: `\%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: `{}`)
    Environment variables

Outputs:

out_file: (a pathlike object or string representing a file)
61.1 interfaces.elastix.registration

61.1.1 AnalyzeWarp

Link to code
Wraps the executable command transformix. Use transformix to get details from the input transform (generate the corresponding deformation field, generate the determinant of the Jacobian map or the Jacobian map itself)

Example

```python
>>> from nipype.interfaces.elastix import AnalyzeWarp
>>> reg = AnalyzeWarp()
>>> reg.inputs.transform_file = 'TransformParameters.0.txt'
>>> reg.cmdline
'transformix -def all -jac all -jacmat all -threads 1 -out ./ -tp
→TransformParameters.0.txt'
```

Inputs:

```plaintext
[Mandatory]
transform_file: (a pathlike object or string representing an existing file)
   transform-parameter file, only 1 argument: ``-tp %s``
output_path: (a pathlike object or string representing an existing directory, nipype default value: ./)
   output directory argument: ``-out %s``

[Optional]
points: ('all', nipype default value: all)
   transform all points from the input-image, which effectively generates a deformation field.
   argument: ``-def %s``, position: 0
jac: ('all', nipype default value: all)
```

(continues on next page)
generate an image with the determinant of the spatial Jacobian argument: ``-jac %s``

jacmat: (all, nipype default value: all)
generate an image with the spatial Jacobian matrix at each voxel argument: ``-jacmat %s``
moving_image: (a pathlike object or string representing an existing file) input image to deform (not used) argument: ``-in %s``

num_threads: (an integer (int or long), nipype default value: 1) set the maximum number of threads of elastix argument: ``-threads %01d``

args: (a unicode string) Additional parameters to the command argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

disp_field: (a pathlike object or string representing a file) displacements field
jactdet_map: (a pathlike object or string representing a file) det(Jacobian) map
jacmat_map: (a pathlike object or string representing a file) Jacobian matrix map

61.1.2 ApplyWarp

Link to code

Wraps the executable command transformix. Use transformix to apply a transform on an input image. The transform is specified in the transform-parameter file.

Example

```python
>>> from nipype.interfaces.elastix import ApplyWarp
>>> reg = ApplyWarp()
>>> reg.inputs.moving_image = 'moving1.nii'
>>> reg.inputs.transform_file = 'TransformParameters.0.txt'
>>> reg.cmdline
'transformix -in moving1.nii -threads 1 -out ./ -tp TransformParameters.0.txt'
```

Inputs:

[Mandatory]
transform_file: (a pathlike object or string representing an existing file) transform-parameter file, only 1 argument: ``-tp %s``
moving_image: (a pathlike object or string representing an existing file) input image to deform argument: ``-in %s``
output_path: (a pathlike object or string representing an existing
directory, nipype default value: .)
output directory
argument: ` `-out `%s``

[Optional]
num_threads: (an integer (int or long), nipype default value: 1)
set the maximum number of threads of elastix
argument: ` `-threads %01d``
args: (a unicode string)
additional parameters to the command
argument: ` `%s``
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a value
of class 'str', nipype default value: {})
Environment variables

Outputs:

warped_file: (a pathlike object or string representing a file)
input moving image warped to fixed image

61.1.3 PointsWarp

Link to code
Wraps the executable command transformix.
Use transformix to apply a transform on an input point set. The transform is specified in the transform-parameter file.

Example

```python
>>> from nipype.interfaces.elastix import PointsWarp
>>> reg = PointsWarp()
>>> reg.inputs.points_file = 'surf1.vtk'
>>> reg.inputs.transform_file = 'TransformParameters.0.txt'
>>> reg.cmdline
'transformix -threads 1 -out ./ -def surf1.vtk -tp TransformParameters.0.txt'
```

Inputs:

[Mandatory]
points_file: (a pathlike object or string representing an existing
file)
input points (accepts .vtk triangular meshes).
argument: ` `-def `%s``
transform_file: (a pathlike object or string representing an existing
file)
transform-parameter file, only 1
argument: ` `-tp `%s``
output_path: (a pathlike object or string representing an existing
directory, nipype default value: .)
output directory
argument: ` `-out `%s``

[Optional]
num_threads: (an integer (int or long), nipype default value: 1)
61.1.4 Registration

Link to code
Wraps the executable command elastix.
Elastix nonlinear registration interface

Example

```python
>>> from nipype.interfaces.elastix import Registration
>>> reg = Registration()
>>> reg.inputs.fixed_image = 'fixed1.nii'
>>> reg.inputs.moving_image = 'moving1.nii'
>>> reg.inputs.parameters = ['elastix.txt']
>>> reg.cmdline
'elastix -f fixed1.nii -m moving1.nii -threads 1 -out ./ -p elastix.txt'
```

Inputs:

- **fixed_image**: (a pathlike object or string representing an existing file)
  - fixed image
  - argument: `--f %s`
- **moving_image**: (a pathlike object or string representing an existing file)
  - moving image
  - argument: `--m %s`
- **parameters**: (a list of items which are a pathlike object or string representing an existing file)
  - parameter file, elastix handles 1 or more -p argument: `--p %s...`
- **output_path**: (a pathlike object or string representing an existing directory, nipype default value: ./)
  - output directory
  - argument: `--out %s`

[Optional]

- **fixed_mask**: (a pathlike object or string representing an existing file)
  - mask for fixed image
  - argument: `--fMask %s`
moving_mask: (a pathlike object or string representing an existing file)
mask for moving image
argument: `--mMask %s`

initial_transform: (a pathlike object or string representing an existing file)
parameter file for initial transform
argument: `--t0 %s`

num_threads: (an integer (int or long), nipype default value: 1)
set the maximum number of threads of elastix
argument: `--threads %01d`

args: (a unicode string)
Additional parameters to the command
argument: `%%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

transform: (a list of items which are a pathlike object or string representing an existing file)
output transform
warped_file: (a pathlike object or string representing a file)
input moving image warped to fixed image
warped_files: (a list of items which are a pathlike object or string representing a file)
input moving image warped to fixed image at each level
warped_files_flags: (a list of items which are a boolean)
flag indicating if warped image was generated

61.2 interfaces.elastix.utils

61.2.1 EditTransform

Link to code
Manipulates an existing transform file generated with elastix

Example

```python
>>> from nipype.interfaces.elastix import EditTransform
>>> tfm = EditTransform()
>>> tfm.inputs.transform_file = 'TransformParameters.0.txt'  # doctest: +SKIP
>>> tfm.inputs.reference_image = 'fixed1.nii'  # doctest: +SKIP
>>> tfm.inputs.output_type = 'unsigned char'
>>> tfm.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
transform_file: (a pathlike object or string representing an existing file)
transform-parameter file, only 1
[Optional]
reference_image: (a pathlike object or string representing an existing file)
    set a new reference image to change the target coordinate system.
interpolation: ('cubic' or 'linear' or 'nearest', nipype default value: cubic)
    set a new interpolator for transformation
    argument: `FinalBSplineInterpolationOrder`
output_type: ('float' or 'unsigned char' or 'unsigned short' or 'short' or 'unsigned long' or 'long' or 'double')
    set a new output pixel type for resampled images
    argument: `ResultImagePixelType`
output_format: ('nii.gz' or 'nii' or 'mhd' or 'hdr' or 'vtk')
    set a new image format for resampled images
    argument: `ResultImageFormat`
output_file: (a pathlike object or string representing a file)
    the filename for the resulting transform file

Outputs:

output_file: (a pathlike object or string representing an existing file)
    output transform file
62.1 interfaces.freesurfer.longitudinal

62.1.1 FuseSegmentations

Link to code
Wraps the executable command mri_fuse_segmentations.
fuse segmentations together from multiple timepoints

Examples

```python
>>> from nipype.interfaces.freesurfer import FuseSegmentations
>>> fuse = FuseSegmentations()
>>> fuse.inputs.subject_id = 'tp.long.A.template'
>>> fuse.inputs.timepoints = ['tp1', 'tp2']
>>> fuse.inputs.out_file = 'aseg.fused.mgz'
>>> fuse.inputs.in_segmentations = ['aseg.mgz', 'aseg.mgz']
>>> fuse.inputs.in_segmentations_noCC = ['aseg.mgz', 'aseg.mgz']
>>> fuse.inputs.in_norms = ['norm.mgz', 'norm.mgz', 'norm.mgz']
>>> fuse.cmdline
'mri_fuse_segmentations -n norm.mgz -a aseg.mgz -c aseg.mgz tp.long.A.template
  → tp1 tp2'
```

Inputs:

[Mandatory]
timepoints: (a list of items which are a string)
  subject_ids or timepoints to be processed
  argument: ``%s`` , position: -2
out_file: (a pathlike object or string representing a file)
  output fused segmentation file
in_segmentations: (a list of items which are a pathlike object or
  string representing an existing file)
  name of aseg file to use (default: aseg.mgz) must include the aseg
  files for all the given timepoints
  argument: ``-a %s``
in_segmentations_noCC: (a list of items which are a pathlike object
  (continues on next page)
or string representing an existing file)
name of aseg file w/o CC labels (default: aseg.auto_noCCseg.mgz)
must include the corresponding file for all the given timepoints
argument: `'-c %s'`
in_norms: (a list of items which are a pathlike object or string
representing an existing file)
-n <filename> - name of norm file to use (default: norm.mgs) must
include the corresponding norm file for all given timepoints as well
as for the current subject
argument: `'-n %s'`

[Optional]
subject_id: (a string)
subject_id being processed
argument: `'%s'`, position: -3
subjects_dir: (a pathlike object or string representing an existing
directory)
subjects directory
args: (a unicode string)
Additional parameters to the command
argument: `'%s'`
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {}) Environ variables

Outputs:

out_file: (a pathlike object or string representing a file)
output fused segmentation file

62.1.2 RobustTemplate

Link to code
Wraps the executable command mri_robust_template.
construct an unbiased robust template for longitudinal volumes

Examples

```python
>>> from nipype.interfaces.freesurfer import RobustTemplate
>>> template = RobustTemplate()
>>> template.inputs.in_files = ['structural.nii', 'functional.nii']
>>> template.inputs.auto_detect_sensitivity = True
>>> template.inputs.average_metric = 'mean'
>>> template.inputs.initial_timepoint = 1
>>> template.inputs.fixed_timepoint = True
>>> template.inputs.no_iteration = True
>>> template.inputs.subsample_threshold = 200
>>> template.cmdline  #doctest:
'mri_robust_template --satit --average 0 --fixtp --mov structural.nii functional. →
structural.nii --inittp 1 --noit --template mri_robust_template_out.mgz --subsample 200'
>>> template.inputs.out_file = 'T1.nii'
>>> template.cmdline  #doctest:
'mri_robust_template --satit --average 0 --fixtp --mov structural.nii functional. →
structural.nii --inittp 1 --noit --template T1.nii --subsample 200'
```
>>> template.inputs.transform_outputs = ['structural.lta', ...
    'functional.lta']
>>> template.inputs.scaled_intensity_outputs = ['structural-iscale.txt', ...
    'functional-iscale.txt']

```bash
>>> template.cmdline  
#doctest: +ELLIPSIS
'mri_robust_template --satit --average 0 --fixtp --mov structural.nii functional.
→ni --inittp 1 --noit --template T1.nii --iscaleout ../structural-iscale.txt ...
→lta'
```

```bash
>>> template.inputs.transform_outputs = True
>>> template.inputs.scaled_intensity_outputs = True
>>> template.cmdline  
#doctest: +ELLIPSIS
'mri_robust_template --satit --average 0 --fixtp --mov structural.nii functional.
→ni --inittp 1 --noit --template T1.nii --iscaleout ../is1.txt ../is2.txt --
→subsample 200 --lta ../tp1.lta ../tp2.lta'
```

```bash
>>> template.run()  
#doctest: +SKIP
```

References

[https://surfer.nmr.mgh.harvard.edu/fswiki/mri_robust_template]

Inputs:

[Mandatory]
in_files: (a list of items which are a pathlike object or string representing an existing file)
    input movable volumes to be aligned to common mean/median template
    argument: `--mov %s`
out_file: (a pathlike object or string representing a file, nipype default value: mri_robust_template_out.mgz)
    output template volume (final mean/median image)
    argument: `--template %s`
auto_detect_sensitivity: (a boolean)
    auto-detect good sensitivity (recommended for head or full brain scans)
    argument: `--satit`
    mutually_exclusive: outlier_sensitivity
outlier_sensitivity: (a float)
    set outlier sensitivity manually (e.g. `--sat 4.685`). Higher values mean less sensitivity.
    argument: `--sat %.4f`
    mutually_exclusive: auto_detect_sensitivity

[Optional]
transform_outputs: (a list of items which are a pathlike object or string representing a file or a boolean)
    output xforms to template (for each input)
    argument: `--lta %s`
intensity_scaling: (a boolean)
    allow also intensity scaling (default off)
    argument: `--iscale`
scaled_intensity_outputs: (a list of items which are a pathlike object or string representing a file or a boolean)
    final intensity scales (will activate --iscale)
    argument: `--iscaleout %s`

(continues on next page)
subsample_threshold: (an integer (int or long))
  subsample if dim > # on all axes (default no subs.)
  argument: '--subsample %d`
average_metric: ('median' or 'mean')
  construct template from: 0 Mean, 1 Median (default)
  argument: '--average %d`
initial_timepoint: (an integer (int or long))
  use TP# for spacial init (default random), 0: no init
  argument: '--inittp %d`
fixed_timepoint: (a boolean)
  map everthing to init TP# (init TP is not resampled)
  argument: '--fixtp`
no_iteration: (a boolean)
  do not iterate, just create first template
  argument: '--noit`
initial_transforms: (a list of items which are a pathlike object or
  string representing an existing file)
  use initial transforms (lta) on source
  argument: '--ixforms %s`
in_intensity_scales: (a list of items which are a pathlike object or
  string representing an existing file)
  use initial intensity scales
  argument: '--iscalein %s`
num_threads: (an integer (int or long))
  allows for specifying more threads
subjects_dir: (a pathlike object or string representing an existing
  directory)
  subjects directory
args: (a unicode string)
  Additional parameters to the command
  argument: '%s`
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})`
Environment variables

**Outputs:**

out_file: (a pathlike object or string representing an existing file)
  output template volume (final mean/median image)
transform_outputs: (a list of items which are a pathlike object or
  string representing an existing file)
  output xform files from moving to template
scaled_intensity_outputs: (a list of items which are a pathlike
  object or string representing an existing file)
  output final intensity scales

---

### 62.2 interfaces.freesurfer.model

#### 62.2.1 Binarize

Link to code

Wraps the executable command `mri_binarize`. Use FreeSurfer `mri_binarize` to threshold an input volume
Examples

```python
>>> binvol = Binarize(in_file='structural.nii', min=10, binary_file='foo_out.nii')
>>> binvol.cmdline
'mri_binarize --o foo_out.nii --i structural.nii --min 10.000000'
```

Inputs:

[Mandatory]
- **in_file**: (a pathlike object or string representing an existing file)
  - input volume
  - argument: `--i %s`

[Optional]
- **min**: (a float)
  - min thresh
  - argument: `--min %f`
  - mutually_exclusive: `wm_ven_csf`
- **max**: (a float)
  - max thresh
  - argument: `--max %f`
  - mutually_exclusive: `wm_ven_csf`
- **rmin**: (a float)
  - compute min based on rmin*globalmean
  - argument: `--rmin %f`
- **rmax**: (a float)
  - compute max based on rmax*globalmean
  - argument: `--rmax %f`
- **match**: (a list of items which are an integer (int or long))
  - match instead of threshold
  - argument: `--match %d...`
- **wm**: (a boolean)
  - set match vals to 2 and 41 (aseg for cerebral WM)
  - argument: `--wm`
- **ventricles**: (a boolean)
  - set match vals those for aseg ventricles+choroid (not 4th)
  - argument: `--ventricles`
- **wm_ven_csf**: (a boolean)
  - WM and ventricular CSF, including choroid (not 4th)
  - argument: `--wm+vcsf`
  - mutually_exclusive: `min`, `max`
- **binary_file**: (a pathlike object or string representing a file)
  - binary output volume
  - argument: `--o %s`
- **out_type**: (`'nii'` or `'nii.gz'` or `'mgz'`)
  - output file type
- **count_file**: (a boolean or a pathlike object or string representing a file)
  - save number of hits in ascii file (hits, ntotvox, pct)
  - argument: `--count %s`
- **bin_val**: (an integer (int or long))
  - set vox within thresh to val (default is 1)
  - argument: `--binval %d`
- **bin_val_not**: (an integer (int or long))
  - set vox outside range to val (default is 0)
  - argument: `--binvalnot %d`
- **invert**: (a boolean)
  - set binval=0, binvalnot=1

(continues on next page)
argument: `--inv`
frame_no: (an integer (int or long))
    use 0-based frame of input (default is 0)
    argument: `--frame %s`
merge_file: (a pathlike object or string representing an existing
    file)
    merge with mergevol
    argument: `--merge %s`
mask_file: (a pathlike object or string representing an existing
    file)
    must be within mask
    argument: `--mask maskvol`
mask_thres: (a float)
    set thresh for mask
    argument: `--mask-thresh %f`
abs: (a boolean)
    take abs of invol first (ie, make unsigned)
    argument: `--abs`
bin_col_num: (a boolean)
    set binarized voxel value to its column number
    argument: `--bincol`
zero_edges: (a boolean)
    zero the edge voxels
    argument: `--zero-edges`
zero_slice_edge: (a boolean)
    zero the edge slice voxels
    argument: `--zero-slice-edges`
dilate: (an integer (int or long))
    niters: dilate binarization in 3D
    argument: `--dilate %d`
erode: (an integer (int or long))
    nerode: erode binarization in 3D (after any dilation)
    argument: `--erode %d`
erode2d: (an integer (int or long))
    nerode2d: erode binarization in 2D (after any 3D erosion)
    argument: `--erode2d %d`
subjects_dir: (a pathlike object or string representing an existing
    directory)
    subjects directory
args: (a unicode string)
    Additional parameters to the command
    argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables

Outputs:

binary_file: (a pathlike object or string representing an existing
    file)
    binarized output volume
count_file: (a pathlike object or string representing a file)
    ascii file containing number of hits
62.2.2 Concatenate

Link to code

Wraps the executable command `mri_concat`.

Use Freesurfer `mri_concat` to combine several input volumes into one output volume. Can concatenate by frames, or compute a variety of statistics on the input volumes.

Examples

Combine two input volumes into one volume with two frames

```python
>>> concat = Concatenate()
>>> concat.inputs.in_files = ['cont1.nii', 'cont2.nii']
>>> concat.inputs.concatenated_file = 'bar.nii'
>>> concat.cmdline
'mri_concat --o bar.nii --i cont1.nii --i cont2.nii'
```

Inputs:

|Mandatory|
in_files: (a list of items which are a pathlike object or string representing an existing file)
  - Individual volumes to be concatenated
    argument: `--i %s...`

|Optional|
concatenated_file: (a pathlike object or string representing a file)
  - Output volume
    argument: `--o %s`

sign: ("abs" or "pos" or "neg")
  - Take only pos or neg voxles from input, or take abs
    argument: `--%s`

stats: ("sum" or "var" or "std" or "max" or "min" or "mean")
  - Compute the sum, var, std, max, min or mean of the input volumes
    argument: `--%s`

paired_stats: ("sum" or "avg" or "diff" or "diff-norm" or "diff-norm1" or "diff-norm2")
  - Compute paired sum, avg, or diff
    argument: `--paired-%s`

gmean: (an integer (int or long))
  - create matrix to average Ng groups, Nper=Ntot/Ng
    argument: `--gmean %d`

mean_div_n: (a boolean)
  - compute mean/nframes (good for var)
    argument: `--mean-div-n`

multiply_by: (a float)
  - Multiply input volume by some amount
    argument: `--mul %f`

add_val: (a float)
  - Add some amount to the input volume
    argument: `--add %f`

multiply_matrix_file: (a pathlike object or string representing an existing file)
  - Multiply input by an ascii matrix in file
    argument: `--mtx %s`

combine: (a boolean)
  - Combine non-zero values into single frame volume
    argument: `--combine`

keep_dtype: (a boolean)
62.2.3 GLMFit

Link to code
Wraps the executable command mri_glmfit.
Use FreeSurfer’s mri_glmfit to specify and estimate a general linear model.

Examples

```python
>>> glmfit = GLMFit()
>>> glmfit.inputs.in_file = 'functional.nii'
>>> glmfit.inputs.one_sample = True
>>> glmfit.cmdline == 'mri_glmfit --glmdir %s --y functional.nii --osgm' % os.getcwd()
True
```

Inputs:

```python
[Optional]
in_file: (a pathlike object or string representing a file)
input 4D file
argument: '%%s'
```

(continues on next page)
### Optional Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>glm_dir</td>
<td>save outputs to dir</td>
</tr>
<tr>
<td></td>
<td>argument: <code>--glmmdir %s</code></td>
</tr>
<tr>
<td>fsgd</td>
<td>(a tuple of the form: (a pathlike object or string representing an existing file, 'doss' or 'dods'))</td>
</tr>
<tr>
<td></td>
<td>freesurfer descriptor file</td>
</tr>
<tr>
<td></td>
<td>argument: <code>--fsgd %s %s</code></td>
</tr>
<tr>
<td>design</td>
<td>design matrix file</td>
</tr>
<tr>
<td></td>
<td>argument: <code>--X %s</code></td>
</tr>
<tr>
<td>contrast</td>
<td>contrast file</td>
</tr>
<tr>
<td></td>
<td>argument: <code>--C %s...</code></td>
</tr>
<tr>
<td>one_sample</td>
<td>construct X and C as a one-sample group mean</td>
</tr>
<tr>
<td></td>
<td>argument: <code>--osgm</code></td>
</tr>
<tr>
<td>no_contrast_ok</td>
<td>do not fail if no contrasts specified</td>
</tr>
<tr>
<td></td>
<td>argument: <code>--no-contrasts-ok</code></td>
</tr>
<tr>
<td>per_voxel_reg</td>
<td>per-voxel regressors</td>
</tr>
<tr>
<td></td>
<td>argument: <code>--pvr %s...</code></td>
</tr>
<tr>
<td>self_reg</td>
<td>self-regressor from index col row slice</td>
</tr>
<tr>
<td></td>
<td>argument: <code>--selfreg %d %d %d</code></td>
</tr>
<tr>
<td>weighted_ls</td>
<td>weighted least squares</td>
</tr>
<tr>
<td></td>
<td>argument: <code>--wls %s</code></td>
</tr>
<tr>
<td>fixed_fx_var</td>
<td>for fixed effects analysis</td>
</tr>
<tr>
<td></td>
<td>argument: <code>--yffxvar %s</code></td>
</tr>
<tr>
<td>fixed_fx_dof</td>
<td>dof for fixed effects analysis</td>
</tr>
<tr>
<td></td>
<td>argument: <code>--ffxdof %d</code></td>
</tr>
<tr>
<td>weight_file</td>
<td>weight for each input at each voxel</td>
</tr>
<tr>
<td></td>
<td>argument: <code>--w-inv</code></td>
</tr>
<tr>
<td>weight_inv</td>
<td>invert weights</td>
</tr>
<tr>
<td></td>
<td>argument: <code>--w-inv</code></td>
</tr>
</tbody>
</table>

### Mutually Exclusive Arguments

- **fsgd**, **design**, **one_sample**
- **fsgd**, **design**, **contrast**
- **fsgd**, **design**, **no_contrast_ok**
- **fsgd**, **design**, **per_voxel_reg**
- **fsgd**, **design**, **self_reg**
- **fsgd**, **design**, **weighted_ls**
- **fsgd**, **design**, **fixed_fx_var**
- **fsgd**, **design**, **fixed_fx_dof**
- **fsgd**, **design**, **fixed_fx_dof_file**
- **fsgd**, **design**, **weight_file**
- **fsgd**, **design**, **weight_inv**

(continues on next page)
mutually_exclusive: weighted_ls

weight_sqrt: (a boolean)
  sqrt of weights
  argument: `''--w-sqrt'`
  mutually_exclusive: weighted_ls

fwhm: (a floating point number >= 0.0)
  smooth input by fwhm
  argument: `''--fwhm %f'`

var_fwhm: (a floating point number >= 0.0)
  smooth variance by fwhm
  argument: `''--var-fwhm %f'`

no_mask_smooth: (a boolean)
  do not mask when smoothing
  argument: `''--no-mask-smooth''`

no_est_fwhm: (a boolean)
  turn off FWHM output estimation
  argument: `''--no-est-fwhm''`

mask_file: (a pathlike object or string representing an existing file)
  binary mask
  argument: `''--mask %s''`

label_file: (a pathlike object or string representing an existing file)
  use label as mask, surfaces only
  argument: `''--label %s''`
  mutually_exclusive: cortex

invert_mask: (a boolean)
  invert mask
  argument: `''--mask-inv''`

prune: (a boolean)
  remove voxels that do not have a non-zero value at each frame (def)
  argument: `''--prune''`

no_prune: (a boolean)
  do not prune
  argument: `''--no-prune''`
  mutually_exclusive: prune_thresh

prune_thresh: (a float)
  prune threshold. Default is FLT_MIN
  argument: `''--prune-thr %f''`
  mutually_exclusive: no_prune

compute_log_y: (a boolean)
  compute natural log of y prior to analysis
  argument: `''--logy''`

save_estimate: (a boolean)
  save signal estimate (yhat)
  argument: `''--yhat-save''`

save_residual: (a boolean)
  save residual error (eres)
  argument: `''--eres-save''`

save_res_corr_mtx: (a boolean)
  save residual error spatial correlation matrix (eres.scm). Big!
  argument: `''--eres-scm''`

surf: (a boolean)
analysis is on a surface mesh
argument: `''--surf %s %s %s''`
requires: subject_id, hemi

subject_id: (a unicode string)
subject id for surface geometry

hemi: ('lh' or 'rh')
surface hemisphere

surf_geo: (a unicode string, nipype default value: white)
surface geometry name (e.g. white, pial)

simulation: (a tuple of the form: ('perm' or 'mc-full' or 'mc-z', an
integer (int or long), a float, a unicode string))
nulltype nsim thresh csdbasename
argument: `''--sim %s %d %f %s''`
sim_sign: (abs' or 'pos' or 'neg')
abs, pos, or neg
argument: `''--sim-sign %s''`
uniform: (a tuple of the form: (a float, a float))
use uniform distribution instead of gaussian
argument: `''--uniform %f %f''`
pca: (a boolean)
perform pca/svd analysis on residual
argument: `''--pca''`
calc_AR1: (a boolean)
compute and save temporal AR1 of residual
argument: `''--tar1''`

save_cond: (a boolean)
flag to save design matrix condition at each voxel
argument: `''--save-cond''`

vox_dump: (a tuple of the form: (an integer (int or long), an integer
(int or long), an integer (int or long)))
dump voxel GLM and exit
argument: `''--voxdump %d %d %d''`

seed: (an integer (int or long))
used for synthesizing noise
argument: `''--seed %d''`

synth: (a boolean)
replace input with gaussian
argument: `''--synth''`

resynth_test: (an integer (int or long))
test GLM by resynthesis
argument: `''--resynthtest %d''`

profile: (an integer (int or long))
niters : test speed
argument: `''--profile %d''`

force_perm: (a boolean)
force permutation test, even when design matrix is not orthog
argument: `''--perm-force''`

diag: (an integer (int or long))
Gdiag_no : set diagnostic level
argument: `''--diag %d''`

diag_cluster: (a boolean)
save sig volume and exit from first sim loop
argument: `''--diag-cluster''`

debug: (a boolean)
turn on debugging
argument: `''--debug''`

check_opts: (a boolean)
don't run anything, just check options and exit
argument: '--checkopts'
allow_repeated_subjects: (a boolean)
  allow subject names to repeat in the fsgd file (must appear before
--fsgd
  argument: '--allowsubjrep'
allow_ill_cond: (a boolean)
  allow ill-conditioned design matrices
  argument: '--illcond'
sim_done_file: (a pathlike object or string representing a file)
  create file when simulation finished
  argument: '--sim-done %s'
subjects_dir: (a pathlike object or string representing an existing
    directory)
  subjects directory
args: (a unicode string)
  Additional parameters to the command
  argument: '%s'
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
Environment variables

Outputs:

glm_dir: (a pathlike object or string representing an existing
    directory)
  output directory
beta_file: (a pathlike object or string representing an existing
  file)
  map of regression coefficients
error_file: (a pathlike object or string representing a file)
  map of residual error
error_var_file: (a pathlike object or string representing a file)
  map of residual error variance
error_stddev_file: (a pathlike object or string representing a file)
  map of residual error standard deviation
estimate_file: (a pathlike object or string representing a file)
  map of the estimated Y values
mask_file: (a pathlike object or string representing a file)
  map of the mask used in the analysis
fwhm_file: (a pathlike object or string representing a file)
  text file with estimated smoothness
dof_file: (a pathlike object or string representing a file)
  text file with effective degrees-of-freedom for the analysis
gamma_file: (a list of items which are any value)
  map of contrast of regression coefficients
gamma_var_file: (a list of items which are any value)
  map of regression contrast variance
sig_file: (a list of items which are any value)
  map of F-test significance (in -log10p)
ftest_file: (a list of items which are any value)
  map of test statistic values
spatial_eigenvectors: (a pathlike object or string representing a
  file)
  map of spatial eigenvectors from residual PCA
frame_eigenvectors: (a pathlike object or string representing a file)
matrix of frame eigenvectors from residual PCA
singular_values: (a pathlike object or string representing a file)
matrix singular values from residual PCA
svd_stats_file: (a pathlike object or string representing a file)
text file summarizing the residual PCA

### 62.2.4 Label2Annot

**Link to code**

Wraps the executable command `mris_label2annot`.

Converting a set of surface labels to an annotation file

**Examples**

```python
>>> from nipype.interfaces.freesurfer import Label2Annot
>>> l2a = Label2Annot()
>>> l2a.inputs.hemisphere = 'lh'
>>> l2a.inputs.subject_id = '10335'
>>> l2a.inputs.in_labels = ['lh.aparc.label']
>>> l2a.inputs.orig = 'lh.pial'
>>> l2a.inputs.out_annot = 'test'
>>> l2a.cmdline
'mris_label2annot --hemi lh --l lh.aparc.label --a test --s 10335'
```

**Inputs:**

[Mandatory]
- **hemisphere**: ('lh' or 'rh')
  - Input hemisphere
  - argument: `''--hemi %s''`
- **subject_id**: (a string, nipype default value: subject_id)
  - Subject name/ID
  - argument: `''--s %s''`
- **in_labels**: (a list of items which are any value)
  - List of input label files
  - argument: `''--l %s...''`
- **out_annot**: (a string)
  - Name of the annotation to create
  - argument: `''--a %s''`
- **orig**: (a pathlike object or string representing an existing file)
  - implicit {hemisphere}.orig

[Optional]
- **keep_max**: (a boolean)
  - Keep label with highest 'stat' value
  - argument: `''--maxstatwinner''`
- **verbose_off**: (a boolean)
  - Turn off overlap and stat override messages
  - argument: `''--noverbose''`
- **color_table**: (a pathlike object or string representing an existing file)
  - File that defines the structure names, their indices, and their color
  - argument: `''--ctab %s''`
- **copy_inputs**: (a boolean)

(continues on next page)
copy implicit inputs and create a temp subjects_dir

subjects_dir: (a pathlike object or string representing an existing directory)

subjects directory

args: (a unicode string)

Additional parameters to the command argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: `{}`)

Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)

Output annotation file

62.2.5 Label2Label

Link to code

Wraps the executable command mri_label2label.

Converts a label in one subject’s space to a label in another subject’s space using either talairach or spherical as an intermediate registration space.

If a source mask is used, then the input label must have been created from a surface (ie, the vertex numbers are valid). The format can be anything supported by mri_convert or curv or paint. Vertices in the source label that do not meet threshold in the mask will be removed from the label.

Examples

```python
>>> from nipype.interfaces.freesurfer import Label2Label
>>> l2l = Label2Label()
>>> l2l.inputs.hemisphere = 'lh'
>>> l2l.inputs.subject_id = '10335'
>>> l2l.inputs.sphere_reg = 'lh.pial'
>>> l2l.inputs.white = 'lh.pial'
>>> l2l.inputs.source_subject = 'fsaverage'
>>> l2l.inputs.source_label = 'lh-pial.stl'
>>> l2l.inputs.source_white = 'lh.pial'
>>> l2l.inputs.source_sphere_reg = 'lh.pial'

'mri_label2label --hemi lh --trglabel lh-pial_converted.stl --regmethod surface --srclabel lh-pial.stl --srcsubject fsaverage --trgsubject 10335'
```

Inputs:

[Mandatory]

hemisphere: (‘lh’ or ‘rh’)

Input hemisphere

argument: `--hemi %s`

subject_id: (a string, nipype default value: subject_id)

Target subject

argument: `--trgsubject %s`

sphere_reg: (a pathlike object or string representing an existing file)

Implicit input `<hemisphere>.sphere.reg`

white: (a pathlike object or string representing an existing file)
62.2.6 Label2Vol

Link to code
Wraps the executable command mri_label2vol.
Make a binary volume from a Freesurfer label

Examples

```python
>>> binvol = Label2Vol(label_file='cortex.label', template_file='structural.nii',
                      reg_file='register.dat', fill_thresh=0.5, vol_label_file='foo_out.nii')
>>> binvol.cmdline
'mri_label2vol --fillthresh 0.5 --label cortex.label --reg register.dat --temp
  structural.nii --o foo_out.nii'
```

Inputs:
label_file: (a list of items which are a pathlike object or string representing an existing file)
  list of label files
  argument: `--label %s...`
  mutually_exclusive: label_file, annot_file, seg_file, aparc_aseg
annot_file: (a pathlike object or string representing an existing file)
  surface annotation file
  argument: `--annot %s`
  mutually_exclusive: label_file, annot_file, seg_file, aparc_aseg
  requires: subject_id, hemi
seg_file: (a pathlike object or string representing an existing file)
  segmentation file
  argument: `--seg %s`
  mutually_exclusive: label_file, annot_file, seg_file, aparc_aseg
aparc_aseg: (a boolean)
  use aparc+aseg.mgz in subjectdir as seg
  argument: `--aparc+aseg`
  mutually_exclusive: label_file, annot_file, seg_file, aparc_aseg

[Optional]
reg_file: (a pathlike object or string representing an existing file)
  tkregister style matrix VolXYZ = R*LabelXYZ
  argument: `--reg %s`
  mutually_exclusive: reg_file, reg_header, identity
reg_header: (a pathlike object or string representing an existing file)
  label template volume
  argument: `--regheader %s`
  mutually_exclusive: reg_file, reg_header, identity
identity: (a boolean)
  set R=I
  argument: `--identity`
  mutually_exclusive: reg_file, reg_header, identity
invert_mtx: (a boolean)
  Invert the registration matrix
  argument: `--invertmtx`
fill_thres:(0.0 <= a floating point number <= 1.0)
  thresh : between 0 and 1
  argument: `--fillthresh %g`
label_voxel_volume: (a float)
  volume of each label point (def 1mm3)
  argument: `--labvoxvol %f`
proj: (a tuple of the form: ('abs' or 'frac', a float, a float, a float))
  project along surface normal
  argument: `--proj %s %f %f %f`
  requires: subject_id, hemi
subject_id: (a unicode string)
  subject id
  argument: `--subject %s`
hemi: ('lh' or 'rh')

(continues on next page)
hemisphere to use lh or rh
argument: `--hemi %s`
surface: (a unicode string)
use surface instead of white
argument: `--surf %s`
vol_label_file: (a pathlike object or string representing a file)
output volume
argument: `--o %s`
label_hit_file: (a pathlike object or string representing a file)
file with each frame is nhits for a label
argument: `--hits %s`
map_label_stat: (a pathlike object or string representing a file)
map the label stats field into the vol
argument: `--label-stat %s`
native_vox2ras: (a boolean)
use native vox2ras xform instead of tkregister-style
argument: `--native-vox2ras`
subjects_dir: (a pathlike object or string representing an existing
directory)
subjects directory
args: (a unicode string)
Additional parameters to the command
argument: `%%%s`
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: { })
Environment variables

Outputs:

vol_label_file: (a pathlike object or string representing an existing
file)
output volume

62.2.7 MRISPreproc

Link to code
Wraps the executable command mris_preproc.
Use FreeSurfer mris_preproc to prepare a group of contrasts for a second level analysis

Examples

```python
>>> preproc = MRISPreproc()
>>> preproc.inputs.target = 'fsaverage'
>>> preproc.inputs.hemi = 'lh'
>>> preproc.inputs.vol_measure_file = [('cont1.nii', 'register.dat'),
                                      ('cont1a.nii', 'register.dat')]
>>> preproc.inputs.out_file = 'concatenated_file.mgz'
>>> preproc.cmdline
'mris_preproc --hemi lh --out concatenated_file.mgz --target fsaverage --iv cont1.nii register.dat --iv cont1a.nii register.dat'
```

Inputs:

[Optional]

[Mandatory]
target: (a unicode string)
target subject name
  argument: ``--target %s``

hemi: ('lh' or 'rh')
  hemisphere for source and target
  argument: ``--hemi %s``

[Optional]
out_file: (a pathlike object or string representing a file)
  output filename
  argument: ``--out %s``

surf_measure: (a unicode string)
  Use subject/surf/hemi.surf_measure as input
  argument: ``--meas %s``
  mutually_exclusive: surf_measure, surf_measure_file, surf_area

surf_area: (a unicode string)
  Extract vertex area from subject/surf/hemi.surfname to use as input.
  argument: ``--area %s``
  mutually_exclusive: surf_measure, surf_measure_file, surf_area

subjects: (a list of items which are any value)
  subjects from who measures are calculated
  argument: ``--s %s...``
  mutually_exclusive: subjects, fsgd_file, subject_file

fsgd_file: (a pathlike object or string representing an existing file)
  specify subjects using fsgd file
  argument: ``--fsgd %s``
  mutually_exclusive: subjects, fsgd_file, subject_file

subject_file: (a pathlike object or string representing an existing file)
  file specifying subjects separated by white space
  argument: ``--f %s``
  mutually_exclusive: subjects, fsgd_file, subject_file

surf_measure_file: (a list of items which are a pathlike object or
  string representing an existing file)
  file alternative to surfmeas, still requires list of subjects
  argument: ``--is %s...``
  mutually_exclusive: surf_measure, surf_measure_file, surf_area

source_format: (a unicode string)
  source format
  argument: ``--srcfmt %s``

surf_dir: (a unicode string)
  alternative directory (instead of surf)
  argument: ``--surfdir %s``

vol_measure_file: (a list of items which are a tuple of the form: (a
  pathlike object or string representing an existing file, a
  pathlike object or string representing an existing file))
  list of volume measure and reg file tuples
  argument: ``--iv %s %s...``

proj_frac: (a float)
  projection fraction for vol2surf
  argument: ``--projfrac %s``

fwhm: (a float)
  smooth by fwhm mm on the target surface
  argument: ``--fwhm %f``
  mutually_exclusive: num_iters

num_iters: (an integer (int or long))
  niter : smooth by niter on the target surface

(continues on previous page)
argument: `--niters %d`
mutually exclusive: fwhm

fwhm_source: (a float)
  smooth by fwhm mm on the source surface
argument: `--fwhm-src %f`
mutually exclusive: num_iters_source

num_iters_source: (an integer (int or long))
  niters : smooth by niters on the source surface
argument: `--niterssrc %d`
mutually exclusive: fwhm_source

smooth_cortex_only: (a boolean)
  only smooth cortex (ie, exclude medial wall)
argument: `--smooth-cortex-only`

subjects_dir: (a pathlike object or string representing an existing directory)
  subjects directory

args: (a unicode string)
  Additional parameters to the command
argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing a file)
  preprocessed output file

62.2.8 MRISPreprocReconAll

Link to code
Wraps the executable command mris_preproc.
Extends MRISPreproc to allow it to be used in a recon-all workflow

Examples

```python
>>> preproc = MRISPreprocReconAll()
>>> preproc.inputs.target = 'fsaverage'
>>> preproc.inputs.hemi = 'lh'
>>> preproc.inputs.vol_measure_file = [('cont1.nii', 'register.dat'), ('cont1a.nii', 'register.dat')]
>>> preproc.inputs.out_file = 'concatenated_file.mgz'
>>> preproc.cmdline
'mris_preproc --hemi lh --out concatenated_file.mgz --s subject_id --target fsaverage --iv cont1.nii register.dat --iv cont1a.nii register.dat'
```

Inputs:

[Mandatory]
target: (a unicode string)
  target subject name
  argument: `--target %s`
hemi: ('lh' or 'rh')
  hemisphere for source and target
  argument: `--hemi %s`

(continues on next page)
surf_measure_file: (a pathlike object or string representing an existing file)
  file necessary for surfmeas
argument: `--meas %s`
mutually_exclusive: surf_measure, surf_measure_file, surf_area

surfreg_files: (a list of items which are a pathlike object or string representing an existing file)
  lh and rh input surface registration files
argument: `--surfreg %s`
requires: lh_surfreg_target, rh_surfreg_target

lh_surfreg_target: (a pathlike object or string representing a file)
  Implicit target surface registration file
requires: surfreg_files

rh_surfreg_target: (a pathlike object or string representing a file)
  Implicit target surface registration file
requires: surfreg_files

subject_id: (a string, nipype default value: subject_id)
  subject from whom measures are calculated
argument: `--s %s`
mutually_exclusive: subjects, fsgd_file, subject_file, subject_id
copy_inputs: (a boolean)
  If running as a node, set this to True this will copy some implicit inputs to the node directory.

out_file: (a pathlike object or string representing a file)
  output filename
argument: `--out %s`

surf_measure: (a unicode string)
  Use subject/surf/hemi.surf_measure as input
argument: `--meas %s`
mutually_exclusive: surf_measure, surf_measure_file, surf_area

surf_area: (a unicode string)
  Extract vertex area from subject/surf/hemi.surfname to use as input.
argument: `--area %s`
mutually_exclusive: surf_measure, surf_measure_file, surf_area

subjects: (a list of items which are any value)
  subjects from who measures are calculated
argument: `--s %s...`
mutually_exclusive: subjects, fsgd_file, subject_file

fsgd_file: (a pathlike object or string representing an existing file)
  specify subjects using fsgd file
argument: `--fsgd %s`
mutually_exclusive: subjects, fsgd_file, subject_file

subject_file: (a pathlike object or string representing an existing file)
  file specifying subjects separated by white space
argument: `--f %s`
mutually_exclusive: subjects, fsgd_file, subject_file

source_format: (a unicode string)
  source format
argument: `--srcfmt %s`

surf_dir: (a unicode string)
  alternative directory (instead of surf)
argument: `--surfdir %s`

vol_measure_file: (a list of items which are a tuple of the form: (a}
pathlike object or string representing an existing file, a
list of volume measure and reg file tuples
argument: `--iv %s %s...`

proj_frac: (a float)
projection fraction for vol2surf
argument: `--projfrac %s`

fwhm: (a float)
smooth by fwhm mm on the target surface
argument: `--fwhm %f`
mutually exclusive: num_iters

num_iters: (an integer (int or long))
niters : smooth by niters on the target surface
argument: `--nitors %d`
mutually exclusive: fwhm

fwhm_source: (a float)
smooth by fwhm mm on the source surface
argument: `--fwhm-src %f`
mutually exclusive: num_iters_source

num_iters_source: (an integer (int or long))
niters : smooth by niters on the source surface
argument: `--nitorsrc %d`
mutually exclusive: fwhm_source

smooth_cortex_only: (a boolean)
only smooth cortex (ie, exclude medial wall)
argument: `--smooth-cortex-only`

subjects_dir: (a pathlike object or string representing an existing
directory)

args: (a unicode string)
Additional parameters to the command
argument: `%%s`

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})

Environment variables

Outputs:

out_file: (a pathlike object or string representing a file)
preprocessed output file

62.2.9 MS_LDA

Link to code
Wraps the executable command mri_ms_LDA.
Perform LDA reduction on the intensity space of an arbitrary # of FLASH images

Examples

```python
>>> grey_label = 2
>>> white_label = 3
>>> zero_value = 1
>>> optimalWeights = MS_LDA(lda_labels=[grey_label, white_label],
  label_file='label.mgz', weight_file='weights.txt',
  shift_zero_value=False, vol_synch_file='synth_out.mgz',
  conform=True, use_weights=True,
  images=['FLASH1.mgz', 'FLASH2.mgz', 'FLASH3.mgz'])
```
>>> optimalWeights.cmdline
'mri_ms_LDA -conform -label label.mgz -lda 2 3 -shift 1 -W -synth synth_out.mgz -
→weight weights.txt FLASH1.mgz FLASH2.mgz FLASH3.mgz'

Inputs:

[Mandatory]
lda_labels: (a list of from 2 to 2 items which are an integer (int or
long))
  pair of class labels to optimize
  argument: '--lda %s'
weight_file: (a pathlike object or string representing a file)
  filename for the LDA weights (input or output)
  argument: '--weight %s'
volsynth_file: (a pathlike object or string representing a file)
  filename for the synthesized output volume
  argument: '--synth %s'
images: (a list of items which are a pathlike object or string
  representing an existing file)
  list of input FLASH images
  argument: `"%s"`, position: -1

[Optional]
label_file: (a pathlike object or string representing an existing
  file)
  filename of the label volume
  argument: `"-label %s"
mask_file: (a pathlike object or string representing an existing
  file)
  filename of the brain mask volume
  argument: `"-mask %s"
shift: (an integer (int or long))
  shift all values equal to the given value to zero
  argument: `"-shift %d"
conform: (a boolean)
  Conform the input volumes (brain mask typically already conformed)
  argument: `"-conform"
use_weights: (a boolean)
  Use the weights from a previously generated weight file
  argument: `"-W"
subjects_dir: (a pathlike object or string representing an existing
directory)
  subjects directory
args: (a unicode string)
  Additional parameters to the command
  argument: `"%s"
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})'
  Environment variables

Outputs:

weight_file: (a pathlike object or string representing an existing
  file)
volsynth_file: (a pathlike object or string representing an existing
  file)
62.2.10 OneSampleTTest

Link to code
Wraps the executable command mri_glmfit.

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing a file)
  input 4D file
  argument: `''--y %s''`

[Optional]
glm_dir: (a unicode string)
  save outputs to dir
  argument: `''--glmdir %s''`

fsgd: (a tuple of the form: (a pathlike object or string representing
  an existing file, 'doss' or 'dods'))
  freesurfer descriptor file
  argument: `''--fsgd %s %s''`
  mutually_exclusive: fsgd, design, one_sample

design: (a pathlike object or string representing an existing file)
  design matrix file
  argument: `''--X %s''`
  mutually_exclusive: fsgd, design, one_sample

contrast: (a list of items which are a pathlike object or string
  representing an existing file)
  contrast file
  argument: `''--C %s...''`

one_sample: (a boolean)
  construct X and C as a one-sample group mean
  argument: `''--osgm''`
  mutually_exclusive: one_sample, fsgd, design, contrast

no_contrast_ok: (a boolean)
  do not fail if no contrasts specified
  argument: `''--no-contrasts-ok''`

per_voxel_reg: (a list of items which are a pathlike object or string
  representing an existing file)
  per-voxel regressors
  argument: `''--pvr %s...''`

self_reg: (a tuple of the form: (an integer (int or long), an integer
  (int or long), an integer (int or long)))
  self-regressor from index col row slice
  argument: `''--selfreg %d %d %d''`

weighted_ls: (a pathlike object or string representing an existing
  file)
  weighted least squares
  argument: `''--wls %s''`
  mutually_exclusive: weight_file, weight_inv, weight_sqrt

fixed_fx_var: (a pathlike object or string representing an existing
  file)
  for fixed effects analysis
  argument: `''--yffxvar %s''`

fixed_fx_dof: (an integer (int or long))
  dof for fixed effects analysis
  argument: `''--ffxdof %d''`
  mutually_exclusive: fixed_fx_dof_file

fixed_fx_dof_file: (a pathlike object or string representing a file)
  text file with dof for fixed effects analysis

(continues on next page)
argument: `--ffxdofdat %d`
mutually_exclusive: fixed_fx_dof

weight_file: (a pathlike object or string representing an existing file)
weight for each input at each voxel
mutually_exclusive: weighted_ls

weight_inv: (a boolean)
invert weights
argument: `--w-inv`
mutually_exclusive: weighted_ls

weight_sqrt: (a boolean)
sqrt of weights
argument: `--w-sqrt`
mutually_exclusive: weighted_ls

fwhm: (a floating point number >= 0.0)
smooth input by fwhm
argument: `--fwhm %f`

var_fwhm: (a floating point number >= 0.0)
smooth variance by fwhm
argument: `--var-fwhm %f`

no_mask_smooth: (a boolean)
do not mask when smoothing
argument: `--no-mask-smooth`

no_est_fwhm: (a boolean)
turn off FWHM output estimation
argument: `--no-est-fwhm`

mask_file: (a pathlike object or string representing an existing file)
binary mask
argument: `--mask %s`

label_file: (a pathlike object or string representing an existing file)
use label as mask, surfaces only
argument: `--label %s`
mutually_exclusive: cortex

cortex: (a boolean)
use subjects ?h.cortex.label as label
argument: `--cortex`
mutually_exclusive: label_file

invert_mask: (a boolean)
invert mask
argument: `--mask-inv`

prune: (a boolean)
remove voxels that do not have a non-zero value at each frame (def)
argument: `--prune`

no_prune: (a boolean)
do not prune
argument: `--no-prune`
mutually_exclusive: prune_thresh

prune_thresh: (a float)
prune threshold. Default is FLT_MIN
argument: `--prune_thr %f`
mutually_exclusive: noprunenest

compute_log_y: (a boolean)
compute natural log of y prior to analysis
argument: `--logy`

save_estimate: (a boolean)
save signal estimate (yhat)
argument: `--yhat-save`

save residual: (a boolean)
save residual error (eres)
argument: `--eres-save`

save_res_corr_mtx: (a boolean)
save residual error spatial correlation matrix (eres.scm). Big!
argument: `--eres-scm`

surf: (a boolean)
analysis is on a surface mesh
argument: `--surf %s %s %s`
requires: subject_id, hemi

subject_id: (a unicode string)
subject id for surface geometry

hemi: ('lh' or 'rh')
surface hemisphere

surf_geo: (a unicode string, nipype default value: white)
surface geometry name (e.g. white, pial)

simulation: (a tuple of the form: ('perm' or 'mc-full' or 'mc-z', an
integer (int or long), a float, a unicode string))
nulltype nsim thresh csdbasename
argument: `--sim %s %d %f %s`

sim_sign: ('abs' or 'pos' or 'neg')
abs, pos, or neg
argument: `--sim-sign %s`

uniform: (a tuple of the form: (a float, a float))
use uniform distribution instead of gaussian
argument: `--uniform %f %f`

pca: (a boolean)
perform pca/svd analysis on residual
argument: `--pca`

calc_AR1: (a boolean)
compute and save temporal AR1 of residual
argument: `--tar1`

save_cond: (a boolean)
flag to save design matrix condition at each voxel
argument: `--save-cond`

vox_dump: (a tuple of the form: (an integer (int or long), an integer
(int or long), an integer (int or long)))
dump voxel GLM and exit
argument: `--voxdump %d %d %d`

seed: (an integer (int or long))
used for synthesizing noise
argument: `--seed %d`

synth: (a boolean)
replace input with gaussian
argument: `--synth`

resynth_test: (an integer (int or long))
test GLM by resynthesis
argument: `--resynthtest %d`

profile: (an integer (int or long))
niters : test speed
argument: `--profile %d`

force_perm: (a boolean)
force permutation test, even when design matrix is not orthog
argument: `--perm-force`

diag: (an integer (int or long))
Gdiag_no : set diagnostic level
   argument: `--diag %d`

diag_cluster: (a boolean)
   save sig volume and exit from first sim loop
   argument: `--diag-cluster`

debug: (a boolean)
   turn on debugging
   argument: `--debug`

check_opts: (a boolean)
   don't run anything, just check options and exit
   argument: `--checkopts`

allow_repeated_subjects: (a boolean)
   allow subject names to repeat in the fsgd file (must appear before
   --fsgd
   argument: `--allowsubjrep`

allow_ill_cond: (a boolean)
   allow ill-conditioned design matrices
   argument: `--illcond`

sim_done_file: (a pathlike object or string representing a file)
   create file when simulation finished
   argument: `--sim-done %s`

subjects_dir: (a pathlike object or string representing an existing
directory)
   subjects directory

args: (a unicode string)
   Additional parameters to the command
   argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value
   of class 'str' and with values which are a bytes or None or a
   value of class 'str', nipype default value: {})  
   Environment variables

Outputs:

- glm_dir: (a pathlike object or string representing an existing
directory)
  output directory

- beta_file: (a pathlike object or string representing an existing
  file)
  map of regression coefficients

- error_file: (a pathlike object or string representing a file)
  map of residual error

- error_var_file: (a pathlike object or string representing a file)
  map of residual error variance

- error_stddev_file: (a pathlike object or string representing a file)
  map of residual error standard deviation

- estimate_file: (a pathlike object or string representing a file)
  map of the estimated Y values

- mask_file: (a pathlike object or string representing a file)
  map of the mask used in the analysis

- fwhm_file: (a pathlike object or string representing a file)
  text file with estimated smoothness

- dof_file: (a pathlike object or string representing a file)
  text file with effective degrees-of-freedom for the analysis

- gamma_file: (a list of items which are any value)
  map of contrast of regression coefficients

- gamma_var_file: (a list of items which are any value)
map of regression contrast variance

sig_file: (a list of items which are any value)
map of F-test significance (in -log10p)

ftest_file: (a list of items which are any value)
map of test statistic values

spatial_eigenvectors: (a pathlike object or string representing a file)
map of spatial eigenvectors from residual PCA

frame_eigenvectors: (a pathlike object or string representing a file)
matrix of frame eigenvectors from residual PCA

singular_values: (a pathlike object or string representing a file)
matrix singular values from residual PCA

svd_stats_file: (a pathlike object or string representing a file)
text file summarizing the residual PCA

## 62.2.11 SegStats

Link to code

Wraps the executable command `mri_segstats`.

Use FreeSurfer `mri_segstats` for ROI analysis

### Examples

```python
>>> import nipype.interfaces.freesurfer as fs

>>> ss = fs.SegStats()

>>> ss.inputs.annot = ('PWS04', 'lh', 'aparc')

>>> ss.inputs.in_file = 'functional.nii'

>>> ss.inputs.subjects_dir = '.

>>> ss.inputs.avgwf_txt_file = 'avgwf.txt'

>>> ss.inputs.summary_file = 'summary.stats'

>>> ss.cmdline
'mri_segstats --annot PWS04 lh aparc --avgwf ./avgwf.txt --i functional.nii --sum
˓→./summary.stats'
```

### Inputs:

**[Mandatory]**
segmentation_file: (a pathlike object or string representing an existing file)
segmentation volume path
argument: `--seg %s`
mutable_exclusive: segmentation_file, annot, surf_label

annot: (a tuple of the form: (a unicode string, 'lh' or 'rh', a unicode string))
subject hemi parc : use surface parcellation
argument: `--annot %s %s %s`
mutable_exclusive: segmentation_file, annot, surf_label

surf_label: (a tuple of the form: (a unicode string, 'lh' or 'rh', a unicode string))
subject hemi label : use surface label
argument: `--slabel %s %s %s`
mutable_exclusive: segmentation_file, annot, surf_label

**[Optional]**
summary_file: (a pathlike object or string representing a file)
Segmentation stats summary table file
  argument: `'--sum %s'`, position: -1

partial_volume_file: (a pathlike object or string representing an existing file)
  Compensate for partial voluming
  argument: `'--pv %s'`

in_file: (a pathlike object or string representing an existing file)
  Use the segmentation to report stats on this volume
  argument: `'--i %s'`

frame: (an integer (int or long))
  Report stats on nth frame of input volume
  argument: `'--frame %d'`

multiply: (a float)
  multiply input by val
  argument: `'--mul %f'`

calc_snr: (a boolean)
  save mean/std as extra column in output table
  argument: `'--snr'`

calc_power: ('sqr' or 'sqrt')
  Compute either the sqr or the sqrt of the input
  argument: `'--%s'`

color_table_file: (a pathlike object or string representing an existing file)
  color table file with seg id names
  argument: `'--ctab %s'`

mutually_exclusive: color_table_file, default_color_table, gca_color_table

default_color_table: (a boolean)
  use $FREESURFER_HOME/FreeSurferColorLUT.txt
  argument: `'--ctab-default'`

mutually_exclusive: color_table_file, default_color_table, gca_color_table

gca_color_table: (a pathlike object or string representing an existing file)
  get color table from GCA (CMA)
  argument: `'--ctab-gca %s'`

mutually_exclusive: color_table_file, default_color_table, gca_color_table

segment_id: (a list of items which are any value)
  Manually specify segmentation ids
  argument: `'--id %s...'`

exclude_id: (an integer (int or long))
  Exclude seg id from report
  argument: `'--excludeid %d'`

exclude_ctx_gm_wm: (a boolean)
  exclude cortical gray and white matter
  argument: `'--excl-ctxgmwm'`

wm_vol_from_surf: (a boolean)
  Compute wm volume from surf
  argument: `'--surf-wm-vol'`

cortex_vol_from_surf: (a boolean)
  Compute cortex volume from surf
  argument: `'--surf-ctx-vol'`

non_empty_only: (a boolean)
  Only report nonempty segmentations
  argument: `'--nonempty'`

empty: (a boolean)
Report on segmentations listed in the color table
argument: `--empty`

mask_file: (a pathlike object or string representing an existing file)
Mask volume (same size as seg
argument: `--mask %s`

mask_thresh: (a float)
Binarize mask with this threshold <0.5>
argument: `--maskthresh %f`

mask_sign: ('abs' or 'pos' or 'neg' or `--masksign %s')
Sign for mask threshold: pos, neg, or abs

mask_frame: (an integer (int or long))
Mask with this (0 based) frame of the mask volume
requires: mask_file

mask_invert: (a boolean)
Invert binarized mask volume
argument: `--maskinvert`

mask_erode: (an integer (int or long))
Erode mask by some amount
argument: `--maskerode %d`

brain_vol: ('brain-vol-from-seg' or 'brainmask')
Compute brain volume either with `brainmask` or `brain-vol-from-seg`
argument: `--%s`

brainmask_file: (a pathlike object or string representing an existing file)
Load brain mask and compute the volume of the brain as the non-zero voxels in this volume
argument: `--brainmask %s`

etiv: (a boolean)
Compute ICV from talairach transform
argument: `--etiv`

etiv_only: ('etiv' or 'old-etiv' or `--%s-only')
Compute etiv and exit. Use `etiv` or `old-etiv`

avgwf_txt_file: (a boolean or a pathlike object or string representing a file)
Save average waveform into file (bool or filename)
argument: `--avgwf %s`

avgwf_file: (a boolean or a pathlike object or string representing a file)
Save as binary volume (bool or filename)
argument: `--avgwfvol %s`

sf_avg_file: (a boolean or a pathlike object or string representing a file)
Save mean across space and time
argument: `--sfavg %s`

vox: (a list of items which are an integer (int or long))
Replace seg with all 0s except at C R S (three int inputs)
argument: `--vox %s`

supratent: (a boolean)
Undocumented input flag
argument: `--supratent`

subcort_gm: (a boolean)
Compute volume of subcortical gray matter
argument: `--subcortgray`

total_gray: (a boolean)
Compute volume of total gray matter
argument: `--totalgray`

euler: (a boolean)
Write out number of defect holes in orig.nofix based on the euler number
argument: `--euler`

in_intensity: (a pathlike object or string representing a file)
Undocumented input norm.mgz file
argument: `--in %s --in-intensity-name %s`

intensity_units: ('MR')
Intensity units
argument: `--in-intensity-units %s`
requires: in_intensity

subjects_dir: (a pathlike object or string representing an existing directory)
subjects directory

args: (a unicode string)
Additional parameters to the command
argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

summary_file: (a pathlike object or string representing an existing file)
Segmentation summary statistics table

avgwf_txt_file: (a pathlike object or string representing a file)
Text file with functional statistics averaged over segs

avgwf_file: (a pathlike object or string representing a file)
Volume with functional statistics averaged over segs

sf_avg_file: (a pathlike object or string representing a file)
Text file with func statistics averaged over segs and framss

62.2.12 SegStatsReconAll

Link to code
Wraps the executable command mri_segstats.
This class inherits SegStats and modifies it for use in a recon-all workflow. This implementation mandates implicit inputs that SegStats. To ensure backwards compatibility of SegStats, this class was created.

Examples

```python
>>> from nipype.interfaces.freesurfer import SegStatsReconAll
>>> segstatsreconall = SegStatsReconAll()
>>> segstatsreconall.inputs.annot = ('PWS04', 'lh', 'aparc')
>>> segstatsreconall.inputs.avgwf_txt_file = 'avgwf.txt'
>>> segstatsreconall.inputs.summary_file = 'summary.stats'
>>> segstatsreconall.inputs.subject_id = '10335'
>>> segstatsreconall.inputs.ribbon = 'wm.mgz'
>>> segstatsreconall.inputs.transform = 'trans.mat'
>>> segstatsreconall.inputs.presurf_seg = 'wm.mgz'
>>> segstatsreconall.inputs.lh_orig_nofix = 'lh.pial'
>>> segstatsreconall.inputs.rh_orig_nofix = 'lh.pial'
```
>>> segstatsreconall.inputs.lh_pial = 'lh.pial'
>>> segstatsreconall.inputs.rh_pial = 'lh.pial'
>>> segstatsreconall.inputs.lh_white = 'lh.pial'
>>> segstatsreconall.inputs.rh_white = 'lh.pial'
>>> segstatsreconall.inputs.empty = True
>>> segstatsreconall.inputs.brain_vol = 'brain-vol-from-seg'
>>> segstatsreconall.inputs.exclude_ctx_gm_wm = True
>>> segstatsreconall.inputs.supratent = True
>>> segstatsreconall.inputs.subcort_gm = True
>>> segstatsreconall.inputs.etiv = True
>>> segstatsreconall.inputs.wm_vol_from_surf = True
>>> segstatsreconall.inputs.cortex_vol_from_surf = True
>>> segstatsreconall.inputs.total_gray = True
>>> segstatsreconall.inputs.euler = True
>>> segstatsreconall.inputs.exclude_id = 0
>>> segstatsreconall.cmdline
'mri_segstats --annot PWS04 lh aparc --avgwf ./avgwf.txt --brain-vol-from-seg --surf-ctx-vol --empty --etiv --euler --excl-ctxgmmw --excludeid 0 --subcortgray --subject 10335 --supratent --totalgray --surf-wm-vol --sum ./summary.stats'

Inputs:

[Optional]
subject_id: (a string, nipype default value: subject_id)
    Subject id being processed
    argument: `--subject %s`
ribbon: (a pathlike object or string representing an existing file)
    Input file mri/ribbon.mgz
transform: (a pathlike object or string representing an existing file)
    Input transform file
lh_orig_nofix: (a pathlike object or string representing an existing file)
    Input lh.orig.nofix
rh_orig_nofix: (a pathlike object or string representing an existing file)
    Input rh.orig.nofix
lh_white: (a pathlike object or string representing an existing file)
    Input file must be <subject_id>/surf/lh.white
rh_white: (a pathlike object or string representing an existing file)
    Input file must be <subject_id>/surf/rh.white
lh_pial: (a pathlike object or string representing an existing file)
    Input file must be <subject_id>/surf/lh.pial
rh_pial: (a pathlike object or string representing an existing file)
    Input file must be <subject_id>/surf/rh.pial
segmentation_file: (a pathlike object or string representing an existing file)
    segmentation volume path
    argument: `--seg %s`
    mutually_exclusive: segmentation_file, annot, surf_label
annot: (a tuple of the form: (a unicode string, 'lh' or 'rh', a unicode string))
    subject hemi parc : use surface parcellation
    argument: `--annot %s %s %s`
    mutually_exclusive: segmentation_file, annot, surf_label
surf_label: (a tuple of the form: (a unicode string, 'lh' or 'rh', a unicode string))
subject hemi label : use surface label
argument: `--slab %s %s %s`
mutually_exclusive: segmentation_file, annot, surf_label

[Optional]
presurf_seg: (a pathlike object or string representing an existing file)
Input segmentation volume
aseg: (a pathlike object or string representing an existing file)
Mandatory implicit input in 5.3
copy_inputs: (a boolean)
If running as a node, set this to True otherwise, this will copy the implicit inputs to the node directory.
summary_file: (a pathlike object or string representing a file)
Segmentation stats summary table file
argument: `--sum %s`, position: -1
partial_volume_file: (a pathlike object or string representing an existing file)
Compensate for partial voluming
argument: `--pv %s`
in_file: (a pathlike object or string representing an existing file)
Use the segmentation to report stats on this volume
argument: `--i %s`
frame: (an integer (int or long))
Report stats on nth frame of input volume
argument: `--frame %d`
multiply: (a float)
multiply input by val
argument: `--mul %f`
calc_snr: (a boolean)
save mean/std as extra column in output table
argument: `--snr`
calc_power: ('sqr' or 'sqrt')
Compute either the sqr or the sqrt of the input
argument: `--%s`
color_table_file: (a pathlike object or string representing an existing file)
color table file with seg id names
argument: `--ctab %s`
mutually_exclusive: color_table_file, default_color_table, gca_color_table
default_color_table: (a boolean)
use $FREESURFER_HOME/FreeSurferColorLUT.txt
argument: `--ctab-default`
mutually_exclusive: color_table_file, default_color_table, gca_color_table
gca_color_table: (a pathlike object or string representing an existing file)
get color table from GCA (CMA)
argument: `--ctab-gca %s`
mutually_exclusive: color_table_file, default_color_table, gca_color_table
segment_id: (a list of items which are any value)
Manually specify segmentation ids
argument: `--id %s...`
exclude_id: (an integer (int or long))
Exclude seg id from report

(continues on previous page)
argument: `--excludeid %d`

exclude_ctx_gm_wm: (a boolean)
exclude cortical gray and white matter
argument: `--excl-ctxgmwm`

wm_vol_from_surf: (a boolean)
Compute wm volume from surf
argument: `--surf-wm-vol`

cortex_vol_from_surf: (a boolean)
Compute cortex volume from surf
argument: `--surf-ctx-vol`

non_empty_only: (a boolean)
Only report nonempty segmentations
argument: `--nonempty`

empty: (a boolean)
Report on segmentations listed in the color table
argument: `--empty`

mask_file: (a pathlike object or string representing an existing file)
Mask volume (same size as seg
argument: `--mask %s`

mask_thresh: (a float)
binarize mask with this threshold <0.5>
argument: `--maskthresh %f`

mask_sign: ('abs' or 'pos' or 'neg' or `--masksign %s'
Sign for mask threshold: pos, neg, or abs

mask_frame: (an integer (int or long))
Mask with this (0 based) frame of the mask volume
requires: mask_file

mask_invert: (a boolean)
Invert binarized mask volume
argument: `--maskinvert`

mask_erode: (an integer (int or long))
Erode mask by some amount
argument: `--maskerode %d`

brain_vol: ('brain-vol-from-seg' or 'brainmask')
Compute brain volume either with `brainmask` or `brain-vol-from-seg`
argument: `--%s`

brainmask_file: (a pathlike object or string representing an existing file)
Load brain mask and compute the volume of the brain as the non-zero voxels in this volume
argument: `--brainmask %s`

etiv: (a boolean)
Compute ICV from talairach transform
argument: `--etiv`

etiv_only: ('etiv' or 'old-etiv' or `--%s-only')
Compute etiv and exit. Use `etiv` or `old-etiv`

avgwf_txt_file: (a boolean or a pathlike object or string representing a file)
Save average waveform into file (bool or filename)
argument: `--avgwf %s`

avgwf_file: (a boolean or a pathlike object or string representing a file)
Save as binary volume (bool or filename)
argument: `--avgwfvol %s`

sf_avg_file: (a boolean or a pathlike object or string representing a...
file)
    Save mean across space and time
    argument: `--sfavg %s`

vox: (a list of items which are an integer (int or long))
    Replace seg with all Os except at C R S (three int inputs)
    argument: `--vox %s`

supratent: (a boolean)
    Undocumented input flag
    argument: `--supratent`

subcort_gm: (a boolean)
    Compute volume of subcortical gray matter
    argument: `--subcortgray`

total_gray: (a boolean)
    Compute volume of total gray matter
    argument: `--totalgray`

euler: (a boolean)
    Write out number of defect holes in orig.nofix based on the euler
    number
    argument: `--euler`

in_intensity: (a pathlike object or string representing a file)
    Undocumented input norm.mgz file
    argument: `--in %s --in-intensity-name %s`

intensity_units: (`MR`)  
    Intensity units
    argument: `--in-intensity-units %s`
    requires: in_intensity

subjects_dir: (a pathlike object or string representing an existing
    directory)
    subjects directory

args: (a unicode string)
    Additional parameters to the command
    argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})  
    Environment variables

Outputs:

summary_file: (a pathlike object or string representing an existing
    file)
    Segmentation summary statistics table

avgwf_txt_file: (a pathlike object or string representing a file)
    Text file with functional statistics averaged over segs

avgwf_file: (a pathlike object or string representing a file)
    Volume with functional statistics averaged over segs

sf_avg_file: (a pathlike object or string representing a file)
    Text file with func statistics averaged over segs and framss

62.2.13 SphericalAverage

Link to code
Wraps the executable command mris_spherical_average.
This program will add a template into an average surface.
Examples

```python
>>> from nipype.interfaces.freesurfer import SphericalAverage

>>> sphericalavg = SphericalAverage()
>>> sphericalavg.inputs.out_file = 'test.out'

>>> sphericalavg.inputs.in_average = '.

>>> sphericalavg.inputs.in_surf = 'lh.pial'

>>> sphericalavg.inputs.hemisphere = 'lh'

>>> sphericalavg.inputs.fname = 'lh.entorhinal'

>>> sphericalavg.inputs.which = 'label'

>>> sphericalavg.inputs.subject_id = '10335'

>>> sphericalavg.inputs.erode = 2

>>> sphericalavg.inputs.threshold = 5

>>> sphericalavg.cmdline
'mris_spherical_average -erode 2 -o 10335 -t 5.0 label lh.entorhinal lh pial .
→test.out'
```

Inputs:

```
[Mandatory]

in_surf: (a pathlike object or string representing an existing file)
    Input surface file
    argument: `%%s`, position: -3

hemisphere: ({'lh' or 'rh'})
    Input hemisphere
    argument: `%%s`, position: -4

fname: (a string)
    Filename from the average subject directory.
    Example: to use rh.entorhinal.label as the input label
    filename, set fname to 'rh.entorhinal' and which to '
    label'. The program will then search for
    '{in_average}/label/rh.entorhinal.label'
    argument: `%%s`, position: -5

which: ({'coords' or 'label' or 'vals' or 'curv' or 'area'})
    No documentation
    argument: `%%s`, position: -6

subject_id: (a string)
    Output subject id
    argument: `-%o %s`

[Optional]

out_file: (a pathlike object or string representing a file)
    Output filename
    argument: `%%s`, position: -1

in_average: (a pathlike object or string representing an existing
directory)
    Average subject
    argument: `%%s`, position: -2

erode: (an integer (int or long))
    Undocumented
    argument: `-%e erode %d`

in_orig: (a pathlike object or string representing an existing file)
    Original surface filename
    argument: `-%o orig %s`

threshold: (a float)
    Undocumented
    argument: `-%t %.1f`

subjects_dir: (a pathlike object or string representing an existing
continues on next page)
directory)
subjects directory
args: (a unicode string)
    Additional parameters to the command
    argument: ``\%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})
    Environment variables

Outputs:

out_file: (a pathlike object or string representing a file)
    Output label

62.3  interfaces.freesurfer.preprocess

62.3.1  ApplyVolTransform

Link to code
Wraps the executable command mri_vol2vol.
Use FreeSurfer mri_vol2vol to apply a transform.

Examples

```python
>>> from nipype.interfaces.freesurfer import ApplyVolTransform
>>> applyreg = ApplyVolTransform()
>>> applyreg.inputs.source_file = 'structural.nii'
>>> applyreg.inputs.reg_file = 'register.dat'
>>> applyreg.inputs.transformed_file = 'struct_warped.nii'
>>> applyreg.inputs.fs_target = True
>>> applyreg.cmdline
'mri_vol2vol --fstarg --reg register.dat --mov structural.nii --o struct_warped.nii'
```

Inputs:

[Mandatory]
source_file: (a pathlike object or string representing an existing file)
    Input volume you wish to transform
    argument: `\--mov %s`'
target_file: (a pathlike object or string representing an existing file)
    Output template volume
    argument: `\--targ %s`
    mutually_exclusive: target_file, tal, fs_target
tal: (a boolean)
    map to a sub FOV of MNI305 (with \--reg only)
    argument: `\--tal`
    mutually_exclusive: target_file, tal, fs_target
fs_target: (a boolean)
    use orig.mgz from subject in regfile as target
    argument: `\--fstarg`
    mutually_exclusive: target_file, tal, fs_target

(continues on next page)
<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>reg_file</strong></td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>tkRAS-to-tkRAS matrix</td>
<td>(tkregister2 format)</td>
</tr>
<tr>
<td>argument:<code>--reg %s</code></td>
<td>mutally_exclusive: reg_file, lta_file, lta_inv_file, fsl_reg_file,</td>
</tr>
<tr>
<td></td>
<td>xfm_reg_file, reg_header, mni_152_reg, subject</td>
</tr>
<tr>
<td><strong>lta_file</strong></td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>Linear Transform Array file</td>
<td>mutally_exclusive: reg_file, lta_file, lta_inv_file, fsl_reg_file,</td>
</tr>
<tr>
<td></td>
<td>xfm_reg_file, reg_header, mni_152_reg, subject</td>
</tr>
<tr>
<td><strong>lta_inv_file</strong></td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>LTA, invert</td>
<td>mutally_exclusive: reg_file, lta_file, lta_inv_file, fsl_reg_file,</td>
</tr>
<tr>
<td></td>
<td>xfm_reg_file, reg_header, mni_152_reg, subject</td>
</tr>
<tr>
<td><strong>fsl_reg_file</strong></td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>fslRAS-to-fslRAS matrix</td>
<td>(FSL format)</td>
</tr>
<tr>
<td>argument:<code>--fsl %s</code></td>
<td>mutally_exclusive: reg_file, lta_file, lta_inv_file, fsl_reg_file,</td>
</tr>
<tr>
<td></td>
<td>xfm_reg_file, reg_header, mni_152_reg, subject</td>
</tr>
<tr>
<td><strong>xfm_reg_file</strong></td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>ScannerRAS-to-ScannerRAS matrix</td>
<td>(MNI format)</td>
</tr>
<tr>
<td>argument:<code>--xfm %s</code></td>
<td>mutally_exclusive: reg_file, lta_file, lta_inv_file, fsl_reg_file,</td>
</tr>
<tr>
<td></td>
<td>xfm_reg_file, reg_header, mni_152_reg, subject</td>
</tr>
<tr>
<td><strong>reg_header</strong></td>
<td>(a boolean)</td>
</tr>
<tr>
<td>ScannerRAS-to-ScannerRAS matrix = identity</td>
<td>argument:<code>--regheader</code></td>
</tr>
<tr>
<td></td>
<td>mutally_exclusive: reg_file, lta_file, lta_inv_file, fsl_reg_file,</td>
</tr>
<tr>
<td></td>
<td>xfm_reg_file, reg_header, mni_152_reg, subject</td>
</tr>
<tr>
<td><strong>mni_152_reg</strong></td>
<td>(a boolean)</td>
</tr>
<tr>
<td>target MNI152 space</td>
<td>argument:<code>--regheader</code></td>
</tr>
<tr>
<td></td>
<td>mutally_exclusive: reg_file, lta_file, lta_inv_file, fsl_reg_file,</td>
</tr>
<tr>
<td></td>
<td>xfm_reg_file, reg_header, mni_152_reg, subject</td>
</tr>
<tr>
<td><strong>subject</strong></td>
<td>(a unicode string)</td>
</tr>
<tr>
<td>set matrix = identity and use subject for any templates</td>
<td>argument:<code>--s %s</code></td>
</tr>
<tr>
<td></td>
<td>mutally_exclusive: reg_file, lta_file, lta_inv_file, fsl_reg_file,</td>
</tr>
<tr>
<td></td>
<td>xfm_reg_file, reg_header, mni_152_reg, subject</td>
</tr>
</tbody>
</table>

**[Optional]**
| **transformed_file** | (a pathlike object or string representing a file)                        |
| Output volume        | argument:``--o %s``                                                      |
| **tal_resolution**   | (a float)                                                                 |
| Resolution to sample when using tal | argument:``--talres %.10f``                                         |
| **inverse**          | (a boolean)                                                              |
| sample from target to source | argument:``--inv``                                                      |
| **interp**           | ('trilin' or 'nearest' or 'cubic')                                       |
| Interpolation method | (<trilin> or nearest)                                                    |

(continues on next page)
argument: `'--interp %s'`

no_resample: (a boolean)
Do not resample; just change vox2ras matrix
argument: `'--no-resample'`

m3z_file: (a pathlike object or string representing a file)
This is the morph to be applied to the volume. Unless the morph is in mri/transforms (eg.: for talairach.m3z computed by reconall), you will need to specify the full path to this morph and use the --noDefM3zPath flag.
argument: `'--m3z %s'`

no_def_m3z_path: (a boolean)
To be used with the m3z flag. Instructs the code not to look for them3z morph in the default location (SUBJECTS_DIR/subj/mri/transforms), but instead just use the path indicated in --m3z.
argument: `'--noDefM3zPath'`
requires: m3z_file

invert_morph: (a boolean)
Compute and use the inverse of the non-linear morph to resample the input volume. To be used by --m3z.
argument: `'--inv-morph'`
requires: m3z_file

subjects_dir: (a pathlike object or string representing an existing directory)
subjects directory

args: (a unicode string)
Additional parameters to the command
argument: `'%s'`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

---

Transformed_file: (a pathlike object or string representing an existing file)
Path to output file if used normally

---

### 62.3.2 BBRegister

**Link to code**

Wraps the executable command `bbregister`.

Use FreeSurfer `bbregister` to register a volume to the Freesurfer anatomical.

This program performs within-subject, cross-modal registration using a boundary-based cost function. It is required that you have an anatomical scan of the subject that has already been recon-all-ed using freesurfer.

**Examples**

```python
>>> from nipype.interfaces.freesurfer import BBRegister
>>> bbreg = BBRegister(subject_id='me', source_file='structural.nii', init='header', contrast_type='t2')
>>> bbreg.cmdline
'bbregister --t2 --init-header --reg structural_bbreg_me.dat --mov structural.nii --s me'
```

**Inputs:**
subject_id: (a unicode string)
  freesurfer subject id
  argument: `--s %s`

source_file: (a pathlike object or string representing a file)
  source file to be registered
  argument: `--mov %s`

contrast_type: ('t1' or 't2' or 'bold' or 'dti')
  contrast type of image
  argument: `--%s`

[Optional]
init: ('coreg' or 'rr' or 'spm' or 'fsl' or 'header' or 'best')
  initialize registration with mri_coreg, spm, fsl, or header
  argument: `--init-%s`
  mutually_exclusive: init_reg_file
init_reg_file: (a pathlike object or string representing an existing file)
  existing registration file
  argument: `--init-reg %s`
  mutually_exclusive: init

intermediate_file: (a pathlike object or string representing an existing file)
  Intermediate image, e.g. in case of partial FOV
  argument: `--int %s`
reg_frame: (an integer (int or long))
  0-based frame index for 4D source file
  argument: `--frame %d`
  mutually_exclusive: reg_middle_frame
reg_middle_frame: (a boolean)
  Register middle frame of 4D source file
  argument: `--mid-frame`
  mutually_exclusive: reg_frame
out_reg_file: (a pathlike object or string representing a file)
  output registration file
  argument: `--reg %s`

spm_nifti: (a boolean)
  force use of nifti rather than analyze with SPM
  argument: `--spm-nii`
epi_mask: (a boolean)
  mask out B0 regions in stages 1 and 2
  argument: `--epi-mask`
dof: (6 or 9 or 12)
  number of transform degrees of freedom
  argument: `--%d`
fsldof: (an integer (int or long))
  degrees of freedom for initial registration (FSL)
  argument: `--fsl-dof %d`
out_fsl_file: (a boolean or a pathlike object or string representing a file)
  write the transformation matrix in FSL FLIRT format
  argument: `--fslmat %s`
out_lta_file: (a boolean or a pathlike object or string representing a file)
  write the transformation matrix in LTA format
  argument: `--lta %s`
registered_file: (a boolean or a pathlike object or string...
represents a file
output warped sourcefile either True or filename
argument: `--o %s`
init_cost_file: (a boolean or a pathlike object or string
representing a file)
output initial registration cost file
argument: `--initcost %s`
subjects_dir: (a pathlike object or string representing an existing
directory)
subjects directory
args: (a unicode string)
Additional parameters to the command
argument: `"%s"
envir: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_reg_file: (a pathlike object or string representing an existing
file)
Output registration file
out_fsl_file: (a pathlike object or string representing an existing
file)
Output FLIRT-style registration file
out_lta_file: (a pathlike object or string representing an existing
file)
Output LTA-style registration file
min_cost_file: (a pathlike object or string representing an existing
file)
Output registration minimum cost file
init_cost_file: (a pathlike object or string representing an existing
file)
Output initial registration cost file
registered_file: (a pathlike object or string representing an
existing file)
Registered and resampled source file

62.3.3 CALabel

Link to code
Wraps the executable command `mri_ca_label`.
For complete details, see the FS Documentation

Examples

```python
>>> from nipype.interfaces import freesurfer
>>> ca_label = freesurfer.CALabel()
>>> ca_label.inputs.in_file = 'norm.mgz'
>>> ca_label.inputs.out_file = 'out.mgz'
>>> ca_label.inputs.transform = 'trans.mat'
>>> ca_label.inputs.template = 'Template_6.nii'  # in practice use .gcs extension
>>> ca_label.cmdline
'mri_ca_label norm.mgz trans.mat Template_6.nii out.mgz'
```

Inputs:
Mandatory

- **in_file**: (a pathlike object or string representing an existing file)
  - Input volume for CALabel
  - argument: `'\$s'`, position: -4
- **out_file**: (a pathlike object or string representing a file)
  - Output file for CALabel
  - argument: `'\$s'`, position: -1
- **transform**: (a pathlike object or string representing an existing file)
  - Input transform for CALabel
  - argument: `'\$s'`, position: -3
- **template**: (a pathlike object or string representing an existing file)
  - Input template for CALabel
  - argument: `'\$s'`, position: -2

Optional

- **in_vol**: (a pathlike object or string representing an existing file)
  - Set input volume
  - argument: `'\-r \$s'`
- **intensities**: (a pathlike object or string representing an existing file)
  - Input label intensities file (used in longitudinal processing)
  - argument: `'\-r \$s'`
- **no_big_ventricles**: (a boolean)
  - No big ventricles
  - argument: `'\-nobigventricles'`
- **align**: (a boolean)
  - Align CALabel
  - argument: `'\-align'`
- **prior**: (a float)
  - Prior for CALabel
  - argument: `'\-prior %.1f'`
- **relabelunlikely**: (a tuple of the form: (an integer (int or long), a float))
  - Reclassify voxels at least some std devs from the mean using some size Gaussian window
  - argument: `'\-relabelunlikely %d %.1f'`
- **label**: (a pathlike object or string representing an existing file)
  - Undocumented flag. Autorecon3 uses `../label/{hemisphere}.cortex.label` as input file
  - argument: `'\-l %s'`
- **aseg**: (a pathlike object or string representing an existing file)
  - Undocumented flag. Autorecon3 uses `../mri/aseg.presurf.mgz` as input file
  - argument: `'\-aseg %s'`
- **num_threads**: (an integer (int or long))
  - Allows for specifying more threads
- **subjects_dir**: (a pathlike object or string representing an existing directory)
  - Subjects directory
- **args**: (a unicode string)
  - Additional parameters to the command
  - argument: `'\$s'`
- **environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: `{}')
  - Environment variables

Outputs:
62.3.4 CANormalize

Link to code

Wraps the executable command `mri_ca_normalize`. This program creates a normalized volume using the brain volume and an input gca file. For complete details, see the FS Documentation.

Examples

```python
>>> from nipype.interfaces import freesurfer
>>> ca_normalize = freesurfer.CANormalize()
>>> ca_normalize.inputs.in_file = "T1.mgz"
>>> ca_normalize.inputs.atlas = "atlas.nii.gz"  # in practice use .gca atlases
>>> ca_normalize.inputs.transform = "trans.mat"  # in practice use .lta transforms
>>> ca_normalize.cmdline
'mri_ca_normalize T1.mgz atlas.nii.gz trans.mat T1_norm.mgz'
```

Inputs:

- **in_file**: (a pathlike object or string representing an existing file)
  - The input file for CANormalize
  - argument: ``"%s"``, position: -4

- **atlas**: (a pathlike object or string representing an existing file)
  - The atlas file in gca format
  - argument: ``"%s"``, position: -3

- **transform**: (a pathlike object or string representing an existing file)
  - The transform file in lta format
  - argument: ``"%s"``, position: -2

- **out_file**: (a pathlike object or string representing a file)
  - The output file for CANormalize
  - argument: ``"%s"``, position: -1

- **mask**: (a pathlike object or string representing an existing file)
  - Specifies volume to use as mask
  - argument: ``"-mask %s"``

- **control_points**: (a pathlike object or string representing a file)
  - File name for the output control points
  - argument: ``"-c %s"``

- **long_file**: (a pathlike object or string representing a file)
  - undocumented flag used in longitudinal processing
  - argument: ``"-long %s"``

- **subjects_dir**: (a pathlike object or string representing an existing directory)
  - subjects directory

- **args**: (a unicode string)
  - Additional parameters to the command
  - argument: ``"%s"``

- **environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: `{}`)
  - Environment variables
Outputs:

<table>
<thead>
<tr>
<th>out_file: (a pathlike object or string representing a file)</th>
</tr>
</thead>
<tbody>
<tr>
<td>The output file for Normalize</td>
</tr>
<tr>
<td>control_points: (a pathlike object or string representing a file)</td>
</tr>
<tr>
<td>The output control points for Normalize</td>
</tr>
</tbody>
</table>

62.3.5 CARegister

Link to code

Wraps the executable command `mri_ca_register`.

Generates a multi-dimensional talairach transform from a gca file and talairach.lta file

For complete details, see the FS Documentation

Examples

```python
>>> from nipype.interfaces import freesurfer
>>> ca_register = freesurfer.CARegister()
>>> ca_register.inputs.in_file = "norm.mgz"
>>> ca_register.inputs.out_file = "talairach.m3z"
>>> ca_register.cmdline
'mri_ca_register norm.mgz talairach.m3z'
```

Inputs:

[Mandatory]

- in_file: (a pathlike object or string representing an existing file)
  - The input volume for CARegister
    - argument: ``%s``, position: -3

[Optional]

- out_file: (a pathlike object or string representing a file)
  - The output volume for CARegister
    - argument: ``%s``, position: -1

- template: (a pathlike object or string representing an existing file)
  - The template file in gca format
    - argument: ``%s``, position: -2

- mask: (a pathlike object or string representing an existing file)
  - Specifies volume to use as mask
    - argument: ``-mask %s``

- invert_and_save: (a boolean)
  - Invert and save the .m3z multi-dimensional talairach transform to x, y, and z .mgz files
    - argument: ``--invert-and-save``

- no_big_ventricles: (a boolean)
  - No big ventricles
    - argument: ``--nobigventricles``

- transform: (a pathlike object or string representing an existing file)
  - Specifies transform in lta format
    - argument: ```-T %s```

- align: (a string)
  - Specifies when to perform alignment
    - argument: ```-align-%s```

- levels: (an integer (int or long))
  - defines how many surrounding voxels will be used in interpolations,
    - default is 6
argument: `'--levels %d'`
A: (an integer (int or long))
  undocumented flag used in longitudinal processing
argument: `'--A %d'`
I_files: (a list of items which are a pathlike object or string
  representing a file)
  undocumented flag used in longitudinal processing
argument: `'--l %s'`
num_threads: (an integer (int or long))
  allows for specifying more threads
subjects_dir: (a pathlike object or string representing an existing
directory)
  subjects directory
args: (a unicode string)
  Additional parameters to the command
argument: `'--s'`
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
  Environment variables

Outputs:
out_file: (a pathlike object or string representing a file)
The output file for CARegister

62.3.6 ConcatenateLTA

Link to code
Wraps the executable command mri_concatenate_lta.
Concatenates two consecutive LTA transformations into one overall transformation
Out = LTA2*LTA1

Examples

```python
>>> from nipype.interfaces.freesurfer import ConcatenateLTA
>>> conc_lta = ConcatenateLTA()
>>> conc_lta.inputs.in_lta1 = 'lta1.lta'
>>> conc_lta.inputs.in_lta2 = 'lta2.lta'
>>> conc_lta.cmdline
'mri_concatenate_lta lta1.lta lta2.lta lta1_concat.lta'
```
You can use `identity.nofile` as the filename for in_lta2, e.g.:

```python
>>> conc_lta.inputs.in_lta2 = 'identity.nofile'
>>> conc_lta.inputs.invert_1 = True
>>> conc_lta.inputs.out_file = 'inv1.lta'
>>> conc_lta.cmdline
'mri_concatenate_lta --invert1 lta1.lta identity.nofile inv1.lta'
```
To create a RAS2RAS transform:

```python
>>> conc_lta.inputs.out_type = 'RAS2RAS'
>>> conc_lta.cmdline
'mri_concatenate_lta --invert1 --out_type 1 lta1.lta identity.nofile inv1.lta'
```
[Mandatory]
in_lta1: (a pathlike object or string representing an existing file)
    maps some src1 to dst1
    argument: ``%s`` , position: -3
in_lta2: (a pathlike object or string representing an existing file
    or 'identity.nofile')
    maps dst1(src2) to dst2
    argument: ``%s`` , position: -2

[Optional]
out_file: (a pathlike object or string representing a file)
    the combined LTA maps: src1 to dst2 = LTA2*LTA1
    argument: ``%s`` , position: -1
invert_1: (a boolean)
    invert in_lta1 before applying it
    argument: ``-invert1``
invert_2: (a boolean)
    invert in_lta2 before applying it
    argument: ``-invert2``
invert_out: (a boolean)
    invert output LTA
    argument: ``-invertout``
out_type: ('VOX2VOX' or 'RAS2RAS')
    set final LTA type
    argument: ``-out_type %d``
tal_source_file: (a pathlike object or string representing an
    existing file)
    if in_lta2 is talairach.xfm, specify source for talairach
    argument: ``-tal %s`` , position: -5
    requires: tal_template_file
tal_template_file: (a pathlike object or string representing an
    existing file)
    if in_lta2 is talairach.xfm, specify template for talairach
    argument: ``%s`` , position: -4
    requires: tal_source_file
subject: (a unicode string)
    set subject in output LTA
    argument: ``-subject %s``
subjects_dir: (a pathlike object or string representing an existing
    directory)
    subjects directory
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {}))
    Environment variables

Outputs:

out_file: (a pathlike object or string representing a file)
    the combined LTA maps: src1 to dst2 = LTA2*LTA1

62.3.7 DICOMConvert

Link to code
Wraps the executable command mri_convert.
use fs mri_convert to convert dicom files

**Examples**

```python
>>> from nipype.interfaces.freesurfer import DICOMConvert
>>> cvt = DICOMConvert()
>>> cvt.inputs.dicom_dir = 'dicomdir'
>>> cvt.inputs.file_mapping = [('nifti', '*.nii'), ('info', 'dicom*.txt'), ('dti', 'dti*.bv')]
```

**Inputs:**

[Mandatory]
- `dicom_dir`: (a pathlike object or string representing an existing directory)  
  - dicom directory from which to convert dicom files
- `base_output_dir`: (a pathlike object or string representing a directory)  
  - directory in which subject directories are created

[Optional]
- `subject_dir_template`: (a unicode string, nipype default value: S.%04d)  
  - template for subject directory name
- `subject_id`: (any value)  
  - subject identifier to insert into template
- `file_mapping`: (a list of items which are a tuple of the form: (a unicode string, a unicode string))  
  - defines the output fields of interface
- `out_type`: ('cor' or 'mgh' or 'mgz' or 'minc' or 'analyze' or 'analyze4d' or 'spm' or 'afni' or 'brik' or 'bfloat' or 'sdt' or 'outline' or 'otl' or 'gdf' or 'nifti' or 'nii' or 'niigz', nipype default value: niigz)  
  - defines the type of output file produced
- `dicom_info`: (a pathlike object or string representing an existing file)  
  - File containing summary information from mri_parse_sdcmdir
- `seq_list`: (a list of items which are a unicode string)  
  - list of pulse sequence names to be converted. requires: dicom_info
- `ignore_single_slice`: (a boolean)  
  - ignore volumes containing a single slice
    - requires: dicom_info
- `subjects_dir`: (a pathlike object or string representing an existing directory)  
  - subjects directory
- `args`: (a unicode string)  
  - Additional parameters to the command argument: ``%s``
- `environ`: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})  
  - Environment variables

**Outputs:**

None
62.3.8 EditWMwithAseg

Link to code

Wraps the executable command `mri_edit_wm_with_aseg`. Edits a wm file using a segmentation

Examples

```python
>>> from nipype.interfaces.freesurfer import EditWMwithAseg
>>> editwm = EditWMwithAseg()
>>> editwm.inputs.in_file = "T1.mgz"
>>> editwm.inputs.brain_file = "norm.mgz"
>>> editwm.inputs.seg_file = "aseg.mgz"
>>> editwm.inputs.out_file = "wm.asegedit.mgz"
>>> editwm.inputs.keep_in = True
>>> editwm.cmdline
'mri_edit_wm_with_aseg -keep-in T1.mgz norm.mgz aseg.mgz wm.asegedit.mgz'
```

Inputs:

|Mandatory|
in_file: (a pathlike object or string representing an existing file)
  - Input white matter segmentation file
    argument: `"%s"`, position: -4
brain_file: (a pathlike object or string representing an existing file)
  - Input brain/T1 file
    argument: `"%s"`, position: -3
seg_file: (a pathlike object or string representing an existing file)
  - Input presurf segmentation file
    argument: `"%s"`, position: -2
out_file: (a pathlike object or string representing a file)
  - File to be written as output
    argument: `"%s"`, position: -1

[Optional]
keep_in: (a boolean)
  - Keep edits as found in input volume
    argument: `"-keep-in"`
subjects_dir: (a pathlike object or string representing an existing directory)
  - subjects directory
args: (a unicode string)
  - Additional parameters to the command
    argument: `"%s"`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) (Environment variables)

Outputs:

out_file: (a pathlike object or string representing a file)
  - Output edited WM file

62.3.9 FitMSPrams

Link to code

Wraps the executable command `mri_ms_fitparms`. Estimate tissue parameters from a set of FLASH images.

**Examples**

```python
>>> from nipype.interfaces.freesurfer import FitMSParams
>>> msfit = FitMSParams()
>>> msfit.inputs.in_files = ['flash_05.mgz', 'flash_30.mgz']
>>> msfit.inputs.out_dir = 'flash_parameters'
>>> msfit.cmdline
'mri_ms_fitparms flash_05.mgz flash_30.mgz flash_parameters'
```

**Inputs:**

[Mandatory]
- `in_files`: (a list of items which are a pathlike object or string representing an existing file)
  - list of FLASH images (must be in mgh format)
  - argument: ````%s`````, position: -2

[Optional]
- `tr_list`: (a list of items which are an integer (int or long))
  - list of TRs of the input files (in msec)
- `te_list`: (a list of items which are a float)
  - list of TEs of the input files (in msec)
- `flip_list`: (a list of items which are an integer (int or long))
  - list of flip angles of the input files
- `xfm_list`: (a list of items which are a pathlike object or string representing an existing file)
  - list of transform files to apply to each FLASH image
- `out_dir`: (a pathlike object or string representing a directory)
  - directory to store output in
  - argument: ````%s`````, position: -1
- `subjects_dir`: (a pathlike object or string representing an existing directory)
  - subjects directory
- `args`: (a unicode string)
  - Additional parameters to the command
  - argument: ````%s````
- `environ`: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: { })
  - Environment variables

**Outputs:**

- `t1_image`: (a pathlike object or string representing an existing file)
  - image of estimated T1 relaxation values
- `pd_image`: (a pathlike object or string representing an existing file)
  - image of estimated proton density values
- `t2star_image`: (a pathlike object or string representing an existing file)
  - image of estimated T2* values

### 62.3.10 MNIBiasCorrection

Wraps the executable command `mri_nu_correct.mni`.
Wrapper for nu_correct, a program from the Montreal Neurological Institute (MNI) used for correcting intensity non-uniformity (ie, bias fields). You must have the MNI software installed on your system to run this. See [www.bic.mni.mcgill.ca/software/N3] for more info.
mri_nu_correct.mni uses float internally instead of uchar. It also rescales the output so that the global mean is the same as that of the input. These two changes are linked and can be turned off with --no-float

Examples

```python
>>> from nipype.interfaces.freesurfer import MNIBiasCorrection
>>> correct = MNIBiasCorrection()
>>> correct.inputs.in_file = "norm.mgz"
>>> correct.inputs.iterations = 6
>>> correct.inputs.protocol_iterations = 1000
>>> correct.inputs.distance = 50
>>> correct.cmdline
'mri_nu_correct.mni --distance 50 --i norm.mgz --n 6 --o norm_output.mgz --proto-\n→iters 1000'
```

References:


Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  input volume. Input can be any format accepted by mri_convert.
  argument: ``--i %s``

[Optional]
out_file: (a pathlike object or string representing a file)
  output volume. Output can be any format accepted by mri_convert. If
  the output format is COR, then the directory must exist.
  argument: ``--o %s``
iterations: (an integer (int or long), nipype default value: 4)
  Number of iterations to run nu_correct. Default is 4. This is the
  number of times that nu_correct is repeated (ie, using the output
  from the previous run as the input for the next). This is different
  than the --n flag to nu_correct.
  argument: ``--n %d``
protocol_iterations: (an integer (int or long))
  Passes Np as argument of the -iterations flag of nu_correct. This is
  different than the --n flag above. Default is not to pass nu_correct
  the -iterations flag.
  argument: ``--proto-iters %d``
distance: (an integer (int or long))
  N3 -distance option
  argument: ``--distance %d``
no_rescale: (a boolean)
  do not rescale so that global mean of output == input global mean
  argument: ``--no-rescale``
mask: (a pathlike object or string representing an existing file)
  brainmask volume. Input can be any format accepted by mri_convert.
  argument: ``--mask %s``
transform: (a pathlike object or string representing an existing file)
tal.xfm. Use mri_make_uchar instead of conforming argument: `--uchar %s`

stop: (a float)
Convergence threshold below which iteration stops (suggest 0.01 to 0.0001)
argument: `--stop %f`

shrink: (an integer (int or long))
Shrink parameter for finer sampling (default is 4)
argument: `--shrink %d`

subjects_dir: (a pathlike object or string representing an existing directory)
argument: `subjects directory`

args: (a unicode string)
Additional parameters to the command argument: `args`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
output volume

### 62.3.11 MRICovert

**Link to code**
Wraps the executable command mri_convert.
use fs mri_convert to manipulate files

**Note:** Adds niigz as an output type option

**Examples**

```python
>>> mc = MRICovert()
>>> mc.inputs.in_file = 'structural.nii'
>>> mc.inputs.out_file = 'outfile.mgz'
>>> mc.inputs.out_type = 'mgz'
>>> mc.cmdline
'mri_convert --out_type mgz --input_volume structural.nii --output_volume outfile.mgz'
```

**Inputs:**

[Mandatory]

in_file: (a pathlike object or string representing an existing file)
File to read/convert
argument: `--input_volume %s`, position: -2

[Optional]

read_only: (a boolean)
read the input volume
argument: `--read_only`

no_write: (a boolean)
do not write output
argument: `--no_write`

in_info: (a boolean)
display input info
argument: `--in_info`

out_info: (a boolean)
display output info
argument: `--out_info`

in_stats: (a boolean)
display input stats
argument: `--in_stats`

out_stats: (a boolean)
display output stats
argument: `--out_stats`

in_matrix: (a boolean)
display input matrix
argument: `--in_matrix`

out_matrix: (a boolean)
display output matrix
argument: `--out_matrix`

in_i_size: (an integer (int or long))
input i size
argument: `--in_i_size %d`

in_j_size: (an integer (int or long))
input j size
argument: `--in_j_size %d`

in_k_size: (an integer (int or long))
input k size
argument: `--in_k_size %d`

force_ras: (a boolean)
use default when orientation info absent
argument: `--force_ras_good`

in_i_dir: (a tuple of the form: (a float, a float, a float))
<R direction> <A direction> <S direction>
argument: `--in_i_direction %f %f %f`

in_j_dir: (a tuple of the form: (a float, a float, a float))
<R direction> <A direction> <S direction>
argument: `--in_j_direction %f %f %f`

in_k_dir: (a tuple of the form: (a float, a float, a float))
<R direction> <A direction> <S direction>
argument: `--in_k_direction %f %f %f`

in_orientation: ('LAI' or 'LIA' or 'ALI' or 'AIL' or 'ILA' or 'IAL'
or 'LAS' or 'LSA' or 'ALS' or 'SLA' or 'SLA' or 'SAL' or 'LPI' or
'LIP' or 'PLI' or 'PIL' or 'ILP' or 'IPL' or 'LPS' or 'LSP' or
'PLS' or 'PSL' or 'SPL' or 'RAI' or 'RIA' or 'ARI' or
'AIR' or 'IRA' or 'IAR' or 'RAS' or 'ARS' or 'ASR' or
'SRA' or 'SAR' or 'SRP' or 'RSP' or 'PSR' or 'PSR' or 'SRP' or
'PRS' or 'RPS' or 'SRA' or 'SAR' or 'SRP' or 'SAR' or
'SRA' or 'SAR' or 'SRP' or 'SAR' or
'SRA' or 'SAR' or 'SRP' or 'SAR')
specify the input orientation
argument: `--in_orientation %s`

in_center: (a list of at most 3 items which are a float)
<R coordinate> <A coordinate> <S coordinate>
argument: `--in_center %s`

sphinx: (a boolean)
change orientation info to sphinx
argument: `--sphinx`

out_i_count: (an integer (int or long))
some count ?? in i direction
argument: \``--out_i_count %d``

out_j_count: (an integer (int or long))
some count ?? in j direction
argument: \``--out_j_count %d``

out_k_count: (an integer (int or long))
some count ?? in k direction
argument: \``--out_k_count %d``

vox_size: (a tuple of the form: (a float, a float, a float))
<size_x> <size_y> <size_z> specify the size (mm) - useful for
upsampling or downsampling
argument: \``--voxsize %f %f %f``

out_i_size: (an integer (int or long))
output i size
argument: \``--out_i_size %d``

out_j_size: (an integer (int or long))
output j size
argument: \``--out_j_size %d``

out_k_size: (an integer (int or long))
output k size
argument: \``--out_k_size %d``

out_i_dir: (a tuple of the form: (a float, a float, a float))
<R direction> <A direction> <S direction>
argument: \``--out_i_direction %f %f %f``

out_j_dir: (a tuple of the form: (a float, a float, a float))
<R direction> <A direction> <S direction>
argument: \``--out_j_direction %f %f %f``

out_k_dir: (a tuple of the form: (a float, a float, a float))
<R direction> <A direction> <S direction>
argument: \``--out_k_direction %f %f %f``

out_orientation: ('LAI' or 'LIA' or 'ALI' or 'AIL' or 'ILA' or 'IAL'
or 'LAS' or 'LSA' or 'ALS' or 'ASL' or 'SLA' or 'SAL' or 'LPI' or
'LIP' or 'PLI' or 'PIL' or 'ILP' or 'IPL' or 'LSP' or 'LPS' or 'PSL'
or 'SPL' or 'SPL' or 'SPS' or 'RLP' or 'RPL' or 'RIP' or 'RIP' or 'PRI'
or 'ARI' or 'IRA' or 'ARI' or 'ARL' or 'SRL' or 'SRL' or 'RLS' or
'SAR' or 'SAR' or 'API' or 'API' or 'PRI' or 'PRI' or 'RIP' or 'RIP'
or 'IPS' or 'IPS' or 'PSR' or 'PSS' or 'PSS' or 'SPR' or 'SPR')
specify the output orientation
argument: \``--out_orientation %s``

out_center: (a tuple of the form: (a float, a float, a float))
<R coordinate> <A coordinate> <S coordinate>
argument: \``--out_center %f %f %f``

out_datatype: ('uchar' or 'short' or 'int' or 'float')
output data type <uchar|short|int|float>
argument: \``--out_data_type %s``

resample_type: ('interpolate' or 'weighted' or 'nearest' or 'sinc' or
'cubic')
<interpolate|weighted|nearest|sinc|cubic> (default is interpolate)
argument: \``--resample_type %s``

no_scale: (a boolean)
dont rescale values for COR
argument: \``--no_scale 1``

no_change: (a boolean)
don't change type of input to that of template
argument: \``--nochange``

tr: (an integer (int or long))
TR in msec
argument: `''-tr %d``
te: (an integer (int or long))
  TE in msec
  argument: `''-te %d``
ti: (an integer (int or long))
  TI in msec (note upper case flag)
  argument: `''-ti %d``
autoalign_matrix: (a pathlike object or string representing an
  existing file)
  text file with autoalign matrix
  argument: `''--autoalign %s``
unwarp_gradient: (a boolean)
  unwarp gradient nonlinearity
  argument: `''--unwarp_gradient_nonlinearity``
apply_transform: (a pathlike object or string representing an
  existing file)
  apply xfm file
  argument: `''--apply_transform %s``
apply_inv_transform: (a pathlike object or string representing an
  existing file)
  apply inverse transformation xfm file
  argument: `''--apply_inverse_transform %s``
deolve_transform: (a unicode string)
  subject id
  argument: `''--devolvexfm %s``
crop_center: (a tuple of the form: (an integer (int or long), an
  integer (int or long), an integer (int or long)))
  <x> <y> <z> crop to 256 around center (x, y, z)
  argument: `''--crop %d %d %d``
crop_size: (a tuple of the form: (an integer (int or long), an
  integer (int or long), an integer (int or long)))
  <dx> <dy> <dz> crop to size <dx, dy, dz>
  argument: `''--cropsize %d %d %d``
cut_ends: (an integer (int or long))
  remove ncut slices from the ends
  argument: `''--cutends %d``
slice_crop: (a tuple of the form: (an integer (int or long), an
  integer (int or long)))
  s_start s_end : keep slices s_start to s_end
  argument: `''--slice-crop %d %d``
slice_reverse: (a boolean)
  reverse order of slices, update vox2ras
  argument: `''--slice-reverse``
slice_bias: (a float)
  apply half-cosine bias field
  argument: `''--slice-bias %f``
fwhm: (a float)
  smooth input volume by fwhm mm
  argument: `''--fwhm %f``
in_type: ('cor' or 'mgh' or 'mgz' or 'minc' or 'analyze' or
  'analyze4d' or 'spm' or 'afni' or 'brik' or 'bshort' or 'bfloat'
  or 'sdt' or 'outline' or 'otl' or 'gdf' or 'nifti' or 'nii' or
  'niigz' or 'ge' or 'gelx' or 'lx' or 'ximg' or 'siemens' or
  'dicom' or 'siemens_dicom')
  input file type
  argument: `''--in_type %s``
out_type: ('cor' or 'mgh' or 'mgz' or 'minc' or 'analyze' or...
nipype Documentation, Release 1.2.1

(continued from previous page)

'analyze4d' or 'spm' or 'afni' or 'brik' or 'bshort' or 'bfloat'
or 'sdt' or 'outline' or 'otl' or 'gdf' or 'nifti' or 'nii' or 'niigz')
output file type
argument: ```--out_type %s```

ascii: (a boolean)
save output as ascii col>row>slice>frame
argument: ```--ascii```

reorder: (a tuple of the form: (an integer (int or long), an integer
(int or long), an integer (int or long)))
olddim1 olddim2 olddim3
argument: ```--reorder %d %d %d```

invert_contrast: (a float)
threshold for inversetg contrast
argument: ```--invert_contrast %f```

out_file: (a pathlike object or string representing a file)
output filename or True to generate one
argument: ```--output_volume %s```, position: -1

conform: (a boolean)
conform to 1mm voxel size in coronal slice direction with 256^3 or
more
argument: ```--conform```

conform_min: (a boolean)
conform to smallest size
argument: ```--conform_min```

conform_size: (a float)
conform to size_in_mm
argument: ```--conform_size %s```

cw256: (a boolean)
conform to dimensions of 256^3
argument: ```--cw256```

parse_only: (a boolean)
pars input only
argument: ```--parse_only```

subject_name: (a unicode string)
subject name ???
argument: ```--subject_name %s```

reslice_like: (a pathlike object or string representing an existing
file)
reslice output to match file
argument: ```--reslice_like %s```

template_type: ('cor' or 'mgh' or 'mgz' or 'minc' or 'analyze' or
'analyze4d' or 'spm' or 'afni' or 'brik' or 'bshort' or 'bfloat'
or 'sdt' or 'outline' or 'otl' or 'gdf' or 'nifti' or 'nii' or
'niigz')
template file type
argument: ```--template_type %s```

split: (a boolean)
split output frames into separate output files.
argument: ```--split```

frame: (an integer (int or long))
keep only 0-based frame number
argument: ```--frame %d```

midframe: (a boolean)
keep only the middle frame
argument: ```--mid-frame```

(continues on next page)
skip_n: (an integer (int or long))
  skip the first n frames
  argument: "--nskip %d"
drop_n: (an integer (int or long))
  drop the last n frames
  argument: "--ndrop %d"
frame_subsample: (a tuple of the form: (an integer (int or long), an integer (int or long), an integer (int or long)))
  start delta end : frame subsampling (end = -1 for end)
  argument: "--fsubsample %d %d %d"
in_scale: (a float)
  input intensity scale factor
  argument: "--scale %f"
out_scale: (a float)
  output intensity scale factor
  argument: "--out-scale %d"
in_like: (a pathlike object or string representing an existing file)
  input looks like
  argument: "--in_like %s"
fill_parcellation: (a boolean)
  fill parcellation
  argument: "--fill_parcellation"
smooth_parcellation: (a boolean)
  smooth parcellation
  argument: "--smooth_parcellation"
zero_outlines: (a boolean)
  zero outlines
  argument: "--zero_outlines"
color_file: (a pathlike object or string representing an existing file)
  color file
  argument: "--color_file %s"
no_translate: (a boolean)
  argument: "--no_translate"
status_file: (a pathlike object or string representing a file)
  status file for DICOM conversion
  argument: "--status %s"
sdcmlist: (a pathlike object or string representing an existing file)
  list of DICOM files for conversion
  argument: "--sdcmlist %s"
template_info: (a boolean)
  dump info about template
  argument: "--template_info"
crop_gdf: (a boolean)
  apply GDF cropping
  argument: "--crop_gdf"
zero_ge_z_offset: (a boolean)
  zero ge z offset ???
  argument: "--zero_ge_z_offset"
subjects_dir: (a pathlike object or string representing an existing directory)
  subjects directory
args: (a unicode string)
  Additional parameters to the command
  argument: "%s"
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}))

Environment variables

Outputs:

out_file: (a list of items which are a pathlike object or string representing an existing file)
converted output file

62.3.12 MRIsCALabel

Link to code
Wraps the executable command mris_ca_label.
For a single subject, produces an annotation file, in which each cortical surface vertex is assigned a neuroanatomical label. This automatic procedure employs data from a previously-prepared atlas file. An atlas file is created from a training set, capturing region data manually drawn by neuroanatomists combined with statistics on variability correlated to geometric information derived from the cortical model (sulcus and curvature). Besides the atlases provided with FreeSurfer, new ones can be prepared using mris_ca_train).

Examples

```python
>>> from nipype.interfaces import freesurfer
>>> ca_label = freesurfer.MRIsCALabel()
>>> ca_label.inputs.subject_id = "test"
>>> ca_label.inputs.hemisphere = "lh"
>>> ca_label.inputs.canonsurf = "lh.pial"
>>> ca_label.inputs.curv = "lh.pial"
>>> ca_label.inputs.sulc = "lh.pial"
>>> ca_label.inputs.classifier = "im1.nii" # in practice, use .gcs extension
>>> ca_label.inputs.smoothwm = "lh.pial"
>>> ca_label.cmdline
'mris_ca_label test lh lh.pial im1.nii lh.aparc.annot'
```

Inputs:

[Mandatory]
subject_id: (a string, nipype default value: subject_id)
  Subject name or ID
  argument: `--subject_id`, position: -5
hemisphere: ('lh' or 'rh')
  Hemisphere ('lh' or 'rh')
  argument: `--hemisphere`, position: -4
canonsurf: (a pathlike object or string representing an existing file)
  Input canonical surface file
  argument: `--canonsurf`, position: -3
classifier: (a pathlike object or string representing an existing file)
  Classifier array input file
  argument: `--classifier`, position: -2
smoothwm: (a pathlike object or string representing an existing file)
  implicit input (hemisphere).smoothwm
curv: (a pathlike object or string representing an existing file)
  implicit input (hemisphere).curv
sulc: (a pathlike object or string representing an existing file)
    implicit input (hemisphere).sulc

[Optional]
out_file: (a pathlike object or string representing a file)
    Annotated surface output file
    argument: '\%s', position: -1
label: (a pathlike object or string representing an existing file)
    Undocumented flag. Autorecon3 uses ../~label/{hemisphere}.cortex.label as input file
    argument: '-l %s'
aseg: (a pathlike object or string representing an existing file)
    Undocumented flag. Autorecon3 uses ../mri/aseg.presurf.mgz as input file
    argument: '-aseg %s'
seed: (an integer (int or long))
    argument: '-seed %d'
copy_inputs: (a boolean)
    Copies implicit inputs to node directory and creates a temp subjects_directory. Use this when running as a node
num_threads: (an integer (int or long))
    allows for specifying more threads
subjects_dir: (a pathlike object or string representing an existing directory)
    subjects directory
args: (a unicode string)
    Additional parameters to the command
    argument: '\%s'
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
    Environment variables

Outputs:

out_file: (a pathlike object or string representing a file)
    Output volume from MRIsCALabel

62.3.13 Normalize

Link to code
Wraps the executable command mri_normalize.
Normalize the white-matter, optionally based on control points. The input volume is converted into a new volume where white matter image values all range around 110.

Examples

```
>>> from nipype.interfaces import freesurfer
>>> normalize = freesurfer.Normalize()
>>> normalize.inputs.in_file = "T1.mgz"
>>> normalize.inputs.gradient = 1
>>> normalize.cmdline
'mri_normalize -g 1 T1.mgz T1_norm.mgz'
```

Inputs:
[Mandatory]
in_file: (a pathlike object or string representing an existing file)
The input file for Normalize
argument: `"%s"`, position: -2

[Optional]
out_file: (a pathlike object or string representing a file)
The output file for Normalize
argument: `"%s"`, position: -1
gradient: (an integer (int or long))
use max intensity/mm gradient g (default=1)
argument: `"-g %d"`
mask: (a pathlike object or string representing an existing file)
The input mask file for Normalize
argument: `"-mask %s"`
segmentation: (a pathlike object or string representing an existing file)
The input segmentation for Normalize
argument: `"-aseg %s"`
transform: (a pathlike object or string representing an existing file)
Transform file from the header of the input file
subjects_dir: (a pathlike object or string representing an existing directory)
subjects directory
args: (a unicode string)
Additional parameters to the command
argument: `"%s"`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

out_file: (a pathlike object or string representing a file)
The output file for Normalize

62.3.14 ParseDICOMDir

Link to code
Wraps the executable command mri_parse_sdcmdir.
Uses mri_parse_sdcmdir to get information from dicom directories

Examples

```python
>>> from nipype.interfaces.freesurfer import ParseDICOMDir
>>> dcminfo = ParseDICOMDir()
>>> dcminfo.inputs.dicom_dir = '.
>>> dcminfo.inputs.sortbyrun = True
>>> dcminfo.inputs.summarize = True
>>> dcminfo.cmdline
'mri_parse_sdcmdir --d . --o dicominfo.txt --sortbyrun --summarize'
```

Inputs:
[Mandatory]
dicom_dir: (a pathlike object or string representing an existing
directory)
    path to siemens dicom directory
    argument: `--d %s`

[Optional]
dicom_info_file: (a pathlike object or string representing a file,
nipype default value: dicominfo.txt)
    file to which results are written
    argument: `--o %s`
sortbyrun: (a boolean)
    assign run numbers
    argument: `--sortbyrun`
summarize: (a boolean)
    only print out info for run leaders
    argument: `--summarize`
subjects_dir: (a pathlike object or string representing an existing
directory)
    subjects directory
args: (a unicode string)
    Additional parameters to the command
    argument: `-%s`
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})"
    Environment variables

Outputs:
dicom_info_file: (a pathlike object or string representing an
existing file)
    text file containing dicom information

62.3.15 ReconAll

Link to code
Wraps the executable command recon-all.
Uses recon-all to generate surfaces and parcellations of structural data from anatomical images of a subject.

Examples

```python
>>> from nipype.interfaces.freesurfer import ReconAll
>>> reconall = ReconAll()
>>> reconall.inputs.subject_id = 'foo'
>>> reconall.inputs.directive = 'all'
>>> reconall.inputs.subjects_dir = '.
>>> reconall.inputs.T1_files = 'structural.nii'
>>> reconall.cmdline
'recon-all -all -i structural.nii -subjid foo -sd .'
>>> reconall.inputs.flags = '-qcache'
>>> reconall.cmdline
'recon-all -all -i structural.nii -qcache -subjid foo -sd .'
>>> reconall.inputs.flags = ['-cw256', '-qcache']
>>> reconall.cmdline
'recon-all -all -i structural.nii -cw256 -qcache -subjid foo -sd .'
```
Hemisphere may be specified regardless of directive:

```bash
>>> reconall.inputs.flags = []
>>> reconall.inputs.hemi = 'lh'
>>> reconall.cmdline
'recon-all -all -i structural.nii -hemi lh -subjid foo -sd .'
```

```
>>> reconall.inputs.directive = 'autorecon-hemi'
>>> reconall.cmdline
'recon-all -autorecon-hemi lh -i structural.nii -subjid foo -sd .'
```

Hemispheric subfields can accept T1 and T2 images:

```bash
>>> reconall_subfields = ReconAll()
>>> reconall_subfields.inputs.subject_id = 'foo'
>>> reconall_subfields.inputs.directive = 'all'
>>> reconall_subfields.inputs.subjects_dir = '.
>>> reconall_subfields.inputs.T1_files = 'structural.nii'
>>> reconall_subfields.inputs.hippocampal_subfields_T1 = True
>>> reconall_subfields.cmdline
'recon-all -all -i structural.nii -hippocampal-subfields-T1 -subjid foo -sd .'
>>> reconall_subfields.inputs.hippocampal_subfields_T2 = (...
...  'structural.nii', 'test')
>>> reconall_subfields.cmdline
'recon-all -all -i structural.nii -hippocampal-subfields-T1T2 structural.nii test...
...  -subjid foo -sd .'
>>> reconall_subfields.inputs.hippocampal_subfields_T1 = False
>>> reconall_subfields.cmdline
'recon-all -all -i structural.nii -hippocampal-subfields-T2 structural.nii test...
...  -subjid foo -sd .'
```

Inputs:

[Optional]
subject_id: (a unicode string, nipype default value: recon_all)
  subject name
  argument: `'-subjid %s'`
directive: ('all' or 'autorecon1' or 'autorecon2' or
  'autorecon2-volonly' or 'autorecon2-perhemi' or
  'autorecon2-inflatel' or 'autorecon2-cp' or 'autorecon2-wm' or
  'autorecon3' or 'autorecon3-T2pial' or 'autorecon-pial' or
  'autorecon-hemi' or 'localGI' or 'qcache', nipype default value: all)
  process directive
  argument: `'-%s'`, position: 0
hemi: ('lh' or 'rh')
  hemisphere to process
  argument: `'-hemi %s'`
T1_files: (a list of items which are a pathlike object or string
  representing an existing file)
  name of T1 file to process
  argument: `'-i %s'`
T2_file: (a pathlike object or string representing an existing file)
  Convert T2 image to orig directory
  argument: `'-T2 %s'`
FLAIR_file: (a pathlike object or string representing an existing file)
  Convert FLAIR image to orig directory
(continues on next page)
argument: ``-FLAIR %s``

use_T2: (a boolean)
  Use T2 image to refine the pial surface
  argument: ``-T2pial``
  mutually_exclusive: use_FLAIR

use_FLAIR: (a boolean)
  Use FLAIR image to refine the pial surface
  argument: ``-FLAIRpial``
  mutually_exclusive: use_T2

openmp: (an integer (int or long))
  Number of processors to use in parallel
  argument: ``-openmp %d``

parallel: (a boolean)
  Enable parallel execution
  argument: ``-parallel``

hires: (a boolean)
  Conform to minimum voxel size (for voxels < 1mm)
  argument: ``-hires``

mprage: (a boolean)
  Assume scan parameters are MGH MP-RAGE protocol, which produces
darker gray matter
  argument: ``-mprage``

big_ventricles: (a boolean)
  For use in subjects with enlarged ventricles
  argument: ``-bigventricles``

brainstem: (a boolean)
  Segment brainstem structures
  argument: ``-brainstem-structures``

hippocampal_subfields_T1: (a boolean)
  segment hippocampal subfields using input T1 scan
  argument: ``-hippocampal-subfields-T1``

hippocampal_subfields_T2: (a tuple of the form: (a pathlike object or
  string representing an existing file, a unicode string))
  segment hippocampal subfields using T2 scan, identified by ID (may
  be combined with hippocampal_subfields_T1)
  argument: ``-hippocampal-subfields-T2 %s %s``

expert: (a pathlike object or string representing an existing file)
  Set parameters using expert file
  argument: ``-expert %s``

xopts: ('use' or 'clean' or 'overwrite')
  Use, delete or overwrite existing expert options file
  argument: ``-xopts-%s``

subjects_dir: (a pathlike object or string representing an existing
directory)
  path to subjects directory
  argument: ``-sd %s``

flags: (a list of items which are a unicode string)
  additional parameters
  argument: ``%s``

talairach: (a unicode string)
  Flags to pass to talairach commands
  mutually_exclusive: expert

mri_normalize: (a unicode string)
  Flags to pass to mri_normalize commands
  mutually_exclusive: expert

mri_watershed: (a unicode string)
  Flags to pass to mri_watershed commands
mutually_exclusive: expert
mri_em_register: (a unicode string)
   Flags to pass to mri_em_register commands
mutually_exclusive: expert
mri_ca_normalize: (a unicode string)
   Flags to pass to mri_ca_normalize commands
mutually_exclusive: expert
mri_ca_register: (a unicode string)
   Flags to pass to mri_ca_register commands
mutually_exclusive: expert
mri_remove_neck: (a unicode string)
   Flags to pass to mri_remove_neck commands
mutually_exclusive: expert
mri_ca_label: (a unicode string)
   Flags to pass to mri_ca_label commands
mutually_exclusive: expert
mri_segstats: (a unicode string)
   Flags to pass to mri_segstats commands
mutually_exclusive: expert
mri_mask: (a unicode string)
   Flags to pass to mri_mask commands
mutually_exclusive: expert
mri_segment: (a unicode string)
   Flags to pass to mri_segment commands
mutually_exclusive: expert
mri_edit_wm_with_aseg: (a unicode string)
   Flags to pass to mri_edit_wm_with_aseg commands
mutually_exclusive: expert
mri_pretess: (a unicode string)
   Flags to pass to mri_pretess commands
mutually_exclusive: expert
mri_fill: (a unicode string)
   Flags to pass to mri_fill commands
mutually_exclusive: expert
mri_tessellate: (a unicode string)
   Flags to pass to mri_tessellate commands
mutually_exclusive: expert
mris_smooth: (a unicode string)
   Flags to pass to mris_smooth commands
mutually_exclusive: expert
mris Inflate: (a unicode string)
   Flags to pass to mris Inflate commands
mutually_exclusive: expert
mris_sphere: (a unicode string)
   Flags to pass to mris_sphere commands
mutually_exclusive: expert
mris_fix_topology: (a unicode string)
   Flags to pass to mris_fix_topology commands
mutually_exclusive: expert
mris_make_surfaces: (a unicode string)
   Flags to pass to mris_make_surfaces commands
mutually_exclusive: expert
mris_surf2vol: (a unicode string)
   Flags to pass to mris_surf2vol commands
mutually_exclusive: expert
mris_register: (a unicode string)
   Flags to pass to mris_register commands
mutually_exclusive: expert

mrsp_paint: (a unicode string)
  Flags to pass to mrsp_paint commands
mutually_exclusive: expert

mriscalabel: (a unicode string)
  Flags to pass to mriscalabel commands
mutually_exclusive: expert

mris_anatomical_stats: (a unicode string)
  Flags to pass to mris_anatomical_stats commands
mutually_exclusive: expert

mri_aparc2aseg: (a unicode string)
  Flags to pass to mri_aparc2aseg commands
mutually_exclusive: expert

args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

subjects_dir: (a pathlike object or string representing an existing directory)
  Freesurfer subjects directory.
subject_id: (a unicode string)
  Subject name for whom to retrieve data
T1: (a pathlike object or string representing an existing file)
  Intensity normalized whole-head volume
aseg: (a pathlike object or string representing an existing file)
  Volumetric map of regions from automatic segmentation
brain: (a pathlike object or string representing an existing file)
  Intensity normalized brain-only volume
brainmask: (a pathlike object or string representing an existing file)
  Skull-stripped (brain-only) volume
filled: (a pathlike object or string representing an existing file)
  Subcortical mass volume
norm: (a pathlike object or string representing an existing file)
  Normalized skull-stripped volume
nu: (a pathlike object or string representing an existing file)
  Non-uniformity corrected whole-head volume
orig: (a pathlike object or string representing an existing file)
  Base image conformed to Freesurfer space
rawavg: (a pathlike object or string representing an existing file)
  Volume formed by averaging input images
ribbon: (a list of items which are a pathlike object or string representing an existing file)
  Volumetric maps of cortical ribbons
wm: (a pathlike object or string representing an existing file)
  Segmented white-matter volume
wmparc: (a pathlike object or string representing an existing file)
  Aparc parcellation projected into subcortical white matter
curv: (a list of items which are a pathlike object or string representing an existing file)
  Maps of surface curvature
avg_curv: (a list of items which are a pathlike object or string representing an existing file)
    Average atlas curvature, sampled to subject
inflated: (a list of items which are a pathlike object or string representing an existing file)
    Inflated surface meshes
pial: (a list of items which are a pathlike object or string representing an existing file)
    Gray matter/pia mater surface meshes
area_pial: (a list of items which are a pathlike object or string representing an existing file)
    Mean area of triangles each vertex on the pial surface is associated with
curv_pial: (a list of items which are a pathlike object or string representing an existing file)
    Curvature of pial surface
smoothwm: (a list of items which are a pathlike object or string representing an existing file)
    Smoothed original surface meshes
sphere: (a list of items which are a pathlike object or string representing an existing file)
    Spherical surface meshes
sulc: (a list of items which are a pathlike object or string representing an existing file)
    Surface maps of sulcal depth
thickness: (a list of items which are a pathlike object or string representing an existing file)
    Surface maps of cortical thickness
volume: (a list of items which are a pathlike object or string representing an existing file)
    Surface maps of cortical volume
white: (a list of items which are a pathlike object or string representing an existing file)
    White/gray matter surface meshes
jacobian_white: (a list of items which are a pathlike object or string representing an existing file)
    Distortion required to register to spherical atlas
graymid: (a list of items which are a pathlike object or string representing an existing file)
    Graymid/midthickness surface meshes
label: (a list of items which are a pathlike object or string representing an existing file)
    Volume and surface label files
annot: (a list of items which are a pathlike object or string representing an existing file)
    Surface annotation files
aparc_aseg: (a list of items which are a pathlike object or string representing an existing file)
    Aparc parcellation projected into aseg volume
sphere_reg: (a list of items which are a pathlike object or string representing an existing file)
    Spherical registration file
aseg_stats: (a list of items which are a pathlike object or string representing an existing file)
    Automated segmentation statistics file
wmparc_stats: (a list of items which are a pathlike object or string representing an existing file)
White matter parcellation statistics file
aparc_stats: (a list of items which are a pathlike object or string representing an existing file)
  Aparc parcellation statistics files
BA_stats: (a list of items which are a pathlike object or string representing an existing file)
  Brodmann Area statistics files
aparc_a2009s_stats: (a list of items which are a pathlike object or string representing an existing file)
  Aparc a2009s parcellation statistics files
curv_stats: (a list of items which are a pathlike object or string representing an existing file)
  Curvature statistics files
entorhinal_exvivo_stats: (a list of items which are a pathlike object or string representing an existing file)
  Entorhinal exvivo statistics files

62.3.16 Resample

Link to code
Wraps the executable command mri_convert.
Use FreeSurfer mri_convert to up or down-sample image files

Examples

>>> from nipype.interfaces import freesurfer
>>> resampler = freesurfer.Resample()
>>> resampler.inputs.in_file = 'structural.nii'
>>> resampler.inputs.resampled_file = 'resampled.nii'
>>> resampler.inputs.voxel_size = (2.1, 2.1, 2.1)
>>> resampler.cmdline
'mri_convert -vs 2.10 2.10 2.10 -i structural.nii -o resampled.nii'

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  file to resample
  argument: `-i %s`, position: -2
voxel_size: (a tuple of the form: (a float, a float, a float))
  triplet of output voxel sizes
  argument: `-vs %.2f %.2f %.2f`

[Optional]
resampled_file: (a pathlike object or string representing a file)
  output filename
  argument: `-o %s`, position: -1
subjects_dir: (a pathlike object or string representing an existing directory)
  subjects directory
args: (a unicode string)
  Additional parameters to the command
  argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {}}

Environment variables

Outputs:

resampled_file: (a pathlike object or string representing an existing file)
output filename

62.3.17 RobustRegister

Link to code
Wraps the executable command mri_robust_register.
Perform intramodal linear registration (translation and rotation) using robust statistics.

Examples

```python
>>> from nipype.interfaces.freesurfer import RobustRegister
>>> reg = RobustRegister()
>>> reg.inputs.source_file = 'structural.nii'
>>> reg.inputs.target_file = 'T1.nii'
>>> reg.inputs.auto_sens = True
>>> reg.inputs.init_orient = True
>>> reg.cmdline # doctest: +ELLIPSIS
'mri_robust_register --satit --initorient --lta .../structural_robustreg.lta --mov structural.nii --dst T1.nii'
```

References


Inputs:

[Mandatory]
source_file: (a pathlike object or string representing an existing file)
volume to be registered
argument: `--mov %s`
target_file: (a pathlike object or string representing an existing file)
target volume for the registration
argument: `--dst %s`

auto_sens: (a boolean)
auto-detect good sensitivity
argument: `--satit`
mutually_exclusive: outlier_sens

outlier_sens: (a float)
set outlier sensitivity explicitly
argument: `--sat %.4f`
mutually_exclusive: auto_sens

[Optional]
out_reg_file: (a bool or None or a pathlike object or string representing a file, nipype default value: True)
registration file; either True or filename
registered_file: (a boolean or a pathlike object or string representing a file)
    registered image; either True or filename
argument: '``--lta %s``'
weights_file: (a boolean or a pathlike object or string representing a file)
    weights image to write; either True or filename
argument: '``--weights %s``'
est_int_scale: (a boolean)
    estimate intensity scale (recommended for unnormalized images)
argument: '``--iscale``'
trans_only: (a boolean)
    find 3 parameter translation only
argument: '``--transonly``'
in_xfm_file: (a pathlike object or string representing an existing file)
    use initial transform on source
argument: '``--transform``'
half_source: (a boolean or a pathlike object or string representing a file)
    write source volume mapped to halfway space
argument: '``--halfmov %s``'
half_targ: (a boolean or a pathlike object or string representing a file)
    write target volume mapped to halfway space
argument: '``--halfdst %s``'
half_weights: (a boolean or a pathlike object or string representing a file)
    write weights volume mapped to halfway space
argument: '``--halfweights %s``'
half_source_xfm: (a boolean or a pathlike object or string representing a file)
    write transform from source to halfway space
argument: '``--halfmovlta %s``'
half_targ_xfm: (a boolean or a pathlike object or string representing a file)
    write transform from target to halfway space
argument: '``--halfdstlta %s``'
least_squares: (a boolean)
    use least squares instead of robust estimator
argument: '``--leastsquares``'
o_init: (a boolean)
    skip transform init
argument: '``--noinit``'
init_orient: (a boolean)
    use moments for initial orient (recommended for stripped brains)
argument: '``--initorient``'
max_iterations: (an integer (int or long))
    maximum # of times on each resolution
argument: '``--maxit %d``'
high_iterations: (an integer (int or long))
    max # of times on highest resolution
argument: '``--highit %d``'
iteration_thresh: (a float)
    stop iterations when below threshold
argument: '``--epsit %.3f``'

(continues on next page)
subsample_thresh: (an integer (int or long))
subsample if dimension is above threshold size
argument: ``--subsample %d``

outlier_limit: (a float)
set maximal outlier limit in satit
argument: ``--wlimit %.3f``

write_vo2vox: (a boolean)
output vox2vox matrix (default is RAS2RAS)
argument: ``--vo2vox``

no_multi: (a boolean)
work on highest resolution
argument: ``--nomulti``

mask_source: (a pathlike object or string representing an existing image)
image to mask source volume with
argument: ``--maskmov %s``

mask_target: (a pathlike object or string representing an existing file)
image to mask target volume with
argument: ``--maskdst %s``

force_double: (a boolean)
use double-precision intensities
argument: ``--doubleprec``

force_float: (a boolean)
use float intensities
argument: ``--floattype``

subjects_dir: (a pathlike object or string representing an existing directory)
subjects directory

args: (a unicode string)
Additional parameters to the command
argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

out_reg_file: (a pathlike object or string representing an existing file)
output registration file
registered_file: (a pathlike object or string representing an existing file)
output image with registration applied
weights_file: (a pathlike object or string representing an existing file)
image of weights used
half_source: (a pathlike object or string representing an existing file)
source image mapped to halfway space
half_targ: (a pathlike object or string representing an existing file)
target image mapped to halfway space
half_weights: (a pathlike object or string representing an existing file)
weights image mapped to halfway space
half_source_xfm: (a pathlike object or string representing an existing file)
transform file to map source image to halfway space

half_targ_xfm: (a pathlike object or string representing an existing file)
transform file to map target image to halfway space

62.3.18 SegmentCC

Link to code
Wraps the executable command mri_cc.
This program segments the corpus callosum into five separate labels in the subcortical segmentation volume `aseg.mgz`. The divisions of the cc are equally spaced in terms of distance along the primary eigendirection (pretty much the long axis) of the cc. The lateral extent can be changed with the -T <thickness> parameter, where <thickness> is the distance off the midline (so -T 1 would result in the whole CC being 3mm thick). The default is 2 so it’s 5mm thick. The aseg.stats values should be volume.

Examples

```python
>>> from nipype.interfaces import freesurfer

SegmentCC_node = freesurfer.SegmentCC()  
SegmentCC_node.inputs.in_file = "aseg.mgz"
SegmentCC_node.inputs.in_norm = "norm.mgz"
SegmentCC_node.inputs.out_rotation = "cc.lta"
SegmentCC_node.inputs.subject_id = "test"
SegmentCC_node.cmdline
"mri_cc -aseg aseg.mgz -o aseg.auto.mgz -lta cc.lta test"
```

Inputs:

[Mandatory]

in_file: (a pathlike object or string representing an existing file)
Input aseg file to read from subjects directory
argument: `\"-aseg \$s\"`

in_norm: (a pathlike object or string representing an existing file)
Required undocumented input {subject}/mri/norm.mgz

out_rotation: (a pathlike object or string representing a file)
Global filepath for writing rotation lta
argument: `\"-lta \$s\"`

subject_id: (a string, nipype default value: subject_id)
Subject name
argument: `\"\$s\"`, position: -1

[Optional]

out_file: (a pathlike object or string representing a file)
Filename to write aseg including CC
argument: `\"-o \$s\"`

copy_inputs: (a boolean)
If running as a node, set this to True.This will copy the input files to the node directory.

subjects_dir: (a pathlike object or string representing an existing directory)
subjects directory

args: (a unicode string)
Additional parameters to the command
62.3.19 SegmentWM

Link to code

Wraps the executable command `mri_segment`.
This program segments white matter from the input volume. The input volume should be normalized such that white matter voxels are ~110-valued, and the volume is conformed to 256^3.

Examples

```python
>>> from nipype.interfaces import freesurfer
>>> SegmentWM_node = freesurfer.SegmentWM()
>>> SegmentWM_node.inputs.in_file = "norm.mgz"
>>> SegmentWM_node.inputs.out_file = "wm.seg.mgz"
>>> SegmentWM_node.cmdline
'mri_segment norm.mgz wm.seg.mgz'
```

Inputs:

|Mandatory|
in_file: (a pathlike object or string representing an existing file)
  - Input file for SegmentWM
    - argument: ``%s``
    - position: -2

|Optional|
subjects_dir: (a pathlike object or string representing an existing directory)
  - subjects directory
args: (a unicode string)
  - Additional parameters to the command
    - argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
62.3.20 Smooth

Link to code
Wraps the executable command `mris_volsmooth`.
Use FreeSurfer `mris_volsmooth` to smooth a volume
This function smoothes cortical regions on a surface and non-cortical regions in volume.

**Note:** Cortical voxels are mapped to the surface (3D->2D) and then the smoothed values from the surface are put back into the volume to fill the cortical ribbon. If data is smoothed with this algorithm, one has to be careful about how further processing is interpreted.

**Examples**

```python
>>> from nipype.interfaces.freesurfer import Smooth
>>> smoothvol = Smooth(in_file='functional.nii', smoothed_file = 'foo_out.nii',
→reg_file='register.dat', surface_fwhm=10, vol_fwhm=6)
>>> smoothvol.cmdline
'mris_volsmooth --i functional.nii --reg register.dat --o foo_out.nii --fwhm 10.
→000000 --vol-fwhm 6.000000'
```

**Inputs:**

[Mandatory]
- `in_file`: (a pathlike object or string representing an existing file)
  - source volume
  - argument: `'--i %s'`
- `reg_file`: (a pathlike object or string representing an existing file)
  - registers volume to surface anatomical
  - argument: `'--reg %s'`
- `surface_fwhm`: (a floating point number >= 0.0)
  - surface FWHM in mm
  - argument: `'--fwhm %f'`
  - mutually_exclusive: num_iters
  - requires: reg_file
- `num_iters`: (a long integer >= 1)
  - number of iterations instead of fwhm
  - argument: `'--niters %d'`
  - mutually_exclusive: surface_fwhm

[Optional]
- `smoothed_file`: (a pathlike object or string representing a file)
  - output volume
  - argument: `'--o %s'`
- `proj_frac_avg`: (a tuple of the form: (a float, a float, a float))
  - average a long normal min max delta
  - argument: `'--projfrac-avg %.2f %.2f %.2f'`
  - mutually_exclusive: proj_frac
- `proj_frac`: (a float)
  - project frac of thickness a long surface normal
  - argument: `'--projfrac %s'`
  - mutually_exclusive: proj_frac_avg
- `vol_fwhm`: (a floating point number >= 0.0)
  - volume smoothing outside of surface
  - argument: `'--vol-fwhm %f'`
- `subjects_dir`: (a pathlike object or string representing an existing directory)

(continues on next page)
62.3.21 SynthesizeFLASH

Link to code
Wraps the executable command mri_synthesize.
Synthesize a FLASH acquisition from T1 and proton density maps.

Examples

```python
>>> from nipype.interfaces.freesurfer import SynthesizeFLASH
>>> syn = SynthesizeFLASH(tr=20, te=3, flip_angle=30)
>>> syn.inputs.t1_image = 'T1.mgz'
>>> syn.inputs.pd_image = 'PD.mgz'
>>> syn.inputs.out_file = 'flash_30syn.mgz'
>>> syn.cmdline
'mri_synthesize 20.00 3.000 30.00 T1.mgz PD.mgz flash_30syn.mgz'
```

Inputs:

```plaintext
[Mandatory]
tr: (a float)
  repetition time (in msec)
  argument: ``%.2f``, position: 2
flip_angle: (a float)
  flip angle (in degrees)
  argument: ``%.2f``, position: 3
te: (a float)
  echo time (in msec)
  argument: ``%.3f``, position: 4
t1_image: (a pathlike object or string representing an existing file)
  image of T1 values
  argument: ``%s``, position: 5
pd_image: (a pathlike object or string representing an existing file)
  image of proton density values
  argument: ``%s``, position: 6

[Optional]
fixed_weighting: (a boolean)
  use a fixed weighting to generate optimal gray/white contrast
  argument: ``-w``
  position: 1
out_file: (a pathlike object or string representing a file)
  image to write
```

(continues on next page)
argument: `"%s"`
subjects_dir: (a pathlike object or string representing an existing directory)
  subjects directory
args: (a unicode string)
  Additional parameters to the command
  argument: `"%s"
envirion: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
  synthesized FLASH acquisition

62.3.22 UnpackSDICOMDir

Link to code

Wraps the executable command unpacksdcmdir.

Use unpacksdcmdir to convert dicom files

Call unpacksdcmdir -help from the command line to see more information on using this command.

Examples

```python
>>> from nipype.interfaces.freesurfer import UnpackSDICOMDir
>>> unpack = UnpackSDICOMDir()
>>> unpack.inputs.source_dir = '.'
>>> unpack.inputs.output_dir = '.'
>>> unpack.inputs.run_info = (5, 'mprage', 'nii', 'struct')
>>> unpack.inputs.dir_structure = 'generic'
>>> unpack.cmdline
'unpacksdcmdir -generic -targ . -run 5 mprage nii struct -src .'
```

Inputs:

[Mandatory]
source_dir: (a pathlike object or string representing an existing directory)
  directory with the DICOM files
  argument: `"-src %s"`
run_info: (a tuple of the form: (an integer (int or long), a unicode string, a unicode string, a unicode string))
  runno subdir format name : spec unpacking rules on cmdline
  argument: `"-run %d %s %s %s"`
  mutually_exclusive: run_info, config, seq_config
config: (a pathlike object or string representing an existing file)
  specify unpacking rules in file
  argument: `"-cfg %s"`
  mutually_exclusive: run_info, config, seq_config
seq_config: (a pathlike object or string representing an existing file)
  specify unpacking rules based on sequence
  argument: `"-seqcfg %s"`
  mutually_exclusive: run_info, config, seq_config

(continues on next page)
[Optional]
output_dir: (a pathlike object or string representing a directory)
   top directory into which the files will be unpacked
   argument: ``-targ %s``
dir_structure: ('fsfast' or 'generic')
   unpack to specified directory structures
   argument: ``-%s``
no_info_dump: (a boolean)
   do not create infodump file
   argument: ``-noinfodump``
scan_only: (a pathlike object or string representing an existing file)
   only scan the directory and put result in file
   argument: ``-scanonly %s``
log_file: (a pathlike object or string representing an existing file)
   explicitly set log file
   argument: ``-log %s``
spm_zeropad: (an integer (int or long))
   set frame number zero padding width for SPM
   argument: ``-nspmzeropad %d``
no_unpack_err: (a boolean)
   do not try to unpack runs with errors
   argument: ``-no-unpackerr``
subjects_dir: (a pathlike object or string representing an existing directory)
   subjects directory
args: (a unicode string)
   Additional parameters to the command
   argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
>>> skullstrip.inputs.in_file = "T1.mgz"
>>> skullstrip.inputs.t1 = True
>>> skullstrip.inputs.transform = "transforms/talairach_with_skull.lta"
>>> skullstrip.inputs.out_file = "brainmask.auto.mgz"
>>> skullstrip.cmdline
'mri_watershed -T1 transforms/talairach_with_skull.lta T1.mgz brainmask.auto.mgz'

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  input volume
  argument: ``\$s`` , position: -2
out_file: (a pathlike object or string representing a file, nipype
  default value: brainmask.auto.mgz)
  output volume
  argument: ``\$s`` , position: -1

[Optional]
t1: (a boolean)
  specify T1 input volume (T1 grey value = 110)
  argument: ``-T1``
brain_atlas: (a pathlike object or string representing an existing
  file)
  argument: ``-brain_atlas %s`` , position: -4
transform: (a pathlike object or string representing a file)
  undocumented
  argument: ``\$s`` , position: -3
subjects_dir: (a pathlike object or string representing an existing
  directory)
  subjects directory
args: (a unicode string)
  Additional parameters to the command
  argument: ``\$s``
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
  Environment variables

Outputs:

out_file: (a pathlike object or string representing a file)
  skull stripped brain volume

62.4 interfaces.freesurfer.registration

62.4.1 EMRegister

Link to code
Wraps the executable command mri_em_register.
This program creates a transform in lta format

Examples
>>> from nipype.interfaces.freesurfer import EMRegister
>>> register = EMRegister()
>>> register.inputs.in_file = 'norm.mgz'
>>> register.inputs.template = 'aseg.mgz'
>>> register.inputs.out_file = 'norm_transform.lta'
>>> register.inputs.skull = True
>>> register.inputs.nbrspacing = 9
>>> register.cmdline
'mri_em_register -uns 9 -skull norm.mgz aseg.mgz norm_transform.lta'

Inputs:

- **in_file**: (a pathlike object or string representing an existing file)
  - in brain volume
  - argument: ``%s``, position: -3
- **template**: (a pathlike object or string representing an existing file)
  - template gca
  - argument: ``%s``, position: -2

- **out_file**: (a pathlike object or string representing a file)
  - output transform
  - argument: ``%s``, position: -1
- **skull**: (a boolean)
  - align to atlas containing skull (uns=5)
  - argument: ``-skull``
- **mask**: (a pathlike object or string representing an existing file)
  - use volume as a mask
  - argument: ``-mask %s``
- **nbrspacing**: (an integer (int or long))
  - align to atlas containing skull setting unknown_nbr_spacing = nbrspacing
  - argument: ``-uns %d``
- **transform**: (a pathlike object or string representing an existing file)
  - Previously computed transform
  - argument: ``-t %s``
- **num_threads**: (an integer (int or long))
  - allows for specifying more threads
- **subjects_dir**: (a pathlike object or string representing an existing directory)
  - subjects directory
- **args**: (a unicode string)
  - Additional parameters to the command
  - argument: ``%s``
- **environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Wraps the executable command mpr2mni305.
For complete details, see FreeSurfer documentation

Examples

```python
>>> from nipype.interfaces.freesurfer import MPRtoMNI305, Info
>>> mprtoni305 = MPRtoMNI305()
>>> mprtoni305.inputs.target = 'structural.nii'
>>> mprtoni305.inputs.reference_dir = '.' # doctest: +SKIP
>>> mprtoni305.cmdline # doctest: +SKIP
'mpr2mni305 output'
>>> mprtoni305.inputs.out_file = 'struct_out' # doctest: +SKIP
>>> mprtoni305.cmdline # doctest: +SKIP
'mpr2mni305 struct_out' # doctest: +SKIP
>>> mprtoni305.inputs.environ['REFDIR'] == os.path.join(Info.home(), 'average')
True
>>> mprtoni305.inputs.environ['MPR2MNI305_TARGET'] # doctest: +SKIP
'structural'
>>> mprtoni305.run() # doctest: +SKIP
```

Inputs:

- **reference_dir**: (a pathlike object or string representing an existing directory, nipype default value: )
  
  TODO
- **target**: (a string, nipype default value: )
  
  input atlas file
- **in_file**: (a pathlike object or string representing a file, nipype default value: <undefined>)
  
  the input file prefix for MPRtoMNI305
  
  argument: ``%s``
- **subjects_dir**: (a pathlike object or string representing an existing directory)
  
  subjects directory
- **args**: (a unicode string)
  
  Additional parameters to the command
  
  argument: ``%s``
- **environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) 
  
  Environment variables

Outputs:

- **out_file**: (a pathlike object or string representing a file)
  
  The output file '<in_file>_to_<target>_t4_vox2vox.txt'
- **log_file**: (a pathlike object or string representing an existing file, nipype default value: output.nipype)
  
  The output log

62.4.3 MRICoreg

Link to code

Wraps the executable command mri_coreg.
This program registers one volume to another
mri_coreg is a C reimplementation of spm_coreg in FreeSurfer

Examples

```python
>>> from nipype.interfaces.freesurfer import MRICoreg
>>> coreg = MRICoreg()
>>> coreg.inputs.source_file = 'moving1.nii'
>>> coreg.inputs.reference_file = 'fixed1.nii'
>>> coreg.inputs.subjects_dir = '.

'mri_coreg --lta .../registration.lta --ref fixed1.nii --mov moving1.nii --sd .'
```

If passing a subject ID, the reference mask may be disabled:

```python
>>> coreg = MRICoreg()
>>> coreg.inputs.source_file = 'moving1.nii'
>>> coreg.inputs.subjects_dir = '.

'mri_coreg --s fsaverage --no-ref-mask --lta .../registration.lta --mov moving1.nii --sd .'
```

Spatial scales may be specified as a list of one or two separations:

```python
>>> coreg.inputs.sep = [4]

'mri_coreg --s fsaverage --no-ref-mask --lta .../registration.lta --sep 4 --mov moving1.nii --sd .'
```

```python
>>> coreg.inputs.sep = [4, 5]

'mri_coreg --s fsaverage --no-ref-mask --lta .../registration.lta --sep 4 --sep 5 --mov moving1.nii --sd .'
```

Inputs:

[Mandatory]
source_file: (a pathlike object or string representing a file)
  source file to be registered
  argument: ``--mov %s``
reference_file: (a pathlike object or string representing a file)
  reference (target) file
  argument: ``--ref %s``
  mutually_exclusive: subject_id
subject_id: (a unicode string)
  freesurfer subject ID (implies ``reference_mask == aparc+aseg.mgz``
  unless otherwise specified)
  argument: ``--s %s'', position: 1
  mutually_exclusive: reference_file
  requires: subjects_dir

[Optional]
out_lta_file: (a bool or None or a pathlike object or string
  representing a file, nipype default value: True)
  output registration file (LTA format)
  argument: ``--lta %s``
```

(continues on next page)
out_reg_file: (a bool or None or a pathlike object or string representing a file)  
output registration file (REG format)  
argument: '``--regdat %s``'

out_params_file: (a bool or None or a pathlike object or string representing a file)  
output parameters file  
argument: '``--params %s``'

subjects_dir: (a pathlike object or string representing an existing directory)  
FreeSurfer SUBJECTS_DIR  
argument: '``--sd %s``'

dof: (6 or 9 or 12)  
number of transform degrees of freedom  
argument: '``--dof %d``'

reference_mask: (a bool or None or a unicode string)  
mask reference volume with given mask, or None if `False`  
argument: '``--ref-mask %s``', position: 2

source_mask: (a unicode string)  
mask source file with given mask  
argument: '``--mov-mask``'

num_threads: (an integer (int or long))  
number of OpenMP threads  
argument: '``--threads %d``'

no_coord_dithering: (a boolean)  
turn off coordinate dithering  
argument: '``--no-coord-dither``'

no_intensity_dithering: (a boolean)  
turn off intensity dithering  
argument: '``--no-intensity-dither``'

sep: (a list of from 1 to 2 items which are any value)  
set spatial scales, in voxels (default [2, 4])  
argument: '``--sep %s...``'

initial_translation: (a tuple of the form: (a float, a float, a float))  
initial translation in mm (implies no_cras0)  
argument: '``--trans %g %g %g``'

initial_rotation: (a tuple of the form: (a float, a float, a float))  
initial rotation in degrees  
argument: '``--rot %g %g %g``'

initial_scale: (a tuple of the form: (a float, a float, a float))  
initial scale  
argument: '``--scale %g %g %g``'

initial_shear: (a tuple of the form: (a float, a float, a float))  
initial shear (Hxy, Hxz, Hyz)  
argument: '``--shear %g %g %g``'

no_cras0: (a boolean)  
do not set translation parameters to align centers of source and reference files  
argument: '``--no-cras0``'

max_iters: (a long integer >= 1)  
maximum iterations (default: 4)  
argument: '``--nitermax %d``'

ftol: (a float)  
floating-point tolerance (default=1e-7)  
argument: '``--ftol %e``'

linmintol: (a float)  
(continues on next page)
**62.4.4 Paint**

**Link to code**
Wraps the executable command `mrisp_paint`.

This program is useful for extracting one of the arrays ("a variable") from a surface-registration template file. The output is a file containing a surface-worth of per-vertex values, saved in "curvature" format. Because the template data is sampled to a particular surface mesh, this conjures the idea of "painting to a surface".

---

```
argument: `--linmintol %e`
saturation_threshold: (0.0 <= a floating point number <= 100.0)
    saturation threshold (default=9.999)
    argument: `--sat %g`
conform_reference: (a boolean)
    conform reference without rescaling
    argument: `--conf-ref`
no_brute_force: (a boolean)
    do not brute force search
    argument: `--no-bf`
brute_force_limit: (a float)
    constrain brute force search to +/- lim
    argument: `--bf-lim %g`
    mutually_exclusive: no_brute_force
brute_force_samples: (an integer (int or long))
    number of samples in brute force search
    argument: `--bf-nsamp %d`
    mutually_exclusive: no_brute_force
no_smooth: (a boolean)
    do not apply smoothing to either reference or source file
    argument: `--no-smooth`
ref_fwhm: (a float)
    apply smoothing to reference file
    argument: `--ref-fwhm`
source_oob: (a boolean)
    count source voxels that are out-of-bounds as 0
    argument: `--mov-oob`
args: (a unicode string)
    Additional parameters to the command
    argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})  
    Environment variables
```

**Outputs:**

```
out_reg_file: (a pathlike object or string representing an existing file)
    output registration file
out_lta_file: (a pathlike object or string representing an existing file)
    output LTA-style registration file
out_params_file: (a pathlike object or string representing an existing file)
    output parameters file
```
Examples

```python
>>> from nipype.interfaces.freesurfer import Paint
>>> paint = Paint()
>>> paint.inputs.in_surf = 'lh.pial'
>>> paint.inputs.template = 'aseg.mgz'
>>> paint.inputs.averages = 5
>>> paint.inputs.out_file = 'lh.avg_curv'
>>> paint.cmdline
'mrisp_paint -a 5 aseg.mgz lh.pial lh.avg_curv'
```

Inputs:

[Mandatory]

- `in_surf`: (a pathlike object or string representing an existing file)
  Surface file with grid (vertices) onto which the template data is to be sampled or 'painted'
  argument: `'\%s'`, position: -2

- `template`: (a pathlike object or string representing an existing file)
  Template file
  argument: `'\%s'`, position: -3

[Optional]

- `template_param`: (an integer (int or long))
  Frame number of the input template

- `averages`: (an integer (int or long))
  Average curvature patterns
  argument: `'-a %d'`

- `out_file`: (a pathlike object or string representing a file)
  File containing a surface-worth of per-vertex values, saved in 'curvature' format.
  argument: `'\%s'`, position: -1

- `subjects_dir`: (a pathlike object or string representing an existing directory)
  subjects directory

- `args`: (a unicode string)
  Additional parameters to the command
  argument: `'\%s'`

- `environ`: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: `{}`)
  Environment variables

Outputs:

- `out_file`: (a pathlike object or string representing a file)
  File containing a surface-worth of per-vertex values, saved in 'curvature' format.

62.4.5 Register

Link to code

Wraps the executable command `mris_register`. This program registers a surface to an average surface template.
Examples

```python
>>> from nipype.interfaces.freesurfer import Register
>>> register = Register()
>>> register.inputs.in_surf = 'lh.pial'
>>> register.inputs.in_smoothwm = 'lh.pial'
>>> register.inputs.in_sulc = 'lh.pial'
>>> register.inputs.target = 'aseg.mgz'
>>> register.inputs.out_file = 'lh.pial.reg'
>>> register.inputs.curv = True
>>> register.cmdline = 'mris_register -curv lh.pial aseg.mgz lh.pial.reg'
```

Inputs:

- **in_surf**: (a pathlike object or string representing an existing file)
  - Surface to register, often `{hemi}.sphere`
  - Argument: ``-s``, position: -3

- **target**: (a pathlike object or string representing an existing file)
  - The data to register to. In normal recon-all usage, this is a template file for average surface.
  - Argument: ``-t``, position: -2

- **in_sulc**: (a pathlike object or string representing an existing file)
  - Undocumented mandatory input file
  - Argument: `"${SUBJECTS_DIR}/surf/{hemisphere}.sulc"

- **out_file**: (a pathlike object or string representing a file)
  - Output surface file to capture registration
  - Argument: ``-o`` , position: -1

- **curv**: (a boolean)
  - Use smoothwm curvature for final alignment
  - Argument: ``-curv``
  - Requires: `in_smoothwm`

- **in_smoothwm**: (a pathlike object or string representing an existing file)
  - Undocumented input file `"${SUBJECTS_DIR}/surf/{hemisphere}.smoothwm"

- **subjects_dir**: (a pathlike object or string representing an existing directory)
  - Subjects directory

- **args**: (a unicode string)
  - Additional parameters to the command
  - Argument: `"%s"`

- **environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  - Environment variables

Outputs:

- **out_file**: (a pathlike object or string representing a file)
  - Output surface file to capture registration

62.4.6 RegisterAVItoTalairach

Link to code

Wraps the executable command `avi2talxfm`,

converts the vox2vox from talairach_avi to a talairach.xfm file
This is a script that converts the vox2vox from talairach_avi to a talairach.xfm file. It is meant to replace the following cmd line:

tkregister2cmdl –mov $InVol –targ $FREESURFER_HOME/average/mni305.cor.mgz –xfmout ${XFMP} –vox2vox talsrcimg_to_${target}_t4_vox2vox.txt –noedit –reg tal-

Examples

```python
>>> from nipype.interfaces.freesurfer import RegisterAVItoTalairach
>>> register = RegisterAVItoTalairach()
>>> register.inputs.in_file = 'structural.mgz'
>>> register.inputs.target = 'mni305.cor.mgz'
>>> register.inputs.vox2vox = 'talsrcimg_to_structural_t4_vox2vox.txt'
>>> register.cmdline
'avii2talxfm structural.mgz mni305.cor.mgz talsrcimg_to_structural_t4_vox2vox.txt '
'talairach.auto.xfm'
```

```
>>> register.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
The input file
argument: ``\%s``
position: 0
target: (a pathlike object or string representing an existing file)
The target file
argument: ``\%s``
position: 1
vox2vox: (a pathlike object or string representing an existing file)
The vox2vox file
argument: ``\%s``
position: 2

[Optional]
out_file: (a pathlike object or string representing a file, nipype
default value: talairach.auto.xfm)
The transform output
argument: ``\%s``
position: 3
subjects_dir: (a pathlike object or string representing an existing
directory)
subjects directory
args: (a unicode string)
Additional parameters to the command
argument: ``\%s``
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing a file)
The output file for `RegisterAVItoTalairach`

log_file: (a pathlike object or string representing an existing file, nipype default value: output.nipype)
The output log

## 62.5 interfaces.freesurfer.utils

### 62.5.1 AddXFormToHeader

Link to code

Wraps the executable command `mri_add_xform_to_header`. Just adds specified xform to the volume header

(!) WARNING: transform input MUST be an absolute path to a DataSink’ed transform or the output will reference a transform in the workflow cache directory!

```python
>>> from nipype.interfaces.freesurfer import AddXFormToHeader
>>> adder = AddXFormToHeader()
>>> adder.inputs.in_file = 'norm.mgz'
>>> adder.inputs.transform = 'trans.mat'
>>> adder.cmdline
'mri_add_xform_to_header trans.mat norm.mgz output.mgz'

>>> adder.inputs.copy_name = True
>>> adder.cmdline
'mri_add_xform_to_header -c trans.mat norm.mgz output.mgz'

>>> adder.run()  # doctest: +SKIP
```

### References:

[https://surfer.nmr.mgh.harvard.edu/fswiki/mri_add_xform_to_header](https://surfer.nmr.mgh.harvard.edu/fswiki/mri_add_xform_to_header)

### Inputs:

[Mandatory]

- `in_file`: (a pathlike object or string representing an existing file)
  - input volume
  - argument: `"%s"`, position: -2
- `transform`: (a pathlike object or string representing a file)
  - xfm file
  - argument: `"%s"`, position: -3

[Optional]

- `out_file`: (a pathlike object or string representing a file, nipype default value: output.mgz)
  - output volume
  - argument: `"%s"`, position: -1
- `copy_name`: (a boolean)
  - do not try to load the xfmfile, just copy name
  - argument: `"-c"`
- `verbose`: (a boolean)
  - be verbose
  - argument: `"-v"`
- `subjects_dir`: (a pathlike object or string representing an existing file)

(continues on next page)
62.5.2 Aparc2Aseg

Link to code

Wraps the executable command `mri_aparc2aseg`. Maps the cortical labels from the automatic cortical parcellation (aparc) to the automatic segmentation volume (aseg). The result can be used as the aseg would. The algorithm is to find each aseg voxel labeled as cortex (3 and 42) and assign it the label of the closest cortical vertex. If the voxel is not in the ribbon (as defined by mri/ lh.ribbon and rh.ribbon), then the voxel is marked as unknown (0). This can be turned off with –noribbon. The cortical parcellation is obtained from subject/label/hemi.aparc.annot which should be based on the curvature.buckner40.filled.desikan_killiany.gcs atlas. The aseg is obtained from subject/mri/aseg.mgz and should be based on the RB40_talairach_2005-07-20.gca atlas. If these atlases are used, then the segmentations can be viewed with tkmedit and the FreeSurferColorLUT.txt color table found in $FREESURFER_HOME. These are the default atlases used by recon-all.

Examples

```python
>>> from nipype.interfaces.freesurfer import Aparc2Aseg
>>> aparc2aseg = Aparc2Aseg()
>>> aparc2aseg.inputs.lh_white = 'lh.pial'
>>> aparc2aseg.inputs.rh_white = 'lh.pial'
>>> aparc2aseg.inputs.lh_pial = 'lh.pial'
>>> aparc2aseg.inputs.rh_pial = 'lh.pial'
>>> aparc2aseg.inputs.lh_ribbon = 'label.mgz'
>>> aparc2aseg.inputs.rh_ribbon = 'label.mgz'
>>> aparc2aseg.inputs.ribbon = 'label.mgz'
>>> aparc2aseg.inputs.lh_annotation = 'lh.pial'
>>> aparc2aseg.inputs.rh_annotation = 'lh.pial'
>>> aparc2aseg.inputs.out_file = 'aparc+aseg.mgz'
>>> aparc2aseg.inputs.label_wm = True
>>> aparc2aseg.inputs.rip_unknown = True
>>> aparc2aseg.cmdline # doctest: +SKIP
'mri_aparc2aseg --labelwm --o aparc+aseg.mgz --rip-unknown --s subject_id'
```

Inputs:

- **subject_id**: (a string, nipype default value: subject_id)
  - Subject being processed
  - argument: `''--s %s''`
- **out_file**: (a pathlike object or string representing a file)
Full path of file to save the output segmentation in argument: `--o %s`
lh_white: (a pathlike object or string representing an existing file)
Input file must be <subject_id>/surf/lh.white
rh_white: (a pathlike object or string representing an existing file)
Input file must be <subject_id>/surf/rh.white
lh_pial: (a pathlike object or string representing an existing file)
Input file must be <subject_id>/surf/lh.pial
rh_pial: (a pathlike object or string representing an existing file)
Input file must be <subject_id>/surf/rh.pial
lh_ribbon: (a pathlike object or string representing an existing file)
Input file must be <subject_id>/mri/lh.ribbon.mgz
rh_ribbon: (a pathlike object or string representing an existing file)
Input file must be <subject_id>/mri/rh.ribbon.mgz
ribbon: (a pathlike object or string representing an existing file)
Input file must be <subject_id>/mri/ribbon.mgz
lh_annotation: (a pathlike object or string representing an existing file)
Input file must be <subject_id>/label/lh.aparc.annot
rh_annotation: (a pathlike object or string representing an existing file)
Input file must be <subject_id>/label/rh.aparc.annot

[Optional]
filled: (a pathlike object or string representing an existing file)
Implicit input filled file. Only required with FS v5.3.
aseg: (a pathlike object or string representing an existing file)
Input aseg file
argument: `--aseg %s`
volmask: (a boolean)
Volume mask flag
argument: `--volmask`
ctxseg: (a pathlike object or string representing an existing file)
argument: `--ctxseg %s`
label_wm: (a boolean)
For each voxel labeled as white matter in the aseg, re-assign its label to be that of the closest cortical point if its distance is less than dmaxctx
argument: `--labelwm`
hypo_wm: (a boolean)
Label hypointensities as WM
argument: `--hypo-as-wm`
rip_unknown: (a boolean)
Do not label WM based on 'unknown' corical label
argument: `--rip-unknown`
a2009s: (a boolean)
Using the a2009s atlas
argument: `--a2009s`
copy_inputs: (a boolean)
If running as a node, set this to True. This will copy the input files to the node directory.
subjects_dir: (a pathlike object or string representing an existing directory)
subjects directory
args: (a unicode string)
Additional parameters to the command argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

out_file: (a pathlike object or string representing a file) Output aseg file argument: ``%s``

62.5.3 Apas2Aseg

Link to code
Wraps the executable command apas2aseg. Converts aparc+aseg.mgz into something like aseg.mgz by replacing the cortical segmentations 1000-1035 with 3 and 2000-2035 with 42. The advantage of this output is that the cortical label conforms to the actual surface (this is not the case with aseg.mgz).

Examples

```python
>>> from nipype.interfaces.freesurfer import Apas2Aseg
>>> apas2aseg = Apas2Aseg()
>>> apas2aseg.inputs.in_file = 'aseg.mgz'
>>> apas2aseg.inputs.out_file = 'output.mgz'
>>> apas2aseg.cmdline
'apas2aseg --i aseg.mgz --o output.mgz'
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file) Input aparc+aseg.mgz argument: ``--i %s``
out_file: (a pathlike object or string representing a file) Output aseg file argument: ``--o %s``

[Optional]
subjects_dir: (a pathlike object or string representing an existing directory) subjects directory
args: (a unicode string) Additional parameters to the command argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

out_file: (a pathlike object or string representing a file) Output aseg file argument: ``%s``
## 62.5.4 ApplyMask

Link to code

Wraps the executable command `mri_mask`.

Use Freesurfer's `mri_mask` to apply a mask to an image.

The mask file need not be binarized; it can be thresholded above a given value before application. It can also optionally be transformed into input space with an LTA matrix.

**Inputs:**

```
[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  input image (will be masked)
  argument: ``\%s``, position: -3
mask_file: (a pathlike object or string representing an existing file)
  image defining mask space
  argument: ``\%s``, position: -2

[Optional]
out_file: (a pathlike object or string representing a file)
  final image to write
  argument: ``\%s``, position: -1
xfm_file: (a pathlike object or string representing an existing file)
  LTA-format transformation matrix to align mask with input
  argument: ``-xform %s``
invert_xfm: (a boolean)
  invert transformation
  argument: ``-invert``
xfm_source: (a pathlike object or string representing an existing file)
  image defining transform source space
  argument: ``-lta_src %s``
xfm_target: (a pathlike object or string representing an existing file)
  image defining transform target space
  argument: ``-lta_dst %s``
use_abs: (a boolean)
  take absolute value of mask before applying
  argument: ``-abs``
mask_thresh: (a float)
  threshold mask before applying
  argument: ``-T %.4f``
keep_mask_deletion_edits: (a boolean)
  transfer voxel-deletion edits (voxels=1) from mask to out vol
  argument: ``-keep_mask_deletion_edits``
transfer: (an integer (int or long))
  transfer only voxel value # from mask to out
  argument: ``-transfer %d``
subjects_dir: (a pathlike object or string representing an existing directory)
  subjects directory
args: (a unicode string)
  Additional parameters to the command
  argument: ``\%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
```
Environment variables
Outputs:

```
out_file: (a pathlike object or string representing an existing file)
    masked image
```

### 62.5.5 CheckTalairachAlignment

**Link to code**

Wraps the executable command `talairach_afd`. This program detects Talairach alignment failures

**Examples**

```python
>>> from nipype.interfaces.freesurfer import CheckTalairachAlignment

>>> checker = CheckTalairachAlignment()

>>> checker.inputs.in_file = 'trans.mat'

>>> checker.inputs.threshold = 0.005

>>> checker.cmdline
'talairach_afd -T 0.005 -xfm trans.mat'

>>> checker.run() # doctest: +SKIP
```

**Inputs:**

- **Mandatory**
  - `in_file`: (a pathlike object or string representing an existing file)
    - specify the `talairach.xfm` file to check
  - `subject`: (a string)
    - specify subject’s name

- **Optional**
  - `threshold`: (a float, nipype default value: 0.01)
    - Talairach transforms for subjects with p-values <= T are considered as very unlikely default=0.010
  - `subjects_dir`: (a pathlike object or string representing an existing directory)
    - subjects directory
  - `args`: (a unicode string)
    - Additional parameters to the command
  - `environ`: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

**Outputs:**

```
out_file: (a pathlike object or string representing an existing file)
    The input file for CheckTalairachAlignment
```
62.5.6 Contrast

Link to code

Wraps the executable command `pctsurfcon`.

Compute surface-wise gray/white contrast

Examples

```python
>>> from nipype.interfaces.freesurfer import Contrast
>>> contrast = Contrast()
>>> contrast.inputs.subject_id = '10335'
>>> contrast.inputs.hemisphere = 'lh'
>>> contrast.inputs.white = 'lh.white'  # doctest: +SKIP
>>> contrast.inputs.thickness = 'lh.thickness'  # doctest: +SKIP
>>> contrast.inputs.annotation = '../label/lh.aparc.annot'  # doctest: +SKIP
>>> contrast.inputs.cortex = '../label/lh.cortex.label'  # doctest: +SKIP
>>> contrast.inputs.rawavg = '../mri/rawavg.mgz'  # doctest: +SKIP
>>> contrast.inputs.orig = '../mri/orig.mgz'  # doctest: +SKIP
>>> contrast.cmdline  # doctest: +SKIP
'pctsurfcon --lh-only --s 10335'
```

Inputs:

[Mandatory]
subject_id: (a string, nipype default value: subject_id)

  Subject being processed
  argument: `--s %s`

hemisphere: ('lh' or 'rh')

  Hemisphere being processed
  argument: `--%s-only`

thickness: (a pathlike object or string representing an existing file)

  Input file must be `<subject_id>/surf/?h.thickness`

white: (a pathlike object or string representing an existing file)

  Input file must be `<subject_id>/surf/<hemisphere>.white`

annotation: (a pathlike object or string representing an existing file)

  Input annotation file must be `<subject_id>/label/<hemisphere>.aparc.annot`

cortex: (a pathlike object or string representing an existing file)

  Input cortex label must be `<subject_id>/label/<hemisphere>.cortex.label`

orig: (a pathlike object or string representing an existing file)

  Implicit input file mri/orig.mgz

rawavg: (a pathlike object or string representing an existing file)

  Implicit input file mri/rawavg.mgz

[Optional]
copy_inputs: (a boolean)

  If running as a node, set this to True. This will copy the input files to the node directory.

subjects_dir: (a pathlike object or string representing an existing directory)

  subjects directory

args: (a unicode string)

  Additional parameters to the command
  argument: `''`

eviron: (a dictionary with keys which are a bytes or None or a value

(continues on next page)
of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}.

Environment variables

Outputs:

out_contrast: (a pathlike object or string representing a file)
  Output contrast file from Contrast

out_stats: (a pathlike object or string representing a file)
  Output stats file from Contrast

out_log: (a pathlike object or string representing an existing file)
  Output log from Contrast

62.5.7 Curvature

Link to code
Wraps the executable command mris_curvature.
This program will compute the second fundamental form of a cortical surface. It will create two new files <hemi>.<surface>.H and <hemi>.<surface>.K with the mean and Gaussian curvature respectively.

Examples

```python
>>> from nipype.interfaces.freesurfer import Curvature
>>> curv = Curvature()
>>> curv.inputs.in_file = 'lh.pial'
>>> curv.inputs.save = True
>>> curv.cmdline
'mris_curvature -w lh.pial'
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  Input file for Curvature
  argument: "%s", position: -2

[Optional]
threshold: (a float)
  Undocumented input threshold
  argument: "-thresh %.3f"

n: (a boolean)
  Undocumented boolean flag
  argument: "-n"

averages: (an integer (int or long))
  Perform this number iterative averages of curvature measure before saving
  argument: "-a %d"

save: (a boolean)
  Save curvature files (will only generate screen output without this option)
  argument: "-w"

distances: (a tuple of the form: (an integer (int or long), an integer (int or long)))
  Undocumented input integer distances
  argument: "-distances %d %d"

copy_input: (a boolean)
Copy input file to current directory
subjects_dir: (a pathlike object or string representing an existing
directory)
subjects directory
args: (a unicode string)
    Additional parameters to the command
    argument: '%s'
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
    Environment variables

Outputs:
out_mean: (a pathlike object or string representing a file)
    Mean curvature output file
out_gauss: (a pathlike object or string representing a file)
    Gaussian curvature output file

### 62.5.8 CurvatureStats

Link to code
Wraps the executable command `mris_curvature_stats`. In its simplest usage, `mris_curvature_stats` will compute a set of statistics on its input `<curvFile>`. These statistics are the mean and standard deviation of the particular curvature on the surface, as well as the results from several surface-based integrals. Additionally, `mris_curvature_stats` can report the max/min curvature values, and compute a simple histogram based on all curvature values. Curvatures can also be normalised and constrained to a given range before computation. Principal curvature (K, H, k1 and k2) calculations on a surface structure can also be performed, as well as several functions derived from k1 and k2. Finally, all output to the console, as well as any new curvatures that result from the above calculations can be saved to a series of text and binary-curvature files.

#### Examples

```python
>>> from nipype.interfaces.freesurfer import CurvatureStats
>>> curvstats = CurvatureStats()
>>> curvstats.inputs.hemisphere = 'lh'
>>> curvstats.inputs.curvfile1 = 'lh.pial'
>>> curvstats.inputs.curvfile2 = 'lh.pial'
>>> curvstats.inputs.surface = 'lh.pial'
>>> curvstats.inputs.out_file = 'lh.curv.stats'
>>> curvstats.inputs.values = True
>>> curvstats.inputs.min_max = True
>>> curvstats.inputs.write = True
>>> curvstats.cmdline
'mris_curvature_stats -m -o lh.curv.stats -F pial -G --writeCurvatureFiles
--subject_id lh pial pial'
```

Inputs:

[Mandatory]
curvfile1: (a pathlike object or string representing an existing file)
**Input file for CurvatureStats**
- **argument**: `'%s'`, **position**: -2
  - **curvfile2**: (a pathlike object or string representing an existing file)
- **argument**: `'%s'`, **position**: -1
  - **hemisphere**: ('lh' or 'rh')
- **subject_id**: (a string, nipype default value: subject_id)
  - **subject being processed**
    - **argument**: `'%s'`, **position**: -3
- **subject_id**: (a string, nipype default value: subject_id)
  - **subject being processed**
    - **argument**: `'%s'`, **position**: -4

**[Optional]**
- **surface**: (a pathlike object or string representing an existing file)
  - **Specify surface file for CurvatureStats**
    - **argument**: `'-F %s'`
- **out_file**: (a pathlike object or string representing a file)
  - **Output curvature stats file**
    - **argument**: `'-o %s'`
- **min_max**: (a boolean)
  - **Output min / max information for the processed curvature.**
    - **argument**: `'-m'`
- **values**: (a boolean)
  - **Triggers a series of derived curvature values**
    - **argument**: `'-G'`
- **write**: (a boolean)
  - **Write curvature files**
    - **argument**: `'--writeCurvatureFiles'`
- **copy_inputs**: (a boolean)
  - **If running as a node, set this to True. This will copy the input files to the node directory.**
- **subjects_dir**: (a pathlike object or string representing an existing directory)
  - **subjects directory**
- **args**: (a unicode string)
  - **Additional parameters to the command**
    - **argument**: `'%s'`
- **environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) 
  - **Environment variables**

**Outputs:**
- **out_file**: (a pathlike object or string representing a file)
  - **Output curvature stats file**

### 62.5.9 EulerNumber

**Link to code**

Wraps the executable command `mris_euler_number`. This program computes EulerNumber for a cortical surface
Examples

```python
>>> from nipype.interfaces.freesurfer import EulerNumber
>>> ft = EulerNumber()
>>> ft.inputs.in_file = 'lh.pial'
>>> ft.cmdline
'mris_euler_number lh.pial'
```

Inputs:

- **in_file**: (a pathlike object or string representing an existing file)
  - Input file for EulerNumber
  - argument: `'\%s'`, position: -1
- **subjects_dir**: (a pathlike object or string representing an existing directory)
  - subjects directory
- **args**: (a unicode string)
  - Additional parameters to the command
  - argument: `'\%s'`
- **environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: `{}`)
  - Environment variables

Outputs:

- **out_file**: (a pathlike object or string representing a file)
  - Output file for EulerNumber

62.5.10 ExtractMainComponent

Link to code

Wraps the executable command `mris_extract_main_component`. Extract the main component of a tesselated surface

Examples

```python
>>> from nipype.interfaces.freesurfer import ExtractMainComponent
>>> mcmp = ExtractMainComponent(in_file='lh.pial')
>>> mcmp.cmdline
'mris_extract_main_component lh.pial lh.maincmp'
```

Inputs:

- **in_file**: (a pathlike object or string representing an existing file)
  - input surface file
  - argument: `'\%s'`, position: 1
- **out_file**: (a pathlike object or string representing a file)
  - surface containing main component
  - argument: `'\%s'`, position: 2
- **args**: (a unicode string)
Additional parameters to the command argument: `''%s''`
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {}) Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
surface containing main component

62.5.11 FixTopology

Link to code
Wraps the executable command `mris_fix_topology`. This program computes a mapping from the unit sphere onto the surface of the cortex from a previously generated approximation of the cortical surface, thus guaranteeing a topologically correct surface.

Examples

```python
>>> from nipype.interfaces.freesurfer import FixTopology
>>> ft = FixTopology()
>>> ft.inputs.in_orig = 'lh.orig' # doctest: +SKIP
>>> ft.inputs.in_inflated = 'lh.inflated' # doctest: +SKIP
>>> ft.inputs.sphere = 'lh.qsphere.nofix' # doctest: +SKIP
>>> ft.inputs.hemisphere = 'lh'
>>> ft.inputs.subject_id = '10335'
>>> ft.inputs.mgz = True
>>> ft.inputs.ga = True
>>> ft.cmdline # doctest: +SKIP
'mris_fix_topology -ga -mgz -sphere qsphere.nofix 10335 lh'
```

Inputs:

[Mandatory]
in_orig: (a pathlike object or string representing an existing file)
    Undocumented input file `<hemisphere>.orig`
in_inflated: (a pathlike object or string representing an existing file)
    Undocumented input file `<hemisphere>.inflated`
in_brain: (a pathlike object or string representing an existing file)
    Implicit input `brain.mgz`
in_wm: (a pathlike object or string representing an existing file)
    Implicit input `wm.mgz`
hemisphere: (a string)
    Hemisphere being processed
    argument: `''%s''`, position: -1
subject_id: (a string, nipype default value: subject_id)
    Subject being processed
    argument: `''%s''`, position: -2
copy_inputs: (a boolean)
    If running as a node, set this to True otherwise, the topology fixing will be done in place.

[Optional]
seed: (an integer (int or long))
    Seed for setting random number generator
    argument: `'--seed %d'`

ga: (a boolean)
    No documentation. Direct questions to analysis-
    bugs@nmr.mgh.harvard.edu
    argument: `'-ga'`

ggz: (a boolean)
    No documentation. Direct questions to analysis-
    bugs@nmr.mgh.harvard.edu
    argument: `'-mgz'`

sphere: (a pathlike object or string representing a file)
    Sphere input file
    argument: `'-sphere %s'`

subjects_dir: (a pathlike object or string representing an existing
    directory)
    subjects directory

args: (a unicode string)
    Additional parameters to the command
    argument: `'%s'`

environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})
    Environment variables

Outputs:

out_file: (a pathlike object or string representing a file)
    Output file for FixTopology

62.5.12 Jacobian

Link to code

Wraps the executable command mris_jacobian.
This program computes the Jacobian of a surface mapping.

Examples

```python
>>> from nipype.interfaces.freesurfer import Jacobian
>>> jacobian = Jacobian()
>>> jacobian.inputs.in_origsurf = 'lh.pial'
>>> jacobian.inputs.in_mappedsurf = 'lh.pial'
>>> jacobian.cmdline
'mris_jacobian lh.pial lh.pial lh.jacobian'
```

Inputs:

[Mandatory]
in_origsurf: (a pathlike object or string representing an existing
    file)
    Original surface
    argument: `'%s'`, position: -3

in_mappedsurf: (a pathlike object or string representing an existing
    file)
    Mapped surface
    argument: `'%s'`, position: -2
[Optional]
out_file: (a pathlike object or string representing a file)
  Output Jacobian of the surface mapping
  argument: `\%s`, position: -1
subjects_dir: (a pathlike object or string representing an existing
directory)
  subjects directory
args: (a unicode string)
  Additional parameters to the command
  argument: `\%s`
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
  Environment variables

Outputs:

out_file: (a pathlike object or string representing a file)
  Output Jacobian of the surface mapping

62.5.13 LTAConvert

Link to code
Wraps the executable command lta_convert.
Convert different transformation formats. Some formats may require you to pass an image if the geometry
information is missing form the transform file format.
For complete details, see the lta_convert documentation.

Inputs:

[Mandatory]
in_lta: (a pathlike object or string representing an existing file or
  'identity.nofile')
  input transform of LTA type
  argument: `--inlta %s`
  mutually_exclusive: in_lta, in_fsl, in_mni, in_reg, in_niftyreg, in_itk
in_fsl: (a pathlike object or string representing an existing file)
  input transform of FSL type
  argument: `--infsl %s`
  mutually_exclusive: in_lta, in_fsl, in_mni, in_reg, in_niftyreg, in_itk
in_mni: (a pathlike object or string representing an existing file)
  input transform of MNI/XFM type
  argument: `--inmni %s`
  mutually_exclusive: in_lta, in_fsl, in_mni, in_reg, in_niftyreg, in_itk
in_reg: (a pathlike object or string representing an existing file)
  input transform of TK REG type (deprecated format)
  argument: `--inreg %s`
  mutually_exclusive: in_lta, in_fsl, in_mni, in_reg, in_niftyreg, in_itk
in_niftyreg: (a pathlike object or string representing an existing
  file)
  input transform of Nifty Reg type (inverse RAS2RAS)
  argument: `--inniftyreg %s`

(continues on next page)
mutually_exclusive: in_lta, in_fsl, in_mni, in_reg, in_niftyreg, in_itk

in_itk: (a pathlike object or string representing an existing file)
input transform of ITK type
argument: --initk %s
mutually_exclusive: in_lta, in_fsl, in_mni, in_reg, in_niftyreg, in_itk

[Optional]
out_lta: (a boolean or a pathlike object or string representing a file)
output linear transform (LTA Freesurfer format)
argument: --outlta %s
out_fsl: (a boolean or a pathlike object or string representing a file)
output transform in FSL format
argument: --outfsl %s
out_mni: (a boolean or a pathlike object or string representing a file)
output transform in MNI/XFM format
argument: --outmni %s
out_reg: (a boolean or a pathlike object or string representing a file)
output transform in reg dat format
argument: --outreg %s
out_itk: (a boolean or a pathlike object or string representing a file)
output transform in ITK format
argument: --outitk %s
invert: (a boolean)
argument: --invert
ltavox2vox: (a boolean)
argument: --ltavox2vox
requires: out_lta
source_file: (a pathlike object or string representing an existing file)
argument: --src %s
target_file: (a pathlike object or string representing an existing file)
argument: --trg %s
target_conform: (a boolean)
argument: --trgconform
args: (a unicode string)
Additional parameters to the command
argument: %s
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_lta: (a pathlike object or string representing an existing file)
output linear transform (LTA Freesurfer format)
out_fsl: (a pathlike object or string representing an existing file)
output transform in FSL format
out_mni: (a pathlike object or string representing an existing file)
output transform in MNI/XFM format
out_reg: (a pathlike object or string representing an existing file)
output transform in reg dat format
out_itk: (a pathlike object or string representing an existing file)
output transform in ITK format

62.5.14 MRIFill

Link to code
Wraps the executable command mri_fill.
This program creates hemispheric cutting planes and fills white matter with specific values for subsequent surface tessellation.

Examples

```python
>>> from nipype.interfaces.freesurfer import MRIFill
>>> fill = MRIFill()
>>> fill.inputs.in_file = 'wm.mgz' # doctest: +SKIP
>>> fill.inputs.out_file = 'filled.mgz' # doctest: +SKIP
>>> fill.cmdline # doctest: +SKIP
'mri_fill wm.mgz filled.mgz'
```

Inputs:

- **in_file**: (a pathlike object or string representing an existing file)
  
  Input white matter file
  
  argument: ``-i %s``, position: -2

- **out_file**: (a pathlike object or string representing a file)
  
  Output filled volume file name for MRIFill
  
  argument: ``-o %s``, position: -1

- **segmentation**: (a pathlike object or string representing an existing file)
  
  Input segmentation file for MRIFill
  
  argument: ``-segmentation %s``

- **transform**: (a pathlike object or string representing an existing file)
  
  Input transform file for MRIFill
  
  argument: ``-xform %s``

- **log_file**: (a pathlike object or string representing a file)
  
  Output log file for MRIFill
  
  argument: ``-a %s``

- **subjects_dir**: (a pathlike object or string representing an existing directory)

- **args**: (a unicode string)
  
  Additional parameters to the command
  
  argument: ``-%s``

- **environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

Outputs:
62.5.15 MRIMarchingCubes

Link to code
Wraps the executable command mri_mc. Uses Freesurfer’s mri_mc to create surfaces by tessellating a given input volume.

Example

```python
>>> import nipype.interfaces.freesurfer as fs
>>> mc = fs.MRIMarchingCubes()
>>> mc.inputs.in_file = 'aseg.mgz'
>>> mc.inputs.label_value = 17
>>> mc.inputs.out_file = 'lh.hippocampus'
>>> mc.run()  # doctest: +SKIP
```

Inputs:

- **in_file**: (a pathlike object or string representing an existing file)
  - Description: Input volume to tesselate voxels from.
  - Argument: ``%s``, position: 1

- **label_value**: (an integer (int or long))
  - Description: Label value which to tesselate from the input volume. (integer, if input is "filled.mgz" volume, 127 is rh, 255 is lh)
  - Argument: ``%d``, position: 2

[Optional]

- **connectivity_value**: (an integer (int or long), nipype default value: 1)
  - Description: Alter the marching cubes connectivity: 1=6+, 2=18, 3=6, 4=26 (default=1)
  - Argument: ``%d``, position: -1

- **out_file**: (a pathlike object or string representing a file)
  - Description: Output filename or True to generate one
  - Argument: ``./%s``, position: -2

- **subjects_dir**: (a pathlike object or string representing an existing directory)
  - Description: Subjects directory

- **args**: (a unicode string)
  - Description: Additional parameters to the command
  - Argument: ``%s``

- **environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

- **surface**: (a pathlike object or string representing an existing file)
  - Description: Binary surface of the tessellation
62.5.16 MRIPretess

Link to code
Wraps the executable command mri_pretess. Uses Freesurfer's mri_pretess to prepare volumes to be tessellated.

Description
Changes white matter (WM) segmentation so that the neighbors of all voxels labeled as WM have a face in common - no edges or corners allowed.

Example

```python
>>> import nipype.interfaces.freesurfer as fs
>>> pretess = fs.MRIPretess()
>>> pretess.inputs.in_filled = 'wm.mgz'
>>> pretess.inputs.in_norm = 'norm.mgz'
>>> pretess.inputs.nocorners = True
>>> pretess.cmdline
'mri_pretess -nocorners wm.mgz wm norm.mgz wm_pretesswm.mgz'
>>> pretess.run() # doctest: +SKIP
```

Inputs:

**[Mandatory]**

- `in_filled`: (a pathlike object or string representing an existing file)
  filled volume, usually `wm.mgz`
  argument: `"%s"`, position: -4
- `label`: (a unicode string or an integer (int or long), nipype default value: `wm`)
  label to be picked up, can be a Freesurfer's string like `"wm"` or a label value (e.g. 127 for rh or 255 for lh)
  argument: `"%s"`, position: -3
- `in_norm`: (a pathlike object or string representing an existing file)
  the normalized, brain-extracted T1w image. Usually `norm.mgz`
  argument: `"%s"`, position: -2

**[Optional]**

- `out_file`: (a pathlike object or string representing a file)
  the output file after mri_pretess.
  argument: `"%s"`, position: -1
- `nocorners`: (a boolean)
  do not remove corner configurations in addition to edge ones.
  argument: `"-nocorners"`
- `keep`: (a boolean)
  keep WM edits
  argument: `"-keep"`
- `test`: (a boolean)
  adds a voxel that should be removed by mri_pretess. The value of the voxel is set to that of an ON-edited WM, so it should be kept with `-keep`. The output will NOT be saved.
  argument: `"-test"`
- `subjects_dir`: (a pathlike object or string representing an existing directory)
  subjects directory
- `args`: (a unicode string)
  Additional parameters to the command

(continues on next page)
argument: `\%s`  
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})  
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)  
output file after mri_pretess

62.5.17 MRITessellate

Link to code
Wraps the executable command mri_tessellate.  
Uses Freesurfer’s mri_tessellate to create surfaces by tessellating a given input volume

Example

```python
>>> import nipype.interfaces.freesurfer as fs
>>> tess = fs.MRITessellate()
>>> tess.inputs.in_file = 'aseg.mgz'
>>> tess.inputs.label_value = 17
>>> tess.inputs.out_file = 'lh.hippocampus'
>>> tess.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)  
Input volume to tesselate voxels from.  
argument: `\%s`, position: -3

label_value: (an integer (int or long))  
Label value which to tesselate from the input volume. (integer, if input is "filled.mgz" volume, 127 is rh, 255 is lh)  
argument: `\%d`, position: -2

[Optional]
out_file: (a pathlike object or string representing a file)  
output filename or True to generate one  
argument: `\%s`, position: -1
tesselate_all_voxels: (a boolean)  
Tessellate the surface of all voxels with different labels  
argument: `\-a`
use_real_RAS_coordinates: (a boolean)  
Saves surface with real RAS coordinates where c_(r,a,s) != 0  
argument: `\-n`
subjects_dir: (a pathlike object or string representing an existing directory)  
subjects directory
args: (a unicode string)  
Additional parameters to the command  
argument: `\%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
62.5.18 MRIsCalc

Link to code
Wraps the executable command mriscalc.
mris_calc is a simple calculator that operates on FreeSurfer curvatures and volumes. In most cases, the calculator functions with three arguments: two inputs and an <ACTION> linking them. Some actions, however, operate with only one input <file1>. In all cases, the first input <file1> is the name of a FreeSurfer curvature overlay (e.g. rh.curv) or volume file (e.g. orig.mgz). For two inputs, the calculator first assumes that the second input is a file. If, however, this second input file doesn’t exist, the calculator assumes it refers to a float number, which is then processed according to <ACTION>. Note: <file1> and <file2> should typically be generated on the same subject.

Examples

```python
>>> from nipype.interfaces.freesurfer import MRIsCalc
>>> example = MRIsCalc()
>>> example.inputs.in_file1 = 'lh.area' # doctest: +SKIP
>>> example.inputs.in_file2 = 'lh.area.pial' # doctest: +SKIP
>>> example.inputs.action = 'add'
>>> example.inputs.out_file = 'area.mid'
>>> example.cmdline # doctest: +SKIP
'mris_calc -o lh.area.mid lh.area add lh.area.pial'
```

Inputs:

- `in_file1`: (a pathlike object or string representing an existing file)
  - Input file 1
  - argument: ``%s``, position: -3
- `in_file2`: (a pathlike object or string representing an existing file)
  - Input file 2
  - argument: ``%s``, position: -1
  - mutually_exclusive: in_float, in_int
- `in_float`: (a float)
  - Input float
  - argument: ``%f``, position: -1
  - mutually_exclusive: in_file2, in_int
- `in_int`: (an integer (int or long))
  - Input integer
argument: `\`%d``', position: -1
mutually_exclusive: in_file2, in_float
subjects_dir: (a pathlike object or string representing an existing
directory)
subjects directory
args: (a unicode string)
    Additional parameters to the command
argument: `\`%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})
    Environment variables

Outputs:

out_file: (a pathlike object or string representing a file)
    Output file after calculation

62.5.19 MRIsCombine

Link to code
Wraps the executable command `mris_convert`.
Uses Freesurfer's `mris_convert` to combine two surface files into one.
For complete details, see the `mris_convert` Documentation.
If given an `out_file` that does not begin with 'lh.' or 'rh.', `mris_convert` will prepend 'lh.' to
the file name. To avoid this behavior, consider setting `out_file = './<filename>'`, or leaving `out_file`
blank.
In a Node/Workflow, `out_file` is interpreted literally.

Example

```python
>>> import nipype.interfaces.freesurfer as fs
>>> mris = fs.MRIsCombine()
>>> mris.inputs.in_files = ['lh.pial', 'rh.pial']
>>> mris.inputs.out_file = 'bh.pial'
>>> mris.cmdline
'mris_convert --combinesurfs lh.pial rh.pial bh.pial'
>>> mris.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_files: (a list of from 2 to 2 items which are a pathlike object or
    string representing a file)
    Two surfaces to be combined.
    argument: `\`--combinesurfs %s\`', position: 1
out_file: (a pathlike object or string representing a file)
    Output filename. Combined surfaces from in_files.
    argument: `\`%s\`', position: -1

[Optional]
subjects_dir: (a pathlike object or string representing an existing
directory)
    subjects directory
args: (a unicode string)
Additional parameters to the command
argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
    Output filename. Combined surfaces from in_files.

62.5.20 MRIsConvert

Link to code
Wraps the executable command mris_convert.
Uses Freesurfer’s mris_convert to convert surface files to various formats

Example

```python
>>> import nipype.interfaces.freesurfer as fs
>>> mris = fs.MRIsConvert()
>>> mris.inputs.in_file = 'lh.pial'
>>> mris.inputs.out_datatype = 'gii'
>>> mris.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
    File to read/convert
    argument: ``%s``, position: -2
out_file: (a pathlike object or string representing a file)
    output filename or True to generate one
    argument: ``%s``, position: -1
mutually_exclusive: out_datatype
out_datatype: ('asc' or 'ico' or 'tri' or 'stl' or 'vtk' or 'gii' or
    'mgh' or 'mgz')
    These file formats are supported: ASCII: .ascICO: .ico, .tri GEO:
    .geo STL: .stl VTK: .vtk GIFTI: .gii MGH surface-encoded 'volume':
    .mgh, .mgz
    mutually_exclusive: out_file

[Optional]
annot_file: (a pathlike object or string representing an existing file)
    input is annotation or gifti label data
    argument: ``--annot %s``
parcstats_file: (a pathlike object or string representing an existing file)
    infile is name of text file containing label/val pairs
    argument: ``--parcstats %s``
label_file: (a pathlike object or string representing an existing file)
    infile is .label file, label is name of this label
    argument: ``--label %s``

(continues on next page)
scalarcurv_file: (a pathlike object or string representing an existing file)
  input is scalar curv overlay file (must still specify surface)
  argument: '``-c %s``'

functional_file: (a pathlike object or string representing an existing file)
  input is functional time-series or other multi-frame data (must specify surface)
  argument: '``-f %s``'

labelstats_outfile: (a pathlike object or string representing a file)
  outfile is name of gifti file to which label stats will be written
  argument: '``--labelstats %s``'

patch: (a boolean)
  input is a patch, not a full surface
  argument: '``-p``'

rescale: (a boolean)
  rescale vertex xyz so total area is same as group average
  argument: '``-r``'

normal: (a boolean)
  output is an ascii file where vertex data
  argument: '``-n``'

xyz_ascii: (a boolean)
  Print only surface xyz to ascii file
  argument: '``-a``'

vertex: (a boolean)
  Writes out neighbors of a vertex in each row
  argument: '``-v``'

scale: (a float)
  scale vertex xyz by scale
  argument: '``-s %.3f``'

dataarray_num: (an integer (int or long))
  if input is gifti, 'num' specifies which data array to use
  argument: '``--da_num %d``'

talairachxfm_subjid: (a string)
  apply talairach xfm of subject to vertex xyz
  argument: '``-t %s``'

origname: (a string)
  read orig positions
  argument: '``-o %s``'

to_scan: (a boolean)
  convert coordinates from native FS (tkr) coords to scanner coords
  argument: '``--to-scanner``'

to_tkr: (a boolean)
  convert coordinates from scanner coords to native FS (tkr) coords
  argument: '``--to-tkr``'

subjects_dir: (a pathlike object or string representing an existing directory)
  subjects directory

args: (a unicode string)
  Additional parameters to the command
  argument: '``%s``'

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  Environment variables

Outputs:
62.5.21 MRIsExpand

Link to code
Wraps the executable command `mris_expand`.
Expands a surface (typically `?h.white`) outwards while maintaining smoothness and self-intersection constraints.

Examples

```python
>>> from nipype.interfaces.freesurfer import MRIsExpand
>>> mris_expand = MRIsExpand(thickness=True, distance=0.5)
>>> mris_expand.inputs.in_file = 'lh.white'
>>> mris_expand.cmdline
'mris_expand -thickness lh.white 0.5 expanded'
>>> mris_expand.inputs.out_name = 'graymid'
>>> mris_expand.cmdline
'mris_expand -thickness lh.white 0.5 graymid'
```

Inputs:

[Mandatory]
- **in_file**: (a pathlike object or string representing an existing file)
  - Surface to expand
    - argument: `"%s"`, position: -3
- **distance**: (a float)
  - Distance in mm or fraction of cortical thickness
    - argument: `"%g"`, position: -2

[Optional]
- **out_name**: (a unicode string, nipype default value: expanded)
  - Output surface file
    - If no path, uses directory of `in_file`
    - If no path AND missing "lh." or "rh.", derive from `in_file`
    - argument: `"%s"`, position: -1
- **thickness**: (a boolean)
  - Expand by fraction of cortical thickness, not mm
    - argument: `"-thickness"`
- **thickness_name**: (a unicode string)
  - Name of thickness file (implicit: "thickness")
    - If no path, uses directory of `in_file`
    - If no path AND missing "lh." or "rh.", derive from `in_file`
    - argument: `"-thickness_name %s"`
- **pial**: (a unicode string)
  - Name of pial file (implicit: "pial")
    - If no path, uses directory of `in_file`
    - If no path AND missing "lh." or "rh.", derive from `in_file`
    - argument: `"-pial %s"`
- **sphere**: (a unicode string, nipype default value: sphere)
  - WARNING: Do not change this trait
- **spring**: (a float)
  - Spring term (implicit: 0.05)
    - argument: `"-S %g"`
- **dt**: (a float)
```
(continues on next page)
dt (implicit: 0.25)
argument: ``-T %g``

write_iterations: (an integer (int or long))
Write snapshots of expansion every N iterations
argument: ``-W %d``

smooth_averages: (an integer (int or long))
Smooth surface with N iterations after expansion
argument: ``-A %d``

nsurfaces: (an integer (int or long))
Number of surfacces to write during expansion
argument: ``-N %d``

subjects_dir: (a pathlike object or string representing an existing
directory)
subjects directory
args: (a unicode string)
Additional parameters to the command
argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing a file)
Output surface file

62.5.22 MRIsInflate

Link to code
Wraps the executable command mris_inflate.
This program will inflate a cortical surface.

Examples

```python
>>> from nipype.interfaces.freesurfer import MRIsInflate
>>> inflate = MRIsInflate()
>>> inflate.inputs.in_file = 'lh.pial'
>>> inflate.inputs.no_save_sulc = True
>>> inflate.cmdline
# doctest: +SKIP
'mris_inflate -no-save-sulc lh.pial lh.inflated'
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
Input file for MRIsInflate
argument: ````, position: -2

[Optional]
out_file: (a pathlike object or string representing a file)
Output file for MRIsInflate
argument: ````, position: -1
out_sulc: (a pathlike object or string representing a file)
Output sulc file
mutually_exclusive: no_save_sulc

(continues on next page)
no_save_sulc: (a boolean)
    Do not save sulc file as output
    argument: `--no-save-sulc`
    mutually_exclusive: out_sulc
subjects_dir: (a pathlike object or string representing an existing
directory)
    subjects directory
args: (a unicode string)
    Additional parameters to the command
    argument: `%%s`
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})
    Environment variables

Outputs:

out_file: (a pathlike object or string representing a file)
    Output file for MRIsInflate
out_sulc: (a pathlike object or string representing a file)
    Output sulc file

62.5.23 MakeAverageSubject

Link to code
Wraps the executable command make_average_subject.
Make an average freesurfer subject

Examples

```python
>>> from nipype.interfaces.freesurfer import MakeAverageSubject
>>> avg = MakeAverageSubject(subjects_ids=['s1', 's2'])
>>> avg.cmdline
'make_average_subject --out average --subjects s1 s2'
```

Inputs:

[Mandatory]
subjects_ids: (a list of items which are a unicode string)
    freesurfer subjects ids to average
    argument: `--subjects %s`

[Optional]
out_name: (a pathlike object or string representing a file, nipype
    default value: average)
    name for the average subject
    argument: `--out %s`
subjects_dir: (a pathlike object or string representing an existing
directory)
    subjects directory
args: (a unicode string)
    Additional parameters to the command
    argument: `%%s`
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})

Environment variables

Outputs:

average_subject_name: (a unicode string)

Output registration file

### 62.5.24 MakeSurfaces

**Link to code**

Wraps the executable command `mris_make_surfaces`.

This program positions the tessellation of the cortical surface at the white matter surface, then the gray matter surface and generate surface files for these surfaces as well as a ‘curvature’ file for the cortical thickness, and a surface file which approximates layer IV of the cortical sheet.

**Examples**

```python
>>> from nipype.interfaces.freesurfer import MakeSurfaces
>>> makesurfaces = MakeSurfaces()
>>> makesurfaces.inputs.hemisphere = 'lh'
>>> makesurfaces.inputs.subject_id = '10335'
>>> makesurfaces.inputs.in_orig = 'lh.pial'
>>> makesurfaces.inputs.in_wm = 'wm.mgz'
>>> makesurfaces.inputs.in_filled = 'norm.mgz'
>>> makesurfaces.inputs.in_label = 'aparc+aseg.nii'
>>> makesurfaces.inputs.in_T1 = 'T1.mgz'
>>> makesurfaces.inputs.orig_pial = 'lh.pial'
>>> makesurfaces.cmdline
'mris_make_surfaces -T1 T1.mgz -orig pial -orig_pial pial 10335 lh'
```

**Inputs:**

[Mandatory]

- **hemisphere**: ('lh' or 'rh')
  - Hemisphere being processed
  - argument: `''%s''`, position: -1

- **subject_id**: (a string, nipype default value: subject_id)
  - Subject being processed
  - argument: `''%s''`, position: -2

- **in_orig**: (a pathlike object or string representing an existing file)
  - Implicit input file `<hemisphere>.orig`
  - argument: `''-orig %s''`

- **in_wm**: (a pathlike object or string representing an existing file)
  - Implicit input file `wm.mgz`

- **in_filled**: (a pathlike object or string representing an existing file)
  - Implicit input file `filled.mgz`

[Optional]

- **in_white**: (a pathlike object or string representing an existing file)
  - Implicit input that is sometimes used

- **in_label**: (a pathlike object or string representing an existing file)
  - Implicit input label/`<hemisphere>.aparc.annot`
  - mutually_exclusive: noaparc

- **orig_white**: (a pathlike object or string representing an existing file)
Specify a white surface to start with argument: ``-orig_white %s``

Specify a pial surface to start with argument: ``-orig_pial %s`` requires: in_label

Undocumented flag argument: ``-fix_mtl``

Undocumented flag argument: ``-nowhite``

Undocumented flag argument: ``-whiteonly``

Input segmentation file argument: ``-aseg %s``

Input brain or T1 file argument: ``-T1 %s``

No documentation. Direct questions to analysis-bugs@nmr.mgh.harvard.edu argument: ``-mgz``

No documentation. Direct questions to analysis-bugs@nmr.mgh.harvard.edu argument: ``-noaparc``

maximum: (a float)

No documentation (used for longitudinal processing) argument: ``-max %.1f``

No documentation (used for longitudinal processing) argument: ``-long``

White surface name argument: ``-white %s``

If running as a node, set this to True. This will copy the input files to the node directory.

Subjects directory argument: ``%s``

Additional parameters to the command argument: ``%s``

Environment variables

Outputs:
out_white: (a pathlike object or string representing a file)
  Output white matter hemisphere surface
out_curv: (a pathlike object or string representing a file)
  Output curv file for MakeSurfaces
out_area: (a pathlike object or string representing a file)
  Output area file for MakeSurfaces
out_cortex: (a pathlike object or string representing a file)
  Output cortex file for MakeSurfaces
out_pial: (a pathlike object or string representing a file)
  Output pial surface for MakeSurfaces
out_thickness: (a pathlike object or string representing a file)
  Output thickness file for MakeSurfaces

62.5.25 ParcellationStats

Link to code
Wraps the executable command mris_anatomical_stats.
This program computes a number of anatomical properties.

Examples

```python
>>> from nipype.interfaces.freesurfer import ParcellationStats
>>> import os

>>> parcstats = ParcellationStats()
>>> parcstats.inputs.subject_id = '10335'
>>> parcstats.inputs.hemisphere = 'lh'
>>> parcstats.inputs.transform = './../mri/transforms/talairach.xfm'
>>> parcstats.inputs.brainmask = './../mri/brainmask.mgz'
>>> parcstats.inputs.aseg = './../mri/aseg.presurf.mgz'
>>> parcstats.inputs.ribbon = './../mri/ribbon.mgz'
>>> parcstats.inputs.lh_pial = 'lh.pial'
>>> parcstats.inputs.lh_white = 'lh.white'
>>> parcstats.inputs.thickness = 'lh.thickness'
>>> parcstats.inputs.out_table = 'lh.test.stats'
>>> parcstats.inputs.out_color = 'test.ctab'

'./mri/transforms/talairach.xfm' # doctest: +SKIP

'mris_anatomical_stats -c test.ctab -f lh.test.stats 10335 lh white'
```

Inputs:

[Mandatory]
subject_id: (a string, nipype default value: subject_id)
  Subject being processed
    argument: ``%s``, position: -3
hemisphere: ('lh' or 'rh')
  Hemisphere being processed
    argument: ``%s``, position: -2
wm: (a pathlike object or string representing an existing file)
  Input file must be <subject_id>/mri/wm.mgz
lh_white: (a pathlike object or string representing an existing file)
  Input file must be <subject_id>/surf/lh.white

(continues on next page)
rh_white: (a pathlike object or string representing an existing file)
   Input file must be <subject_id>/surf/rh.white
lh_pial: (a pathlike object or string representing an existing file)
   Input file must be <subject_id>/surf/lh.pial
rh_pial: (a pathlike object or string representing an existing file)
   Input file must be <subject_id>/surf/rh.pial
transform: (a pathlike object or string representing an existing file)
   Input file must be <subject_id>/mri/transforms/talairach.xfm
thickness: (a pathlike object or string representing an existing file)
   Input file must be <subject_id>/surf/?h.thickness
brainmask: (a pathlike object or string representing an existing file)
   Input file must be <subject_id>/mri/brainmask.mgz
aseg: (a pathlike object or string representing an existing file)
   Input file must be <subject_id>/mri/aseg.presurf.mgz
ribbon: (a pathlike object or string representing an existing file)
   Input file must be <subject_id>/mri/ribbon.mgz

[Optional]
cortex_label: (a pathlike object or string representing an existing file)
   Implicit input file {hemi}.cortex.label
surface: (a string)
   Input surface (e.g. 'white')
   argument: '%s', position: -1
mgz: (a boolean)
   Look for mgz files
   argument: '-mgz'
in_cortex: (a pathlike object or string representing an existing file)
   Input cortex label
   argument: '-cortex %s'
in_annotation: (a pathlike object or string representing an existing file)
   Compute properties for each label in the annotation file separately
   argument: '-a %s'
   mutually_exclusive: in_label
in_label: (a pathlike object or string representing an existing file)
   Limit calculations to specified label
   argument: '-l %s'
   mutually_exclusive: in_annotation, out_color

tabular_output: (a boolean)
   Tabular output
   argument: '-b'
out_table: (a pathlike object or string representing a file)
   Table output to tablefile
   argument: '-f %s'
   requires: tabular_output
out_color: (a pathlike object or string representing a file)
   Output annotation files's colortable to text file
   argument: '-c %s'
   mutually_exclusive: in_label

copy_inputs: (a boolean)
   If running as a node, set this to True. This will copy the input files to the node directory.
th3: (a boolean)
  turns on new vertex-wise volume calc for mris_anat_stats
  argument: ``-th3``
  requires: cortex_label
subjects_dir: (a pathlike object or string representing an existing directory)
  subjects directory
args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  Environment variables

Outputs:

out_table: (a pathlike object or string representing a file)
  Table output to tablefile
out_color: (a pathlike object or string representing a file)
  Output annotation files's colortable to text file

62.5.26 RelabelHypointensities

Link to code
Wraps the executable command mri_relabel_hypointensities.
Relabel Hypointensities

Examples

```python
>>> from nipype.interfaces.freesurfer import RelabelHypointensities
>>> relabelhypos = RelabelHypointensities()
>>> relabelhypos.inputs.lh_white = 'lh.pial'
>>> relabelhypos.inputs.rh_white = 'lh.pial'
>>> relabelhypos.inputs.surf_directory = '.'
>>> relabelhypos.inputs.aseg = 'aseg.mgz'
>>> relabelhypos.cmdline
'mri_relabel_hypointensities aseg.mgz . aseg.hypos.mgz'
```

Inputs:

[Mandatory]
lh_white: (a pathlike object or string representing an existing file)
  Implicit input file must be lh.white
rh_white: (a pathlike object or string representing an existing file)
  Implicit input file must be rh.white
aseg: (a pathlike object or string representing an existing file)
  Input aseg file
    argument: ``%s``, position: -3

[Optional]
surf_directory: (a pathlike object or string representing an existing directory, nipype default value: .)
  Directory containing lh.white and rh.white
    argument: ``%s``, position: -2
out_file: (a pathlike object or string representing a file)
  (continues on next page)
Output aseg file
argument: ``%s``, position: -1

subjects_dir: (a pathlike object or string representing an existing directory)
subjects directory

args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing a file)
    Output aseg file
    argument: ``%s``

62.5.27 RemoveIntersection

Link to code
Wraps the executable command `mris_remove_intersection`.
This program removes the intersection of the given MRI

Examples

```python
>>> from nipype.interfaces.freesurfer import RemoveIntersection
>>> ri = RemoveIntersection()
>>> ri.inputs.in_file = 'lh.pial'
>>> ri.cmdline
'mris_remove_intersection lh.pial lh.pial'
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
    Input file for RemoveIntersection
    argument: ``%s``, position: -2

[Optional]
out_file: (a pathlike object or string representing a file)
    Output file for RemoveIntersection
    argument: ``%s``, position: -1

subjects_dir: (a pathlike object or string representing an existing directory)
subjects directory

args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:
**out_file**: (a pathlike object or string representing a file)

Output file for RemoveNeck

### 62.5.28 RemoveNeck

**Link to code**

Wraps the executable command `mri_remove_neck`. Crops the neck out of the MRI image

**Examples**

```python
>>> from nipype.interfaces.freesurfer import TalairachQC
>>> remove_neck = RemoveNeck()
>>> remove_neck.inputs.in_file = 'norm.mgz'
>>> remove_neck.inputs.transform = 'trans.mat'
>>> remove_neck.inputs.template = 'trans.mat'
>>> remove_neck.cmdline
'mri_remove_neck norm.mgz trans.mat trans.mat norm_noneck.mgz'
```

**Inputs:**

[Mandatory]

in_file: (a pathlike object or string representing an existing file)

  Input file for RemoveNeck
  argument: '%s', position: -4

transform: (a pathlike object or string representing an existing file)

  Input transform file for RemoveNeck
  argument: '%s', position: -3

template: (a pathlike object or string representing an existing file)

  Input template file for RemoveNeck
  argument: '%s', position: -2

[Optional]

out_file: (a pathlike object or string representing a file)

  Output file for RemoveNeck
  argument: '%s', position: -1

radius: (an integer (int or long))

  Radius
  argument: '-radius %d'

subjects_dir: (a pathlike object or string representing an existing directory)

  subjects directory

args: (a unicode string)

  Additional parameters to the command
  argument: '%s'

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) 

  Environment variables

**Outputs:**

out_file: (a pathlike object or string representing a file)

  Output file with neck removed
62.5.29 SampleToSurface

Link to code
Wraps the executable command mri_vol2surf.
Sample a volume to the cortical surface using Freesurfer’s mri_vol2surf.
You must supply a sampling method, range, and units. You can project either a given distance (in mm) or a
given fraction of the cortical thickness at that vertex along the surface normal from the target surface, and then
set the value of that vertex to be either the value at that point or the average or maximum value found along the
projection vector.
By default, the surface will be saved as a vector with a length equal to the number of vertices on the target
surface. This is not a problem for Freesurfer programs, but if you intend to use the file with interfaces to another
package, you must set the reshape input to True, which will factor the surface vector into a matrix with
dimensions compatible with proper Nifti files.

Examples

```python
>>> import nipype.interfaces.freesurfer as fs
>>> sampler = fs.SampleToSurface(hemi="lh")
>>> sampler.inputs.source_file = "cope1.nii.gz"
>>> sampler.inputs.reg_file = "register.dat"
>>> sampler.inputs.sampling_method = "average"
>>> sampler.inputs.sampling_range = 1
>>> sampler.inputs.sampling_units = "frac"
>>> sampler.cmdline  # doctest: +ELLIPSIS
'mri_vol2surf --hemi lh --o ...lh.cope1.mgz --reg register.dat --projfrac-avg 1.
 →000 --mov cope1.nii.gz'
>>> res = sampler.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
source_file: (a pathlike object or string representing an existing file)
  volume to sample values from
  argument: ``--mov %s``
hemi: ("lh" or "rh")
  target hemisphere
  argument: ``--hemi %s``
reg_file: (a pathlike object or string representing an existing file)
  source-to-reference registration file
  argument: ``--reg %s``
  mutually_exclusive: reg_file, reg_header, mni152reg
reg_header: (a boolean)
  register based on header geometry
  argument: ``--regheader %s``
  mutually_exclusive: reg_file, reg_header, mni152reg
  requires: subject_id
mni152reg: (a boolean)
  source volume is in MNI152 space
  argument: ``--mni152reg``
  mutually_exclusive: reg_file, reg_header, mni152reg
sampling_method: ("point" or "max" or "average")
  how to sample -- at a point or at the max or average over a range
  argument: ``%s``
  mutually_exclusive: projection_stem
  requires: sampling_range, sampling_units
projection_stem: (a string)

(continues on next page)
stem for precomputed linear estimates and volume fractions
mutually_exclusive: sampling_method

[Optional]
reference_file: (a pathlike object or string representing an existing
   file)
   reference volume (default is orig.mgz)
   argument: '``--ref %s``
surface: (a string)
   target surface (default is white)
   argument: '``--surf %s``
apply_rot: (a tuple of the form: (a float, a float, a float))
   rotation angles (in degrees) to apply to reg matrix
   argument: '``--rot %.3f %.3f %.3f``'
apply_trans: (a tuple of the form: (a float, a float, a float))
   translation (in mm) to apply to reg matrix
   argument: '``--trans %.3f %.3f %.3f``'
override_reg_subj: (a boolean)
   override the subject in the reg file header
   argument: '``--srcsubject %s``'
   requires: subject_id
sampling_range: (a float or a tuple of the form: (a float, a float, a float))
   sampling range - a point or a tuple of (min, max, step)
sampling_units: ('mm' or 'frac')
   sampling range type -- either 'mm' or 'frac'
smooth_vol: (a float)
   smooth input volume (mm fwhm)
   argument: '``--fwhm %.3f``'
smooth_surf: (a float)
   smooth output surface (mm fwhm)
   argument: '``--surf-fwhm %.3f``'
interp_method: ('nearest' or 'trilinear')
   interpolation method
   argument: '``--interp %s``'
cortex_mask: (a boolean)
   mask the target surface with hemi.cortex.label
   argument: '``--cortex``'
   mutually_exclusive: mask_label
mask_label: (a pathlike object or string representing an existing
   file)
   label file to mask output with
   argument: '``--mask %s``'
   mutually_exclusive: cortex_mask
float2int_method: ('round' or 'tkregister')
   method to convert reg matrix values (default is round)
   argument: '``--float2int %s``'
fix_tk_reg: (a boolean)
   make reg matrix round-compatible
   argument: '``--fixtkreg``'
subject_id: (a string)
   subject id
target_subject: (a string)
   sample to surface of different subject than source
   argument: '``--trgsubject %s``'
surf_reg: (a boolean or a unicode string)
   use surface registration to target subject
(continues on next page)
<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--surfreg %s</code></td>
<td>Requires: target_subject</td>
</tr>
<tr>
<td>ico_order: (an integer (int or long))</td>
<td>Icosahedron order when target_subject is 'ico'</td>
</tr>
<tr>
<td><code>--icoorder %d</code></td>
<td>Requires: target_subject</td>
</tr>
<tr>
<td>reshape: (a boolean)</td>
<td>Reshape surface vector to fit in non-mgh format</td>
</tr>
<tr>
<td><code>--reshape</code></td>
<td>Mutually exclusive: no_reshape</td>
</tr>
<tr>
<td>no_reshape: (a boolean)</td>
<td>Do not reshape surface vector (default)</td>
</tr>
<tr>
<td><code>--noreshape</code></td>
<td>Mutually exclusive: reshape</td>
</tr>
<tr>
<td>reshape_slices: (an integer (int or long))</td>
<td>Number of 'slices' for reshaping</td>
</tr>
<tr>
<td><code>--rf %d</code></td>
<td></td>
</tr>
<tr>
<td>scale_input: (a float)</td>
<td>Multiple all intensities by scale factor</td>
</tr>
<tr>
<td><code>--scale %.3f</code></td>
<td></td>
</tr>
<tr>
<td>frame: (an integer (int or long))</td>
<td>Save only one frame (0-based)</td>
</tr>
<tr>
<td><code>--frame %d</code></td>
<td></td>
</tr>
<tr>
<td>out_file: (a pathlike object or string representing a file)</td>
<td>Surface file to write</td>
</tr>
<tr>
<td><code>--o %s</code></td>
<td></td>
</tr>
<tr>
<td>out_type: ('cor' or 'mgh' or 'mgz' or 'minc' or 'analyze' or 'analyze4d' or 'spm' or 'afni' or 'brik' or 'bshort' or 'bfloat' or 'sdt' or 'outline' or 'otl' or 'gdf' or 'nifti' or 'niigz' or 'gii')</td>
<td>Output file type</td>
</tr>
<tr>
<td><code>--out_type %s</code></td>
<td></td>
</tr>
<tr>
<td>hits_file: (a boolean or a pathlike object or string representing an existing file)</td>
<td>Save image with number of hits at each voxel</td>
</tr>
<tr>
<td><code>--srchit %s</code></td>
<td></td>
</tr>
<tr>
<td>hits_type: ('cor' or 'mgh' or 'mgz' or 'minc' or 'analyze' or 'analyze4d' or 'spm' or 'afni' or 'brik' or 'bshort' or 'bfloat' or 'sdt' or 'outline' or 'otl' or 'gdf' or 'nifti' or 'niigz')</td>
<td>Hits file type</td>
</tr>
<tr>
<td><code>--srchit_type</code></td>
<td></td>
</tr>
<tr>
<td>vox_file: (a boolean or a pathlike object or string representing a file)</td>
<td>Text file with the number of voxels intersecting the surface</td>
</tr>
<tr>
<td><code>--nvox %s</code></td>
<td></td>
</tr>
<tr>
<td>subjects_dir: (a pathlike object or string representing an existing directory)</td>
<td>Subjects directory</td>
</tr>
<tr>
<td>args: (a unicode string)</td>
<td>Additional parameters to the command</td>
</tr>
<tr>
<td>environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})</td>
<td>Environment variables</td>
</tr>
</tbody>
</table>

Outputs:
out_file: (a pathlike object or string representing an existing file) surface file
hits_file: (a pathlike object or string representing an existing file) image with number of hits at each voxel
vox_file: (a pathlike object or string representing an existing file) text file with the number of voxels intersecting the surface

62.5.30 SmoothTessellation

Link to code
Wraps the executable command mris_smooth.
This program smooths the tessellation of a surface using 'mris_smooth'
See also:
SurfaceSmooth() Interface For smoothing a scalar field along a surface manifold

Example

```python
>>> import nipype.interfaces.freesurfer as fs
>>> smooth = fs.SmoothTessellation()
>>> smooth.inputs.in_file = 'lh.hippocampus.stl'
>>> smooth.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  Input volume to tesselate voxels from.
  argument: ``-%s``, position: -2

[Optional]
curvature_averaging_iterations: (an integer (int or long))
  Number of curvature averaging iterations (default=10)
  argument: ``-a %d``
smoothing_iterations: (an integer (int or long))
  Number of smoothing iterations (default=10)
  argument: ``-n %d``
snapshot_writing_iterations: (an integer (int or long))
  Write snapshot every "n" iterations
  argument: ``-w %d``
use_gaussian_curvature_smoothing: (a boolean)
  Use Gaussian curvature smoothing
  argument: ``-g``
gaussian_curvature_norm_steps: (an integer (int or long))
  Use Gaussian curvature smoothing
  argument: ``-%d``
gaussian_curvature_smoothing_steps: (an integer (int or long))
  Use Gaussian curvature smoothing
  argument: ``-%d``
disable_estimates: (a boolean)
  Disables the writing of curvature and area estimates
  argument: ``-nw``
normalize_area: (a boolean)
  Normalizes the area after smoothing
  argument: ``-area``
use_momentum: (a boolean)
62.5.31 Sphere

Link to code

Wraps the executable command mris_sphere. This program will add a template into an average surface.

Examples

```python
>>> from nipype.interfaces.freesurfer import Sphere
>>> sphere = Sphere()
>>> sphere.inputs.in_file = 'lh.pial'
>>> sphere.cmdline
'mris_sphere lh.pial lh.sphere'
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  Input file for Sphere
  argument: `%-s`, position: -2

[Optional]
out_file: (a pathlike object or string representing a file)
  Output file for Sphere
  argument: `%-s`, position: -1
seed: (an integer (int or long))

Outputs:
surface: (a pathlike object or string representing an existing file)
  Smoothed surface file
62.5.32 Surface2VolTransform

Link to code
Wraps the executable command mri_surf2vol. Use FreeSurfer mri_surf2vol to apply a transform.

Examples

```python
>>> from nipype.interfaces.freesurfer import Surface2VolTransform
>>> xfm2vol = Surface2VolTransform()
>>> xfm2vol.inputs.source_file = 'lh.cope1.mgz'
>>> xfm2vol.inputs.reg_file = 'register.mat'
>>> xfm2vol.inputs.hemi = 'lh'
>>> xfm2vol.inputs.template_file = 'cope1.nii.gz'
>>> xfm2vol.inputs.subjects_dir = '.

'"mri_surf2vol --hemi lh --volreg register.mat --surfval lh.cope1.mgz --sd . --template cope1.nii.gz --outvol lh.cope1_asVol.nii --vtxvol lh.cope1_asVol_vertext.nii"

>>> res = xfm2vol.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
source_file: (a pathlike object or string representing an existing file)
This is the source of the surface values
argument: '--surfval %s'
mutually_exclusive: mkmask
hemi: (a unicode string)
  hemisphere of data
  argument: ```--hemi %s```

reg_file: (a pathlike object or string representing an existing file)
  tkRAS-to-tkRAS matrix (tkregister2 format)
  argument: ```--volreg %s```
  mutually_exclusive: subject_id

[Optional]
transformed_file: (a pathlike object or string representing a file)
  Output volume
  argument: ```--outvol %s```
template_file: (a pathlike object or string representing an existing file)
  Output template volume
  argument: ```--template %s```

mkmask: (a boolean)
  make a mask instead of loading surface values
  argument: ```--mkmask```
  mutually_exclusive: source_file

vertexvol_file: (a pathlike object or string representing a file)
  Path name of the vertex output volume, which is the same as output volume except that the value of each voxel is the vertex-id that is mapped to that voxel.
  argument: ```--vtxvol %s```
surf_name: (a unicode string)
  surfname (default is white)
  argument: ```--surf %s```

projfrac: (a float)
  thickness fraction
  argument: ```--projfrac %s```

subjects_dir: (a unicode string)
  freesurfer subjects directory defaults to $SUBJECTS_DIR
  argument: ```--sd %s```

subject_id: (a unicode string)
  subject id
  argument: ```--identity %s```
  mutually_exclusive: reg_file

args: (a unicode string)
  Additional parameters to the command
  argument: ```%s```
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  Environment variables

Outputs:

transformed_file: (a pathlike object or string representing an existing file)
  Path to output file if used normally
vertexvol_file: (a pathlike object or string representing a file)
  vertex map volume path id. Optional

62.5.33 SurfaceSmooth

Link to code
Wraps the executable command `mri_surf2surf`. Smooth a surface image with `mri_surf2surf`. The surface is smoothed by an interactive process of averaging the value at each vertex with those of its adjacent neighbors. You may supply either the number of iterations to run or a desired effective FWHM of the smoothing process. If the latter, the underlying program will calculate the correct number of iterations internally.

See also:

SmoothTessellation() Interface For smoothing a tessellated surface (e.g. in gifti or .stl)

**Examples**

```python
>>> import nipype.interfaces.freesurfer as fs

>>> smoother = fs.SurfaceSmooth()

>>> smoother.inputs.in_file = "lh.cope1.mgz"

>>> smoother.inputs.subject_id = "subj_1"

>>> smoother.inputs.hemi = "lh"

>>> smoother.inputs.fwhm = 5

>>> smoother.cmdline  # doctest: +ELLIPSIS
'mri_surf2surf --cortex --fwhm 5.0000 --hemi lh --sval lh.cope1.mgz --tval ...lh. 
→copei_smooth5.mgz --s subj_1'

>>> smoother.run()  # doctest: +SKIP
```

**Inputs:**

[Mandatory]

`in_file`: (a pathlike object or string representing a file)
  - source surface file
    - argument: `''--sval %s''`
  `subject_id`: (a string)
    - subject id of surface file
      - argument: `''--s %s''`
  `hemi`: ('lh' or 'rh')
    - hemisphere to operate on
      - argument: `''--hemi %s''`

[Optional]

`fwhm`: (a float)
  - effective FWHM of the smoothing process
    - argument: `''--fwhm %.4f''`
    - mutually_exclusive: smooth_iters

`smooth_iters`: (an integer (int or long))
  - iterations of the smoothing process
    - argument: `''--smooth %d''`
    - mutually_exclusive: fwhm

`cortex`: (a boolean, nipype default value: True)
  - only smooth within $hemi.cortex.label
    - argument: `''--cortex''`

`reshape`: (a boolean)
  - reshape surface vector to fit in non-mgh format
    - argument: `''--reshape''`

`out_file`: (a pathlike object or string representing a file)
  - surface file to write
    - argument: `''--tval %s''`

`subjects_dir`: (a pathlike object or string representing an existing directory)
  - subjects directory

`args`: (a unicode string)
  - Additional parameters to the command

(continues on next page)
62.5.34 SurfaceSnapshots

Link to code
Wraps the executable command tksurfer.
Use Tksurfer to save pictures of the cortical surface.
By default, this takes snapshots of the lateral, medial, ventral, and dorsal surfaces. See the six_images option
to add the anterior and posterior surfaces.
You may also supply your own tcl script (see the Freesurfer wiki for information on scripting tksurfer). The
screenshot stem is set as the environment variable "_SNAPSHOT_STEM", which you can use in your own
scripts.
Node that this interface will not run if you do not have graphics enabled on your system.

Examples

```python
>>> import nipype.interfaces.freesurfer as fs
>>> shots = fs.SurfaceSnapshots(subject_id="fsaverage", hemi="lh", surface="pial")
>>> shots.inputs.overlay = "zstat1.nii.gz"
>>> shots.inputs.overlay_range = (2.3, 6)
>>> shots.inputs.overlay_reg = "register.dat"
>>> res = shots.run() # doctest: +SKIP
```

Inputs:

<table>
<thead>
<tr>
<th>Mandatory</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>subject_id: (a string)</td>
<td>subject to visualize</td>
</tr>
<tr>
<td></td>
<td>argument: ´%s´, position: 1</td>
</tr>
<tr>
<td>hemi: ('lh' or 'rh')</td>
<td>hemisphere to visualize</td>
</tr>
<tr>
<td></td>
<td>argument: ´%s´, position: 2</td>
</tr>
<tr>
<td>surface: (a string)</td>
<td>surface to visualize</td>
</tr>
<tr>
<td></td>
<td>argument: ´%s´, position: 3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[Optional]</th>
</tr>
</thead>
<tbody>
<tr>
<td>show_curv: (a boolean)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>show_gray_curv: (a boolean)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>overlay: (a pathlike object or string representing an existing file)</td>
</tr>
</tbody>
</table>
load an overlay volume/surface
argument: `'-overlay %s'`
requires: overlay_range

overlay_reg: (a pathlike object or string representing an existing file)
registration matrix file to register overlay to surface
argument: `'-overlay-reg %s'
mutually_exclusive: overlay_reg, identity_reg, mni152_reg

identity_reg: (a boolean)
use the identity matrix to register the overlay to the surface
argument: `'-overlay-reg-identity'
mutually_exclusive: overlay_reg, identity_reg, mni152_reg

mni152_reg: (a boolean)
use to display a volume in MNI152 space on the average subject
argument: `'-mni152reg'
mutually_exclusive: overlay_reg, identity_reg, mni152_reg

overlay_range: (a float or a tuple of the form: (a float, a float) or a tuple of the form: (a float, a float, a float))
overlay range--either min, (min, max) or (min, mid, max)
argument: `'-%s'

overlay_range_offset: (a float)
overlay range will be symmetric around offset value
argument: `'-foffset %.3f'

truncate_overlay: (a boolean)
truncates the overlay display
argument: `'-truncphaseflag l'

reverse_overlay: (a boolean)
reverse the overlay display
argument: `'-revphaseflag l'

invert_overlay: (a boolean)
invert the overlay display
argument: `'-invphaseflag l'

demean_overlay: (a boolean)
remove mean from overlay
argument: `'-zm'

annot_file: (a pathlike object or string representing an existing file)
path to annotation file to display
argument: `'-annotation %s'
mutually_exclusive: annot_name

annot_name: (a string)
name of annotation to display (must be in $subject/label directory
argument: `'-annotation %s'
mutually_exclusive: annot_file

label_file: (a pathlike object or string representing an existing file)
path to label file to display
argument: `'-label %s'
mutually_exclusive: label_name

label_name: (a string)
name of label to display (must be in $subject/label directory
argument: `'-label %s'
mutually_exclusive: label_file

colortable: (a pathlike object or string representing an existing file)
load colortable file
argument: `'-colortable %s'`
label_under: (a boolean)
  draw label/annotation under overlay
  argument: ``--labels-under``

label_outline: (a boolean)
  draw label/annotation as outline
  argument: ``--label-outline``

patch_file: (a pathlike object or string representing an existing file)
  load a patch
  argument: ``--patch %s``

orig_suffix: (a string)
  set the orig surface suffix string
  argument: ``--orig %s``

sphere_suffix: (a string)
  set the sphere.reg suffix string
  argument: ``--sphere %s``

show_color_scale: (a boolean)
  display the color scale bar
  argument: ``--colscalebarflag 1``

show_color_text: (a boolean)
  display text in the color scale bar
  argument: ``--colscaletext 1``
six_images: (a boolean)
  also take anterior and posterior snapshots

screenshot_stem: (a string)
  stem to use for screenshot file names

stem_template_args: (a list of items which are a string)
  input names to use as arguments for a string-formatted stem template
  requires: screenshot_stem

tcl_script: (a pathlike object or string representing an existing file)
  override default screenshot script
  argument: ``%s``

subjects_dir: (a pathlike object or string representing an existing directory)
  subjects directory

args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:
snapshots: (a list of items which are a pathlike object or string representing an existing file)
  tiff images of the surface from different perspectives

62.5.35 SurfaceTransform

Link to code

Wraps the executable command mri_surf2surf.
Transform a surface file from one subject to another via a spherical registration.
Both the source and target subject must reside in your Subjects Directory, and they must have been processed with recon-all, unless you are transforming to one of the icosahedron meshes.
Examples

```python
>>> from nipype.interfaces.freesurfer import SurfaceTransform
>>> sxfm = SurfaceTransform()
>>> sxfm.inputs.source_file = "lh.cope1.nii.gz"
>>> sxfm.inputs.source_subject = "my_subject"
>>> sxfm.inputs.target_subject = "fsaverage"
>>> sxfm.inputs.hemi = "lh"
>>> sxfm.run()  # doctest: +SKIP
```

Inputs:

[Optional]
- `target_ico_order`: (1 or 2 or 3 or 4 or 5 or 6 or 7)
  - order of the icosahedron if target_subject is 'ico'
  - argument: `''--trgicoorder %d''`
- `source_type`: ('cor' or 'mgh' or 'mgz' or 'minc' or 'analyze' or 'analyze4d' or 'spm' or 'afni' or 'brik' or 'bshort' or 'bfloat' or 'sdt' or 'outline' or 'otl' or 'gdf' or 'nifti' or 'nii' or 'niigz')
  - source file format
  - argument: `''--sfmt %s''`
  - requires: `source_file`
- `target_type`: ('cor' or 'mgh' or 'mgz' or 'minc' or 'analyze' or 'analyze4d' or 'spm' or 'afni' or 'brik' or 'bshort' or 'bfloat' or 'sdt' or 'outline' or 'otl' or 'gdf' or 'nifti' or 'nii' or 'niigz' or 'gii')
  - output format
  - argument: `''--tfmt %s''`
- `reshape`: (a boolean)
  - reshape output surface to conform with Nifti
  - argument: `''--reshape''`
- `reshape_factor`: (an integer (int or long))
  - number of slices in reshaped image
  - argument: `''--reshape-factor''`
- `out_file`: (a pathlike object or string representing a file)
  - surface file to write

[Mandatory]
- `source_file`: (a pathlike object or string representing an existing file)
  - surface file with source values
  - argument: `''--sval %s''`
  - mutually exclusive: `source_annot_file`
- `source_annot_file`: (a pathlike object or string representing an existing file)
  - surface annotation file
  - argument: `''--sval-annot %s''`
  - mutually exclusive: `source_file`
- `source_subject`: (a string)
  - subject id for source surface
  - argument: `''--srcsubject %s''`
- `hemi`: ('lh' or 'rh')
  - hemisphere to transform
  - argument: `''--hemi %s''`
- `target_subject`: (a string)
  - subject id of target surface
  - argument: `''--trgsubject %s''`

(continues on next page)
62.5.36 TalairachAVI

Link to code
Wraps the executable command talairach_avi. Front-end for Avi Snyders image registration tool. Computes the talairach transform that maps the input volume to the MNI average_305. This does not add the xfm to the header of the input file. When called by recon-all, the xfm is added to the header after the transform is computed.

Examples

```python
>>> from nipype.interfaces.freesurfer import TalairachAVI
>>> example = TalairachAVI()
>>> example.inputs.in_file = 'norm.mgz'
>>> example.inputs.out_file = 'trans.mat'
>>> example.cmdline
'talairach_avi --i norm.mgz --xfm trans.mat'

>>> example.run() # doctest: +SKIP
```

Inputs:

- **in_file**: (a pathlike object or string representing an existing file)
  - Input volume
  - Argument: `--i %s`

- **out_file**: (a pathlike object or string representing a file)
  - Output xfm file
  - Argument: `--xfm %s`

- **atlas**: (a string)
  - Alternate target atlas (in freesurfer/average dir)
  - Argument: `--atlas %s`

- **subjects_dir**: (a pathlike object or string representing an existing directory)
  - Subjects directory

- **args**: (a unicode string)
  - Additional parameters to the command
62.5.37 TalairachQC

Link to code
Wraps the executable command `tal_QC_AZS`.

Examples

```python
>>> from nipype.interfaces.freesurfer import TalairachQC
>>> qc = TalairachQC()
>>> qc.inputs.log_file = 'dirs.txt'
>>> qc.cmdline
'tal_QC_AZS dirs.txt'
```

62.5.38 Tkregister2

Link to code
Wraps the executable command `tkregister2`.
Examples

Get transform matrix between orig (tkRAS) and native (scannerRAS) coordinates in Freesurfer. Implements the first step of mapping surfaces to native space in this guide.

```python
>>> from nipype.interfaces.freesurfer import Tkregister2
>>> tk2 = Tkregister2(reg_file='T1_to_native.dat')
>>> tk2.inputs.moving_image = 'T1.mgz'
>>> tk2.inputs.target_image = 'structural.nii'
>>> tk2.inputs.reg_header = True
>>> tk2.cmdline
'tkregister2 --mov T1.mgz --noedit --reg T1_to_native.dat --regheader --targ structural.nii'
>>> tk2.run()  # doctest: +SKIP
```

The example below uses tkregister2 without the manual editing stage to convert FSL-style registration matrix (.mat) to FreeSurfer-style registration matrix (.dat)

```python
>>> from nipype.interfaces.freesurfer import Tkregister2
>>> tk2 = Tkregister2()
>>> tk2.inputs.moving_image = 'epi.nii'
>>> tk2.inputs.fsl_in_matrix = 'flirt.mat'
>>> tk2.cmdline
'tkregister2 --fsl flirt.mat --mov epi.nii --noedit --reg register.dat'
>>> tk2.run()  # doctest: +SKIP
```

Inputs:

- **moving_image**: (a pathlike object or string representing an existing file)
  - moving volume
    - argument: ```--mov %s```
  - reg_file: (a pathlike object or string representing a file, nipype default value: register.dat)
    - freesurfer-style registration file
      - argument: ```--reg %s```
- **target_image**: (a pathlike object or string representing an existing file)
  - target volume
    - argument: ```--targ %s```
    - mutually_exclusive: fstarg
- **fstarg**: (a boolean)
  - use subject's T1 as reference
    - argument: ```--fstarg```
    - mutually_exclusive: target_image
- **fsl_in_matrix**: (a pathlike object or string representing an existing file)
  - fsl-style registration input matrix
    - argument: ```--fsl %s```
- **xfm**: (a pathlike object or string representing an existing file)
  - use a matrix in MNI coordinates as initial registration
    - argument: ```--xfm %s```
- **lta_in**: (a pathlike object or string representing an existing file)
  - use a matrix in MNI coordinates as initial registration
    - argument: ```--lta %s```
- **invert_lta_in**: (a boolean)

(continues on next page)
Invert input LTA before applying
requires: lta_in
fsl_out: (a bool or None or a pathlike object or string representing
          a file)
          compute an FSL-compatible registration matrix
          argument: `''--fslregout %s''`
lta_out: (a bool or None or a pathlike object or string representing
          a file)
          output registration file (LTA format)
          argument: `''--ltaout %s''`
invert_lta_out: (a boolean)
          Invert input LTA before applying
          argument: `''--ltaout-inv''`
          requires: lta_in
subject_id: (a string)
          freesurfer subject ID
          argument: `''--s %s''`
noedit: (a boolean, nipype default value: True)
          do not open edit window (exit)
          argument: `''--noedit''`
reg_header: (a boolean)
          compute registration from headers
          argument: `''--regheader''`
fstal: (a boolean)
          set mov to be tal and reg to be tal xfm
          argument: `''--fstal''`
movscale: (a float)
          adjust registration matrix to scale mov
          argument: `''--movscale %f''`
movscale: (a float)
          adjust registration matrix to scale mov
          argument: `''--movscale %f''`
mutually_exclusive: target_image, moving_image, reg_file
subjects_dir: (a pathlike object or string representing an existing
directory)
args: (a unicode string)
          Additional parameters to the command
          argument: `''%s''`
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipype default value: {})
          Environment variables

Outputs:

reg_file: (a pathlike object or string representing an existing file)
          freesurfer-style registration file
fsl_file: (a pathlike object or string representing a file)
          FSL-style registration file
lta_file: (a pathlike object or string representing a file)
          LTA-style registration file

62.5.39 VolumeMask

Link to code
Wraps the executable command mris_volmask.
Computes a volume mask, at the same resolution as the <subject>/mri/brain.mgz. The volume mask contains 4 values: LH_WM (default 10), LH_GM (default 100), RH_WM (default 20), RH_GM (default 200). The algorithm uses the 4 surfaces situated in <subject>/surf/ [lh|rh][white|pial] and labels voxels based on the
signed-distance function from the surface.

Examples

```python
>>> from nipype.interfaces.freesurfer import VolumeMask
>>> volmask = VolumeMask()
>>> volmask.inputs.left_whitelabel = 2
>>> volmask.inputs.left_ribbonlabel = 3
>>> volmask.inputs.right_whitelabel = 41
>>> volmask.inputs.right_ribbonlabel = 42
>>> volmask.inputs.lh_pial = 'lh.pial'
>>> volmask.inputs.rh_pial = 'lh.pial'
>>> volmask.inputs.lh_white = 'lh.pial'
>>> volmask.inputs.rh_white = 'lh.pial'
>>> volmask.inputs.subject_id = '10335'
>>> volmask.inputs.save_ribbon = True
>>> volmask.cmdline
'mris_volmask --label_left_ribbon 3 --label_left_white 2 --label_right_ribbon 42 --label_right_white 41 --save_ribbon 10335'
```

Inputs:

<table>
<thead>
<tr>
<th>Mandatory</th>
</tr>
</thead>
</table>
| left_whitelabel: (an integer (int or long))
  | left white matter label
  | argument: `''--label_left_white %d''`
| left_ribbonlabel: (an integer (int or long))
  | left cortical ribbon label
  | argument: `''--label_left_ribbon %d''`
| right_whitelabel: (an integer (int or long))
  | Right white matter label
  | argument: `''--label_right_white %d''`
| right_ribbonlabel: (an integer (int or long))
  | Right cortical ribbon label
  | argument: `''--label_right_ribbon %d''`
| lh_pial: (a pathlike object or string representing an existing file)
  | Implicit input left pial surface
| rh_pial: (a pathlike object or string representing an existing file)
  | Implicit input right pial surface
| lh_white: (a pathlike object or string representing an existing file)
  | Implicit input left white matter surface
| rh_white: (a pathlike object or string representing an existing file)
  | Implicit input right white matter surface
| subject_id: (a string, nipype default value: subject_id)
  | Subject being processed
  | argument: `''%s''`, position: -1

<table>
<thead>
<tr>
<th>Optional</th>
</tr>
</thead>
</table>
| aseg: (a pathlike object or string representing an existing file)
  | Implicit aseg.mgz segmentation. Specify a different aseg by using the 'in_aseg' input.
  | mutually exclusive: in_aseg
| in_aseg: (a pathlike object or string representing an existing file)
  | Input aseg file for VolumeMask
  | argument: `''--aseg_name %s''`
  | mutually exclusive: aseg
| save_ribbon: (a boolean)
  | option to save just the ribbon for the hemispheres in the format
```
(continues on next page)
rh.ribbon.mgz
argument: `--save_ribbon`
copy_inputs: (a boolean)
If running as a node, set this to True. This will copy the implicit
input files to the node directory.
subjects_dir: (a pathlike object or string representing an existing
directory)
subjects directory
args: (a unicode string)
Additional parameters to the command
argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_ribbon: (a pathlike object or string representing a file)
Output cortical ribbon mask
lh_ribbon: (a pathlike object or string representing a file)
Output left cortical ribbon mask
rh_ribbon: (a pathlike object or string representing a file)
Output right cortical ribbon mask

62.5.40 copy2subjdir()
Link to code
Method to copy an input to the subjects directory

62.5.41 createoutputdirs()
Link to code
create all output directories. If not created, some freesurfer interfaces fail
63.1 interfaces.fsl.aroma

63.1.1 ICA_AROMA

Link to code
Wraps the executable command ICA_AROMA.py.
Interface for the ICA_AROMA.py script.
ICA-AROMA (i.e. ‘ICA-based Automatic Removal Of Motion Artifacts’) concerns a data-driven method to
identify and remove motion-related independent components from fMRI data. To that end it exploits a small,
but robust set of theoretically motivated features, preventing the need for classifier re-training and therefore
providing direct and easy applicability.
See link for further documentation: https://github.com/rhr-pruim/ICA-AROMA

Example

```python
>>> from nipype.interfaces.fsl import ICA_AROMA
>>> from nipype.testing import example_data

AROMA_obj = ICA_AROMA()
>>> AROMA_obj.inputs.in_file = 'functional.nii'
>>> AROMA_obj.inputs.mat_file = 'func_to_struct.mat'
>>> AROMA_obj.inputs.fnirt_warp_file = 'warpfield.nii'
>>> AROMA_obj.inputs.motion_parameters = 'fsl_mcflirt_movpar.txt'
>>> AROMA_obj.inputs.mask = 'mask.nii.gz'
>>> AROMA_obj.inputs.denoise_type = 'both'
>>> AROMA_obj.inputs.out_dir = 'ICA_testout'
>>> AROMA_obj.cmdline  # doctest: +ELLIPSIS
'ICA_AROMA.py -den both -warp warpfield.nii -i functional.nii -m mask.nii.gz -
affmat func_to_struct.mat -mc fsl_mcflirt_movpar.txt -o .../ICA_testout'
```

Inputs:

[Mandatory]
feat_dir: (a pathlike object or string representing an existing directory)
If a feat directory exists and temporal filtering has not been run
yet, ICA_AROMA can use the files in this directory.

argument: `'-feat %s'`

mutually_exclusive: in_file, mat_file, fnirt_warp_file,

motion_parameters

in_file: (a pathlike object or string representing an existing file)

volume to be denoised

argument: `'-i %s'`

mutually_exclusive: feat_dir

out_dir: (a pathlike object or string representing a directory,

nipype default value: out)

output directory

argument: `'-o %s'`

motion_parameters: (a pathlike object or string representing an

existing file)

motion parameters file

argument: `'-mc %s'`

mutually_exclusive: feat_dir

denoise_type: ('nonaggr' or 'aggr' or 'both' or 'no', nipype default

value: nonaggr)

Type of denoising strategy:

-no: only classification, no denoising

-nonaggr (default): non-aggressive denoising, i.e. partial

component regression

-aggr: aggressive denoising, i.e. full component regression

-both: both aggressive and non-aggressive denoising (two outputs)

argument: `'-den %s'`

[Optional]

mask: (a pathlike object or string representing an existing file)

path/name volume mask

argument: `'-m %s'`

mutually_exclusive: feat_dir

dim: (an integer (int or long))

Dimensionality reduction when running MELODIC (default is automatic

estimation)

argument: `'-dim %d'`

TR: (a float)

TR in seconds. If this is not specified the TR will be extracted

from the header of the fMRI nifti file.

argument: `'-tr %.3f'`

melodic_dir: (a pathlike object or string representing an existing
directory)

path to MELODIC directory if MELODIC has already been run

argument: `'-meldir %s'`

mat_file: (a pathlike object or string representing an existing file)

path/name of the mat-file describing the affine registration (e.g.

FSL FLIRT) of the functional data to structural space (.mat file)

argument: `'-affmat %s'`

mutually_exclusive: feat_dir

fnirt_warp_file: (a pathlike object or string representing an existing file)

File name of the warp-file describing the non-linear registration

(e.g. FSL FNIRT) of the structural data to MNI152 space (.nii.gz)

argument: `'-warp %s'`

mutually_exclusive: feat_dir

args: (a unicode string)

Additional parameters to the command
argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {}) Environment variables

Outputs:

aggr_denoised_file: (a pathlike object or string representing an
existing file)
  if generated: aggressively denoised volume
nonaggr_denoised_file: (a pathlike object or string representing an
existing file)
  if generated: non aggressively denoised volume
out_dir: (a pathlike object or string representing an existing
directory)
  directory contains (in addition to the denoised files): melodic.ica
  + classified_motion_components + classification_overview +
  feature_scores + melodic_ic_mni)

63.2 interfaces.fsl.dti

63.2.1 BEDPOSTX5

Link to code
Wraps the executable command bedpostx.
BEDPOSTX stands for Bayesian Estimation of Diffusion Parameters Obtained using Sampling Techniques.
The X stands for modelling Crossing Fibres. bedpostx runs Markov Chain Monte Carlo sampling to build up
distributions on diffusion parameters at each voxel. It creates all the files necessary for running probabilistic
tractography. For an overview of the modelling carried out within bedpostx see this technical report.

Note: Consider using nipype.workflows.fsl.dmri.create_bedpostx_pipeline() instead.

Example

```python
>>> from nipype.interfaces import fsl
>>> bedp = fsl.BEDPOSTX5(bvecs='bvecs', bvals='bvals', dwi='diffusion.nii',
...   mask='mask.nii', n_fibres=1)
>>> bedp.cmdline
'bedpostx bedpostx -b 0 --burnin_noard=0 --forcedir -n 1 -j 5000 -s 1 --' +
'updateproposalevery=40'
```

Inputs:

[Mandatory]
dwi: (a pathlike object or string representing an existing file)
  diffusion weighted image data file
mask: (a pathlike object or string representing an existing file)
  bet binary mask file
bvecs: (a pathlike object or string representing an existing file)
  b vectors file
bvals: (a pathlike object or string representing an existing file)
  b values file
n_fibres: (a long integer >= 1, nipype default value: 2)
Maximum number of fibres to fit in each voxel
argument: ``-n %d``

out_dir: (a pathlike object or string representing a directory, nipype default value: bedpostx)
output directory
argument: ``%s``, position: 1

[Optional]
logdir: (a pathlike object or string representing a directory)
argument: ``--logdir=%s``

model: (1 or 2 or 3)
  use monoexponential (1, default, required for single-shell) or
  multiexponential (2, multi-shell) model
argument: ``-model %d``
fudge: (an integer (int or long))
  ARD fudge factor
argument: ``-w %d``
n_jumps: (an integer (int or long), nipype default value: 5000)
  Num of jumps to be made by MCMC
argument: ``-j %d``
burn_in: (a long integer >= 0, nipype default value: 0)
  Total num of jumps at start of MCMC to be discarded
argument: ``-b %d``
sample_every: (a long integer >= 0, nipype default value: 1)
  Num of jumps for each sample (MCMC)
argument: ``-s %d``
gradnonlin: (a boolean)
  consider gradient nonlinearities, default off
argument: ``-g``
gradv_dev: (a pathlike object or string representing an existing file)
  grad_dev file, if gradnonlin, -g is True
use_gpu: (a boolean)
  Use the GPU version of bedpostx
burn_in_no_ard: (a long integer >= 0, nipype default value: 0)
  num of burnin jumps before the ard is imposed
argument: ``--burnin_noard=%d``
update_proposal_every: (a long integer >= 1, nipype default value: 40)
  Num of jumps for each update to the proposal density std (MCMC)
argument: ``--updateproposalevery=%d``
seed: (an integer (int or long))
  seed for pseudo random number generator
argument: ``--seed=%d``
no_ard: (a boolean)
  Turn ARD off on all fibres
argument: ``--noard``
mutually_exclusive: no_ard, all_ard
all_ard: (a boolean)
  Turn ARD on on all fibres
argument: ``--allard``
mutually_exclusive: no_ard, all_ard
no_spat: (a boolean)
  Initialise with tensor, not spatially
argument: ``--nospat``
mutually_exclusive: no_spat, non_linear, cnlinear
non_linear: (a boolean)
  Initialise with nonlinear fitting

(continues on next page)
argument: `''--nonlinear''`
mutually_exclusive: no_spat, non_linear, cnlinear
cnlinear: (a boolean)
  Initialise with constrained nonlinear fitting
argument: `''--cnlinear''`
mutually_exclusive: no_spat, non_linear, cnlinear
rician: (a boolean)
  use Rician noise modeling
argument: `''--rician''`
f0_noard: (a boolean)
  Noise floor model: add to the model an unattenuated signal
  compartment f0
argument: `''--f0''`
mutually_exclusive: f0_noard, f0_ard
f0_ard: (a boolean)
  Noise floor model: add to the model an unattenuated signal
  compartment f0
argument: `''--f0 --ardf0''`
mutually_exclusive: f0_noard, f0_ard, all_ard
force_dir: (a boolean, nipype default value: True)
  use the actual directory name given (do not add + to make a new
  directory)
argument: `''--forcedir''`
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
  'NIFTI_PAIR_GZ')
  FSL output type
args: (a unicode string)
  Additional parameters to the command
argument: `''%s''`
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
Environment variables

Outputs:

mean_dsamples: (a pathlike object or string representing an existing
  file)
  Mean of distribution on diffusivity d
mean_fsamples: (a list of items which are a pathlike object or string
  representing an existing file)
  Mean of distribution on f anisotropy
mean_S0samples: (a pathlike object or string representing an existing
  file)
  Mean of distribution on T2wbaseline signal intensity S0
mean_phsamples: (a list of items which are a pathlike object or
  string representing an existing file)
  Mean of distribution on phi
mean_thsamples: (a list of items which are a pathlike object or
  string representing an existing file)
  Mean of distribution on theta
merged_thsamples: (a list of items which are a pathlike object or
  string representing an existing file)
  Samples from the distribution on theta
merged_phsamples: (a list of items which are a pathlike object or
  string representing an existing file)
  Samples from the distribution on phi
merged_fsamples: (a list of items which are a pathlike object or string representing an existing file)

Samples from the distribution on anisotropic volume fraction
dyads: (a list of items which are a pathlike object or string representing an existing file)

Mean of PDD distribution in vector form.
dyads Dispersion: (a list of items which are a pathlike object or string representing an existing file)

Dispersion

References:
None

63.2.2 DTIFit

Link to code
Wraps the executable command dtifit.
Use FSL dtifit command for fitting a diffusion tensor model at each voxel

Example

```python
>>> from nipype.interfaces import fsl
>>> dti = fsl.DTIFit()
>>> dti.inputs.dwi = 'diffusion.nii'
>>> dti.inputs.bvecs = 'bvecs'
>>> dti.inputs.bvals = 'bvals'
>>> dti.inputs.base_name = 'TP'
>>> dti.inputs.mask = 'mask.nii'
>>> dti.cmdline
'dtifit -k diffusion.nii -o TP -m mask.nii -r bvecs -b bvals'
```

Inputs:

[Mandatory]
dwi: (a pathlike object or string representing an existing file)

  diffusion weighted image data file
  argument: ``-k %s``, position: 0

mask: (a pathlike object or string representing an existing file)

  bet binary mask file
  argument: ``-m %s``, position: 2

bvecs: (a pathlike object or string representing an existing file)

  b vectors file
  argument: ``-r %s``, position: 3

bvals: (a pathlike object or string representing an existing file)

  b values file
  argument: ``-b %s``, position: 4

[Optional]
base_name: (a unicode string, nipype default value: dtifit_)

  base_name that all output files will start with
  argument: ``-o %s``, position: 1

min_z: (an integer (int or long))

  min z
  argument: ``-z %d``
max_z: (an integer (int or long))  
  argument: ``-Z %d``

min_y: (an integer (int or long))  
  argument: ``-y %d``

max_y: (an integer (int or long))  
  argument: ``-Y %d``

min_x: (an integer (int or long))  
  argument: ``-x %d``

max_x: (an integer (int or long))  
  argument: ``-X %d``

save_tensor: (a boolean)  
  save the elements of the tensor  
  argument: ``--save_tensor``

sse: (a boolean)  
  output sum of squared errors  
  argument: ``--sse``

cni: (a pathlike object or string representing an existing file)  
  input confound regressors  
  argument: ``--cni=%s``

little_bit: (a boolean)  
  only process small area of brain  
  argument: ``--littlebit``

gradnonlin: (a pathlike object or string representing an existing file)  
  gradient non linearities  
  argument: ``--gradnonlin=%s``

output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')  
  FSL output type

args: (a unicode string)  
  Additional parameters to the command  
  argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  Environment variables

Outputs:

V1: (a pathlike object or string representing an existing file)  
  path/name of file *with* the 1st eigenvector

V2: (a pathlike object or string representing an existing file)  
  path/name of file *with* the 2nd eigenvector

V3: (a pathlike object or string representing an existing file)  
  path/name of file *with* the 3rd eigenvector

L1: (a pathlike object or string representing an existing file)  
  path/name of file *with* the 1st eigenvalue

L2: (a pathlike object or string representing an existing file)  
  path/name of file *with* the 2nd eigenvalue

L3: (a pathlike object or string representing an existing file)  
  path/name of file *with* the 3rd eigenvalue

MD: (a pathlike object or string representing an existing file)
path/name of file with the mean diffusivity
FA: (a pathlike object or string representing an existing file)
path/name of file with the fractional anisotropy
MO: (a pathlike object or string representing an existing file)
path/name of file with the mode of anisotropy
S0: (a pathlike object or string representing an existing file)
path/name of file with the raw T2 signal with no diffusion weighting
tensor: (a pathlike object or string representing an existing file)
path/name of file with the 4D tensor volume
sse: (a pathlike object or string representing an existing file)
path/name of file with the summed squared error

References:
None

63.2.3 DistanceMap

Link to code
Wraps the executable command distancemap.
Use FSL's distancemap to generate a map of the distance to the nearest nonzero voxel.

Example

```python
>>> import nipype.interfaces.fsl as fsl
>>> mapper = fsl.DistanceMap()
>>> mapper.inputs.in_file = "skeleton_mask.nii.gz"
>>> mapper.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
    image to calculate distance values for
    argument: '--in=%s'

[Optional]
mask_file: (a pathlike object or string representing an existing
    file)
    binary mask to contrain calculations
    argument: '--mask=%s'
invert_input: (a boolean)
    invert input image
    argument: '--invert'
local_max_file: (a boolean or a pathlike object or string
    representing a file)
    write an image of the local maxima
    argument: '--localmax=%s'
distance_map: (a pathlike object or string representing a file)
    distance map to write
    argument: '--out=%s'
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
    FSL output type
args: (a unicode string)
Additional parameters to the command
argument: `"%s``
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})  
Environment variables

Outputs:

distance_map: (a pathlike object or string representing an existing
  file)
  value is distance to nearest nonzero voxels
local_max_file: (a pathlike object or string representing a file)
  image of local maxima

References:
None

63.2.4 FindTheBiggest

Link to code

Wraps the executable command find_the_biggest.
Use FSL find_the_biggest for performing hard segmentation on the outputs of connectivity-based thresholding
in probtrack. For complete details, see the FDT Documentation.

Example

```python
>>> from nipype.interfaces import fsl
>>> ldir = ['seeds_to_M1.nii', 'seeds_to_M2.nii']
>>> fBig = fsl.FindTheBiggest(in_files=ldir, out_file='biggestSegmentation')
>>> fBig.cmdline
'find_the_biggest seeds_to_M1.nii seeds_to_M2.nii biggestSegmentation'
```

Inputs:

[Mandatory]
in_files: (a list of items which are a pathlike object or string
  representing an existing file)
  a list of input volumes or a singleMatrixFile
  argument: `"%s"`, position: 0

[Optional]
out_file: (a pathlike object or string representing a file)
  file with the resulting segmentation
  argument: `"%s"`, position: 2
output_type: (`'NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
  'NIFTI_PAIR_GZ')
  FSL output type
args: (a unicode string)
  Additional parameters to the command
  argument: `"%s``
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})  
  Environment variables
Outputs:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out_file</td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td></td>
<td>output file indexed in order of input files</td>
</tr>
<tr>
<td></td>
<td>argument: <code>\%s</code></td>
</tr>
</tbody>
</table>

References:

None

63.2.5 MakeDyadicVectors

Link to code

Wraps the executable command `make_dyadic_vectors`.
Create vector volume representing mean principal diffusion direction and its uncertainty (dispersion)

Inputs:

[Mandatory]
theta_vol: (a pathlike object or string representing an existing file)
  argument: ``\%s``, position: 0
phi_vol: (a pathlike object or string representing an existing file)
  argument: ``\%s``, position: 1

[Optional]
mask: (a pathlike object or string representing an existing file)
  argument: ``\%s``, position: 2
output: (a pathlike object or string representing a file, nipype default value: dyads)
  argument: ``\%s``, position: 3
perc: (a float)
  the {perc}% angle of the output cone of uncertainty (output will be in degrees)
  argument: ``\%f``, position: 4
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
  FSL output type
args: (a unicode string)
  Additional parameters to the command
  argument: ``\%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  Environment variables

Outputs:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dyads:</td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>dispersion</td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
</tbody>
</table>

References:

None

63.2.6 ProbTrackX

Link to code
Wraps the executable command probtrackx.
Use FSL probtrackx for tractography on bedpostx results

Examples

```python
>>> from nipype.interfaces import fsl

pbx = fsl.ProbTrackX(samples_base_name='merged', mask='mask.nii', seed='MASK_average_thal_right.nii', mode='seedmask', xfm='trans.mat', n_samples=3, n_steps=10, force_dir=True, opd=True, os2t=True, target_masks = ['targets_MASK1.nii', 'targets_MASK2.nii'], fsamples='merged_fsamples.nii', phsamples='merged_phsamples.nii', out_dir='.')

>>> pbx.cmdline
'probtrackx --forcedir -m mask.nii --mode=seedmask --nsamples=3 --nsteps=10 --opd=True --os2t --dir=.'
```

Inputs:

```
[Mandatory]
thsamples: (a list of items which are a pathlike object or string representing an existing file)
phsamples: (a list of items which are a pathlike object or string representing an existing file)
fsamples: (a list of items which are a pathlike object or string representing an existing file)
mask: (a pathlike object or string representing an existing file)
    bet binary mask file in diffusion space
    argument: ``-m %s``
seed: (a pathlike object or string representing an existing file or a list of items which are a pathlike object or string representing an existing file or a list of items which are a list of from 3 to 3 items which are an integer (int or long))
    seed volume(s), or voxel(s) or freesurfer label file
    argument: ``--seed=%s``

[Optional]
mode: ('simple' or 'two_mask_symm' or 'seedmask')
    options: simple (single seed voxel), seedmask (mask of seed voxels),
    twomask_symm (two bet binary masks)
    argument: ``--mode=%s``
mask2: (a pathlike object or string representing an existing file)
    second bet binary mask (in diffusion space) in twomask_symm mode
    argument: ``--mask2=%s``
mesh: (a pathlike object or string representing an existing file)
    Freesurfer-type surface descriptor (in ascii format)
    argument: ``--mesh=%s``
samples_base_name: (a unicode string, nipype default value: merged)
    the rootname/base_name for samples files
    argument: ``--samples=%s``
target_masks: (a list of items which are a pathlike object or string representing a file)
    list of target masks - required for seeds_to_targets classification
    argument: ``--targetmasks=%s``
waypoints: (a pathlike object or string representing an existing file)
    waypoint mask or ascii list of waypoint masks - only keep paths going through ALL the masks
```

(continues on next page)
argument: '--waypoints=%s'

network: (a boolean)
activate network mode - only keep paths going through at least one
seed mask (required if multiple seed masks)
argument: '--network'

seed_ref: (a pathlike object or string representing an existing file)
reference vol to define seed space in simple mode - diffusion space
assumed if absent
argument: '--seedref=%s'

out_dir: (a pathlike object or string representing an existing
directory)
directory to put the final volumes in
argument: '--dir=%s'

force_dir: (a boolean, nipype default value: True)
use the actual directory name given - i.e. do not add + to make a
new directory
argument: '--forcedir'

opd: (a boolean, nipype default value: True)
outputs path distribution
argument: '--opd'

correct_path_distribution: (a boolean)
correct path distribution for the length of the pathways
argument: '--pd'

os2t: (a boolean)
Outputs seeds to targets
argument: '--os2t'

avoid_mp: (a pathlike object or string representing an existing file)
reject pathways passing through locations given by this mask
argument: '--avoid=%s'

stop_mask: (a pathlike object or string representing an existing
file)
stop tracking at locations given by this mask file
argument: '--stop=%s'

xfm: (a pathlike object or string representing an existing file)
transformation matrix taking seed space to DTI space (either FLIRT
matrix or FNIRT warp_field) - default is identity
argument: '--xfm=%s'

inv_xfm: (a pathlike object or string representing a file)
transformation matrix taking DTI space to seed space (compulsory
when using a warp_field for seeds_to_dti)
argument: '--invxfm=%s'

n_samples: (an integer (int or long), nipype default value: 5000)
number of samples - default=5000
argument: '--nsamples=%d'

n_steps: (an integer (int or long))
number of steps per sample - default=2000
argument: '--nsteps=%d'

dist_thresh: (a float)
discards samples shorter than this threshold (in mm - default=0)
argument: '--distthresh=%.3f'

c_thresh: (a float)
curvature threshold - default=0.2
argument: '--cthr=%.3f'

sample_random_points: (a boolean)
sample random points within seed voxels
argument: '--sampvox'

step_length: (a float)
step_length in mm - default=0.5
argument: `--steplength=%.3f`

loop_check: (a boolean)
perform loop_checks on paths - slower, but allows lower curvature
threshold
argument: `--loopcheck`

use_anisotropy: (a boolean)
use anisotropy to constrain tracking
argument: `--usef`

rand_fib: (0 or 1 or 2 or 3)
options: 0 - default, 1 - to randomly sample initial fibres (with f > fibthresh), 2 - to sample in proportion fibres (with f>fibthresh)
to f, 3 - to sample ALL populations at random (even if f<fibthresh)
argument: `--randfib=%d`

fibst: (an integer (int or long))
force a starting fibre for tracking - default=1, i.e. first fibre
orientation. Only works if randfib==0
argument: `--fibst=%d`

mod_euler: (a boolean)
use modified euler streamlining
argument: `--modeuler`

random_seed: (a boolean)
random seed
argument: `--rseed`

s2tastext: (a boolean)
output seed-to-target counts as a text file (useful when seeding
from a mesh)
argument: `--s2tastext`

verbose: (0 or 1 or 2)
Verbose level, [0-2]. Level 2 is required to output particle files.
argument: `--verbose=%d`

output_type: (`NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
'NIFTI_PAIR_GZ')
FSL output type

args: (a unicode string)
Additional parameters to the command
argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {}) Environment variables

Outputs:

log: (a pathlike object or string representing an existing file)
path/name of a text record of the command that was run

fdt_paths: (a list of items which are a pathlike object or string
representing an existing file)
path/name of a 3D image file containing the output connectivity
distribution to the seed mask

way_total: (a pathlike object or string representing an existing
file)
path/name of a text file containing a single number corresponding to
the total number of generated tracts that have not been rejected by
inclusion/exclusion mask criteria

targets: (a list of items which are a pathlike object or string
representing an existing file)
References:
None

### 63.2.7 ProbTrackX2

**Link to code**
Wraps the executable command `probtrackx2`.
Use FSL `probtrackx2` for tractography on `bedpostx` results

**Examples**

```python
>>> from nipype.interfaces import fsl
>>> pbx2 = fsl.ProbTrackX2()
>>> pbx2.inputs.seed = 'seed_source.nii.gz'
>>> pbx2.inputs.thsamples = 'merged_th1samples.nii.gz'
>>> pbx2.inputs.fsamples = 'merged_flsamples.nii.gz'
>>> pbx2.inputs.phsamples = 'merged_ph1samples.nii.gz'
>>> pbx2.inputs.mask = 'nodif_brain_mask.nii.gz'
>>> pbx2.inputs.out_dir = '.
>>> pbx2.inputs.n_samples = 3
>>> pbx2.inputs.n_steps = 10
>>> pbx2.cmdline
'probtrackx2 --forcedir -m nodif_brain_mask.nii.gz --nsamples=3 --nsteps=10 --opd
˓→--dir=. --samples=merged --seed=seed_source.nii.gz'
```

**Inputs:**

[Mandatory]

- `thsamples`: (a list of items which are a pathlike object or string representing an existing file)
- `phsamples`: (a list of items which are a pathlike object or string representing an existing file)
- `fsamples`: (a list of items which are a pathlike object or string representing an existing file)
- `mask`: (a pathlike object or string representing an existing file)
  - bet binary mask file in diffusion space
  - argument: `'-m %s'`
- `seed`: (a pathlike object or string representing an existing file or a list of items which are a pathlike object or string representing an existing file or a list of items which are a list of from 3 to 3 items which are an integer (int or long))
  - seed volume(s), or voxel(s) or freesurfer label file
  - argument: `'---seed=%s'`

[Optional]

- `simple`: (a boolean)
  - rack from a list of voxels (seed must be a ASCII list of coordinates)
argument: ``--simple``

fopd: (a pathlike object or string representing an existing file)

Other mask for binning tract distribution
argument: ``--fopd=%s``

waycond: ('OR' or 'AND')
Waypoint condition. Either "AND" (default) or "OR"
argument: ``--waycond=%s``

wayorder: (a boolean)
Reject streamlines that do not hit waypoints in given order. Only valid if waycond=AND
argument: ``--wayorder``

onewaycondition: (a boolean)
Apply waypoint conditions to each half tract separately
argument: ``--onewaycondition``

omatrix1: (a boolean)
Output matrix1 - SeedToSeed Connectivity
argument: ``--omatrix1``

distthresh1: (a float)
Discards samples (in matrix1) shorter than this threshold (in mm - default=0)
argument: ``--distthresh1=%.3f``

omatrix2: (a boolean)
Output matrix2 - SeedToLowResMask
argument: ``--omatrix2``
requires: target2
target2: (a pathlike object or string representing an existing file)
Low resolution binary brain mask for storing connectivity distribution in matrix2 mode
argument: ``--target2=%s``

omatrix3: (a boolean)
Output matrix3 (NxN connectivity matrix)
argument: ``--omatrix3``
requires: target3, lrtarget3
target3: (a pathlike object or string representing an existing file)
Mask used for NxN connectivity matrix (or Nxn if lrtarget3 is set)
argument: ``--target3=%s``
lrtarget3: (a pathlike object or string representing an existing file)
Column-space mask used for Nxn connectivity matrix
argument: ``--lrtarget3=%s``
distthresh3: (a float)
Discards samples (in matrix3) shorter than this threshold (in mm - default=0)
argument: ``--distthresh3=%.3f``

omatrix4: (a boolean)
Output matrix4 - DtiMaskToSeed (special Oxford Sparse Format)
argument: ``--omatrix4``
colmask4: (a pathlike object or string representing an existing file)
Mask for columns of matrix4 (default=seed mask)
argument: ``--colmask4=%s``
target4: (a pathlike object or string representing an existing file)
Brain mask in DTI space
argument: ``--target4=%s``

meshspace: ('caret' or 'freesurfer' or 'first' or 'vox')
Mesh reference space - either "caret" (default) or "freesurfer" or "first" or "vox"
argument: ``--meshspace=%s``
samples_base_name: (a unicode string, nipype default value: merged)
   the rootname/base_name for samples files
   argument: `--samples=%s`

target_masks: (a list of items which are a pathlike object or string
   representing a file)
   list of target masks - required for seeds_to_targets classification
   argument: `--targetmasks=%s`

waypoints: (a pathlike object or string representing an existing
   file)
   waypoint mask or ascii list of waypoint masks - only keep paths
   going through ALL the masks
   argument: `--waypoints=%s`

network: (a boolean)
   activate network mode - only keep paths going through at least one
   seed mask (required if multiple seed masks)
   argument: `--network`

seed_ref: (a pathlike object or string representing an existing file)
   reference vol to define seed space in simple mode - diffusion space
   assumed if absent
   argument: `--seedref=%s`

out_dir: (a pathlike object or string representing an existing
directory)
   directory to put the final volumes in
   argument: `--dir=%s`

force_dir: (a boolean, nipype default value: True)
   use the actual directory name given - i.e. do not add + to make a
   new directory
   argument: `--forcedir`

opd: (a boolean, nipype default value: True)
   outputs path distributions
   argument: `--opd`

correct_path_distribution: (a boolean)
   correct path distribution for the length of the pathways
   argument: `--pd`

os2t: (a boolean)
   Outputs seeds to targets
   argument: `--os2t`

avoid_mp: (a pathlike object or string representing an existing file)
   reject pathways passing through locations given by this mask
   argument: `--avoid=%s`

stop_mask: (a pathlike object or string representing an existing
   file)
   stop tracking at locations given by this mask file
   argument: `--stop=%s`

xfm: (a pathlike object or string representing an existing file)
   transformation matrix taking seed space to DTI space (either FLIRT
   matrix or FNIRT warp_field) - default is identity
   argument: `--xfm=%s`

inv_xfm: (a pathlike object or string representing a file)
   transformation matrix taking DTI space to seed space (compulsory
   when using a warp_field for seeds_to_dti)
   argument: `--invxfm=%s`

n_samples: (an integer (int or long), nipype default value: 5000)
   number of samples - default=5000
   argument: `--nsamples=%d`

n_steps: (an integer (int or long))
   number of steps per sample - default=2000
argument: `--nsteps=%d`
dist_thresh: (a float)
discards samples shorter than this threshold (in mm - default=0)
argument: `--distthresh=%.3f`
c_thresh: (a float)
curvature threshold - default=0.2
argument: `--cthr=%.3f`
sample_random_points: (a boolean)
sample random points within seed voxels
argument: `--sampvox`
step_length: (a float)
step_length in mm - default=0.5
argument: `--steplength=%.3f`
loop_check: (a boolean)
perform loop_checks on paths - slower, but allows lower curvature
threshold
argument: `--loopcheck`
use_anisotropy: (a boolean)
use anisotropy to constrain tracking
argument: `--usef`
rand_fib: (0 or 1 or 2 or 3)
options: 0 - default, 1 - to randomly sample initial fibres (with f
> fibthresh), 2 - to sample in proportion fibres (with f>fibthresh)
to f, 3 - to sample ALL populations at random (even if f<fibthresh)
argument: `--randfib=%d`
fibst: (an integer (int or long))
force a starting fibre for tracking - default=1, i.e. first fibre
orientation. Only works if randfib==0
argument: `--fibst=%d`
mod_euler: (a boolean)
use modified euler streamlining
argument: `--modeuler`
random_seed: (a boolean)
random seed
argument: `--rseed`
s2tastext: (a boolean)
output seed-to-target counts as a text file (useful when seeding
from a mesh)
argument: `--s2tastext`
verbose: (0 or 1 or 2)
Verbose level, [0-2]. Level 2 is required to output particle files.
argument: `--verbose=%d`
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
 'NIFTI_PAIR_GZ')
FSL output type
args: (a unicode string)
Additional parameters to the command
argument: `%%s`
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

network_matrix: (a pathlike object or string representing an existing
file)
the network matrix generated by --omatrix1 option

matrix1_dot: (a pathlike object or string representing an existing file)
Output matrix1.dot - SeedToSeed Connectivity

lookup_tractspace: (a pathlike object or string representing an existing file)
lookup_tractspace generated by --omatrix2 option

matrix2_dot: (a pathlike object or string representing an existing file)
Output matrix2.dot - SeedToLowResMask

matrix3_dot: (a pathlike object or string representing an existing file)
Output matrix3 - NxN connectivity matrix

log: (a pathlike object or string representing an existing file)
path/name of a text record of the command that was run

fdt_paths: (a list of items which are a pathlike object or string representing an existing file)
path/name of a 3D image file containing the output connectivity distribution to the seed mask

way_total: (a pathlike object or string representing an existing file)
path/name of a text file containing a single number corresponding to the total number of generated tracts that have not been rejected by inclusion/exclusion mask criteria

targets: (a list of items which are a pathlike object or string representing an existing file)
a list with all generated seeds_to_target files

particle_files: (a list of items which are a pathlike object or string representing an existing file)
Files describing all of the tract samples. Generated only if verbose is set to 2

References:
None

63.2.8 ProjThresh

Link to code
Wraps the executable command proj_thresh.
Use FSL proj_thresh for thresholding some outputs of protrack For complete details, see the FDT Documentation <http://www.fmrib.ox.ac.uk/fsl/fdt/fdt_thresh.html>

Example

```python
>>> from nipype.interfaces import fsl
>>> ldir = ['seeds_to_M1.nii', 'seeds_to_M2.nii']
>>> pThresh = fsl.ProjThresh(in_files=ldir, threshold=3)
>>> pThresh.cmdline
'proj_thresh seeds_to_M1.nii seeds_to_M2.nii 3'
```

Inputs:

[Mandatory]
in_files: (a list of items which are a pathlike object or string

(continues on next page)
representing an existing file)
a list of input volumes
 argument: ``%s``, position: 0
threshold: (an integer (int or long))
 threshold indicating minimum number of seed voxels entering this
 mask region
 argument: ``%d``, position: 1

[Optional]
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
 'NIFTI_PAIR_GZ')
 FSL output type
args: (a unicode string)
 Additional parameters to the command
 argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
 of class 'str' and with values which are a bytes or None or a
 value of class 'str', nipype default value: {})
 Environment variables

Outputs:

out_files: (a list of items which are a pathlike object or string
 representing an existing file)
 path/name of output volume after thresholding

References:
None

63.2.9 TractSkeleton

Link to code
Wraps the executable command tbss_skeleton.
Use FSL’s tbss_skeleton to skeletonise an FA image or project arbitrary values onto a skeleton.
There are two ways to use this interface. To create a skeleton from an FA image, just supply the in_file
 and set skeleton_file to True (or specify a skeleton filename). To project values onto a skeleton, you
 must set project_data to True, and then also supply values for threshold, distance_map, and
data_file. The search_mask_file and use_cingulum_mask inputs are also used in data projec-
tion, but use_cingulum_mask is set to True by default. This mask controls where the projection algorithm
searches within a circular space around a tract, rather than in a single perpindicular direction.

Example

```python
>>> import nipype.interfaces.fsl as fsl
>>> skeletonor = fsl.TractSkeleton()
>>> skeletonor.inputs.in_file = "all_FA.nii.gz"
>>> skeletonor.inputs.skeleton_file = True
>>> skeletonor.inputs.project_data = True
>>> skeletonor.inputs.threshold = 10
>>> skeletonor.inputs.distance_map = "distance_map.nii.gz"
>>> skeletonor.inputs.data_file = "data_file.nii.gz"
>>> skeletonor.inputs.search_mask_file = "search_mask.nii.gz"
>>> skeletonor.inputs.use_cingulum_mask = True
>>> skeletonor.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
 input image (typcially mean FA volume)
argument: `'-i %s'`

[Optional]

project_data: (a boolean)
  project data onto skeleton
  argument: `'-p %.3f %s %s %s %s'`
  requires: threshold, distance_map, data_file

threshold: (a float)
  skeleton threshold value

distance_map: (a pathlike object or string representing an existing file)
  distance map image

search_mask_file: (a pathlike object or string representing an existing file)
  mask in which to use alternate search rule

use_cingulum_mask: (a boolean, nipype default value: True)
  perform alternate search using built-in cingulum mask

mutually_exclusive: use_cingulum_mask

data_file: (a pathlike object or string representing an existing file)
  4D data to project onto skeleton (usually FA)

alt_data_file: (a pathlike object or string representing an existing file)
  4D non-FA data to project onto skeleton

alt_skeleton: (a pathlike object or string representing an existing file)
  alternate skeleton to use

projected_data: (a pathlike object or string representing a file)
  input data projected onto skeleton

skeleton_file: (a boolean or a pathlike object or string representing a file)
  write out skeleton image

output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
  FSL output type

args: (a unicode string)
  Additional parameters to the command

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  Environment variables

Outputs:

projected_data: (a pathlike object or string representing a file)
  input data projected onto skeleton

skeleton_file: (a pathlike object or string representing a file)
  tract skeleton image

References:

None
63.2.10 VecReg

Link to code
Wraps the executable command vecreg.
Use FSL vecreg for registering vector data For complete details, see the FDT Documentation <http://www.fmrib.ox.ac.uk/fsl/fdt/fdt_vecreg.html>

Example

```python
>>> from nipype.interfaces import fsl
>>> vreg = fsl.VecReg(in_file='diffusion.nii', affine_mat='trans.→mat', ref_vol='mni.nii', out_file='diffusion_vreg.nii')
>>> vreg.cmdline
'vecreg -t trans.mat -i diffusion.nii -o diffusion_vreg.nii -r mni.nii'
```

Inputs:

<table>
<thead>
<tr>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_file</td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>affine_mat</td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>ref_vol</td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>out_file</td>
<td>(a pathlike object or string representing a file)</td>
</tr>
<tr>
<td>warp_field</td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>rotation_mat</td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>rotation_warp</td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>interpolation</td>
<td>('nearestneighbour' or 'trilinear' or 'sinc' or 'spline')</td>
</tr>
<tr>
<td>mask</td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>ref_mask</td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
</tbody>
</table>

(continues on next page)
argument: `--refmask=%s`
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
FSL output type
args: (a unicode string)
  Additional parameters to the command
  argument: `\%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
  path/name of filename for the registered vector or tensor field

References:

None

63.2.11 XFibres5

Link to code

Wraps the executable command xfibres.
Perform model parameters estimation for local (voxelwise) diffusion parameters

Inputs:

[Mandatory]
dwi: (a pathlike object or string representing an existing file)
  diffusion weighted image data file
  argument: `--data=%s`
mask: (a pathlike object or string representing an existing file)
  brain binary mask file (i.e. from BET)
  argument: `--mask=%s`
bvecs: (a pathlike object or string representing an existing file)
  b vectors file
  argument: `--bvecs=%s`
bvals: (a pathlike object or string representing an existing file)
  b values file
  argument: `--bvals=%s`
n_fibres: (a long integer >= 1, nipype default value: 2)
  Maximum number of fibres to fit in each voxel
  argument: `--nfibres=%d`

[Optional]
gradnonlin: (a pathlike object or string representing an existing file)
  gradient file corresponding to slice
  argument: `--gradnonlin=%s`
logdir: (a pathlike object or string representing a directory, nipype default value: .)
  argument: `--logdir=%s`
model: (1 or 2 or 3)
  use monoexponential (1, default, required for single-shell) or multiexponential (2, multi-shell) model
(continues on next page)
argument: ``--model=%d``
fudge: (an integer (int or long))
    ARD fudge factor
    argument: ``--fudge=%d``
n_jumps: (an integer (int or long), nipype default value: 5000)
    Num of jumps to be made by MCMC
    argument: ``--njumps=%d``
burn_in: (a long integer >= 0, nipype default value: 0)
    Total num of jumps at start of MCMC to be discarded
    argument: ``--burnin=%d``
burn_in_no_ard: (a long integer >= 0, nipype default value: 0)
    Num of burnin jumps before the ard is imposed
    argument: ``--burnin_noard=%d``
sample_every: (a long integer >= 0, nipype default value: 1)
    Num of jumps for each sample (MCMC)
    argument: ``--sampleevery=%d``
update_proposal_every: (a long integer >= 1, nipype default value: 40)
    Num of jumps for each update to the proposal density std (MCMC)
    argument: ``--updateproposalevery=%d``
seed: (an integer (int or long))
    seed for pseudo random number generator
    argument: ``--seed=%d``
no_ard: (a boolean)
    Turn ARD off on all fibres
    argument: ``--noard``
    mutually_exclusive: no_ard, all_ard
all_ard: (a boolean)
    Turn ARD on on all fibres
    argument: ``--allard``
    mutually_exclusive: no_ard, all_ard
no_spat: (a boolean)
    Initialise with tensor, not spatially
    argument: ``--nospat``
    mutually_exclusive: no_spat, non_linear, cnlinear
non_linear: (a boolean)
    Initialise with nonlinear fitting
    argument: ``--nonlinear``
    mutually_exclusive: no_spat, non_linear, cnlinear
cnlinear: (a boolean)
    Initialise with constrained nonlinear fitting
    argument: ``--cnonlinear``
    mutually_exclusive: no_spat, non_linear, cnlinear
rician: (a boolean)
    use Rician noise modeling
    argument: ``--rician``
f0_noard: (a boolean)
    Noise floor model: add to the model an unattenuated signal compartment f0
    argument: ``--f0``
    mutually_exclusive: f0_noard, f0_ard
f0_ard: (a boolean)
    Noise floor model: add to the model an unattenuated signal compartment f0
    argument: ``--f0 --ardf0``
    mutually_exclusive: f0_noard, f0_ard, all_ard
force_dir: (a boolean, nipype default value: True)
use the actual directory name given (do not add + to make a new
directory)
argument: `--forcedir`
output_type: (`'NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
 'NIFTI_PAIR_GZ')
FSL output type
args: (a unicode string)
Additional parameters to the command
argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: { })
Environment variables

Outputs:

- dyads: (a list of items which are a pathlike object or string
  representing an existing file)
  Mean of PDD distribution in vector form.
- fsamples: (a list of items which are a pathlike object or string
  representing an existing file)
  Samples from the distribution on f anisotropy
- mean_dsamples: (a pathlike object or string representing an existing
  file)
  Mean of distribution on diffusivity d
- mean_fsamples: (a list of items which are a pathlike object or string
  representing an existing file)
  Mean of distribution on f anisotropy
- mean_S0samples: (a pathlike object or string representing an existing
  file)
  Mean of distribution on T2wbaseline signal intensity S0
- mean_tausamples: (a pathlike object or string representing an existing
  file)
  Mean of distribution on tau samples (only with rician noise)
- phsamples: (a list of items which are a pathlike object or string
  representing an existing file)
  phi samples, per fiber
- thsamples: (a list of items which are a pathlike object or string
  representing an existing file)
  theta samples, per fiber

References:
None

63.3 interfaces.fsl.epi

63.3.1 ApplyTOPUP

Link to code
Wraps the executable command applytopup.
Interface for FSL topup, a tool for estimating and correcting susceptibility induced distortions. General reference and use example.
Examples

```python
>>> from nipype.interfaces.fsl import ApplyTOPUP
>>> applytopup = ApplyTOPUP()
>>> applytopup.inputs.in_files = "[epi.nii", "epi_rev.nii"]"
>>> applytopup.inputs.encoding_file = "topup_encoding.txt"
>>> applytopup.inputs.in_topup_fieldcoef = "topup_fieldcoef.nii.gz"
>>> applytopup.inputs.in_topup_movpar = "topup_movpar.txt"
>>> applytopup.inputs.output_type = "NIFTI_GZ"
>>> applytopup.cmdline # doctest: +ELLIPSIS
'applytopup --datain=topup_encoding.txt --imain=epi.nii,.epi_rev.nii --inindex=1,2,
--topup=topup --out=epi_corrected.nii.gz'
>>> res = applytopup.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
- in_files: (a list of items which are a pathlike object or string representing an existing file)
  - name of file with images
    - argument: `--imain=%s`
  
- encoding_file: (a pathlike object or string representing an existing file)
  - name of text file with PE directions/times
    - argument: `--datain=%s`

[Optional]
- in_index: (a list of items which are an integer (int or long))
  - comma separated list of indices corresponding to --datain
    - argument: `--inindex=%s`

- in_topup_fieldcoef: (a pathlike object or string representing an existing file)
  - topup file containing the field coefficients
    - argument: `--topup=%s`
    - requires: in_topup_movpar

- in_topup_movpar: (a pathlike object or string representing an existing file)
  - topup movpar.txt file
    - requires: in_topup_fieldcoef

- out_corrected: (a pathlike object or string representing a file)
  - output (warped) image
    - argument: `--out=%s`

- method: ('jac' or 'lsr')
  - use jacobian modulation (jac) or least-squares resampling (lsr)
    - argument: `--method=%s`

- interp: ('trilinear' or 'spline')
  - interpolation method
    - argument: `--interp=%s`

- datatype: ('char' or 'short' or 'int' or 'float' or 'double')
  - force output data type
    - argument: `--d=%s`

- output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
  - FSL output type

- args: (a unicode string)
  - Additional parameters to the command
    - argument: `%s`

- environ: (a dictionary with keys which are a bytes or None or a value

(continues on next page)
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {}

Environment variables

Outputs:

out_corrected: (a pathlike object or string representing an existing
file)
name of 4D image file with unwarped images

References:

None

63.3.2 EPIDeWarp

Link to code
Wraps the executable command epidewarp.fsl.
Wraps the unwarping script epidewarp.fsl.

Warning: deprecated in FSL, please use nipype.workflows.dmri.preprocess.epi.
sdc_fmb() instead.

Examples

```python
>>> from nipype.interfaces.fsl import EPIDeWarp
>>> dewarp = EPIDeWarp()
>>> dewarp.inputs.epi_file = "functional.nii"
>>> dewarp.inputs.mag_file = "magnitude.nii"
>>> dewarp.inputs.dph_file = "phase.nii"
>>> dewarp.inputs.output_type = "NIFTI_GZ"
>>> dewarp.cmdline
# doctest: +ELLIPSIS
'epidewarp.fsl --mag magnitude.nii --dph phase.nii --epi functional.nii --esp 0.
˓58 --exf dw .../exf dw.nii.gz --nocleanup --sigma 2 --tediff 2.46 --tmpdir ...
˓temp --vsm .../vsm.nii.gz'
>>> res = dewarp.run() # doctest: +SKIP

Inputs:

mag_file: (a pathlike object or string representing an existing file)
	Magnitude file
	argument: `--mag %s`, position: 0
dph_file: (a pathlike object or string representing an existing file)
	Phase file assumed to be scaled from 0 to 4095
	argument: `--dph %s`

exf_file: (a pathlike object or string representing an existing file)
	example func volume (or use epi)
	argument: `--exf %s`
epi_file: (a pathlike object or string representing an existing file)
	EPI volume to un warp
	argument: `--epi %s`
tediff: (a float, nipype default value: 2.46)
difference in B0 field map TEs
argument: `--tediff %s`

esp: (a float, nipype default value: 0.58)
EPI echo spacing
argument: `--esp %s`

sigma: (an integer (int or long), nipype default value: 2)
2D spatial gaussing smoothing stdev (default = 2mm)
argument: `--sigma %s`

vsm: (a string)
voxel shift map
argument: `--vsm %s`

exfdw: (a string)
dewarped example func volume
argument: `--exfdw %s`

epidw: (a string)
dewarped epi volume
argument: `--epidw %s`

tmpdir: (a string)
tmpdir
argument: `--tmpdir %s`

nocleanup: (a boolean, nipype default value: True)
no cleanup
argument: `--nocleanup`

cleanup: (a boolean)
cleanup
argument: `--cleanup`

output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
'NIFTI_PAIR_GZ')
FSL output type

args: (a unicode string)
Additional parameters to the command
argument: `%%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

unwarped_file: (a pathlike object or string representing a file)
unwarped epi file

vsm_file: (a pathlike object or string representing a file)
voxel shift map

exfdw: (a pathlike object or string representing a file)
dewarped functional volume example

exf_mask: (a pathlike object or string representing a file)
Mask from example functional volume

References:

None

**63.3.3 Eddy**

Link to code
Wraps the executable command `eddy_openmp`.
Interface for FSL eddy, a tool for estimating and correcting eddy currents induced distortions. User guide and
Examples

```python
>>> from nipype.interfaces.fsl import Eddy
>>> eddy = Eddy()
>>> eddy.inputs.in_file = 'epi.nii'
>>> eddy.inputs.in_mask = 'epi_mask.nii'
>>> eddy.inputs.in_index = 'epi_index.txt'
>>> eddy.inputs.in_acqp = 'epi_acqp.txt'
>>> eddy.inputs.in_bvec = 'bvecs.scheme'
>>> eddy.inputs.in_bval = 'bvals.scheme'
>>> eddy.inputs.use_cuda = True
>>> eddy.cmdline
'dedy_cuda --ff=10.0 --acqp=epi_acqp.txt --bvals=bvals.scheme --bvecs=bvecs.scheme --imain=epi.nii --index=epi_index.txt --mask=epi_mask.nii --niter=5 --nvoxhp=1000 --out=.../eddy_corrected'
>>> eddy.inputs.use_cuda = False
>>> eddy.cmdline
'dedy_openmp --ff=10.0 --acqp=epi_acqp.txt --bvals=bvals.scheme --bvecs=bvecs.scheme --imain=epi.nii --index=epi_index.txt --mask=epi_mask.nii --niter=5 --nvoxhp=1000 --out=.../eddy_corrected'
>>> res = eddy.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)  
File containing all the images to estimate distortions for argument: `--imain=%s`
in_mask: (a pathlike object or string representing an existing file)  
Mask to indicate brain argument: `--mask=%s`
in_index: (a pathlike object or string representing an existing file)  
File containing indices for all volumes in --imain into --acqp and --topup argument: `--index=%s`
in_acqp: (a pathlike object or string representing an existing file)  
File containing acquisition parameters argument: `--acqp=%s`
in_bvec: (a pathlike object or string representing an existing file)  
File containing the b-vectors for all volumes in --imain argument: `--bvecs=%s`
in_bval: (a pathlike object or string representing an existing file)  
File containing the b-values for all volumes in --imain argument: `--bvals=%s`

[Optional]
out_base: (a unicode string, nipype default value: eddy_corrected)  
basename for output (warped) image argument: `--out=%s`
session: (a pathlike object or string representing an existing file)  
File containing session indices for all volumes in --imain argument: `--session=%s`
in_topup_fieldcoef: (a pathlike object or string representing an existing file)  
topup file containing the field coefficients argument: `--topup=%s`

(continues on next page)
requires: in_topup_movpar
in_topup_movpar: (a pathlike object or string representing an
existing file)
topup movpar.txt file
requires: in_topup_fieldcoef

flm: ('linear' or 'quadratic' or 'cubic')
First level EC model
argument: '--flm=%s'
slm: ('none' or 'linear' or 'quadratic')
Second level EC model
argument: '--slm=%s'
fep: (a boolean)
Fill empty planes in x- or y-directions
argument: '--fep'
interp: ('spline' or 'trilinear')
Interpolation model for estimation step
argument: '--interp=%s'
nvoxhp: (an integer (int or long), nipype default value: 1000)
# of voxels used to estimate the hyperparameters
argument: '--nvoxhp=%s'
fudge_factor: (a float, nipype default value: 10.0)
Fudge factor for hyperparameter error variance
argument: '--ff=%s'
dont_sep_offs_move: (a boolean)
Do NOT attempt to separate field offset from subject movement
argument: '--dont_sep_offs_move'
dont_peas: (a boolean)
Do NOT perform a post-eddy alignment of shells
argument: '--dont_peas'
fwhm: (a float)
FWHM for conditioning filter when estimating the parameters
argument: '--fwhm=%s'
niter: (an integer (int or long), nipype default value: 5)
Number of iterations
argument: '--niter=%s'
method: ('jac' or 'lsr')
Final resampling method (jacobian/least squares)
argument: '--resamp=%s'
repol: (a boolean)
Detect and replace outlier slices
argument: '--repol'
num_threads: (an integer (int or long), nipype default value: 1)
Number of openmp threads to use
is_shelled: (a boolean)
Override internal check to ensure that date are acquired on a set of
b-value shells
argument: '--data_is_shelled'
field: (a unicode string)
NonTOPUP fieldmap scaled in Hz - filename has to be provided without
an extension. TOPUP is strongly recommended
argument: '--field=%s'
field_mat: (a pathlike object or string representing an existing
file)
Matrix that specifies the relative locations of the field specified
by --field and first volume in file --imain
argument: '--field_mat=%s'

(continues on next page)
Run eddy using cuda gpu

cnr_maps: (a boolean)
  Output CNR-Maps
  argument: `--cnr_maps`

residuals: (a boolean)
  Output Residuals
  argument: `--residuals`

output_type: ('NIFTI' or 'NIFTIPAIR' or 'NIFTIGZ' or
  'NIFTIPAIR_GZ')
  FSL output type

args: (a unicode string)
  Additional parameters to the command
  argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a value
  of class 'str', nipype default value: {})
  Environment variables

Outputs:

out_corrected: (a pathlike object or string representing an existing
  file)
  4D image file containing all the corrected volumes

out_parameter: (a pathlike object or string representing an existing
  file)
  text file with parameters defining the field and movement for each
  scan

out_rotated_bvecs: (a pathlike object or string representing an
  existing file)
  File containing rotated b-values for all volumes

out_movement_rms: (a pathlike object or string representing an
  existing file)
  Summary of the "total movement" in each volume

out_restricted_movement_rms: (a pathlike object or string
  representing an existing file)
  Summary of the "total movement" in each volume disregarding
  translation in the PE direction

out_shell_alignment_parameters: (a pathlike object or string
  representing an existing file)
  File containing rigid body movement parameters between the different
  shells as estimated by a post-hoc mutual information based
  registration

out_outlier_report: (a pathlike object or string representing an
  existing file)
  Text-file with a plain language report on what outlier slices eddy
  has found

out_cnr_maps: (a pathlike object or string representing an existing
  file)
  path/name of file with the cnr_maps

out_residuals: (a pathlike object or string representing an existing
  file)
  path/name of file with the residuals

References:

None
63.3.4 EddyCorrect

Link to code
Wraps the executable command eddy_correct.

**Warning:** Deprecated in FSL. Please use `nipype.interfaces.fsl.epi.Eddy` instead

**Example**

```python
>>> from nipype.interfaces.fsl import EddyCorrect
>>> eddyc = EddyCorrect(in_file='diffusion.nii',
...                     out_file="diffusion_edc.nii", ref_num=0)
>>> eddyc.cmdline
'eddy_correct diffusion.nii diffusion_edc.nii 0'
```

**Inputs:**

- **[Mandatory]**
  - `in_file`: (a pathlike object or string representing an existing file)
  - 4D input file
  - argument: ``%s``, position: 0
  - `ref_num`: (an integer (int or long), nipype default value: 0)
  - reference number
  - argument: ``%d``, position: 2

- **[Optional]**
  - `out_file`: (a pathlike object or string representing a file)
  - 4D output file
  - argument: ``%s``, position: 1
  - `output_type`: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
  - FSL output type
  - `args`: (a unicode string)
  - Additional parameters to the command
  - argument: ``%s``
  - `environ`: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

**Outputs:**

- `eddy_corrected`: (a pathlike object or string representing an existing file)
  - path/name of 4D eddy corrected output file

**References:**

None

63.3.5 EddyQuad

Link to code
Wraps the executable command eddy_quad.
Interface for FSL eddy_quad, a tool for generating single subject reports and storing the quality assessment indices for each subject. [User guide](#)
Examples

```python
>>> from nipype.interfaces.fsl import EddyQuad
>>> quad = EddyQuad()
>>> quad.inputs.base_name = 'eddy_corrected'
>>> quad.inputs.idx_file = 'epi_index.txt'
>>> quad.inputs.param_file = 'epi_acqp.txt'
>>> quad.inputs.mask_file = 'epi_mask.nii'
>>> quad.inputs.bval_file = 'bvals.scheme'
>>> quad.inputs.bvec_file = 'bvecs.scheme'
>>> quad.inputs.output_dir = 'eddy_corrected.qc'
>>> quad.inputs.field = 'fieldmap_phase_fslprepared.nii'
>>> quad.inputs.verbose = True
>>> quad.cmdline
'eddy_quad eddy_corrected --bvals bvals.scheme --bvecs bvecs.scheme --field fieldmap_phase_fslprepared.nii --eddyIdx epi_index.txt --mask epi_mask.nii --output-dir eddy_corrected.qc --eddyParams epi_acqp.txt --verbose'
>>> res = quad.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]

idx_file: (a pathlike object or string representing an existing file)
  File containing indices for all volumes into acquisition parameters
  argument: ``--eddyIdx %s``

param_file: (a pathlike object or string representing an existing file)
  File containing acquisition parameters
  argument: ``--eddyParams %s``

mask_file: (a pathlike object or string representing an existing file)
  Binary mask file
  argument: ``--mask %s``

bval_file: (a pathlike object or string representing an existing file)
  b-values file
  argument: ``--bvals %s``

[Optional]

base_name: (a unicode string, nipype default value: eddy_corrected)
  Basename (including path) for EDDY output files, i.e., corrected images and QC files
  argument: ``%s``, position: 0

bvec_file: (a pathlike object or string representing an existing file)
  b-vectors file - only used when <base_name>.eddy_residuals file is present
  argument: ``--bvecs %s``

output_dir: (a unicode string)
  Output directory - default = '<base_name>.qc'
  argument: ``--output-dir %s``

field: (a pathlike object or string representing an existing file)
  TOPUP estimated field (in Hz)
  argument: ``--field %s``

slice_spec: (a pathlike object or string representing an existing file)
  Text file specifying slice/group acquisition
  argument: ``--slspec %s``

(continues on next page)
**verbos:** (a boolean)
  Display debug messages
  argument: `--verbose`

**output_type:** (`'NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ'`)
  FSL output type

**args:** (a unicode string)
  Additional parameters to the command
  argument: `%(args)s`

**environ:** (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: `{}`)
  Environment variables

**Outputs:**

<table>
<thead>
<tr>
<th>Pathlike Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>qc_json</td>
<td>Single subject database containing quality metrics and data info.</td>
</tr>
<tr>
<td>qc_pdf</td>
<td>Single subject QC report.</td>
</tr>
<tr>
<td>avg_b_png</td>
<td>Image showing mid-sagittal, -coronal and -axial slices of each averaged b-shell volume.</td>
</tr>
<tr>
<td>avg_b0_pe_png</td>
<td>Image showing mid-sagittal, -coronal and -axial slices of each averaged pe-direction b0 volume. Generated when using the -f option.</td>
</tr>
<tr>
<td>cnr_png</td>
<td>Image showing mid-sagittal, -coronal and -axial slices of each b-shell CNR volume. Generated when CNR maps are available.</td>
</tr>
<tr>
<td>vdm_png</td>
<td>Image showing mid-sagittal, -coronal and -axial slices of the voxel displacement map. Generated when using the -f option.</td>
</tr>
<tr>
<td>residuals</td>
<td>Text file containing the volume-wise mask-averaged squared residuals. Generated when residual maps are available.</td>
</tr>
<tr>
<td>clean_volumes</td>
<td>Text file containing a list of clean volumes, based on the eddy squared residuals. To generate a version of the pre-processed dataset without outlier volumes, use: <code>fslselectvols -i &lt;eddy_corrected_data&gt; -o eddy_corrected_data_clean --vols=vols_no_outliers.txt</code></td>
</tr>
</tbody>
</table>

**References:**

None

**63.3.6 EpiReg**

Link to code

Wraps the executable command `epi_reg`.

Runs FSL epi_reg script for simultaneous coregistration and fieldmap unwarping.
Examples

```python
>>> from nipype.interfaces.fsl import EpiReg
>>> epireg = EpiReg()
>>> epireg.inputs.epi='epi.nii'
>>> epireg.inputs.t1_head='T1.nii'
>>> epireg.inputs.t1_brain='T1_brain.nii'
>>> epireg.inputs.out_base='epi2struct'
>>> epireg.inputs.fmap='fieldmap_phase_fslprepared.nii'
>>> epireg.inputs.fmapmag='fieldmap_mag.nii'
>>> epireg.inputs.fmapmagbrain='fieldmap_mag_brain.nii'
>>> epireg.inputs.echospacing=0.00067
>>> epireg.inputs.pedir='y'
>>> epireg.cmdline  # doctest: +ELLIPSIS
'epi_reg --echospacing=0.000670 --fmap=fieldmap_phase_fslprepared.nii --fmapmag=fieldmap_mag.nii --fmapmagbrain=fieldmap_mag_brain.nii --noclean --pedir=y --epi=epi.nii --t1=T1.nii --t1brain=T1_brain.nii --out=epi2struct'
>>> epireg.run()  # doctest: +SKIP
```

Inputs:

<table>
<thead>
<tr>
<th>[Mandatory]</th>
<th>(a pathlike object or string representing an existing file)</th>
</tr>
</thead>
<tbody>
<tr>
<td>epi:</td>
<td>EPI image</td>
</tr>
<tr>
<td>t1_head:</td>
<td>wholehead T1 image</td>
</tr>
<tr>
<td>t1_brain:</td>
<td>brain extracted T1 image</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[Optional]</th>
<th>(a string, nipype default value: epi2struct)</th>
</tr>
</thead>
<tbody>
<tr>
<td>out_base:</td>
<td>output base name</td>
</tr>
<tr>
<td>fmap:</td>
<td>fieldmap image (in rad/s)</td>
</tr>
<tr>
<td>fmapmag:</td>
<td>fieldmap magnitude image - wholehead</td>
</tr>
<tr>
<td>fmapmagbrain:</td>
<td>fieldmap magnitude image - brain extracted</td>
</tr>
<tr>
<td>wmseg:</td>
<td>white matter segmentation of T1 image, has to be named like the T1brain and end on _wmseg</td>
</tr>
<tr>
<td>echospacing:</td>
<td>Effective EPI echo spacing (sometimes called dwell time)</td>
</tr>
</tbody>
</table>
file)
weighting image (in T1 space)
argument: `--weight=%s`

no_fmapreg: (a boolean)
do not perform registration of fmap to T1 (use if fmap already
registered)
argument: `--nofmapreg`

no_clean: (a boolean, nipype default value: True)
do not clean up intermediate files
argument: `--noclean`

output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
'NIFTI_PAIR_GZ')
FSL output type

args: (a unicode string)
Additional parameters to the command
argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
unwarped and coregistered epi input

out_1vol: (a pathlike object or string representing an existing file)
unwarped and coregistered single volume

fmap2str_mat: (a pathlike object or string representing an existing
file)
rigid fieldmap-to-structural transform

fmap2epi_mat: (a pathlike object or string representing an existing
file)
rigid fieldmap-to-epi transform

fmap_epi: (a pathlike object or string representing an existing file)
fieldmap in epi space

fmap_str: (a pathlike object or string representing an existing file)
fieldmap in structural space

fmapmag_str: (a pathlike object or string representing an existing
file)
fieldmap magnitude image in structural space

epi2str_inv: (a pathlike object or string representing an existing
file)
rigid structural-to-epi transform

epi2str_mat: (a pathlike object or string representing an existing
file)
rigid epi-to-structural transform

shiftmap: (a pathlike object or string representing an existing file)
shiftmap in epi space

fullwarp: (a pathlike object or string representing an existing file)
warpfield to unwarp epi and transform into structural space

wmseg: (a pathlike object or string representing an existing file)
white matter segmentation used in flirt bbr

seg: (a pathlike object or string representing an existing file)
white matter, gray matter, csf segmentation

wmedge: (a pathlike object or string representing an existing file)
white matter edges for visualization
References:
None

63.3.7 PrepareFieldmap

Link to code
Wraps the executable command `fslprepare_fieldmap`.
Interface for the `fsl_prepare_fieldmap` script (FSL 5.0)
Prepares a fieldmap suitable for FEAT from SIEMENS data - saves output in rad/s format (e.g. `fsl_prepare_fieldmap SIEMENS images_3_gre_field_mapping images_4_gre_field_mapping fmap_rads 2.65`).

Examples

```python
>>> from nipype.interfaces.fsl import PrepareFieldmap
>>> prepare = PrepareFieldmap()
>>> prepare.inputs.in_phase = "phase.nii"
>>> prepare.inputs.in_magnitude = "magnitude.nii"
>>> prepare.inputs.output_type = "NIFTI_GZ"
>>> prepare.cmdline # doctest: +ELLIPSIS
"fsl_prepare_fieldmap SIEMENS phase.nii magnitude.nii .../phase_fslprepared.nii.gz 2.460000"
>>> res = prepare.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
in_phase: (a pathlike object or string representing an existing file)
    Phase difference map, in SIEMENS format range from 0-4096 or 0-8192
    argument: `\%s`, position: 2
in_magnitude: (a pathlike object or string representing an existing file)
    Magnitude difference map, brain extracted
    argument: `\%s`, position: 3
delta_TE: (a float, nipype default value: 2.46)
    echo time difference of the fieldmap sequence in ms. (usually 2.46ms in Siemens)
    argument: `\%f`, position: -2

[Optional]
scanner: (a string, nipype default value: SIEMENS)
    must be SIEMENS
    argument: `\%s`, position: 1
nocheck: (a boolean, nipype default value: False)
    do not perform sanity checks for image size/range/dimensions
    argument: `--nocheck`, position: -1
out_fieldmap: (a pathlike object or string representing a file)
    output name for prepared fieldmap
    argument: `\%s`, position: 4
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
    FSL output type
    args: (a unicode string)
        Additional parameters to the command
        argument: `\%s`
environ: (a dictionary with keys which are a bytes or None or a value
```
of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}))

Environment variables

Outputs:

out_fieldmap: (a pathlike object or string representing an existing file)
output name for prepared fieldmap

References:

None

63.3.8 SigLoss

Link to code

Wraps the executable command sigloss. Estimates signal loss from a field map (in rad/s)

Examples

```python
>>> from nipype.interfaces.fsl import SigLoss
>>> sigloss = SigLoss()
>>> sigloss.inputs.in_file = "phase.nii"
>>> sigloss.inputs.echo_time = 0.03
>>> sigloss.inputs.output_type = "NIFTI_GZ"
>>> sigloss.cmdline # doctest: +ELLIPSIS
'sigloss --te=0.030000 -i phase.nii -s .../phase_sigloss.nii.gz'
>>> res = sigloss.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
    b0 fieldmap file
    argument: `\'-i %s`'

[Optional]
out_file: (a pathlike object or string representing a file)
    output signal loss estimate file
    argument: `\'-s %s`'
mask_file: (a pathlike object or string representing an existing file)
    brain mask file
    argument: `\'-m %s`'
echo_time: (a float)
    echo time in seconds
    argument: `\'-te=%f`'
slice_direction: ('x' or 'y' or 'z')
    slicing direction
    argument: `\'-d %s`'
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
    FSL output type
args: (a unicode string)
Additional parameters to the command
argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

**Outputs:**

out_file: (a pathlike object or string representing an existing file)

signal loss estimate file

**References:**

None

### 63.3.9 TOPUP

**Link to code**
Wraps the executable command `topup`.

Interface for FSL topup, a tool for estimating and correcting susceptibility induced distortions. See FSL documentation for reference, usage examples, and exemplary config files.

**Examples**

```python
>>> from nipype.interfaces.fsl import TOPUP
>>> topup = TOPUP()
>>> topup.inputs.in_file = "b0_b0rev.nii"
>>> topup.inputs.encoding_file = "topup_encoding.txt"
>>> topup.inputs.output_type = "NIFTI_GZ"
>>> topup.cmdline # doctest: +ELLIPSIS
'topup --config=b0b0.cnf --datain=topup_encoding.txt --imain=b0_b0rev.nii --
   out=b0_b0rev_base --iout=b0_b0rev_corrected.nii.gz --fout=b0_b0rev_field.nii.gz --
   jacout=jac --logout=b0_b0rev_topup.log --rbmout=xfm --dfout=warpfield'
>>> res = topup.run() # doctest: +SKIP
```

**Inputs:**

[Mandatory]

in_file: (a pathlike object or string representing an existing file)
    name of 4D file with images
    argument: `''--imain=%s''`
encoding_file: (a pathlike object or string representing an existing file)
    name of text file with PE directions/times
    argument: `''--datain=%s''`
mutually_exclusive: encoding_direction
encoding_direction: (a list of items which are 'y' or 'x' or 'z' or
    'x-' or 'y-' or 'z-')
    encoding direction for automatic generation of encoding_file
    argument: `''--datain=%s''`
mutually_exclusive: encoding_file
requires: readout_times
readout_times: (a list of items which are a float)
    readout times (dwell times by # phase-encode steps minus 1)
    mutually_exclusive: encoding_file
requires: encoding_direction

[Optional]
out_base: (a pathlike object or string representing a file)
  base-name of output files (spline coefficients (Hz) and movement
  parameters)
  argument: `''--out=%s''`
out_field: (a pathlike object or string representing a file)
  name of image file with field (Hz)
  argument: `''--fout=%s''`
out_warp_prefix: (a unicode string, nipype default value: warpfield)
  prefix for the warpfield images (in mm)
  argument: `''--dfout=%s''`
out_mat_prefix: (a unicode string, nipype default value: xfm)
  prefix for the realignment matrices
  argument: `''--txmout=%s''`
out_jac_prefix: (a unicode string, nipype default value: jac)
  prefix for the warpfield images
  argument: `''--jacout=%s''`
out_corrected: (a pathlike object or string representing a file)
  name of 4D image file with unwarped images
  argument: `''--iout=%s''`
out_logfile: (a pathlike object or string representing a file)
  name of log-file
  argument: `''--logout=%s''`
warp_res: (a float)
  (approximate) resolution (in mm) of warp basis for the different
  sub-sampling levels
  argument: `''--warpres=%f''`
subsamp: (an integer (int or long))
  sub-sampling scheme
  argument: `''--subsamp=%d''`
fwhm: (a float)
  FWHM (in mm) of gaussian smoothing kernel
  argument: `''--fwhm=%f''`
config: (a string, nipype default value: b02b0.cnf)
  Name of config file specifying command line arguments
  argument: `''--config=%s''`
max_iter: (an integer (int or long))
  max # of non-linear iterations
  argument: `''--miter=%d''`
reg_lambda: (a float)
  Weight of regularisation, default depending on --ssqlambda and
  --regmod switches.
  argument: `''--lambda=%0.f''`
ssqlambda: (1 or 0)
  Weight lambda by the current value of the ssd. If used (=1), the
  effective weight of regularisation term becomes higher for the
  initial iterations, therefore initial steps are a little smoother
  than they would without weighting. This reduces the risk of finding
  a local minimum.
  argument: `''--ssqlambda=%d''`
regmod: ('bending_energy' or 'membrane_energy')
  Regularisation term implementation. Defaults to bending_energy. Note
  that the two functions have vastly different scales. The membrane
  energy is based on the first derivatives and the bending energy on
  the second derivatives. The second derivatives will typically be

(continues on next page)
much smaller than the first derivatives, so input lambda will have
to be larger for bending_energy to yield approximately the same
level of regularisation.
argument: `--regmod=%s`
estmov: (1 or 0)
estimate movements if set
argument: `--estmov=%d`
minmet: (0 or 1)
Minimisation method 0=Levenberg-Marquardt, 1=Scaled Conjugate
Gradient
argument: `--minmet=%d`
splineorder: (an integer (int or long))
order of spline, 2->Quadratic spline, 3->Cubic spline
argument: `--splineorder=%d`
numprec: ('double' or 'float')
Precision for representing Hessian, double or float.
argument: `--numprec=%s`
interp: ('spline' or 'linear')
Image interpolation model, linear or spline.
argument: `--interp=%s`
scale: (0 or 1)
If set (=1), the images are individually scaled to a common mean
argument: `--scale=%d`
regrid: (1 or 0)
If set (=1), the calculations are done in a different grid
argument: `--regrid=%d`
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
'NIFTI_PAIR_GZ')
FSL output type
args: (a unicode string)
Additional parameters to the command
argument: `%%s`
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:
out_fieldcoef: (a pathlike object or string representing an existing
file)
file containing the field coefficients
out_movpar: (a pathlike object or string representing an existing
file)
movpar.txt output file
out_enc_file: (a pathlike object or string representing a file)
encoding directions file output for applytopup
out_field: (a pathlike object or string representing a file)
name of image file with field (Hz)
out_warp: (a list of items which are a pathlike object or string
representing an existing file)
warpfield images
out_jacs: (a list of items which are a pathlike object or string
representing an existing file)
Jacobians images
out_mats: (a list of items which are a pathlike object or string
representing an existing file)
realignment matrices
out_corrected: (a pathlike object or string representing a file)
    name of 4D image file with unwarped images
out_logfile: (a pathlike object or string representing a file)
    name of log-file

References:
None

63.4 interfaces.fsl.fix

63.4.1 Classifier

Link to code
Classify ICA components using a specific training dataset (<thresh> is in the range 0-100, typically 5-20).
Inputs:

[Mandatory]
trained_wts_file: (a pathlike object or string representing an existing file)
    trained-weights file
    argument: ``%s``, position: 2
thresh: (an integer (int or long))
    Threshold for cleanup.
    argument: ``%d``, position: -1

[Optional]
mel_ica: (a pathlike object or string representing an existing directory)
    Melodic output directory or directories
    argument: ``%s``, position: 1
artifacts_list_file: (a pathlike object or string representing a file)
    Text file listing which ICs are artifacts; can be the output from classification or can be created manually
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
    Environment variables

Outputs:

artifacts_list_file: (a pathlike object or string representing a file)
    Text file listing which ICs are artifacts; can be the output from classification or can be created manually

63.4.2 Cleaner

Link to code
Extract features (for later training and/or classifying)
Inputs:
artifacts_list_file: (a pathlike object or string representing an existing file)
Text file listing which ICs are artifacts; can be the output from classification or can be created manually
argument: ``%s``, position: 1

[Optional]
cleanup_motion: (a boolean)
cleanup motion confounds, looks for design.fsf for highpass filter cut-off
argument: ``-m`` position: 2
highpass: (a float, nipype default value: 100)
cleanup motion confounds
argument: ``-m -h %f`` position: 2
aggressive: (a boolean)
Apply aggressive (full variance) cleanup, instead of the default less-aggressive (unique variance) cleanup.
argument: ``-A`` position: 3
confound_file: (a pathlike object or string representing a file)
Include additional confound file.
argument: ``-x %s`` position: 4
confound_file_1: (a pathlike object or string representing a file)
Include additional confound file.
argument: ``-x %s`` position: 5
confound_file_2: (a pathlike object or string representing a file)
Include additional confound file.
argument: ``-x %s`` position: 6
args: (a unicode string)
Additional parameters to the command
argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:
cleaned_functional_file: (a pathlike object or string representing an existing file)
Cleaned session data

63.4.3 FeatureExtractor

Link to code
Extract features (for later training and/or classifying)

Inputs:

mel_ica: (a pathlike object or string representing an existing directory)
Melodic output directory or directories
argument: ``%s`` position: -1
args: (a unicode string)
Additional parameters to the command
argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
(continues on next page)
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})

Environment variables

**Outputs:**

```
mel_ica: (a pathlike object or string representing an existing
directory)
    Melodic output directory or directories
    argument: ``%s``, position: -1
```

### 63.4.4 Training

**Link to code**

Train the classifier based on your own FEAT/MELODIC output directory.

**Inputs:**

```
[Optional]

mel_icas: (a list of items which are a pathlike object or string
    representing an existing directory)
    Melodic output directories
    argument: ``%s``, position: -1

trained_wts_filestem: (a unicode string)
    trained-weights filestem, used for trained_wts_file and output
directories
    argument: ``%s``, position: 1

looc: (a boolean)
    full leave-one-out test with classifier training
    argument: ``-l``, position: 2

args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})
    Environment variables
```

**Outputs:**

```
trained_wts_file: (a pathlike object or string representing an
    existing file)
    Trained-weights file
```

### 63.4.5 TrainingSetCreator

**Link to code**

Goes through set of provided melodic output directories, to find all the ones that have a hand_labels_noise.txt file in them.

This is outsourced as a separate class, so that the pipeline is rerun everytime a handlabeled file has been changed, or a new one created.

**Inputs:**

```
[Optional]

mel_icas_in: (a list of items which are a pathlike object or string
    representing an existing directory)
```

(continues on next page)
Melodic output directories

argument: `'\%s\'`, position: -1

Outputs:

mel_icas_out: (a list of items which are a pathlike object or string representing an existing directory)
Hand labels for noise vs signal
argument: `'\%s\'`, position: -1

63.5 interfaces.fsl.maths

63.5.1 AR1Image

Link to code

Wraps the executable command `fslmaths`.

Use `fslmaths` to generate an AR1 coefficient image across a given dimension. (Should use `-odt float` and probably demean first)

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
image to operate on
argument: `'\%s\'`, position: 2

[Optional]
dimension: (`'T'` or `'X'` or `'Y'` or `'Z'`, nipype default value: `T`)
dimension to find AR(1) coefficient across
argument: `'\-%s\'`, position: 4
out_file: (a pathlike object or string representing a file)
image to write
argument: `'\%s\'`, position: -2
internal_datatype: (`'float'` or `'char'` or `'int'` or `'short'` or `'double'
or `'input'`)
datatype to use for calculations (default is float)
argument: `'\-dt %s\'`, position: 1
output_datatype: (`'float'` or `'char'` or `'int'` or `'short'` or `'double'
or `'input'`)
datatype to use for output (default uses input type)
argument: `'\-odt %s\'`, position: -1
nan2zeros: (a boolean)
change NaNs to zeros before doing anything
argument: `'\-nan\'`, position: 3
output_type: (`'NIFTI'` or `'NIFTI_PAIR'` or `'NIFTI_GZ'` or `'NIFTI_PAIR_GZ'`)
FSL output type
args: (a unicode string)
Additional parameters to the command
argument: `'\%s\'`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: `{}")
Environment variables

Outputs:
References:
None

63.5.2 ApplyMask

Link to code
Wraps the executable command fslmaths.
Use fslmaths to apply a binary mask to another image.

Inputs:

[Mandatory]
mask_file: (a pathlike object or string representing an existing file)
  binary image defining mask space
  argument: `"-mas %s"`, position: 4
in_file: (a pathlike object or string representing an existing file)
  image to operate on
  argument: `"%s"`, position: 2

[Optional]
out_file: (a pathlike object or string representing a file)
  image to write
  argument: `"%s"`, position: -2
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double' or 'input')
  datatype to use for calculations (default is float)
  argument: `"-dt %s"`, position: 1
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double' or 'input')
  datatype to use for output (default uses input type)
  argument: `"-odt %s"`, position: -1
nan2zeros: (a boolean)
  change NaNs to zeros before doing anything
  argument: `"-nan"`, position: 3
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
  FSL output type
args: (a unicode string)
  Additional parameters to the command
  argument: `"%s"`
envviron: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: { })
  Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
  image written after calculations

References:
None
63.5.3 BinaryMaths

Link to code

Wraps the executable command *fslmaths*.
Use *fslmaths* to perform mathematical operations using a second image or a numeric value.

Inputs:

<table>
<thead>
<tr>
<th>[Mandatory]</th>
<th>operation: ('add' or 'sub' or 'mul' or 'div' or 'rem' or 'max' or 'min')</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>operation to perform</td>
</tr>
<tr>
<td></td>
<td>argument: <code>-%s</code>, position: 4</td>
</tr>
</tbody>
</table>

| operand_file: (a pathlike object or string representing an existing file) |
| second image to perform operation with                                    |
| argument: ```%s```, position: 5                                           |
| mutually_exclusive: operand_value                                        |

| operand_value: (a float) |
| value to perform operation with                                          |
| argument: ```%.8f```, position: 5                                         |
| mutually_exclusive: operand_file                                        |

| in_file: (a pathlike object or string representing an existing file) |
| image to operate on                                                    |
| argument: ```%s```, position: 2                                          |

| [Optional] | out_file: (a pathlike object or string representing a file) |
|            | image to write                                               |
|            | argument: ```%s```, position: 2                               |

| internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double' or 'input') |
| datatype to use for calculations (default is float)                           |
| argument: ```-dt %s```, position: 1                                             |

| output_datatype: ('float' or 'char' or 'int' or 'short' or 'double' or 'input') |
| datatype to use for output (default uses input type)                          |
| argument: ```-odt %s```, position: -1                                           |

| nan2zeros: (a boolean) |
| change NaNs to zeros before doing anything                                   |
| argument: ```-nan```, position: 3                                              |

| output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ') |
| FSL output type                                                             |

| args: (a unicode string) |
| Additional parameters to the command                                        |
| argument: ```%s```                                                   |

| environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) |
| Environment variables                                                      |

Outputs:

| out_file: (a pathlike object or string representing an existing file) |
| image written after calculations                                   |

References:

None
63.5.4 ChangeDataType

Link to code
Wraps the executable command fslmaths.
Use fslmaths to change the datatype of an image.
Inputs:

[Mandatory]
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
or 'input')
output data type
argument: `'-odt %s'`, position: -1
in_file: (a pathlike object or string representing an existing file)
image to operate on
argument: `'%s'`, position: 2

[Optional]
out_file: (a pathlike object or string representing a file)
image to write
argument: `'%s'`, position: -2
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
or 'input')
datatype to use for calculations (default is float)
argument: `'-dt %s'`, position: 1
nan2zeros: (a boolean)
change NaNs to zeros before doing anything
argument: `'-nan'`, position: 3
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
'NIFTI_PAIR_GZ')
FSL output type
args: (a unicode string)
Additional parameters to the command
argument: `'%s'`
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
image written after calculations

References:
None

63.5.5 DilateImage

Link to code
Wraps the executable command fslmaths.
Use fslmaths to perform a spatial dilation of an image.
Inputs:

[Mandatory]
operation: ('mean' or 'modal' or 'max')
filtering operation to perform in dilation
argument: `'-dil%s'`, position: 6

(continues on next page)
in_file: (a pathlike object or string representing an existing file)
    image to operate on
    argument: `\"%s\"`, position: 2

[Optional]
kernel_shape: ('3D' or '2D' or 'box' or 'boxv' or 'gauss' or 'sphere'
    or 'file')
    kernel shape to use
    argument: `\"-kernel %s\"`, position: 4
kernel_size: (a float)
    kernel size - voxels for box/boxv, mm for sphere, mm sigma for gauss
    argument: `\"%.4f\"`, position: 5
    mutually_exclusive: kernel_file
kernel_file: (a pathlike object or string representing an existing
    file)
    use external file for kernel
    argument: `\"%s\"`, position: 5
    mutually_exclusive: kernel_size
out_file: (a pathlike object or string representing a file)
    image to write
    argument: `\"%s\"`, position: -2
internal_datatypetype: ('float' or 'char' or 'int' or 'short' or 'double'
    or 'input')
    datatype to use for calculations (default is float)
    argument: `\"-dt %s\"`, position: 1
output_datatypetype: ('float' or 'char' or 'int' or 'short' or 'double'
    or 'input')
    datatype to use for output (default uses input type)
    argument: `\"-odt %s\"`, position: -1
nan2zeros: (a boolean)
    change NaNs to zeros before doing anything
    argument: `\"-nan\"`, position: 3
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
    'NIFTI_PAIR_GZ')
    FSL output type
args: (a unicode string)
    Additional parameters to the command
    argument: `\"%s\``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})}
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
    image written after calculations

References:

None

63.5.6 Erodelmage

Link to code

Wraps the executable command fslmaths.
Use fslmaths to perform a spatial erosion of an image.
Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  image to operate on
  argument: ``%s``, position: 2

[Optional]
minimum_filter: (a boolean, nipype default value: False)
  if true, minimum filter rather than erosion by zeroing-out
  argument: ``%s``, position: 6
kernel_shape: ('3D' or '2D' or 'box' or 'boxv' or 'gauss' or 'sphere'
  or 'file')
  kernel shape to use
  argument: ``-kernel %s``, position: 4
kernel_size: (a float)
  kernel size - voxels for box/boxv, mm for sphere, mm sigma for gauss
  argument: ``%.4f``, position: 5
mutually_exclusive: kernel_file
kernel_file: (a pathlike object or string representing an existing
  file)
  use external file for kernel
  argument: ``%s``, position: 5
mutually_exclusive: kernel_size
out_file: (a pathlike object or string representing a file)
  image to write
  argument: ``%s``, position: -2
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
  or 'input')
  datatype to use for calculations (default is float)
  argument: ``-dt %s``, position: 1
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
  or 'input')
  datatype to use for output (default uses input type)
  argument: ``-odt %s``, position: -1
nan2zeros: (a boolean)
  change NaNs to zeros before doing anything
  argument: ``-nan``, position: 3
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
  'NIFTI_PAIR_GZ')
  FSL output type
args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
  Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
  image written after calculations

References:

None
63.5.7 IsotropicSmooth

Link to code
Wraps the executable command fslmaths.
Use fslmaths to spatially smooth an image with a gaussian kernel.

Inputs:

[Mandatory]

fwhm: (a float)
  fwhm of smoothing kernel [mm]
  argument: `"-s %.5f"`, position: 4
  mutually_exclusive: sigma

sigma: (a float)
  sigma of smoothing kernel [mm]
  argument: `"-s %.5f"`, position: 4
  mutually_exclusive: fwhm

in_file: (a pathlike object or string representing an existing file)
  image to operate on
  argument: `"%s"`, position: 2

[Optional]

out_file: (a pathlike object or string representing a file)
  image to write
  argument: `"%s"`, position: -2

internal_datatype: ("float" or 'char' or 'int' or 'short' or 'double'
  or 'input')
  datatype to use for calculations (default is float)
  argument: `"-dt %s"`, position: 1

output_datatype: ("float" or 'char' or 'int' or 'short' or 'double'
  or 'input')
  datatype to use for output (default uses input type)
  argument: `"-odt %s"`, position: -1

nan2zeros: (a boolean)
  change NaNs to zeros before doing anything
  argument: `"-nan"`, position: 3

output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
  'NIFTI_PAIR_GZ')
  FSL output type

args: (a unicode string)
  Additional parameters to the command
  argument: `"%s"`

environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a value
  of class 'str', nipype default value: {})
  Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
  image written after calculations

References:
None

63.5.8 MathsCommand

Link to code
Wraps the executable command fslmaths.
Inputs:

<table>
<thead>
<tr>
<th>[Mandatory]</th>
<th>in_file: (a pathlike object or string representing an existing file)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>image to operate on</td>
</tr>
<tr>
<td></td>
<td>argument: <code>\%s</code>, position: 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[Optional]</th>
<th>out_file: (a pathlike object or string representing a file)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>image to write</td>
</tr>
<tr>
<td></td>
<td>argument: <code>\%s</code>, position: -2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double' or 'input')</th>
</tr>
</thead>
<tbody>
<tr>
<td>datatype to use for calculations (default is float)</td>
</tr>
<tr>
<td>argument: <code>\-dt\ %s</code>, position: 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>output_datatype: ('float' or 'char' or 'int' or 'short' or 'double' or 'input')</th>
</tr>
</thead>
<tbody>
<tr>
<td>datatype to use for output (default uses input type)</td>
</tr>
<tr>
<td>argument: <code>\-odt\ %s</code>, position: -1</td>
</tr>
</tbody>
</table>

| nan2zeros: (a boolean)                                           |
| change NaNs to zeros before doing anything                      |
| argument: ``\-nan``\, position: 3                              |

| output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ') |
| FSL output type                                                  |

| args: (a unicode string)                                        |
| Additional parameters to the command                            |
| argument: ``\%s``                                              |

| environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) |
| Environment variables                                           |

Outputs:

<table>
<thead>
<tr>
<th>out_file: (a pathlike <strong>object</strong> or string representing an existing file)</th>
</tr>
</thead>
<tbody>
<tr>
<td>image written after calculations</td>
</tr>
</tbody>
</table>

References:

None

63.5.9 MaxImage

Link to code

Wraps the executable command fslmaths. Use fslmaths to generate a max image across a given dimension.

Examples

```python
>>> from nipype.interfaces.fsl.maths import MaxImage
>>> maxer = MaxImage()
>>> maxer.inputs.in_file = "functional.nii"  # doctest: +SKIP
>>> maxer.dimension = "T"
>>> maxer.cmdline  # doctest: +SKIP
'fslmaths functional.nii -Tmax functional_max.nii'
```
[Mandatory]
in_file: (a pathlike object or string representing an existing file)
    image to operate on
    argument: ``%-s``, position: 2

[Optional]
dimension: ('T' or 'X' or 'Y' or 'Z', nipype default value: T)
    dimension to max across
    argument: ``-%smax``, position: 4
out_file: (a pathlike object or string representing a file)
    image to write
    argument: ``%-s``, position: -2
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
    or 'input')
    datatype to use for calculations (default is float)
    argument: ``-dt %s``, position: 1
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
    or 'input')
    datatype to use for output (default uses input type)
    argument: ``-odt %s``, position: -1
nan2zeros: (a boolean)
    change NaNs to zeros before doing anything
    argument: ``-nan``, position: 3
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
    'NIFTI_PAIR_GZ')
    FSL output type
args: (a unicode string)
    Additional parameters to the command
    argument: ``-%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {}) Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
    image written after calculations

References:

None

63.5.10 MaxImage

Link to code
Wraps the executable command fslmaths.
Use fslmaths to generate an image of index of max across a given dimension.
Inputs:

[Optional]
dimension: ('T' or 'X' or 'Y' or 'Z', nipype default value: T) (continues on next page)
63.5.11 MeanImage

Link to code
Wraps the executable command fslmaths.
Use fslmaths to generate a mean image across a given dimension.

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  image to operate on
  argument: `'-%s'`, position: 2

[Optional]
dimension: ('T' or 'X' or 'Y' or 'Z', nipype default value: T)
  dimension to mean across
  argument: `'-%smean'`, position: 4
out_file: (a pathlike object or string representing a file)
  image to write
  argument: `'-%s'`, position: -2
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
  or 'input')
  datatype to use for calculations (default is float)
  argument: `'-%dt %s'`, position: 1
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
  or 'input')
  datatype to use for output (default uses input type)
  argument: `'-%odt %s'`, position: -1
nan2zeros: (a boolean)
  change NaNs to zeros before doing anything
  argument: `'-%nan'`, position: 3
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
  'NIFTI_PAIR_GZ')
  FSL output type
args: (a unicode string)
  Additional parameters to the command
  argument: `'-%s'`
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: { })
  Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
  image written after calculations

References:

None
or 'input')
datatype to use for calculations (default is float)
argument: `--dt %s`, position: 1
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
or 'input')
datatype to use for output (default uses input type)
argument: `--odt %s`, position: -1
nan2zeros: (a boolean)
change NaNs to zeros before doing anything
argument: `--nan`, position: 3
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
'NIFTI_PAIR_GZ')
FSL output type
args: (a unicode string)
Additional parameters to the command
argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})  
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
image written after calculations

References:

None

63.5.12 MedianImage

Link to code
Wraps the executable command fslmaths.
Use fslmaths to generate a median image across a given dimension.

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
image to operate on
argument: `%s`, position: 2

[Optional]
dimension: ('T' or 'X' or 'Y' or 'Z', nipype default value: T)
dimension to median across
argument: `--median`, position: 4
out_file: (a pathlike object or string representing a file)
image to write
argument: `%s`, position: -2
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
or 'input')
datatype to use for calculations (default is float)
argument: `--dt %s`, position: 1
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
or 'input')
datatype to use for output (default uses input type)
argument: `--odt %s`, position: -1
nan2zeros: (a boolean)
    change NaNs to zeros before doing anything
argument: `--nan`, position: 3
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
    FSL output type
args: (a unicode string)
    Additional parameters to the command
argument: `%%s`
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str', and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})
    Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
    image written after calculations

References:
None

63.5.13 MinImage

Link to code
Wraps the executable command fslmaths.
Use fslmaths to generate a minimum image across a given dimension.
Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
    image to operate on
    argument: `%%s`, position: 2

[Optional]
dimension: ('T' or 'X' or 'Y' or 'Z', nipype default value: T)
    dimension to min across
    argument: `--%smin`, position: 4
out_file: (a pathlike object or string representing a file)
    image to write
    argument: `%%s`, position: -2
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
    or 'input')
    datatype to use for calculations (default is float)
    argument: `--dt %%s`, position: 1
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
    or 'input')
    datatype to use for output (default uses input type)
    argument: `--odt %%s`, position: -1
nan2zeros: (a boolean)
    change NaNs to zeros before doing anything
    argument: `--nan`, position: 3
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')

(continues on next page)
FSL output type
args: (a unicode string)
Additional parameters to the command
argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
image written after calculations

References:
None

63.5.14 MultiImageMaths

Link to code
Wraps the executable command fsln maths.
Use fsln maths to perform a sequence of mathematical operations.

Examples

```python
>>> from nipype.interfaces.fsl import MultiImageMaths
>>> maths = MultiImageMaths()
>>> maths.inputs.in_file = "functional.nii"
>>> maths.inputs.op_string = "-add $s -mul -1 -div $s"
>>> maths.inputs.operand_files = ["functional2.nii", "functional3.nii"]
>>> maths.inputs.out_file = "functional4.nii"
>>> maths.cmdline
→functional4.nii'
```

Inputs:

[Mandatory]
op_string: (a string)
  python formatted string of operations to perform
  argument: `%s`%, position: 4
operand_files: (a list of items which are a pathlike object or string
  representing an existing file)
  list of file names to plug into op string
in_file: (a pathlike object or string representing an existing file)
  image to operate on
  argument: `%s`%, position: 2

[Optional]
out_file: (a pathlike object or string representing a file)
  image to write
  argument: `%s`%, position: -2
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
or 'input')
datatype to use for calculations (default is float)
argument: ``-dt %s``, position: 1
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
or 'input')
datatype to use for output (default uses input type)
argument: ``-odt %s``, position: -1
nan2zeros: (a boolean)
change NaNs to zeros before doing anything
argument: ``-nan``, position: 3
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
'NIFTI_PAIR_GZ')
FSL output type
args: (a unicode string)
Additional parameters to the command
argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
image written after calculations

References:

None

63.5.15 PercentileImage

Link to code

Wraps the executable command fslmaths.
Use fslmaths to generate a percentile image across a given dimension.

Examples

```python
>>> from nipype.interfaces.fsl.maths import MaxImage
>>> percen = PercentileImage()
>>> percen.inputs.in_file = "functional.nii" # doctest: +SKIP
>>> percen.dimension = "T"
>>> percen.perc = 90
>>> percen.cmdline # doctest: +SKIP
'fslmaths functional.nii -Tperc 90 functional_perc.nii'
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
image to operate on
argument: ``%s`` , position: 2

[Optional]
dimension: ('T' or 'X' or 'Y' or 'Z', nipype default value: T)
dimension to percentile across
argument: ``-%sperc`` , position: 4
perc: (0 <= a long integer <= 100)
nth percentile (0-100) of FULL RANGE across dimension
argument: ``-%f``, position: 5
out_file: (a pathlike object or string representing a file)
image to write
argument: ``-%s``, position: -2
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
or 'input')
datatype to use for calculations (default is float)
argument: ``-dt %s``, position: 1
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
or 'input')
datatype to use for output (default uses input type)
argument: ``-odt %s``, position: -1
nan2zeros: (a boolean)
change NaNs to zeros before doing anything
argument: ``-nan``, position: 3
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
NIFTI_PAIR_GZ')
FSL output type
args: (a unicode string)
Additional parameters to the command
argument: ``-%s``
environ: (a dictionary with keys which are a bytes or None or a value of
class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
image written after calculations

References:

None

63.5.16 SpatialFilter

Link to code
Wraps the executable command fslmaths.
Use fslmaths to spatially filter an image.
Inputs:

[Mandatory]
operation: ('mean' or 'median' or 'meanu')
operation to filter with
argument: ``-f%s``, position: 6
in_file: (a pathlike object or string representing an existing file)
image to operate on
argument: ``-%s``, position: 2

[Optional]
kernel_shape: ('3D' or '2D' or 'box' or 'boxv' or 'gauss' or 'sphere'
or 'file')
kernel shape to use
argument: ``-kernel %s``, position: 4
kernel_size: (a float)
  kernel size - voxels for box/boxv, mm for sphere, mm sigma for gauss
  argument: ``%.4f``, position: 5
  mutually_exclusive: kernel_file

kernel_file: (a pathlike object or string representing an existing file)
  use external file for kernel
  argument: ``%s``, position: 5
  mutually_exclusive: kernel_size

out_file: (a pathlike object or string representing a file)
  image to write
  argument: ``%s``, position: -2

internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
  or 'input')
  datatype to use for calculations (default is float)
  argument: ``-dt %s``, position: 1

output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
  or 'input')
  datatype to use for output (default uses input type)
  argument: ``-odt %s``, position: -1

nan2zeros: (a boolean)
  change NaNs to zeros before doing anything
  argument: ``-nan`` , position: 3

output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
  FSL output type

args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a value
  of class 'str', nipype default value: {})
  Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
  image written after calculations

References:

None

63.5.17 StdImage

Link to code

Wraps the executable command fslmaths.

Use fslmaths to generate a standard deviation in an image across a given dimension.

Inputs:

[Mandatory]

in_file: (a pathlike object or string representing an existing file)
  image to operate on
  argument: ``%s``, position: 2

[Optional]
63.5.18 TemporalFilter

Link to code
Wraps the executable command fslmaths.
Use fslmaths to apply a low, high, or bandpass temporal filter to a timeseries.

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  image to operate on
  argument: `"-%s"`, position: 2

[Optional]
lowpass_sigma: (a float, nipype default value: -1)
  lowpass filter sigma (in volumes)
  argument: `"-%.6f"`, position: 5

highpass_sigma: (a float, nipype default value: -1)
  highpass filter sigma (in volumes)
  argument: `"-%bptf %.6f"`, position: 4

(continues on next page)
out_file: (a pathlike object or string representing a file)
  image to write
  argument: ```%s```, position: -2
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
  or 'input')
  datatype to use for calculations (default is float)
  argument: ```-dt %s```, position: 1
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
  or 'input')
  datatype to use for output (default uses input type)
  argument: ```-odt %s```, position: -1
nan2zeros: (a boolean)
  change NaNs to zeros before doing anything
  argument: ```-nan```, position: 3
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
  'NIFTI_PAIR_GZ')
  FSL output type
args: (a unicode string)
  Additional parameters to the command
  argument: ```%s```
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
requires: use_robust_range
out_file: (a pathlike object or string representing a file)
  image to write
  argument: ``\$s`` , position: -2
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
or 'input')
  datatype to use for calculations (default is float)
  argument: ``-dt \$s`` , position: 1
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
or 'input')
  datatype to use for output (default uses input type)
  argument: ``-odt \$s`` , position: -1
nan2zeros: (a boolean)
  change NaNs to zeros before doing anything
  argument: ``-nan`` , position: 3
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
  'NIFTI_PAIR_GZ')
  FSL output type
args: (a unicode string)
  Additional parameters to the command
  argument: ``\$s``
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
  Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
  image written after calculations

References:
None

63.5.20 UnaryMaths

Link to code
Wraps the executable command fslmaths.
Use fslmaths to perform a variety of mathematical operations on an image.
Inputs:

[Mandatory]
operation: ('exp' or 'log' or 'sin' or 'cos' or 'tan' or 'asin' or
  'acos' or 'atan' or 'sqr' or 'sqrt' or 'recip' or 'abs' or 'bin'
or 'binv' or 'fillh' or 'fillh26' or 'index' or 'edge' or 'nan' or
  'nann' or 'rand' or 'randn' or 'range')
  operation to perform
  argument: ``-\$s`` , position: 4
in_file: (a pathlike object or string representing an existing file)
  image to operate on
  argument: ``\$s`` , position: 2

[Optional]
out_file: (a pathlike object or string representing a file)
  image to write
argument: `\"%s\"`, position: -2
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
 or 'input')
datatype to use for calculations (default is float)
argument: `\"-dt %s\"`, position: 1
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
 or 'input')
datatype to use for output (default uses input type)
argument: `\"-odt %s\"`, position: -1
nan2zeros: (a boolean)
change NaNs to zeros before doing anything
argument: `\"-nan\"`, position: 3
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
 'NIFTI_PAIR_GZ')
FSL output type
args: (a unicode string)
Additional parameters to the command
argument: `\"%s\"`
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {}) 
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
image written after calculations

References:
None

63.6 interfaces.fsl.model

63.6.1 Cluster

Link to code
Wraps the executable command cluster.
Uses FSL cluster to perform clustering on statistical output

Examples

```python
>>> cl = Cluster()
>>> cl.inputs.threshold = 2.3
>>> cl.inputs.in_file = 'zstat1.nii.gz'
>>> cl.inputs.out_localmax_txt_file = 'stats.txt'
>>> cl.inputs.use_mm = True
>>> cl.cmdline
'cluster --in=zstat1.nii.gz --olmax=stats.txt --thresh=2.3000000000 --mm'
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
input volume
argument: `--in=%s`
threshold: (a float)
threshold for input volume
argument: `--thresh=%.10f`

[Optional]
out_index_file: (a boolean or a pathlike object or string representing a file)
output of cluster index (in size order)
argument: `--oindex=%s`
out_threshold_file: (a boolean or a pathlike object or string representing a file)
thresholded image
argument: `--othresh=%s`
out_localmax_txt_file: (a boolean or a pathlike object or string representing a file)
local maxima text file
argument: `--olmax=%s`
out_localmax_vol_file: (a boolean or a pathlike object or string representing a file)
output of local maxima volume
argument: `--olmaxim=%s`
out_size_file: (a boolean or a pathlike object or string representing a file)
filename for output of size image
argument: `--osize=%s`
out_max_file: (a boolean or a pathlike object or string representing a file)
filename for output of max image
argument: `--omax=%s`
out_mean_file: (a boolean or a pathlike object or string representing a file)
filename for output of mean image
argument: `--omean=%s`
out_pval_file: (a boolean or a pathlike object or string representing a file)
filename for image output of log pvals
argument: `--opvals=%s`

pthreshold: (a float)
p-threshold for clusters
argument: `--pthresh=%.10f`
requires: dlh, volume

peak_distance: (a float)
minimum distance between local maxima/minima, in mm (default 0)
argument: `--peakdist=%.10f`

cope_file: (a pathlike object or string representing a file)
cope volume
argument: `--cope=%s`

volume: (an integer (int or long))
number of voxels in the mask
argument: `--volume=%d`

dlh: (a float)
smoothness estimate = sqrt(det(Lambda))
argument: `--dlh=%.10f`

fractional: (a boolean, nipype default value: False)
interprets the threshold as a fraction of the robust range
argument: `--fractional`
connectivity: (an integer (int or long))
    the connectivity of voxels (default 26)
    argument: `--connectivity=%d`
use_mm: (a boolean, nipype default value: False)
    use mm, not voxel, coordinates
    argument: `--mm`
find_min: (a boolean, nipype default value: False)
    find minima instead of maxima
    argument: `--min`
no_table: (a boolean, nipype default value: False)
    suppresses printing of the table info
    argument: `--no_table`
minclustersize: (a boolean, nipype default value: False)
    prints out minimum significant cluster size
    argument: `--minclustersize`
xfm_file: (a pathlike object or string representing a file)
    filename for Linear: input->standard-space transform. Non-linear:
    input->highres transform
    argument: `--xfm=%s`
std_space_file: (a pathlike object or string representing a file)
    filename for standard-space volume
    argument: `--stdvol=%s`
um_maxima: (an integer (int or long))
    no of local maxima to report
    argument: `--num=%d`
warpfield_file: (a pathlike object or string representing a file)
    file containing warpfield
    argument: `--warpvol=%s`
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
    'NIFTI_PAIR_GZ')
    FSL output type
args: (a unicode string)
    Additional parameters to the command
    argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})
References:
None

63.6.2 ContrastMgr

Link to code
Wraps the executable command contrast_mgr.
Use FSL contrast_mgr command to evaluate contrasts
In interface mode this file assumes that all the required inputs are in the same location. This has deprecated for
FSL versions 5.0.7+ as the necessary corrections file is no longer generated by FILMGLS.
Inputs:

[Mandatory]
tcon_file: (a pathlike object or string representing an existing
    file)
    contrast file containing T-contrasts
    argument: ``-%s``, position: -1
param_estimates: (a list of items which are a pathlike object or
    string representing an existing file)
    Parameter estimates for each column of the design matrix
corrections: (a pathlike object or string representing an existing
    file)
    statistical corrections used within FILM modelling
dof_file: (a pathlike object or string representing an existing file)
    degrees of freedom
sigmasquareds: (a pathlike object or string representing an existing
    file)
    summary of residuals, See Woolrich, et. al., 2001

[Optional]
fcon_file: (a pathlike object or string representing an existing
    file)
    contrast file containing F-contrasts
    argument: ``-f %s``
contrast_num: (a long integer >= 1)
    contrast number to start labeling copes from
    argument: ``-cope``
suffix: (a unicode string)
    suffix to put on the end of the cope filename before the contrast
    number, default is nothing
    argument: ``-suffix %s``
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
    'NIFTI_PAIR_GZ')
    FSL output type
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: { })
    Environment variables

Outputs:
copes: (a list of items which are a pathlike object or string
    representing an existing file)
    Contrast estimates for each contrast
varcopes: (a list of items which are a pathlike object or string representing an existing file)
  Variance estimates for each contrast
zstats: (a list of items which are a pathlike object or string representing an existing file)
  z-stat file for each contrast
tstats: (a list of items which are a pathlike object or string representing an existing file)
  t-stat file for each contrast
fstats: (a list of items which are a pathlike object or string representing an existing file)
  f-stat file for each contrast
zfstats: (a list of items which are a pathlike object or string representing an existing file)
  z-stat file for each F contrast
neffs: (a list of items which are a pathlike object or string representing an existing file)
  neff file ?? for each contrast

References:
None

63.6.3 DualRegression

Link to code
Wraps the executable command dual_regression.
Wrapper Script for Dual Regression Workflow

Examples

```python
>>> dual_regression = DualRegression()
>>> dual_regression.inputs.in_files = ["functional.nii", "functional2.nii","functional3.nii"]
>>> dual_regression.inputs.group_IC_maps_4D = "allFA.nii"
>>> dual_regression.inputs.des_norm = False
>>> dual_regression.inputs.ica_components = "allFA.nii"
>>> dual_regression.inputs.ica_components_4D = "allFA.nii"
>>> dual_regression.inputs.out_dir = "my_output_directory"
>>> dual_regression.cmdline
'dual_regression allFA.nii 0 -1 10 my_output_directory functional.nii functional2.nii functional3.nii'
>>> dual_regression.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
in_files: (a list of items which are a pathlike object or string representing an existing file)
  list all subjects' preprocessed, standard-space 4D datasets
  argument: "%s", position: -1
group_IC_maps_4D: (a pathlike object or string representing an existing file)
  4D image containing spatial IC maps (melodic_IC) from the whole-group ICA analysis
argument: ``%s``, position: 1
n_perm: (an integer (int or long))
Number of permutations for randomise; set to 1 for just raw tstat
output, set to 0 to not run randomise at all.
argument: ``%i``, position: 5

[Optional]
des_norm: (a boolean, nipype default value: True)
Whether to variance-normalise the timecourses used as the stage-2
regressors; True is default and recommended
argument: ``%i``, position: 2
one_sample_group_mean: (a boolean)
perform 1-sample group-mean test instead of generic permutation test
argument: ``-1```, position: 3
design_file: (a pathlike object or string representing an existing
file)
Design matrix for final cross-subject modelling with randomise
argument: ``%s```, position: 3
con_file: (a pathlike object or string representing an existing file)
Design contrasts for final cross-subject modelling with randomise
argument: ``%s```, position: 4
out_dir: (a pathlike object or string representing a directory,
nipype default value: output)
This directory will be created to hold all output and logfiles
argument: ``%s```, position: 6
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
'NIFTI_PAIR_GZ')
FSL output type
args: (a unicode string)
Additional parameters to the command
argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {}) 
Environment variables

Outputs:

out_dir: (a pathlike object or string representing an existing
directory)

References:

None

63.6.4 FEAT

Link to code
Wraps the executable command feat.
Uses FSL feat to calculate first level stats
Inputs:

[Mandatory]
fsf_file: (a pathlike object or string representing an existing file)
File specifying the feat design spec file
argument: ``%s```, position: 0

(continues on next page)
[Optional]
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
  FSL output type
args: (a unicode string)
  Additional parameters to the command
  argument: '%s'
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
  Environment variables

Outputs:

feat_dir: (a pathlike object or string representing an existing directory)

References:
None

63.6.5 FEATModel

Link to code
Wraps the executable command feat_model.
Uses FSL feat_model to generate design.mat files

Inputs:

[Mandatory]
fsf_file: (a pathlike object or string representing an existing file)
  File specifying the feat design spec file
  argument: '%s', position: 0
ev_files: (a list of items which are a pathlike object or string
  representing an existing file)
  Event spec files generated by level1design
  argument: '%s', position: 1

[Optional]
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
  FSL output type
args: (a unicode string)
  Additional parameters to the command
  argument: '%s'
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
  Environment variables

Outputs:

design_file: (a pathlike object or string representing an existing file)
  Mat file containing ascii matrix for design
design_image: (a pathlike object or string representing an existing file)
Graphical representation of design matrix
design_cov: (a pathlike object or string representing an existing file)
Graphical representation of design covariance
con_file: (a pathlike object or string representing an existing file)
Contrast file containing contrast vectors
fcon_file: (a pathlike object or string representing a file)
Contrast file containing contrast vectors

References:
None

63.6.6 FEATRegister

Link to code
Register feat directories to a specific standard
Inputs:

[Mandatory]
feat_dirs: (a list of items which are a pathlike object or string representing an existing directory)
Lower level feat dirs
reg_image: (a pathlike object or string representing an existing file)
image to register to (will be treated as standard)

[Optional]
reg_dof: (an integer (int or long), nipype default value: 12)
registration degrees of freedom

Outputs:

fsf_file: (a pathlike object or string representing an existing file)
FSL feat specification file

63.6.7 FILMGLS

Link to code
Wraps the executable command film_gls.
Use FSL film_gls command to fit a design matrix to voxel timeseries

Examples
Initialize with no options, assigning them when calling run:

```python
>>> from nipype.interfaces import fsl
>>> fgls = fsl.FILMGLS()
>>> res = fgls.run('in_file', 'design_file', 'thresh', rn='stats') #doctest: +SKIP
```

Assign options through the inputs attribute:

```python
>>> fgls = fsl.FILMGLS()
>>> fgls.inputs.in_file = 'functional.nii'
>>> fgls.inputs.design_file = 'design.mat'
>>> fgls.inputs.threshold = 10
```
Specify options when creating an instance:

```python
gls = fsl.FILMGLS(in_file='functional.nii', design_file='design.mat',
threshold=10, results_dir='stats')
res = gls.run() #doctest: +SKIP
```

Inputs:

- **in_file**: (a pathlike object or string representing an existing file)
  - input data file
  - argument: `'\%s'`, position: `-3`

- **design_file**: (a pathlike object or string representing an existing file)
  - design matrix file
  - argument: `'\%s'`, position: `-2`

- **threshold**: (a floating point number `>= 0.0`, nipype default value: `1000.0`)
  - threshold
  - argument: `'\%f'`, position: `-1`

- **smooth_autocorr**: (a boolean)
  - Smooth auto corr estimates
  - argument: `'-sa'`

- **mask_size**: (an integer (int or long))
  - susan mask size
  - argument: `'-ms %d'`

- **brightness_threshold**: (a long integer `>= 0`)
  - susan brightness threshold, otherwise it is estimated
  - argument: `'-epith %d'`

- **full_data**: (a boolean)
  - output full data
  - argument: `'-v'`

- **autocorr_estimate_only**: (a boolean)
  - perform autocorrelation estimation only
  - argument: `'-ac'`
  - mutually exclusive: autocorr_estimate_only, fit_armodel, tukey_window, multitaper_product, use_pava, autocorr_noestimate

- **fit_armodel**: (a boolean)
  - fits autoregressive model - default is to use tukey with M=sqrt(numvols)
  - argument: `'-ar'`
  - mutually exclusive: autocorr_estimate_only, fit_armodel, tukey_window, multitaper_product, use_pava, autocorr_noestimate

- **tukey_window**: (an integer (int or long))
  - tukey window size to estimate autocorr
  - argument: `'-tukey %d'`
  - mutually exclusive: autocorr_estimate_only, fit_armodel, tukey_window, multitaper_product, use_pava, autocorr_noestimate

- **multitaper_product**: (an integer (int or long))
  - multitapering with slepian tapers and num is the time-bandwidth product
  - argument: `'-mt %d'`
  - mutually exclusive: autocorr_estimate_only, fit_armodel,
tukey_window, multitaper_product, use_pava, autocorr_noestimate
use_pava: (a boolean)
    estimates autocorr using PAVA
    argument: `-pava`
autocorr_noestimate: (a boolean)
    do not estimate autocorrs
    argument: `-noest`
    mutually_exclusive: autocorr_estimate_only, fit_armodel,
    tukey_window, multitaper_product, use_pava, autocorr_noestimate
output_pwdata: (a boolean)
    output prewhitened data and average design matrix
    argument: `-output_pwdata`
results_dir: (a pathlike object or string representing a directory,
    nipype default value: results)
    directory to store results in
    argument: `-%s`
output_type: (`NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
    'NIFTI_PAIR_GZ')
    FSL output type
args: (a unicode string)
    Additional parameters to the command
    argument: `-%s`
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})'
    Environment variables

Outputs:

param_estimates: (a list of items which are a pathlike object or
    string representing an existing file)
    Parameter estimates for each column of the design matrix
residual4d: (a pathlike object or string representing an existing file)
    Model fit residual mean-squared error for each time point
dof_file: (a pathlike object or string representing an existing file)
    degrees of freedom
sigmasquareds: (a pathlike object or string representing an existing file)
    summary of residuals, See Woolrich, et. al., 2001
results_dir: (a pathlike object or string representing an existing directory)
    directory storing model estimation output
corrections: (a pathlike object or string representing an existing file)
    statistical corrections used within FILM modeling
thresholdac: (a pathlike object or string representing an existing file)
    The FILM autocorrelation parameters
logfile: (a pathlike object or string representing an existing file)
    FILM run logfile

References:

None
63.6.8 FLAMEO

Link to code
Wraps the executable command flameo.
Use FSL flameo command to perform higher level model fits

Examples

Initialize FLAMEO with no options, assigning them when calling run:

```python
>>> from nipype.interfaces import fsl
>>> flameo = fsl.FLAMEO()
>>> flameo.inputs.cope_file = 'cope.nii.gz'
>>> flameo.inputs.var_cope_file = 'varcope.nii.gz'
>>> flameo.inputs.cov_split_file = 'cov_split.mat'
>>> flameo.inputs.design_file = 'design.mat'
>>> flameo.inputs.t_con_file = 'design.con'
>>> flameo.inputs.mask_file = 'mask.nii'
>>> flameo.inputs.run_mode = 'fe'
>>> flameo.cmdline
'flameo --copefile=cope.nii.gz --covsplitfile=cov_split.mat --designfile=design.mat --ld=stats --maskfile=mask.nii --runmode=fe --tcontrastsfile=design.con --varcopefile=varcope.nii.gz'
```

Inputs:

[Mandatory]
cope_file: (a pathlike object or string representing an existing file)
    cope regressor data file
    argument: ``--copefile=%s``
mask_file: (a pathlike object or string representing an existing file)
    mask file
    argument: ``--maskfile=%s``
design_file: (a pathlike object or string representing an existing file)
    design matrix file
    argument: ``--designfile=%s``
t_con_file: (a pathlike object or string representing an existing file)
    ascii matrix specifying t-contrasts
    argument: ``--tcontrastsfile=%s``
cov_split_file: (a pathlike object or string representing an existing file)
    ascii matrix specifying the groups the covariance is split into
    argument: ``--covsplitfile=%s``
run_mode: ('fe' or 'ols' or 'flame1' or 'flame12')
    inference to perform
    argument: ``--runmode=%s``

[Optional]
var_cope_file: (a pathlike object or string representing an existing file)
    varcope weightings data file
    argument: ``--varcopefile=%s``
dof_var_cope_file: (a pathlike object or string representing an existing file)
    dof data file for varcope data
argument: `--dofvarcopefile=%s`

f_con_file: (a pathlike object or string representing an existing file)
ascii matrix specifying f-contrasts
argument: `--fcontrastsfile=%s`

n_jumps: (an integer (int or long))
number of jumps made by mcmc
argument: `--njumps=%d`

burnin: (an integer (int or long))
number of jumps at start of mcmc to be discarded
argument: `--burnin=%d`

sample_every: (an integer (int or long))
number of jumps for each sample
argument: `--sampleevery=%d`

fix_mean: (a boolean)
fix mean for tfit
argument: `--fixmean`

infer_outliers: (a boolean)
infer outliers - not for fe
argument: `--inferoutliers`

no_pe_outputs: (a boolean)
do not output pe files
argument: `--nopeoutput`

sigma_dofs: (an integer (int or long))
sigma (in mm) to use for Gaussian smoothing the DOFs in FLAME 2. Default is 1mm, -1 indicates no smoothing
argument: `--sigma_dofs=%d`

outlier_iter: (an integer (int or long))
Number of max iterations to use when inferring outliers. Default is 12.
argument: `--ioni=%d`

log_dir: (a pathlike object or string representing a directory, nipype default value: stats)
argument: `--ld=%s`

output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
FSL output type

args: (a unicode string)
Additional parameters to the command
argument: `%%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

pes: (a list of items which are a pathlike object or string representing an existing file)
Parameter estimates for each column of the design matrix for each voxel
res4d: (a list of items which are a pathlike object or string representing an existing file)
Model fit residual mean-squared error for each time point

copes: (a list of items which are a pathlike object or string representing an existing file)
Contrast estimates for each contrast
var_copes: (a list of items which are a pathlike object or string representing an existing file)
    Variance estimates for each contrast
zstats: (a list of items which are a pathlike object or string representing an existing file)
    z-stat file for each contrast
tstats: (a list of items which are a pathlike object or string representing an existing file)
    t-stat file for each contrast
zfstats: (a list of items which are a pathlike object or string representing an existing file)
    z-stat file for each f contrast
fstats: (a list of items which are a pathlike object or string representing an existing file)
    f-stat file for each contrast
mrefvars: (a list of items which are a pathlike object or string representing an existing file)
    mean random effect variances for each contrast
tdof: (a list of items which are a pathlike object or string representing an existing file)
    temporal dof file for each contrast
weights: (a list of items which are a pathlike object or string representing an existing file)
    weights file for each contrast
stats_dir: (a pathlike object or string representing a directory)
    directory storing model estimation output

References:
None None

63.6.9 GLM

Link to code
Wraps the executable command fsl_glm.
FSL GLM:

Example

```python
>>> import nipype.interfaces.fsl as fsl
>>> glm = fsl.GLM(in_file='functional.nii', design='maps.nii', output_type='NIFTI →')
>>> glm.cmdline
'fsl_glm -i functional.nii -d maps.nii -o functional_glm.nii'
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
    input file name (text matrix or 3D/4D image file)
    argument: `'-i %s'`, position: 1
design: (a pathlike object or string representing an existing file)
    file name of the GLM design matrix (text time courses for temporal regression or an image file for spatial regression)
    argument: `'-d %s'`, position: 2
out_file: (a pathlike object or string representing a file)
    filename for GLM parameter estimates (GLM betas)
    argument: `\`-o %s`\', position: 3
contrasts: (a pathlike object or string representing an existing file)
    matrix of t-statics contrasts
    argument: `\`-c %s`\'
mask: (a pathlike object or string representing an existing file)
    mask image file name if input is image
    argument: `\`-m %s`\'
dof: (an integer (int or long))
    set degrees of freedom explicitly
    argument: `\`--dof=%d`\'
des_norm: (a boolean)
    switch on normalization of the design matrix columns to unit std deviation
    argument: `\`--des_norm`\'
dat_norm: (a boolean)
    switch on normalization of the data time series to unit std deviation
    argument: `\`--dat_norm`\'
var_norm: (a boolean)
    perform MELODIC variance-normalisation on data
    argument: `\`--vn`\'
demean: (a boolean)
    switch on demeaning of design and data
    argument: `\`--demean`\'
out_cope: (a pathlike object or string representing a file)
    output file name for COPE (either as txt or image
    argument: `\`--out_cope=%s`\'
out_z_name: (a pathlike object or string representing a file)
    output file name for Z-stats (either as txt or image
    argument: `\`--out_z=%s`\'
out_t_name: (a pathlike object or string representing a file)
    output file name for t-stats (either as txt or image
    argument: `\`--out_t=%s`\'
out_p_name: (a pathlike object or string representing a file)
    output file name for p-values of Z-stats (either as text file or image
    argument: `\`--out_p=%s`\'
out_f_name: (a pathlike object or string representing a file)
    output file name for F-value of full model fit
    argument: `\`--out_f=%s`\'
out_pf_name: (a pathlike object or string representing a file)
    output file name for p-value for full model fit
    argument: `\`--out_pf=%s`\'
out_res_name: (a pathlike object or string representing a file)
    output file name for residuals
    argument: `\`--out_res=%s`\'
out_varcb_name: (a pathlike object or string representing a file)
    output file name for variance of COPEs
    argument: `\`--out_varcb=%s`\'
out_sigsq_name: (a pathlike object or string representing a file)
    output file name for residual noise variance sigma-square
    argument: `\`--out_sigsq=%s`\'
out_data_name: (a pathlike object or string representing a file)
    output file name for pre-processed data
    argument: '``--out_data=%s``'
out_vnscales_name: (a pathlike object or string representing a file)
    output file name for scaling factors for variance normalisation
    argument: '``--out_vnscales=%s``'
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
    'NIFTI_PAIR_GZ')
    FSL output type
args: (a unicode string)
    Additional parameters to the command
    argument: '``%s``'
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})
    Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
    file name of GLM parameters (if generated)
out_cope: (a list of items which are a pathlike object or string
    representing an existing file)
    output file name for COPEs (either as text file or image)
out_z: (a list of items which are a pathlike object or string
    representing an existing file)
    output file name for COPEs (either as text file or image)
out_t: (a list of items which are a pathlike object or string
    representing an existing file)
    output file name for t-stats (either as text file or image)
out_p: (a list of items which are a pathlike object or string
    representing an existing file)
    output file name for p-values of Z-stats (either as text file or
    image)
out_f: (a list of items which are a pathlike object or string
    representing an existing file)
    output file name for F-value of full model fit
out_pf: (a list of items which are a pathlike object or string
    representing an existing file)
    output file name for p-value for full model fit
out_res: (a list of items which are a pathlike object or string
    representing an existing file)
    output file name for residuals
out_varcb: (a list of items which are a pathlike object or string
    representing an existing file)
    output file name for variance of COPEs
out_sigsq: (a list of items which are a pathlike object or string
    representing an existing file)
    output file name for residual noise variance sigma-square
out_data: (a list of items which are a pathlike object or string
    representing an existing file)
    output file for preprocessed data
out_vnscales: (a list of items which are a pathlike object or string
    representing an existing file)
    output file name for scaling factors for variance normalisation
References:
None

### 63.6.10 L2Model

**Link to code**
Generate subject specific second level model

**Examples**

```python
>>> from nipype.interfaces.fsl import L2Model
>>> model = L2Model(num_copes=3)  # 3 sessions
```

**Inputs:**

<table>
<thead>
<tr>
<th>Mandatory</th>
<th>num_copes: (a long integer &gt;= 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>number of copes to be combined</td>
</tr>
</tbody>
</table>

**Outputs:**

| design_mat: (a pathlike object or string representing an existing file) |
| design contrast file |
| design_grp: (a pathlike object or string representing an existing file) |
| design group file |

### 63.6.11 Level1Design

**Link to code**
Generate FEAT specific files

**Examples**

```python
>>> level1design = Level1Design()
>>> level1design.inputs.interscan_interval = 2.5
>>> level1design.inputs.bases = {'dgamma': {'derivs': False}}
>>> level1design.inputs.session_info = 'session_info.npz'
>>> level1design.run()  # doctest: +SKIP
```

**Inputs:**

<table>
<thead>
<tr>
<th>Mandatory</th>
<th>interscan_interval: (a float)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Interscan interval (in secs)</td>
</tr>
<tr>
<td>session_info: (any value)</td>
<td></td>
</tr>
<tr>
<td>session specific information generated by <code>modelgen.SpecifyModel</code></td>
<td></td>
</tr>
<tr>
<td>bases: (a dictionary with keys which are 'dgamma' and with values which are a dictionary with keys which are 'derivs' and with values which are a boolean or a dictionary with keys which are 'gamma' and with values which are a dictionary with keys which are 'derivs' or 'gammasigma' or 'gammadelay' and with values which are</td>
<td></td>
</tr>
</tbody>
</table>

(continues on next page)
any value or a dictionary with keys which are 'custom' and with values which are a dictionary with keys which are 'bfcustomopath' and with values which are a unicode string or a dictionary with keys which are 'none' and with values which are a dictionary with keys which are any value and with values which are any value or a dictionary with keys which are 'none' and with values which are None.

name of basis function and options e.g., {'dgamma': {'derivs': True}}

model_serial_correlations: (a boolean)

Option to model serial correlations using an autoregressive estimator (order 1). Setting this option is only useful in the context of the fsf file. If you set this to False, you need to repeat this option for FILMGLS by setting autocorr_noestim to True.

[Optional]

orthogonalization: (a dictionary with keys which are an integer (int or long) and with values which are a dictionary with keys which are an integer (int or long) and with values which are a boolean or an integer (int or long), nipype default value: {})

which regressors to make orthogonal e.g., {1: {0:0,1:0,2:0}, 2: {0:1,1:1,2:0}} to make the second regressor in a 2-regressor model orthogonal to the first.

contrasts: (a list of items which are a tuple of the form: (a unicode string, 'T', a list of items which are a unicode string, a list of items which are a float) or a tuple of the form: (a unicode string, 'T', a list of items which are a unicode string, a list of items which are a float) or a tuple of the form: (a unicode string, 'F', a list of items which are a unicode string, a list of items which are a float) or a tuple of the form: (a unicode string, 'T', a list of items which are a unicode string, a list of items which are a float) or a tuple of the form: (a unicode string, 'T', a list of items which are a unicode string, a list of items which are a float) or a tuple of the form: (a unicode string, 'T', a list of items which are a unicode string, a list of items which are a float) or a tuple of the form: (a unicode string, 'T', a list of items which are a unicode string, a list of items which are a float) or a tuple of the form: (a unicode string, 'T', a list of items which are a unicode string, a list of items which are a float) or a tuple of the form: (a unicode string, 'T', a list of items which are a unicode string, a list of items which are a float) or a tuple of the form: (a unicode string, 'T', a list of items which are a unicode string, a list of items which are a float) or a tuple of the form: (a unicode string, 'T', a list of items which are a unicode string, a list of items which are a float) or a tuple of the form: (a unicode string, 'T', a list of items which are a unicode string, a list of items which are a float) or a tuple of the form: (a unicode string, 'T', a list of items which are a unicode string, a list of items which are a float))

List of contrasts with each contrast being a list of the form - 

[('name', 'stat', [condition list], [weight list], [session list])].

if session list is None or not provided, all sessions are used. For F contrasts, the condition list should contain previously defined T-contrasts.

Outputs:

fsf_files: (a list of items which are a pathlike object or string representing an existing file)

FSL feat specification files

ev_files: (a list of items which are a list of items which are a pathlike object or string representing an existing file)

condition information files

63.6.12 MELODIC

Link to code

Wraps the executable command melodic.

Multivariate Exploratory Linear Optimised Decomposition into Independent Components
Examples

```python
>>> melodic_setup = MELODIC()
>>> melodic_setup.inputs.approach = 'tica'
>>> melodic_setup.inputs.in_files = ['functional.nii', 'functional2.nii',
        'functional3.nii']
>>> melodic_setup.inputs.no_bet = True
>>> melodic_setup.inputs.bg_threshold = 10
>>> melodic_setup.inputs.tr_sec = 1.5
>>> melodic_setup.inputs.mm_thresh = 0.5
>>> melodic_setup.inputs.out_stats = True
>>> melodic_setup.inputs.t_des = 'timeDesign.mat'
>>> melodic_setup.inputs.t_con = 'timeDesign.con'
>>> melodic_setup.inputs.s_des = 'subjectDesign.mat'
>>> melodic_setup.inputs.s_con = 'subjectDesign.con'
>>> melodic_setup.inputs.out_dir = 'groupICA.out'
>>> melodic_setup.cmdline
'melodic -i functional.nii,functional2.nii,functional3.nii -a tica --bgthreshold=10.000000 --mmthresh=0.500000 --nobet -o groupICA.out --Ostats --Scon=subjectDesign.con --Sdes=subjectDesign.mat --Tcon=timeDesign.con --Tdes=timeDesign.mat --tr=1.500000'
>>> melodic_setup.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]

- **in_files**: (a list of items which are a pathlike object or string representing an existing file)
  - input file names (either single file name or a list)
  - argument: `'-i %s'`, position: 0

[Optional]

- **out_dir**: (a pathlike object or string representing a directory)
  - output directory name
  - argument: `'-o %s'`

- **mask**: (a pathlike object or string representing an existing file)
  - file name of mask for thresholding
  - argument: `'-m %s'`

- **no_mask**: (a boolean)
  - switch off masking
  - argument: `'-nomask'`

- **update_mask**: (a boolean)
  - switch off mask updating
  - argument: `'-update_mask'`

- **no_bet**: (a boolean)
  - switch off BET
  - argument: `'-nobet'`

- **bg_threshold**: (a float)
  - brain/non-brain threshold used to mask non-brain voxels, as a percentage (only if --nobet selected)
  - argument: `'-bgthreshold=%f'`

- **dim**: (an integer (int or long))
  - dimensionality reduction into #num dimensions (default: automatic estimation)
  - argument: `'-d %d'`

- **dim_est**: (a unicode string)
  - use specific dim. estimation technique: lap, bic, mdl, aic, mean
  - (default: lap)

(continues on next page)
sep_whiten: (a boolean)
switch on separate whitening
argument: `--sep_whiten`

sep_vn: (a boolean)
switch off joined variance normalization
argument: `--sep_vn`

migp: (a boolean)
switch on MIGP data reduction
argument: `--migp`

migpN: (an integer (int or long))
number of internal Eigenmaps
argument: `--migpN %d`

migp_shuffle: (a boolean)
randomise MIGP file order (default: TRUE)
argument: `--migp_shuffle`

migp_factor: (an integer (int or long))
Internal Factor of mem-threshold relative to number of Eigenmaps
(default: 2)
argument: `--migp_factor %d`

num_ICs: (an integer (int or long))
number of IC's to extract (for deflation approach)
argument: `--n %d`

approach: (a unicode string)
approach for decomposition, 2D: defl, symm (default), 3D: tica
(default), concat
argument: `--a %s`

non_linearity: (a unicode string)
nonlinearity: gauss, tanh, pow3, pow4
argument: `--nl=%s`

var_norm: (a boolean)
switch off variance normalization
argument: `--vn`

pbsc: (a boolean)
switch off conversion to percent BOLD signal change
argument: `--pbsc`

cov_weight: (a float)
voxel-wise weights for the covariance matrix (e.g. segmentation
information)
argument: `--covarweight=%f`

epsilon: (a float)
minimum error change
argument: `--eps=%f`

epsilonS: (a float)
minimum error change for rank-1 approximation in TICA
argument: `--epsS=%f`

maxit: (an integer (int or long))
maximum number of iterations before restart
argument: `--maxit=%d`

max_restart: (an integer (int or long))
maximum number of restarts
argument: `--maxrestart=%d`

mm_thresh: (a float)
threshold for Mixture Model based inference
argument: `--mmthresh=%f`

no_mm: (a boolean)
switch off mixture modelling on IC maps
ICs:  (a pathlike object or string representing an existing file)  
   filename of the IC components file for mixture modelling  
   argument: `--ICs=%s`  
mix:  (a pathlike object or string representing an existing file)  
   mixing matrix for mixture modelling / filtering  
   argument: `--mix=%s`  
smode:  (a pathlike object or string representing an existing file)  
   matrix of session modes for report generation  
   argument: `--smode=%s`  
rem_cmp:  (a list of items which are an integer (int or long))  
   component numbers to remove  
   argument: `-%d`  
report:  (a boolean)  
   generate Melodic web report  
   argument: `--report`  
bg_image:  (a pathlike object or string representing an existing file)  
   specify background image for report (default: mean image)  
   argument: `--bgimage=%s`  
tr_sec:  (a float)  
   TR in seconds  
   argument: `--tr=%f`  
log_power:  (a boolean)  
   calculate log of power for frequency spectrum  
   argument: `--logPower`  
t_des:  (a pathlike object or string representing an existing file)  
   design matrix across time-domain  
   argument: `--Tdes=%s`  
t_con:  (a pathlike object or string representing an existing file)  
   t-contrast matrix across time-domain  
   argument: `--Tcon=%s`  
s_des:  (a pathlike object or string representing an existing file)  
   design matrix across subject-domain  
   argument: `--Sdes=%s`  
s_con:  (a pathlike object or string representing an existing file)  
   t-contrast matrix across subject-domain  
   argument: `--Scon=%s`  
out_all:  (a boolean)  
   output everything  
   argument: `--Oall``  
out_unmix:  (a boolean)  
   output unmixing matrix  
   argument: `--Ounmix``  
out_stats:  (a boolean)  
   output thresholded maps and probability maps  
   argument: `--Ostats`  
out_pca:  (a boolean)  
   output PCA results  
   argument: `--Opca``  
out_white:  (a boolean)  
   output whitening/dewhitening matrices  
   argument: `--Owhite``  
out_orig:  (a boolean)  
   output the original ICs  
   argument: `--Oorig``  
out_mean:  (a boolean)  
   output mean volume  
(continues on next page)
argument: ``--Omean``
report_maps: (a unicode string)
control string for spatial map images (see slicer)
argument: ``--report_maps=%s``
remove_deriv: (a boolean)
removes every second entry in paradigm file (EV derivatives)
argument: ``--remove_deriv``
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
'NIFTI_PAIR_GZ')
FSL output type
args: (a unicode string)
Additional parameters to the command
argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_dir: (a pathlike object or string representing an existing
directory)
report_dir: (a pathlike object or string representing an existing
directory)

References:

None

63.6.13 MultipleRegressDesign

Link to code
Generate multiple regression design

Note: FSL does not demean columns for higher level analysis.

Please see FSL documentation for more details on model specification for higher level analysis.

Examples

```python
>>> from nipype.interfaces.fsl import MultipleRegressDesign
>>> model = MultipleRegressDesign()
>>> model.inputs.contrasts = [['group mean', 'T', ['reg1'], [1]]]
>>> model.inputs.regressors = dict(reg1=[1, 1, 1], reg2=[2., -4, 3])
>>> model.run() # doctest: +SKIP
```

Inputs:

[Optional]

contrasts: (a list of items which are a tuple of the form: (a unicode
string, 'T', a list of items which are a unicode string, a list of
items which are a float) or a tuple of the form: (a unicode
string, 'F', a list of items which are a tuple of the form: (a
unicode string, 'T', a list of items which are a unicode string, a
list of items which are a float)))
List of contrasts with each contrast being a list of the form -


([{'name', 'stat', [condition list], [weight list]}]. If session list is None or not provided, all sessions are used. For F contrasts, the condition list should contain previously defined T-contrasts without any weight list.

regressors: (a dictionary with keys which are a unicode string and with values which are a list of items which are a float) dictionary containing named lists of regressors

[Optional]
groups: (a list of items which are an integer (int or long)) list of group identifiers (defaults to single group)

**Outputs:**

design_mat: (a pathlike object or string representing an existing file)

design_con: (a pathlike object or string representing an existing file)

design_fts: (a pathlike object or string representing an existing file)

design_grp: (a pathlike object or string representing an existing file)

### 63.6.14 Randomise

**Link to code**

Wraps the executable command `randomise`.

FSL Randomise: feeds the 4D projected FA data into GLM modelling and thresholding in order to find voxels which correlate with your model

**Example**

```python
>>> import nipype.interfaces.fsl as fsl
>>> rand = fsl.Randomise(in_file='allFA.nii', mask = 'mask.nii', tcon='design.con', design_mat='design.mat')
>>> rand.cmdline
'randomise -i allFA.nii -o "randomise" -d design.mat -t design.con -m mask.nii'
```

**Inputs:**

[Mandatory]
in_file: (a pathlike object or string representing an existing file)

4D input file

argument: ``-i %s``, position: 0

[Optional]
base_name: (a unicode string, nipype default value: randomise)

the rootname that all generated files will have

argument: ``-o "%s"``, position: 1

design_mat: (a pathlike object or string representing an existing file)

design matrix file

(continues on next page)
argument: `\'-d %s\'`, position: 2
tcon: (a pathlike object or string representing an existing file)
t contrasts file
argument: `\'-t %s\'`, position: 3
fcon: (a pathlike object or string representing an existing file)
f contrasts file
argument: `\'-f %s\'`
mask: (a pathlike object or string representing an existing file)
mask image
argument: `\'-m %s\'`
x_block_labels: (a pathlike object or string representing an existing file)
exchangeability block labels file
argument: `\'-e %s\'`
demean: (a boolean)
demean data temporally before model fitting
argument: `\'-D\'`
one_sample_group_mean: (a boolean)
perform 1-sample group-mean test instead of generic permutation test
argument: `\'-1\'`
show_total_perms: (a boolean)
print out how many unique permutations would be generated and exit
argument: `\'-q\'`
show_info_parallel_mode: (a boolean)
print out information required for parallel mode and exit
argument: `\'-Q\'`
vox_p_values: (a boolean)
output voxelwise (corrected and uncorrected) p-value images
argument: `\'-x\'`
tfce: (a boolean)
carry out Threshold-Free Cluster Enhancement
argument: `\'-T\'`
tfce2D: (a boolean)
carry out Threshold-Free Cluster Enhancement with 2D optimisation
argument: `\'-T2\'`
f_only: (a boolean)
calculate f-statistics only
argument: `\'-f_only\'`
raw_stats_imgs: (a boolean)
output raw (unpermuted) statistic images
argument: `\'-R\'`
p_vec_n_dist_files: (a boolean)
output permutation vector and null distribution text files
argument: `\'-P\'`
num_perm: (an integer (int or long))
number of permutations (default 5000, set to 0 for exhaustive)
argument: `\'-n %d\'`
seed: (an integer (int or long))
specific integer seed for random number generator
argument: `\'-seed=%d\'`
var_smooth: (an integer (int or long))
use variance smoothing (std is in mm)
argument: `\'-v %d\'`
c_thresh: (a float)
carry out cluster-based thresholding
argument: `\'-c %.1f\'`
cm_thresh: (a float)
carry out cluster-mass-based thresholding
argument: ``--C %.1f``
f_c_thresh: (a float)
carry out f cluster thresholding
argument: ``--F %.2f``
f_cm_thresh: (a float)
carry out f cluster-mass thresholding
argument: ``--S %.2f``
tfce_H: (a float)
TFCE height parameter (default=2)
argument: ``--tfce_H=%.2f``
tfce_E: (a float)
TFCE extent parameter (default=0.5)
argument: ``--tfce_E=%.2f``
tfce_C: (a float)
TFCE connectivity (6 or 26; default=6)
argument: ``--tfce_C=%.2f``
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
FSL output type
args: (a unicode string)
Additional parameters to the command
argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

tstat_files: (a list of items which are a pathlike object or string representing an existing file)
t contrast raw statistic
fstat_files: (a list of items which are a pathlike object or string representing an existing file)
f contrast raw statistic
t_p_files: (a list of items which are a pathlike object or string representing an existing file)
f contrast uncorrected p values files
f_p_files: (a list of items which are a pathlike object or string representing an existing file)
f contrast uncorrected p values files
t_corrected_p_files: (a list of items which are a pathlike object or string representing an existing file)
t contrast FWE (Family-wise error) corrected p values files
f_corrected_p_files: (a list of items which are a pathlike object or string representing an existing file)
f contrast FWE (Family-wise error) corrected p values files

References:
None

63.6.15 SMM

Link to code
Wraps the executable command mm --ld=logdir.

Inputs:

[Mandatory]
spatial_data_file: (a pathlike object or string representing an existing file)
  statistics spatial map
  argument: ``--sdf="%s``', position: 0
mask: (a pathlike object or string representing an existing file)
  mask file
  argument: ``--mask="%s``', position: 1

[Optional]
no_deactivation_class: (a boolean)
  enforces no deactivation class
  argument: ``--zfstatmode``', position: 2
output_type: ("NIIFTI" or 'NIIFTI_PAIR' or 'NIIFTI_GZ' or 'NIIFTI_PAIR_GZ')
  FSL output type
args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  Environment variables

Outputs:

null_p_map: (a pathlike object or string representing an existing file)
activation_p_map: (a pathlike object or string representing an existing file)
deactivation_p_map: (a pathlike object or string representing an existing file)

References:
None

63.6.16 SmoothEstimate

Link to code
Wraps the executable command smoothest.
Estimates the smoothness of an image

Examples

```python
>>> est = SmoothEstimate()
>>> est.inputs.zstat_file = 'zstat1.nii.gz'
>>> est.inputs.mask_file = 'mask.nii'
>>> est.cmdline
'smoothest --mask=mask.nii --zstat=zstat1.nii.gz'
```

Inputs:
**dof**: (an integer (int or long))
   number of degrees of freedom
   argument: `--dof=%d`
   mutually_exclusive: zstat_file

**mask_file**: (a pathlike object or string representing an existing file)
   brain mask volume
   argument: `--mask=%s`

**[Optional]**
**residual_fit_file**: (a pathlike object or string representing an existing file)
   residual-fit image file
   argument: `--res=%s`
   requires: dof

**zstat_file**: (a pathlike object or string representing an existing file)
   zstat image file
   argument: `--zstat=%s`
   mutually_exclusive: dof

**output_type**: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
   FSL output type

**args**: (a unicode string)
   Additional parameters to the command
   argument: `%s`

**environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
   Environment variables

**Outputs:**

**dlh**: (a float)
   smoothness estimate sqrt(det(Lambda))

**volume**: (an integer (int or long))
   number of voxels in mask

**resels**: (a float)
   number of resels

**References:**

None

**63.6.17 load_template()**

**Link to code**
Load a template from the model_templates directory

**Parameters**

**name**  [str] The name of the file to load

**Returns**

template : string.Template
63.7 interfaces.fsl.possum

63.7.1 B0Calc

Link to code
Wraps the executable command b0calc.
B0 inhomogeneities occur at interfaces of materials with different magnetic susceptibilities, such as tissue-air interfaces. These differences lead to distortion in the local magnetic field, as Maxwell’s equations need to be satisfied. An example of B0 inhomogeneity is the first volume of the 4D volume `$FSLDIR/data/possum/b0_ppm.nii.gz`.

Examples

```python
>>> from nipype.interfaces.fsl import B0Calc
>>> b0calc = B0Calc()
>>> b0calc.inputs.in_file = 'tissue+air_map.nii'
>>> b0calc.inputs.z_b0 = 3.0
>>> b0calc.inputs.output_type = "NIFTI_GZ"
>>> b0calc.cmdline
'b0calc -i tissue+air_map.nii -o tissue+air_map_b0field.nii.gz --chi0=4.000000e-07 -d -9.450000e-06 --extendboundary=1.00 --b0x=0.00 --gx=0.0000 --b0y=0.00 --gy=0.0000 --b0=3.00 --gz=0.0000'
```

Inputs:

<table>
<thead>
<tr>
<th>Mandatory</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_file: (a pathlike object or string representing an existing file)</td>
<td>out_file: (a pathlike object or string representing a file)</td>
</tr>
<tr>
<td>filename of input image (usually a tissue/air segmentation)</td>
<td>filename of B0 output volume</td>
</tr>
</tbody>
</table>

x_grad: (a float, nipype default value: 0.0)
Value for zeroth-order x-gradient field (per mm)
argument: `--gx=%0.4f`

y_grad: (a float, nipype default value: 0.0)
Value for zeroth-order y-gradient field (per mm)
argument: `--gy=%0.4f`

z_grad: (a float, nipype default value: 0.0)
Value for zeroth-order z-gradient field (per mm)
argument: `--gz=%0.4f`

x_b0: (a float, nipype default value: 0.0)
Value for zeroth-order B0 field (x-component), in Tesla
argument: `--b0x=%0.2f`
mutually_exclusive: xyz_b0

y_b0: (a float, nipype default value: 0.0)
Value for zeroth-order B0 field (y-component), in Tesla
argument: `--b0y=%0.2f`
mutually_exclusive: xyz_b0

z_b0: (a float, nipype default value: 1.0)
Value for zeroth-order B0 field (z-component), in Tesla
argument: `--b0z=%0.2f`
mutually_exclusive: xyz_b0

xyz_b0: (a tuple of the form: (a float, a float, a float))
Zeroth-order B0 field in Tesla

(continues on next page)
argument: ``--b0x=%0.2f --b0y=%0.2f --b0z=%0.2f``
mutually_exclusive: x_b0, y_b0, z_b0
delta: (a float, nipype default value: -9.45e-06)
Delta value (chi_tissue - chi_air)
argument: ``'-d %e'``
chi_air: (a float, nipype default value: 4e-07)
susceptibility of air
argument: ``'--chi0=%e'``
compute_xyz: (a boolean, nipype default value: False)
calculate and save all 3 field components (i.e. x,y,z)
argument: ``'--xyz'``
extendboundary: (a float, nipype default value: 1.0)
Relative proportion to extend voxels at boundary
argument: ``'--extendboundary=%0.2f'``
directconv: (a boolean, nipype default value: False)
use direct (image space) convolution, not FFT
argument: ``'--directconv'``
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
FSL output type
args: (a unicode string)
Additional parameters to the command
argument: ``'%-s'``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

cout_file: (a pathlike object or string representing an existing file)
filename of B0 output volume

References:
None

63.8 interfaces.fsl.preprocess

63.8.1 ApplyWarp

Link to code
Wraps the executable command applywarp.
FSL’s applywarp wrapper to apply the results of a FNIRT registration

Examples

```python
>>> from nipype.interfaces import fsl
>>> from nipype.testing import example_data
>>> aw = fsl.ApplyWarp()
>>> aw.inputs.in_file = example_data('structural.nii')
>>> aw.inputs.ref_file = example_data('mni.nii')
>>> aw.inputs.field_file = 'my_coefficients_field.nii' #doctest: +SKIP
>>> res = aw.run() #doctest: +SKIP
```

Inputs:
in_file: (a pathlike object or string representing an existing file)
  image to be warped
  argument: ``--in=%s``, position: 0
ref_file: (a pathlike object or string representing an existing file)
  reference image
  argument: ``--ref=%s``, position: 1

[Optional]
out_file: (a pathlike object or string representing a file)
  output filename
  argument: ``--out=%s``, position: 2
field_file: (a pathlike object or string representing an existing file)
  file containing warp field
  argument: ``--warp=%s``
abswarp: (a boolean)
  treat warp field as absolute: \(x' = w(x)\)
  argument: ``--abs``
  mutually_exclusive: relwarp
relwarp: (a boolean)
  treat warp field as relative: \(x' = x + w(x)\)
  argument: ``--rel``, position: -1
  mutually_exclusive: abswarp
datatype: ('char' or 'short' or 'int' or 'float' or 'double')
  Force output data type [char short int float double].
  argument: ``--datatype=%s``
supersample: (a boolean)
  intermediary supersampling of output, default is off
  argument: ``--super``
superlevel: ('a' or an integer (int or long))
  level of intermediary supersampling, a for 'automatic' or integer level. Default = 2
  argument: ``--superlevel=%s``
premat: (a pathlike object or string representing an existing file)
  filename for pre-transform (affine matrix)
  argument: ``--premat=%s``
postmat: (a pathlike object or string representing an existing file)
  filename for post-transform (affine matrix)
  argument: ``--postmat=%s``
mask_file: (a pathlike object or string representing an existing file)
  filename for mask image (in reference space)
  argument: ``--mask=%s``
interp: ('nn' or 'trilinear' or 'sinc' or 'spline')
  interpolation method
  argument: ``--interp=%s``
  position: -2
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
  FSL output type
args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
References:

None

63.8.2 ApplyXFM

Link to code
Wraps the executable command flirt. Currently just a light wrapper around FLIRT, with no modifications. ApplyXFM is used to apply an existing transform to an image.

Examples

```python
>>> import nipype.interfaces.fsl as fsl
>>> from nipype.testing import example_data

>>> applyxfm = fsl.preprocess.ApplyXFM()
>>> applyxfm.inputs.in_file = example_data('structural.nii')
>>> applyxfm.inputs.in_matrix_file = example_data('trans.mat')
>>> applyxfm.inputs.out_file = 'newfile.nii'
>>> applyxfm.inputs.reference = example_data('mni.nii')
>>> applyxfm.inputs.apply_xfm = True
>>> result = applyxfm.run() # doctest: +SKIP
```

Inputs:

|Mandatory|
in_file: (a pathlike object or string representing an existing file)
  - input file
    - argument: ``-in %s``, position: 0

|Reference|
|reference: (a pathlike object or string representing an existing file)
  - reference file
    - argument: ``-ref %s``, position: 1

|Optional|
|apply_xfm: (a boolean, nipype default value: True)
  - apply transformation supplied by in_matrix_file or uses_qform to use
    the affine matrix stored in the reference header
    - argument: ``-applyxfm``

|Output|
|out_file: (a pathlike object or string representing a file)
  - registered output file
    - argument: ``-out %s``, position: 2

|Matrix|
|out_matrix_file: (a pathlike object or string representing a file)
  - output affine matrix in 4x4 ascii format
    - argument: ``-omat %s``, position: 3

|Log|
|out_log: (a pathlike object or string representing a file)
  - output log
    - requires: save_log

|Matrix Inverse|
|in_matrix_file: (a pathlike object or string representing a file)
  - input 4x4 affine matrix
    - argument: ``-init %s``

|Isotropic|
|apply_isoxfm: (a float)
  - as applyxfm but forces isotropic resampling
```

(continues on next page)
argument: `''-applyisoxfm %f``
mutually_exclusive: apply_xfm
datatype: (`char' or 'short' or 'int' or 'float' or 'double')
force output data type
argument: `''-datatype %s``

cost: (`mutualinfo' or 'corratio' or 'normcorr' or 'normmi' or
'leastsq' or 'labeldiff' or 'bbr')
cost function
argument: `''-cost %s``
cost_func: (`mutualinfo' or 'corratio' or 'normcorr' or 'normmi' or
'leastsq' or 'labeldiff' or 'bbr')
cost function
argument: `''-searchcost %s``
uses_qform: (a boolean)
initialize using sform or qform
argument: `''-usesqform``
display_init: (a boolean)
display initial matrix
argument: `''-displayinit``
angle_rep: (`quaternion' or 'euler')
representation of rotation angles
argument: `''-anglerep %s``
interp: (`trilinear' or 'nearestneighbour' or 'sinc' or 'spline')
final interpolation method used in reslicing
argument: `''-interp %s``
sinc_width: (an integer (int or long))
full-width in voxels
argument: `''-sincwidth %d``
sinc_window: (`rectangular' or 'hanning' or 'blackman')
sinc window
argument: `''-sincwindow %s``
bins: (an integer (int or long))
number of histogram bins
argument: `''-bins %d``
dof: (an integer (int or long))
number of transform degrees of freedom
argument: `''-dof %d``
no_resample: (a boolean)
do not change input sampling
argument: `''-noresample``
force_scaling: (a boolean)
force rescaling even for low-res images
argument: `''-forcescaling``
min_sampling: (a float)
set minimum voxel dimension for sampling
argument: `''-minsampling %f``
padding_size: (an integer (int or long))
for applyxfm: interpolates outside image by size
argument: `''-paddingsize %d``
searchr_x: (a list of from 2 to 2 items which are an integer (int or long))
search angles along x-axis, in degrees
argument: `''-searchrx %s``
searchr_y: (a list of from 2 to 2 items which are an integer (int or long))
search angles along y-axis, in degrees
argument: `''-searchry %s``
searchr_z: (a list of from 2 to 2 items which are an integer (int or long))
    search angles along z-axis, in degrees
    argument: `''-searchrz %s''`
no_search: (a boolean)
    set all angular searches to ranges 0 to 0
    argument: `''-nosearch''`
coarse_search: (an integer (int or long))
    coarse search delta angle
    argument: `''-coarsesearch %d''`
fine_search: (an integer (int or long))
    fine search delta angle
    argument: `''-finesearch %d''`
schedule: (a pathlike object or string representing an existing file)
    replaces default schedule
    argument: `''-schedule %s''`
ref_weight: (a pathlike object or string representing an existing file)
    File for reference weighting volume
    argument: `''-refweight %s''`
in_weight: (a pathlike object or string representing an existing file)
    File for input weighting volume
    argument: `''-inweight %s''`
no_clamp: (a boolean)
    do not use intensity clamping
    argument: `''-noclamp''`
no_resample_blur: (a boolean)
    do not use blurring on downsampling
    argument: `''-noresampblur''`
rigid2D: (a boolean)
    use 2D rigid body mode - ignores dof
    argument: `''-2D''`
save_log: (a boolean)
    save to log file
verbose: (an integer (int or long))
    verbose mode, 0 is least
    argument: `''-verbose %d''`
bgvalue: (a float)
    use specified background value for points outside FOV
    argument: `''-setbackground %f''`
wm_seg: (a pathlike object or string representing a file)
    white matter segmentation volume needed by BBR cost function
    argument: `''-wmseg %s''`
wmcords: (a pathlike object or string representing a file)
    white matter boundary coordinates for BBR cost function
    argument: `''-wmcords %s''`
wmnorms: (a pathlike object or string representing a file)
    white matter boundary normals for BBR cost function
    argument: `''-wmnorms %s''`
fieldmap: (a pathlike object or string representing a file)
    fieldmap image in rads/s - must be already registered to the
    reference image
    argument: `''-fieldmap %s''`
fieldmapmask: (a pathlike object or string representing a file)
    mask for fieldmap image
    argument: `''-fieldmapmask %s''`
pedir: (an integer (int or long))
    phase encode direction of EPI - 1/2/3=x/y/z & -1/-2/-3=-x/-y/-z
    argument: `--pedir %d`

echospacing: (a float)
    value of EPI echo spacing - units of seconds
    argument: `--echospacing %f`

bbrtype: ('signed' or 'global_abs' or 'local_abs')
    type of bbr cost function: signed [default], global_abs, local_abs
    argument: `--bbrtype %s`

bbrslope: (a float)
    value of bbr slope
    argument: `--bbrslope %f`

output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
    FSL output type

args: (a unicode string)
    Additional parameters to the command
    argument: `--%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
    Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
    path/name of registered file (if generated)

out_matrix_file: (a pathlike object or string representing an existing file)
    path/name of calculated affine transform (if generated)

out_log: (a pathlike object or string representing a file)
    path/name of output log (if generated)

References:

None

63.8.3 BET

Link to code

Wraps the executable command bet.
FSL BET wrapper for skull stripping
For complete details, see the BET Documentation.

Examples

```python
>>> from nipype.interfaces import fsl
>>> btr = fsl.BET()
>>> btr.inputs.in_file = 'structural.nii'
>>> btr.inputs.frac = 0.7
>>> btr.inputs.out_file = 'brain_anat.nii'
>>> btr.cmdline
'bet structural.nii brain_anat.nii -f 0.70'
>>> res = btr.run() # doctest: +SKIP
```

Inputs:
in_file: (a pathlike object or string representing an existing file)
   input file to skull strip
   argument: ``%s``, position: 0

out_file: (a pathlike object or string representing a file)
   name of output skull stripped image
   argument: ``%s``, position: 1

do_not_output: (a boolean)
   Don't generate segmented output
   argument: ``-n``

frac: (a float)
   fractional intensity threshold
   argument: ``-f %.2f``

vertical_gradient: (a float)
   vertical gradient in fractional intensity threshold (-1, 1)
   argument: ``-g %.2f``

radius: (an integer (int or long))
   head radius
   argument: ``-r %d``

center: (a list of at most 3 items which are an integer (int or long))
   center of gravity in voxels
   argument: ``-c %s``

threshold: (a boolean)
   apply thresholding to segmented brain image and mask
   argument: ``-t``

mesh: (a boolean)
   generate a vtk mesh brain surface
   argument: ``-e``

robust: (a boolean)
   robust brain centre estimation (iterates BET several times)
   argument: ``-R``

remove_eyes: (a boolean)
   eye & optic nerve cleanup (can be useful in SIENA)
   argument: ``-S``

mutually_exclusive: functional, reduce_bias, robust, padding,
   remove_eyes, surfaces, t2_guided

mutually_exclusive: functional, reduce_bias, robust, padding,
   remove_eyes, surfaces, t2_guided

surfaces: (a boolean)
run bet2 and then betsurf to get additional skull and scalp surfaces
(includes registrations)
argument: `'-A'`
mutually_exclusive: functional, reduce_bias, robust, padding,
remove_eyes, surfaces, t2_guided
t2_guided: (a pathlike object or string representing a file)
as with creating surfaces, when also feeding in non-brain-extracted
T2 (includes registrations)
argument: `'-A2 %s'`
mutually_exclusive: functional, reduce_bias, robust, padding,
remove_eyes, surfaces, t2_guided
functional: (a boolean)
apply to 4D fMRI data
argument: `'-F'`
mutually_exclusive: functional, reduce_bias, robust, padding,
remove_eyes, surfaces, t2_guided
reduce_bias: (a boolean)
bias field and neck cleanup
argument: `'-B'`
mutually_exclusive: functional, reduce_bias, robust, padding,
remove_eyes, surfaces, t2_guided
output_type: (`'NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
'NIFTI_PAIR_GZ')
FSL output type
args: (a unicode string)
Additional parameters to the command
argument: `'%s'`
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: { })
Environment variables

Outputs:

out_file: (a pathlike object or string representing a file)
path/name of skullstripped file (if generated)
mask_file: (a pathlike object or string representing a file)
path/name of binary brain mask (if generated)
outline_file: (a pathlike object or string representing a file)
path/name of outline file (if generated)
meshfile: (a pathlike object or string representing a file)
path/name of vtk mesh file (if generated)
inskull_mask_file: (a pathlike object or string representing a file)
path/name of inskull mask (if generated)
inskull_mesh_file: (a pathlike object or string representing a file)
path/name of inskull mesh outline (if generated)
outskull_mask_file: (a pathlike object or string representing a file)
path/name of outskull mask (if generated)
outskull_mesh_file: (a pathlike object or string representing a file)
path/name of outskull mesh outline (if generated)
outskin_mask_file: (a pathlike object or string representing a file)
path/name of outskin mask (if generated)
outskin_mesh_file: (a pathlike object or string representing a file)
path/name of outskin mesh outline (if generated)
skull_mask_file: (a pathlike object or string representing a file)
path/name of skull mask (if generated)
63.8.4 FAST

Link to code
Wraps the executable command fast.
FSL FAST wrapper for segmentation and bias correction
For complete details, see the FAST Documentation.

Examples

```python
>>> from nipype.interfaces import fsl
>>> fastr = fsl.FAST()
>>> fastr.inputs.in_files = 'structural.nii'
>>> fastr.inputs.out_basename = 'fast_'
>>> fastr.cmdline
'fast -o fast_ -S 1 structural.nii'
>>> out = fastr.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_files: (a list of items which are a pathlike object or string representing an existing file)
  image, or multichannel set of images, to be segmented
  argument: ``-\%s``, position: -1

[Optional]
out_basename: (a pathlike object or string representing a file)
  base name of output files
  argument: ``-o %s``
number_classes: (1 <= a long integer <= 10)
  number of tissue-type classes
  argument: ``-n %d``
output_biasfield: (a boolean)
  output estimated bias field
  argument: ``-b``
output_biascorrected: (a boolean)
  output restored image (bias-corrected image)
  argument: ``-B``
img_type: (1 or 2 or 3)
  int specifying type of image: (1 = T1, 2 = T2, 3 = PD)
  argument: ``-t %d``
bias_iters: (1 <= a long integer <= 10)
  number of main-loop iterations during bias-field removal
  argument: ``-I %d``
bias_lowpass: (4 <= a long integer <= 40)
  bias field smoothing extent (FWHM) in mm
  argument: ``-l %d``
init_seg_smooth: (0.0001 <= a floating point number <= 0.1)
  initial segmentation spatial smoothness (during bias field estimation)
  argument: ``-f %.3f``
segments: (a boolean)
  outputs a separate binary image for each tissue type
(continues on next page)
argument: `'-g'`
init_transform: (a pathlike object or string representing an existing file)
   `<standard2input.mat>` initialise using priors
   argument: `'-a %s'`
other_priors: (a list of from 3 to 3 items which are a pathlike object or string representing a file)
   alternative prior images
   argument: `'-A %s'`
no_pve: (a boolean)
   turn off PVE (partial volume estimation)
   argument: `'--nopve'`
no_bias: (a boolean)
   do not remove bias field
   argument: `'-N'`
use_priors: (a boolean)
   use priors throughout
   argument: `'-P'`
segment_iters: (1 <= a long integer <= 50)
   number of segmentation-initialisation iterations
   argument: `'-W %d'`
mixel_smooth: (0.0 <= a floating point number <= 1.0)
   spatial smoothness for mixeltype
   argument: `'-R %.2f'`
iters_afterbias: (1 <= a long integer <= 20)
   number of main-loop iterations after bias-field removal
   argument: `'-O %d'`
hyper: (0.0 <= a floating point number <= 1.0)
   segmentation spatial smoothness
   argument: `'-H %.2f'`
verbose: (a boolean)
   switch on diagnostic messages
   argument: `'-v'`
manual_seg: (a pathlike object or string representing an existing file)
   Filename containing intensities
   argument: `'-s %s'`
probability_maps: (a boolean)
   outputs individual probability maps
   argument: `'-p'`
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
   FSL output type
args: (a unicode string)
   Additional parameters to the command
   argument: `'%%s'`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: { })
   Environment variables

Outputs:

tissue_class_map: (a pathlike object or string representing an existing file)
   path/name of binary segmented volume file one val for each class
   _seg

(continues on next page)
tissue_class_files: (a list of items which are a pathlike object or string representing a file)
restored_image: (a list of items which are a pathlike object or string representing a file)
mixeltype: (a pathlike object or string representing a file)
path/name of mixeltype volume file _mixeltype
partial_volume_map: (a pathlike object or string representing a file)
path/name of partial volume file_pveseg
partial_volume_files: (a list of items which are a pathlike object or string representing a file)
bias_field: (a list of items which are a pathlike object or string representing a file)
probability_maps: (a list of items which are a pathlike object or string representing a file)

References:
None

63.8.5 FIRST

Link to code
Wraps the executable command run_first_all.
FSL run_first_all wrapper for segmentation of subcortical volumes
http://www.fmrib.ox.ac.uk/fsl/first/index.html

Examples

>>> from nipype.interfaces import fsl
>>> first = fsl.FIRST()
>>> first.inputs.in_file = 'structural.nii'
>>> first.inputs.out_file = 'segmented.nii'
>>> res = first.run()  #doctest: +SKIP

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  input data file
  argument: ``-i %s``, position: -2
out_file: (a pathlike object or string representing a file, nipype
default value: segmented)
  output data file
  argument: ``-o %s``, position: -1

[Optional]
verbose: (a boolean)
  Use verbose logging.
  argument: ``-v``
  position: 1
brain_extracted: (a boolean)
  Input structural image is already brain-extracted
  argument: ``-b``
  position: 2
no_cleanup: (a boolean)
  Input structural image is already brain-extracted
  argument: ``-d``
  position: 3

(continues on next page)
method: ('auto' or 'fast' or 'none', nipype default value: auto)
   Method must be one of auto, fast, none, or it can be entered using
   the 'method_as_numerical_threshold' input
   argument: `'-m %s'`, position: 4
   mutually_exclusive: method_as_numerical_threshold

method_as_numerical_threshold: (a float)
   Specify a numerical threshold value or use the 'method' input to
   choose auto, fast, or none
   argument: `'-m %.4f'`, position: 4

list_of_specific_structures: (a list of at least 1 items which are a
   unicode string)
   Runs only on the specified structures (e.g. L_Hipp, R_HippL_Accu,
   R_Accu, L_Amyg, R_AmygL_Caud, R_Caud, L_Pall, R_PallL_Puta, R_Puta,
   L_Thal, R_Thal, BrStem
   argument: `'-s %s'`, position: 5

affine_file: (a pathlike object or string representing an existing
   file)
   Affine matrix to use (e.g. img2std.mat) (does not re-run
   registration)
   argument: `'-a %s'`, position: 6

output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
   'NIFTI_PAIR_GZ')
   FSL output type

args: (a unicode string)
   Additional parameters to the command
   argument: `'%s'`

environ: (a dictionary with keys which are a bytes or None or a value
   of class 'str' and with values which are a bytes or None or a
   value of class 'str', nipype default value: {})}
   Environment variables

Outputs:

vtk_surfaces: (a list of items which are a pathlike object or string
   representing an existing file)
   VTK format meshes for each subcortical region

bvars: (a list of items which are a pathlike object or string
   representing an existing file)
   bvars for each subcortical region

original_segmentations: (a pathlike object or string representing an
   existing file)
   3D image file containing the segmented regions as integer values.
   Uses CMA labelling

segmentation_file: (a pathlike object or string representing an
   existing file)
   4D image file containing a single volume per segmented region

References:

None

63.8.6 FLIRT

Link to code
Wraps the executable command flirt.
FSL FLIRT wrapper for coregistration
For complete details, see the FLIRT Documentation.
To print out the command line help, use: fsl.FLIRT().inputs_help()

Examples

```python
>>> from nipype.interfaces import fsl
>>> from nipype.testing import example_data

>>> flt = fsl.FLIRT(bins=640, cost_func='mutualinfo')
>>> flt.inputs.in_file = 'structural.nii'
>>> flt.inputs.reference = 'mni.nii'
>>> flt.inputs.output_type = "NIFTI_GZ"
>>> flt.cmdline
# doctest: +ELLIPSIS
'flirt -in structural.nii -ref mni.nii -out structural_flirt.nii.gz -omat structural_flirt.mat -bins 640 -searchcost mutualinfo'

>>> res = flt.run() #doctest: +SKIP
```

Inputs:

- **in_file**: (a pathlike object or string representing an existing file)
  - input file
  - argument: `'\-in %s'`, position: 0

- **reference**: (a pathlike object or string representing an existing file)
  - reference file
  - argument: `'\-ref %s'`, position: 1

- **out_file**: (a pathlike object or string representing a file)
  - registered output file
  - argument: `'\-out %s'`, position: 2

- **out_matrix_file**: (a pathlike object or string representing a file)
  - output affine matrix in 4x4 ascii format
  - argument: `'\-omat %s'`, position: 3

- **out_log**: (a pathlike object or string representing a file)
  - output log
  - requires: save_log

- **in_matrix_file**: (a pathlike object or string representing a file)
  - input 4x4 affine matrix
  - argument: `'\-init %s'`

- **apply_xfm**: (a boolean)
  - apply transformation supplied by in_matrix_file or uses_qform to use the affine matrix stored in the reference header
  - argument: `'\-applyxfm'`

- **apply_isoxfm**: (a float)
  - as applyxfm but forces isotropic resampling
  - argument: `'\-applyisoxfm %f'`
  - mutually_exclusive: apply_xfm

- **datatype**: ('char' or 'short' or 'int' or 'float' or 'double')
  - force output data type
  - argument: `'\-datatype %s'`

- **cost**: ('mutualinfo' or 'corratio' or 'normcorr' or 'normmi' or 'leastsq' or 'labeldiff' or 'bbr')
  - cost function
  - argument: `'\-cost %s'`

- **cost_func**: ('mutualinfo' or 'corratio' or 'normcorr' or 'normmi' or 'leastsq' or 'labeldiff' or 'bbr')
  - cost function
  - argument: `'\-searchcost %s'`

(continues on next page)
uses_qform: (a boolean)
    initialize using sform or qform
    argument: `--usesqform`

display_init: (a boolean)
    display initial matrix
    argument: `--displayinit`

angle_rep: (‘quaternion’ or ‘euler’)
    representation of rotation angles
    argument: `--anglerep %s`

interp: (‘trilinear’ or ‘nearestneighbour’ or ‘sinc’ or ‘spline’)
    final interpolation method used in reslicing
    argument: `--interp %s`

sinc_width: (an integer (int or long))
    full-width in voxels
    argument: `--sincwidth %d`

sinc_window: (‘rectangular’ or ‘hanning’ or ‘blackman’)
    sinc window
    argument: `--sincwindow %s`

bins: (an integer (int or long))
    number of histogram bins
    argument: `--bins %d`

dof: (an integer (int or long))
    number of transform degrees of freedom
    argument: `--dof %d`

no_resample: (a boolean)
    do not change input sampling
    argument: `--noresample`

force_scaling: (a boolean)
    force rescaling even for low-res images
    argument: `--forcescaling`

min_sampling: (a float)
    set minimum voxel dimension for sampling
    argument: `--minsampling %f`

padding_size: (an integer (int or long))
    for applyxfm: interpolates outside image by size
    argument: `--paddingsize %d`

searchr_x: (a list of from 2 to 2 items which are an integer (int or long))
    search angles along x-axis, in degrees
    argument: `--searchrx %s`

searchr_y: (a list of from 2 to 2 items which are an integer (int or long))
    search angles along y-axis, in degrees
    argument: `--searchry %s`

searchr_z: (a list of from 2 to 2 items which are an integer (int or long))
    search angles along z-axis, in degrees
    argument: `--searchrz %s`

no_search: (a boolean)
    set all angular searches to ranges 0 to 0
    argument: `--nosearch`

coarse_search: (an integer (int or long))
    coarse search delta angle
    argument: `--coarsesearch %d`

fine_search: (an integer (int or long))
    fine search delta angle
    argument: `--finesearch %d`
schedule: (a pathlike object or string representing an existing file)
   replaces default schedule
   argument: `--schedule %s`
ref_weight: (a pathlike object or string representing an existing file)
   File for reference weighting volume
   argument: `--refweight %s`
in_weight: (a pathlike object or string representing an existing file)
   File for input weighting volume
   argument: `--inweight %s`
no_clamp: (a boolean)
   do not use intensity clamping
   argument: `--noclamp`
no_resample_blur: (a boolean)
   do not use blurring on downsampling
   argument: `--noresampblur`
rigid2D: (a boolean)
   use 2D rigid body mode - ignores dof
   argument: `--2D`
save_log: (a boolean)
   save to log file
verbose: (an integer (int or long))
   verbose mode, 0 is least
   argument: `--verbose %d`
bvvalue: (a float)
   use specified background value for points outside FOV
   argument: `--setbackground %f`
wm_seg: (a pathlike object or string representing a file)
   white matter segmentation volume needed by BBR cost function
   argument: `--wmseg %s`
wmcords: (a pathlike object or string representing a file)
   white matter boundary coordinates for BBR cost function
   argument: `--wmcoords %s`
wmnorms: (a pathlike object or string representing a file)
   white matter boundary normals for BBR cost function
   argument: `--wnnorms %s`
fieldmap: (a pathlike object or string representing a file)
   fieldmap image in rads/s - must be already registered to the
   reference image
   argument: `--fieldmap %s`
fieldmapmask: (a pathlike object or string representing a file)
   mask for fieldmap image
   argument: `--fieldmapmask %s`
pedir: (an integer (int or long))
   phase encode direction of EPI - 1/2/3=x/y/z & -1/-2/-3=-x/-y/-z
   argument: `--pedir %d`
echospacing: (a float)
   value of EPI echo spacing - units of seconds
   argument: `--echospacing %f`
bbrtype: ("signed" or "global_abs" or "local_abs")
   type of bbr cost function: signed [default], global_abs, local_abs
   argument: `--bbrtype %s`
bbrslope: (a float)
   value of bbr slope
   argument: `--bbrslope %f`
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 
(continues on next page)
'NIFTI_PAIR_GZ')
FSL output type
args: (a unicode string)
   Additional parameters to the command
   argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
   of class 'str' and with values which are a bytes or None or a
   value of class 'str', nipype default value: {})
Environment variables

Outputs:
out_file: (a pathlike object or string representing an existing file)
   path/name of registered file (if generated)
out_matrix_file: (a pathlike object or string representing an existing file)
   path/name of calculated affine transform (if generated)
out_log: (a pathlike object or string representing a file)
   path/name of output log (if generated)

References:
None

63.8.7 FNIRT

Link to code
Wraps the executable command fnirt.
FSL FNIRT wrapper for non-linear registration
For complete details, see the FNIRT Documentation.

Examples

```python
>>> from nipype.interfaces import fsl
>>> from nipype.testing import example_data

>>> fnt = fsl.FNIRT(affine_file=example_data('trans.mat'))

res = fnt.run(ref_file=example_data('mni.nii'), in_file=example_data('structural.nii'))  # doctest: +SKIP

T1 -> Mni153

>>> from nipype.interfaces import fsl

>>> fnirt_mprage = fsl.FNIRT()

>>> fnirt_mprage.inputs.in_fwhm = [8, 4, 2, 2]

>>> fnirt_mprage.inputs.subsampling_scheme = [4, 2, 1, 1]

Specify the resolution of the warps

>>> fnirt_mprage.inputs.warp_resolution = (6, 6, 6)

>>> res = fnirt_mprage.run(in_file='structural.nii', ref_file='mni.nii', warped_file='warped.nii', fieldcoeff_file='fieldcoeff.nii')  # doctest: +SKIP

We can check the command line and confirm that it’s what we expect.

>>> fnirt_mprage.cmdline  # doctest: +SKIP
'fnirt --cout=fieldcoeff.nii --in=structural.nii --infwhm=8,4,2,2 --ref=mni.nii --subsamp=4,2,1,1 --warpres=6,6,6 --iout=warped.nii'
```
Inputs:

[Mandatory]
ref_file: (a pathlike object or string representing an existing file)
  name of reference image
  argument: `--ref=%s`
in_file: (a pathlike object or string representing an existing file)
  name of input image
  argument: `--in=%s`

[Optional]
affine_file: (a pathlike object or string representing an existing file)
  name of file containing affine transform
  argument: `--aff=%s`
inwarp_file: (a pathlike object or string representing an existing file)
  name of file containing initial non-linear warps
  argument: `--inwarp=%s`
in_intensitymap_file: (a list of from 1 to 2 items which are a pathlike object or string representing an existing file)
  name of file/files containing initial intensity mapping usually generated by previous fnirt run
  argument: `--intin=%s`
fieldcoeff_file: (a boolean or a pathlike object or string representing a file)
  name of output file with field coefficients or true
  argument: `--cout=%s`
warped_file: (a pathlike object or string representing a file)
  name of output image
  argument: `--iout=%s`
field_file: (a boolean or a pathlike object or string representing a file)
  name of output file with field or true
  argument: `--fout=%s`
jacobian_file: (a boolean or a pathlike object or string representing a file)
  name of file for writing out the Jacobian of the field (for diagnostic or VBM purposes)
  argument: `--jout=%s`
modulatedref_file: (a boolean or a pathlike object or string representing a file)
  name of file for writing out intensity modulated --ref (for diagnostic purposes)
  argument: `--refout=%s`
out_intensitymap_file: (a boolean or a pathlike object or string representing a file)
  name of files for writing information pertaining to intensity mapping
  argument: `--intout=%s`
log_file: (a pathlike object or string representing a file)
  Name of log-file
  argument: `--logout=%s`
config_file: ('T1_2_MNI152_2mm' or 'FA_2_FMRIB58_1mm' or a pathlike object or string representing an existing file)
  Name of config file specifying command line arguments
  argument: `--config=%s`
refmask_file: (a pathlike object or string representing an existing file)
nipype Documentation, Release 1.2.1

(continued from previous page)

file)
name of file with mask in reference space
argument: `'--refmask=%s'`

inmask_file: (a pathlike object or string representing an existing file)
name of file with mask in input image space
argument: `'--inmask=%s'`

skip_refmask: (a boolean)
Skip specified refmask if set, default false
argument: `'--applyrefmask=0'`
mutually_exclusive: apply_refmask

skip_inmask: (a boolean)
skip specified inmask if set, default false
argument: `'--applyinmask=0'`
mutually_exclusive: apply_inmask

apply_refmask: (a list of items which are 0 or 1)
list of iterations to use reference mask on (1 to use, 0 to skip)
argument: `'--applyrefmask=%s'`
mutually_exclusive: skip_refmask

apply_inmask: (a list of items which are 0 or 1)
list of iterations to use input mask on (1 to use, 0 to skip)
argument: `'--applyinmask=%s'`
mutually_exclusive: skip_inmask

skip_implicit_ref_masking: (a boolean)
skip implicit masking based on value in --ref image. Default = 0
argument: `'--imprefm=0'`

skip_implicit_in_masking: (a boolean)
skip implicit masking based on value in --in image. Default = 0
argument: `'--impinm=0'`

refmask_val: (a float)
Value to mask out in --ref image. Default =0.0
argument: `'--imprefval=%f'`

inmask_val: (a float)
Value to mask out in --in image. Default =0.0
argument: `'--impinval=%f'`

max_nonlin_iter: (a list of items which are an integer (int or long))
Max # of non-linear iterations list, default [5, 5, 5, 5]
argument: `'--miter=%s'`

subsampling_scheme: (a list of items which are an integer (int or long))
sub-sampling scheme, list, default [4, 2, 1, 1]
argument: `'--subsamp=%s'`

warp_resolution: (a tuple of the form: (an integer (int or long), an integer (int or long)))
(approximate) resolution (in mm) of warp basis in x-, y- and z-direction, default 10, 10, 10
argument: `'--warpres=%d,%d,%d'`

spline_order: (an integer (int or long))
Order of spline, 2->Quadratic spline, 3->Cubic spline. Default=3
argument: `'--splineorder=%d'`

in_fwhm: (a list of items which are an integer (int or long))
FWHM (in mm) of gaussian smoothing kernel for input volume, default [6, 4, 2, 2]
argument: `'--infwhm=%s'`

ref_fwhm: (a list of items which are an integer (int or long))
FWHM (in mm) of gaussian smoothing kernel for ref volume, default [4, 2, 0, 0]

(continues on next page)
regularization_model: ('membrane_energy' or 'bending_energy')
   Model for regularisation of warp-field [membrane_energy
   bending_energy], default bending_energy
argument: '--regmod=%s'
regularization_lambda: (a list of items which are a float)
   Weight of regularisation, default depending on --ssqlambda and
   --regmod switches. See user documentation.
argument: '--lambda=%s'
skip_lambda_ssq: (a boolean)
   If true, lambda is not weighted by current ssq, default false
argument: '--ssqlambda=0'
jacobian_range: (a tuple of the form: (a float, a float))
   Allowed range of Jacobian determinants, default 0.01, 100.0
argument: '--jacrange=%f,%f'
derive_from_ref: (a boolean)
   If true, ref image is used to calculate derivatives. Default false
argument: '--refderiv'
intensity_mapping_model: ('none' or 'global_linear' or
   'global_non_linear' or 'local_linear' or
   'global_non_linear_with_bias' or 'local_non_linear')
   Model for intensity-mapping
argument: '--intmod=%s'
intensity_mapping_order: (an integer (int or long))
   Order of poynomial for mapping intensities, default 5
argument: '--intorder=%d'
biasfield_resolution: (a tuple of the form: (an integer (int or
   long), an integer (int or long), an integer (int or long)))
   Resolution (in mm) of bias-field modelling local intensities,
   default 50, 50, 50
argument: '--biasres=%d,%d,%d'
bias_regularization_lambda: (a float)
   Weight of regularisation for bias-field, default 10000
argument: '--biaslambda=%f'
skip_intensity_mapping: (a boolean)
   Skip estimate intensity-mapping default false
argument: '--estint=0'
mutually_exclusive: apply_intensity_mapping
apply_intensity_mapping: (a list of items which are 0 or 1)
   List of subsampling levels to apply intensity mapping for (0 to
   skip, 1 to apply)
argument: '--estint=%s'
mutually_exclusive: skip_intensity_mapping
hessian_precision: ('double' or 'float')
   Precision for representing Hessian, double or float. Default double
argument: '--numprec=%s'
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
   'NIFTI_PAIR_GZ')
   FSL output type
args: (a unicode string)
   Additional parameters to the command
argument: '%s'
environ: (a dictionary with keys which are a bytes or None or a value
   of class 'str' and with values which are a bytes or None or a
   value of class 'str', nipype default value: {})
Environment variables

Outputs:
fieldcoeff_file: (a pathlike object or string representing an existing file) file with field coefficients
warped_file: (a pathlike object or string representing an existing file) warped image
field_file: (a pathlike object or string representing a file) file with warp field
jacobian_file: (a pathlike object or string representing a file) file containing Jacobian of the field
modulatedref_file: (a pathlike object or string representing a file) file containing intensity modulated --ref
out_intensitymap_file: (a list of from 2 to 2 items which are a pathlike object or string representing a file) files containing info pertaining to intensity mapping
log_file: (a pathlike object or string representing a file) Name of log-file

References:

None

63.8.8 FUGUE

Link to code

Wraps the executable command fugue.

FSL FUGUE set of tools for EPI distortion correction

FUGUE is, most generally, a set of tools for EPI distortion correction.

Distortions may be corrected for

1. improving registration with non-distorted images (e.g. structurals), or
2. dealing with motion-dependent changes.

FUGUE is designed to deal only with the first case - improving registration.

Examples

Unwarping an input image (shift map is known):

```python
>>> from nipype.interfaces.fsl.preprocess import FUGUE
>>> fugue = FUGUE()
>>> fugue.inputs.in_file = 'epi.nii'
>>> fugue.inputs.mask_file = 'epi_mask.nii'
>>> fugue.inputs.shift_in_file = 'vsm.nii'  # Previously computed with fugue as well
>>> fugue.inputs.unwarp_direction = 'y'
>>> fugue.inputs.output_type = "NIFTI_GZ"
>>> fugue.cmdline  # doctest: +ELLIPSIS
'fugue --in=epi.nii --mask=epi_mask.nii --loadshift=vsm.nii --unwarpdir=y --unwarp=epi_unwarped.nii.gz'
>>> fugue.run()  # doctest: +SKIP
```

Warping an input image (shift map is known):

```python
>>> from nipype.interfaces.fsl.preprocess import FUGUE
>>> fugue = FUGUE()
>>> fugue.inputs.in_file = 'epi.nii'
>>> fugue.inputs.forward_warping = True
>>> fugue.inputs.mask_file = 'epi_mask.nii'
```

(continues on next page)
Computing the vsm (unwrapped phase map is known):

```python
>>> from nipype.interfaces.fsl.preprocess import FUGUE
>>> fugue = FUGUE()
>>> fugue.inputs.phasemap_in_file = 'epi_phasediff.nii'
>>> fugue.inputs.mask_file = 'epi_mask.nii'
>>> fugue.inputs.dwell_to_asym_ratio = (0.77e-3 * 3) / 2.46e-3
>>> fugue.inputs.unwarp_direction = 'y'
>>> fugue.inputs.save_shift = True
>>> fugue.inputs.output_type = "NIFTI_GZ"
>>> fugue.cmdline # doctest: +ELLIPSIS
'fugue --dwelltoasym=0.9390243902 --mask=epi_mask.nii --phasemap=epi_phasediff.vsm.nii --saveshift=epi_phasediff_vsm.nii.gz --unwarpdir=y'
>>> fugue.run() #doctest: +SKIP
```

Inputs:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_file</td>
<td>(a pathlike object or string representing an existing file)</td>
<td>filename of input volume argument: <code>&quot;--in=%s&quot;</code></td>
</tr>
<tr>
<td>shift_in_file</td>
<td>(a pathlike object or string representing an existing file)</td>
<td>filename for reading pixel shift volume argument: <code>&quot;--loadshift=%s&quot;</code></td>
</tr>
<tr>
<td>phasemap_in_file</td>
<td>(a pathlike object or string representing an existing file)</td>
<td>filename for input phase image argument: <code>&quot;--phasemap=%s&quot;</code></td>
</tr>
<tr>
<td>fmap_in_file</td>
<td>(a pathlike object or string representing an existing file)</td>
<td>filename for loading fieldmap (rad/s) argument: <code>&quot;--loadfmap=%s&quot;</code></td>
</tr>
<tr>
<td>unwarped_file</td>
<td>(a pathlike object or string representing a file)</td>
<td>apply unwarping and save as filename argument: <code>&quot;--unwarp=%s&quot;</code> mutually_exclusive: warped_file requires: in_file</td>
</tr>
<tr>
<td>warped_file</td>
<td>(a pathlike object or string representing a file)</td>
<td>apply forward warping and save as filename argument: <code>&quot;--warp=%s&quot;</code> mutually_exclusive: unwarped_file requires: in_file</td>
</tr>
<tr>
<td>forward_warping</td>
<td>(a boolean, nipype default value: False)</td>
<td>apply forward warping instead of unwarping</td>
</tr>
<tr>
<td>dwell_to_asym_ratio</td>
<td>(a float)</td>
<td>set the dwell to asym time ratio argument: <code>&quot;--dwelltoasym=%.10f&quot;</code></td>
</tr>
<tr>
<td>dwell_time</td>
<td>(a float)</td>
<td></td>
</tr>
</tbody>
</table>
set the EPI dwell time per phase-encode line - same as echo spacing
- (sec)
  argument: `''--dwell=%.10f''`

asym_se_time: (a float)
  set the fieldmap asymmetric spin echo time (sec)
  argument: `''--asym=%.10f''`

median_2dfilter: (a boolean)
  apply 2D median filtering
  argument: `''--median''`

despike_2dfilter: (a boolean)
  apply a 2D de-spiking filter
  argument: `''--despike''`

no_gap_fill: (a boolean)
  do not apply gap-filling measure to the fieldmap
  argument: `''--nofill''`

no_extend: (a boolean)
  do not apply rigid-body extrapolation to the fieldmap
  argument: `''--noextend''`

smooth2d: (a float)
  apply 2D Gaussian smoothing of sigma N (in mm)
  argument: `''--smooth2=%.2f''`

smooth3d: (a float)
  apply 3D Gaussian smoothing of sigma N (in mm)
  argument: `''--smooth3=%.2f''`

poly_order: (an integer (int or long))
  apply polynomial fitting of order N
  argument: `''--poly=%d''`

despike_threshold: (a float)
  specify the threshold for de-spiking (default=3.0)
  argument: `''--despikethreshold=%f''`

unwarp_direction: ('x' or 'y' or 'z' or 'x-' or 'y-' or 'z-')
  specifies direction of warping (default y)
  argument: `''--unwarpdir=%s''`

phase_conjugate: (a boolean)
  apply phase conjugate method of unwarping
  argument: `''--phaseconj''`

icorr: (a boolean)
  apply intensity correction to unwarping (pixel shift method only)
  argument: `''--icorr''`
  requires: shift_in_file

icorr_only: (a boolean)
  apply intensity correction only
  argument: `''--icorronly''`
  requires: unwarped_file

mask_file: (a pathlike object or string representing an existing file)
  filename for loading valid mask
  argument: `''--mask=%s''`

nokspace: (a boolean)
  do not use k-space forward warping
  argument: `''--nokspace''`

(continues on previous page)
save_shift: (a boolean)
    write pixel shift volume
    mutually_exclusive: save_unmasked_shift
shift_out_file: (a pathlike object or string representing a file)
    filename for saving pixel shift volume
    argument: `--saveshift=%s`
save_unmasked_shift: (a boolean)
    saves the unmasked shiftmap when using --saveshift
    argument: `--unmaskshift`
    mutually_exclusive: save_shift
save_fmap: (a boolean)
    write field map volume
    mutually_exclusive: save_unmasked_fmap
fmap_out_file: (a pathlike object or string representing a file)
    filename for saving fieldmap (rad/s)
    argument: `--savefmap=%s`
save_unmasked_fmap: (a boolean)
    saves the unmasked fieldmap when using --savefmap
    argument: `--unmaskfmap`
    mutually_exclusive: save_fmap
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
    FSL output type
args: (a unicode string)
    Additional parameters to the command
    argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
    Environment variables

Outputs:

unwarped_file: (a pathlike object or string representing a file)
    unwarped file
warped_file: (a pathlike object or string representing a file)
    forward warped file
shift_out_file: (a pathlike object or string representing a file)
    voxel shift map file
fmap_out_file: (a pathlike object or string representing a file)
    fieldmap file

References:
None

63.8.9 MCFLIRT

Link to code
Wraps the executable command mcflirt.
FSL MCFLIRT wrapper for within-modality motion correction
For complete details, see the MCFLIRT Documentation.
Examples

```python
>>> from nipype.interfaces import fsl
>>> mcflt = fsl.MCFLIRT()
>>> mcflt.inputs.in_file = 'functional.nii'
>>> mcflt.inputs.cost = 'mutualinfo'
>>> mcflt.inputs.out_file = 'moco.nii'
>>> mcflt.cmdline
'mcflirt -in functional.nii -cost mutualinfo -out moco.nii'
>>> res = mcflt.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]

in_file: (a pathlike object or string representing an existing file)
  timeseries to motion-correct
  argument: ``-in %s``, position: 0

[Optional]

out_file: (a pathlike object or string representing a file)
  file to write
  argument: ``-out %s``

cost: ('mutualinfo' or 'woods' or 'corratio' or 'normcorr' or 'normmi' or 'leastsquares')
  cost function to optimize
  argument: ``-cost %s``

bins: (an integer (int or long))
  number of histogram bins
  argument: ``-bins %d``

dof: (an integer (int or long))
  degrees of freedom for the transformation
  argument: ``-dof %d``

ref_vol: (an integer (int or long))
  volume to align frames to
  argument: ``-refvol %d``

scaling: (a float)
  scaling factor to use
  argument: ``-scaling %.2f``

smooth: (a float)
  smoothing factor for the cost function
  argument: ``-smooth %.2f``

rotation: (an integer (int or long))
  scaling factor for rotation tolerances
  argument: ``-rotation %d``

stages: (an integer (int or long))
  stages (if 4, perform final search with sinc interpolation
  argument: ``-stages %d``

init: (a pathlike object or string representing an existing file)
  initial transformation matrix
  argument: ``-init %s``

interpolation: ('spline' or 'nn' or 'sinc')
  interpolation method for transformation
  argument: ``-%s_final``

use_gradient: (a boolean)
  run search on gradient images
  argument: ``-gdt``

use_contour: (a boolean)
  run search on contour images
```

(continues on next page)
argument: `'-edge'`

mean_vol: (a boolean)
  register to mean volume
  argument: `'-meanvol'`

stats_imgs: (a boolean)
  produce variance and std. dev. images
  argument: `'-stats'`

save_mats: (a boolean)
  save transformation matrices
  argument: `'-mats'`

save_plots: (a boolean)
  save transformation parameters
  argument: `'-plots'`

save_rms: (a boolean)
  save rms displacement parameters
  argument: `'-rmsabs -rmsrel'`

ref_file: (a pathlike object or string representing an existing file)
  target image for motion correction
  argument: `'-reffile %s'`

output_type: (`'NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
  'NIFTI_PAIR_GZ')
  FSL output type

args: (a unicode string)
  Additional parameters to the command
  argument: `'%s'`

environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})

Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
  motion-corrected timeseries

variance_img: (a pathlike object or string representing an existing file)
  variance image

std_img: (a pathlike object or string representing an existing file)
  standard deviation image

mean_img: (a pathlike object or string representing an existing file)
  mean timeseries image (if mean_vol=True)

par_file: (a pathlike object or string representing an existing file)
  text-file with motion parameters

mat_file: (a list of items which are a pathlike object or string
  representing an existing file)
  transformation matrices

rms_files: (a list of items which are a pathlike object or string
  representing an existing file)
  absolute and relative displacement parameters

References:

None

63.8.10 PRELUDE

Link to code
Wraps the executable command prelude.
FSL prelude wrapper for phase unwrapping

Examples

Please insert examples for use of this command

Inputs:

[Mandatory]
complex_phase_file: (a pathlike object or string representing an existing file)
   complex phase input volume
   argument: `--complex=%s`
   mutually_exclusive: magnitude_file, phase_file
magnitude_file: (a pathlike object or string representing an existing file)
   file containing magnitude image
   argument: `--abs=%s`
   mutually_exclusive: complex_phase_file
phase_file: (a pathlike object or string representing an existing file)
   raw phase file
   argument: `--phase=%s`
   mutually_exclusive: complex_phase_file

[Optional]
unwrapped_phase_file: (a pathlike object or string representing a file)
   file containing unwrapped phase
   argument: `--unwrap=%s`
num_partitions: (an integer (int or long))
   number of phase partitions to use
   argument: `--numphasesplit=%d`
labelprocess2d: (a boolean)
   does label processing in 2D (slice at a time)
   argument: `--labelslices`
process2d: (a boolean)
   does all processing in 2D (slice at a time)
   argument: `--slices`
   mutually_exclusive: labelprocess2d
process3d: (a boolean)
   forces all processing to be full 3D
   argument: `--force3D`
   mutually_exclusive: labelprocess2d, process2d
threshold: (a float)
   intensity threshold for masking
   argument: `--thresh=%.10f`
mask_file: (a pathlike object or string representing an existing file)
   filename of mask input volume
   argument: `--mask=%s`
start: (an integer (int or long))
   first image number to process (default 0)
   argument: `--start=%d`
end: (an integer (int or long))
   final image number to process (default Inf)
   argument: `--end=%d`
savemask_file: (a pathlike object or string representing a file)
(continues on next page)
saving the mask volume
argument: ``--savemask=%s``

rawphase_file: (a pathlike object or string representing a file)
saving the raw phase output
argument: ``--rawphase=%s``

label_file: (a pathlike object or string representing a file)
saving the area labels output
argument: ``--labels=%s``

removeramps: (a boolean)
remove phase ramps during unwrapping
argument: ``--removeramps``

output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
FSL output type

args: (a unicode string)
Additional parameters to the command
argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

unwrapped_phase_file: (a pathlike object or string representing an
existing file)
unwrapped phase file

References:
None

63.8.11 SUSAN

Link to code
Wraps the executable command susan.
FSL SUSAN wrapper to perform smoothing
For complete details, see the SUSAN Documentation.

Examples

```python
>>> from nipype.interfaces import fsl
>>> from nipype.testing import example_data
>>> anatfile # doctest: +SKIP
anatomical.nii # doctest: +SKIP
>>> sus = fsl.SUSAN()
>>> sus.inputs.in_file = example_data('structural.nii')
>>> sus.inputs.brightness_threshold = 2000.0
>>> sus.inputs.fwhm = 8.0
>>> result = sus.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
filename of input timeseries
argument: ``%s``
brightness_threshold: (a float)
  brightness threshold and should be greater than noise level and less
  than contrast of edges to be preserved.
  argument: ``%.10f``
fwhm: (a float)
  fwhm of smoothing, in mm, gets converted using sqrt(8*log(2))
  argument: ``%.10f``

[Optional]
dimension: (3 or 2, nipype default value: 3)
  within-plane (2) or fully 3D (3)
  argument: ``%d``
use_median: (1 or 0, nipype default value: 1)
  whether to use a local median filter in the cases where single-point
  noise is detected
  argument: ``%d``
usans: (a list of at most 2 items which are a tuple of the form: (a
  pathlike object or string representing an existing file, a float),
  nipype default value: []
  determines whether the smoothing area (USAN) is to be found from
  secondary images (0, 1 or 2). A negative value for any brightness
  threshold will auto-set the threshold at 10% of the robust range
out_file: (a pathlike object or string representing a file)
  output file name
  argument: ``%s``
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
  'NIFTI_PAIR_GZ')
  FSL output type
args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
  Environment variables

Outputs:

smoothed_file: (a pathlike object or string representing an existing
  file)
  smoothed output file

References:

None

63.8.12 SliceTimer

Link to code
Wraps the executable command slicetimer.
FSL slicetimer wrapper to perform slice timing correction
Examples

```python
>>> from nipype.interfaces import fsl
>>> from nipype.testing import example_data

>>> st = fsl.SliceTimer()
>>> st.inputs.in_file = example_data('functional.nii')
>>> st.inputs.interleaved = True
>>> result = st.run() #doctest: +SKIP
```

Inputs:

- **in_file**: (a pathlike object or string representing an existing file)
  - Filename of input timeseries
  - Argument: `''--in=%s''`, position: 0

- **out_file**: (a pathlike object or string representing a file)
  - Filename of output timeseries
  - Argument: `''--out=%s''`

- **index_dir**: (a boolean)
  - Slice indexing from top to bottom
  - Argument: `''--down''`

- **time_repetition**: (a float)
  - Specify TR of data - default is 3s
  - Argument: `''--repeat=%f''`

- **slice_direction**: (1 or 2 or 3)
  - Direction of slice acquisition (x=1, y=2, z=3) - default is z
  - Argument: `''--direction=%d''`

- **interleaved**: (a boolean)
  - Use interleaved acquisition
  - Argument: `''--odd''`

- **custom_timings**: (a pathlike object or string representing an existing file)
  - Slice timings, in fractions of TR, range 0:1 (default is 0.5 = no shift)
  - Argument: `''--tcustom=%s''`

- **global_shift**: (a float)
  - Shift in fraction of TR, range 0:1 (default is 0.5 = no shift)
  - Argument: `''--tglobal''`

- **custom_order**: (a pathlike object or string representing an existing file)
  - Filename of single-column custom interleave order file (first slice is referred to as 1 not 0)
  - Argument: `''--ocustom=%s''`

- **output_type**: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
  - FSL output type

- **args**: (a unicode string)
  - Additional parameters to the command
  - Argument: `''%s''`

- **environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: `{}`)
  - Environment variables

Outputs:
slice_time_corrected_file: (a pathlike object or string representing an existing file)
slice time corrected file

References:
None

63.9 interfaces.fsl.utils

63.9.1 AvScale

Link to code
Wraps the executable command avscale. Use FSL avscale command to extract info from mat file output of FLIRT

Examples

```python
>>> avscale = AvScale()
>>> avscale.inputs.mat_file = 'flirt.mat'
>>> res = avscale.run()  # doctest: +SKIP
```

Inputs:

[Optional]
all_param: (a boolean)
  argument: ``--allparams``
mat_file: (a pathlike object or string representing an existing file)
  mat file to read
  argument: ``%s``, position: -2
ref_file: (a pathlike object or string representing an existing file)
  reference file to get center of rotation
  argument: ``%s``, position: -1
args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  Environment variables

Outputs:

rotation_translation_matrix: (a list of items which are a list of items which are a float)
  Rotation and Translation Matrix
scales: (a list of items which are a float)
  Scales (x,y,z)
skews: (a list of items which are a float)
  Skews
average_scaling: (a float)
  Average Scaling
determinant: (a float)
  Determinant
forward_half_transform: (a list of items which are a list of items which are a float)
Forward Half Transform
backward_half_transform: (a list of items which are a list of items which are a float)

Backwards Half Transform
left_right_orientation_preserved: (a boolean)
   True if LR orientation preserved
rot_angles: (a list of items which are a float)
rotation angles
translations: (a list of items which are a float)
translations

63.9.2 Complex

Link to code
Wraps the executable command fslcomplex. fslcomplex is a tool for converting complex data

Examples

```python
>>> cplx = Complex()
>>> cplx.inputs.complex_in_file = "complex.nii"
>>> cplx.real_polar = True
>>> res = cplx.run() # doctest: +SKIP
```

Inputs:

[Optional]
complex_in_file: (a pathlike object or string representing an existing file)
   argument: ``%s``, position: 2
complex_in_file2: (a pathlike object or string representing an existing file)
   argument: ``%s``, position: 3
real_in_file: (a pathlike object or string representing an existing file)
   argument: ``%s``, position: 2
imaginary_in_file: (a pathlike object or string representing an existing file)
   argument: ``%s``, position: 3
magnitude_in_file: (a pathlike object or string representing an existing file)
   argument: ``%s``, position: 2
phase_in_file: (a pathlike object or string representing an existing file)
   argument: ``%s``, position: 3
complex_out_file: (a pathlike object or string representing a file)
   argument: ``%s``, position: -3
mutually_exclusive: complex_out_file, magnitude_out_file,
   phase_out_file, real_out_file, imaginary_out_file, real_polar,
   real_cartesian
magnitude_out_file: (a pathlike object or string representing a file)
   argument: ``%s``, position: -4
mutually_exclusive: complex_out_file, real_out_file,
   imaginary_out_file, real_cartesian, complex_cartesian,
   complex_polar, complex_split, complex_merge

(continues on next page)
phase_out_file: (a pathlike object or string representing a file)
  argument: ``%s``, position: -3
  mutually_exclusive: complex_out_file, real_out_file,
                  imaginary_out_file, real_cartesian, complex_cartesian,
                  complex_polar, complex_split, complex_merge
real_out_file: (a pathlike object or string representing a file)
  argument: ``%s``, position: -4
  mutually_exclusive: complex_out_file, magnitude_out_file,
                  phase_out_file, real_polar, complex_cartesian, complex_polar,
                  complex_split, complex_merge
imaginary_out_file: (a pathlike object or string representing a file)
  argument: ``%s``, position: -3
  mutually_exclusive: complex_out_file, magnitude_out_file,
                  phase_out_file, real_polar, complex_cartesian, complex_polar,
                  complex_split, complex_merge
start_vol: (an integer (int or long))
  argument: ``%d``, position: -2
end_vol: (an integer (int or long))
  argument: ``%d``, position: -1
real_polar: (a boolean)
  argument: ``-realpolar``, position: 1
  mutually_exclusive: real_polar, real_cartesian, complex_cartesian,
                  complex_polar, complex_split, complex_merge
real_cartesian: (a boolean)
  argument: ``-realcartesian``, position: 1
  mutually_exclusive: real_polar, real_cartesian, complex_cartesian,
                  complex_polar, complex_split, complex_merge
complex_cartesian: (a boolean)
  argument: ``-complex``, position: 1
  mutually_exclusive: real_polar, real_cartesian, complex_cartesian,
                  complex_polar, complex_split, complex_merge
complex_polar: (a boolean)
  argument: ``-complexpolar``, position: 1
  mutually_exclusive: real_polar, real_cartesian, complex_cartesian,
                  complex_polar, complex_split, complex_merge
complex_split: (a boolean)
  argument: ``-complexsplit``, position: 1
  mutually_exclusive: real_polar, real_cartesian, complex_cartesian,
                  complex_polar, complex_split, complex_merge
complex_merge: (a boolean)
  argument: ``-complexmerge``, position: 1
  mutually_exclusive: real_polar, real_cartesian, complex_cartesian,
                  complex_polar, complex_split, complex_merge
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
            'NIFTI_PAIR_GZ')
  FSL output type
args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipype default value: {}))
  Environment variables

Outputs:

magnitude_out_file: (a pathlike object or string representing a file)
References:
None

63.9.3 ConvertWarp

Link to code
Wraps the executable command convertwarp.
Use FSL convertwarp for combining multiple transforms into one.

Examples

```python
>>> from nipype.interfaces.fsl import ConvertWarp
>>> warputils = ConvertWarp()
>>> warputils.inputs.warp1 = "warpfield.nii"
>>> warputils.inputs.reference = "T1.nii"
>>> warputils.inputs.relwarp = True
>>> warputils.inputs.output_type = "NIFTI_GZ"
>>> warputils.cmdline  # doctest: +ELLIPSIS
'convertwarp --ref=T1.nii --rel --warp1=warpfield.nii --out=T1_concatwarp.nii.gz'
>>> res = warputils.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
reference: (a pathlike object or string representing an existing file)
   Name of a file in target space of the full transform.
   argument: ``'--ref=%s'``, position: 1

[Optional]
out_file: (a pathlike object or string representing a file)
   Name of output file, containing warps that are the combination of all those given as arguments. The format of this will be a field-file (rather than spline coefficients) with any affine components included.
   argument: ``'--out=%s'``, position: -1
premat: (a pathlike object or string representing an existing file)
   filename for pre-transform (affine matrix)
   argument: ``'--premat=%s'``
warp1: (a pathlike object or string representing an existing file)
   Name of file containing initial warp-fields/coefficients (follows premat). This could e.g. be a fnirt-transform from a subjects structural scan to an average of a group of subjects.
   argument: ``'--warp1=%s'``
midmat: (a pathlike object or string representing an existing file)
   Name of file containing mid-warp-affine transform
   argument: ``'--midmat=%s'``
warp2: (a pathlike object or string representing an existing file)
   Name of file containing secondary warp-fields/coefficients (after...
warp1/midmat but before postmat). This could e.g. be a fnirt-transform from the average of a group of subjects to some standard space (e.g. MNI152).

argument: `--warp2=%s`

postmat: (a pathlike object or string representing an existing file)

Name of file containing an affine transform (applied last). It could e.g. be an affine transform that maps the MNI152-space into a better approximation to the Talairach-space (if indeed there is one).

argument: `--postmat=%s`

shift_in_file: (a pathlike object or string representing an existing file)

Name of file containing a "shiftmap", a non-linear transform with displacements only in one direction (applied first, before premat). This would typically be a fieldmap that has been pre-processed using fugue that maps a subjects functional (EPI) data onto an undistorted space (i.e. a space that corresponds to his/her true anatomy).

argument: `--shiftmap=%s`

shift_direction: ('y-' or 'y' or 'x' or 'x-' or 'z' or 'z-')

Indicates the direction that the distortions from --shiftmap goes. It depends on the direction and polarity of the phase-encoding in the EPI sequence.

argument: `--shiftdir=%s`

requires: shift_in_file

cons_jacobian: (a boolean)

Constrain the Jacobian of the warpfield to lie within specified min/max limits.

argument: `--constrainj`

jacobian_min: (a float)

Minimum acceptable Jacobian value for constraint (default 0.01)

argument: `--jmin=%f`

jacobian_max: (a float)

Maximum acceptable Jacobian value for constraint (default 100.0)

argument: `--jmax=%f`

abswarp: (a boolean)

If set it indicates that the warps in --warp1 and --warp2 should be interpreted as absolute. I.e. the values in --warp1/2 are the coordinates in the next space, rather than displacements. This flag is ignored if --warp1/2 was created by fnirt, which always creates relative displacements.

argument: `--abs`

mutually_exclusive: relwarp

relwarp: (a boolean)

If set it indicates that the warps in --warp1/2 should be interpreted as relative. I.e. the values in --warp1/2 are displacements from the coordinates in the next space.

argument: `--rel`

mutually_exclusive: abswarp

out_abswarp: (a boolean)

If set it indicates that the warps in --out should be absolute, i.e. the values in --out are displacements from the coordinates in --ref.

argument: `--absout`

mutually_exclusive: out_relwarp

out_relwarp: (a boolean)

If set it indicates that the warps in --out should be relative, i.e. the values in --out are displacements from the coordinates in --ref.

argument: `--relout`

mutually_exclusive: out_abswarp
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
  FSL output type
args: (a unicode string)
  Additional parameters to the command
  argument: `\%s`'
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
  Environment variables

Outputs:
out_file: (a pathlike object or string representing an existing file)
  Name of output file, containing the warp as field or coefficients.

References:
None

63.9.4 ConvertXFM

Link to code
Wraps the executable command convert_xfm.
Use the FSL utility convert_xfm to modify FLIRT transformation matrices.

Examples

```python
>>> import nipype.interfaces.fsl as fsl
>>> invt = fsl.ConvertXFM()
>>> invt.inputs.in_file = "flirt.mat"
>>> invt.inputs.invert_xfm = True
>>> invt.inputs.out_file = 'flirt_inv.mat'
>>> invt.cmdline
'convert_xfm -omat flirt_inv.mat -inverse flirt.mat'
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  input transformation matrix
  argument: `\%s`, position: -1

[Optional]
in_file2: (a pathlike object or string representing an existing file)
  second input matrix (for use with fix_scale_skew or concat_xfm)
  argument: `\%s`, position: -2
invert_xfm: (a boolean)
  invert input transformation
  argument: `\-inverse`, position: -3
  mutually_exclusive: invert_xfm, concat_xfm, fix_scale_skew
concat_xfm: (a boolean)
  write joint transformation of two input matrices
  argument: `\-concat`, position: -3
  mutually_exclusive: invert_xfm, concat_xfm, fix_scale_skew
  requires: in_file2
fix_scale_skew: (a boolean)
    use secondary matrix to fix scale and skew
    argument: `--fixscaleskew`, position: -3
    mutually_exclusive: invert_xfm, concat_xfm, fix_scale_skew
    requires: in_file2

out_file: (a pathlike object or string representing a file)
    final transformation matrix
    argument: `--omat %s`, position: 1

output_type: (`NIFTI` or `NIFTI_PAIR` or `NIFTI_GZ` or `NIFTI_PAIR_GZ`)
    FSL output type
args: (a unicode string)
    Additional parameters to the command
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})
    Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
    output transformation matrix

References:

None

63.9.5 CopyGeom

Link to code

Wraps the executable command `fslcpgeom`.

Use `fslcpgeom` to copy the header geometry information to another image. Copy certain parts of the header information (image dimensions, voxel dimensions, voxel dimensions units string, image orientation/origin or qform/sform info) from one image to another. Note that only copies from Analyze to Analyze or Nifti to Nifti will work properly. Copying from different files will result in loss of information or potentially incorrect settings.

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
    source image
    argument: `-%s`, position: 0
dest_file: (a pathlike object or string representing an existing file)
    destination image
    argument: `-%s`, position: 1

[Optional]
ignore_dims: (a boolean)
    Do not copy image dimensions
    argument: `--d`, position: -1
output_type: (`NIFTI` or `NIFTI_PAIR` or `NIFTI_GZ` or `NIFTI_PAIR_GZ`)
    FSL output type
args: (a unicode string)
    Additional parameters to the command
63.9.6 ExtractROI

Link to code
Wraps the executable command fslroi.
Uses FSL Fslroi command to extract region of interest (ROI) from an image.
You can a) take a 3D ROI from a 3D data set (or if it is 4D, the same ROI is taken from each time point and a new 4D data set is created), b) extract just some time points from a 4D data set, or c) control time and space limits to the ROI. Note that the arguments are minimum index and size (not maximum index). So to extract voxels 10 to 12 inclusive you would specify 10 and 3 (not 10 and 12).

Examples

```python
>>> from nipype.interfaces.fsl import ExtractROI
>>> from nipype.testing import anatfile

>>> fslroi = ExtractROI(in_file=anatfile, roi_file='bar.nii', t_min=0, ...
...                     t_size=1)
>>> fslroi.cmdline == 'fslroi %s bar.nii 0 1' % anatfile
True
```

References:
None
argument: `\`%d\'`, position: 7
t_min: (an integer (int or long))
  argument: `\`%d\'`, position: 8
t_size: (an integer (int or long))
  argument: `\`%d\'`, position: 9
crop_list: (a list of items which are a tuple of the form: (an
  integer (int or long), an integer (int or long)))
  list of two tuples specifying crop options
  argument: `\`%s\'`, position: 2
  mutually_exclusive: x_min, x_size, y_min, y_size, z_min, z_size,
  t_min, t_size
output_type: (`NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
  'NIFTI_PAIR_GZ')
  FSL output type
args: (a unicode string)
  Additional parameters to the command
  argument: `\`%s\'`
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
  Environment variables

Outputs:

roi_file: (a pathlike object or string representing an existing file)

References:

None

63.9.7 FilterRegressor

Link to code
Wraps the executable command fsl_regfilt.
Data de-noising by regressing out part of a design matrix
Uses simple OLS regression on 4D images
Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  input file name (4D image)
  argument: `\`-i %s\'`, position: 1
design_file: (a pathlike object or string representing an existing
  file)
  name of the matrix with time courses (e.g. GLM design or MELODIC
  mixing matrix)
  argument: `\`-d %s\'`, position: 3
filter_columns: (a list of items which are an integer (int or long))
  (1-based) column indices to filter out of the data
  argument: `\`-f %s\'`, position: 4
  mutually_exclusive: filter_all
filter_all: (a boolean)
  use all columns in the design file in denoising
  argument: `\`-f %s\'`, position: 4
  mutually_exclusive: filter_columns

(continues on next page)
out_file: (a pathlike object or string representing a file)
   output file name for the filtered data
      argument: `\`-o %s`, position: 2
mask: (a pathlike object or string representing an existing file)
   mask image file name
      argument: `\`-m %s`
var_norm: (a boolean)
   perform variance-normalization on data
      argument: `\`--vn`
out_vnscales: (a boolean)
   output scaling factors for variance normalization
      argument: `\`--out_vnscales`
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
   FSL output type
args: (a unicode string)
   Additional parameters to the command
      argument: `\`%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}))
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
   output file name for the filtered data

References:
None

63.9.8 ImageMaths

Link to code
Wraps the executable command fslmaths.
Use FSL fslmaths command to allow mathematical manipulation of images FSL info

Examples

```python
>>> from nipype.interfaces import fsl
>>> from nipype.testing import anatfile
>>>
>>> maths = fsl.ImageMaths(in_file=anatfile, op_string= '-add 5', ...
...   out_file='foo_maths.nii')
>>> maths.cmdline == 'fslmaths %s -add 5 foo_maths.nii' % anatfile
True
```

Inputs:

[Mandatory]

in_file: (a pathlike object or string representing an existing file)
   argument: `\`%s\`, position: 1

[Optional]

in_file2: (a pathlike object or string representing an existing file)
mask_file: (a pathlike object or string representing an existing file)
    use (following image>0) to mask current image
    argument: `'-mas %s'`
out_file: (a pathlike object or string representing a file)
    argument: `'-%s'`, position: -2
op_string: (a unicode string)
    string defining the operation, i.e. -add
    argument: `'-%s'`, position: 2
suffix: (a unicode string)
    out_file suffix
out_data_type: ('char' or 'short' or 'int' or 'float' or 'double' or 'input')
    output datatype, one of (char, short, int, float, double, input)
    argument: `'-odt %s'`, position: -1
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
    FSL output type
args: (a unicode string)
    Additional parameters to the command
    argument: `'-%s'`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
<x y z> requested spatial coordinate (instead of mask)
argument: `'-%c %s'`

use_mm: (a boolean)
use mm instead of voxel coordinates (for -c option)
argument: `'--usemm'`

show_all: (a boolean)
show all voxel time series (within mask) instead of averaging
argument: `'--showall'`

eig: (a boolean)
calculate Eigenvariate(s) instead of mean (output will have 0 mean)
argument: `'--eig'`

order: (an integer (int or long), nipype default value: 1)
select number of Eigenvariates
argument: `'--order=%d'`

nobin: (a boolean)
do not binarise the mask for calculation of Eigenvariates
argument: `'--no_bin'`

transpose: (a boolean)
output results in transpose format (one row per voxel/mean)
argument: `'--transpose'`

output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
FSL output type

args: (a unicode string)
Additional parameters to the command
argument: `'%s'`

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
path/name of output text matrix

References:
None

63.9.10 ImageStats

Link to code
Wraps the executable command fslstats.
Use FSL fslstats command to calculate stats from images FSL info

Examples

```python
>>> from nipype.interfaces.fsl import ImageStats
>>> from nipype.testing import funcfile
>>> stats = ImageStats(in_file=funcfile, op_string= '-M')
>>> stats.cmdline == 'fslstats %s -M' %funcfile
True
```
[Mandatory]
in_file: (a pathlike object or string representing an existing file)
    input file to generate stats of
    argument: `''%s''`, position: 3
op_string: (a unicode string)
    string defining the operation, options are applied in order, e.g. -M
    -l 10 -M will report the non-zero mean, apply a threshold and then
    report the new nonzero mean
    argument: `''%s''`, position: 4

[Optional]
split_4d: (a boolean)
    give a separate output line for each 3D volume of a 4D timeseries
    argument: `''-t''`, position: 1
mask_file: (a pathlike object or string representing an existing
    file)
    mask file used for option `-k %s`
index_mask_file: (a pathlike object or string representing an
    existing file)
    generate separate n submasks from indexMask, for index values 1..n
    where n is the maximum index value in indexMask, and generate
    statistics for each submask
    argument: `''-K %s''`, position: 2
output_type: (`'NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
    'NIFTI_PAIR_GZ'`)  
    FSL output type
args: (a unicode string)
    Additional parameters to the command
    argument: `''%s''`
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})  
    Environment variables

Outputs:

class out_stat: (any value)
    stats output

References:

None

63.9.11 InvWarp

Link to code
Wraps the executable command invwarp.
Use FSL Invwarp to invert a FNIRT warp

Examples

```python
>>> from nipype.interfaces.fsl import InvWarp
>>> invwarp = InvWarp()
>>> invwarp.inputs.warp = "struct2mni.nii"
>>> invwarp.inputs.reference = "anatomical.nii"
>>> invwarp.inputs.output_type = "NIFTI_GZ"
```
>>> invwarp.cmdline
'invwarp --out=struct2mni_inverse.nii.gz --ref=anatomical.nii --warp=struct2mni.
→ni'
>>> res = invwarp.run() # doctest: +SKIP

Inputs:

[Mandatory]

warp: (a pathlike object or string representing an existing file)
    Name of file containing warp-coefficients/fields. This would
typically be the output from the --cout switch of fnirt (but can
also use fields, like the output from --fout).
    argument: `--warp=%s`

reference: (a pathlike object or string representing an existing file)
    Name of a file in target space. Note that the target space is now
different from the target space that was used to create the --warp
file. It would typically be the file that was specified with the
--in argument when running fnirt.
    argument: `--ref=%s`

(Optional)

inverse_warp: (a pathlike object or string representing a file)
    Name of output file, containing warps that are the "reverse" of
those in --warp. This will be a field-file (rather than a file of
spline coefficients), and it will have any affine component included
as part of the displacements.
    argument: `--out=%s`

absolute: (a boolean)
    If set it indicates that the warps in --warp should be interpreted
as absolute, provided that it is not created by fnirt (which always
uses relative warps). If set it also indicates that the output --out
should be absolute.
    argument: `--abs`
    mutually_exclusive: relative

relative: (a boolean)
    If set it indicates that the warps in --warp should be interpreted
as relative. I.e. the values in --warp are displacements from the
coordinates in the --ref space. If set it also indicates that the output
--out should be relative.
    argument: `--rel`
    mutually_exclusive: absolute

niter: (an integer (int or long))
    Determines how many iterations of the gradient-descent search that
should be run.
    argument: `--niter=%d`

regularise: (a float)
    Regularization strength (default=1.0).
    argument: `--regularise=%f`

noconstraint: (a boolean)
    Do not apply Jacobian constraint
    argument: `--noconstraint`

jacobian_min: (a float)
    Minimum acceptable Jacobian value for constraint (default 0.01)
    argument: `--jmin=%f`

jacobian_max: (a float)
    Maximum acceptable Jacobian value for constraint (default 100.0)
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
FSL output type
args: (a unicode string)
Additional parameters to the command
argument: ```%s```  
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: `{}`)
Environment variables

Outputs:

inverse_warp: (a pathlike object or string representing an existing file)
Name of output file, containing warps that are the "reverse" of those in --warp.

References:
None

63.9.12 Merge

Link to code
Wraps the executable command fslmerge.
Use fslmerge to concatenate images
Images can be concatenated across time, x, y, or z dimensions. Across the time (t) dimension the TR is set by default to 1 sec.
Note: to set the TR to a different value, specify 't' for dimension and specify the TR value in seconds for the tr input. The dimension will be automatically updated to 'tr'.

Examples

```python
>>> from nipype.interfaces.fsl import Merge
>>> merger = Merge()
>>> merger.inputs.in_files = ['functional2.nii', 'functional3.nii']
>>> merger.inputs.dimension = 't'
>>> merger.inputs.output_type = 'NIFTI_GZ'
>>> merger.cmdline
'fslmerge -t functional2_merged.nii.gz functional2.nii functional3.nii'
>>> merger.inputs.tr = 2.25
>>> merger.cmdline
'fslmerge -tr functional2_merged.nii.gz functional2.nii functional3.nii 2.25'
```

Inputs:

[Mandatory]
in_files: (a list of items which are a pathlike object or string representing an existing file)
argument: ```%s```, position: 2
dimension: (`t' or 'x' or 'y' or 'z' or 'a')
dimension along which to merge, optionally set tr input when dimension is t
argument: ```-%s```, position: 0
[Optional]
tr: (a float)
use to specify TR in seconds (default is 1.00 sec), overrides
dimension and sets it to tr
argument: `%2f`, position: -1

merged_file: (a pathlike object or string representing a file)
argument: `%s`, position: 1
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
'NIFTI_PAIR_GZ')
FSL output type
args: (a unicode string)
Additional parameters to the command
argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

merged_file: (a pathlike object or string representing an existing
file)

References:
None

63.9.13 MotionOutliers

Link to code
Wraps the executable command fsl_motion_outliers.
Use FSL fsl_motion_outliers'http://fsl.fmrib.ox.ac.uk/fsl/fslwiki/FSLMotionOutliers' to find outliers in time-
series (4d) data. Examples ~~~~~~~~ >>> from nipype.interfaces.fsl import MotionOutliers >>> mo = Mo-
tionOutliers() >>> mo.inputs.in_file = "epi.nii" >>> mo.cmdline # doctest: +ELLIPSIS 'fsl_motion_outliers -i
epi.nii -o epi_outliers.txt -p epi_metrics.png -s epi_metrics.txt' >>> res = mo.run() # doctest: +SKIP

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
unfiltered 4D image
argument: `%i %s`

[Optional]
out_file: (a pathlike object or string representing a file)
output outlier file name
argument: `%o %s`
mask: (a pathlike object or string representing an existing file)
mask image for calculating metric
argument: `%m %s`
metric: ("refms" or "dvars" or "refmse" or "fd" or "fdrms")
metrics: refms - RMS intensity difference to reference volume as
metric [default metric], refmse - Mean Square Error version of
refms (used in original version of fsl_motion_outliers), dvars -
DVARS, fd - frame displacement, fdrms - FD with RMS matrix
calculation
argument: `--%s`
threshold: (a float)
specify absolute threshold value (otherwise use box-plot cutoff = P75 + 1.5*IQR)
argument: `--thresh=%g`
no_motion_correction: (a boolean)
do not run motion correction (assumed already done)
argument: `--nomoco`
dummy: (an integer (int or long))
number of dummy scans to delete (before running anything and creating EVs)
argument: `--dummy=%d`
out_metric_values: (a pathlike object or string representing a file)
output metric values (DVARS etc.) file name
argument: `--s %s`
out_metric_plot: (a pathlike object or string representing a file)
output metric values plot (DVARS etc.) file name
argument: `--p %s`
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
FSL output type
args: (a unicode string)
Additional parameters to the command
argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
out_metric_values: (a pathlike object or string representing an existing file)
out_metric_plot: (a pathlike object or string representing an existing file)

References:

None

63.9.14 Overlay

Link to code
Wraps the executable command overlay.
Use FSL's overlay command to combine background and statistical images into one volume

Examples

```python
>>> from nipype.interfaces import fsl
>>> combine = fsl.Overlay()
>>> combine.inputs.background_image = 'mean_func.nii.gz'
>>> combine.inputs.auto_thresh_bg = True
>>> combine.inputs.stat_image = 'zstat1.nii.gz'
>>> combine.inputs.stat_thresh = (3.5, 10)
```
combine.inputs.show_negative_stats = True
res = combine.run() #doctest: +SKIP

Inputs:

[Mandatory]
background_image: (a pathlike object or string representing an existing file)
image to use as background
argument: ``%s``, position: 4

auto_thresh_bg: (a boolean)
automatically threshold the background image
argument: ``-a`` (position: 5
mutually exclusive: auto_thresh_bg, full_bg_range, bg_thresh

full_bg_range: (a boolean)
use full range of background image
argument: ``-A`` (position: 5
mutually exclusive: auto_thresh_bg, full_bg_range, bg_thresh

bg_thresh: (a tuple of the form: (a float, a float))
min and max values for background intensity
argument: ``%.3f %.3f`` (position: 5
mutually exclusive: auto_thresh_bg, full_bg_range, bg_thresh

stat_image: (a pathlike object or string representing an existing file)
statistical image to overlay in color
argument: ``%s``, position: 6

stat_thresh: (a tuple of the form: (a float, a float))
min and max values for the statistical overlay
argument: ``%.2f %.2f`` (position: 7

[Optional]
transparency: (a boolean, nipype default value: True)
make overlay colors semi-transparent
argument: ``%s``, position: 1

out_type: ('float' or 'int', nipype default value: float)
write output with float or int
argument: ``%s``, position: 2

use_checkerboard: (a boolean)
use checkerboard mask for overlay
argument: ``-c`` (position: 3

show_negative_stats: (a boolean)
display negative statistics in overlay
argument: ``%s``, position: 8
mutually exclusive: stat_image2

stat_image2: (a pathlike object or string representing an existing file)
second statistical image to overlay in color
argument: ``%s``, position: 9
mutually exclusive: show_negative_stats

stat_thresh2: (a tuple of the form: (a float, a float))
min and max values for second statistical overlay
argument: ``%.2f %.2f`` (position: 10

out_file: (a pathlike object or string representing a file)
combined image volume
argument: ``%s``, position: -1

output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
FSL output type
args: (a unicode string)
   Additional parameters to the command
   argument: `\%s`
environ: (a dictionary with keys which are a bytes or None or a value
   of class 'str' and with values which are a bytes or None or a value
   of class 'str', nipype default value: {})
   Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
   combined image volume

References:
None

63.9.15 PlotMotionParams

Link to code
Wraps the executable command fsl_tsplot.
Use fsl_tsplot to plot the estimated motion parameters from a realignment program.

Examples

```python
>>> import nipype.interfaces.fsl as fsl
>>> plotter = fsl.PlotMotionParams()
>>> plotter.inputs.in_file = 'functional.par'
>>> plotter.inputs.in_source = 'fsl'
>>> plotter.inputs.plot_type = 'rotations'
>>> res = plotter.run()  # doctest: +SKIP
```

Notes

The ‘in_source’ attribute determines the order of columns that are expected in the source file. FSL prints motion parameters in the order rotations, translations, while SPM prints them in the opposite order. This interface should be able to plot timecourses of motion parameters generated from other sources as long as they fall under one of these two patterns. For more flexibility, see the fsl.PlotTimeSeries interface.

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file
   or a list of items which are a pathlike object or string
   representing an existing file)
   file with motion parameters
   argument: `\%s`, position: 1
in_source: ('spm' or 'fsl')
   which program generated the motion parameter file - fsl, spm
plot_type: ('rotations' or 'translations' or 'displacement')
   which motion type to plot - rotations, translations, displacement
   argument: `\%s`

[Optional]
plot_size: (a tuple of the form: (an integer (int or long), an
Wraps the executable command `fsl_tsplot`.
Use fsl_tsplot to create images of time course plots.

Examples

```python
>>> import nipype.interfaces.fsl as fsl
>>> plotter = fsl.PlotTimeSeries()
>>> plotter.inputs.in_file = 'functional.par'
>>> plotter.inputs.title = 'Functional timeseries'
>>> plotter.inputs.labels = ['run1', 'run2']
>>> plotter.run() #doctest: +SKIP
```

References:

None

63.9.16 PlotTimeSeries

Link to code

Wraps the executable command `fsl_tsplot`.
Use fsl_tsplot to create images of time course plots.

Examples

```python
>>> import nipype.interfaces.fsl as fsl
>>> plotter = fsl.PlotTimeSeries()
>>> plotter.inputs.in_file = 'functional.par'
>>> plotter.inputs.title = 'Functional timeseries'
>>> plotter.inputs.labels = ['run1', 'run2']
>>> plotter.run() #doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file or a list of items which are a pathlike object or string representing an existing file) file or list of files with columns of timecourse information
argument: ``\`%s\`` , position: 1

[Optional]
plot_start: (an integer (int or long)) first column from in-file to plot argument: ``\`--start=%d\`` mutually_exclusive: plot_range
plot_finish: (an integer (int or long))
final column from in-file to plot
argument: `--finish=%d`
mutually_exclusive: plot_range

plot_range: (a tuple of the form: (an integer (int or long), an integer (int or long)))
first and last columns from the in-file to plot
argument: `%(s)`
mutually_exclusive: plot_start, plot_finish
title: (a unicode string)
plot title
argument: `%(s)`

legend_file: (a pathlike object or string representing an existing file)
legend file
argument: `--legend=%s`

labels: (a unicode string or a list of items which are a unicode string)
label or list of labels
argument: `%(s)`
y_min: (a float)
minimum y value
argument: `--ymin=%.2f`
mutually_exclusive: y_range

y_max: (a float)
maximum y value
argument: `--ymax=%.2f`
mutually_exclusive: y_range

y_range: (a tuple of the form: (a float, a float))
min and max y axis values
argument: `%(s)`
mutually_exclusive: y_min, y_max

x_units: (an integer (int or long), nipype default value: 1)
scaling units for x-axis (between 1 and length of in file)
argument: `-u %d`

plot_size: (a tuple of the form: (an integer (int or long), an integer (int or long)))
plot image height and width
argument: `%(s)`

x_precision: (an integer (int or long))
precision of x-axis labels
argument: `--precision=%d`

sci_notation: (a boolean)
switch on scientific notation
argument: `--sci`

out_file: (a pathlike object or string representing a file)
image to write
argument: `--o %s`

output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
FSL output type

args: (a unicode string)
Additional parameters to the command
argument: `%(s)`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables
### Outputs:

<table>
<thead>
<tr>
<th>out_file: (a pathlike object or string representing an existing file)</th>
</tr>
</thead>
<tbody>
<tr>
<td>image to write</td>
</tr>
</tbody>
</table>

### References:

None

#### 63.9.17 PowerSpectrum

**Link to code**

Wraps the executable command `fslpspec`. Use FSL PowerSpectrum command for power spectrum estimation.

**Examples**

```python
>>> from nipype.interfaces import fsl
>>> pspec = fsl.PowerSpectrum()
>>> pspec.inputs.in_file = 'functional.nii'
>>> res = pspec.run()  # doctest: +SKIP
```

### Inputs:

[Mandatory]

<table>
<thead>
<tr>
<th>in_file: (a pathlike object or string representing an existing file)</th>
</tr>
</thead>
<tbody>
<tr>
<td>input 4D file to estimate the power spectrum</td>
</tr>
<tr>
<td>argument: <code>\</code>%s``, position: 0</td>
</tr>
</tbody>
</table>

[Optional]

<table>
<thead>
<tr>
<th>out_file: (a pathlike object or string representing a file)</th>
</tr>
</thead>
<tbody>
<tr>
<td>name of output 4D file for power spectrum</td>
</tr>
<tr>
<td>argument: <code>\</code>%s``, position: 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSL output type</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>args: (a unicode string)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Additional parameters to the command</td>
</tr>
<tr>
<td>argument: <code>\</code>%s``</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment variables</td>
</tr>
</tbody>
</table>

### Outputs:

<table>
<thead>
<tr>
<th>out_file: (a pathlike object or string representing an existing file)</th>
</tr>
</thead>
<tbody>
<tr>
<td>path/name of the output 4D power spectrum file</td>
</tr>
</tbody>
</table>

### References:

None

#### 63.9.18 Reorient2Std

**Link to code**

Wraps the executable command `fslreorient2std`. 
fslreorient2std is a tool for reorienting the image to match the approximate orientation of the standard template images (MNI152).

**Examples**

```python
>>> reorient = Reorient2Std()
>>> reorient.inputs.in_file = "functional.nii"
>>> res = reorient.run()  # doctest: +SKIP
```

**Inputs:**

- **in_file**: (a pathlike object or string representing an existing file)
  - argument: `"%s"`

**Optional**

- **out_file**: (a pathlike object or string representing a file)
  - argument: `"%s"
- **output_type**: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
  - FSL output type
- **args**: (a unicode string)
  - Additional parameters to the command
  - argument: `"%s"
- **environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: `{}`)
  - Environment variables

**Outputs:**

- **out_file**: (a pathlike object or string representing an existing file)

**References:**

None

### 63.9.19 RobustFOV

**Link to code**

Wraps the executable command robustfov. Automatically crops an image removing lower head and neck. Interface is stable 5.0.0 to 5.0.9, but default brainsize changed from 150mm to 170mm.

**Inputs:**

- **in_file**: (a pathlike object or string representing an existing file)
  - input filename
  - argument: `"-i %s"`, position: 0

**Optional**

- **out_roi**: (a pathlike object or string representing a file)
  - ROI volume output name
  - argument: `"-r %s"
- **brainsize**: (an integer (int or long))
  - size of brain in z-dimension (default 170mm/150mm)
  - argument: `"-b %d"

(continues on next page)
out_transform: (a pathlike object or string representing a file)
    Transformation matrix in_file to out_roi output name
    argument: `'~m %s`'
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
    'NIFTI_PAIR_GZ')
    FSL output type
args: (a unicode string)
    Additional parameters to the command
    argument: `'%s'`
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {}) 
    Environment variables

Outputs:

out_roi: (a pathlike object or string representing an existing file)
    ROI volume output name
out_transform: (a pathlike object or string representing an existing
    file)
    Transformation matrix in_file to out_roi output name

References:

None

63.9.20 SigLoss

Link to code

Wraps the executable command `sigloss`. Estimates signal loss from a field map (in rad/s)

Examples

```python
>>> sigloss = SigLoss()
>>> sigloss.inputs.in_file = "phase.nii"
>>> sigloss.inputs.echo_time = 0.03
>>> res = sigloss.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
    b0 fieldmap file
    argument: `'-i %s'`

[Optional]
out_file: (a pathlike object or string representing a file)
    output signal loss estimate file
    argument: `'-s %s'`
mask_file: (a pathlike object or string representing an existing file)
    brain mask file
    argument: `'-m %s'`

echo_time: (a float)
    echo time in seconds
argument: ``--te=%f``
slice_direction: ('x' or 'y' or 'z')
slicing direction
argument: ``-d %s``
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
FSL output type
args: (a unicode string)
   Additional parameters to the command
argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
   Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
   signal loss estimate file

References:

None

63.9.21 Slice

Link to code
Wraps the executable command fslslice.
Use fslslice to split a 3D file into lots of 2D files (along z-axis).

Examples

```python
>>> from nipype.interfaces.fsl import Slice
>>> slice = Slice()
>>> slice.inputs.in_file = 'functional.nii'
>>> slice.inputs.out_base_name = 'sl'
>>> slice.cmdline
'fslslice functional.nii sl'
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
   input filename
   argument: ```%s``` , position: 0

[Optional]
out_base_name: (a unicode string)
   outputs prefix
   argument: ```%s``` , position: 1
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
   FSL output type
args: (a unicode string)
   Additional parameters to the command
   argument: ```%s```
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

out_files: (a list of items which are a pathlike object or string representing an existing file)

References:

None

63.9.22 Slicer

Link to code
Wraps the executable command slicer.
Use FSL's slicer command to output a png image from a volume.

Examples

```python
>>> from nipype.interfaces import fsl
>>> from nipype.testing import example_data
>>>
slice = fsl.Slicer()
>>> slice.inputs.in_file = example_data('functional.nii')
>>> slice.inputs.all_axial = True
>>> slice.inputs.image_width = 750
>>> res = slice.run() #doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  input volume
  argument: `''%s''`, position: 1

[Optional]
image_edges: (a pathlike object or string representing an existing file)
  volume to display edge overlay for (useful for checking registration
  argument: `''%s''`, position: 2
label_slices: (a boolean, nipype default value: True)
  display slice number
  argument: `''-L''`, position: 3
colour_map: (a pathlike object or string representing an existing file)
  use different colour map from that stored in nifti header
  argument: `''-L %s''`, position: 4
intensity_range: (a tuple of the form: (a float, a float))
  min and max intensities to display
  argument: `''-i %.3f %.3f''`, position: 5
threshold_edges: (a float)
  use threshold for edges
  argument: `''-t %.3f''`, position: 6
dither_edges: (a boolean)
produce semi-transparent (dithered) edges
argument: `'-t'`, position: 7

nearest_neighbour: (a boolean)
use nearest neighbor interpolation for output
argument: `'-n'`, position: 8

show_orientation: (a boolean, nipype default value: True)
label left-right orientation
argument: `'-s'`, position: 9

double_slice: ("x" or "y" or "z")
output picture of single slice in the x, y, or z plane
argument: `'-%s'`, position: 10
mutually_exclusive: single_slice, middle_slices, all_axial,
sample_axial
requires: slice_number

slice_number: (an integer (int or long))
slice number to save in picture
argument: `'-%d'`, position: 11

middle_slices: (a boolean)
output picture of mid-sagittal, axial, and coronal slices
argument: `'-a'`, position: 10
mutually_exclusive: single_slice, middle_slices, all_axial,
sample_axial

all_axial: (a boolean)
output all axial slices into one picture
argument: `'-A'`, position: 10
mutually_exclusive: single_slice, middle_slices, all_axial,
sample_axial
requires: image_width

sample_axial: (an integer (int or long))
output every n axial slices into one picture
argument: `'-S %d'`, position: 10
mutually_exclusive: single_slice, middle_slices, all_axial,
sample_axial
requires: image_width

image_width: (an integer (int or long))
max picture width
argument: `'-d'`, position: -2

out_file: (a pathlike object or string representing a file)
picture to write
argument: `'%s'`, position: -1

scaling: (a float)
image scale
argument: `'-s %f'`, position: 0

output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or
'NIFTI_PAIR_GZ')
FSL output type

args: (a unicode string)
Additional parameters to the command
argument: `'%s'`

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {}) Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
References:
None

63.9.23 Smooth

Link to code
Wraps the executable command fslmaths.
Use fslmaths to smooth the image

Examples

Setting the kernel width using sigma:

```python
>>> sm = Smooth()
>>> sm.inputs.output_type = 'NIFTI_GZ'
>>> sm.inputs.in_file = 'functional2.nii'
>>> sm.inputs.sigma = 8.0
>>> sm.cmdline # doctest: +ELLIPSIS
'fslmaths functional2.nii -kernel gauss 8.000 -fmean functional2_smooth.nii.gz'
```

Setting the kernel width using fwhm:

```python
>>> sm = Smooth()
>>> sm.inputs.output_type = 'NIFTI_GZ'
>>> sm.inputs.in_file = 'functional2.nii'
>>> sm.inputs.fwhm = 8.0
>>> sm.cmdline # doctest: +ELLIPSIS
'fslmaths functional2.nii -kernel gauss 3.397 -fmean functional2_smooth.nii.gz'
```

One of sigma or fwhm must be set:

```python
>>> from nipype.interfaces.fsl import Smooth
>>> sm = Smooth()
>>> sm.inputs.output_type = 'NIFTI_GZ'
>>> sm.inputs.in_file = 'functional2.nii'
>>> sm.cmdline # doctest: +ELLIPSIS
Traceback (most recent call last):
  ...: ValueError: Smooth requires a value for one of the inputs ...
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  argument: ``%s``, position: 0
sigma: (a float)
  gaussian kernel sigma in mm (not voxels)
  argument: ``-kernel gauss %.03f -fmean``
  position: 1
  mutually_exclusive: fwhm
fwhm: (a float)
  gaussian kernel fwhm, will be converted to sigma in mm (not voxels)
  argument: ``-kernel gauss %.03f -fmean``
  position: 1
  mutually_exclusive: sigma
63.9. interfaces.fsl.utils

63.9.24 Split

References:
None

63.9.24 Split

Wraps the executable command fslsplit. Uses FSL Fslsplit command to separate a volume into images in time, x, y or z dimension.

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  input filename
  argument: `"%s"`, position: 0
dimension: ('t' or 'x' or 'y' or 'z')
  dimension along which the file will be split
  argument: `"-%s"`, position: 2

[Optional]
out_base_name: (a unicode string)
  outputs prefix
  argument: `"%s"`, position: 1
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
  FSL output type
args: (a unicode string)
  Additional parameters to the command
  argument: `"%s"`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  Environment variables

Outputs:

out_files: (a list of items which are a pathlike object or string representing an existing file)
63.9.25 SwapDimensions

Link to code
Wraps the executable command fslswapdim.
Use fslswapdim to alter the orientation of an image.
This interface accepts a three-tuple corresponding to the new orientation. You may either provide dimension ids in the form of (-)x, (-)y, or (-z), or nifti-style dimension codes (RL, LR, AP, PA, IS, SI).
Inputs:

<table>
<thead>
<tr>
<th>Mandatory</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_file: (a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>new_dims: (a tuple of the form: ('x' or '-x' or 'y' or '-y' or 'z' or '-z' or 'RL' or 'LR' or 'AP' or 'PA' or 'IS' or 'SI', 'x' or '-x' or 'y' or '-y' or 'z' or '-z' or 'RL' or 'LR' or 'AP' or 'PA' or 'IS' or 'SI', 'x' or '-x' or 'y' or '-y' or 'z' or '-z' or 'RL' or 'LR' or 'AP' or 'PA' or 'IS' or 'SI'))</td>
</tr>
<tr>
<td>[Optional]</td>
</tr>
<tr>
<td>out_file: (a pathlike object or string representing a file)</td>
</tr>
<tr>
<td>output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')</td>
</tr>
<tr>
<td>args: (a unicode string)</td>
</tr>
<tr>
<td>environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})</td>
</tr>
</tbody>
</table>

Outputs:

| out_file: (a pathlike object or string representing an existing file) |

References:
None

63.9.26 WarpPoints

Link to code
Wraps the executable command img2imgcoord.
Use FSL img2imgcoord to transform point sets. Accepts plain text files and vtk files.

Note: transformation of TrackVis trk files is not yet implemented
Examples

```python
>>> from nipype.interfaces.fsl import WarpPoints
>>> warppoints = WarpPoints()
>>> warppoints.inputs.in_coords = 'surf.txt'
>>> warppoints.inputs.src_file = 'epi.nii'
>>> warppoints.inputs.dest_file = 'T1.nii'
>>> warppoints.inputs.warp_file = 'warpfield.nii'
>>> warppoints.inputs.coord_mm = True
>>> warppoints.cmdline  # doctest: +ELLIPSIS
'img2imgcoord -mm -dest T1.nii -src epi.nii -warp warpfield.nii surf.txt'
>>> res = warppoints.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]

src_file: (a pathlike object or string representing an existing file)
  filename of source image
  argument: `'-src %s'`

dest_file: (a pathlike object or string representing an existing file)
  filename of destination image
  argument: `'-dest %s'`

in_coords: (a pathlike object or string representing an existing file)
  filename of file containing coordinates
  argument: `'%s'`, position: -1

[Optional]

xfm_file: (a pathlike object or string representing an existing file)
  filename of affine transform (e.g. source2dest.mat)
  argument: `'-xfm %s'`
  mutually_exclusive: warp_file

warp_file: (a pathlike object or string representing an existing file)
  filename of warpfield (e.g. intermediate2dest_warp.nii.gz)
  argument: `'-warp %s'`
  mutually_exclusive: xfm_file

coord_vox: (a boolean)
  all coordinates in voxels - default
  argument: `'-vox'`
  mutually_exclusive: coord_mm

coord_mm: (a boolean)
  all coordinates in mm
  argument: `'-mm'`
  mutually_exclusive: coord_vox

out_file: (a pathlike object or string representing a file)
  output file name

args: (a unicode string)
  Additional parameters to the command
  argument: `'%s'`

environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
63.9.27 WarpPointsFromStd

Link to code
Wraps the executable command `std2imgcoord`.
Use FSL `std2imgcoord` to transform point sets to standard space coordinates. Accepts plain text coordinates files.

Examples

```python
>>> from nipype.interfaces.fsl import WarpPointsFromStd
>>> warppoints = WarpPointsFromStd()
>>> warppoints.inputs.in_coords = 'surf.txt'
>>> warppoints.inputs.img_file = 'T1.nii'
>>> warppoints.inputs.std_file = 'mni.nii'
>>> warppoints.inputs.warp_file = 'warpfield.nii'
>>> warppoints.inputs.coord_mm = True
>>> warppoints.cmdline  # doctest: +ELLIPSIS
'std2imgcoord -mm -img T1.nii -std mni.nii -warp warpfield.nii surf.txt'
>>> res = warppoints.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
- `img_file`: (a pathlike object or string representing an existing file)
  - filename of a destination image
  - argument: `''-img %s''`
- `std_file`: (a pathlike object or string representing an existing file)
  - filename of the image in standard space
  - argument: `''-std %s''`
- `in_coords`: (a pathlike object or string representing an existing file)
  - filename of file containing coordinates
  - argument: `''%s''`, position: -2

[Optional]
- `xfm_file`: (a pathlike object or string representing an existing file)
  - filename of affine transform (e.g. source2dest.mat)
  - argument: `''-xfm %s''`
  - mutually_exclusive: warp_file
- `warp_file`: (a pathlike object or string representing an existing file)
  - filename of warpfield (e.g. intermediate2dest_warp.nii.gz)
  - argument: `''-warp %s''`
  - mutually_exclusive: xfm_file
- `coord_vox`: (a boolean)
  - all coordinates in voxels - default
  - argument: `''-vox''`
  - mutually_exclusive: coord_mm
- `coord_mm`: (a boolean)
  - all coordinates in mm
  - argument: `''-mm''`
  - mutually_exclusive: coord_vox
- `args`: (a unicode string)
Additional parameters to the command
argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {}) Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
Name of output file, containing the warp as field or coefficients.

63.9.28 WarpPointsToStd

Link to code
Wraps the executable command img2stdcoord.
Use FSL img2stdcoord to transform point sets to standard space coordinates. Accepts plain text files and vtk files.

Note: transformation of TrackVis trk files is not yet implemented

Examples

```python
>>> from nipype.interfaces.fsl import WarpPointsToStd
>>> warppoints = WarpPointsToStd()
>>> warppoints.inputs.in_coords = 'surf.txt'
>>> warppoints.inputs.img_file = 'T1.nii'
>>> warppoints.inputs.std_file = 'mni.nii'
>>> warppoints.inputs.warp_file = 'warpfield.nii'
>>> warppoints.inputs.coord_mm = True
>>> warppoints.cmdline # doctest: +ELLIPSIS
'img2stdcoord -mm -img T1.nii -std mni.nii -warp warpfield.nii surf.txt'
```

Inputs:

[Mandatory]
img_file: (a pathlike object or string representing an existing file)
filename of input image
argument: ``-img %s``
std_file: (a pathlike object or string representing an existing file)
filename of destination image
argument: ``-std %s``
in_coords: (a pathlike object or string representing an existing file)
filename of file containing coordinates
argument: ``%s``", position: -1

[Optional]
premat_file: (a pathlike object or string representing an existing file)
filename of pre-warp affine transform (e.g.
example_func2highres.mat)
argument: ``-premat %s``
xfm_file: (a pathlike object or string representing an existing file)
filename of affine transform (e.g. source2dest.mat)
argument: `'-xfm %s'`
mutually_exclusive: warp_file
warp_file: (a pathlike object or string representing an existing file)
filename of warpfield (e.g. intermediate2dest_warp.nii.gz)
argument: `'-warp %s'`
mutually_exclusive: xfm_file
coord_vox: (a boolean)
all coordinates in voxels - default
argument: `'-vox'`
mutually_exclusive: coord_mm
coord_mm: (a boolean)
all coordinates in mm
argument: `'-mm'`
mutually_exclusive: coord_vox
out_file: (a pathlike object or string representing a file)
output file name
args: (a unicode string)
Additional parameters to the command
argument: `'%s'`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:
out_file: (a pathlike object or string representing an existing file)
Name of output file, containing the warp as field or coefficients.

63.9.29 WarpUtils

Link to code
Wraps the executable command fnirtfileutils.
Use FSL fnirtfileutils to convert field->coefficients, coefficients->field, coefficients->other_coefficients etc

Examples

```python
>>> from nipype.interfaces.fsl import WarpUtils
>>> warputils = WarpUtils()
>>> warputils.inputs.in_file = "warpfield.nii"
>>> warputils.inputs.reference = "T1.nii"
>>> warputils.inputs.out_format = 'spline'
>>> warputils.inputs.warp_resolution = (10,10,10)
>>> warputils.inputs.output_type = "NIFTI_GZ"
>>> warputils.cmdline # doctest: +ELLIPSIS
'fnirtfileutils --in=warpfield.nii --outformat=spline --ref=T1.nii --warpres=10.0000,10.0000,10.0000 --out=warpfield_coeffs.nii.gz'
>>> res = invwarp.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
Name of file containing warp-coefficients/fields. This would
typically be the output from the --cout switch of fnirt (but can also use fields, like the output from --fout).
argument: `--in=%s`
reference: (a pathlike object or string representing an existing file)
Name of a file in target space. Note that the target space is now different from the target space that was used to create the --warp file. It would typically be the file that was specified with the --in argument when running fnirt.
argument: `--ref=%s`
write_jacobian: (a boolean, nipype default value: False)
Switch on --jac flag with automatically generated filename

[Optional]
out_format: ('spline' or 'field')
Specifies the output format. If set to field (default) the output will be a (4D) field-file. If set to spline the format will be a (4D) file of spline coefficients.
argument: `--outformat=%s`
warp_resolution: (a tuple of the form: (a float, a float, a float))
Specifies the resolution/knot-spacing of the splines pertaining to the coefficients in the --out file. This parameter is only relevant if --outformat is set to spline. It should be noted that if the --in file has a higher resolution, the resulting coefficients will pertain to the closest (in a least-squares sense) file in the space of fields with the --warpres resolution. It should also be noted that the resolution will always be an integer multiple of the voxel size.
argument: `--warpres=%0.4f,%0.4f,%0.4f`
knot_space: (a tuple of the form: (an integer (int or long), an integer (int or long), an integer (int or long)))
Alternative (to --warpres) specification of the resolution of the output spline-field.
argument: `--knotspace=%d,%d,%d`
out_file: (a pathlike object or string representing a file)
Name of output file. The format of the output depends on what other parameters are set. The default format is a (4D) field-file. If the --outformat is set to spline the format will be a (4D) file of spline coefficients.
argument: `--out=%s`, position: -1
out_jacobian: (a pathlike object or string representing a file)
Specifies that a (3D) file of Jacobian determinants corresponding to --in should be produced and written to filename.
argument: `--jac=%s`
with_affine: (a boolean)
Specifies that the affine transform (i.e. that which was specified for the --aff parameter in fnirt) should be included as displacements in the --out file. That can be useful for interfacing with software that cannot decode FSL/fnirt coefficient-files (where the affine transform is stored separately from the displacements).
argument: `--withaff`
output_type: ('NIFTI' or 'NIFTI_PAIR' or 'NIFTI_GZ' or 'NIFTI_PAIR_GZ')
FSL output type
args: (a unicode string)
Additional parameters to the command
argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

Environment variables

Outputs:

out_file: (a pathlike object or string representing a file)
Name of output file, containing the warp as field or coefficients.

out_jacobian: (a pathlike object or string representing a file)
Name of output file, containing the map of the determinant of the Jacobian

References:

None
64.1 interfaces.minc.base

64.1.1 aggregate_filename()

Link to code
Try to work out a sensible name given a set of files that have been combined in some way (e.g. averaged). If we
can’t work out a sensible prefix, we use the first filename in the list.

Examples

```python
>>> from nipype.interfaces.minc.base import aggregate_filename
>>> f = aggregate_filename(['/tmp/foo1.mnc', '/tmp/foo2.mnc', '/tmp/foo3.mnc'],
                          'averaged')
>>> os.path.split(f)[1] # This has a full path, so just check the filename.
'foo_averaged.mnc'

>>> f = aggregate_filename(['/tmp/foo1.mnc', '/tmp/blah1.mnc'], 'averaged')
>>> os.path.split(f)[1] # This has a full path, so just check the filename.
'foo1_averaged.mnc'
```

64.2 interfaces.minc.minc

64.2.1 Average

Link to code
Wraps the executable command mincaverage.
Average a number of MINC files.

Examples

```python
>>> from nipype.interfaces.minc import Average
>>> from nipype.interfaces.minc.testdata import nonempty_minc_data
```
```python
>>> files = [nonempty_minc_data(i) for i in range(3)]
>>> average = Average(input_files=files, output_file='/tmp/tmp.mnc')
>>> average.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
input_files: (a list of items which are a pathlike object or string representing an existing file)
input file(s)
argument: ``%s``, position: -2
mutually-exclusive: input_files, filelist

disable the use of an input file list
filelist: (a pathlike object or string representing an existing file)
Specify the name of a file containing input file names.
argument: ``-filelist %s``
mutually-exclusive: input_files, filelist

[Optional]
output_file: (a pathlike object or string representing a file)
output file
argument: ``%s``, position: -1
two: (a boolean)
Create a MINC 2 output file.
argument: ``-2``
clobber: (a boolean, nipype default value: True)
Overwrite existing file.
argument: ``-clobber``
verbose: (a boolean)
Print out log messages (default).
argument: ``-verbose``
mutually-exclusive: verbose, quiet
quiet: (a boolean)
Do not print out log messages.
argument: ``-quiet``
mutually-exclusive: verbose, quiet
debug: (a boolean)
Print out debugging messages.
argument: ``-debug``
check_dimensions: (a boolean)
Check that dimension info matches across files (default).
argument: ``-check_dimensions``
mutually-exclusive: check_dimensions, no_check_dimensions
no_check_dimensions: (a boolean)
Do not check dimension info.
argument: ``-nocheck_dimensions``
mutually-exclusive: check_dimensions, no_check_dimensions
format_filetype: (a boolean)
Use data type of first file (default).
argument: ``-filetype``
mutually-exclusive: format_filetype, format_byte, format_short,
format_int, format_long, format_float, format_double,
format_signed, format_unsigned
format_byte: (a boolean)
Write out byte data.
argument: ``-byte``
mutually-exclusive: format_filetype, format_byte, format_short,
format_int, format_long, format_float, format_double,
format_signed, format_unsigned

(continues on next page)
format_short: (a boolean)

Write out short integer data.

argument: `-short`

mutually_exclusive: format_filetype, format_byte, format_short, format_int, format_long, format_float, format_double, format_signed, format_unsigned

format_int: (a boolean)

Write out 32-bit integer data.

argument: `-int`

mutually_exclusive: format_filetype, format_byte, format_short, format_int, format_long, format_float, format_double, format_signed, format_unsigned

format_long: (a boolean)

Superseded by -int.

argument: `-long`

mutually_exclusive: format_filetype, format_byte, format_short, format_int, format_long, format_float, format_double, format_signed, format_unsigned

format_float: (a boolean)

Write out single-precision floating-point data.

argument: `-float`

mutually_exclusive: format_filetype, format_byte, format_short, format_int, format_long, format_float, format_double, format_signed, format_unsigned

format_double: (a boolean)

Write out double-precision floating-point data.

argument: `-double`

mutually_exclusive: format_filetype, format_byte, format_short, format_int, format_long, format_float, format_double, format_signed, format_unsigned

format_signed: (a boolean)

Write signed integer data.

argument: `-signed`

mutually_exclusive: format_filetype, format_byte, format_short, format_int, format_long, format_float, format_double, format_signed, format_unsigned

format_unsigned: (a boolean)

Write unsigned integer data (default).

argument: `-unsigned`

mutually_exclusive: format_filetype, format_byte, format_short, format_int, format_long, format_float, format_double, format_signed, format_unsigned

max_buffer_size_in_kb: (a long integer >= 0, nipype default value: 4096)

Specify the maximum size of the internal buffers (in kbytes).

argument: `-max_buffer_size_in_kb %d`

normalize: (a boolean)

Normalize data sets for mean intensity.

argument: `-normalize`

mutually_exclusive: normalize, nonormalize

nonormalize: (a boolean)

Do not normalize data sets (default).

argument: `-nonormalize`

mutually_exclusive: normalize, nonormalize

voxel_range: (a tuple of the form: (an integer (int or long), an integer (int or long)))

Valid range for output data.
argument: `'-range %d %d`'
sdfile: (a pathlike object or string representing a file)
    Specify an output sd file (default=none).
    argument: `'-sdfile %s`'
copy_header: (a boolean)
    Copy all of the header from the first file (default for one file).
    argument: `'-copy_header'`
    mutually_exclusive: copy_header, no_copy_header
no_copy_header: (a boolean)
    Do not copy all of the header from the first file (default for many
    files)).
    argument: `'-nocopy_header'`
    mutually_exclusive: copy_header, no_copy_header
avgdim: (a unicode string)
    Specify a dimension along which we wish to average.
    argument: `'-avgdim %s`'
binarize: (a boolean)
    Binarize the volume by looking for values in a given range.
    argument: `'-binarize'`
binrange: (a tuple of the form: (a float, a float))
    Specify a range for binarization. Default value: 1.79769e+308
    -1.79769e+308.
    argument: `'-binrange %s %s`'
binvalue: (a float)
    Specify a target value (+/- 0.5) forbinarization. Default value:
    -1.79769e+308
    argument: `'-binvalue %s`'
weights: (a list of items which are a unicode string)
    Specify weights for averaging ("<w1>,<w2>,...").
    argument: `'-weights %s`'
width_weighted: (a boolean)
    Weight by dimension widths when -avgdim is used.
    argument: `'-width_weighted'`
    requires: avgdim
args: (a unicode string)
    Additional parameters to the command
    argument: `'%s'`
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {}))
    Environment variables

Outputs:

output_file: (a pathlike object or string representing an existing
    file)
    output file

64.2.2 BBox

Link to code
Wraps the executable command mincbbox.
Determine a bounding box of image.
Examples

```python
>>> from nipype.interfaces.minc import BBox
>>> from nipype.interfaces.minc.testdata import nonempty_minc_data

>>> file0 = nonempty_minc_data(0)
>>> bbox = BBox(input_file=file0)
>>> bbox.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
input_file: (a pathlike object or string representing an existing file)
  - input file
    - argument: ``%s``, position: -2

[Optional]
output_file: (a pathlike object or string representing a file)
  - output file containing bounding box corners
threshold: (an integer (int or long))
  - VIO_Real value threshold for bounding box. Default value: 0.
    - argument: ``-threshold``
one_line: (a boolean)
  - Output on one line (default): start_x y z width_x y z
    - argument: ``-one_line``
    - mutually_exclusive: one_line, two_lines
two_lines: (a boolean)
  - Output on two lines: start_x y z
    - width_x y z
    - argument: ``-two_lines``
    - mutually_exclusive: one_line, two_lines
format_mincresample: (a boolean)
  - Output format for mincresample: (-step x y z -start x y z -nelements x y z
    - argument: ``-mincresample``
format_mincreshape: (a boolean)
  - Output format for mincreshape: (-start x, y, z -count dx, dy, dz
    - argument: ``-mincreshape``
format_minccrop: (a boolean)
  - Output format for minccrop: (-xlim x1 x2 -ylim y1 y2 -zlim z1 z2
    - argument: ``-minccrop``
out_file: (a pathlike object or string representing a file)
  - argument: ``> %s``, position: -1
args: (a unicode string)
  - Additional parameters to the command
    - argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  - Environment variables

Outputs:

output_file: (a pathlike object or string representing an existing file)
  - output file containing bounding box corners
## 64.2.3 Beast

**Link to code**

Wraps the executable command `mincbeast`.

Extract brain image using BEaST (Brain Extraction using non-local Segmentation Technique).

### Examples

```python
>>> from nipype.interfaces.minc import Beast
>>> from nipype.interfaces.minc.testdata import nonempty_minc_data

>>> file0 = nonempty_minc_data(0)
>>> beast = Beast(input_file=file0)
>>> beast.run()  # doctest: +SKIP
```

### Inputs:

[Mandatory]

- `library_dir`: (a pathlike object or string representing a directory)
  - library directory
    - argument: `'-%s'`, position: -3
- `input_file`: (a pathlike object or string representing a file)
  - input file
    - argument: `'-%s'`, position: -2

[Optional]

- `probability_map`: (a boolean)
  - Output the probability map instead of crisp mask.
    - argument: `'--probability'`
- `flip_images`: (a boolean)
  - Flip images around the mid-sagittal plane to increase patch count.
    - argument: `'--flip'`
- `load_moments`: (a boolean)
  - Do not calculate moments instead use precalculated library moments.
    - (for optimization purposes)
    - argument: `'--load_moments'`
- `fill_holes`: (a boolean)
  - Fill holes in the binary output.
    - argument: `'--fill'`
- `median_filter`: (a boolean)
  - Apply a median filter on the probability map.
    - argument: `'--median'`
- `nlm_filter`: (a boolean)
  - Apply an NLM filter on the probability map (experimental).
    - argument: `'--nlm_filter'`
- `clobber`: (a boolean, nipype default value: True)
  - Overwrite existing file.
    - argument: `'--clobber'`
- `configuration_file`: (a pathlike object or string representing a file)
  - Specify configuration file.
    - argument: `'--configuration %s'`
- `voxel_size`: (an integer (int or long), nipype default value: 4)
  - Specify voxel size for calculations (4, 2, or 1). Default value: 4.
  - Assumes no multiscale. Use configuration file for multiscale.
    - argument: `'--voxel_size %s'`
- `abspath`: (a boolean, nipype default value: True)
  - File paths in the library are absolute (default is relative to
    (continues on next page)
library root).
argument: `'-abspath'`

patch_size: (an integer (int or long), nipype default value: 1)
    Specify patch size for single scale approach. Default value: 1.
argument: `'-patch_size %s'`

search_area: (an integer (int or long), nipype default value: 2)
    Specify size of search area for single scale approach. Default value: 2.
argument: `'-search_area %s'`

certainty_level_alpha: (a float, nipype default value: 0.5)
    Specify confidence level Alpha. Default value: 0.5
argument: `'-alpha %s'`

smoothness_factor_beta: (a float, nipype default value: 0.5)
    Specify smoothness factor Beta. Default value: 0.25
argument: `'-beta %s'`

threshold_patch_selection: (a float, nipype default value: 0.95)
    Specify threshold for patch selection. Default value: 0.95
argument: `'-threshold %s'`

number_selected_images: (an integer (int or long), nipype default value: 20)
    Specify number of selected images. Default value: 20
argument: `'-selection_num %s'`

same_resolution: (a boolean)
    Output final mask with the same resolution as input file.
argument: `'-same_resolution'`

output_file: (a pathlike object or string representing a file)
    Output file
argument: `'%s'`, position: -1

args: (a unicode string)
    Additional parameters to the command
argument: `'%s'`

environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})  
Environment variables

Outputs:

output_file: (a pathlike object or string representing an existing
    file)
    output mask file

64.2.4 BestLinReg

Link to code
Wraps the executable command bestlinreg.
Hierachial linear fitting between two files.
The bestlinreg script is part of the EZminc package:
https://github.com/BIC-MNI/EZminc/blob/master/scripts/bestlinreg.pl

Examples

```python
>>> from nipype.interfaces.minc import BestLinReg
>>> from nipype.interfaces.minc.testdata import nonempty_minc_data
```
>>> input_file = nonempty_minc_data(0)
>>> target_file = nonempty_minc_data(1)
>>> linreg = BestLinReg(source=input_file, target=target_file)
>>> linreg.run() # doctest: +SKIP

Inputs:

**[Mandatory]**

- **source**: (a pathlike object or string representing an existing file)
  - source Minc file
  - argument: `'-%s'`, position: -4
- **target**: (a pathlike object or string representing an existing file)
  - target Minc file
  - argument: `'-%s'`, position: -3

**[Optional]**

- **output_xfm**: (a pathlike object or string representing a file)
  - output xfm file
  - argument: `'-%s'`, position: -2
- **output_mnc**: (a pathlike object or string representing a file)
  - output mnc file
  - argument: `'-%s'`, position: -1
- **verbose**: (a boolean)
  - Print out log messages. Default: False.
  - argument: `'-verbose'`
- **clobber**: (a boolean, nipype default value: True)
  - Overwrite existing file.
  - argument: `'-clobber'`
- **args**: (a unicode string)
  - Additional parameters to the command
  - argument: `'-%s'`
- **environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

- **output_xfm**: (a pathlike object or string representing an existing file)
  - output xfm file
- **output_mnc**: (a pathlike object or string representing an existing file)
  - output mnc file

### 64.2.5 BigAverage

**Link to code**

Wraps the executable command `mincbigaverage`. Average 1000's of MINC files in linear time.

`mincbigaverage` is designed to discretise the problem of averaging either a large number of input files or averaging a smaller number of large files. (>1GB each). There is also some code included to perform “robust” averaging in which only the most common features are kept via down-weighting outliers beyond a standard deviation.

One advantage of `mincbigaverage` is that it avoids issues around the number of possible open files in HDF/netCDF. In short if you have more than 100 files open at once while averaging things will slow down significantly.

`mincbigaverage` does this via a iterative approach to averaging files and is a direct drop in replacement for
mincaverage. That said not all the arguments of mincaverage are supported in mincbigaverage but they should be.
This tool is part of the minc-widgets package:

Examples

```python
>>> from nipype.interfaces.minc import BigAverage
>>> from nipype.interfaces.minc.testdata import nonempty_minc_data

>>> files = [nonempty_minc_data(i) for i in range(3)]
>>> average = BigAverage(input_files=files, output_float=True, robust=True)
>>> average.run() # doctest: +SKIP
```

Inputs:

- **[Mandatory]**
  - **input_files**: (a list of items which are a pathlike object or string representing an existing file)
    - input file(s)
      - argument: ````%s```, position: -2

- **[Optional]**
  - **output_file**: (a pathlike object or string representing a file)
    - output file
      - argument: ````%s```, position: -1
  - **verbose**: (a boolean)
    - Print out log messages. Default: False.
      - argument: ````--verbose```
  - **clobber**: (a boolean, nipype default value: True)
    - Overwrite existing file.
      - argument: ````--clobber```
  - **output_float**: (a boolean)
    - Output files with float precision.
      - argument: ````--float```
  - **robust**: (a boolean)
    - Perform robust averaging, features that are outside 1 standarddeviation from the mean are downweighted. Works well for noisydata with artifacts. see the --tmpdir option if you have alarge number of input files.
      - argument: ````--robust```
  - **tmpdir**: (a pathlike object or string representing a directory)
    - temporary files directory
      - argument: ````--tmpdir %s```
  - **sd_file**: (a pathlike object or string representing a file)
    - Place standard deviation image in specified file.
      - argument: ````--sdfile %s```
  - **args**: (a unicode string)
    - Additional parameters to the command
      - argument: ````%s```
  - **environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
output_file: (a pathlike object or string representing an existing file)
  output file
sd_file: (a pathlike object or string representing an existing file)
  standard deviation image

### 64.2.6 Blob

**Link to code**

Wraps the executable command `mincblob`.

Calculate blobs from minc deformation grids.

**Examples**

```python
>>> from nipype.interfaces.minc import Blob
>>> from nipype.interfaces.minc.testdata import minc2Dfile

>>> blob = Blob(input_file=minc2Dfile, output_file='/tmp/tmp.mnc', trace=True)
```

**Inputs:**

- **input_file**: (a pathlike object or string representing an existing file)
  - input file to blob
    - argument: `"%s"`, position: -2

- **output_file**: (a pathlike object or string representing a file)
  - output file
    - argument: `"%s"`, position: -1

- **trace**: (a boolean)
  - compute the trace (approximate growth and shrinkage) -- FAST
    - argument: `"-trace"`

- **determinant**: (a boolean)
  - compute the determinant (exact growth and shrinkage) -- SLOW
    - argument: `"-determinant"`

- **translation**: (a boolean)
  - compute translation (structure displacement)
    - argument: `"-translation"`

- **magnitude**: (a boolean)
  - compute the magnitude of the displacement vector
    - argument: `"-magnitude"`

- **args**: (a unicode string)
  - Additional parameters to the command
    - argument: `"%s"`

- **environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: `{}`)
  - Environment variables

**Outputs:**

- **output_file**: (a pathlike object or string representing an existing file)
  - output file
64.2.7 Blur

Link to code

Wraps the executable command `mincblur`. Convolves an input volume with a Gaussian blurring kernel of user-defined width. Optionally, the first partial derivatives and the gradient magnitude volume can be calculated.

Examples

```python
>>> from nipype.interfaces.minc import Blur
>>> from nipype.interfaces.minc.testdata import minc3Dfile

(1) Blur an input volume with a 6mm fwhm isotropic Gaussian blurring kernel:

```python
>>> blur = Blur(input_file=minc3Dfile, fwhm=6, output_file_base='/tmp/out_6')
>>> blur.run()  # doctest: +SKIP
```

mincblur will create /tmp/out_6_blur.mnc.

(2) Calculate the blurred and gradient magnitude data:

```python
>>> blur = Blur(input_file=minc3Dfile, fwhm=6, gradient=True, output_file_base='/tmp/out_6')
>>> blur.run()  # doctest: +SKIP
```

will create /tmp/out_6_blur.mnc and /tmp/out_6_dxyz.mnc.

(3) Calculate the blurred data, the partial derivative volumes and the gradient magnitude for the same data:

```python
>>> blur = Blur(input_file=minc3Dfile, fwhm=6, partial=True, output_file_base='/tmp/out_6')
>>> blur.run()  # doctest: +SKIP
```

will create /tmp/out_6_blur.mnc, /tmp/out_6_dx.mnc, /tmp/out_6_dy.mnc, /tmp/out_6_dz.mnc and /tmp/out_6_dxyz.mnc.

Inputs:

[Mandatory]

- **input_file**: (a pathlike object or string representing an existing file)

  - argument: `"%s"`, position: -2

- **fwhm**: (a float)

  - Full-width-half-maximum of gaussian kernel. Default value: 0.

  - argument: `"-fwhm %s"`

  - mutually_exclusive: fwhm, fwhm3d, standard_dev

- **standard_dev**: (a float)

  - Standard deviation of gaussian kernel. Default value: 0.

  - argument: `"-standarddev %s"`

  - mutually_exclusive: fwhm, fwhm3d, standard_dev

- **fwhm3d**: (a tuple of the form: (a float, a float, a float))

  - Full-width-half-maximum of gaussian kernel. Default value:

  - `-1.79769e+308 -1.79769e+308 -1.79769e+308`

  - argument: `"-3dfwhm %s %s %s"`

  - mutually_exclusive: fwhm, fwhm3d, standard_dev

[Optional]

- **output_file_base**: (a pathlike object or string representing a file)

  - argument: `"%s"`, position: -1

- **clobber**: (a boolean, nipype default value: True)
Overwrite existing file.
argument: `-clobber`

**gaussian:** (a boolean)
Use a gaussian smoothing kernel (default).
argument: `-gaussian`
mutually_exclusive: gaussian, rect

**rect:** (a boolean)
Use a rect (box) smoothing kernel.
argument: `-rect`
mutually_exclusive: gaussian, rect

**gradient:** (a boolean)
Create the gradient magnitude volume as well.
argument: `-gradient`

**partial:** (a boolean)
Create the partial derivative and gradient magnitude volumes as well.
argument: `-partial`

**no_apodize:** (a boolean)
Do not apodize the data before blurring.
argument: `-no_apodize`

**dimensions:** (3 or 1 or 2)
Number of dimensions to blur (either 1, 2 or 3). Default value: 3.
argument: `-dimensions %s`

**args:** (a unicode string)
Additional parameters to the command
argument: `%s`

**environ:** (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

<table>
<thead>
<tr>
<th>Outputs:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>output_file:</strong> (a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>Blurred output file.</td>
</tr>
<tr>
<td><strong>gradient_dxyz:</strong> (a pathlike object or string representing a file)</td>
</tr>
<tr>
<td>Gradient dxyz.</td>
</tr>
<tr>
<td><strong>partial_dx:</strong> (a pathlike object or string representing a file)</td>
</tr>
<tr>
<td>Partial gradient dx.</td>
</tr>
<tr>
<td><strong>partial_dy:</strong> (a pathlike object or string representing a file)</td>
</tr>
<tr>
<td>Partial gradient dy.</td>
</tr>
<tr>
<td><strong>partial_dz:</strong> (a pathlike object or string representing a file)</td>
</tr>
<tr>
<td>Partial gradient dz.</td>
</tr>
<tr>
<td><strong>partial_dxyz:</strong> (a pathlike object or string representing a file)</td>
</tr>
<tr>
<td>Partial gradient dxyz.</td>
</tr>
</tbody>
</table>

64.2.8 Calc

[Link to code](#)
Wraps the executable command `minccalc`.
Compute an expression using MINC files as input.
Examples

```python
>>> from nipype.interfaces.minc import Calc
>>> from nipype.interfaces.minc.testdata import nonempty_minc_data

>>> file0 = nonempty_minc_data(0)
>>> file1 = nonempty_minc_data(1)
>>> calc = Calc(input_files=[file0, file1], output_file='/tmp/calc.mnc',
              expression='A[0] + A[1]')  # add files together
>>> calc.run()  # doctest: +SKIP
```

Inputs:

- **input_files**: (a list of items which are a pathlike object or string representing an existing file)
  - input file(s) for calculation
  - argument: ``-%s``, position: -2
- **filelist**: (a pathlike object or string representing a file)
  - Specify the name of a file containing input file names.
  - argument: ``-filelist %s``
    - mutually_exclusive: input_files, filelist
- **expression**: (a unicode string)
  - Expression to use in calculations.
  - argument: ``-expression '%s'``
    - mutually_exclusive: expression, expfile
- **expfile**: (a pathlike object or string representing a file)
  - Name of file containing expression.
  - argument: ``-expfile %s``
    - mutually_exclusive: expression, expfile

(Optional)

- **output_file**: (a pathlike object or string representing a file)
  - output file
  - argument: ``-%s``, position: -1
- **two**: (a boolean)
  - Create a MINC 2 output file.
  - argument: ``-2``
- **clobber**: (a boolean, nipype default value: True)
  - Overwrite existing file.
  - argument: ``-clobber``
- **verbose**: (a boolean)
  - Print out log messages (default).
  - argument: ``-verbose``
    - mutually_exclusive: verbose, quiet
- **quiet**: (a boolean)
  - Do not print out log messages.
  - argument: ``-quiet``
    - mutually_exclusive: verbose, quiet
- **debug**: (a boolean)
  - Print out debugging messages.
  - argument: ``-debug``
- **copy_header**: (a boolean)
  - Copy all of the header from the first file.
  - argument: ``-copy_header``
    - mutually_exclusive: copy_header, no_copy_header
- **no_copy_header**: (a boolean)
  - Do not copy all of the header from the first file.
argument: `\`-nocopy_header`\`
mutually_exclusive: copy_header, no_copy_header

format_filetype: (a boolean)
Use data type of first file (default).
argument: `\`-filetype`\`
mutually_exclusive: format_filetype, format_byte, format_short,
format_int, format_long, format_float, format_double,
format_signed, format_unsigned

format_byte: (a boolean)
Write out byte data.
argument: `\`-byte`\`
mutually_exclusive: format_filetype, format_byte, format_short,
format_int, format_long, format_float, format_double,
format_signed, format_unsigned

format_short: (a boolean)
Write out short integer data.
argument: `\`-short`\`
mutually_exclusive: format_filetype, format_byte, format_short,
format_int, format_long, format_float, format_double,
format_signed, format_unsigned

format_int: (a boolean)
Write out 32-bit integer data.
argument: `\`-int`\`
mutually_exclusive: format_filetype, format_byte, format_short,
format_int, format_long, format_float, format_double,
format_signed, format_unsigned

format_long: (a boolean)
Superseded by -int.
argument: `\`-long`\`
mutually_exclusive: format_filetype, format_byte, format_short,
format_int, format_long, format_float, format_double,
format_signed, format_unsigned

format_float: (a boolean)
Write out single-precision floating-point data.
argument: `\`-float`\`
mutually_exclusive: format_filetype, format_byte, format_short,
format_int, format_long, format_float, format_double,
format_signed, format_unsigned

format_double: (a boolean)
Write out double-precision floating-point data.
argument: `\`-double`\`
mutually_exclusive: format_filetype, format_byte, format_short,
format_int, format_long, format_float, format_double,
format_signed, format_unsigned

format_signed: (a boolean)
Write signed integer data.
argument: `\`-signed`\`
mutually_exclusive: format_filetype, format_byte, format_short,
format_int, format_long, format_float, format_double,
format_signed, format_unsigned

format_unsigned: (a boolean)
Write unsigned integer data (default).
argument: `\`-unsigned`\`
mutually_exclusive: format_filetype, format_byte, format_short,
format_int, format_long, format_float, format_double,
format_signed, format_unsigned

voxel_range: (a tuple of the form: (an integer (int or long), an
integer (int or long))
Valid range for output data.
argument: `--range %d %d`

max_buffer_size_in_kb: (a long integer >= 0)
Specify the maximum size of the internal buffers (in kbytes).
argument: `--max_buffer_size_in_kb %d`

check_dimensions: (a boolean)
Check that files have matching dimensions (default).
argument: `--check_dimensions`
mutually_exclusive: check_dimensions, no_check_dimensions

no_check_dimensions: (a boolean)
Do not check that files have matching dimensions.
argument: `--nocheck_dimensions`
mutually_exclusive: check_dimensions, no_check_dimensions

ignore_nan: (a boolean)
Ignore invalid data (NaN) for accumulations.
argument: `--ignore_nan`

propagate_nan: (a boolean)
Invalid data in any file at a voxel produces a NaN (default).
argument: `--propagate_nan`

output_nan: (a boolean)
Output NaN when an illegal operation is done (default).
argument: `--nan`
mutually_exclusive: output_nan, output_zero, output_illegal_value

output_zero: (a boolean)
Output zero when an illegal operation is done.
argument: `--zero`
mutually_exclusive: output_nan, output_zero, output_illegal_value

output_illegal: (a boolean)
Value to write out when an illegal operation is done. Default value: 1.79769e+308
argument: `--illegal_value`
mutually_exclusive: output_nan, output_zero, output_illegal_value

outfiles: (a list of items which are a tuple of the form: (a unicode string, a pathlike object or string representing a file))

eval_width: (an integer (int or long))
Number of voxels to evaluate simultaneously.
argument: `--eval_width %s`

args: (a unicode string)
Additional parameters to the command
argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

output_file: (a pathlike object or string representing an existing file)
output file

64.2.9 Convert

Link to code
Wraps the executable command mincconvert.
convert between MINC 1 to MINC 2 format.
### Examples

```python
>>> from nipype.interfaces.minc import Convert
>>> from nipype.interfaces.minc.testdata import minc2Dfile

```c = Convert(input_file=minc2Dfile, output_file='/tmp/out.mnc', two=True)
# Convert to MINC2 format.
```c.run() # doctest: +SKIP

**Inputs:**

[Mandatory]
- `input_file`: (a pathlike object or string representing an existing file)
  - input file for converting
  - argument: `'\%s'`, position: -2

[Optional]
- `output_file`: (a pathlike object or string representing a file)
  - output file
  - argument: `'\%s'`, position: -1
- `clobber`: (a boolean, nipype default value: True)
  - Overwrite existing file.
  - argument: `'\-clobber'`
- `two`: (a boolean)
  - Create a MINC 2 output file.
  - argument: `'\-2'`
- `template`: (a boolean)
  - Create a template file. The dimensions, variables, and attributes of the input file are preserved but all data is set to zero.
  - argument: `'\-template'`
- `compression`: (0 or 1 or 2 or 3 or 4 or 5 or 6 or 7 or 8 or 9)
  - Set the compression level, from 0 (disabled) to 9 (maximum).
  - argument: `'\-compress %s'`
- `chunk`: (a long integer >= 0)
  - Set the target block size for chunking (0 default, >1 block size).
  - argument: `'\-chunk %d'`
- `args`: (a unicode string)
  - Additional parameters to the command
  - argument: `'\%s'`
- `environ`: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})”

**Outputs:**

- `output_file`: (a pathlike object or string representing an existing file)
  - output file

### 64.2.10 Copy

**Link to code**

Wraps the executable command `minccopy`.

Copy image values from one MINC file to another. Both the input and output files must exist, and the images in both files must have an equal number dimensions and equal dimension lengths.

**NOTE:** This program is intended primarily for use with scripts such as mincedit. It does not follow the typical design rules of most MINC command-line tools and therefore should be used only with caution.
Inputs:

[Mandatory]
input_file: (a pathlike object or string representing an existing file)
  input file to copy
  argument: `%s`, position: -2

[Optional]
output_file: (a pathlike object or string representing a file)
  output file
  argument: `%s`, position: -1

pixel_values: (a boolean)
  Copy pixel values as is.
  argument: `-pixel_values`
  mutually exclusive: pixel_values, real_values

real_values: (a boolean)
  Copy real pixel intensities (default).
  argument: `-real_values`
  mutually exclusive: pixel_values, real_values

args: (a unicode string)
  Additional parameters to the command
  argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})  
  Environment variables

Outputs:

output_file: (a pathlike object or string representing an existing file)
  output file

64.2.11 Dump

Link to code
Wraps the executable command mincdump.
Dump a MINC file. Typically used in conjunction with mincgen (see Gen).

Examples

```python
>>> from nipype.interfaces.minc import Dump
>>> from nipype.interfaces.minc.testdata import minc2Dfile

>>> dump = Dump(input_file=minc2Dfile)
>>> dump.run() # doctest: +SKIP

>>> dump = Dump(input_file=minc2Dfile, output_file='/tmp/out.txt', precision=(3,4))
>>> dump.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
input_file: (a pathlike object or string representing an existing file)
file)
input file
argument: `\%s`, position: -2

[Optional]

output_file: (a pathlike object or string representing a file)
output file

coordinate_data: (a boolean)
Coordinate variable data and header information.
argument: `-c`
mutually_exclusive: coordinate_data, header_data

header_data: (a boolean)
Header information only, no data.
argument: `-h`
mutually_exclusive: coordinate_data, header_data

annotations_brief: ('c' or 'f')
Brief annotations for C or Fortran indices in data.
argument: `-b %s`
mutually_exclusive: annotations_brief, annotations_full

annotations_full: ('c' or 'f')
Full annotations for C or Fortran indices in data.
argument: `-f %s`
mutually_exclusive: annotations_brief, annotations_full

variables: (a list of items which are a unicode string)
Output data for specified variables only.
argument: `-v %s`

line_length: (a long integer >= 0)
Line length maximum in data section (default 80).
argument: `-l %d`

netcdf_name: (a unicode string)
Name for netCDF (default derived from file name).
argument: `-n %s`

precision: (an integer (int or long) or a tuple of the form: (an integer (int or long), an integer (int or long)))
Display floating-point values with less precision
argument: `\%s`

out_file: (a pathlike object or string representing a file)
argument: `> %s`, position: -1

args: (a unicode string)
Additional parameters to the command
argument: `\%s`

eviron: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

output_file: (a pathlike object or string representing an existing file)
output file

---

64.2.12 Extract

Link to code
Wraps the executable command `mincextract`.
Dump a hyperslab of MINC file data.
Examples

```python
>>> from nipype.interfaces.minc import Extract
>>> from nipype.interfaces.minc.testdata import minc2Dfile

>>> extract = Extract(input_file=minc2Dfile)
>>> extract.run()  # doctest: +SKIP

>>> extract = Extract(input_file=minc2Dfile, start=[3, 10, 5], count=[4, 4, 4])  # doctest: +SKIP
--> extract a 4x4x4 slab at offset [3, 10, 5]
>>> extract.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
input_file: (a pathlike object or string representing an existing file)
  input file
  argument: ``-%s``, position: -2

[Optional]
output_file: (a pathlike object or string representing a file)
  output file
write_ascii: (a boolean)
  Write out data as ascii strings (default).
  argument: ``-ascii``
  mutually_exclusive: write_ascii, write_ascii, write_byte,
    write_short, write_int, write_long, write_float, write_double,
    write_signed, write_unsigned
write_byte: (a boolean)
  Write out data as bytes.
  argument: ``-byte``
  mutually_exclusive: write_ascii, write_ascii, write_byte,
    write_short, write_int, write_long, write_float, write_double,
    write_signed, write_unsigned
write_short: (a boolean)
  Write out data as short integers.
  argument: ``-short``
  mutually_exclusive: write_ascii, write_ascii, write_byte,
    write_short, write_int, write_long, write_float, write_double,
    write_signed, write_unsigned
write_int: (a boolean)
  Write out data as 32-bit integers.
  argument: ``-int``
  mutually_exclusive: write_ascii, write_ascii, write_byte,
    write_short, write_int, write_long, write_float, write_double,
    write_signed, write_unsigned
write_long: (a boolean)
  Superseded by write_int.
  argument: ``-long``
  mutually_exclusive: write_ascii, write_ascii, write_byte,
    write_short, write_int, write_long, write_float, write_double,
    write_signed, write_unsigned
write_float: (a boolean)
  Write out data as single precision floating-point values.
  argument: ``-float``
  mutually_exclusive: write_ascii, write_ascii, write_byte,
(continues on next page)
write_short, write_int, write_long, write_float, write_double,
write_signed, write_unsigned
write_double: (a boolean)
Write out data as double precision floating-point values.
argument: `--double`
mutually_exclusive: write_ascii, write_ascii, write_byte,
write_short, write_int, write_long, write_float, write_double,
write_signed, write_unsigned
write_signed: (a boolean)
Write out signed data.
argument: `--signed`
mutually_exclusive: write_signed, write_unsigned
write_unsigned: (a boolean)
Write out unsigned data.
argument: `--unsigned`
mutually_exclusive: write_signed, write_unsigned
write_range: (a tuple of the form: (a float, a float))
Specify the range of output values
Default value: 1.79769e+308 1.79769e+308.
argument: `--range %s %s`
normalize: (a boolean)
Normalize integer pixel values to file max and min.
argument: `--normalize`
mutually_exclusive: normalize, nonormalize
nonormalize: (a boolean)
Turn off pixel normalization.
argument: `--nonormalize`
mutually_exclusive: normalize, nonormalize
image_range: (a tuple of the form: (a float, a float))
Specify the range of real image values for normalization.
argument: `--image_range %s %s`
image_minimum: (a float)
Specify the minimum real image value for normalization.Default value: 1.79769e+308.
argument: `--image_minimum %s`
image_maximum: (a float)
Specify the maximum real image value for normalization.Default value: 1.79769e+308.
argument: `--image_maximum %s`
start: (a list of items which are an integer (int or long))
Specifies corner of hyperslab (C conventions for indices).
argument: `--start %s`
count: (a list of items which are an integer (int or long))
Specifies edge lengths of hyperslab to read.
argument: `--count %s`
flip_positive_direction: (a boolean)
Flip images to always have positive direction.
argument: `--positive_direction`
mutually_exclusive: flip_positive_direction,
flip_negative_direction, flip_any_direction
flip_negative_direction: (a boolean)
Flip images to always have negative direction.
argument: `--negative_direction`
mutually_exclusive: flip_positive_direction,
flip_negative_direction, flip_any_direction
flip_any_direction: (a boolean)
Do not flip images (Default).
argument: `'-any_direction'`
mutually_exclusive: flip_positive_direction, flip_negative_direction, flip_any_direction
flip_x_positive: (a boolean)
Flip images to give positive xspace:step value (left-to-right).
argument: `'+xdirection'`
mutually_exclusive: flip_x_positive, flip_x_negative, flip_x_any
flip_x_negative: (a boolean)
Flip images to give negative xspace:step value (right-to-left).
argument: `'-xdirection'`
mutually_exclusive: flip_x_positive, flip_x_negative, flip_x_any
flip_x_any: (a boolean)
Don't flip images along x-axis (default).
argument: `'-xanydirection'`
mutually_exclusive: flip_x_positive, flip_x_negative, flip_x_any
flip_y_positive: (a boolean)
Flip images to give positive yspace:step value (post-to-ant).
argument: `'+ydirection'`
mutually_exclusive: flip_y_positive, flip_y_negative, flip_y_any
flip_y_negative: (a boolean)
Flip images to give negative yspace:step value (ant-to-post).
argument: `'-ydirection'`
mutually_exclusive: flip_y_positive, flip_y_negative, flip_y_any
flip_y_any: (a boolean)
Don't flip images along y-axis (default).
argument: `'-yanydirection'`
mutually_exclusive: flip_y_positive, flip_y_negative, flip_y_any
flip_z_positive: (a boolean)
Flip images to give positive zspace:step value (inf-to-sup).
argument: `'+zdirection'`
mutually_exclusive: flip_z_positive, flip_z_negative, flip_z_any
flip_z_negative: (a boolean)
Flip images to give negative zspace:step value (sup-to-inf).
argument: `'-zdirection'`
mutually_exclusive: flip_z_positive, flip_z_negative, flip_z_any
flip_z_any: (a boolean)
Don't flip images along z-axis (default).
argument: `'-zanydirection'`
mutually_exclusive: flip_z_positive, flip_z_negative, flip_z_any
out_file: (a pathlike object or string representing a file)
argument: `'> %s'`, position: -1
args: (a unicode string)
Additional parameters to the command
argument: `'%s'`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

output_file: (a pathlike object or string representing an existing file)
output file in raw/text format
64.2.13 Gennlxfm

Link to code
Wraps the executable command gennlxfm. Generate nonlinear xfms. Currently only identity xfms are supported!
This tool is part of minc-widgets:
https://github.com/BIC-MNI/minc-widgets/blob/master/gennlxfm/gennlxfm

Examples

```python
>>> from nipype.interfaces.minc import Gennlxfm
>>> from nipype.interfaces.minc.testdata import minc2Dfile
>>> gennlxfm = Gennlxfm(step=1, like=minc2Dfile)
>>> gennlxfm.run() # doctest: +SKIP
```

Inputs:

- `output_file`: (a pathlike object or string representing a file)
  - `output file` argument: ``%s``, position: -1
- `verbose`: (a boolean)
  - Print out log messages. Default: False.
  - `verbose` argument: ``-verbose``
- `clobber`: (a boolean, nipype default value: True)
  - Overwrite existing file.
  - `clobber` argument: ``-clobber``
- `ident`: (a boolean)
  - `ident` argument: ``-ident``
- `step`: (an integer (int or long))
  - Output ident xfm step [default: 1].
  - `step` argument: ``-step %s``
- `like`: (a pathlike object or string representing an existing file)
  - Generate a nlxfm like this file.
  - `like` argument: ``-like %s``
- `args`: (a unicode string)
  - Additional parameters to the command
  - `args` argument: ``%s``
- `environ`: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

- `output_file`: (a pathlike object or string representing an existing file)
  - `output file`
- `output_grid`: (a pathlike object or string representing an existing file)
  - `output grid`

64.2.14 Math

Link to code
Wraps the executable command mincmath. Various mathematical operations supplied by mincmath.
Examples

```python
>>> from nipype.interfaces.minc import Math
>>> from nipype.interfaces.minc.testdata import minc2Dfile

Scale: volume*3.0 + 2:

```python
>>> scale = Math(input_files=[minc2Dfile], scale=(3.0, 2))
``` python
>>> scale.run() # doctest: +SKIP

Test if >= 1.5:

```python
>>> gt = Math(input_files=[minc2Dfile], test_gt=1.5)
``` python
>>> gt.run() # doctest: +SKIP

Inputs:

[Mandatory]
input_files: (a list of items which are a pathlike object or string representing an existing file)
input file(s) for calculation
argument: ``%s``, position: -2
mutually_exclusive: input_files, filelist

filelist: (a pathlike object or string representing an existing file)
Specify the name of a file containing input file names.
argument: ``-filelist %s``
mutually_exclusive: input_files, filelist

[Optional]
output_file: (a pathlike object or string representing a file)
output file
argument: ``%s``, position: -1
clobber: (a boolean, nipype default value: True)
Overwrite existing file.
argument: ``-clobber``
two: (a boolean)
Create a MINC 2 output file.
argument: ``-2``
copy_header: (a boolean)
Copy all of the header from the first file (default for one file).
argument: ``-copy_header``
mutually_exclusive: copy_header, no_copy_header
no_copy_header: (a boolean)
Do not copy all of the header from the first file (default for many files)).
argument: ``-nocopy_header``
mutually_exclusive: copy_header, no_copy_header

format_filetype: (a boolean)
Use data type of first file (default).
argument: ``-filetype``
mutually_exclusive: format_filetype, format_byte, format_short, format_int, format_long, format_float, format_double, format_signed, format_unsigned

format_byte: (a boolean)
Write out byte data.
argument: ``-byte``
mutually_exclusive: format_filetype, format_byte, format_short, format_int, format_long, format_float, format_double, format_signed, format_unsigned

(continues on next page)
format_short: (a boolean)
  Write out short integer data.
  argument: `--short`
  mutually_exclusive: format_filetype, format_byte, format_short,
  format_int, format_long, format_float, format_double,
  format_signed, format_unsigned
format_int: (a boolean)
  Write out 32-bit integer data.
  argument: `--int`
  mutually_exclusive: format_filetype, format_byte, format_short,
  format_int, format_long, format_float, format_double,
  format_signed, format_unsigned
format_long: (a boolean)
  Superseded by --int.
  argument: `--long`
  mutually_exclusive: format_filetype, format_byte, format_short,
  format_int, format_long, format_float, format_double,
  format_signed, format_unsigned
format_float: (a boolean)
  Write out single-precision floating-point data.
  argument: `--float`
  mutually_exclusive: format_filetype, format_byte, format_short,
  format_int, format_long, format_float, format_double,
  format_signed, format_unsigned
format_double: (a boolean)
  Write out double-precision floating-point data.
  argument: `--double`
  mutually_exclusive: format_filetype, format_byte, format_short,
  format_int, format_long, format_float, format_double,
  format_signed, format_unsigned
format_signed: (a boolean)
  Write signed integer data.
  argument: `--signed`
  mutually_exclusive: format_filetype, format_byte, format_short,
  format_int, format_long, format_float, format_double,
  format_signed, format_unsigned
format_unsigned: (a boolean)
  Write unsigned integer data (default).
  argument: `--unsigned`
  mutually_exclusive: format_filetype, format_byte, format_short,
  format_int, format_long, format_float, format_double,
  format_signed, format_unsigned
voxel_range: (a tuple of the form: (an integer (int or long), an
integer (int or long)))
  Valid range for output data.
  argument: `--range %d %d`
max_buffer_size_in_kb: (a long integer >= 0, nipype default value: 4096)
  Specify the maximum size of the internal buffers (in kbytes).
  argument: `--max_buffer_size_in_kb %d`
check_dimensions: (a boolean)
  Check that dimension info matches across files (default).
  argument: `--check_dimensions`
  mutually_exclusive: check_dimensions, no_check_dimensions
no_check_dimensions: (a boolean)
  Do not check dimension info.
  argument: `--nocheck_dimensions`
mutually_exclusive: check_dimensions, no_check_dimensions
dimension: (a unicode string)
    Specify a dimension along which we wish to perform a calculation.
    argument: "-dimension %s"
ignore_nan: (a boolean)
    Ignore invalid data (NaN) for accumulations.
    argument: "-ignore_nan"
propagate_nan: (a boolean)
    Invalid data in any file at a voxel produces a NaN (default).
    argument: "-propagate_nan"
output_nan: (a boolean)
    Output NaN when an illegal operation is done (default).
    argument: "-nan"
mutually_exclusive: output_nan, output_zero, output_illegal_value
output_zero: (a boolean)
    Output zero when an illegal operation is done.
    argument: "-zero"
mutually_exclusive: output_nan, output_zero, output_illegal_value
output_illegal: (a boolean)
    Value to write out when an illegal operation is done. Default value: 1.79769e+308
    argument: "-illegal_value"
mutually_exclusive: output_nan, output_zero, output_illegal_value
test_gt: (a boolean or a float)
    Test for vol1 > vol2 or vol1 > constant.
    argument: "-gt"
test_lt: (a boolean or a float)
    Test for vol1 < vol2 or vol1 < constant.
    argument: "-lt"
test_eq: (a boolean or a float)
    Test for integer vol1 == vol2 or vol1 == constant.
    argument: "-eq"
test_ne: (a boolean or a float)
    Test for integer vol1 != vol2 or vol1 != const.
    argument: "-ne"
test_ge: (a boolean or a float)
    Test for vol1 >= vol2 or vol1 >= const.
    argument: "-ge"
test_le: (a boolean or a float)
    Test for vol1 <= vol2 or vol1 <= const.
    argument: "-le"
calc_add: (a boolean or a float)
    Add N volumes or volume + constant.
    argument: "-add"
calc_sub: (a boolean or a float)
    Subtract 2 volumes or volume - constant.
    argument: "-sub"
calc_mul: (a boolean or a float)
    Multiply N volumes or volume * constant.
    argument: "-mul"
calc_div: (a boolean or a float)
    Divide 2 volumes or volume / constant.
    argument: "-div"
invert: (a float)
    Calculate 1/c.
    argument: "-invert -const %s"
Calculate \( V \).
  argument: `\'-not\'`

\texttt{sqrt}: (a boolean)
  Take square root of a volume.
  argument: `\'-sqrt\'`

\texttt{square}: (a boolean)
  Take square of a volume.
  argument: `\'-square\'`

\texttt{abs}: (a boolean)
  Take absolute value of a volume.
  argument: `\'-abs\'`

\texttt{exp}: (a tuple of the form: (a float, a float))
  Calculate \( c_2 \cdot \exp(c_1 \cdot x) \). Both constants must be specified.
  argument: `\'-exp -\text{\texttt{const2}} %s %s\'`

\texttt{log}: (a tuple of the form: (a float, a float))
  Calculate \( \log(x/c_2)/c_1 \). The constants \( c_1 \) and \( c_2 \) default to 1.
  argument: `\'-log -\text{\texttt{const2}} %s %s\'`

\texttt{scale}: (a tuple of the form: (a float, a float))
  Scale a volume: \( volume \cdot c_1 + c_2 \).
  argument: `\'-scale -\text{\texttt{const2}} %s %s\'`

\texttt{clamp}: (a tuple of the form: (a float, a float))
  Clamp a volume to lie between two values.
  argument: `\'-clamp -\text{\texttt{const2}} %s %s\'`

\texttt{segment}: (a tuple of the form: (a float, a float))
  Segment a volume using range of \( -\text{\texttt{const2}} \): within range = 1, outside range = 0.
  argument: `\'-segment -\text{\texttt{const2}} %s %s\'`

\texttt{nsegment}: (a tuple of the form: (a float, a float))
  Opposite of \( -\text{\texttt{segment}} \): within range = 0, outside range = 1.
  argument: `\'-nsegment -\text{\texttt{const2}} %s %s\'`

\texttt{isnan}: (a boolean)
  Test for NaN values in \( V \).
  argument: `\'-isnan\'`

\texttt{nisnan}: (a boolean)
  Negation of \( -\text{\texttt{isnan}} \).
  argument: `\'-nisnan\'`

\texttt{percentdiff}: (a float)
  Percent difference between 2 volumes, thresholded (const def=0.0).
  argument: `\'-percentdiff\'`

\texttt{count_valid}: (a boolean)
  Count the number of valid values in \( N \) volumes.
  argument: `\'-count_valid\'`

\texttt{maximum}: (a boolean)
  Find maximum of \( N \) volumes.
  argument: `\'-maximum\'`

\texttt{minimum}: (a boolean)
  Find minimum of \( N \) volumes.
  argument: `\'-minimum\'`

\texttt{calc_and}: (a boolean)
  Calculate \( V_{\text{\texttt{1}}} \&& V_{\text{\texttt{2}}} \&& \ldots \).
  argument: `\'-\text{\texttt{and}}\'`

\texttt{calc_or}: (a boolean)
  Calculate \( V_{\text{\texttt{1}}} \|| V_{\text{\texttt{2}}} || \ldots \).
  argument: `\'-\text{\texttt{or}}\'`

\texttt{args}: (a unicode string)
  Additional parameters to the command
  argument: `\'%(s)'}
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

output_file: (a pathlike object or string representing an existing file) output file

64.2.15 NlpFit

Link to code
Wraps the executable command nlpfit.
Hierarchial non-linear fitting with bluring.
This tool is part of the minc-widgets package:
https://github.com/BIC-MNI/minc-widgets/blob/master/nlpfit/nlpfit

Examples

```python
>>> from nipype.interfaces.minc import NlpFit
>>> from nipype.interfaces.minc.testdata import nonempty_minc_data, nlp_config
>>> from nipype.testing import example_data

>>> source = nonempty_minc_data(0)
>>> target = nonempty_minc_data(1)
>>> source_mask = nonempty_minc_data(2)
>>> config = nlp_config
>>> initial = example_data('minc_initial.xfm')
>>> nlpfit = NlpFit(config_file=config, init_xfm=initial, source_mask=source_mask,
... source=source, target=target)
>>> nlpfit.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
source: (a pathlike object or string representing an existing file)
  source Minc file
  argument: ``%s``, position: -3

target: (a pathlike object or string representing an existing file)
  target Minc file
  argument: ``%s``, position: -2

cfg
  config_file: (a pathlike object or string representing an existing file)
    File containing the fitting configuration use.
    argument: ``--config_file %s``

init_xfm: (a pathlike object or string representing an existing file)
  Initial transformation (default identity).
  argument: ``--init_xfm %s``

source_mask: (a pathlike object or string representing an existing file)
  Source mask to use during fitting.
  argument: ``--source_mask %s``

(continues on next page)
64.2.16 Norm

Link to code

Wraps the executable command mincnorm.

Normalise a file between a max and minimum (possibly) using two histogram pct's.

Examples

```python
>>> from nipype.interfaces.minc import Norm
>>> from nipype.interfaces.minc.testdata import minc2Dfile
>>> n = Norm(input_file=minc2Dfile, output_file='/tmp/out.mnc')  # Normalise the file.
>>> n.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
input_file: (a pathlike object or string representing an existing file)
  input file to normalise
  argument: ```%s``', position: -2

[Optional]
output_file: (a pathlike object or string representing a file)
  output file
  argument: ```%s``', position: -1
output_threshold_mask: (a pathlike object or string representing a file)
    File in which to store the threshold mask.
    argument: `'-threshold_mask %s'`
clobber: (a boolean, nipype default value: True)
    Overwrite existing file.
    argument: `'-clobber'`
mask: (a pathlike object or string representing an existing file)
    Calculate the image normalisation within a mask.
    argument: `'-mask %s'`
clobber: (a boolean, nipype default value: True)
    Force the output range between limits [default].
    argument: `'-clobber'`
cutoff: (0.0 <= a floating point number <= 100.0)
    Cutoff value to use to calculate thresholds by a histogram PCT in %.
    [default: 0.01]
    argument: `'-cutoff %s'`
lower: (a float)
    Lower real value to use.
    argument: `'-lower %s'`
upper: (a float)
    Upper real value to use.
    argument: `'-upper %s'`
out_floor: (a float)
    Output files maximum [default: 0]
    argument: `'-out_floor %s'`
out_ceil: (a float)
    Output files minimum [default: 100]
    argument: `'-out_ceil %s'`
threshold: (a boolean)
    Threshold the image (set values below threshold_perc to -out_floor).
    argument: `'-threshold'`
threshold_perc: (0.0 <= a floating point number <= 100.0)
    Threshold percentage (0.1 == lower 10% of intensity range) [default: 0.1].
    argument: `'-threshold_perc %s'`
threshold_bmt: (a boolean)
    Use the resulting image BiModalT as the threshold.
    argument: `'-threshold_bmt'`
threshold_blur: (a float)
    Blur FWHM for intensity edges then thresholding [default: 2].
    argument: `'-threshold_blur %s'`
args: (a unicode string)
    Additional parameters to the command
    argument: `'%s'`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str'
    and with values which are a bytes or None or a value of class 'str', nipype default value: {}) 
    Environment variables

Outputs:

output_file: (a pathlike object or string representing an existing file)
    output file
output_threshold_mask: (a pathlike object or string representing a file)
64.2.17 Pik

Link to code
Wraps the executable command mincpik.
Generate images from minc files.
Mincpik uses Imagemagick to generate images from Minc files.

Examples

```python
>>> from nipype.interfaces.minc import Pik
>>> from nipype.interfaces.minc.testdata import nonempty_minc_data

>>> file0 = nonempty_minc_data(0)
>>> pik = Pik(input_file=file0, title='foo')
>>> pik.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
input_file: (a pathlike object or string representing an existing file)
- input file
- argument: ``%s``, position: -2

[Optional]
jpg: (a boolean)
- Output a jpg file.
- mutually_exclusive: jpg, png

png: (a boolean)
- Output a png file (default).
- mutually_exclusive: jpg, png

output_file: (a pathlike object or string representing a file)
- output file
- argument: ``%s``, position: -1

clobber: (a boolean, nipype default value: True)
- Overwrite existing file.
- argument: ``-clobber``

scale: (an integer (int or long), nipype default value: 2)
- Scaling factor for resulting image. By default images are output at twice their original resolution.
- argument: ``--scale %s``

width: (an integer (int or long))
- Autoscale the resulting image to have a fixed image width (in pixels).
- argument: ``--width %s``

depth: (8 or 16)
- Bitdepth for resulting image 8 or 16 (MSB machines only!)
- argument: ``--depth %s``

title: (a boolean or a unicode string)
- argument: ``%s``

title_size: (an integer (int or long))
- Font point size for the title.
- argument: ``--title_size %s``
requires: title
annotated_bar: (a boolean)
    create an annotated bar to match the image (use height of the output
    image)
    argument: `--anot_bar`
minc_range: (a tuple of the form: (a float, a float))
    Valid range of values for MINC file.
    argument: `--range %s %s`
image_range: (a tuple of the form: (a float, a float))
    Range of image values to use for pixel intensity.
    argument: `--image_range %s %s`
    mutually_exclusive: image_range, auto_range
auto_range: (a boolean)
    Automatically determine image range using a 5 and 95% PcT.
    (histogram)
    argument: `--auto_range`
    mutually_exclusive: image_range, auto_range
start: (an integer (int or long))
    Slice number to get. (note this is in voxel co-ordinates).
    argument: `--slice %s`
slice_z: (a boolean)
    Get an axial/transverse (z) slice.
    argument: `--z`
    mutually_exclusive: slice_z, slice_y, slice_x
slice_y: (a boolean)
    Get a coronal (y) slice.
    argument: `--y`
    mutually_exclusive: slice_z, slice_y, slice_x
slice_x: (a boolean)
    Get a sagittal (x) slice.
    argument: `--x`
    mutually_exclusive: slice_z, slice_y, slice_x
triplanar: (a boolean)
    Create a triplanar view of the input file.
    argument: `--triplanar`
tile_size: (an integer (int or long))
    Pixel size for each image in a triplanar.
    argument: `--tilesize %s`
sagittal_offset: (an integer (int or long))
    Offset the sagittal slice from the centre.
    argument: `--sagittal_offset %s`
sagittal_offset_perc: (0 <= a long integer <= 100)
    Offset the sagittal slice by a percentage from the centre.
    argument: `--sagittal_offset_perc %d`
vertical_triplanar_view: (a boolean)
    Create a vertical triplanar view (Default).
    argument: `--vertical`
    mutually_exclusive: vertical_triplanar_view,
    horizontal_triplanar_view
horizontal_triplanar_view: (a boolean)
    Create a horizontal triplanar view.
    argument: `--horizontal`
    mutually_exclusive: vertical_triplanar_view,
    horizontal_triplanar_view
lookup: (a unicode string)
    Arguments to pass to minclookup
    argument: `--lookup %s`
64.2.18 Resample

Link to code
Wraps the executable command mincresample.
Resample a minc file.

Examples

```python
>>> from nipype.interfaces.minc import Resample
>>> from nipype.interfaces.minc.testdata import minc2Dfile
>>> r = Resample(input_file=minc2Dfile, output_file='/tmp/out.mnc') # Resample
    the file.
>>> r.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
input_file: (a pathlike object or string representing an existing file)
    input file for resampling
    argument: `-%s`, position: -2

[Optional]
output_file: (a pathlike object or string representing a file)
    output file
    argument: `-%s`, position: -1
input_grid_files: (a list of items which are a pathlike object or string representing a file)
    input grid file(s)
two: (a boolean)
    Create a MINC 2 output file.
    argument: `--2`
clobber: (a boolean, nipype default value: True)
    Overwrite existing file.
    argument: `--clobber`
trilinear_interpolation: (a boolean)
    Do trilinear interpolation.
    argument: `--trilinear`
mutually_exclusive: trilinear_interpolation, tricubic_interpolation,
    nearest_neighbour_interpolation, sinc_interpolation
tricubic_interpolation: (a boolean)
    Do tricubic interpolation.
    argument: `--tricubic`
```
argument: `'-tricubic'`
  mutually_exclusive: trilinear_interpolation, tricubic_interpolation,
  nearest_neighbour_interpolation, sinc_interpolation


half_width_sinc_window: (5 or 1 or 2 or 3 or 4 or 6 or 7 or 8 or 9 or 10)
  Set half-width of sinc window (1-10). Default value: 5.
  argument: `'-width %s'`
  requires: sinc_interpolation

sinc_window_hanning: (a boolean)
  Set sinc window type to Hanning.
  argument: `'-hanning'`
  mutually_exclusive: sinc_window_hanning, sinc_window_hamming
  requires: sinc_interpolation

sinc_window_hamming: (a boolean)
  Set sinc window type to Hamming.
  argument: `'-hamming'`
  mutually_exclusive: sinc_window_hanning, sinc_window_hamming
  requires: sinc_interpolation

transformation: (a pathlike object or string representing an existing file)
  File giving world transformation. (Default = identity).
  argument: `'-transformation %s'`

invert_transformation: (a boolean)
  Invert the transformation before using it.
  argument: `'-invert_transformation'`

vio_transform: (a boolean)
  VIO_Transform the input sampling with the transform (default).
  argument: `'-tfm_input_sampling'`
  mutually_exclusive: vio_transform, no_input_sampling

no_input_sampling: (a boolean)
  Use the input sampling without transforming (old behaviour).
  argument: `'-use_input_sampling'`
  mutually_exclusive: vio_transform, no_input_sampling

like: (a pathlike object or string representing an existing file)
  Specifies a model file for the resampling.
  argument: `'-like %s'`

format_byte: (a boolean)
  Write out byte data.
  argument: `'-byte'`
  mutually_exclusive: format_byte, format_short, format_int,
    format_long, format_float, format_double, format_signed,
    format_unsigned

format_short: (a boolean)
  Write out short integer data.
  argument: `'-short'`
  mutually_exclusive: format_byte, format_short, format_int,
    format_long, format_float, format_double, format_signed,
format_unsigned: (a boolean)
Write out unsigned integer data.
argument: `--unsigned`
mutually_exclusive: format_byte, format_short, format_int,
format_long, format_float, format_double, format_signed,
format_unsigned

format_int: (a boolean)
Write out 32-bit integer data.
argument: `--int`
mutually_exclusive: format_byte, format_short, format_int,
format_long, format_float, format_double, format_signed,
format_unsigned

format_long: (a boolean)
Superseded by -int.
argument: `--long`
mutually_exclusive: format_byte, format_short, format_int,
format_long, format_float, format_double, format_signed,
format_unsigned

format_float: (a boolean)
Write out single-precision floating-point data.
argument: `--float`
mutually_exclusive: format_byte, format_short, format_int,
format_long, format_float, format_double, format_signed,
format_unsigned

format_double: (a boolean)
Write out double-precision floating-point data.
argument: `--double`
mutually_exclusive: format_byte, format_short, format_int,
format_long, format_float, format_double, format_signed,
format_unsigned

format_signed: (a boolean)
Write signed integer data.
argument: `--signed`
mutually_exclusive: format_byte, format_short, format_int,
format_long, format_float, format_double, format_signed,
format_unsigned

format_unsigned: (a boolean)
Write unsigned integer data (default).
argument: `--unsigned`
mutually_exclusive: format_byte, format_short, format_int,
format_long, format_float, format_double, format_signed,
format_unsigned

output_range: (a tuple of the form: (a float, a float))
Valid range for output data. Default value: -1.79769e+308
-1.79769e+308.
argument: `--range %s %s`

transverse_slices: (a boolean)
Write out transverse slices.
argument: `--transverse`
mutually_exclusive: transverse, sagittal, coronal

sagittal_slices: (a boolean)
Write out sagittal slices
argument: `--sagittal`
mutually_exclusive: transverse, sagittal, coronal

coronal_slices: (a boolean)
Write out coronal slices
argument: `--coronal`
mutually_exclusive: transverse, sagittal, coronal

no_fill: (a boolean)
Use value zero for points outside of input volume.
argument: `--nofill`
mutually_exclusive: nofill, fill

(continues on next page)
fill: (a boolean)
  Use a fill value for points outside of input volume.
  argument: ``-fill``
  mutually_exclusive: nofill, fill

fill_value: (a float)
  Specify a fill value for points outside of input volume. Default value: 1.79769e+308.
  argument: ``-fillvalue %s``
  requires: fill

keep_real_range: (a boolean)
  Keep the real scale of the input volume.
  argument: ``-keep_real_range``
  mutually_exclusive: keep_real_range, nokeep_real_range

nokeep_real_range: (a boolean)
  Do not keep the real scale of the data (default).
  argument: ``-nokeep_real_range``
  mutually_exclusive: keep_real_range, nokeep_real_range

spacetype: (a unicode string)
  Set the spacetype attribute to a specified string.
  argument: ``-spacetype %s``

talairach: (a boolean)
  Output is in Talairach space.
  argument: ``-talairach``

origin: (a tuple of the form: (a float, a float, a float))
  Origin of first pixel in 3D space. Default value: 1.79769e+308 1.79769e+308 1.79769e+308.
  argument: ``-origin %s %s %s``

standard_sampling: (a boolean)
  Set the sampling to standard values (step, start and dircos).
  argument: ``-standard_sampling``

units: (a unicode string)
  Specify the units of the output sampling.
  argument: ``-units %s``

nelements: (a tuple of the form: (an integer (int or long), an integer (int or long), an integer (int or long)))
  Number of elements along each dimension (X, Y, Z).
  argument: ``-nelements %s %s %s``
  mutually_exclusive: nelements, nelements_x_y_or_z

xnelements: (an integer (int or long))
  Number of elements along the X dimension.
  argument: ``-xnelements %s``
  mutually_exclusive: nelements, nelements_x_y_or_z
  requires: ynelements, znelements

ynelements: (an integer (int or long))
  Number of elements along the Y dimension.
  argument: ``-ynelements %s``
  mutually_exclusive: nelements, nelements_x_y_or_z
  requires: xnelements, znelements

znelements: (an integer (int or long))
  Number of elements along the Z dimension.
  argument: ``-znelements %s``
  mutually_exclusive: nelements, nelements_x_y_or_z
  requires: xnelements, ynelements

step: (a tuple of the form: (an integer (int or long), an integer (int or long), an integer (int or long)))
  Step size along each dimension (X, Y, Z). Default value: (0, 0, 0).
  argument: ``-step %s %s %s``
mutually_exclusive: nelements, nelements_x_y_or_z

xstep: (an integer (int or long))
Step size along the X dimension. Default value: 0.
argument: `-xstep %s`
mutually_exclusive: step, step_x_y_or_z
requires: ystep, zstep

ystep: (an integer (int or long))
Step size along the Y dimension. Default value: 0.
argument: `-ystep %s`
mutually_exclusive: step, step_x_y_or_z
requires: xstep, zstep

zstep: (an integer (int or long))
Step size along the Z dimension. Default value: 0.
argument: `-zstep %s`
mutually_exclusive: step, step_x_y_or_z
requires: xstep, ystep

start: (a tuple of the form: (a float, a float, a float))
Start point along each dimension (X, Y, Z). Default value:
1.79769e+308 1.79769e+308 1.79769e+308.
argument: `-start %s %s %s`
mutually_exclusive: nelements, nelements_x_y_or_z

xstart: (a float)
Start point along the X dimension. Default value: 1.79769e+308.
argument: `-xstart %s`
mutually_exclusive: start, start_x_y_or_z
requires: ystart, zstart

ystart: (a float)
Start point along the Y dimension. Default value: 1.79769e+308.
argument: `-ystart %s`
mutually_exclusive: start, start_x_y_or_z
requires: xstart, zstart

zstart: (a float)
Start point along the Z dimension. Default value: 1.79769e+308.
argument: `-zstart %s`
mutually_exclusive: start, start_x_y_or_z
requires: xstart, ystart

dircos: (a tuple of the form: (a float, a float, a float))
Direction cosines along each dimension (X, Y, Z). Default value:
1.79769e+308 1.79769e+308 1.79769e+308 1.79769e+308 ...
1.79769e+308 1.79769e+308 1.79769e+308.
argument: `-dircos %s %s %s`
mutually_exclusive: nelements, nelements_x_y_or_z

xdircos: (a float)
Direction cosines along the X dimension. Default value: 1.79769e+308
1.79769e+308 1.79769e+308.
argument: `-xdircos %s`
mutually_exclusive: dircos, dircos_x_y_or_z
requires: ydircos, zdircos

ydircos: (a float)
Direction cosines along the Y dimension. Default value: 1.79769e+308
1.79769e+308 1.79769e+308.
argument: `-ydircos %s`
mutually_exclusive: dircos, dircos_x_y_or_z
requires: xdircos, zdircos

zdircos: (a float)
Direction cosines along the Z dimension. Default value: 1.79769e+308
1.79769e+308 1.79769e+308.
(continues on next page)
64.2.19 Reshape

Link to code
Wraps the executable command mincreshape.
Cut a hyperslab out of a minc file, with dimension reordering.
This is also useful for rewriting with a different format, for example converting to short (see example below).

Examples

```python
>>> from nipype.interfaces.minc import Reshape
>>> from nipype.interfaces.minc.testdata import nonempty_minc_data

>>> input_file = nonempty_minc_data(0)
>>> reshape_to_short = Reshape(input_file=input_file, write_short=True)
>>> reshape_to_short.run()  # doctest: +SKIP
```

Inputs:

<table>
<thead>
<tr>
<th>Mandatory</th>
<th>[Mandatory]</th>
</tr>
</thead>
<tbody>
<tr>
<td>input_file</td>
<td>input file, argument: <code>%s</code> , position: -2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optional</th>
<th>[Optional]</th>
</tr>
</thead>
<tbody>
<tr>
<td>output_file</td>
<td>output file, argument: <code>%s</code> , position: -1</td>
</tr>
<tr>
<td>verbose</td>
<td>Print out log messages. Default: False.</td>
</tr>
<tr>
<td>clobber</td>
<td>Overwrite existing file. argument: <code>-clobber</code></td>
</tr>
<tr>
<td>write_short</td>
<td>Convert to short integer data. argument: <code>-short</code></td>
</tr>
<tr>
<td>args</td>
<td>(a unicode string)</td>
</tr>
</tbody>
</table>

Outputs:

<table>
<thead>
<tr>
<th>output_file</th>
<th>(a pathlike object or string representing an existing file)</th>
</tr>
</thead>
</table>

Environment variables

| environ | (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) |

64.2. interfaces.minc.minc
Additional parameters to the command
argument: `\`%s\`'
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {}). Environment variables

Outputs:

output_file: (a pathlike object or string representing an existing
  file)
  output file

64.2.20 ToEcat

Link to code
Wraps the executable command minctoecat.
Convert a 2D image, a 3D volumes or a 4D dynamic volumes written in MINC file format to a 2D, 3D or 4D
Ecat7 file.

Examples

```python
>>> from nipype.interfaces.minc import ToEcat
>>> from nipype.interfaces.minc.testdata import minc2Dfile

>>> c = ToEcat(input_file=minc2Dfile)
>>> c.run() # doctest: +SKIP

>>> c = ToEcat(input_file=minc2Dfile, voxels_as_integers=True)
>>> c.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
input_file: (a pathlike object or string representing an existing
  file)
  input file to convert
  argument: `\`%s\``, position: -2

[Optional]
output_file: (a pathlike object or string representing a file)
  output file
  argument: `\`%s\``, position: -1
ignore_patient_variable: (a boolean)
  Ignore informations from the minc patient variable.
  argument: `\`-ignore_patient_variable\`'
ignore_study_variable: (a boolean)
  Ignore informations from the minc study variable.
  argument: `\`-ignore_study_variable\`'
ignore_acquisition_variable: (a boolean)
  Ignore informations from the minc acquisition variable.
  argument: `\`-ignore_acquisition_variable\`'
ignore_ecat_acquisition_variable: (a boolean)
  Ignore informations from the minc ecat_acquisition variable.
  argument: `\`-ignore_ecat_acquisition_variable\`'
```
ignore_ecat_main: (a boolean)
   Ignore informations from the minc ecat-main variable.
   argument: `-ignore_ecat_main`
ignore_ecat_subheader_variable: (a boolean)
   Ignore informations from the minc ecat-subhdr variable.
   argument: `-ignore_ecat_subheader_variable`
no_decay_corr_fctr: (a boolean)
   Do not compute the decay correction factors
   argument: `-no_decay_corr_fctr`
voxels_as_integers: (a boolean)
   Voxel values are treated as integers, scale and calibration factors
   are set to unity
   argument: `-label`
args: (a unicode string)
   Additional parameters to the command
   argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value
   of class 'str' and with values which are a bytes or None or a
   value of class 'str', nipype default value: {})
   Environment variables

Outputs:

output_file: (a pathlike object or string representing an existing
   file)
   output file

64.2.21 ToRaw

Link to code
Wraps the executable command minctoraw.
Dump a chunk of MINC file data. This program is largely superceded by mincextract (see Extract).

Examples

```python
>>> from nipype.interfaces.minc import ToRaw
>>> from nipype.interfaces.minc.testdata import minc2Dfile
```

```python
>>> toraw = ToRaw(input_file=minc2Dfile)
>>> toraw.run()  # doctest: +SKIP
```

```python
>>> toraw = ToRaw(input_file=minc2Dfile, write_range=(0, 100))
>>> toraw.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
input_file: (a pathlike object or string representing an existing
   file)
   input file
   argument: `%s`, position: -2

[Optional]
output_file: (a pathlike object or string representing a file)
   output file

(continues on next page)
write_byte: (a boolean)
   Write out data as bytes.
   argument: `-byte`
   mutually_exclusive: write_byte, write_short, write_int, write_long,
   write_float, write_double
write_short: (a boolean)
   Write out data as short integers.
   argument: `-short`
   mutually_exclusive: write_byte, write_short, write_int, write_long,
   write_float, write_double
write_int: (a boolean)
   Write out data as 32-bit integers.
   argument: `-int`
   mutually_exclusive: write_byte, write_short, write_int, write_long,
   write_float, write_double
write_long: (a boolean)
   Superseded by write_int.
   argument: `-long`
   mutually_exclusive: write_byte, write_short, write_int, write_long,
   write_float, write_double
write_float: (a boolean)
   Write out data as single precision floating-point values.
   argument: `-float`
   mutually_exclusive: write_byte, write_short, write_int, write_long,
   write_float, write_double
write_double: (a boolean)
   Write out data as double precision floating-point values.
   argument: `-double`
   mutually_exclusive: write_byte, write_short, write_int, write_long,
   write_float, write_double
write_signed: (a boolean)
   Write out signed data.
   argument: `-signed`
   mutually_exclusive: write_signed, write_unsigned
write_unsigned: (a boolean)
   Write out unsigned data.
   argument: `-unsigned`
   mutually_exclusive: write_signed, write_unsigned
write_range: (a tuple of the form: (a float, a float))
   Specify the range of output values. Default value: 1.79769e+308
   1.79769e+308.
   argument: `-range %s %s`
normalize: (a boolean)
   Normalize integer pixel values to file max and min.
   argument: `-normalize`
   mutually_exclusive: normalize, nonormalize
nonormalize: (a boolean)
   Turn off pixel normalization.
   argument: `-nonormalize`
   mutually_exclusive: normalize, nonormalize
out_file: (a pathlike object or string representing a file)
   argument: `> %s`, position: -1
args: (a unicode string)
   Additional parameters to the command
   argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value
   of class 'str' and with values which are a bytes or None or a
Outputs:

| output_file: (a pathlike object or string representing an existing file) |
| output file in raw format |

### 64.2.22 VolSymm

#### Link to code

Wraps the executable command `volsymm`.

Make a volume symmetric about an axis either linearly and/or nonlinearly. This is done by registering a volume to a flipped image of itself.

This tool is part of the minc-widgets package:

https://github.com/BIC-MNI/minc-widgets/blob/master/volsymm/volsymm

#### Examples

```python
>>> from nipype.interfaces.minc import VolSymm
>>> from nipype.interfaces.minc.testdata import nonempty_minc_data

>>> input_file = nonempty_minc_data(0)
>>> volsymm = VolSymm(input_file=input_file)
>>> volsymm.run() # doctest: +SKIP
```

#### Inputs:

- **input_file**: (a pathlike object or string representing an existing file)
  - input file
    - argument: ``\%s``, position: -3

- **trans_file**: (a pathlike object or string representing a file)
  - output xfm trans file
    - argument: ``\%s``, position: -2

- **output_file**: (a pathlike object or string representing a file)
  - output file
    - argument: ``\%s``, position: -1

- **input_grid_files**: (a list of items which are a pathlike object or string representing a file)
  - input grid file(s)

- **verbose**: (a boolean)
  - Print out log messages. Default: False.
    - argument: ``-verbose``

- **clobber**: (a boolean, nipype default value: True)
  - Overwrite existing file.
    - argument: ``-clobber``

- **fit_linear**: (a boolean)
  - Fit using a linear xfm.
    - argument: ``-linear``

- **fit_nonlinear**: (a boolean)
Fit using a non-linear xfm.
argument: `''-nonlinear``

nofit: (a boolean)
Use the input transformation instead of generating one.
argument: `''-nofit``

cfg_file: (a pathlike object or string representing an existing file)
File containing the fitting configuration (nlpfit -help for info).
argument: `''-config_file %s``

x: (a boolean)
Flip volume in x-plane (default).
argument: `''-x``

y: (a boolean)
Flip volume in y-plane.
argument: `''-y``

z: (a boolean)
Flip volume in z-plane.
argument: `''-z``

args: (a unicode string)
Additional parameters to the command
argument: `''%s``

eviron: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: `{}`)
Environment variables

Outputs:

output_file: (a pathlike object or string representing an existing file)
output file

trans_file: (a pathlike object or string representing an existing file)
xfm trans file

output_grid: (a pathlike object or string representing an existing file)
output grid file

64.2.23 Volcentre

Link to code
Wraps the executable command volcentre.
Centre a MINC image’s sampling about a point, typically (0,0,0).

Example

```python
>>> from nipype.interfaces.minc import Volcentre
>>> from nipype.interfaces.minc.testdata import minc2Dfile
>>> vc = Volcentre(input_file=minc2Dfile)
>>> vc.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
input_file: (a pathlike object or string representing an existing file)
input file to centre  
    argument: `'-%s'`, position: -2

[Optional]
output_file: (a pathlike object or string representing a file)  
    output file  
    argument: `'-%s'`, position: -1
verbose: (a boolean)  
    Print out log messages. Default: False.  
    argument: `'-verbose'`
clobber: (a boolean, nipype default value: True)  
    Overwrite existing file.  
    argument: `'-clobber'`
com: (a boolean)  
    Use the CoM of the volume for the new centre (via mincstats).  
    Default: False  
    argument: `'-com'`
centre: (a tuple of the form: (a float, a float, a float))  
    Centre to use (x,y,z) [default: 0 0 0].  
    argument: `'-centre %s %s %s'`
zero_dircos: (a boolean)  
    Set the direction cosines to identity [default].  
    argument: `'-zero_dircos'`
args: (a unicode string)  
    Additional parameters to the command  
    argument: `'-%s'`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})  
    Environment variables

Outputs:

output_file: (a pathlike object or string representing an existing file)  
    output file

### 64.2.24 Voliso

**Link to code**

Wraps the executable command `voliso`. Changes the steps and starts in order that the output volume has isotropic sampling.

**Examples**

```python
>>> from nipype.interfaces.minc import Voliso
>>> from nipype.interfaces.minc.testdata import minc2Dfile
>>> viso = Voliso(input_file=minc2Dfile, minstep=0.1, avgstep=True)
>>> viso.run()  # doctest: +SKIP
```

**Inputs:**

[Mandatory]
input_file: (a pathlike object or string representing an existing file)  
    input file to convert to isotropic sampling
64.2.25 Volpad

Link to code
Wraps the executable command volpad.
Centre a MINC image’s sampling about a point, typically (0,0,0).

Examples

```
>>> from nipype.interfaces.minc import Volpad
>>> from nipype.interfaces.minc.testdata import minc2Dfile
>>> vp = Volpad(input_file=minc2Dfile, smooth=True, smooth_distance=4)
>>> vp.run()  # doctest: +SKIP
```

Inputs:

- input_file: (a pathlike object or string representing an existing file)
  - input file to centre
  - argument: ``%s``, position: -2

Outputs:

- output_file: (a pathlike object or string representing an existing file)
  - output file
  - argument: ``%s``, position: -1
[Optional]

output_file: (a pathlike object or string representing a file)
  output file
    argument: `--output %s`, position: -1

verbose: (a boolean)
  Print out log messages. Default: False.
    argument: `--verbose`

clobber: (a boolean, nipype default value: True)
  Overwrite existing file.
    argument: `--clobber`

auto: (a boolean)
  Automatically determine padding distances (uses -distance as max).
    Default: False.
    argument: `--auto`

auto_freq: (a float)
  Frequency of voxels over bimodal threshold to stop at [default: 500].
    argument: `--auto_freq %s`

distance: (an integer (int or long))
  Padding distance (in voxels) [default: 4].
    argument: `--distance %s`

smooth: (a boolean)
  Smooth (blur) edges before padding. Default: False.
    argument: `--smooth`

smooth_distance: (an integer (int or long))
  Smoothing distance (in voxels) [default: 4].
    argument: `--smooth_distance %s`

args: (a unicode string)
  Additional parameters to the command
    argument: `%%s`

environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a value
  of class 'str', nipype default value: {}) 
  Environment variables

Outputs:

output_file: (a pathlike object or string representing an existing
  file)
    output file

64.2.26 XfmAvg

Link to code
Wraps the executable command xfmavg.
Average a number of xfm transforms using matrix logs and exponents. The program xfmavg calls Octave for numerical work.
This tool is part of the minc-widgets package:
https://github.com/BIC-MNI/minc-widgets/tree/master/xfmavg

Examples

```python
>>> from nipype.interfaces.minc import XfmAvg
>>> from nipype.interfaces.minc.testdata import nonempty_minc_data, nlp_config
>>> from nipype.testing import example_data
```
>>> xfm1 = example_data('minc_initial.xfm')
>>> xfm2 = example_data('minc_initial.xfm')  # cheating for doctest
>>> xfmavg = XfmAvg(input_files=[xfm1, xfm2])
>>> xfmavg.run()  # doctest: +SKIP

Inputs:

[Mandatory]
input_files: (a list of items which are a pathlike object or string representing an existing file)
  input file(s)
  argument: ``%s``, position: -2

[Optional]
input_grid_files: (a list of items which are a pathlike object or string representing a file)
  input grid file(s)
output_file: (a pathlike object or string representing a file)
  output file
  argument: ``%s``, position: -1
verbose: (a boolean)
  Print out log messages. Default: False.
  argument: ``-verbose``
clobber: (a boolean, nipype default value: True)
  Overwrite existing file.
  argument: ``-clobber``
avg_linear: (a boolean)
  average the linear part [default].
  argument: ``-avg_linear``
avg_nonlinear: (a boolean)
  average the non-linear part [default].
  argument: ``-avg_nonlinear``
ignore_linear: (a boolean)
  opposite of -avg_linear.
  argument: ``-ignore_linear``
ignore_nonlinear: (a boolean)
  opposite of -avg_nonlinear.
  argument: ``-ignore_nonlinear``
args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  Environment variables

Outputs:

output_file: (a pathlike object or string representing an existing file)
  output file
output_grid: (a pathlike object or string representing an existing file)
  output grid file

64.2.27 XfmConcat

Link to code
Wraps the executable command xfmconcat.
Concatenate transforms together. The output transformation is equivalent to applying input1.xfm, then input2.xfm, ..., in that order.

**Examples**

```python
>>> from nipype.interfaces.minc import XfmConcat
>>> from nipype.interfaces.minc.testdata import minc2Dfile
>>> conc = XfmConcat(input_files=['input1.xfm', 'input1.xfm'])
>>> conc.run()  # doctest: +SKIP
```

**Inputs:**

- **input_files**: (a list of items which are a pathlike object or string representing an existing file)
  - input file(s)
  - argument: ``%s``, position: -2

- **input_grid_files**: (a list of items which are a pathlike object or string representing a file)
  - input grid file(s)

- **output_file**: (a pathlike object or string representing a file)
  - output file
  - argument: ``%s``, position: -1

- **verbose**: (a boolean)
  - Print out log messages. Default: False.
  - argument: ``-verbose``

- **clobber**: (a boolean, nipype default value: True)
  - Overwrite existing file.
  - argument: ``-clobber``

- **args**: (a unicode string)
  - Additional parameters to the command
  - argument: ``%s``

- **environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  - Environment variables

**Outputs:**

- **output_file**: (a pathlike object or string representing an existing file)
  - output file

- **output_grids**: (a list of items which are a pathlike object or string representing an existing file)
  - output grids

### 64.2.28 XfmInvert

Link to code

Wraps the executable command xfminvert.

Invert an xfm transform file.
Examples

```python
>>> from nipype.interfaces.minc import XfmAvg
>>> from nipype.testing import example_data

>>> xfm = example_data('minc_initial.xfm')
>>> invert = XfmInvert(input_file=xfm)
>>> invert.run() # doctest: +SKIP
```

Inputs:

<table>
<thead>
<tr>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>input_file</code></td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td></td>
<td>input file</td>
</tr>
<tr>
<td></td>
<td>argument: <code>%s</code>, position: -2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>output_file</code></td>
<td>(a pathlike object or string representing a file)</td>
</tr>
<tr>
<td></td>
<td>output file</td>
</tr>
<tr>
<td></td>
<td>argument: <code>%s</code>, position: -1</td>
</tr>
<tr>
<td><code>verbose</code></td>
<td>(a boolean)</td>
</tr>
<tr>
<td></td>
<td>Print out log messages. Default: False.</td>
</tr>
<tr>
<td></td>
<td>argument: <code>-verbose</code></td>
</tr>
<tr>
<td><code>clobber</code></td>
<td>(a boolean, nipype default value: True)</td>
</tr>
<tr>
<td></td>
<td>Overwrite existing file.</td>
</tr>
<tr>
<td></td>
<td>argument: <code>-clobber</code></td>
</tr>
<tr>
<td><code>args</code></td>
<td>(a unicode string)</td>
</tr>
<tr>
<td></td>
<td>Additional parameters to the command</td>
</tr>
<tr>
<td></td>
<td>argument: <code>%s</code></td>
</tr>
<tr>
<td><code>environ</code></td>
<td>(a dictionary with keys which are a bytes or None or a value of class 'str'</td>
</tr>
<tr>
<td></td>
<td>and with values which are a bytes or None or a value of class 'str', nipype</td>
</tr>
<tr>
<td></td>
<td>default value: {}</td>
</tr>
<tr>
<td></td>
<td>Environment variables</td>
</tr>
</tbody>
</table>

Outputs:

| `output_file`              | (a pathlike object or string representing an existing file)                 |
|                           | output file                                                                 |
| `output_grid`              | (a pathlike object or string representing an existing file)                 |
|                           | output grid file                                                            |

64.3 interfaces.minc.testdata

64.3.1 nonempty_minc_data()

Link to code
65.1 interfaces.mipav.developer

65.1.1 JistBrainMgdmSegmentation

Link to code
Wraps the executable command "java edu.jhu.ece.iacj.jist.cli.run de.mpg.cbs.jist.brain.JistBrainMgdmSegmentation"

Title: MGDM Whole Brain Segmentation
Category: Developer Tools
Description: Estimate brain structures from an atlas for a MRI dataset (multiple input combinations are possible).
Version: 2.0.RC
Inputs:

```
[Optional]
inpM2RAGE: (a pathlike object or string representing an existing file)
    MP2RAGE T1 Map Image
    argument: `--inMP2RAGE %s`
inMP2RAGE2: (a pathlike object or string representing an existing file)
    MP2RAGE T1-weighted Image
    argument: `--inMP2RAGE2 %s`
inPV: (a pathlike object or string representing an existing file)
    PV / Dura Image
    argument: `--inPV %s`
inMPRAGE: (a pathlike object or string representing an existing file)
    MPRAGE T1-weighted Image
    argument: `--inMPRAGE %s`
inFLAIR: (a pathlike object or string representing an existing file)
    FLAIR Image
    argument: `--inFLAIR %s`
inAtlas: (a pathlike object or string representing an existing file)
    Atlas file
    argument: `--inAtlas %s`
inData: (a float)
    Data weight
```
(continues on next page)
argument: `--inData %f`
inCurvature: (a float)
  Curvature weight
  argument: `--inCurvature %f`
inPosterior: (a float)
  Posterior scale (mm)
  argument: `--inPosterior %f`
inMax: (an integer (int or long))
  Max iterations
  argument: `--inMax %d`
inMin: (a float)
  Min change
  argument: `--inMin %f`
inSteps: (an integer (int or long))
  Steps
  argument: `--inSteps %d`
inTopology: ('26/6' or '6/26' or '18/6' or '6/18' or '6/6' or 'wcs'
  or 'wco' or 'no')
  Topology
  argument: `--inTopology %s`
inCompute: ('true' or 'false')
  Compute posteriors
  argument: `--inCompute %s`
inAdjust: ('true' or 'false')
  Adjust intensity priors
  argument: `--inAdjust %s`
inOutput: ('segmentation' or 'memberships')
  Output images
  argument: `--inOutput %s`
xPrefExt: ('nrrd')
  Output File Type
  argument: `--xPrefExt %s`
outSegmented: (a boolean or a pathlike object or string representing
  a file)
  Segmented Brain Image
  argument: `--outSegmented %s`
outLevelset: (a boolean or a pathlike object or string representing a
  file)
  Levelset Boundary Image
  argument: `--outLevelset %s`
outPosterior2: (a boolean or a pathlike object or string representing a
  file)
  Posterior Maximum Memberships (4D)
  argument: `--outPosterior2 %s`
outPosterior3: (a boolean or a pathlike object or string representing a
  file)
  Posterior Maximum Labels (4D)
  argument: `--outPosterior3 %s`
null: (a unicode string)
  Execution Time
  argument: `--null %s`
xDefaultMem: (an integer (int or long))
  Set default maximum heap size
  argument: `--xDefaultMem %d`
xMaxProcess: (an integer (int or long), nipype default value: 1)
  Set default maximum number of processes.
  argument: `--xMaxProcess %d`
args: (a unicode string)
    Additional parameters to the command
    argument: `\%s`

environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})
    Environment variables

Outputs:

outSegmented: (a pathlike object or string representing an existing
    file)
    Segmented Brain Image
outLevelset: (a pathlike object or string representing an existing
    file)
    Levelset Boundary Image
outPosterior2: (a pathlike object or string representing an existing
    file)
    Posterior Maximum Memberships (4D)
outPosterior3: (a pathlike object or string representing an existing
    file)
    Posterior Maximum Labels (4D)

65.1.2 JistBrainMp2rageDuraEstimation

Link to code
Wraps the executable command `'java edu.jhu.ece.iacl.jist.cli.run

title: MP2RAGE Dura Estimation
category: Developer Tools
description: Filters a MP2RAGE brain image to obtain a probability map of dura matter.
version: 3.0.RC
Inputs:

[Optional]
inSecond: (a pathlike object or string representing an existing file)
    Second inversion (Inv2) Image
    argument: `--inSecond %s`
inSkull: (a pathlike object or string representing an existing file)
    Skull Stripping Mask
    argument: `--inSkull %s`
inDistance: (a float)
    Distance to background (mm)
    argument: `--inDistance %f`
inoutput: ('dura_region' or 'boundary' or 'dura_prior' or 'bg_prior'
    or 'intens_prior')
    Outputs an estimate of the dura / CSF boundary or an estimate of the
    entire dura region.
    argument: `--inoutput %s`
xPrefExt: ('nrrd')
    Output File Type
    argument: `--xPrefExt %s`
outDura: (a boolean or a pathlike object or string representing a
    file)
    Dura Image
    argument: `--outDura %s`
null: (a unicode string)
    Execution Time
    argument: `--null %s`

xDefaultMem: (an integer (int or long))
    Set default maximum heap size
    argument: `--xDefaultMem %d`

xMaxProcess: (an integer (int or long), nipype default value: 1)
    Set default maximum number of processes.
    argument: `--xMaxProcess %d`

args: (a unicode string)
    Additional parameters to the command
    argument: `%s`

evernin: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
    Environment variables

Outputs:

outDura: (a pathlike object or string representing an existing file)
    Dura Image

65.1.3 JistBrainMp2rageSkullStripping

Link to code
Wraps the executable command "java edu.jhu.ece.iacl.jist.cli.run de.mpg.cbs.jist.brain.JistBrainMp2rageSkullStripping"

title: MP2RAGE Skull Stripping
category: Developer Tools
description: Estimate a brain mask for a MP2RAGE dataset. At least a T1-weighted or a T1 map image is required.
version: 3.0_RC

Inputs:

[Optional]
inSecond: (a pathlike object or string representing an existing file)
    Second inversion (Inv2) Image
    argument: `--inSecond %s`
inT1: (a pathlike object or string representing an existing file)
    T1 Map (T1_Images) Image (opt)
    argument: `--inT1 %s`
inT1weighted: (a pathlike object or string representing an existing file)
    T1-weighted (UNI) Image (opt)
    argument: `--inT1 %s`
inFilter: (a pathlike object or string representing an existing file)
    Filter Image (opt)
    argument: `--inFilter %s`
inSkip: ('true' or 'false')
    Skip zero values
    argument: `--inSkip %s`
xPrefExt: ('nrrd')
    Output File Type
    argument: `--xPrefExt %s`
outBrain: (a boolean or a pathlike object or string representing a
    file)
Brain Mask Image
argument: `''--outBrain %s''`
outMasked: (a boolean or a pathlike object or string representing a file)
Masked T1 Map Image
argument: `''--outMasked %s''`
outMasked2: (a boolean or a pathlike object or string representing a file)
Masked T1-weighted Image
argument: `''--outMasked2 %s''`
outMasked3: (a boolean or a pathlike object or string representing a file)
Masked Filter Image
argument: `''--outMasked3 %s''`
null: (a unicode string)
Execution Time
argument: `''--null %s''`
xDefaultMem: (an integer (int or long))
Set default maximum heap size
argument: `''--xDefaultMem %d''`
xMaxProcess: (an integer (int or long), nipype default value: 1)
Set default maximum number of processes.
argument: `''--xMaxProcess %d''`
args: (a unicode string)
Additional parameters to the command
argument: `''%s''`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outs:

outBrain: (a pathlike object or string representing an existing file)
Brain Mask Image
outMasked: (a pathlike object or string representing an existing file)
Masked T1 Map Image
outMasked2: (a pathlike object or string representing an existing file)
Masked T1-weighted Image
outMasked3: (a pathlike object or string representing an existing file)
Masked Filter Image

65.1.4 JistBrainPartialVolumeFilter

Link to code
Wraps the executable command 
```
```

title: Partial Volume Filter
category: Developer Tools
description: Filters an image for regions of partial voluming assuming a ridge-like model of intensity.
version: 2.0.RC
Inputs:
[Optional]
inInput: (a pathlike object or string representing an existing file)
Input Image
  argument: ``--inInput %s``
inPV: ('bright' or 'dark' or 'both')
Outputs the raw intensity values or a probability score for the
partial volume regions.
  argument: ``--inPV %s``
inoutput: ('probability' or 'intensity')
  output
  argument: ``--inoutput %s``
xPrefExt: ('nrrd')
Output File Type
  argument: ``--xPrefExt %s``
outPartial: (a boolean or a pathlike object or string representing a
  file)
Partial Volume Image
  argument: ``--outPartial %s``
null: (a unicode string)
Execution Time
  argument: ``--null %s``
xDefaultMem: (an integer (int or long))
Set default maximum heap size
  argument: ``--xDefaultMem %d``
xMaxProcess: (an integer (int or long), nipype default value: 1)
Set default maximum number of processes.
  argument: ``--xMaxProcess %d``
args: (a unicode string)
Additional parameters to the command
  argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {}) 
Environment variables

Outputs:

outPartial: (a pathlike object or string representing an existing
  file)
Partial Volume Image

65.1.5 JistCortexSurfaceMeshInflation

Link to code
Wraps the executable command "java edu.jhu.ece.iacl.jist.cli.run.de.mpg.cbs.jist.cortex.JistCortexSurfaceMeshInflation"

title: Surface Mesh Inflation
category: Developer Tools
description: Inflates a cortical surface mesh. D. Tosun, M. E. Rettmann, X. Han, X. Tao, C. Xu, S. M. Resnick, 
2004.
version: 3.0.RC
contributor: Duygu Tosun

Inputs:

[Optional]
inLevelset: (a pathlike object or string representing an existing
    file)
    Levelset Image
    argument: ```--inLevelset %s``
inSOR: (a float)
    SOR Parameter
    argument: ```--inSOR %f``
inMean: (a float)
    Mean Curvature Threshold
    argument: ```--inMean %f``
inStep: (an integer (int or long))
    Step Size
    argument: ```--inStep %d``
inMax: (an integer (int or long))
    Max Iterations
    argument: ```--inMax %d``
inLorentzian: ('true' or 'false')
    Lorentzian Norm
    argument: ```--inLorentzian %s``
inTopology: ('26/6' or '6/26' or '18/6' or '6/18' or '6/6' or 'wcs'
    or 'wco' or 'no')
    Topology
    argument: ```--inTopology %s``
xPrefExt: ('nrrd')
    Output File Type
    argument: ```--xPrefExt %s``

outOriginal: (a boolean or a pathlike object or string representing a
    file)
    Original Surface
    argument: ```--outOriginal %s``

outInflated: (a boolean or a pathlike object or string representing a
    file)
    Inflated Surface
    argument: ```--outInflated %s``

null: (a unicode string)
    Execution Time
    argument: ```--null %s``
xDefaultMem: (an integer (int or long))
    Set default maximum heap size
    argument: ```--xDefaultMem %d``
xMaxProcess: (an integer (int or long), nipype default value: 1)
    Set default maximum number of processes.
    argument: ```--xMaxProcess %d``

args: (a unicode string)
    Additional parameters to the command
    argument: ```%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})
    Environment variables

Outputs:

outOriginal: (a pathlike object or string representing an existing
    file)
    Original Surface
outInflated: (a pathlike object or string representing an existing
    file)
65.1.6 JistIntensityMp2rageMasking

Link to code
Wraps the executable command "java edu.jhu.ece.iacl.jist.cli.run.de.mpg.cbs.jist.intensity.JistIntensityMp2rageMasking"

title: MP2RAGE Background Masking
category: Developer Tools
description: Estimate a background signal mask for a MP2RAGE dataset.
version: 3.0.RC
Inputs:

[Optional]
inSecond: (a pathlike object or string representing an existing file)
    Second inversion (Inv2) Image
    argument: `--inSecond %s`
inQuantitative: (a pathlike object or string representing an existing file)
    Quantitative T1 Map (T1_Images) Image
    argument: `--inQuantitative %s`
inT1weighted: (a pathlike object or string representing an existing file)
    T1-weighted (UNI) Image
    argument: `--inT1weighted %s`
inBackground: ('exponential' or 'half-normal')
    Model distribution for background noise (default is half-normal, exponential is more stringent).
    argument: `--inBackground %s`
inSkip: ('true' or 'false')
    Skip zero values
    argument: `--inSkip %s`
inMasking: ('binary' or 'proba')
    Whether to use a binary threshold or a weighted average based on the probability.
    argument: `--inMasking %s`
xPrefExt: ('nrrd')
    Output File Type
    argument: `--xPrefExt %s`
outSignal: (a boolean or a pathlike object or string representing a file)
    Signal Proba Image
    argument: `--outSignal_Proba %s`
outSignal2: (a boolean or a pathlike object or string representing a file)
    Signal Mask Image
    argument: `--outSignal_Mask %s`
outMasked: (a boolean or a pathlike object or string representing a file)
    Masked T1 Map Image
    argument: `--outMasked_T1_Map %s`
outMasked2: (a boolean or a pathlike object or string representing a file)
    Masked Iso Image

(continues on next page)
65.1.7 JistLaminarProfileCalculator

Link to code
Wraps the executable command ""java edu.jhu.ece.iacl.jist.cli.run.de.mpg.cbs.jist.laminar.JistLaminarProfileCalculator"".

title: Profile Calculator
category: Developer Tools
description: Compute various moments for intensities mapped along a cortical profile.
version: 3.0.RC

Inputs:

[Optional]
inIntensity: (a pathlike object or string representing an existing file)
    Intensity Profile Image
    argument: `--inIntensity %s`'
inMask: (a pathlike object or string representing an existing file)
    Mask Image (opt, 3D or 4D)
    argument: `--inMask %s`'
incomputed: ('mean' or 'stdev' or 'skewness' or 'kurtosis')
    computed statistic
    argument: `--incomputed %s`'
xPrefExt: ('nrrd')
65.1.8 JistLaminarProfileGeometry

Link to code
Wraps the executable command "java edu.jhu.ece.iacl.jist.cli.run.de.mpg.cbs.jist.laminar.JistLaminarProfileGeometry".

title: Profile Geometry
category: Developer Tools
description: Compute various geometric quantities for a cortical layers.
version: 3.0.RC
Inputs:

[Optional]
inProfile: (a pathlike object or string representing an existing file)
   Profile Surface Image
   argument: '--inProfile %s'
incomputed: ('thickness' or 'curvedness' or 'shape_index' or 'mean_curvature' or 'gauss_curvature' or 'profile_length' or 'profile_curvature' or 'profile_torsion')
   computed measure
   argument: '--incomputed %s'
inregularization: ('none' or 'Gaussian')
   regularization
   argument: '--inregularization %s'
insmoothing: (a float)
   smoothing parameter
   argument: '--insmoothing %f'
inoutside: (a float)
    outside extension (mm)
    argument: `'--inoutside %f'`
xPrefExt: ('nrrd')
    Output File Type
    argument: `'--xPrefExt %s'`
outResult: (a boolean or a pathlike object or string representing a
    file)
    Result
    argument: `'--outResult %s'`
null: (a unicode string)
    Execution Time
    argument: `'--null %s'`
xDefaultMem: (an integer (int or long))
    Set default maximum heap size
    argument: `'--xDefaultMem %d'`
xMaxProcess: (an integer (int or long), nipype default value: 1)
    Set default maximum number of processes.
    argument: `'--xMaxProcess %d'`
args: (a unicode string)
    Additional parameters to the command
    argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})
    Environment variables

Outputs:

outResult: (a pathlike object or string representing an existing
    file)
    Result

65.1.9 JistLaminarProfileSampling

Link to code

Wraps the executable command \\
``java edu.jhu.ece.iacl.jist.cli.run.de.mpg.cbs.jist.laminar.JistLaminarProfileSampling`\n.

title: Profile Sampling
category: Developer Tools
description: Sample some intensity image along a cortical profile across layer surfaces.
version: 3.0.RC

Inputs:

[Optional]
inProfile: (a pathlike object or string representing an existing
    file)
    Profile Surface Image
    argument: `'--inProfile %s'`
inIntensity: (a pathlike object or string representing an existing
    file)
    Intensity Image
    argument: `'--inIntensity %s'`
inCortex: (a pathlike object or string representing an existing file)
    Cortex Mask (opt)
    argument: `'--inCortex %s'`
xPrefExt: ('nrrd')
  Output File Type
  argument: "--xPrefExt %s"
outProfileMapped: (a boolean or a pathlike object or string representing a file)
  Profile-mapped Intensity Image
  argument: "--outProfileMapped %s"
outProfile2: (a boolean or a pathlike object or string representing a file)
  Profile 4D Mask
  argument: "--outProfile2 %s"
null: (a unicode string)
  Execution Time
  argument: "--null %s"
xDefaultMem: (an integer (int or long))
  Set default maximum heap size
  argument: "--xDefaultMem %d"
xMaxProcess: (an integer (int or long), nipype default value: 1)
  Set default maximum number of processes.
  argument: "--xMaxProcess %d"
args: (a unicode string)
  Additional parameters to the command
  argument: "%s"
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  Environment variables

Outputs:

outProfileMapped: (a pathlike object or string representing an existing file)
  Profile-mapped Intensity Image
outProfile2: (a pathlike object or string representing an existing file)
  Profile 4D Mask

65.1.10 JistLaminarROI Averaging

Link to code
Wraps the executable command `"java edu.jhu.ece.iacl.jist.cli.run.de.mpg.cbs.jist.laminar.JistLaminarROI Averaging"

Title: Profile ROI Averaging
Category: Developer Tools
Description: Compute an average profile over a given ROI.
Version: 3.0.RC

Inputs:

[Optional]
inIntensity: (a pathlike object or string representing an existing file)
  Intensity Profile Image
  argument: "--inIntensity %s"
inROI: (a pathlike object or string representing an existing file)
  ROI Mask
  argument: "--inROI %s"
inROI2: (a unicode string)
   ROI Name
   argument: `''--inROI2 %s'`

inMask: (a pathlike object or string representing an existing file)
   Mask Image (opt, 3D or 4D)
   argument: `''--inMask %s'`

xPrefExt: ('nrrd')
   Output File Type
   argument: `''--xPrefExt %s'`

outROI3: (a boolean or a pathlike object or string representing a file)
   ROI Average
   argument: `''--outROI3 %s'`

null: (a unicode string)
   Execution Time
   argument: `''--null %s'`

xDefaultMem: (an integer (int or long))
   Set default maximum heap size
   argument: `''--xDefaultMem %d'`

xMaxProcess: (an integer (int or long), nipype default value: 1)
   Set default maximum number of processes.
   argument: `''--xMaxProcess %d'`

args: (a unicode string)
   Additional parameters to the command
   argument: `''%s'`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
   Environment variables

Outputs:

outROI3: (a pathlike object or string representing an existing file)
   ROI Average

65.1.11 JistLaminarVolumetricLayering

Link to code

title: Volumetric Layering

category: Developer Tools


version: 3.0.RC

contributor: Miriam Waehnert (waehnert@cbs.mpg.de) http://www.cbs.mpg.de/

Inputs:

[Optional]

inInner: (a pathlike object or string representing an existing file)
   Inner Distance Image (GM/WM boundary)
   argument: `''--inInner %s'`

inOuter: (a pathlike object or string representing an existing file)
   Outer Distance Image (CSF/GM boundary)
   argument: `''--inOuter %s'`

(continues on next page)
inNumber: (an integer (int or long))
Number of layers
argument: ``--inNumber %d``

inMax: (an integer (int or long))
Max iterations for narrow band evolution
argument: ``--inMax %d``

inMin: (a float)
Min change ratio for narrow band evolution
argument: ``--inMin %f``

inLayering: ('distance-preserving' or 'volume-preserving')
Layering method
argument: ``--inLayering %s``

inLayering2: ('outward' or 'inward')
Layering direction
argument: ``--inLayering2 %s``

incurvature: (an integer (int or long))
curvature approximation scale (voxels)
argument: ``--incurvature %d``

inratio: (a float)
ratio smoothing kernel size (voxels)
argument: ``--inratio %f``

inpresmooth: ('true' or 'false')
pre-smooth cortical surfaces
argument: ``--inpresmooth %s``

inTopology: ('26/6' or '6/26' or '18/6' or '6/18' or '6/6' or 'wcs' or 'wco' or 'no')
Topology
argument: ``--inTopology %s``

xPrefExt: ('nrrd')
Output File Type
argument: ``--xPrefExt %s``

outContinuous: (a boolean or a pathlike object or string representing a file)
Continuous depth measurement
argument: ``--outContinuous %s``

outDiscrete: (a boolean or a pathlike object or string representing a file)
Discrete sampled layers
argument: ``--outDiscrete %s``

outLayer: (a boolean or a pathlike object or string representing a file)
Layer boundary surfaces
argument: ``--outLayer %s``

null: (a unicode string)
Execution Time
argument: ``--null %s``

xDefaultMem: (an integer (int or long))
Set default maximum heap size
argument: ``--xDefaultMem %d``

xMaxProcess: (an integer (int or long), nipype default value: 1)
Set default maximum number of processes.
argument: ``--xMaxProcess %d``

args: (a unicode string)
Additional parameters to the command
argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str')
argument: ``%s``
value of class 'str', nipype default value: {})

Environment variables

Outputs:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outContinuous</td>
<td>(a pathlike object or string representing an existing file) Continuous depth measurement</td>
</tr>
<tr>
<td>outDiscrete</td>
<td>(a pathlike object or string representing an existing file) Discrete sampled layers</td>
</tr>
<tr>
<td>outLayer</td>
<td>(a pathlike object or string representing an existing file) Layer boundary surfaces</td>
</tr>
</tbody>
</table>

65.1.12 MedicAlgorithmImageCalculator

Link to code

Wraps the executable command "java edu.jhu.ece.iacl.jist.cli.run.edu.jhu.ece.iacl.plugins.utilities.math.MedicAlgorithmImageCalculator"

Title: Image Calculator

Category: Developer Tools

Description: Perform simple image calculator operations on two images. The operations include ‘Add’, ‘Subtract’, ‘Multiply’, and ‘Divide’

Version: 1.10.RC

Documentation-url: http://www.iacl.ece.jhu.edu/

Inputs:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inVolume</td>
<td>(a pathlike object or string representing an existing file) Volume 1</td>
</tr>
<tr>
<td>inVolume2</td>
<td>(a pathlike object or string representing an existing file) Volume 2</td>
</tr>
<tr>
<td>inOperation</td>
<td>('Add' or 'Subtract' or 'Multiply' or 'Divide' or 'Min' or 'Max') Operation</td>
</tr>
<tr>
<td>xPrefExt</td>
<td>('nrrd') Output File Type</td>
</tr>
<tr>
<td>outResult</td>
<td>(a boolean or a pathlike object or string representing a file) Result Volume</td>
</tr>
<tr>
<td>null</td>
<td>(a unicode string) Execution Time</td>
</tr>
<tr>
<td>xDefaultMem</td>
<td>(an integer (int or long)) Set default maximum heap size</td>
</tr>
<tr>
<td>xMaxProcess</td>
<td>(an integer (int or long), nipype default value: 1) Set default maximum number of processes.</td>
</tr>
<tr>
<td>args</td>
<td>(a unicode string)</td>
</tr>
</tbody>
</table>
Additional parameters to the command
argument: '``%s``'
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

outResult: (a pathlike object or string representing an existing file)
Result Volume

65.1.13 MedicAlgorithmLesionToads

Link to code
Wraps the executable command ‘‘java edu.jhu.ece.iacl.jist.cli.run.edu.jhu.ece.iacl.plugins.classification.MedicAlgorithmLesionToads’’.
title: Lesion TOADS
category: Developer Tools
version: 1.9.R
contributor: Navid Shiee (navid.shiee@nih.gov) http://iacl.ece.jhu.edu/~nshiee/

Inputs:

[Optional]
inT1_MPRAGE: (a pathlike object or string representing an existing file)
T1_MPRAGE Image
argument: '``--inT1_MPRAGE %s``'
inT1_SPGR: (a pathlike object or string representing an existing file)
T1_SPGR Image
argument: '``--inT1_SPGR %s``'
inFLAIR: (a pathlike object or string representing an existing file)
FLAIR Image
argument: '``--inFLAIR %s``'
inAtlas: ('With Lesion' or 'No Lesion')
Atlas to Use
argument: '``--inAtlas %s``'
inOutput: ('hard segmentation' or 'hard segmentation+memberships' or 'cruise inputs' or 'dura removal inputs')
Output images
argument: '``--inOutput %s``'
inOutput2: ('true' or 'false')
Output the hard classification using maximum membership (not necessarily topologically correct)
argument: '``--inOutput2 %s``'
inCorrect: ('true' or 'false')
Correct MR field inhomogeneity.
argument: '``--inCorrect %s``'
inOutput3: ('true' or 'false')
Output the estimated inhomogeneity field

(continues on next page)
Argument: `--inOutput3 %s`
inAtlas2: (a pathlike object or string representing an existing file)
   Atlas File - With Lesions
   Argument: `--inAtlas2 %s`
inAtlas3: (a pathlike object or string representing an existing file)
   Atlas File - No Lesion - T1 and FLAIR
   Argument: `--inAtlas3 %s`
inAtlas4: (a pathlike object or string representing an existing file)
   Atlas File - No Lesion - T1 Only
   Argument: `--inAtlas4 %s`
inMaximum: (an integer (int or long))
   Maximum Distance from the interventricular WM boundary to downweight
   the lesion membership to avoid false positives
   Argument: `--inMaximum %d`
inMaximum2: (an integer (int or long))
   Maximum Ventircle Distance
   Argument: `--inMaximum2 %d`
inMaximum3: (an integer (int or long))
   Maximum Intervertricular Distance
   Argument: `--inMaximum3 %d`
inInclude: ('true' or 'false')
   Include lesion in WM class in hard classification
   Argument: `--inInclude %s`
inAtlas5: (a float)
   Controls the effect of the statistical atlas on the segmentation
   Argument: `--inAtlas5 %f`
inSmooting: (a float)
   Controls the effect of neighborhood voxels on the membership
   Argument: `--inSmooting %f`
inMaximum4: (a float)
   Maximum amount of relative change in the energy function considered
   as the convergence criteria
   Argument: `--inMaximum4 %f`
inMaximum5: (an integer (int or long))
   Maximum iterations
   Argument: `--inMaximum5 %d`
inAtlas6: ('rigid' or 'multi_fully_affine')
   Atlas alignment
   Argument: `--inAtlas6 %s`
inConnectivity: ('(26,6)' or '(6,26)' or '(6,18)' or '(18,6)')
   Connectivity (foreground,background)
   Argument: `--inConnectivity %s`
xPrefExt: ('nrrd')
   Output File Type
   Argument: `--xPrefExt %s`
outHard: (a boolean or a pathlike object or string representing a file)
   Hard segmentation
   Argument: `--outHard %s`
outHard2: (a boolean or a pathlike object or string representing a file)
   Hard segmentation from memberships
   Argument: `--outHard2 %s`
outInhomogeneity: (a boolean or a pathlike object or string representing a file)
   Inhomogeneity Field
   Argument: `--outInhomogeneity %s`
Membership Functions

outMembership: (a boolean or a pathlike object or string representing a file)

Lesion Segmentation

outLesion: (a boolean or a pathlike object or string representing a file)

Sulcal CSF Membership

outSulcal: (a boolean or a pathlike object or string representing a file)

Cortical GM Membership

outCortical: (a boolean or a pathlike object or string representing a file)

Filled WM Membership

outFilled: (a boolean or a pathlike object or string representing a file)

WM Mask

outWM: (a boolean or a pathlike object or string representing a file)

Execution Time

null: (a unicode string)

Set default maximum heap size

xDefaultMem: (an integer (int or long))

Set default maximum number of processes.

xMaxProcess: (an integer (int or long), nipype default value: 1)

Additional parameters to the command

args: (a unicode string)

Environment variables

Outputs:

outHard: (a pathlike object or string representing an existing file)

Hard segmentation

outHard2: (a pathlike object or string representing an existing file)

Hard segmentation from memberships

outInhomogeneity: (a pathlike object or string representing an existing file)

Inhomogeneity Field

outMembership: (a pathlike object or string representing an existing file)

Membership Functions

outLesion: (a pathlike object or string representing an existing file)

Lesion Segmentation

outSulcal: (a pathlike object or string representing an existing file)
Sulcal CSF Membership
outCortical: (a pathlike object or string representing an existing file)
Cortical GM Membership
outFilled: (a pathlike object or string representing an existing file)
Filled WM Membership
outWM: (a pathlike object or string representing an existing file)

65.1.14 MedicAlgorithmMipavReorient

Link to code
Wraps the executable command: 
```
java edu.jhu.ece.iacl.jist.cli.run.edu.jhu.ece.iacl.plugins.utilities.volume.MedicAlgorithmMipavReorient
```

title: Reorient Volume
category: Developer Tools
description: Reorient a volume to a particular anatomical orientation.
version: .alpha

Inputs:

[Optional]
inSource: (a list of items which are a pathlike object or string representing a file)
Source
argument: `--inSource %s`
inTemplate: (a pathlike object or string representing an existing file)
Template
argument: `--inTemplate %s`
inNew: ('Dicom axial' or 'Dicom coronal' or 'Dicom sagittal' or 'User defined')
New image orientation
argument: `--inNew %s`
inUser: ('Unknown' or 'Patient Right to Left' or 'Patient Left to Right' or 'Patient Posterior to Anterior' or 'Patient Anterior to Posterior' or 'Patient Inferior to Superior' or 'Patient Superior to Inferior')
User defined X-axis orientation (image left to right)
argument: `--inUser %s`
inUser2: ('Unknown' or 'Patient Right to Left' or 'Patient Left to Right' or 'Patient Posterior to Anterior' or 'Patient Anterior to Posterior' or 'Patient Inferior to Superior' or 'Patient Superior to Inferior')
User defined Y-axis orientation (image top to bottom)
argument: `--inUser2 %s`
inUser3: ('Unknown' or 'Patient Right to Left' or 'Patient Left to Right' or 'Patient Posterior to Anterior' or 'Patient Anterior to Posterior' or 'Patient Inferior to Superior' or 'Patient Superior to Inferior')
User defined Z-axis orientation (into the screen)
argument: `--inUser3 %s`
inUser4: ('Axial' or 'Coronal' or 'Sagittal' or 'Unknown')
User defined Image Orientation
argument: `--inUser4 %s`
inInterpolation: ('Nearest Neighbor' or 'Trilinear' or 'Bspline 3rd order' or 'Bspline 4th order' or 'Cubic Lagrangian' or 'Quintic Lagrangian' or 'Heptic Lagrangian' or 'Windowed Sinc')
Interpolation argument: ```--inInterpolation %s```
inResolution: ('Unchanged' or 'Finest cubic' or 'Coarsest cubic' or 'Same as template')
Resolution argument: ```--inResolution %s```
xPrefExt: ('nrrd')
Output File Type argument: ```--xPrefExt %s```
outReoriented: (a list of items which are a pathlike object or string representing a file)
Reoriented Volume argument: ```--outReoriented %s```
null: (a unicode string)
Execution Time argument: ```--null %s```
xDefaultMem: (an integer (int or long))
Set default maximum heap size argument: ```--xDefaultMem %d```
xMaxProcess: (an integer (int or long), nipype default value: 1)
Set default maximum number of processes. argument: ```--xMaxProcess %d```
args: (a unicode string)
Additional parameters to the command argument: ```%s```
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

None

65.1.15 MedicAlgorithmN3

Link to code
Wraps the executable command `"java edu.jhu.ece.iacJistj.SlicRun jhu.edu.ece.iacJistj.Plugins.classification.MedicAlgorithmN3"
.title: N3 Correction
category: Developer Tools
description: Non-parametric Intensity Non-uniformity Correction, N3, originally by J.G. Sled.
version: 1.8.R
Inputs:

[Optional]
inInput: (a pathlike object or string representing an existing file)
Input Volume argument: ```--inInput %s```
inSignal: (a float)
Default = min + 1, Values at less than threshold are treated as part of the background argument: ```--inSignal %f```
inMaximum: (an integer (int or long))
  Maximum number of Iterations
  argument: ``--inMaximum %d``

inEnd: (a float)
  Usually 0.01-0.00001, The measure used to terminate the iterations
  is the coefficient of variation of change in field estimates between
  successive iterations.
  argument: ``--inEnd %f``

inField: (a float)
  Characteristic distance over which the field varies. The distance
  between adjacent knots in bspline fitting with at least 4 knots
  going in every dimension. The default in the dialog is one third the
  distance (resolution * extents) of the smallest dimension.
  argument: ``--inField %f``

inSubsample: (a float)
  Usually between 1-32, The factor by which the data is subsampled to
  a lower resolution in estimating the slowly varying non-uniformity
  field. Reduce sampling in the finest sampling direction by the
  shrink factor.
  argument: ``--inSubsample %f``

inKernel: (a float)
  Usually between 0.05-0.50, Width of deconvolution kernel used to
  sharpen the histogram. Larger values give faster convergence while
  smaller values give greater accuracy.
  argument: ``--inKernel %f``

inWeiner: (a float)
  Usually between 0.0-1.0
  argument: ``--inWeiner %f``

inAutomatic: ('true' or 'false')
  If true determines the threshold by histogram analysis. If true a
  VOI cannot be used and the input threshold is ignored.
  argument: ``--inAutomatic %s``

xPrefExt: ('nrrd')
  Output File Type
  argument: ``--xPrefExt %s``

outInhomogeneity: (a boolean or a pathlike object or string
  representing a file)
  Inhomogeneity Corrected Volume
  argument: ``--outInhomogeneity %s``

outInhomogeneity2: (a boolean or a pathlike object or string
  representing a file)
  Inhomogeneity Field
  argument: ``--outInhomogeneity2 %s``

null: (a unicode string)
  Execution Time
  argument: ``--null %s``

xDefaultMem: (an integer (int or long))
  Set default maximum heap size
  argument: ``--xDefaultMem %d``

xMaxProcess: (an integer (int or long), nipype default value: 1)
  Set default maximum number of processes.
  argument: ``--xMaxProcess %d``

args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {}}

Environment variables

Outputs:

outInhomogeneity: (a pathlike object or string representing an existing file)
Inhomogeneity Corrected Volume
outInhomogeneity2: (a pathlike object or string representing an existing file)
Inhomogeneity Field

65.1.16 MedicAlgorithmSPECTRE2010

Link to code
Wraps the executable command ‘"java edu.jhu.ece.iacl.jist.cli.run edu.jhu.ece.iacl.plugins.segmentation.skull_strip.MedicAlgorithmSPECTRE2010"‘.

title: SPECTRE 2010
category: Developer Tools
description: Simple Paradigm for Extra-Cranial Tissue REMoval
Algorithm Version: 1.6 GUI Version: 1.10
version: 1.6.R
documentation-url: http://www.iacl.ece.jhu.edu/
contributor: Aaron Carass (aaron_carass@jhu.edu) http://www.iacl.ece.jhu.edu/ Hanlin Wan (hanlin-wan@gmail.com)

Inputs:

[Optional]
inInput: (a pathlike object or string representing an existing file)
Input volume to be skullstripped.
argument: "--inInput %s"
inAtlas: (a pathlike object or string representing an existing file)
SPECTRE atlas description file. A text file enumerating atlas files and landmarks.
argument: "--inAtlas %s"
inInitial: (an integer (int or long))
Erosion of the initial mask, which is based on the probability mask and the classification. The initial mask is output as the d0 volume at the conclusion of SPECTRE.
argument: "--inInitial %d"
inImage: ("T1_SPGR" or 'T1_ALT' or 'T1_MPRAGE' or 'T2' or 'FLAIR')
Set the image modality. MP-RAGE is recommended for most T1 sequence images.
argument: "--inImage %s"
inOutput: ('true' or 'false')
Determines if the output results are transformed back into the space of the original input image.
argument: "--inOutput %s"
inFind: ('true' or 'false')
Find Midsagittal Plane
argument: ```--inFind %s```  

inRun: ('true' or 'false')  
Run Smooth Brain Mask  
argument: ```--inRun %s```  

inResample: ('true' or 'false')  
Determines if the data is resampled to be isotropic during the processing.  
argument: ```--inResample %s```  

inInitial2: (a float)  
Initial probability threshold  
argument: ```--inInitial2 %f```  

inMinimum: (a float)  
Minimum probability threshold  
argument: ```--inMinimum %f```  

inMMC: (an integer (int or long))  
The size of the dilation step within the Modified Morphological Closing.  
argument: ```--inMMC %d```  

inMMC2: (an integer (int or long))  
The size of the erosion step within the Modified Morphological Closing.  
argument: ```--inMMC2 %d```  

inInhomogeneity: ('true' or 'false')  
Set to false by default, this parameter will make FANTASM try to do inhomogeneity correction during it’s iterative cycle.  
argument: ```--inInhomogeneity %s```  

inSmoothing: (a float)  
argument: ```--inSmoothing %f```  

inBackground: (a float)  
argument: ```--inBackground %f```  

inOutput2: ('true' or 'false')  
Output Plane?  
argument: ```--inOutput2 %s```  

inOutput3: ('true' or 'false')  
Output Split-Halves?  
argument: ```--inOutput3 %s```  

inOutput4: ('true' or 'false')  
Output Segmentation on Plane?  
argument: ```--inOutput4 %s```  

inDegrees: ('Rigid - 6' or 'Global rescale - 7' or 'Specific rescale - 9' or 'Affine - 12')  
Degrees of freedom  
argument: ```--inDegrees %s```  

inCost: ('Correlation ratio' or 'Least squares' or 'Normalized cross correlation' or 'Normalized mutual information')  
Cost function  
argument: ```--inCost %s```  

inRegistration: ('Trilinear' or 'Bspline 3rd order' or 'Bspline 4th order' 
or 'Cubic Lagrangian' or 'Quintic Lagrangian' or 'Heptic Lagrangian' or 'Windowed sinc')  
Registration interpolation  
argument: ```--inRegistration %s```  

inOutput5: ('Trilinear' or 'Bspline 3rd order' or 'Bspline 4th order' 
or 'Cubic Lagrangian' or 'Quintic Lagrangian' or 'Heptic Lagrangian' or 'Windowed sinc' or 'Nearest Neighbor')  
Output interpolation  
argument: ```--inOutput5 %s```
inApply: ('All' or 'X' or 'Y' or 'Z')
  Apply rotation
  argument: `--inApply %s`

inMinimum2: (a float)
  Minimum angle
  argument: `--inMinimum %f`

inMaximum: (a float)
  Maximum angle
  argument: `--inMaximum %f`

inCoarse: (a float)
  Coarse angle increment
  argument: `--inCoarse %f`

inFine: (a float)
  Fine angle increment
  argument: `--inFine %f`

inMultiple: (an integer (int or long))
  Multiple of tolerance to bracket the minimum
  argument: `--inMultiple %d`

inNumber: (an integer (int or long))
  Number of iterations
  argument: `--inNumber %d`

inNumber2: (an integer (int or long))
  Number of minima from Level 8 to test at Level 4
  argument: `--inNumber2 %d`

inUse: ('true' or 'false')
  Use the max of the min resolutions of the two datasets when
  resampling
  argument: `--inUse %s`

inSubsample: ('true' or 'false')
  Subsample image for speed
  argument: `--inSubsample %s`

inSkip: ('true' or 'false')
  Skip multilevel search (Assume images are close to alignment)
  argument: `--inSkip %s`

inMultithreading: ('true' or 'false')
  Set to false by default, this parameter controls the multithreaded
  behavior of the linear registration.
  argument: `--inMultithreading %s`

xPrefExt: ('nrrd')
  Output File Type
  argument: `--xPrefExt %s`

outOriginal: (a boolean or a pathlike object or string representing a
file)
  If Output in Original Space Flag is true then outputs the original
input volume. Otherwise outputs the axially reoriented input volume.
  argument: `--outOriginal %s`

outStripped: (a boolean or a pathlike object or string representing a
file)
  Skullstripped result of the input volume with just the brain.
  argument: `--outStripped %s`

outMask: (a boolean or a pathlike object or string representing a
file)
  Binary Mask of the skullstripped result with just the brain
  argument: `--outMask %s`

outPrior: (a boolean or a pathlike object or string representing a
file)
  Probability prior from the atlas registrations
argument: `--outPrior %s`

outFANTASM: (a boolean or a pathlike object or string representing a file)
Tissue classification of of the whole input volume.
argument: `--outFANTASM %s`

outd0: (a boolean or a pathlike object or string representing a file)
Initial Brainmask
argument: `--outd0 %s`

outMidsagittal: (a boolean or a pathlike object or string representing a file)
Plane dividing the brain hemispheres
argument: `--outMidsagittal %s`

outSplitHalves: (a boolean or a pathlike object or string representing a file)
Skullstripped mask of the brain with the hemispheres divided.
argument: `--outSplitHalves %s`

outSegmentation: (a boolean or a pathlike object or string representing a file)
2D image showing the tissue classification on the midsagittal plane
argument: `--outSegmentation %s`

null: (a unicode string)
Execution Time
argument: `--null %s`

xDefaultMem: (an integer (int or long))
Set default maximum heap size
argument: `-xDefaultMem %d`

xMaxProcess: (an integer (int or long), nipype default value: 1)
Set default maximum number of processes.
argument: `-xMaxProcess %d`

args: (a unicode string)
Additional parameters to the command
argument: `%%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

outOriginal: (a pathlike object or string representing an existing file)
If Output in Original Space Flag is true then outputs the original input volume. Otherwise outputs the axially reoriented input volume.

outStripped: (a pathlike object or string representing an existing file)
Skullstripped result of the input volume with just the brain.

outMask: (a pathlike object or string representing an existing file)
Binary Mask of the skullstripped result with just the brain.

outPrior: (a pathlike object or string representing an existing file)
Probability prior from the atlas registrations

outFANTASM: (a pathlike object or string representing an existing file)
Tissue classification of of the whole input volume.

outd0: (a pathlike object or string representing an existing file)
Initial Brainmask

outMidsagittal: (a pathlike object or string representing an existing file)

(continues on next page)
Plane dividing the brain hemispheres

**outSplitHalves**: (a pathlike object or string representing an existing file)

Skullstripped mask of the brain with the hemispheres divided.

**outSegmentation**: (a pathlike object or string representing an existing file)

2D image showing the tissue classification on the midsagittal plane

---

### 65.1.17 **MedicAlgorithmThresholdToBinaryMask**

**Link to code**

Wraps the executable command `"java edu.jhu.ece.iacl.jist.cli.run edu.jhu.ece.iacl.plugins.utilities.volume.MedicAlgorithmThresholdToBinaryMask"`.

**title**: Threshold to Binary Mask  
**category**: Developer Tools  
**description**: Given a volume and an intensity range create a binary mask for values within that range.  
**version**: 1.2.RC  
**documentation-url**: [http://www.iacl.ece.jhu.edu/](http://www.iacl.ece.jhu.edu/)

**Inputs:**

[Optional]

**inLabel**: (a list of items which are a pathlike object or string representing a file)

Input volumes

argument: `"--inLabel %s"`

**inMinimum**: (a float)

Minimum threshold value.

argument: `"--inMinimum %f"`

**inMaximum**: (a float)

Maximum threshold value.

argument: `"--inMaximum %f"

**inUse**: ('true' or 'false')

Use the images max intensity as the max value of the range.

argument: `"--inUse %s"

**xPrefExt**: ('nrrd')

Output File Type

argument: `"--xPrefExt %s"

**outBinary**: (a list of items which are a pathlike object or string representing a file)

Binary Mask

argument: `"--outBinary %s"

**null**: (a unicode string)

Execution Time

argument: `"--null %s"

**xDefaultMem**: (an integer (int or long))

Set default maximum heap size

argument: `"--xDefaultMem %d"

**xMaxProcess**: (an integer (int or long), nipype default value: 1)

Set default maximum number of processes.

argument: `"--xMaxProcess %d"

**args**: (a unicode string)

Additional parameters to the command

argument: `"%s"

**environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str')

(continues on next page)
value of class 'str', nipype default value: {}}
Environment variables

Outputs:

None

### 65.1.18 RandomVol

**Link to code**
Wraps the executable command "java edu.jhu.ece.iacl.jist.cli.run.edu.jhu.bme.smile.demo.RandomVol".

**title:** Random Volume Generator

**category:** Developer Tools

**description:** Generate a random scalar volume.

**version:** 1.12.RC

**documentation-url:** http://www.nitrc.org/projects/jist/

**Inputs:**

```python
[Optional]
inSize: (an integer (int or long))
    Size of Volume in X direction
    argument: `--inSize %d`
inSize2: (an integer (int or long))
    Size of Volume in Y direction
    argument: `--inSize2 %d`
inSize3: (an integer (int or long))
    Size of Volume in Z direction
    argument: `--inSize3 %d`
inSize4: (an integer (int or long))
    Size of Volume in t direction
    argument: `--inSize4 %d`
inStandard: (an integer (int or long))
    Standard Deviation for Normal Distribution
    argument: `--inStandard %d`
inLambda: (a float)
    Lambda Value for Exponential Distribution
    argument: `--inLambda %f`
inMaximum: (an integer (int or long))
    Maximum Value
    argument: `--inMaximum %d`
inMinimum: (an integer (int or long))
    Minimum Value
    argument: `--inMinimum %d`
inField: ('Uniform' or 'Normal' or 'Exponential')
    Field
    argument: `--inField %s`
xPrefExt: ('nrrd')
    Output File Type
    argument: `--xPrefExt %s`
outRand1: (a boolean or a pathlike object or string representing a file)
    Rand1
    argument: `--outRand1 %s`
null: (a unicode string)
    Execution Time
    argument: `--null %s`
xDefaultMem: (an integer (int or long))
   Set default maximum heap size
   argument: `--xDefaultMem %d`

xMaxProcess: (an integer (int or long), nipype default value: 1)
   Set default maximum number of processes.
   argument: `--xMaxProcess %d`

args: (a unicode string)
   Additional parameters to the command
   argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: { })
   Environment variables

Outputs:

outRand1: (a pathlike object or string representing an existing file)
   Rand1
66.1 interfaces.mixins.reporting

66.1.1 ReportCapableInterface

Link to code
Mixin to enable reporting for Nipype interfaces
Inputs:

None

Outputs:

None
67.1 interfaces.mne.base

67.1.1 WatershedBEM

Link to code
Wraps the executable command `mne_watershed_bem`. Uses `mne_watershed_bem` to get information from dicom directories.

Examples

```python
>>> from nipype.interfaces.mne import WatershedBEM
>>> bem = WatershedBEM()
>>> bem.inputs.subject_id = 'subj1'
>>> bem.inputs.subjects_dir = '.
>>> bem.cmdline
'mne_watershed_bem --overwrite --subject subj1 --volume T1'
>>> bem.run() # doctest: +SKIP
```

Inputs:

- **subject_id**: (a unicode string)
  - Subject ID (must have a complete Freesurfer directory)
  - argument: `--subject %s`
- **subjects_dir**: (a pathlike object or string representing an existing directory, nipype default value: <undefined>)
  - Path to Freesurfer subjects directory
- **volume**: ('T1' or 'aparc+aseg' or 'aseg' or 'brain' or 'orig' or 'brainmask' or 'ribbon', nipype default value: T1)
  - The volume from the "mri" directory to use (defaults to T1)
  - argument: `--volume %s`
- **overwrite**: (a boolean, nipype default value: True)
  - Overwrites the existing files
  - argument: `--overwrite`

(continues on next page)
atlas_mode: (a boolean)
  Use atlas mode for registration (default: no rigid alignment)
  argument: `--atlas`
args: (a unicode string)
  Additional parameters to the command
  argument: `--%s`
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})  
Environment variables

Outputs:

mesh_files: (a list of items which are a pathlike object or string
  representing an existing file)
  Paths to the output meshes (brain, inner skull, outer skull, outer
  skin)
brain_surface: (a pathlike object or string representing an existing
  file)
  Brain surface (in Freesurfer format)
inner_skull_surface: (a pathlike object or string representing an
  existing file)
  Inner skull surface (in Freesurfer format)
outer_skull_surface: (a pathlike object or string representing an
  existing file)
  Outer skull surface (in Freesurfer format)
outer_skin_surface: (a pathlike object or string representing an
  existing file)
  Outer skin surface (in Freesurfer format)
fif_file: (a pathlike object or string representing an existing file)
  "fif" format file for EEG processing in MNE
cor_files: (a list of items which are a pathlike object or string
  representing an existing file)
  "COR" format files
68.1 interfaces.mrtrix

68.1.1 MRTrix2TrackVis

Link to code

Converts MRtrix (.tck) tract files into TrackVis (.trk) format using functions from dipy Example ~~~~~~~

```python
>>> import nipype.interfaces.mrtrix as mrt
>>> tck2trk = mrt.MRTrix2TrackVis()
>>> tck2trk.inputs.in_file = 'dwi_CSD_tracked.tck'
>>> tck2trk.inputs.image_file = 'diffusion.nii'
>>> tck2trk.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
`in_file`: (a pathlike object or string representing an existing file)
The input file for the tracks in MRTrix (.tck) format

[Optional]
`image_file`: (a pathlike object or string representing an existing file)
The image the tracks were generated from

`matrix_file`: (a pathlike object or string representing an existing file)
A transformation matrix to apply to the tracts after they have been generated (from FLIRT - affine transformation from image_file to registration_image_file)

`registration_image_file`: (a pathlike object or string representing an existing file)
The final image the tracks should be registered to.

`out_filename`: (a pathlike object or string representing a file, nipype default value: converted.trk)
The output filename for the tracks in TrackVis (.trk) format

Outputs:

`out_file`: (a pathlike object or string representing an existing file)

68.1.2 read_mrtrix_header()

Link to code
68.3 \texttt{read_mrtrix_streamlines()}

Link to code

68.4 \texttt{read_mrtrix_tracks()}

Link to code

68.5 \texttt{transform_to_affine()}

Link to code

68.2 \texttt{interfaces.mrtrix.preprocess}

68.2.1 \texttt{DWI2Tensor}

Link to code
Wraps the executable command \texttt{dwi2tensor}.
Converts diffusion-weighted images to tensor images.

Example

\begin{verbatim}
>>> import nipype.interfaces.mrtrix as mrt
>>> dwi2tensor = mrt.DWI2Tensor()
>>> dwi2tensor.inputs.in_file = 'dwi.mif'
>>> dwi2tensor.inputs.encoding_file = 'encoding.txt'
>>> dwi2tensor.cmdline
'dwi2tensor -grad encoding.txt dwi.mif dwi_tensor.mif'
>>> dwi2tensor.run() # doctest: +SKIP
\end{verbatim}

Inputs:

[Mandatory]
\begin{itemize}
  \item \texttt{in_file}: (a list of items which are a pathlike object or string representing an existing file)
    \begin{itemize}
      \item Diffusion-weighted images
      \item argument: ``\$s``, position: -2
    \end{itemize}
\end{itemize}

[Optional]
\begin{itemize}
  \item \texttt{out_filename}: (a pathlike object or string representing a file)
    \begin{itemize}
      \item Output tensor filename
      \item argument: ``\$s``, position: -1
    \end{itemize}
  \item \texttt{encoding_file}: (a pathlike object or string representing a file)
    \begin{itemize}
      \item Encoding file supplied as a 4xN text file with each line in the format \texttt{[ X Y Z b ]}, where \texttt{[ X Y Z ]} describe the direction of the applied gradient, and \texttt{b} gives the \textbf{b}-value in units (1000 s/mm^2). See \texttt{FSL2MRTRix()}
      \item argument: ``-grad \$s``
    \end{itemize}
  \item \texttt{ignore_slice_by_volume}: (a list of from 2 to 2 items which are an integer (int or long))
    \begin{itemize}
      \item Requires two values (i.e. [34 1] for [Slice Volume] Ignores the image slices specified when computing the tensor. Slice here means the z coordinate of the slice to be ignored.
      \item argument: ```-ignoreslices \$s```, position: 2
    \end{itemize}
  \item \texttt{ignore_volumes}: (a list of at least 1 items which are an integer (int or long))
    \begin{itemize}
  \end{itemize}
\end{itemize}

(continues on next page)
Requires two values (i.e. [2 5 6] for [Volumes]) Ignores the image volumes specified when computing the tensor.

argument: `--ignorevolumes %s`, position: 2

quiet: (a boolean)
Do not display information messages or progress status.
argument: `--quiet`, position: 1

debug: (a boolean)
Display debugging messages.
argument: `--debug`, position: 1

args: (a unicode string)
Additional parameters to the command
argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

Environment variables

Outputs:

tensor: (a pathlike object or string representing an existing file)
path/name of output diffusion tensor image

68.2.2 Erode

Link to code
Wraps the executable command erode.
Erode (or dilates) a mask (i.e. binary) image

Example

```python
>>> import nipype.interfaces.mrtrix as mrt
>>> erode = mrt.Erode()
>>> erode.inputs.in_file = 'mask.mif'
>>> erode.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
Input mask image to be eroded
argument: `%s`, position: -2

[Optional]
out_filename: (a pathlike object or string representing a file)
Output image filename
argument: `%s`, position: -1
number_of_passes: (an integer (int or long))
the number of passes (default: 1)
argument: `-npass %s`
dilate: (a boolean)
Perform dilation rather than erosion
argument: `--dilate`, position: 1
quiet: (a boolean)
Do not display information messages or progress status.
argument: `--quiet`, position: 1
debug: (a boolean)

(continues on next page)
Display debugging messages.
argument: ``--debug``, position: 1

args: (a unicode string)
Additional parameters to the command
argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:
out_file: (a pathlike object or string representing an existing file)
the output image

68.2.3 GenerateWhiteMatterMask

Link to code
Wraps the executable command gen_WM_mask.
Generates a white matter probability mask from the DW images.

Example

```python
>>> import nipype.interfaces.mrtrix as mrt
>>> genWM = mrt.GenerateWhiteMatterMask()
>>> genWM.inputs.in_file = 'dwi.mif'
>>> genWM.inputs.encoding_file = 'encoding.txt'
>>> genWM.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
Diffusion-weighted images
argument: ``%s``, position: -3

binary_mask: (a pathlike object or string representing an existing file)
Binary brain mask
argument: ``%s``, position: -2

encoding_file: (a pathlike object or string representing an existing file)
Gradient encoding, supplied as a 4xN text file with each line is in
the format [ X Y Z b ], where [ X Y Z ] describe the direction of
the applied gradient, and b gives the b-value in units (1000
s/mm^2). See FSL2MRTrix
argument: ``--grad %s``, position: 1

[Optional]
out_WMProb_filename: (a pathlike object or string representing a file)
Output WM probability image filename
argument: ``%s``, position: -1

noise_level_margin: (a float)
Specify the width of the margin on either side of the image to be
used to estimate the noise level (default = 10)
argument: ``--margin %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ```%s```
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})
    Environment variables

Outputs:

WMprobabilitymap: (a pathlike object or string representing an
    existing file)
    WMprobabilitymap

68.2.4 MRConvert

Link to code

Wraps the executable command mrconvert.

Perform conversion between different file types and optionally extract a subset of the input image.

If used correctly, this program can be a very useful workhorse. In addition to converting images between
    different formats, it can be used to extract specific studies from a data set, extract a specific region of interest,
    flip the images, or to scale the intensity of the images.

Example

```python
>>> import nipype.interfaces.mrtrix as mrt
>>> mrcconvert = mrt.MRConvert()
>>> mrcconvert.inputs.in_file = 'dwi_FA.mif'
>>> mrcconvert.inputs.out_filename = 'dwi_FA.nii'
>>> mrcconvert.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
    voxel-order data filename
    argument: ```%s```, position: -2

[Optional]
out_filename: (a pathlike object or string representing a file)
    Output filename
    argument: ```%s```, position: -1
extract_at_axis: (1 or 2 or 3)
    "Extract data only at the coordinates specified. This option
    specifies the Axis. Must be used in conjunction with
    extract_at_coordinate.
    argument: ```-coord %s```, position: 1
extract_at_coordinate: (a list of from 1 to 3 items which are a
    float)
    "Extract data only at the coordinates specified. This option
    specifies the coordinates. Must be used in conjunction with
    extract_at_axis. Three comma-separated numbers giving the size of
    each voxel in mm.
    argument: ```%s```, position: 2
voxel_dims: (a list of from 3 to 3 items which are a float)
    Three comma-separated numbers giving the size of each voxel in mm.
    (continues on next page)
argument: `'-vox %s'`, position: 3
output_datatype: (`'nii' or 'float' or 'char' or 'short' or 'int' or 'long' or 'double'`)
    "i.e. Bfloat". Can be "char", "short", "int", "long", "float" or "double"
argument: `'-output %s'`, position: 2
extension: (`'mif' or 'nii' or 'float' or 'char' or 'short' or 'int'
or 'long' or 'double', nipype default value: mif)
    "i.e. Bfloat". Can be "char", "short", "int", "long", "float" or "double"
layout: (`'nii' or 'float' or 'char' or 'short' or 'int' or 'long' or 'double'`)
specify the layout of the data in memory. The actual layout produced will depend on whether the output image format can support it.
argument: `'-output %s'`, position: 2
resample: (a float)
    Apply scaling to the intensity values.
argument: `'-scale %d'`, position: 3
offset_bias: (a float)
    Apply offset to the intensity values.
argument: `'-scale %d'`, position: 3
replace_NaN_with_zero: (a boolean)
    Replace all NaN values with zero.
argument: `'-zero'`, position: 3
prs: (a boolean)
    Assume that the DW gradients are specified in the PRS frame (Siemens DICOM only).
argument: `'-prs'`, position: 3
args: (a unicode string)
    Additional parameters to the command
argument: `'%s'`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: { })
Environment variables

Outputs:

converted: (a pathlike object or string representing an existing file)
    path/name of 4D volume in voxel order

68.2.5 MRMultiply

Link to code
Wraps the executable command mrmult.
Multiplies two images.

Example

```python
>>> import nipype.interfaces.mrtrix as mrt
>>> MRmult = mrt.MRMultiply()
>>> MRmult.inputs.in_files = ['dwi.mif', 'dwi_WMProb.mif']
>>> MRmult.run() # doctest: +SKIP
```

Inputs:
[Mandatory]
in_files: (a list of items which are a pathlike object or string representing an existing file)
   Input images to be multiplied
   argument: ``%s``, position: -2

[Optional]
out_filename: (a pathlike object or string representing a file)
   Output image filename
   argument: ``%s``, position: -1
quiet: (a boolean)
   Do not display information messages or progress status.
   argument: ``-quiet``, position: 1
debug: (a boolean)
   Display debugging messages.
   argument: ``-debug`` , position: 1
args: (a unicode string)
   Additional parameters to the command
   argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
   Environment variables

Outputs:
out_file: (a pathlike object or string representing an existing file)
   the output image of the multiplication

68.2.6 MRTransform

Link to code
Wraps the executable command `mrtransform`.
Apply spatial transformations or reslice images

Example

```python
>>> MRxform = MRTransform()
>>> MRxform.inputs.in_files = 'anat_coreg.mif'
>>> MRxform.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
in_files: (a list of items which are a pathlike object or string representing an existing file)
   Input images to be transformed
   argument: ``%s``, position: -2

[Optional]
out_filename: (a pathlike object or string representing a file)
   Output image
   argument: ``%s``, position: -1
invert: (a boolean)
   Invert the specified transform before using it
   argument: ``-inverse`` , position: 1
replace_transform: (a boolean)
replace the current transform by that specified, rather than applying it to the current transform
argument: `'-replace'`, position: 1

**transformation_file**: (a pathlike object or string representing an existing file)
The transform to apply, in the form of a 4x4 ascii file.
argument: `'-transform %s'`, position: 1

**template_image**: (a pathlike object or string representing an existing file)
Reslice the input image to match the specified template image.
argument: `'-template %s'`, position: 1

**reference_image**: (a pathlike object or string representing an existing file)
in case the transform supplied maps from the input image onto a reference image, use this option to specify the reference. Note that this implicitly sets the -replace option.
argument: `'-reference %s'`, position: 1

**flip_x**: (a boolean)
assume the transform is supplied assuming a coordinate system with the x-axis reversed relative to the MRtrix convention (i.e. x increases from right to left). This is required to handle transform matrices produced by FSL's FLIRT command. This is only used in conjunction with the -reference option.
argument: `'-flipx'`, position: 1

**quiet**: (a boolean)
Do not display information messages or progress status.
argument: `'-quiet'`, position: 1

**debug**: (a boolean)
Display debugging messages.
argument: `'-debug'`, position: 1

**args**: (a unicode string)
Additional parameters to the command
argument: `'%s'`

**environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

## 68.2.7 MRTrixViewer

Link to code
Wraps the executable command `mrview`.
Loads the input images in the MRTrix Viewer.

### Example

```python
>>> import nipype.interfaces.mrtrix as mrt
>>> MRview = mrt.MRTrixViewer()
>>> MRview.inputs.in_files = 'dwi.mif'
>>> MRview.run()  # doctest: +SKIP
```

**Outputs:**

**out_file**: (a pathlike object or string representing an existing file)
the output image of the transformation

---

1152 Chapter 68. interfaces.mrtrix
in_files: (a list of items which are a pathlike object or string representing an existing file)
   Input images to be viewed
   argument: `--%s`, position: -2

[Optional]
quiet: (a boolean)
   Do not display information messages or progress status.
   argument: `--quiet`, position: 1
debug: (a boolean)
   Display debugging messages.
   argument: `--debug`, position: 1
args: (a unicode string)
   Additional parameters to the command
   argument: `--%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

Environment variables

Outputs:

None

68.2.8 MedianFilter3D

Link to code
Wraps the executable command `median3D`. Smooth images using a 3x3x3 median filter.

Example

```python
>>> import nipype.interfaces.mrtrix as mrt
>>> median3d = mrt.MedianFilter3D()
>>> median3d.inputs.in_file = 'mask.mif'
>>> median3d.run()  # doctest: +SKIP
```

Inputs:

[Optional]
out_filename: (a pathlike object or string representing a file)
   Output image filename
   argument: `--%s`, position: -1
quiet: (a boolean)
   Do not display information messages or progress status.
   argument: `--quiet`, position: 1
debug: (a boolean)
   Display debugging messages.
   argument: `--debug`, position: 1
args: (a unicode string)
   Additional parameters to the command

(continues on next page)
argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
the output image

68.2.9 Tensor2ApparentDiffusion

Link to code
Wraps the executable command tensor2ADC.
Generates a map of the apparent diffusion coefficient (ADC) in each voxel

Example

```python
>>> import nipype.interfaces.mrtrix as mrt
>>> tensor2ADC = mrt.Tensor2ApparentDiffusion()
>>> tensor2ADC.inputs.in_file = 'dwi_tensor.mif'
>>> tensor2ADC.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
Diffusion tensor image
argument: ``%s``, position: -2

[Optional]
out_filename: (a pathlike object or string representing a file)
Output Fractional Anisotropy filename
argument: ``%s`` , position: -1
quiet: (a boolean)
Do not display information messages or progress status.
argument: ``-quiet`` , position: 1
debug: (a boolean)
Display debugging messages.
argument: ``-debug`` , position: 1
args: (a unicode string)
Additional parameters to the command
argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

ADC: (a pathlike object or string representing an existing file)
the output image of the major eigenvectors of the diffusion tensor image.
68.2.10  Tensor2FractionalAnisotropy

Link to code
Wraps the executable command tensor2FA.
Generates a map of the fractional anisotropy in each voxel.

Example

```python
>>> import nipype.interfaces.mrtrix as mrt
>>> tensor2FA = mrt.Tensor2FractionalAnisotropy()
>>> tensor2FA.inputs.in_file = 'dwi_tensor.mif'
>>> tensor2FA.run()  # doctest: +SKIP
```

Inputs:

<table>
<thead>
<tr>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_file</td>
<td>Diffusion tensor image</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out_filename</td>
<td>Output Fractional Anisotropy filename</td>
</tr>
<tr>
<td>quiet</td>
<td>Do not display information messages or progress status.</td>
</tr>
<tr>
<td>debug</td>
<td>Display debugging messages.</td>
</tr>
<tr>
<td>args</td>
<td>Additional parameters to the command</td>
</tr>
<tr>
<td>environ</td>
<td>Environment variables</td>
</tr>
</tbody>
</table>

Outputs:

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FA: the output image of the major eigenvectors of the diffusion tensor image.</td>
</tr>
</tbody>
</table>

68.2.11  Tensor2Vector

Link to code
Wraps the executable command tensor2vector.
Generates a map of the major eigenvectors of the tensors in each voxel.

Example

```python
>>> import nipype.interfaces.mrtrix as mrt
>>> tensor2vector = mrt.Tensor2Vector()
>>> tensor2vector.inputs.in_file = 'dwi_tensor.mif'
>>> tensor2vector.run()  # doctest: +SKIP
```

Inputs:
[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  Diffusion tensor image
  argument: ``%s``, position: -2

[Optional]
out_filename: (a pathlike object or string representing a file)
  Output vector filename
  argument: ``%s``, position: -1
quiet: (a boolean)
  Do not display information messages or progress status.
  argument: ``-quiet`` , position: 1
debug: (a boolean)
  Display debugging messages.
  argument: ``-debug`` , position: 1
args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {}))

Environment variables

Outputs:

vector: (a pathlike object or string representing an existing file)
  the output image of the major eigenvectors of the diffusion tensor image.

68.2.12 Threshold

Link to code
Wraps the executable command threshold.
Create bitwise image by thresholding image intensity.
By default, the threshold level is determined using a histogram analysis to cut out the background. Otherwise, the threshold intensity can be specified using command line options. Note that only the first study is used for thresholding.

Example

```python
>>> import nipype.interfaces.mrtrix as mrt
>>> thresh = mrt.Threshold()
>>> thresh.inputs.in_file = 'wm_mask.mif'
>>> thresh.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  The input image to be thresholded
  argument: ``%s``, position: -2

[Optional]
out_filename: (a pathlike object or string representing a file)
  The output binary image mask.
  argument: ``%s`` , position: -1
absolute_threshold_value: (a float)
    Specify threshold value as absolute intensity.
    argument: ``-abs %s``
percentage_threshold_value: (a float)
    Specify threshold value as a percentage of the peak intensity in the
    input image.
    argument: ``-percent %s``
invert: (a boolean)
    Invert output binary mask
    argument: ``-invert``, position: 1
replace_zeros_with_NaN: (a boolean)
    Replace all zero values with NaN
    argument: ``-nan``, position: 1
quiet: (a boolean)
    Do not display information messages or progress status.
    argument: ``-quiet``, position: 1
debug: (a boolean)
    Display debugging messages.
    argument: ``-debug``, position: 1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})
    Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
    The output binary image mask.

68.3 interfaces.mrtrix.tensors

68.3.1 ConstrainedSphericalDeconvolution

Link to code
Wraps the executable command csdeconv.
Perform non-negativity constrained spherical deconvolution.
Note that this program makes use of implied symmetries in the diffusion profile. First, the fact the signal
attenuation profile is real implies that it has conjugate symmetry, i.e. \( Y(l,-m) = Y(l,m)\) (where * denotes the
complex conjugate). Second, the diffusion profile should be antipodally symmetric (i.e. \( S(x) = S(-x)\)), implying
that all odd \( l \) components should be zero. Therefore, this program only computes the even elements. Note that
the spherical harmonics equations used here differ slightly from those conventionally used, in that the \((-1)^m\)
factor has been omitted. This should be taken into account in all subsequent calculations. Each volume in the
output image corresponds to a different spherical harmonic component, according to the following convention:

- [0] \( Y(0,0) \)
- [1] Im \( Y(2,2) \)
- [2] Im \( Y(2,1) \)
- [3] \( Y(2,0) \)
- [4] Re \( Y(2,1) \)
- [5] Re \( Y(2,2) \)
- [6] Im \( Y(4,4) \)
- [7] Im \( Y(4,3) \)
Example

```python
>>> import nipype.interfaces.mrtrix as mrt
>>> csdeconv = mrt.ConstrainedSphericalDeconvolution()
>>> csdeconv.inputs.in_file = 'dwi.mif'
>>> csdeconv.inputs.encoding_file = 'encoding.txt'
>>> csdeconv.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  diffusion-weighted image
  argument: `"%s"`, position: -3
response_file: (a pathlike object or string representing an existing file)
  the diffusion-weighted signal response function for a single fibre
  population (see EstimateResponse)
  argument: `"%s"`, position: -2

[Optional]
out_filename: (a pathlike object or string representing a file)
  Output filename
  argument: `"%s"`, position: -1
mask_image: (a pathlike object or string representing an existing file)
  only perform computation within the specified binary brain mask image
  argument: `-mask %s`, position: 2
encoding_file: (a pathlike object or string representing an existing file)
  Gradient encoding, supplied as a 4xN text file with each line is in
  the format [ X Y Z b ], where [ X Y Z ] describe the direction of
  the applied gradient, and b gives the b-value in units (1000
  s/mm^2). See FSL2MRTrix
  argument: `-grad %s`, position: 1
filter_file: (a pathlike object or string representing an existing file)
  a text file containing the filtering coefficients for each even
  harmonic order.the linear frequency filtering parameters used for
  the initial linear spherical deconvolution step (default = [ 1 1 1 0
  0 ]).
  argument: `-filter %s`, position: -2
lambda_value: (a float)
  the regularisation parameter lambda that controls the strength of
  the constraint (default = 1.0).
  argument: `"-lambda %s``
maximum_harmonic_order: (an integer (int or long))
  set the maximum harmonic order for the output series. By default,
  the program will use the highest possible lmax given the number of
  diffusion-weighted images.
  argument: `"-lmax %s``
threshold_value: (a float)
  the threshold below which the amplitude of the FOD is assumed to be
  zero, expressed as a fraction of the mean value of the initial FOD
  (default = 0.1)
  argument: `"-threshold %s``
iterations: (an integer (int or long))

(continues on next page)
the maximum number of iterations to perform for each voxel (default = 50)
argument: `--niter %s`

debug: (a boolean)
Display debugging messages.
argument: `--debug`

directions_file: (a pathlike object or string representing an existing file)
    a text file containing the [el az] pairs for the directions:
    Specify the directions over which to apply the non-negativity constraint (by default, the built-in 300 direction set is used)
argument: `--directions %s`, position: -2

normalise: (a boolean)
    normalise the DW signal to the b=0 image
argument: `--normalise`, position: 3

args: (a unicode string)
    Additional parameters to the command
argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

(outputs)

spherical_harmonics_image: (a pathlike object or string representing an existing file)
    Spherical harmonics image

68.3.2 DWI2SphericalHarmonicsImage

Link to code
Wraps the executable command dwi2SH.
Convert base diffusion-weighted images to their spherical harmonic representation.
This program outputs the spherical harmonic decomposition for the set measured signal attenuations. The signal attenuations are calculated by identifying the b-zero images from the diffusion encoding supplied (i.e. those with zero as the b-value), and dividing the remaining signals by the mean b-zero signal intensity. The spherical harmonic decomposition is then calculated by least-squares linear fitting. Note that this program makes use of implied symmetries in the diffusion profile.
First, the fact the signal attenuation profile is real implies that it has conjugate symmetry, i.e. \( Y(l,-m) = Y(l,m)^* \) (where * denotes the complex conjugate). Second, the diffusion profile should be antipodally symmetric (i.e. \( S(x) = S(-x) \)), implying that all odd \( l \) components should be zero. Therefore, this program only computes the even elements.
Note that the spherical harmonics equations used here differ slightly from those conventionally used, in that the \((-1)^m\) factor has been omitted. This should be taken into account in all subsequent calculations.
Each volume in the output image corresponds to a different spherical harmonic component, according to the following convention:
* \([0] Y(0,0)\)
* \([1] \text{Im} \{Y(2,2)\}\)
* \([2] \text{Im} \{Y(2,1)\}\)
* \([3] Y(2,0)\)
* \([4] \text{Re} \{Y(2,1)\}\)
* \([5] \text{Re} \{Y(2,2)\}\)
* \([6] \text{Im} \{Y(4,4)\}\)
* \([7] \text{Im} \{Y(4,3)\}\)
Example

```python
>>> import nipype.interfaces.mrtrix as mrt
>>> dwi2SH = mrt.DWI2SphericalHarmonicsImage()
>>> dwi2SH.inputs.in_file = 'diffusion.nii'
>>> dwi2SH.inputs.encoding_file = 'encoding.txt'
>>> dwi2SH.run()  # doctest: +SKIP
```

### Inputs:

- **in_file**: (a pathlike object or string representing an existing file)
  - Diffusion-weighted images
  - Argument: ``%s``, position: -2
- **encoding_file**: (a pathlike object or string representing an existing file)
  - Gradient encoding, supplied as a 4xN text file with each line is in the format [ X Y Z b ], where [ X Y Z ] describe the direction of the applied gradient, and b gives the b-value in units (1000 s/mm^2). See FSL2MRTrix
  - Argument: ``-grad %s``, position: 1

### Optional:

- **out_filename**: (a pathlike object or string representing a file)
  - Output filename
  - Argument: ``%s``, position: -1
- **maximum_harmonic_order**: (a float)
  - Set the maximum harmonic order for the output series. By default, the program will use the highest possible lmax given the number of diffusion-weighted images.
  - Argument: ``-lmax %s``
- **normalise**: (a boolean)
  - Normalise the DW signal to the b=0 image
  - Argument: ``-normalise``, position: 3
- **args**: (a unicode string)
  - Additional parameters to the command
  - Argument: ``%s``
- **environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})  
  - Environment variables

### Outputs:

- **spherical_harmonics_image**: (a pathlike object or string representing an existing file)
  - Spherical harmonics image

### 68.3.3 Directions2Amplitude

**Link to code**

Wraps the executable command `dir2amp`.

convert directions image to amplitudes
Example

```python
>>> import nipype.interfaces.mrtrix as mrt
>>> amplitudes = mrt.Directions2Amplitude()
>>> amplitudes.inputs.in_file = 'peak_directions.mif'
>>> amplitudes.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
- **in_file**: (a pathlike object or string representing an existing file)
  - the input directions image. Each volume corresponds to the x, y & z component of each direction vector in turn.
  - argument: ``-%s``, position: -2

[Optional]
- **peaks_image**: (a pathlike object or string representing an existing file)
  - the program will try to find the peaks that most closely match those in the image provided
  - argument: ``-peaks %s``
- **num_peaks**: (an integer (int or long))
  - the number of peaks to extract (default is 3)
  - argument: ``-num %s``
- **peak_directions**: (a list of from 2 to 2 items which are a float)
  - the direction of a peak to estimate. The algorithm will attempt to find the same number of peaks as have been specified using this option
    - phi: the azimuthal angle of the direction (in degrees)
    - theta: the elevation angle of the direction (in degrees, from the vertical z-axis)
  - argument: ``-direction %s``
- **display_info**: (a boolean)
  - Display information messages.
  - argument: ``-info``
- **quiet_display**: (a boolean)
  - do not display information messages or progress status.
  - argument: ``-quiet``
- **display_debug**: (a boolean)
  - Display debugging messages.
  - argument: ``-debug``
- **out_file**: (a pathlike object or string representing a file)
  - the output amplitudes image
  - argument: ``-%s``, position: -1
- **args**: (a unicode string)
  - Additional parameters to the command
  - argument: ``-%s``
- **environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  - Environment variables

Outputs:

- **out_file**: (a pathlike object or string representing an existing file)
  - amplitudes image

### 68.3.4 EstimateResponseForSH

Link to code
Wraps the executable command `estimate_response`.  
Estimates the fibre response function for use in spherical deconvolution.

**Example**

```python
>>> import nipype.interfaces.mrtrix as mrt
>>> estresp = mrt.EstimateResponseForSH()
>>> estresp.inputs.in_file = 'dwi.mif'
>>> estresp.inputs.mask_image = 'dwi_WMProb.mif'
>>> estresp.inputs.encoding_file = 'encoding.txt'
>>> estresp.run()  # doctest: +SKIP
```

**Inputs:**

[Mandatory]
- `in_file`: (a pathlike object or string representing an existing file)
  - Diffusion-weighted images
    - argument: ``%s``, position: -3
- `mask_image`: (a pathlike object or string representing an existing file)
  - only perform computation within the specified binary brain mask image
    - argument: ``%s``, position: -2
- `encoding_file`: (a pathlike object or string representing an existing file)
  - Gradient encoding, supplied as a 4xN text file with each line is in the format [ X Y Z b ], where [ X Y Z ] describe the direction of the applied gradient, and b gives the b-value in units (1000 s/mm^2). See FSL2MRTrix
    - argument: ``-grad %s``, position: 1

[Optional]
- `out_filename`: (a pathlike object or string representing a file)
  - Output filename
    - argument: ``%s``, position: -1
- `maximum_harmonic_order`: (an integer (int or long))
  - set the maximum harmonic order for the output series. By default, the program will use the highest possible lmax given the number of diffusion-weighted images.
    - argument: ``-lmax %s``
- `normalise`: (a boolean)
  - normalise the DW signal to the b=0 image
    - argument: ``-normalise``
- `quiet`: (a boolean)
  - Do not display information messages or progress status.
    - argument: ``-quiet``
- `debug`: (a boolean)
  - Display debugging messages.
    - argument: ``-debug``
- `args`: (a unicode string)
  - Additional parameters to the command
    - argument: ``%s``
- `environ`: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  - Environment variables

**Outputs:**
response: (a pathlike object or string representing an existing file)
Spherical harmonics image

68.3.5 FindShPeaks

Link to code
Wraps the executable command find_SH_peaks.
identify the orientations of the N largest peaks of a SH profile

Example

```python
>>> import nipype.interfaces.mrtrix as mrt
>>> shpeaks = mrt.FindShPeaks()
>>> shpeaks.inputs.in_file = 'csd.mif'
>>> shpeaks.inputs.directions_file = 'dirs.txt'
>>> shpeaks.inputs.num_peaks = 2
>>> shpeaks.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  the input image of SH coefficients.
  argument: ``-%s``, position: -3
directions_file: (a pathlike object or string representing an
  existing file)
  the set of directions to use as seeds for the peak finding
  argument: ``-%s``, position: -2

[Optional]
peaks_image: (a pathlike object or string representing an existing
  file)
  the program will try to find the peaks that most closely match those
  in the image provided
  argument: ``-peaks %s``
num_peaks: (an integer (int or long))
  the number of peaks to extract (default is 3)
  argument: ``-num %s``
peak_directions: (a list of from 2 to 2 items which are a float)
  phi theta. the direction of a peak to estimate. The algorithm will
  attempt to find the same number of peaks as have been specified
  using this option phi: the azimuthal angle of the direction (in
  degrees). theta: the elevation angle of the direction (in degrees,
  from the vertical z-axis)
  argument: ``-direction %s``
peak_threshold: (a float)
  only peak amplitudes greater than the threshold will be considered
  argument: ``-threshold %s``
display_info: (a boolean)
  Display information messages.
  argument: ``-info``
quiet_display: (a boolean)
  do not display information messages or progress status.
  argument: ``-quiet``
display_debug: (a boolean)
  Display debugging messages.

(continues on next page)
argument: `"-debug"`

out_file: (a pathlike object or string representing a file)
the output image. Each volume corresponds to the x, y & z component
of each peak direction vector in turn
argument: `"%s"`, position: -1

args: (a unicode string)
Additional parameters to the command
argument: `"%s"

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {}))
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
Peak directions image

68.3.6 GenerateDirections

Link to code
Wraps the executable command gendir.
generate a set of directions evenly distributed over a hemisphere.

Example

```python
>>> import nipype.interfaces.mrtrix as mrt
>>> gendir = mrt.GenerateDirections()
>>> gendir.inputs.num_dirs = 300
>>> gendir.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
num_dirs: (an integer (int or long))
the number of directions to generate.
argument: `"%s"`, position: -2

[Optional]
power: (a float)
specify exponent to use for repulsion power law.
argument: `"-power %s"
niter: (an integer (int or long))
specify the maximum number of iterations to perform.
argument: `"-niter %s"
display_info: (a boolean)
Display information messages.
argument: `"-info"
quiet_display: (a boolean)
do not display information messages or progress status.
argument: `"-quiet"
display_debug: (a boolean)
Display debugging messages.
argument: `"-debug"
out_file: (a pathlike object or string representing a file)
the text file to write the directions to, as [ az el ] pairs.

(continues on next page)
68.3.7concat_files()

Link to code

68.4interfaces.mrtrix.tracking

68.4.1DiffusionTensorStreamlineTrack

Link to code

Wraps the executable command streamtrack. Specialized interface to StreamlineTrack. This interface is used for streamline tracking from diffusion tensor data, and calls the MRtrix function ‘streamtrack’ with the option ‘DT_STREAM’

Example

```python
>>> import nipype.interfaces.mrtrix as mrt
>>> dtstrack = mrt.DiffusionTensorStreamlineTrack()
>>> dtstrack.inputs.in_file = 'data.Bfloat'
>>> dtstrack.inputs.seed_file = 'seed_mask.nii'
>>> dtstrack.run()
# doctest: +SKIP
```

Inputs:

[Mandatory]

gradient_encoding_file: (a pathlike object or string representing an existing file)

Gradient encoding, supplied as a 4xN text file with each line is in the format [ X Y Z b ], where [ X Y Z ] describe the direction of the applied gradient, and b gives the b-value in units (1000 s/mm^2). See FSL2MRtrix

argument: `--grad %s`, position: -2

in_file: (a pathlike object or string representing an existing file)

the image containing the source data. The type of data required depends on the type of tracking as set in the preceding argument.

For DT methods, the base DWI are needed. For SD methods, the SH harmonic coefficients of the FOD are needed.

argument: `-%s`, position: -2

[Optional]

seed_file: (a pathlike object or string representing an existing file)

(continues on next page)
seed_file
argument: `'-seed %s'`
mutually_exclusive: seed_file, seed_spec
seed_spec: (a list of from 4 to 4 items which are a float)
seed specification in mm and radius (x y z r)
argument: `'-seed %s'`, position: 2
mutually_exclusive: seed_file, seed_spec
include_file: (a pathlike object or string representing an existing file)
inclusion file
argument: `'-include %s'`
mutually_exclusive: include_file, include_spec
include_spec: (a list of from 4 to 4 items which are a float)
inclusion specification in mm and radius (x y z r)
argument: `'-include %s'`, position: 2
mutually_exclusive: include_file, include_spec
exclude_file: (a pathlike object or string representing an existing file)
exclusion file
argument: `'-exclude %s'`
mutually_exclusive: exclude_file, exclude_spec
exclude_spec: (a list of from 4 to 4 items which are a float)
exclusion specification in mm and radius (x y z r)
argument: `'-exclude %s'`, position: 2
mutually_exclusive: exclude_file, exclude_spec
mask_file: (a pathlike object or string representing an existing file)
mask file. Only tracks within mask.
argument: `'-mask %s'`
mutually_exclusive: mask_file, mask_spec
mask_spec: (a list of from 4 to 4 items which are a float)
Mask specification in mm and radius (x y z r). Tracks will be terminated when they leave the ROI.
argument: `'-mask %s'`, position: 2
mutually_exclusive: mask_file, mask_spec
inputmodel: ('DT_STREAM' or 'SD_PROB' or 'SD_STREAM', nipype default value: DT_STREAM)
input model type
argument: `'%s'`, position: -3
stop: (a boolean)
stop track as soon as it enters any of the include regions.
argument: `'-stop'`
do_not_precompute: (a boolean)
Turns off precomputation of the legendre polynomial values. Warning: this will slow down the algorithm by a factor of approximately 4.
argument: `'-noprecomputed'`
unidirectional: (a boolean)
Track from the seed point in one direction only (default is to track in both directions).
argument: `'-unidirectional'`
no_mask_interpolation: (a boolean)
Turns off trilinear interpolation of mask images.
argument: `'-nomaskinterp'`
step_size: (a float)
Set the step size of the algorithm in mm (default is 0.2).
argument: `'-step %s'`
minimum_radius_of_curvature: (a float)
Set the minimum radius of curvature (default is 2 mm for DT_STREAM, 0 for SD_STREAM, 1 mm for SD_PROB and DT_PROB)
argument: `'--curvature %s'`

desired_number_of_tracks: (an integer (int or long))
    Sets the desired number of tracks. The program will continue to
    generate tracks until this number of tracks have been selected and
    written to the output file (default is 100 for _STREAM methods, 1000
    for _PROB methods).
    argument: `'--number %d'`

maximum_number_of_tracks: (an integer (int or long))
    Sets the maximum number of tracks to generate. The program will not
    generate more tracks than this number, even if the desired number of
    tracks hasn't yet been reached (default is 100 x number).
    argument: `'--maxnum %d'`

minimum_tract_length: (a float)
    Sets the minimum length of any track in millimeters (default is 10
    mm).
    argument: `'--minlength %s'`

maximum_tract_length: (a float)
    Sets the maximum length of any track in millimeters (default is 200
    mm).
    argument: `'--length %s'`

cutoff_value: (a float)
    Set the FA or FOD amplitude cutoff for terminating tracks (default
    is 0.1).
    argument: `'--cutoff %s'`

initial_cutoff_value: (a float)
    Sets the minimum FA or FOD amplitude for initiating tracks (default
    is twice the normal cutoff).
    argument: `'--initcutoff %s'`

initial_direction: (a list of from 2 to 2 items which are an integer
    (int or long))
    Specify the initial tracking direction as a vector
    argument: `'--initdirection %s'`

out_file: (a pathlike object or string representing a file)
    output data file
    argument: `'%s'`, position: `-1`

args: (a unicode string)
    Additional parameters to the command
    argument: `'%s'`

environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {}) Environment variables

Outputs:

tracked: (a pathlike object or string representing an existing file)
    output file containing reconstructed tracts

68.4.2 FilterTracks

Link to code
Wraps the executable command filter_tracks.
Use regions-of-interest to select a subset of tracks from a given MRtrix track file.
Example

```python
>>> import nipype.interfaces.mrtrix as mrt
>>> filt = mrt.FilterTracks()
>>> filt.inputs.in_file = 'tracks.tck'
>>> filt.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  input tracks to be filtered
  argument: ``\`%s```, position: -2

[Optional]
include_file: (a pathlike object or string representing an existing file)
  inclusion file
  argument: ``\`-include %s\``
  mutually_exclusive: include_file, include_spec
include_spec: (a list of from 4 to 4 items which are a float)
  inclusion specification in mm and radius (x y z r)
  argument: ``\`-include %s\```, position: 2
  mutually_exclusive: include_file, include_spec
exclude_file: (a pathlike object or string representing an existing file)
  exclusion file
  argument: ``\`-exclude %s\``
  mutually_exclusive: exclude_file, exclude_spec
exclude_spec: (a list of from 4 to 4 items which are a float)
  exclusion specification in mm and radius (x y z r)
  argument: ``\`-exclude %s\```, position: 2
  mutually_exclusive: exclude_file, exclude_spec
minimum_tract_length: (a float)
  Sets the minimum length of any track in millimeters (default is 10 mm).
  argument: ``\`-minlength %s\``
out_file: (a pathlike object or string representing a file)
  Output filtered track filename
  argument: ``\`%s```, position: -1
no_mask_interpolation: (a boolean)
  Turns off trilinear interpolation of mask images.
  argument: ``\`-nomaskinterp``
invert: (a boolean)
  invert the matching process, so that tracks that would otherwise have
  been included are now excluded and vice-versa.
  argument: ``\`-invert\``
quiet: (a boolean)
  Do not display information messages or progress status.
  argument: ``\`-quiet\```, position: 1
debug: (a boolean)
  Display debugging messages.
  argument: ``\`-debug\```, position: 1
args: (a unicode string)
  Additional parameters to the command
  argument: ``\`%s\``
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
(continues on next page)
value of class 'str', nipype default value: {}))
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
the output filtered tracks

68.4.3 ProbabilisticSphericallyDeconvolutedStreamlineTrack

Link to code
Wraps the executable command streamtrack.
Performs probabilistic tracking using spherically deconvolved data
Specialized interface to StreamlineTrack. This interface is used for probabilistic tracking from spherically de-
convolved data, and calls the MRtrix function ‘streamtrack’ with the option ‘SD_PROB’

Example

```python
>>> import nipype.interfaces.mrtrix as mrt
>>> sdprobtrack = mrt.ProbabilisticSphericallyDeconvolutedStreamlineTrack()
>>> sdprobtrack.inputs.in_file = 'data.Bfloat'
>>> sdprobtrack.inputs.seed_file = 'seed_mask.nii'
>>> sdprobtrack.run()
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
the image containing the source data. The type of data required depends on the type of tracking as set in the preceeding argument. For DT methods, the base DWI are needed. For SD methods, the SH harmonic coefficients of the FOD are needed.
argument: ``%s``, position: -2

[Optional]
maximum_number_of_trials: (an integer (int or long))
Set the maximum number of sampling trials at each point (only used for probabilistic tracking).
argument: ``-trials %s``
seed_file: (a pathlike object or string representing an existing file)
seed file
argument: ``-seed %s``
mutually_exclusive: seed_file, seed_spec
seed_spec: (a list of from 4 to 4 items which are a float)
seed specification in mm and radius (x y z r)
argument: ``-seed %s``
mutually_exclusive: seed_file, seed_spec
include_file: (a pathlike object or string representing an existing file)
inclusion file
argument: ``-include %s``
mutually_exclusive: include_file, include_spec
include_spec: (a list of from 4 to 4 items which are a float)
inclusion specification in mm and radius (x y z r)

(continues on next page)
```
argument: `\'-include %s\'`, position: 2
mutually_exclusive: include_file, include_spec
exclude_file: (a pathlike object or string representing an existing
file)
exclusion file
argument: `\'-exclude %s\'`
mutually_exclusive: exclude_file, exclude_spec
exclude_spec: (a list of from 4 to 4 items which are a float)
exclusion specification in mm and radius (x y z r)
argument: `\'-exclude %s\'`, position: 2
mutually_exclusive: exclude_file, exclude_spec
mask_file: (a pathlike object or string representing an existing
file)
mask file. Only tracks within mask.
argument: `\'-mask %s\'`
mutually_exclusive: mask_file, mask_spec
mask_spec: (a list of from 4 to 4 items which are a float)
Mask specification in mm and radius (x y z r). Tracks will be
terminated when they leave the ROI.
argument: `\'-mask %s\'`, position: 2
mutually_exclusive: mask_file, mask_spec
inputmodel: (\'DT_STREAM\' or \'SD_PROB\' or \'SD_STREAM\', nipype default
value: DT_STREAM)
input model type
argument: `\'%s\'`, position: -3
stop: (a boolean)
stop track as soon as it enters any of the include regions.
argument: `\'-stop\'`
do_not_precompute: (a boolean)
Turns off precomputation of the legendre polynomial values. Warning:
this will slow down the algorithm by a factor of approximately 4.
argument: `\'-noprecomputed\'`
unidirectional: (a boolean)
Track from the seed point in one direction only (default is to track
in both directions).
argument: `\'-unidirectional\'`
no_mask_interpolation: (a boolean)
Turns off trilinear interpolation of mask images.
argument: `\'-nomaskinterp\'`
step_size: (a float)
Set the step size of the algorithm in mm (default is 0.2).
argument: `\'-step %s\'`
minimum_radius_of_curvature: (a float)
Set the minimum radius of curvature (default is 2 mm for DT_STREAM,
0 for SD_STREAM, 1 mm for SD_PROB and DT_PROB)
argument: `\'-curvature %s\'`
desired_number_of_tracks: (an integer (int or long))
Sets the desired number of tracks. The program will continue to
generate tracks until this number of tracks have been selected and
written to the output file (default is 100 for _STREAM methods, 1000
for _PROB methods).
argument: `\'-number %d\'`
maximum_number_of_tracks: (an integer (int or long))
Sets the maximum number of tracks to generate. The program will not
generate more tracks than this number, even if the desired number of
tracks hasn't yet been reached (default is 100 x number).
argument: `\'-maxnum %d\'
```
minimum_tract_length: (a float)
    Sets the minimum length of any track in millimeters (default is 10
    mm).
    argument: '-minlength %s'
maximum_tract_length: (a float)
    Sets the maximum length of any track in millimeters (default is 200
    mm).
    argument: '-length %s'
cutoff_value: (a float)
    Set the FA or FOD amplitude cutoff for terminating tracks (default
    is 0.1).
    argument: '-cutoff %s'
initial_cutoff_value: (a float)
    Sets the minimum FA or FOD amplitude for initiating tracks (default
    is twice the normal cutoff).
    argument: '-initcutoff %s'
initial_direction: (a list of from 2 to 2 items which are an integer
    (int or long))
    Specify the initial tracking direction as a vector
    argument: '-initdirection %s'
out_file: (a pathlike object or string representing a file)
    output data file
    argument: '%s', position: -1
args: (a unicode string)
    Additional parameters to the command
    argument: '%s'
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})
    Environment variables

Outputs:

tracked: (a pathlike object or string representing an existing file)
    output file containing reconstructed tracts

68.4.4 SphericallyDeconvolutedStreamlineTrack

Link to code
Wraps the executable command streamtrack.
Performs streamline tracking using spherically deconvolved data
Specialized interface to StreamlineTrack. This interface is used for streamline tracking from spherically decon-
volved data, and calls the MRtrix function ‘streamtrack’ with the option ‘SD_STREAM’

Example

```python
>>> import nipype.interfaces.mrtrix as mrt
>>> sdtrack = mrt.SphericallyDeconvolutedStreamlineTrack()
>>> sdtrack.inputs.in_file = 'data.Bfloat'
>>> sdtrack.inputs.seed_file = 'seed_mask.nii'
>>> sdtrack.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)

(continues on next page)
the image containing the source data. The type of data required depends on the type of tracking as set in the preceding argument. For DT methods, the base DWI are needed. For SD methods, the SH harmonic coefficients of the FOD are needed.

argument: ``\`%s`\``, position: -2

[Optional]

**seed_file**: (a pathlike object or string representing an existing file)

seed file

argument: ``\`-seed %s`\``

mutually exclusive: seed_file, seed_spec

**seed_spec**: (a list of from 4 to 4 items which are a float)

seed specification in mm and radius (x y z r)

argument: ``\`-seed %s`\``, position: 2

mutually exclusive: seed_file, seed_spec

**include_file**: (a pathlike object or string representing an existing file)

inclusion file

argument: ``\`-include %s`\``

mutually exclusive: include_file, include_spec

**include_spec**: (a list of from 4 to 4 items which are a float)

inclusion specification in mm and radius (x y z r)

argument: ``\`-include %s`\``, position: 2

mutually exclusive: include_file, include_spec

**exclude_file**: (a pathlike object or string representing an existing file)

exclusion file

argument: ``\`-exclude %s`\``

mutually exclusive: exclude_file, exclude_spec

**exclude_spec**: (a list of from 4 to 4 items which are a float)

exclusion specification in mm and radius (x y z r)

argument: ``\`-exclude %s`\``, position: 2

mutually exclusive: exclude_file, exclude_spec

**mask_file**: (a pathlike object or string representing an existing file)

mask file. Only tracks within mask.

argument: ``\`-mask %s`\``

mutually exclusive: mask_file, mask_spec

**mask_spec**: (a list of from 4 to 4 items which are a float)

Mask specification in mm and radius (x y z r). Tracks will be terminated when they leave the ROI.

argument: ``\`-mask %s`\``, position: 2

mutually exclusive: mask_file, mask_spec

**inputmodel**: ('DT_STREAM' or 'SD_PROB' or 'SD_STREAM', nipype default value: DT_STREAM)

input model type

argument: ``\`%s`\``, position: -3

**stop**: (a boolean)

stop track as soon as it enters any of the include regions.

argument: ``\`-stop`\``

**do_not_precompute**: (a boolean)

Turns off precomputation of the legendre polynomial values. Warning: this will slow down the algorithm by a factor of approximately 4.

argument: ``\`-noprecomputed`\``

**unidirectional**: (a boolean)

Track from the seed point in one direction only (default is to track bidirectionally).

(continues on next page)
in both directions).
argument: `--unidirectional`

no_mask_interpolation: (a boolean)
Turns off trilinear interpolation of mask images.
argument: `--nomaskinterp`

step_size: (a float)
Set the step size of the algorithm in mm (default is 0.2).
argument: `--step %s`

minimum_radius_of_curvature: (a float)
Set the minimum radius of curvature (default is 2 mm for DT_STREAM,
0 for SD_STREAM, 1 mm for SD_PROB and DT_PROB)
argument: `--curvature %s`

desired_number_of_tracks: (an integer (int or long))
Sets the desired number of tracks. The program will continue to
generate tracks until this number of tracks have been selected and
written to the output file (default is 100 for *_STREAM methods, 1000
for *_PROB methods).
argument: `--number %d`

maximum_number_of_tracks: (an integer (int or long))
Sets the maximum number of tracks to generate. The program will not
generate more tracks than this number, even if the desired number of
tracks hasn't yet been reached (default is 100 x number).
argument: `--maxnum %d`

minimum_tract_length: (a float)
Sets the minimum length of any track in millimeters (default is 10
mm).
argument: `--minlength %s`

maximum_tract_length: (a float)
Sets the maximum length of any track in millimeters (default is 200
mm).
argument: `--length %s`

cutoff_value: (a float)
Set the FA or FOD amplitude cutoff for terminating tracks (default
is 0.1).
argument: `--cutoff %s`

initial_cutoff_value: (a float)
Sets the minimum FA or FOD amplitude for initiating tracks (default
is twice the normal cutoff).
argument: `--initcutoff %s`

initial_direction: (a list of from 2 to 2 items which are an integer
(int or long))
Specify the initial tracking direction as a vector
argument: `--initdirection %s`

out_file: (a pathlike object or string representing a file)
output data file
argument: `%%%s`, position: -1

args: (a unicode string)
Additional parameters to the command
argument: `%%%s`

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

tracked: (a pathlike object or string representing an existing file)
output file containing reconstructed tracts

68.4.5 StreamlineTrack

Link to code
Wraps the executable command `streamtrack`.
Performs tractography using one of the following models: ‘dt_prob’, ‘dt_stream’, ‘sd_prob’, ‘sd_stream’.
Where ‘dt’ stands for diffusion tensor, ‘sd’ stands for spherical deconvolution, and ‘prob’ stands for probabilistic.

Example

```python
>>> import nipype.interfaces.mrtrix as mrt

>>> strack = mrt.StreamlineTrack()

>>> strack.inputs.inputmodel = 'SD_PROB'

>>> strack.inputs.in_file = 'data.Bfloat'

>>> strack.inputs.seed_file = 'seed_mask.nii'

>>> strack.inputs.mask_file = 'mask.nii'

>>> strack.cmdline
'streamtrack -mask mask.nii -seed seed_mask.nii SD_PROB data.Bfloat data_tracked.tck'

>>> strack.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  the image containing the source data. The type of data required depends on the type of tracking as set in the preceding argument.
  For DT methods, the base DWI are needed. For SD methods, the SH harmonic coefficients of the FOD are needed.
  argument: ``%s``, position: -2

[Optional]
seed_file: (a pathlike object or string representing an existing file)
  seed file
  argument: ``-seed %s``
  mutually_exclusive: seed_file, seed_spec
seed_spec: (a list of from 4 to 4 items which are a float)
  seed specification in mm and radius (x y z r)
  argument: ``-seed %s``, position: 2
  mutually_exclusive: seed_file, seed_spec
include_file: (a pathlike object or string representing an existing file)
  inclusion file
  argument: ``-include %s``
  mutually_exclusive: include_file, include_spec
include_spec: (a list of from 4 to 4 items which are a float)
  inclusion specification in mm and radius (x y z r)
  argument: ``-include %s``, position: 2
  mutually_exclusive: include_file, include_spec
exclude_file: (a pathlike object or string representing an existing file)
  exclusion file  
  (continues on next page)
argument: `'-exclude %s'
mutually_exclusive: exclude_file, exclude_spec
exclude_spec: (a list of from 4 to 4 items which are a float)
exclusion specification in mm and radius (x y z r)
argument: `'-exclude %s'`, position: 2
mutually_exclusive: exclude_file, exclude_spec
mask_file: (a pathlike object or string representing an existing file)
mask file. Only tracks within mask.
argument: `'-mask %s'
mutually_exclusive: mask_file, mask_spec
mask_spec: (a list of from 4 to 4 items which are a float)
Mask specification in mm and radius (x y z r). Tracks will be
terminated when they leave the ROI.
argument: `'-mask %s'`, position: 2
mutually_exclusive: mask_file, mask_spec
inputmodel: ('DT_STREAM' or 'SD_PROB' or 'SD_STREAM', nipype default
value: DT_STREAM)
input model type
argument: `'-inputmodel %s'`, position: -3
stop: (a boolean)
stop track as soon as it enters any of the include regions.
argument: `'-stop'
do_not_precompute: (a boolean)
Turns off precomputation of the legendre polynomial values. Warning:
this will slow down the algorithm by a factor of approximately 4.
argument: `'-noprecomputed'
unidirectional: (a boolean)
Track from the seed point in one direction only (default is to track
in both directions).
argument: `'-unidirectional'
no_mask_interpolation: (a boolean)
Turns off trilinear interpolation of mask images.
argument: `'-nomaskinterp'
step_size: (a float)
Set the step size of the algorithm in mm (default is 0.2).
argument: `'-step %s'
minimum_radius_of_curvature: (a float)
Set the minimum radius of curvature (default is 2 mm for DT_STREAM,
0 for SD_STREAM, 1 mm for SD_PROB and DT_PROB)
argument: `'-curvature %s'
desired_number_of_tracks: (an integer (int or long))
Sets the desired number of tracks. The program will continue to
generate tracks until this number of tracks have been selected and
written to the output file (default is 100 for *_STREAM methods, 1000
for *_PROB methods).
argument: `'-number %d'
maximum_number_of_tracks: (an integer (int or long))
Sets the maximum number of tracks to generate. The program will not
generate more tracks than this number, even if the desired number of
tracks hasn't yet been reached (default is 100 x number).
argument: `'-maxnum %d'
minimum_tract_length: (a float)
Sets the minimum length of any track in millimeters (default is 10
mm).
argument: `'-minlength %s'
maximum_tract_length: (a float)
Sets the maximum length of any track in millimeters (default is 200 mm).
  argument: ``-length %s``

cutoff_value: (a float)
  Set the FA or FOD amplitude cutoff for terminating tracks (default is 0.1).
  argument: ``-cutoff %s``

initial_cutoff_value: (a float)
  Sets the minimum FA or FOD amplitude for initiating tracks (default is twice the normal cutoff).
  argument: ``-initcutoff %s``

initial_direction: (a list of from 2 to 2 items which are an integer (int or long))
  Specify the initial tracking direction as a vector
  argument: ``-initdirection %s``

out_file: (a pathlike object or string representing a file)
  output data file
  argument: ``%s``, position: -1

args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}))
  Environment variables

Outputs:

tracked: (a pathlike object or string representing an existing file)
  output file containing reconstructed tracts

68.4.6 Tracks2Prob

Link to code
Wraps the executable command tracks2prob.
Convert a tract file into a map of the fraction of tracks to enter each voxel - also known as a tract density image (TDI) - in MRtrix’s image format (.mif). This can be viewed using MRview or converted to Nifti using MRconvert.

Example

```python
>>> import nipype.interfaces.mrtrix as mrt
>>> tdi = mrt.Tracks2Prob()
>>> tdi.inputs.in_file = 'dwi_CSD_tracked.tck'
>>> tdi.inputs.colour = True
>>> tdi.run() # doctest: +SKIP
```

Inputs:

in_file: (a pathlike object or string representing an existing file)
  tract file
  argument: ``%s``, position: -2

[Optional]
template_file: (a pathlike object or string representing an existing
an image file to be used as a template for the output (the output image will have the same transform and field of view)
argument: `'-template %s'`, position: 1

voxel_dims: (a list of from 3 to 3 items which are a float)
Three comma-separated numbers giving the size of each voxel in mm.
argument: `'-vox %s'`, position: 2

colour: (a boolean)
add colour to the output image according to the direction of the tracks.
argument: `'-colour'`, position: 3

fraction: (a boolean)
produce an image of the fraction of fibres through each voxel (as a proportion of the total number in the file), rather than the count.
argument: `'-fraction'`, position: 3

output_datatype: ('Bit' or 'Int8' or 'UInt8' or 'Int16' or 'UInt16'
or 'Int32' or 'UInt32' or 'float32' or 'float64')
"i.e. Bfloat". Can be "char", "short", "int", "long", "float" or "double"
argument: `'-datatype %s'`, position: 2

resample: (a float)
resample the tracks at regular intervals using Hermite interpolation. If omitted, the program will select an appropriate interpolation factor automatically.
argument: `'-resample %d'`, position: 3

out_filename: (a pathlike object or string representing a file)
output data file
argument: `'%s'`, position: -1

args: (a unicode string)
Additional parameters to the command
argument: `'%s'`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

tract_image: (a pathlike object or string representing an existing file)
Output tract count or track density image
69.1 interfaces.mrtrix3.base

69.1.1 MRTrix3Base

Link to code

Inputs:

[Optional]
args: (a unicode string)
  Additional parameters to the command
  argument: `\$s`
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})  
Environment variables

Outputs:

None

69.2 interfaces.mrtrix3.connectivity

69.2.1 BuildConnectome

Link to code

Wraps the executable command tck2connectome.
Generate a connectome matrix from a streamlines file and a node parcellation image

Example

```python
>>> import nipype.interfaces.mrtrix3 as mrt
>>> mat = mrt.BuildConnectome()
>>> mat.inputs.in_file = 'tracks.tck'
>>> mat.inputs.in_parc = 'aparc+aseg.nii'
```

(continues on next page)
```plaintext
>>> mat.cmdline
'tck2connectome tracks.tck aparc+aseg.nii connectome.csv'
>>> mat.run()
```

**Inputs:**

- **in_file**: (a pathlike object or string representing an existing file)
  - input tractography
  - argument: `"%s"`, position: -3
- **out_file**: (a pathlike object or string representing a file, nipype default value: `connectome.csv`)
  - output file after processing
  - argument: `"%s"`, position: -1

### [Mandatory]

- **in_parc**: (a pathlike object or string representing an existing file)
  - parcellation file
  - argument: `"%s"`, position: -2
- **nthreads**: (an integer (int or long))
  - number of threads. if zero, the number of available cpus will be used
  - argument: `"--nthreads %d"`
- **vox_lookup**: (a boolean)
  - use a simple voxel lookup value at each streamline endpoint
  - argument: `"--assignment_voxel_lookup"`
- **search_radius**: (a float)
  - perform a radial search from each streamline endpoint to locate the nearest node. Argument is the maximum radius in mm; if no node is found within this radius, the streamline endpoint is not assigned to any node.
  - argument: `"--assignment_radial_search %f"`
- **search_reverse**: (a float)
  - traverse from each streamline endpoint inwards along the streamline, in search of the last node traversed by the streamline. Argument is the maximum traversal length in mm (set to 0 to allow search to continue to the streamline midpoint).
  - argument: `"--assignment_reverse_search %f"`
- **search_forward**: (a float)
  - project the streamline forwards from the endpoint in search of aparc file voxel. Argument is the maximum traversal length in mm.
  - argument: `"--assignment_forward_search %f"`
- **metric**: ("count" or "meangle" or "invlength" or "invnodevolume" or "mean_scalar" or "invlength_invnodevolume")
  - specify the edge weight metric
  - argument: `"--metric %s"`
- **in_scalar**: (a pathlike object or string representing an existing file)
  - provide the associated image for the mean_scalar metric
  - argument: `"--image %s"`
- **in_weights**: (a pathlike object or string representing an existing file)
  - specify a text scalar file containing the streamline weights
  - argument: `"--tck_weights_in %s"`
- **keep_unassigned**: (a boolean)
  - By default, the program discards the information regarding those...
streamlines that are not successfully assigned to a node pair. Set this option to keep these values (will be the first row/column in the output matrix)
argument: `--keep_unassigned`

zero_diagonal: (a boolean)
set all diagonal entries in the matrix to zero (these represent streamlines that connect to the same node at both ends)
argument: `--zero_diagonal`

args: (a unicode string)
Additional parameters to the command
argument: `--%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
the output response file

69.2.2 LabelConfig

Link to code
Wraps the executable command `labelconfig`. Re-configure parcellation to be incrementally defined.

Example

```python
>>> import nipype.interfaces.mrtrix3 as mrt
>>> labels = mrt.LabelConfig()
>>> labels.inputs.in_file = 'aparc+aseg.nii'
>>> labels.inputs.in_config = 'mrtrix3_labelconfig.txt'
>>> labels.cmdline  # doctest: +ELLIPSIS
'labelconfig aparc+aseg.nii mrtrix3_labelconfig.txt parcellation.mif'
>>> labels.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
input anatomical image
argument: `--%s`, position: -3

out_file: (a pathlike object or string representing a file, nipype default value: parcellation.mif)
output file after processing
argument: `--%s`, position: -1

[Optional]
in_config: (a pathlike object or string representing an existing file)
connectome configuration file
argument: `--%s`, position: -2

lut_basic: (a pathlike object or string representing a file)
get information from a basic lookup table consisting of index / name pairs
argument: `--lut_basic %s`
lut_fs: (a pathlike object or string representing a file)
get information from a FreeSurfer lookup table (typically
"FreeSurferColorLUT.txt")
argument: `--lut_freesurfer %s`
lut_aal: (a pathlike object or string representing a file)
get information from the AAL lookup table (typically
"ROI_MNI_V4.txt")
argument: `--lut_aal %s`
lut_itksnap: (a pathlike object or string representing a file)
get information from an ITK-SNAP lookup table (this includes the
IIT atlas file "LUT_GM.txt")
argument: `--lut_itksnap %s`
spine: (a pathlike object or string representing a file)
provide a manually-defined segmentation of the base of the spine
where the streamlines terminate, so that this can become a node in
the connection matrix.
argument: `--spine %s`
nthreads: (an integer (int or long))
number of threads. if zero, the number of available cpus will be
used
argument: `--nthreads %d`
args: (a unicode string)
Additional parameters to the command
argument: `--%s`
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {}) Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
the output response file

69.2.3 LabelConvert

Link to code
Wraps the executable command labelconvert.
Re-configure parcellation to be incrementally defined.

Example

```python
>>> import nipype.interfaces.mrtrix3 as mrt
>>> labels = mrt.LabelConvert()
>>> labels.inputs.in_file = 'aparc+aseg.nii'
>>> labels.inputs.in_config = 'mrtrix3_labelconfig.txt'
>>> labels.inputs.in_lut = 'FreeSurferColorLUT.txt'
>>> labels.cmdline
'labelconvert aparc+aseg.nii FreeSurferColorLUT.txt mrtrix3_labelconfig.txt
\n-parcellation.mif'
>>> labels.run() # doctest: +SKIP
```

Inputs:
[Mandatory]
in_file: (a pathlike object or string representing an existing file)
   input anatomical image
   argument: ``%s``, position: -4
in_lut: (a pathlike object or string representing an existing file)
   get information from a basic lookup table consisting of index / name pairs
   argument: ``%s``, position: -3
out_file: (a pathlike object or string representing a file, nipype default value: parcellation.mif)
   output file after processing
   argument: ``%s``, position: -1

[Optional]
in_config: (a pathlike object or string representing an existing file)
   connectome configuration file
   argument: ``%s``, position: -2
spine: (a pathlike object or string representing a file)
   provide a manually-defined segmentation of the base of the spine where the streamlines terminate, so that this can become a node in the connection matrix.
   argument: ``-spine %s``
num_threads: (an integer (int or long))
   number of threads. if zero, the number of available cpus will be used
   argument: ``-nthreads %d``
args: (a unicode string)
   Additional parameters to the command
   argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
   Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
   the output response file

69.3 interfaces.mrtrix3.preprocess

69.3.1 DWIBiasCorrect

Link to code
Wraps the executable command dwibiascorrect. Perform B1 field inhomogeneity correction for a DWI volume series. For more information, see <https://mrtrix.readthedocs.io/en/latest/reference/scripts/dwibiascorrect.html>

Example

```python
>>> import nipype.interfaces.mrtrix3 as mrt
>>> bias_correct = mrt.DWIBiasCorrect()
>>> bias_correct.inputs.in_file = 'dwi.mif'
>>> bias_correct.inputs.use_ants = True
```
>>> bias_correct.cmdline
'dwibiascorrect -ants dwi.mif dwi_biascorr.mif'
>>> bias_correct.run() # doctest: +SKIP

Inputs:

[Optional]
in_mask: (a pathlike object or string representing a file)
  input mask image for bias field estimation
  argument: ``-mask %s``

bias: (a pathlike object or string representing a file)
  bias field
  argument: ``-bias %s``

cut_file: (a pathlike object or string representing a file)
  the output bias corrected DWI image
  argument: ``%s``, position: -1

nthreads: (an integer (int or long))
  number of threads. if zero, the number of available cpus will be used
  argument: ``-nthreads %d``

grad_file: (a pathlike object or string representing an existing file)
  dw gradient scheme (MRtrix format)
  argument: ``-grad %s``
  mutually_exclusive: grad_fsl

grad_fsl: (a tuple of the form: (a pathlike object or string representing an existing file, a pathlike object or string representing an existing file))
  (bvecs, bvals) dw gradient scheme (FSL format)
  argument: ``-fslgrad %s %s``
  mutually_exclusive: grad_file

bval_scale: ('yes' or 'no')
  specifies whether the b-values should be scaled by the square of the corresponding DW gradient norm, as often required for multishell or DSI DW acquisition schemes. The default action can also be set in the MRtrix config file, under the BValueScaling entry. Valid choices are yes / no, true / false, 0 / 1 (default: true).
  argument: ``-bvalue_scaling %s``

in_bvec: (a pathlike object or string representing an existing file)
  bvecs file in FSL format
  argument: ``-fslgrad %s %s``

in_bval: (a pathlike object or string representing an existing file)
  bvals file in FSL format
 args: (a unicode string)
Additional parameters to the command
argument: `--%s`
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {}) Environment variables

Outputs:

bias: (a pathlike object or string representing an existing file)
  the output bias field
out_file: (a pathlike object or string representing an existing file)
  the output bias corrected DWI image

69.3.2 DWIDenoise

Link to code
Wraps the executable command dwidenoise.
Denoise DWI data and estimate the noise level based on the optimal threshold for PCA.
DWI data denoising and noise map estimation by exploiting data redundancy in the PCA domain using the
prior knowledge that the eigenspectrum of random covariance matrices is described by the universal Marchenko
Pastur distribution.
Important note: image denoising must be performed as the first step of the image processing pipeline. The
routine will fail if interpolation or smoothing has been applied to the data prior to denoising.
Note that this function does not correct for non-Gaussian noise biases.
For more information, see <https://mrtrix.readthedocs.io/en/latest/reference/commands/dwidenoise.html>

Example

```python
>>> import nipype.interfaces.mrtrix3 as mrt
>>> denoise = mrt.DWIDenoise()
>>> denoise.inputs.in_file = 'dwi.mif'
>>> denoise.inputs.mask = 'mask.mif'
>>> denoise.inputs.noise = 'noise.mif'
>>> denoise.cmdline
    # doctest: +ELLIPSIS
dwidenoise --mask mask.mif --noise noise.mif dwi.mif dwi_denoised.mif
>>> denoise.run()
    # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  input DWI image
  argument: `--%s`, position: -2

[Optional]
mask: (a pathlike object or string representing an existing file)
  mask image
  argument: `--mask %s`, position: 1
extent: (a tuple of the form: (an integer (int or long), an integer
  (int or long), an integer (int or long)))
  set the window size of the denoising filter. (default = 5,5,5)
  argument: `--extent %d,%d,%d`
noise: (a pathlike object or string representing a file)
  the output noise map
argument: `-noise %s`

out_file: (a pathlike object or string representing a file)
the output denoised DWI image

argument: `%s`, position: -1

nthreads: (an integer (int or long))
number of threads. if zero, the number of available cpus will be used

argument: `-nthreads %d`

grad_file: (a pathlike object or string representing an existing file)
dw gradient scheme (MRTRix format)

argument: `-grad %s`
mutually_exclusive: grad_fsl

grad_fsl: (a tuple of the form: (a pathlike object or string representing an existing file, a pathlike object or string representing an existing file))
(bvecs, bvals) dw gradient scheme (FSL format)

argument: `%-fslgrad %s %s`
mutually_exclusive: grad_file

bval_scale: (‘yes’ or ‘no’)
specifies whether the b-values should be scaled by the square of the corresponding DW gradient norm, as often required for multishell or DSI DW acquisition schemes. The default action can also be set in the MRtrix config file, under the BValueScaling entry. Valid choices are yes / no, true / false, 0 / 1 (default: true).

argument: `%-bvalue_scaling %s`

in_bvec: (a pathlike object or string representing an existing file)
bvecs file in FSL format

argument: `%-fslgrad %s %s`

in_bval: (a pathlike object or string representing an existing file)
bvals file in FSL format

args: (a unicode string)
Additional parameters to the command

argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

noise: (a pathlike object or string representing an existing file)
the output noise map

out_file: (a pathlike object or string representing an existing file)
the output denoised DWI image

---

**69.3.3 MRDeGibbs**

Link to code
Wraps the executable command mrdegibbs.
Remove Gibbs ringing artefacts.
This application attempts to remove Gibbs ringing artefacts from MRI images using the method of local subvoxel-shifts proposed by Kellner et al.

This command is designed to run on data directly after it has been reconstructed by the scanner, before any interpolation of any kind has taken place. You should not run this command after any form of motion correction (e.g. not after dwipreproc). Similarly, if you intend running dwidenoise, you should run this command afterwards, since it has the potential to alter the noise structure, which would impact on dwidenoise’s performance.
Note that this method is designed to work on images acquired with full k-space coverage. Running this method on partial Fourier (‘half-scan’) data may lead to suboptimal and/or biased results, as noted in the original reference below. There is currently no means of dealing with this; users should exercise caution when using this method on partial Fourier data, and inspect its output for any obvious artefacts.


**Example**

```python
>>> import nipype.interfaces.mrtrix3 as mrt
>>> unring = mrt.MRDeGibbs()
>>> unring.inputs.in_file = 'dwi.mif'
>>> unring.cmdline
'mrdegibbs -axes 0,1 -maxW 3 -minW 1 -nshifts 20 dwi.mif dwi_unr.mif'
>>> unring.run()  # doctest: +SKIP
```

**Inputs:**

[Mandatory]

<table>
<thead>
<tr>
<th>in_file: (a pathlike object or string representing an existing file)</th>
</tr>
</thead>
<tbody>
<tr>
<td>input DWI image</td>
</tr>
<tr>
<td>argument: <code>'-%s'</code>, position: -2</td>
</tr>
</tbody>
</table>

[Optional]

<table>
<thead>
<tr>
<th>axes: (a list of items which are a value of class 'int', nipype</th>
</tr>
</thead>
<tbody>
<tr>
<td>default value: [0, 1])</td>
</tr>
<tr>
<td>indicate the plane in which the data was acquired (axial = 0,1;</td>
</tr>
<tr>
<td>coronal = 0,2; sagittal = 1,2</td>
</tr>
<tr>
<td>argument: <code>'-%axes %s'</code></td>
</tr>
<tr>
<td>nshifts: (an integer (int or long), nipype default value: 20)</td>
</tr>
<tr>
<td>discretization of subpixel spacing (default = 20)</td>
</tr>
<tr>
<td>argument: <code>'-%nshifts %d'</code></td>
</tr>
<tr>
<td>minW: (an integer (int or long), nipype default value: 1)</td>
</tr>
<tr>
<td>left border of window used for total variation (TV) computation</td>
</tr>
<tr>
<td>(default = 1)</td>
</tr>
<tr>
<td>argument: <code>'-%minW %d'</code></td>
</tr>
<tr>
<td>maxW: (an integer (int or long), nipype default value: 3)</td>
</tr>
<tr>
<td>right border of window used for total variation (TV) computation</td>
</tr>
<tr>
<td>(default = 3)</td>
</tr>
<tr>
<td>argument: <code>'-%maxW %d'</code></td>
</tr>
<tr>
<td>out_file: (a pathlike object or string representing a file)</td>
</tr>
<tr>
<td>the output unringed DWI image</td>
</tr>
<tr>
<td>argument: <code>'-%s'</code>, position: -1</td>
</tr>
<tr>
<td>nthreads: (an integer (int or long))</td>
</tr>
<tr>
<td>number of threads. if zero, the number of available cpus will be</td>
</tr>
<tr>
<td>used</td>
</tr>
<tr>
<td>argument: <code>'-%nthreads %d'</code></td>
</tr>
<tr>
<td>grad_file: (a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>dw gradient scheme (MRTRix format)</td>
</tr>
<tr>
<td>argument: <code>'-%grad %s'</code></td>
</tr>
<tr>
<td>mutually_exclusive: grad_fsl</td>
</tr>
<tr>
<td>grad_fsl: (a tuple of the form: (a pathlike object or string</td>
</tr>
<tr>
<td>representing an existing file, a pathlike object or string</td>
</tr>
<tr>
<td>representing an existing file))</td>
</tr>
<tr>
<td>(bvecs, bvals) dw gradient scheme (FSL format)</td>
</tr>
<tr>
<td>argument: <code>'-%fslgrad %s %s'</code></td>
</tr>
<tr>
<td>mutually_exclusive: grad_file</td>
</tr>
</tbody>
</table>

(continues on next page)
bval_scale: ('yes' or 'no')
specifies whether the b-values should be scaled by the square of
the corresponding DW gradient norm, as often required for multishell
or DSI DW acquisition schemes. The default action can also be set in
the MRtrix config file, under the BValueScaling entry. Valid choices
are yes / no, true / false, 0 / 1 (default: true).
argument: `--bvalue_scaling %s`
in_bvec: (a pathlike object or string representing an existing file)
bvecs file in FSL format
argument: `--fslgrad %s %s`
in_bval: (a pathlike object or string representing an existing file)
bvals file in FSL format
args: (a unicode string)
    Additional parameters to the command
    argument: `--%s`
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})
Environment variables

Outputs:
out_file: (a pathlike object or string representing an existing file)
    the output unringed DWI image

69.3.4 ResponseSD

Link to code
Wraps the executable command dwi2response.
Estimate response function(s) for spherical deconvolution using the specified algorithm.

Example

```python
>>> import nipype.interfaces.mrtrix3 as mrt
>>> resp = mrt.ResponseSD()
>>> resp.inputs.in_file = 'dwi.mif'
>>> resp.inputs.algorithm = 'tournier'
>>> resp.inputs.grad_fsl = ('bvecs', 'bvals')
>>> resp.cmdline
# doctest: +ELLIPSIS
'dwi2response tournier -fslgrad bvecs bvals dwi.mif wm.txt'
>>> resp.run()
# doctest: +SKIP
```

We can also pass in multiple harmonic degrees in the case of multi-shell:
```python
>>> resp.inputs.max_sh = [6,8,10]
>>> resp.cmdline 'dwi2response tournier -fslgrad bvecs bvals -lmax 6,8,10 dwi.mif wm.txt'
```

Inputs:

[Mandatory]
algorithm: ('msmt_5tt' or 'dhollander' or 'tournier' or 'tax')
    response estimation algorithm (multi-tissue)
    argument: `--%s`, position: 1
in_file: (a pathlike object or string representing an existing file)
    input DWI image
    argument: `--%s`, position: -5

[Optional]
mtt_file: (a pathlike object or string representing a file)


```plaintext

input 5tt image
argument: "'\%s'", position: -4

wm_file: (a pathlike object or string representing a file, nipype
default value: wm.txt)
output WM response text file
argument: "'\%s'", position: -3

gm_file: (a pathlike object or string representing a file)
output GM response text file
argument: "'\%s'", position: -2

csf_file: (a pathlike object or string representing a file)
output CSF response text file
argument: "'\%s'", position: -1

in_mask: (a pathlike object or string representing an existing file)
provide initial mask image
argument: "'\-mask %s'"

max_sh: (a list of items which are an integer (int or long))
maximum harmonic degree of response function - single value for
single-shell response, list for multi-shell response
argument: "'\-lmax %s'"

nthreads: (an integer (int or long))
number of threads. if zero, the number of available cpus will be
used
argument: "'\-nthreads %d'"

grad_file: (a pathlike object or string representing an existing
file)
dw gradient scheme (MRtrix format)
argument: "'\-grad %s'"
mutually_exclusive: grad_fsl

grad_fsl: (a tuple of the form: (a pathlike object or string
representing an existing file, a pathlike object or string
representing an existing file))
(bvecs, bvals) dw gradient scheme (FSL format)
argument: "'\-fslgrad %s %s'"
mutually_exclusive: grad_file

bval_scale: ('yes' or 'no')
specifies whether the b - values should be scaled by the square of
the corresponding DW gradient norm, as often required for multishell
or DSI DW acquisition schemes. The default action can also be set in
the MRtrix config file, under the BValueScaling entry. Valid choices
are yes / no, true / false, 0 / 1 (default: true).
argument: "'\-bvalue_scaling %s'"

in_bvec: (a pathlike object or string representing an existing file)
bvecs file in FSL format
argument: "'\-fslgrad %s %s'"

in_bval: (a pathlike object or string representing an existing file)
bvals file in FSL format

args: (a unicode string)
Additional parameters to the command
argument: "'\%s'"

environ: (a dictionary with keys which are a bytes or None or a value
class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {}}
Environment variables
```

**Outputs:**

- `wm_file`: (a pathlike object or string representing a file)
output WM response text file
argument: `\%s`
gm_file: (a pathlike object or string representing a file)
output GM response text file
argument: `\%s`
csf_file: (a pathlike object or string representing a file)
output CSF response text file
argument: `\%s`

69.4 interfaces.mrtrix3.reconst

69.4.1 EstimateFOD

Link to code
Wraps the executable command dwi2fod.
Estimate fibre orientation distributions from diffusion data using spherical deconvolution

Example

```python
>>> import nipype.interfaces.mrtrix3 as mrt
>>> fod = mrt.EstimateFOD()
>>> fod.inputs.algorithm = 'csd'
>>> fod.inputs.in_file = 'dwi.mif'
>>> fod.inputs.wm_txt = 'wm.txt'
>>> fod.inputs.grad_fsl = ('bvecs', 'bvals')
>>> fod.cmdline
# doctest: +ELLIPSIS
'dwi2fod -fslgrad bvecs bvals -lmax 8 csd dwi.mif wm.txt wm.mif gm.mif csf.mif'
```

Inputs:

[Mandatory]
algorithm: ('csd' or 'msmt_csd')
  FOD algorithm
  argument: `\%s`, position: -8
in_file: (a pathlike object or string representing an existing file)
  input DWI image
  argument: `\%s`, position: -7
wm_txt: (a pathlike object or string representing a file)
  WM response text file
  argument: `\%s`, position: -6
wm_odf: (a pathlike object or string representing a file, nipype
default value: wm.mif)
  output WM ODF
  argument: `\%s`, position: -5

[Optional]

gm_txt: (a pathlike object or string representing a file)
  GM response text file
  argument: `\%s`, position: -4
gm_odf: (a pathlike object or string representing a file, nipype
default value: gm.mif)
  output GM ODF
  argument: `\%s`, position: -3
csf_txt: (a pathlike object or string representing a file)
CSF response text file
argument: `-%s`, position: -2

csf_odf: (a pathlike object or string representing a file, nipype
default value: csf.mif)
output CSF ODF
argument: `-%s`, position: -1

mask_file: (a pathlike object or string representing an existing
file)
mask image
argument: `--mask %s`

shell: (a list of items which are a float)
specify one or more dw gradient shells
argument: `--shell %s`

max_sh: (a list of items which are an integer (int or long), nipype
default value: [8])
maximum harmonic degree of response function - single value for
single-shell response, list for multi-shell response
argument: `--lmax %s`

in_dirs: (a pathlike object or string representing an existing file)
specify the directions over which to apply the non-negativity
constraint (by default, the built-in 300 direction set is used).
These should be supplied as a text file containing the [ az el ]
pairs for the directions.
argument: `--directions %s`
nthreads: (an integer (int or long))
number of threads. if zero, the number of available cpus will be
used
argument: `--nthreads %d`

grad_file: (a pathlike object or string representing an existing
file)
dw gradient scheme (MRTrix format)
argument: `--grad %s`

mutually_exclusive: grad_fsl

grad_fsl: (a tuple of the form: (a pathlike object or string
representing an existing file, a pathlike object or string
representing an existing file))
(bvecs, bvals) dw gradient scheme (FSL format)
argument: `--fslgrad %s %s`
mutually_exclusive: grad_file

bval_scale: (`yes` or `no`) specifies whether the b - values should be scaled by the square of
the corresponding D\text{W} gradient norm, as often required for multishell
or DSI DW acquisition schemes. The default action can also be set in
the MRtrix config file, under the BValueScaling entry. Valid choices
are yes / no, true / false, 0 / 1 (default: true).
argument: `--bvalue_scaling %s`

in_bvec: (a pathlike object or string representing an existing file)
bvecs file in FSL format
argument: `--fslgrad %s %s`

in_bval: (a pathlike object or string representing an existing file)
bvals file in FSL format
args: (a unicode string)
Additional parameters to the command
argument: `-%s`
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

<table>
<thead>
<tr>
<th>Outputs:</th>
</tr>
</thead>
<tbody>
<tr>
<td>wm_odf: (a pathlike object or string representing a file)</td>
</tr>
<tr>
<td>output WM ODF</td>
</tr>
<tr>
<td>argument: <code>%s</code></td>
</tr>
<tr>
<td>gm_odf: (a pathlike object or string representing a file)</td>
</tr>
<tr>
<td>output GM ODF</td>
</tr>
<tr>
<td>argument: <code>%s</code></td>
</tr>
<tr>
<td>csf_odf: (a pathlike object or string representing a file)</td>
</tr>
<tr>
<td>output CSF ODF</td>
</tr>
<tr>
<td>argument: <code>%s</code></td>
</tr>
</tbody>
</table>

### 69.4.2 FitTensor

**Link to code**

Wraps the executable command *dwi2tensor*.

Convert diffusion-weighted images to tensor images

**Example**

```python
>>> import nipype.interfaces.mrtrix3 as mrt
>>> tsr = mrt.FitTensor()
>>> tsr.inputs.in_file = 'dwi.mif'
>>> tsr.inputs.in_mask = 'mask.nii.gz'
>>> tsr.inputs.grad_fsl = ('bvecs', 'bvals')
>>> tsr.cmdline
'dwi2tensor -fslgrad bvecs bvals -mask mask.nii.gz dwi.mif dti.mif'
>>> tsr.run()
```

**Inputs:**

[Mandatory]

- *in_file*: (a pathlike object or string representing an existing file)
  - input diffusion weighted images
  - argument: ``%s``, position: -2

- *out_file*: (a pathlike object or string representing a file, nipype default value: *dti.mif*)
  - the output diffusion tensor image
  - argument: ``%s``, position: -1

[Optional]

- *in_mask*: (a pathlike object or string representing an existing file)
  - only perform computation within the specified binary brain mask image
  - argument: ``-mask %s``

- *method*: ('nonlinear' or 'loglinear' or 'sech' or 'rician')
  - select method used to perform the fitting
  - argument: ``-method %s``

- *reg_term*: (a float)
  - specify the strength of the regularisation term on the magnitude of the tensor elements (default = 5000). This only applies to the non-linear methods
  - argument: ``-regularisation %f``

- *nthreads*: (an integer (int or long))
number of threads. If zero, the number of available CPUs will be used.
argument: `--nthreads %d`
grad_file: (a pathlike object or string representing an existing file)
dw gradient scheme (MRTrax format)
argument: `--grad %s`
mutually exclusive: grad_fsl
grad_fsl: (a tuple of the form: (a pathlike object or string representing an existing file, a pathlike object or string representing an existing file))
(bvecs, bvals) dw gradient scheme (FSL format)
argument: `--fslgrad %s %s`
mutually exclusive: grad_file
bval_scale: ('yes' or 'no')
specifies whether the b-values should be scaled by the square of the corresponding DW gradient norm, as often required for multishell or DSI DW acquisition schemes. The default action can also be set in the MRtrax config file, under the BValueScaling entry. Valid choices are yes / no, true / false, 0 / 1 (default: true).
argument: `--bvalue_scaling %s`
in_bvec: (a pathlike object or string representing an existing file)
bvecs file in FSL format
argument: `--fslgrad %s %s`
in_bval: (a pathlike object or string representing an existing file)
bvals file in FSL format
args: (a unicode string)
Additional parameters to the command
argument: `%%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
the output DTI file

69.5 interfaces.mrtrix3.tracking

69.5.1 Tractography

Link to code
Wraps the executable command tckgen.
Performs streamlines tractography after selecting the appropriate algorithm.

Example

```python
>>> import nipype.interfaces.mrtrix3 as mrt
>>> tk = mrt.Tractography()
>>> tk.inputs.in_file = 'fods.mif'
>>> tk.inputs.roi_mask = 'mask.nii.gz'
>>> tk.inputs.seed_sphere = (80, 100, 70, 10)
>>> tk.cmdline
# doctest: +ELLIPSIS
```

(continues on next page)
'tckgen -algorithm iFOD2 -samples 4 -output_seeds out_seeds.nii.gz -mask mask.nii.gz -seed_sphere 80.000000,100.000000,70.000000,10.000000 fods.mif tracked.tck'
>>> tk.run()  # doctest: +SKIP

Inputs:

<table>
<thead>
<tr>
<th>Mandatory</th>
</tr>
</thead>
</table>
|in_file: (a pathlike object or string representing an existing file)
  input file to be processed
  argument: ``%s``, position: -2|
|out_file: (a pathlike object or string representing a file, nipype default value: tracked.tck)
  output file containing tracks
  argument: ``%s``, position: -1|

<table>
<thead>
<tr>
<th>Optional</th>
</tr>
</thead>
</table>
|sph_trait: (a tuple of the form: (a float, a float, a float, a float))
  argument: ``%f,%f,%f,%f``|
|algorithm: ('iFOD2' or 'FACT' or 'iFOD1' or 'Nulldist' or 'SD_Stream' or 'Tensor_Det' or 'Tensor_Prob', nipype default value: iFOD2)
  tractography algorithm to be used
  argument: ``-algorithm %s``|
|roi_incl: (a pathlike object or string representing an existing file or a tuple of the form: (a float, a float, a float, a float))
  specify an inclusion region of interest, streamlines must traverse ALL inclusion regions to be accepted
  argument: ``-include %s``|
|roi_excl: (a pathlike object or string representing an existing file or a tuple of the form: (a float, a float, a float, a float))
  specify an exclusion region of interest, streamlines that enter ANY exclude region will be discarded
  argument: ``-exclude %s``|
|roi_mask: (a pathlike object or string representing an existing file or a tuple of the form: (a float, a float, a float, a float))
  specify a masking region of interest. If defined, streamlines exiting the mask will be truncated
  argument: ``-mask %s``|
|step_size: (a float)
  set the step size of the algorithm in mm (default is 0.1 x voxelsize; for iFOD2: 0.5 x voxelsize)
  argument: ``-step %f``|
|angle: (a float)
  set the maximum angle between successive steps (default is 90deg x stepsize / voxelsize)
  argument: ``-angle %f``|
|n_tracks: (an integer (int or long))
  set the desired number of tracks. The program will continue to generate tracks until this number of tracks have been selected and written to the output file
  argument: ``-number %d``|
|select: (an integer (int or long))
  set the desired number of tracks. The program will continue to generate tracks until this number of tracks have been selected and written to the output file
  argument: ``-select %d``|
|max_tracks: (an integer (int or long))

(continues on next page)
set the maximum number of tracks to generate. The program will not
generate more tracks than this number, even if the desired number of
tracks hasn't yet been reached (default is 100 x number)
argument: `-maxnum %d`

max_length: (a float)
set the maximum length of any track in mm (default is 100 x
voxelsize)
argument: `-maxlength %f`

min_length: (a float)
set the minimum length of any track in mm (default is 5 x voxelsize)
argument: `-minlength %f`

cutoff: (a float)
set the FA or FOD amplitude cutoff for terminating tracks (default
is 0.1)
argument: `-cutoff %f`
cutoff_init: (a float)
set the minimum FA or FOD amplitude for initiating tracks (default
is the same as the normal cutoff)
argument: `-initcutoff %f`

n_trials: (an integer (int or long))
set the maximum number of sampling trials at each point (only used
for probabilistic tracking)
argument: `-trials %d`

unidirectional: (a boolean)
track from the seed point in one direction only (default is to track
in both directions)
argument: `-unidirectional`

init_dir: (a tuple of the form: (a float, a float, a float)))
specify an initial direction for the tracking (this should be
supplied as a vector of 3 comma-separated values
argument: `-initdirection %f,%f,%f`

noprecompt: (a boolean)
do NOT pre-compute legendre polynomial values. Warning: this will
slow down the algorithm by a factor of approximately 4
argument: `-noprecomputed`

power: (an integer (int or long))aise the FOD to the power specified (default is 1/nsamples)
argument: `-power %d`

n_samples: (an integer (int or long), nipype default value: 4)
set the number of FOD samples to take per step for the 2nd order
(iFOD2) method
argument: `-samples %d`

use_rk4: (a boolean)
use 4th-order Runge-Kutta integration (slower, but eliminates
curvature overshoot in 1st-order deterministic methods)
argument: `-rk4`

stop: (a boolean)
stop propagating a streamline once it has traversed all include
regions
argument: `-stop`

downsamp: (a float)
downsample the generated streamlines to reduce output file size
argument: `-downsample %f`

act_file: (a pathlike object or string representing an existing file)
use the Anatomically-Constrained Tractography framework during
tracking; provided image must be in the 5TT (five - tissue - type)
format
argument: `'-act %s'`
backtrack: (a boolean)
  allow tracks to be truncated
argument: `'-backtrack'
crop_at_gmwmi: (a boolean)
  crop streamline endpoints more precisely as they cross the GM-WM
  interface
argument: `'-crop_at_gmwmi'
seed_sphere: (a tuple of the form: (a float, a float, a float, a float))
  spherical seed
argument: `'-seed_sphere %f,%f,%f,%f'
seed_image: (a pathlike object or string representing an existing file)
  seed streamlines entirely at random within mask
argument: `'-seed_image %s'
seed_rnd_voxel: (a tuple of the form: (a pathlike object or string representing an existing file, an integer (int or long)))
  seed a fixed number of streamlines per voxel in a mask image; random placement of seeds in each voxel
argument: `'-seed_random_per_voxel %s %d'
mutually_exclusive: seed_image, seed_grid_voxel
seed_grid_voxel: (a tuple of the form: (a pathlike object or string representing an existing file, an integer (int or long)))
  seed a fixed number of streamlines per voxel in a mask image; place seeds on a 3D mesh grid (grid_size argument is per axis; so a grid_size of 3 results in 27 seeds per voxel)
argument: `'-seed_grid_per_voxel %s %d'
mutually_exclusive: seed_image, seed_rnd_voxel
seed_rejection: (a pathlike object or string representing an existing file)
  seed from an image using rejection sampling (higher values = more probable to seed from
argument: `'-seed_rejection %s'
seed_gmwmi: (a pathlike object or string representing an existing file)
  seed from the grey matter - white matter interface (only valid if using ACT framework)
argument: `'-seed_gmwmi %s'
requires: act_file
seed_dynamic: (a pathlike object or string representing an existing file)
  determine seed points dynamically using the SIFT model (must not provide any other seeding mechanism). Note that while this seeding mechanism improves the distribution of reconstructed streamlines density, it should NOT be used as a substitute for the SIFT method itself.
argument: `'-seed_dynamic %s'
max_seed_attempts: (an integer (int or long))
  set the maximum number of times that the tracking algorithm should attempt to find an appropriate tracking direction from a given seed point
argument: `'-max_seed_attempts %d'
out_seeds: (a pathlike object or string representing a file, nipype default value: out_seeds.nii.gz)
  output the seed location of all successful streamlines to a file
argument: `'-output_seeds %s'"
nthreads: (an integer (int or long))
   number of threads. if zero, the number of available cpus will be
   used
   argument: `--nthreads %d`
grad_file: (a pathlike object or string representing an existing
   file)
   dw gradient scheme (MRtrix format)
   argument: `--grad %s`
   mutually_exclusive: grad_fsl
grad_fsl: (a tuple of the form: (a pathlike object or string
   representing an existing file, a pathlike object or string
   representing an existing file))
   (bvecs, bvals) dw gradient scheme (FSL format)
   argument: `--fslgrad %s %s`
   mutually_exclusive: grad_file
bval_scale: ('yes' or 'no')
   specifies whether the b - values should be scaled by the square of
   the corresponding DW gradient norm, as often required for multishell
   or DSI DW acquisition schemes. The default action can also be set in
   the MRtrix config file, under the BValueScaling entry. Valid choices
   are yes / no, true / false, 0 / 1 (default: true).
   argument: `--bvalue_scaling %s`
in_bvec: (a pathlike object or string representing an existing file)
   bvecs file in FSL format
   argument: `--fslgrad %s %s`
in_bval: (a pathlike object or string representing an existing file)
   bvals file in FSL format
args: (a unicode string)
   Additional parameters to the command
   argument: `%%s`
environ: (a dictionary with keys which are a bytes or None or a value
   of class 'str' and with values which are a bytes or None or a
   value of class 'str', nipype default value: {})
   Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
   the output filtered tracks
out_seeds: (a pathlike object or string representing a file)
   output the seed location of all successful streamlines to a file

69.6 interfaces.mrtrix3.utils

69.6.1 BrainMask

Link to code
Wraps the executable command dwi2mask.
Convert a mesh surface to a partial volume estimation image

Example

```python
>>> import nipype.interfaces.mrtrix3 as mrt
>>> bmsk = mrt.BrainMask()
>>> bmsk.inputs.in_file = 'dwi.mif'
```

(continues on next page)
```
bmsk.cmdline
'dwi2mask dwi.mif brainmask.mif'
```

```
bmsk.run()
```

**Inputs:**

[Mandatory]

- **in_file**: (a pathlike object or string representing an existing file)
  - input diffusion weighted images
  - argument: `"%s"`, position: -2
- **out_file**: (a pathlike object or string representing a file, nipype)
  - default value: brainmask.mif)
  - output brain mask
  - argument: `"%s"`, position: -1

[Optional]

- **nthreads**: (an integer (int or long))
  - number of threads. If zero, the number of available cpus will be used
  - argument: `"-nthreads %d"`
- **grad_file**: (a pathlike object or string representing an existing file)
  - dw gradient scheme (MRTrix format)
  - argument: `"-grad %s"`
  - mutually exclusive: grad_fsl
- **grad_fsl**: (a tuple of the form: (a pathlike object or string representing an existing file, a pathlike object or string representing an existing file))
  - (bvecs, bvals) dw gradient scheme (FSL format)
  - argument: `"-fslgrad %s %s"`
  - mutually exclusive: grad_file
- **bval_scale**: ("yes" or "no")
  - specifies whether the b-values should be scaled by the square of the corresponding DW gradient norm, as often required for multishell or DSI DW acquisition schemes. The default action can also be set in the MRtrix config file, under the BValueScaling entry. Valid choices are yes / no, true / false, 0 / 1 (default: true).
  - argument: `"-bvalue_scaling %s"`
- **in_bvec**: (a pathlike object or string representing an existing file)
  - bvecs file in FSL format
  - argument: `"-fslgrad %s %s"`
- **in_bval**: (a pathlike object or string representing an existing file)
  - bvals file in FSL format
- **args**: (a unicode string)
  - Additional parameters to the command
  - argument: `"%s"`
- **environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: { })
  - Environment variables

**Outputs:**

- **out_file**: (a pathlike object or string representing an existing file)
  - the output response file
69.6.2 ComputeTDI

Link to code
Wraps the executable command `tckmap`.
Use track data as a form of contrast for producing a high-resolution image.

References
- If using -dixel option with TDI contrast only: Smith, R.E., Tournier, J-D., Calamante, F., Connelly, A. A novel paradigm for automated segmentation of very large whole-brain probabilistic tractography data sets. In proc. ISMRM, 2011, 19, 673

Example

```python
>>> import nipype.interfaces.mrtrix3 as mrt
>>> tdi = mrt.ComputeTDI()
>>> tdi.inputs.in_file = 'dti.mif'
>>> tdi.cmdline
'tckmap dti.mif tdi.mif'
>>> tdi.run()
```

Inputs:

<table>
<thead>
<tr>
<th>Mandatory</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in_file</code>: (a pathlike object or string representing an existing file) input tractography</td>
</tr>
<tr>
<td>argument: <code>'\%s'</code>, position: <code>-2</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out_file</code>: (a pathlike object or string representing a file, nipype default value: <code>tdi.mif</code>) output TDI file</td>
</tr>
<tr>
<td>argument: <code>'\%s'</code>, position: <code>-1</code></td>
</tr>
<tr>
<td><code>reference</code>: (a pathlike object or string representing an existing file) a reference image to be used as template</td>
</tr>
<tr>
<td>argument: <code>'\-template %s'</code></td>
</tr>
<tr>
<td><code>vox_size</code>: (a list of items which are an integer (int or long)) voxel dimensions</td>
</tr>
<tr>
<td>argument: <code>'\-vox %s'</code></td>
</tr>
</tbody>
</table>
data_type: ('float' or 'unsigned int')
    specify output image data type
    argument: '--datatype %s'

use_dec: (a boolean)
    perform mapping in DEC space
    argument: '--dec'

dixel: (a pathlike object or string representing a file)
    map streamlines to dixels within each voxel. Directions are stored
    as azimuth elevation pairs.
    argument: '--dixel %s'

max_tod: (an integer (int or long))
    generate a Track Orientation Distribution (TOD) in each voxel.
    argument: '--tod %d'

contrast: ('tdi' or 'length' or 'invlength' or 'scalar_map' or
    'scalar_map_conut' or 'fod_amp' or 'curvature')
    define the desired form of contrast for the output image
    argument: '--contrast %s'

in_map: (a pathlike object or string representing an existing file)
    provide the scalar image map for generating images with 'scalar_map'
    contrasts, or the SHs image for fod_amp
    argument: '--image %s'

stat_vox: ('sum' or 'min' or 'mean' or 'max')
    define the statistic for choosing the final voxel intensities for a
    given contrast
    argument: '--stat_vox %s'

stat_tck: ('mean' or 'sum' or 'min' or 'max' or 'median' or
    'mean_nonzero' or 'gaussian' or 'ends_min' or 'ends_mean' or
    'ends_max' or 'ends_prod')
    define the statistic for choosing the contribution to be made by
    each streamline as a function of the samples taken along their
    lengths.
    argument: '--stat_tck %s'

fwhm_tck: (a float)
    define the statistic for choosing the contribution to be made by
    each streamline as a function of the samples taken along their
    lengths
    argument: '--fwhm_tck %f'

map_zero: (a boolean)
    if a streamline has zero contribution based on the contrast &
    statistic, typically it is not mapped; use this option to still
    contribute to the map even if this is the case (these non-
    contributing voxels can then influence the mean value in each voxel
    of the map)
    argument: '--map_zero'

upsample: (an integer (int or long))
    upsample the tracks by some ratio using Hermite interpolation before
    mapping
    argument: '--upsample %d'

precise: (a boolean)
    use a more precise streamline mapping strategy, that accurately
    quantifies the length through each voxel (these lengths are then
taken into account during TWI calculation)
    argument: '--precise'

ends_only: (a boolean)
    only map the streamline endpoints to the image
    argument: '--ends_only'
tck_weights: (a pathlike object or string representing an existing
file) specify a text scalar file containing the streamline weights argument: ``-tck_weights_in %s``
nthreads: (an integer (int or long)) number of threads. if zero, the number of available cpus will be used argument: ``-nthreads %d``
args: (a unicode string) Additional parameters to the command argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

| out_file: (a pathlike object or string representing a file) output TDI file |

69.6.3 DWIExtract

Link to code

Wraps the executable command dwiextract. Extract diffusion-weighted volumes, b=0 volumes, or certain shells from a DWI dataset

Example

```
>>> import nipype.interfaces.mrtrix3 as mrt
>>> dwiextract = mrt.DWIExtract()
>>> dwiextract.inputs.in_file = 'dwi.mif'
>>> dwiextract.inputs.bzero = True
>>> dwiextract.inputs.out_file = 'b0vols.mif'
>>> dwiextract.inputs.grad_fsl = ('bvecs', 'bvals')
>>> dwiextract.cmdline
"dwiextract -bzero -fslgrad bvecs bvals dwi.mif b0vols.mif"
>>> dwiextract.run()  # doctest: +ELLIPSIS

Inputs:

<table>
<thead>
<tr>
<th>[Mandatory] in_file: (a pathlike object or string representing an existing file) input image</th>
</tr>
</thead>
<tbody>
<tr>
<td>argument: ````%s```, position: -2</td>
</tr>
<tr>
<td>out_file: (a pathlike object or string representing a file) output image</td>
</tr>
<tr>
<td>argument: ````%s```, position: -1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[Optional] bzero: (a boolean) extract b=0 volumes</th>
</tr>
</thead>
<tbody>
<tr>
<td>argument: <code>-bzero</code></td>
</tr>
<tr>
<td>nobzero: (a boolean) extract non b=0 volumes</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>argument: <code>-no_bzero</code></td>
</tr>
<tr>
<td>singleshell: (a boolean)</td>
</tr>
</tbody>
</table>

(continues on next page)
extract volumes with a specific shell
argument: `''-singleshell``
shell: (a list of items which are a float)
specify one or more gradient shells
argument: `''-shell %s``
nthreads: (an integer (int or long))
number of threads. if zero, the number of available cpus will be
used
argument: `''-nthreads %d``
grad_file: (a pathlike object or string representing an existing file)
dw gradient scheme (MRTrix format)
argument: `''-grad %s``
mutually_exclusive: grad_fsl
grad_fsl: (a tuple of the form: (a pathlike object or string representing an existing file, a pathlike object or string representing an existing file))
(bvecs, bvals) dw gradient scheme (FSL format)
argument: `''-fslgrad %s %s``
mutually_exclusive: grad_file
bval_scale: ("yes" or "no")
specifies whether the b - values should be scaled by the square of the corresponding DW gradient norm, as often required for multishell or DSI DW acquisition schemes. The default action can also be set in the MRtrix config file, under the BValueScaling entry. Valid choices are yes / no, true / false, 0 / 1 (default: true).
argument: `''-bvalue_scaling %s``
in_bvec: (a pathlike object or string representing an existing file)
bvecs file in FSL format
argument: `''-fslgrad %s %s``
in_bval: (a pathlike object or string representing an existing file)
bvals file in FSL format
args: (a unicode string)
Additional parameters to the command
argument: `''%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
output image

69.6.4 Generate5tt

Link to code
Wraps the executable command 5ttgen.
Generate a 5TT image suitable for ACT using the selected algorithm

Example

```python
>>> import nipype.interfaces.mrtrix3 as mrt
>>> gen5tt = mrt.Generate5tt()
```
>>> gen5tt.inputs.in_file = 'T1.nii.gz'
>>> gen5tt.inputs.algorithm = 'fsl'
>>> gen5tt.inputs.out_file = '5tt.mif'
>>> gen5tt.cmdline
# doctest: +ELLIPSIS
'5ttgen fsl T1.nii.gz 5tt.mif'
>>> gen5tt.run()
# doctest: +SKIP

Inputs:

[Mandatory]
algorithm: ('fsl' or 'gif' or 'freesurfer')
tissue segmentation algorithm
argument: ``%s``
position: -3

in_file: (a pathlike object or string representing an existing file)
input image
argument: ``%s``
position: -2

out_file: (a pathlike object or string representing a file)
output image
argument: ``%s``
position: -1

[Optional]

nthreads: (an integer (int or long))
number of threads. if zero, the number of available cpus will be used
argument: ``-nthreads %d``

grad_file: (a pathlike object or string representing an existing file)
dw gradient scheme (MRtrix format)
argument: ``-grad %s``
mutually_exclusive: grad_fsl

grad_fsl: (a tuple of the form: (a pathlike object or string representing an existing file, a pathlike object or string representing an existing file))
(bvecs, bvals) dw gradient scheme (FSL format)
argument: ``-fslgrad %s %s``
mutually_exclusive: grad_file

bval_scale: ('yes' or 'no')
specifies whether the b-values should be scaled by the square of the corresponding DW gradient norm, as often required for multishell or DSI DW acquisition schemes. The default action can also be set in the MRtrix config file, under the BValueScaling entry. Valid choices are yes / no, true / false, 0 / 1 (default: true).
argument: ``-bvalue_scaling %s``

in_bvec: (a pathlike object or string representing an existing file)
bvecs file in FSL format
argument: ``-fslgrad %s %s``

in_bval: (a pathlike object or string representing an existing file)
bvals file in FSL format

args: (a unicode string)
Additional parameters to the command
argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:
**69.6.5 MRConvert**

Link to code

Wraps the executable command `mrconvert`. Perform conversion between different file types and optionally extract a subset of the input image.

**Example**

```python
>>> import nipype.interfaces.mrtrix3 as mrt
>>> mrconvert = mrt.MRConvert()
>>> mrconvert.inputs.in_file = 'dwi.nii.gz'
>>> mrconvert.inputs.grad_fsl = ('bvecs', 'bvals')
>>> mrconvert.cmdline # doctest: +ELLIPSIS
'mrconvert -fslgrad bvecs bvals dwi.nii.gz dwi.mif'
>>> mrconvert.run() # doctest: +SKIP
```

**Inputs:**

- **in_file**: (a pathlike object or string representing an existing file)
  - input image
  - argument: ``%s``, position: -2
- **out_file**: (a pathlike object or string representing a file, nipype default value: dwi.mif)
  - output image
  - argument: ``%s``, position: -1

- **coord**: (a list of items which are a float)
  - extract data at the specified coordinates
  - argument: ``-coord %s``
- **vox**: (a list of items which are a float)
  - change the voxel dimensions
  - argument: ``-vox %s``
- **axes**: (a list of items which are an integer (int or long))
  - specify the axes that will be used
  - argument: ``-axes %s``
- **scaling**: (a list of items which are a float)
  - specify the data scaling parameter
  - argument: ``-scaling %s``
- **nthreads**: (an integer (int or long))
  - number of threads. if zero, the number of available cpus will be used
  - argument: ``-nthreads %d``
- **grad_file**: (a pathlike object or string representing an existing file)
  - dw gradient scheme (MRtrix format)
  - argument: ``-grad %s``
  - mutually_exclusive: grad_fsl
- **grad_fsl**: (a tuple of the form: (a pathlike object or string representing an existing file, a pathlike object or string representing an existing file))
  - (bvecs, bvals) dw gradient scheme (FSL format)

(continues on next page)
arguments: '$fslgrad %s %s$
mutually_exclusive: grad_file

bval_scale: ('yes' or 'no')
specifies whether the b-values should be scaled by the square of the corresponding DW gradient norm, as often required for multishell or DSI DW acquisition schemes. The default action can also be set in the MRtrix config file, under the BValueScaling entry. Valid choices are yes/no, true/false, 0/1 (default: true).

arguments: '$-bvalue_scaling %s$

in_bvec: (a pathlike object or string representing existing file)
bvecs file in FSL format

arguments: '$fslgrad %s %s$

in_bval: (a pathlike object or string representing existing file)
bvals file in FSL format

args: (a unicode string)
Additional parameters to the command

arguments: '$$'

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

Environment variables

 Outputs:

out_file: (a pathlike object or string representing existing file)
output image

69.6.6 MRMath

Link to code
Wraps the executable command mrmath.
Compute summary statistic on image intensities along a specified axis of a single image

Example

```python
>>> import nipype.interfaces.mrtrix3 as mrt
>>> mrmath = mrt.MRMath()
>>> mrmath.inputs.in_file = 'dwi.mif'
>>> mrmath.inputs.operation = 'mean'
>>> mrmath.inputs.axis = 3
>>> mrmath.inputs.out_file = 'dwi_mean.mif'
>>> mrmath.inputs.grad_fsl = ('bvecs', 'bvals')
>>> mrmath.cmdline
# doctest: +ELLIPSIS
'mrmath -axis 3 -fslgrad bvecs bvals dwi.mif mean dwi_mean.mif'
>>> mrmath.run()
# doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing existing file)
input image

arguments: '$%s$', position: -3

out_file: (a pathlike object or string representing a file)
output image

arguments: '$%s$', position: -1

operation: ('mean' or 'median' or 'sum' or 'product' or 'rms' or...
'norm' or 'var' or 'std' or 'min' or 'max' or 'absmax' or 'magmax')
operation to computer along a specified axis
argument: ``\`%s```, position: -2

[Optional]
axis: (an integer (int or long))
specified axis to perform the operation along
argument: ``\`-axis %d``
nthreads: (an integer (int or long))
number of threads. if zero, the number of available cpus will be used
argument: ``\`-nthreads %d``
grad_file: (a pathlike object or string representing an existing file)
dw gradient scheme (MRtrix format)
argument: ``\`-grad %s``
mutually_exclusive: grad_fsl
grad_fsl: (a tuple of the form: (a pathlike object or string representing an existing file, a pathlike object or string representing an existing file))
(bvecs, bvals) dw gradient scheme (FSL format)
argument: ``\`-fslgrad %s %s``
mutually_exclusive: grad_file
bval_scale: ('yes' or 'no')
specifies whether the b-values should be scaled by the square of the corresponding DW gradient norm, as often required for multishell or DSI DW acquisition schemes. The default action can also be set in the MRtrix config file, under the BValueScaling entry. Valid choices are yes / no, true / false, 0 / 1 (default: true).
argument: ``\`-bvalue_scaling %s``
in_bvec: (a pathlike object or string representing an existing file)
bvecs file in FSL format
argument: ``\`-fslgrad %s %s``
in_bval: (a pathlike object or string representing an existing file)
bvals file in FSL format
args: (a unicode string)
Additional parameters to the command
argument: ``\`%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: { })
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
output image

69.6.7 Mesh2PVE

Link to code
Wraps the executable command mesh2pve.
Convert a mesh surface to a partial volume estimation image
**Example**

```python
>>> import nipype.interfaces.mrtrix3 as mrt
>>> m2p = mrt.Mesh2PVE()
>>> m2p.inputs.in_file = 'surf1.vtk'
>>> m2p.inputs.reference = 'dwi.mif'
>>> m2p.inputs.in_first = 'T1.nii.gz'
>>> m2p.cmdline
# doctest: +ELLIPSIS
'mesh2pve -first T1.nii.gz surf1.vtk dwi.mif mesh2volume.nii.gz'
>>> m2p.run()
# doctest: +SKIP
```

**Inputs:**

- **in_file**: (a pathlike object or string representing an existing file)
  - input mesh
    - argument: `'\%s'`, position: -3
  - reference: (a pathlike object or string representing an existing file)
    - input reference image
      - argument: `'\%s'`, position: -2
  - out_file: (a pathlike object or string representing a file, nipype default value: mesh2volume.nii.gz)
    - output file containing SH coefficients
      - argument: `'\%s'`, position: -1

- **in_first**: (a pathlike object or string representing an existing file)
  - indicates that the mesh file is provided by FSL FIRST
    - argument: `'-first \%s'`

**Args:** (a unicode string)

- Additional parameters to the command
  - argument: `'\%s'`

**Environ:** (a dictionary with keys which are bytes or None or a value of class 'str' and with values which are bytes or None or a value of class 'str', nipype default value: `{}`)

**Environment variables**

**Outputs:**

- **out_file**: (a pathlike object or string representing an existing file)
  - the output response file

### 69.6.8 TCK2VTK

**Link to code**

Wraps the executable command `tck2vtk`.

Convert a track file to a vtk format, cave: coordinates are in XYZ coordinates not reference

**Example**

```python
>>> import nipype.interfaces.mrtrix3 as mrt
>>> vtk = mrt.TCK2VTK()
>>> vtk.inputs.in_file = 'tracks.tck'
>>> vtk.inputs.reference = 'b0.nii'
>>> vtk.cmdline
# doctest: +ELLIPSIS
'(continues on next page)'
```
'tck2vtk -image b0.nii tracks.tck tracks.vtk'
>>> vtk.run()  # doctest: +SKIP

## Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
   input tractography
   argument: `'\%s'`, position: -2

[Optional]
out_file: (a pathlike object or string representing a file, nipype
   default value: tracks.vtk)
   output VTK file
   argument: `'\%s'`, position: -1
reference: (a pathlike object or string representing an existing
   file)
   if specified, the properties of this image will be used to convert
   track point positions from real (scanner) coordinates into image
   coordinates (in mm).
   argument: `'\-image %s'`
voxel: (a pathlike object or string representing an existing file)
   if specified, the properties of this image will be used to convert
   track point positions from real (scanner) coordinates into image
   coordinates.
   argument: `'\-image %s'`
nthreads: (an integer (int or long))
   number of threads. if zero, the number of available cpus will be
   used
   argument: `'\-nthreads %d'`
args: (a unicode string)
   Additional parameters to the command
   argument: `'\%s'`
environ: (a dictionary with keys which are a bytes or None or a value
   of class 'str' and with values which are a bytes or None or a
   value of class 'str', nipype default value: {}}
   Environment variables

## Outputs:

out_file: (a pathlike object or string representing a file)
   output VTK file

### 69.6.9 TensorMetrics

Link to code
Wraps the executable command `tensor2metric`. Compute metrics from tensors

**Example**

```python
>>> import nipype.interfaces.mrtrix3 as mrt
>>> comp = mrt.TensorMetrics()
>>> comp.inputs.in_file = 'dti.mif'
>>> comp.inputs.out_fa = 'fa.mif'
>>> comp.cmdline  # doctest: +ELLIPSIS
tck2metric -image b0.nii -nthreads 1
```
'tensor2metric -num 1 -fa fa.mif dti.mif'
>>> comp.run()  # doctest: +SKIP

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  input DTI image
  argument: ``%-s``, position: -1

[Optional]
out_fa: (a pathlike object or string representing a file)
  output FA file
  argument: ``-fa %s``
out_adc: (a pathlike object or string representing a file)
  output ADC file
  argument: ``-adc %s``
out_evec: (a pathlike object or string representing a file)
  output selected eigenvector(s) file
  argument: ``-vector %s``
out_eval: (a pathlike object or string representing a file)
  output selected eigenvalue(s) file
  argument: ``-value %s``
component: (a list of items which are any value, nipype default value: [1])
  specify the desired eigenvalue/eigenvector(s). Note that several
  eigenvalues can be specified as a number sequence
  argument: ``-num %s``
in_mask: (a pathlike object or string representing an existing file)
  only perform computation within the specified binary brain mask
  image
  argument: ``-mask %s``
modulate: ('FA' or 'none' or 'eval')
  how to modulate the magnitude of the eigenvectors
  argument: ``-modulate %s``
args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
  Environment variables

Outputs:

out_fa: (a pathlike object or string representing a file)
  output FA file
out_adc: (a pathlike object or string representing a file)
  output ADC file
out_evec: (a pathlike object or string representing a file)
  output selected eigenvector(s) file
out_eval: (a pathlike object or string representing a file)
  output selected eigenvalue(s) file
70.1 interfaces.niftyfit.asl

70.1.1 FitAsl

Link to code
Wraps the executable command fit_asl.
Interface for executable fit_asl from Niftyfit platform.
Use NiftyFit to perform ASL fitting.
ASL fitting routines (following EU Cost Action White Paper recommendations) Fits Cerebral Blood Flow maps in the first instance.
Source code

Examples

```python
>>> from nipype.interfaces import niftyfit
>>> node = niftyfit.FitAsl()
>>> node.inputs.source_file = 'asl.nii.gz'
>>> node.cmdline
'fit_asl -source asl.nii.gz -cbf asl_cbf.nii.gz -error asl_error.nii.gz -syn asl_syn.nii.gz'
```

Inputs:

[Mandatory]
source_file: (a pathlike object or string representing an existing file)
   Filename of the 4D ASL (control/label) source image (mandatory).
   argument: `'-source %s'`, position: 1

[Optional]
pasl: (a boolean)
   Fit PASL ASL data [default]
   argument: `'-pasl'`
pcasl: (a boolean)
   Fit PCASL ASL data
   argument: `'-pcasl'`

(continues on next page)
cbf_file: (a pathlike object or string representing a file)
   Filename of the Cerebral Blood Flow map (in ml/100g/min).
   argument: ``-cbf %s``
error_file: (a pathlike object or string representing a file)
   Filename of the CBF error map.
   argument: ``-error %s``
syn_file: (a pathlike object or string representing a file)
   Filename of the synthetic ASL data.
   argument: ``-syn %s``
t1map: (a pathlike object or string representing an existing file)
   Filename of the estimated input T1 map (in ms).
   argument: ``-t1map %s``
m0map: (a pathlike object or string representing an existing file)
   Filename of the estimated input M0 map.
   argument: ``-m0map %s``
m0mape: (a pathlike object or string representing an existing file)
   Filename of the estimated input M0 map error.
   argument: ``-m0mape %s``
ir_volume: (a pathlike object or string representing an existing file)
   Filename of a [1,2,5]s Inversion Recovery volume (T1/M0 fitting carried out internally).
   argument: ``-IRvolume %s``
ir_output: (a pathlike object or string representing an existing file)
   Output of [1,2,5]s Inversion Recovery fitting.
   argument: ``-IRoutput %s``
mask: (a pathlike object or string representing an existing file)
   Filename of image mask.
   argument: ``-mask %s``, position: 2
t1_art_cmp: (a float)
   T1 of arterial component [1650ms].
   argument: ``-T1a %f``
plasma_coeff: (a float)
   Single plasma/tissue partition coefficient [0.9ml/g].
   argument: ``-L %f``
eff: (a float)
   Labelling efficiency [0.99 (pasl), 0.85 (pcasl)], ensure any background suppression pulses are included in -eff
   argument: ``-eff %f``
out: (a float)
   Outlier rejection for multi CL volumes (enter z-score threshold (e.g. 2.5)) [off].
   argument: ``-out %f``
pld: (a float)
   Post Labelling Delay [2000ms].
   argument: ``-PLD %f``
ldd: (a float)
   Labelling Duration [1800ms].
   argument: ``-LDD %f``
dpld: (a float)
   Difference in labelling delay per slice [0.0 ms/slice.
   argument: ``-dPLD %f``
t_inv1: (a float)
   Saturation pulse time [800ms].
   argument: ``-Tinv1 %f``
t_inv2: (a float)
Inversion time [2000ms].
argument: `-Tinv2 %f`

dt_inv2: (a float)
   Difference in inversion time per slice [0ms/slice].
   argument: `-dTinv2 %f`

gm_t1: (a float)
   T1 of GM [1150ms].
   argument: `-gmT1 %f`

gm_plasma: (a float)
   Plasma/GM water partition [0.95ml/g].
   argument: `-gmL %f`

gm_ttt: (a float)
   Time to GM [ATT+0ms].
   argument: `-gmTTT %f`

wm_t1: (a float)
   T1 of WM [800ms].
   argument: `-wmT1 %f`

wm_plasma: (a float)
   Plasma/WM water partition [0.82ml/g].
   argument: `-wmL %f`

wm_ttt: (a float)
   Time to WM [ATT+0ms].
   argument: `-wmTTT %f`

seg: (a pathlike object or string representing an existing file)
   Filename of the 4D segmentation (in ASL space) for L/T1 estimation
   and PV correction {WM,GM,CSF}.
   argument: `-seg %s`

sig: (a boolean)
   Use sigmoid to estimate L from T1: L(T1|gmL,wmL) [Off].
   argument: `-sig`

pv0: (an integer (int or long))
   Simple PV correction (CBF=vg*CBFg + vw*CBFw, with CBFw=f*CBFg)
   [0.25].
   argument: `-pv0 %d`

pv2: (an integer (int or long))
   In plane PV kernel size [3x3].
   argument: `-pv2 %d`

pv3: (a tuple of the form: (an integer (int or long), an integer (int
   or long), an integer (int or long)))
   3D kernel size [3x3x1].
   argument: `-pv3 %d %d %d`

mul: (a float)
   Multiply CBF by this value (e.g. if CL are mislabelled use -1.0).
   argument: `-mul %f`

mulgm: (a boolean)
   Multiply CBF by segmentation [Off].
   argument: `-sig`

pv_threshold: (a boolean)
   Set PV threshold for switching off LSQR [0.05].
   argument: `-pvthreshold`

segstyle: (a boolean)
   Set CBF as [gm,wm] not [wm,gm].
   argument: `-segstyle`

args: (a unicode string)
   Additional parameters to the command
   argument: `-%s`

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cbf_file</td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>error_file</td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>syn_file</td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
</tbody>
</table>

70.2 interfaces.niftyfit.base

70.2.1 NiftyFitCommand

Link to code
Base support interface for NiftyFit commands.
Inputs:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>args</td>
<td>(a unicode string)</td>
</tr>
</tbody>
</table>
| environ  | (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

Outputs:

None

70.3 interfaces.niftyfit.dwi

70.3.1 DwiTool

Link to code
Wraps the executable command dwi_tool.
Interface for executable dwi_tool from Niftyfit platform.
Use DwiTool.
Diffusion-Weighted MR Prediction. Predicts DWI from previously fitted models and calculates model derived maps.
Source code

Examples

```python
>>> from nipype.interfaces import niftyfit
>>> dwi_tool = niftyfit.DwiTool(dti_flag=True)
>>> dwi_tool.inputs.source_file = 'dwi.nii.gz'
>>> dwi_tool.inputs.bvec_file = 'bvecs'
```
>>> dwi_tool.inputs.bval_file = 'bvals'
>>> dwi_tool.inputs.mask_file = 'mask.nii.gz'
>>> dwi_tool.inputs.b0_file = 'b0.nii.gz'
>>> dwi_tool.inputs.rgbmap_file = 'rgb_map.nii.gz'
>>> dwi_tool.cmdline
'dwi_tool -source dwi.nii.gz -bval bvals -bvec bvecs -b0 b0.nii.gz -mask mask.nii. 
-nii.gz -mdmap dwi_mdmap.nii.gz -rgbmap rgb_map.nii.gz -syn dwi_syn.nii.gz - 
-vlmap dwi_vlmap.nii.gz'

Inputs:

[Optional]
bvec_file: (a pathlike object or string representing an existing file)
The file containing the bvectors of the source DWI.
argument: `''-bvec %s''`, position: 3

b0_file: (a pathlike object or string representing an existing file)
The B0 image corresponding to the source DWI
argument: `''-b0 %s''`, position: 4

mask_file: (a pathlike object or string representing an existing file)
The image mask
argument: `''-mask %s''`, position: 5

mcmap_file: (a pathlike object or string representing a file)
Filename of multi-compartment model parameter map (-ivim,-ball,-nod)
argument: `''-mcmap %s''`

syn_file: (a pathlike object or string representing a file)
Filename of synthetic image. Requires: bvec_file/b0_file.
argument: `''-syn %s''`
requires: bvec_file, b0_file

mdmap_file: (a pathlike object or string representing a file)
Filename of MD map/ADC
argument: `''-mdmap %s''`

famap_file: (a pathlike object or string representing a file)
Filename of FA map
argument: `''-famap %s''`

v1map_file: (a pathlike object or string representing a file)
Filename of PDD map [x,y,z]
argument: `''-v1map %s''`

rgbmap_file: (a pathlike object or string representing a file)
Filename of colour FA map.
argument: `''-rgbmap %s''`

logdti2_file: (a pathlike object or string representing a file)
Filename of output logdti map.
argument: `''-logdti2 %s''`

mono_flag: (a boolean)
Input is a single exponential to non-directional data [default with
no b-vectors]
argument: `--mono` , position: 6
mutually_exclusive: ivim_flag, dti_flag, dti_flag2, ball_flag,
bally_flag, nod_flag, nodv_flag
ivim_flag: (a boolean)
Input is an IVIM model to non-directional data.
argument: `--ivim` , position: 6
mutually_exclusive: mono_flag, dti_flag, dti_flag2, ball_flag,
bally_flag, nod_flag, nodv_flag
dti_flag: (a boolean)
Input is a tensor model diag/off-diag.
argument: `--dti` , position: 6
mutually_exclusive: mono_flag, ivim_flag, dti_flag2, ball_flag,
bally_flag, nod_flag, nodv_flag
dti_flag2: (a boolean)
Input is a tensor model lower triangular
argument: `--dti2` , position: 6
mutually_exclusive: mono_flag, ivim_flag, dti_flag, ball_flag,
bally_flag, nod_flag, nodv_flag
ball_flag: (a boolean)
Input is a ball and stick model.
argument: `--ball` , position: 6
mutually_exclusive: mono_flag, ivim_flag, dti_flag, dti_flag2,
bally_flag, nod_flag, nodv_flag
bally_flag: (a boolean)
Input is a ball and stick model with optimised PDD.
argument: `--ballv` , position: 6
mutually_exclusive: mono_flag, ivim_flag, dti_flag, dti_flag2,
bally_flag, nod_flag, nodv_flag
nod_flag: (a boolean)
Input is a NODDI model
argument: `--nod` , position: 6
mutually_exclusive: mono_flag, ivim_flag, dti_flag, dti_flag2,
bally_flag, bally_flag, nod_flag, nodv_flag
nodv_flag: (a boolean)
Input is a NODDI model with optimised PDD
argument: `--nodv` , position: 6
mutually_exclusive: mono_flag, ivim_flag, dti_flag, dti_flag2,
bally_flag, bally_flag, nod_flag
diso_val: (a float)
Isotropic diffusivity for -nod [3e-3]
argument: `--diso %f`
dpr_val: (a float)
Parallel diffusivity for -nod [1.7e-3].
argument: `--dpr %f`
args: (a unicode string)
Additional parameters to the command
argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

mcmap_file: (a pathlike object or string representing a file)
70.3.2 FitDwi

Link to code

Wraps the executable command fit_dwi.

Interface for executable fit_dwi from Niftyfit platform.

Use NiftyFit to perform diffusion model fitting.

Diffusion-weighted MR Fitting. Fits DWI parameter maps to multi-shell, multi-directional data.

Source code

Examples

```python
>>> from nipype.interfaces import niftyfit
>>> fit_dwi = niftyfit.FitDwi(dti_flag=True)
>>> fit_dwi.inputs.source_file = 'dwi.nii.gz'
>>> fit_dwi.inputs.bvec_file = 'bvecs'
>>> fit_dwi.inputs.bval_file = 'bvals'
>>> fit_dwi.inputs.rgbmap_file = 'rgb.nii.gz'
>>> fit_dwi.cmdline
```

Inputs:

- **source_file**: (a pathlike object or string representing an existing file)
  - The source image containing the dwi data.
  - Argument: `--source %s`, position: 1

- **bval_file**: (a pathlike object or string representing an existing file)
  - The file containing the bvalues of the source DWI.
  - Argument: `--bval %s`, position: 2

- **bvec_file**: (a pathlike object or string representing an existing file)
  - The file containing the bvectors of the source DWI.
  - Argument: `--bvec %s`, position: 3
Filename of TEs (ms).
  argument: ``'-TE %s'``
  mutually_exclusive: te_file

**te_value:** (a pathlike object or string representing an existing file)
  Value of TEs (ms).
  argument: ``'-TE %s'``
  mutually_exclusive: te_file

**mask_file:** (a pathlike object or string representing an existing file)
  The image mask
  argument: "'-mask %s'"

**prior_file:** (a pathlike object or string representing an existing file)
  Filename of parameter priors for -ball and -nod.
  argument: "'-prior %s'"

**rot_sform_flag:** (an integer (int or long))
  Rotate the output tensors according to the q/s form of the image
  (resulting tensors will be in mm coordinates, default: 0).
  argument: "'-rotsform %d'"

**error_file:** (a pathlike object or string representing a file)
  Filename of parameter error maps.
  argument: "'-error %s'"

**res_file:** (a pathlike object or string representing a file)
  Filename of model residual map.
  argument: "'-res %s'"

**syn_file:** (a pathlike object or string representing a file)
  Filename of synthetic image.
  argument: "'-syn %s'"

**nodiff_file:** (a pathlike object or string representing a file)
  Filename of average no diffusion image.
  argument: "'-nodiff %s'"

**mcmap_file:** (a pathlike object or string representing a file)
  Filename of multi-compartment model parameter map (-ivim,-ball,-nod)
  argument: "'-mcmap %s'"
  requires: nodv_flag

**mdmap_file:** (a pathlike object or string representing a file)
  Filename of MD map/ADC
  argument: "'-mdmap %s'"

**famap_file:** (a pathlike object or string representing a file)
  Filename of FA map
  argument: "'-famap %s'"

**v1map_file:** (a pathlike object or string representing a file)
  Filename of PDD map \([x,y,z]\)
  argument: "'-v1map %s'"

**rgbmap_file:** (a pathlike object or string representing a file)
  Filename of colour-coded FA map
  argument: "'-rgbmap %s'"
  requires: dti_flag

**ten_type:** ('lower-tri' or 'diag-off-diag', nipype default value: lower-tri)
  Use lower triangular (tenmap2) or diagonal, off-diagonal tensor
  format

**tenmap_file:** (a pathlike object or string representing a file)
  Filename of tensor map \([\text{diag,offdiag}]\)
  argument: "'-tenmap %s'"
  requires: dti_flag

**tenmap2_file:** (a pathlike object or string representing a file)
Filename of tensor map [lower tri]
argument: `'-tenmap2 %s`
requires: dti_flag

mono_flag: (a boolean)
Fit single exponential to non-directional data [default with no b-vectors]
argument: `'-mono'`, position: 4
mutually_exclusive: ivim_flag, dti_flag, ball_flag, ballv_flag, nod_flag, nodv_flag

ivim_flag: (a boolean)
Fit IVIM model to non-directional data.
argument: `'-ivim'`, position: 4
mutually_exclusive: mono_flag, dti_flag, ball_flag, ballv_flag, nod_flag, nodv_flag

dti_flag: (a boolean)
Fit the tensor model [default with b-vectors].
argument: `'-dti'`, position: 4
mutually_exclusive: mono_flag, ivim_flag, ball_flag, ballv_flag, nod_flag, nodv_flag

ball_flag: (a boolean)
Fit the ball and stick model.
argument: `'-ball'`, position: 4
mutually_exclusive: mono_flag, ivim_flag, dti_flag, ballv_flag, nod_flag, nodv_flag

ballv_flag: (a boolean)
Fit the ball and stick model with optimised PDD.
argument: `'-ballv'`, position: 4
mutually_exclusive: mono_flag, ivim_flag, dti_flag, ball_flag, nod_flag, nodv_flag

nod_flag: (a boolean)
Fit the NODDI model
argument: `'-nod'`, position: 4
mutually_exclusive: mono_flag, ivim_flag, dti_flag, ballv_flag, ball_flag, nodv_flag

nodv_flag: (a boolean)
Fit the NODDI model with optimised PDD
argument: `'-nodv'`, position: 4
mutually_exclusive: mono_flag, ivim_flag, dti_flag, ball_flag, ballv_flag, nod_flag

maxit_val: (an integer (int or long))
Maximum number of non-linear LSQR iterations [100x2 passes]
argument: `'-maxit %d'`
requires: gn_flag

lm_vals: (a tuple of the form: (a float, a float))
LM parameters (initial value, decrease rate) [100,1.2].
argument: `'-lm %f %f'`
requires: gn_flag

gn_flag: (a boolean)
Use Gauss-Newton algorithm [Levenberg-Marquardt].
argument: `'-gn'`
mutually_exclusive: wls_flag

vb_flag: (a boolean)
Use Variational Bayes fitting with known prior (currently identity covariance...).
argument: `'-vb'`
cov_file: (a pathlike object or string representing an existing file)
Filename of the nc x nc covariance matrix [1]
Argument: `--cov %s`
now: a boolean
Use Variational Bayes fitting with known prior (currently identity
covariance...).

Argument: `--wls`
Mutually exclusive: gn_flag

wls_flag: a boolean
Use Variational Bayes fitting with known prior (currently identity
covariance...).

Argument: `--wls`

swls_val: a float
Use location-weighted least squares for DTI fitting [3x3 Gaussian]

Argument: `--swls %f`

Slice_no: an integer (int or long)
Fit to single slice number.

Argument: `--slice %d`

Voxel: a tuple of the form: (an integer (int or long), an integer
(int or long), an integer (int or long))
Fit to single voxel only.

Argument: `--voxel %d %d %d`

diso_val: a float
Isotropic diffusivity for -nod [3e-3]

Argument: `--diso %f`

dpr_val: a float
Parallel diffusivity for -nod [1.7e-3].

Argument: `--dpr %f`

Wm_t2_val: a float
White matter T2 value [80ms].

Argument: `--wmT2 %f`

Csf_t2_val: a float
CSF T2 value [400ms].

Argument: `--csfT2 %f`

Perf_thr: a float
Threshold for perfusion/diffusion effects [100].

Argument: `--perfthreshold %f`

Mcout: a pathlike object or string representing a file
Filename of mc samples (ascii text file)

Argument: `--mcout %s`

Mcsamples: an integer (int or long)
Number of samples to keep [100].

Argument: `--mcsamples %d`

Mcmaxit: an integer (int or long)
Number of iterations to run [10,000].

Argument: `--mcmaxit %d`

Acceptance: a float
Fraction of iterations to accept [0.23].

Argument: `--accpetance %f`

Args: a unicode string
Additional parameters to the command

Argument: `--%s`

Environ: a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

erorr_file: a pathlike object or string representing a file
Filename of parameter error maps

res_file: a pathlike object or string representing a file
Filename of model residual map
syn_file: (a pathlike object or string representing a file)
Filename of synthetic image
	nodiff_file: (a pathlike object or string representing a file)
Filename of average no diffusion image.

mdmap_file: (a pathlike object or string representing a file)
Filename of MD map/ADC

famap_file: (a pathlike object or string representing a file)
Filename of FA map

vlmap_file: (a pathlike object or string representing a file)
Filename of PDD map [x,y,z]

rgbmap_file: (a pathlike object or string representing a file)
Filename of colour FA map

tenmap_file: (a pathlike object or string representing a file)
Filename of tensor map

tenmap2_file: (a pathlike object or string representing a file)
Filename of tensor map [lower tri]

mcmap_file: (a pathlike object or string representing a file)
Filename of multi-compartment model parameter map
(-ivim,-ball,-nod).

mcout: (a pathlike object or string representing a file)
Filename of mc samples (ascii text file)

---

70.4 interfaces.niftify.fit1

70.4.1 FitQt1

Link to code
Wraps the executable command fit_qt1.
Interface for executable fit_qt1 from Niftify platform.
Use Niftify to perform Qt1 fitting.
T1 Fitting Routine (To inversion recovery or spgr data). Fits single component T1 maps in the first instance.

Source code

Examples

```python
>>> from nipype.interfaces.niftifyfit import FitQt1
>>> fit_qt1 = FitQt1()
>>> fit_qt1.inputs.source_file = 'TI4D.nii.gz'

fit_qt1.cmdline

'fit_qt1 -source TI4D.nii.gz -comp TI4D_comp.nii.gz -error TI4D_error.nii.gz -' 'm0map TI4D_m0map.nii.gz -mcmap TI4D_mcmap.nii.gz -res TI4D_res.nii.gz -syn TI4D_ syn.nii.gz -t1map TI4D_t1map.nii.gz'

Inputs:

[Mandatory]
	source_file: (a pathlike object or string representing an existing
file)
Filename of the 4D Multi-Echo T1 source image.
argument: `'-source %s'`, position: 1

[Optional]

t1map_file: (a pathlike object or string representing a file)
Filename of the estimated output T1 map (in ms).

(continues on next page)
m0map_file: (a pathlike object or string representing a file)
Filename of the estimated input M0 map.

argument: `\'-m0map %s\'`

mcmap_file: (a pathlike object or string representing a file)
Filename of the estimated output multi-parameter map.

argument: `\'-mcmap %s\'`

comp_file: (a pathlike object or string representing a file)
Filename of the estimated multi-component T1 map.

argument: `\'-comp %s\'`

error_file: (a pathlike object or string representing a file)
Filename of the error map (symmetric matrix, [Diag,OffDiag]).

argument: `\'-error %s\'`

syn_file: (a pathlike object or string representing a file)
Filename of the synthetic ASL data.

argument: `\'-syn %s\'`

res_file: (a pathlike object or string representing a file)
Filename of the model fit residuals

argument: `\'-res %s\'`

mask: (a pathlike object or string representing an existing file)
Filename of image mask.

argument: `\'-mask %s\'`, position: 2

prior: (a pathlike object or string representing an existing file)
Filename of parameter prior.

argument: `\'-prior %s\'`, position: 3

t_te_value: (a float)
TE Echo Time [0ms!].

argument: `\'-TE %f\'`, position: 4

tr_value: (a float)
TR Repetition Time [10s!].

argument: `\'-TR %f\'`, position: 5

nb_comp: (an integer (int or long))
Number of components to fit [1] (currently IR/SR only)

argument: `\'-nc %d\'`, position: 6

lm_val: (a tuple of the form: (a float, a float))
Set LM parameters (initial value, decrease rate) [100,1.2].

argument: `\'-lm %f %f\'`, position: 7

gn_flag: (a boolean)
Use Gauss-Newton algorithm [Levenberg-Marquardt].

argument: `\'-gn\'`, position: 8

slice_no: (a integer (int or long))
Fit to single slice number.

argument: `\'-slice %d\'`, position: 9

voxel: (a tuple of the form: (an integer (int or long), an integer (int or long), an integer (int or long)))
Fit to single voxel only.

argument: `\'-voxel %d %d %d\'`, position: 10

maxit: (an integer (int or long))
NLSQR iterations [100].

argument: `\'-maxit %d\'`, position: 11

sr_flag: (a boolean)
Saturation Recovery fitting [default].

argument: `\'-SR\'`, position: 12

ir_flag: (a boolean)
Inversion Recovery fitting [default].

argument: `\'-IR\'`, position: 13

tis: (a list of items which are a float)
Inversion times for T1 data [1s, 2s, 5s].

argument: `--TIs %s`, position: 14

tis_list: (a pathlike object or string representing an existing file)
Filename of list of pre-defined TIs.
argument: `--TIslist %s`

t1_list: (a pathlike object or string representing an existing file)
Filename of list of pre-defined T1s
argument: `--T1list %s`

t1min: (a float)
Minimum tissue T1 value [400ms].
argument: `--T1min %f`

t1max: (a float)
Maximum tissue T1 value [4000ms].
argument: `--T1max %f`

spgr: (a boolean)
Spoiled Gradient Echo fitting
argument: `--SPGR`

flips: (a list of items which are a float)
Flip angles
argument: `--flips %s`

flips_list: (a pathlike object or string representing an existing file)
Filename of list of pre-defined flip angles (deg).
argument: `--flipslist %s`

b1map: (a pathlike object or string representing an existing file)
Filename of B1 estimate for fitting (or include in prior).
argument: `--b1map %s`

mcout: (a pathlike object or string representing an existing file)
Filename of mc samples (ascii text file)
argument: `--mcout %s`

mcsamples: (an integer (int or long))
Number of samples to keep [100].
argument: `--mcsamples %d`

mcmaxit: (an integer (int or long))
Number of iterations to run [10,000].
argument: `--mcmaxit %d`

acceptance: (a float)
Fraction of iterations to accept [0.23].
argument: `--acceptance %f`

args: (a unicode string)
Additional parameters to the command
argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

t1map_file: (a pathlike object or string representing a file)
Filename of the estimated output T1 map (in ms)

m0map_file: (a pathlike object or string representing a file)
Filename of the m0 map

mcmap_file: (a pathlike object or string representing a file)
Filename of the estimated output multi-parameter map

comp_file: (a pathlike object or string representing a file)
Filename of the estimated multi-component T1 map.
error_file: (a pathlike object or string representing a file)
    Filename of the error map (symmetric matrix, [Diag,OffDiag])
syn_file: (a pathlike object or string representing a file)
    Filename of the synthetic ASL data
res_file: (a pathlike object or string representing a file)
    Filename of the model fit residuals
71.1 interfaces.niftyreg.base

71.1.1 NiftyRegCommand

Link to code
Base support interface for NiftyReg commands.
Inputs:

<table>
<thead>
<tr>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>omp_core_val:</td>
<td>(an integer (int or long), nipype default value: 1) Number of openmp thread to use</td>
</tr>
<tr>
<td></td>
<td>argument: <code>-omp %i</code></td>
</tr>
<tr>
<td>args:</td>
<td>(a unicode string) Additional parameters to the command</td>
</tr>
<tr>
<td></td>
<td>argument: <code>%s</code></td>
</tr>
<tr>
<td>environ:</td>
<td>(a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})</td>
</tr>
</tbody>
</table>

Environment variables

Outputs:

None

71.1.2 get_custom_path()

Link to code

71.2 interfaces.niftyreg.reg

71.2.1 RegAladin

Link to code
Wraps the executable command reg_aladin.
Interface for executable reg_aladin from NiftyReg platform.

**Examples**

```python
>>> from nipype.interfaces import niftyreg
>>> node = niftyreg.RegAladin()
>>> node.inputs.ref_file = 'im1.nii'
>>> node.inputs.flo_file = 'im2.nii'
>>> node.inputs.rmask_file = 'mask.nii'
>>> node.inputs.omp_core_val = 4
>>> node.cmdline
'reg_aladin -aff im2_aff.txt -flo im2.nii -omp 4 -ref im1.nii -res im2_res.nii.gz -rmask mask.nii'
```

**Inputs:**

[Mandatory]
- **ref_file**: (a pathlike object or string representing an existing file)
  - The input reference/target image
  - argument: `-ref %s`
- **flo_file**: (a pathlike object or string representing an existing file)
  - The input floating/source image
  - argument: `-flo %s`

[Optional]
- **nosym_flag**: (a boolean)
  - Turn off symmetric registration
  - argument: `-noSym`
- **rig_only_flag**: (a boolean)
  - Do only a rigid registration
  - argument: `-rigOnly`
- **aff_direct_flag**: (a boolean)
  - Directly optimise the affine parameters
  - argument: `-affDirect`
- **in_aff_file**: (a pathlike object or string representing an existing file)
  - The input affine transformation
  - argument: `-inaff %s`
- **rmask_file**: (a pathlike object or string representing an existing file)
  - The input reference mask
  - argument: `-rmask %s`
- **fmask_file**: (a pathlike object or string representing an existing file)
  - The input floating mask
  - argument: `-fmask %s`
- **maxit_val**: (a long integer >= 0)
  - Maximum number of iterations
  - argument: `-maxit %d`
- **ln_val**: (a long integer >= 0)
  - Number of resolution levels to create
  - argument: `-ln %d`
- **lp_val**: (a long integer >= 0)
  - Number of resolution levels to perform
  - argument: `-lp %d`

(continues on next page)
smoo_r_val: (a float)
  Amount of smoothing to apply to reference image
  argument: ``-smooR %f``

smoo_f_val: (a float)
  Amount of smoothing to apply to floating image
  argument: ``-smooF %f``

nac_flag: (a boolean)
  Use nifti header to initialise transformation
  argument: ``-nac``

cog_flag: (a boolean)
  Use the masks centre of mass to initialise the transformation
  argument: ``-cog``

v_val: (a long integer >= 0)
  Percent of blocks that are active
  argument: ``-pv %d``

i_val: (a long integer >= 0)
  Percent of inlier blocks
  argument: ``-pi %d``

ref_low_val: (a float)
  Lower threshold value on reference image
  argument: ``-refLowThr %f``

ref_up_val: (a float)
  Upper threshold value on reference image
  argument: ``-refUpThr %f``

flo_low_val: (a float)
  Lower threshold value on floating image
  argument: ``-floLowThr %f``

flo_up_val: (a float)
  Upper threshold value on floating image
  argument: ``-floUpThr %f``

platform_val: (an integer (int or long))
  Platform index
  argument: ``-platf %i``

gpuid_val: (an integer (int or long))
  Device to use id
  argument: ``-gpuid %i``

verbosity_off_flag: (a boolean)
  Turn off verbose output
  argument: ``-voff``

aff_file: (a pathlike object or string representing a file)
  The output affine matrix file
  argument: ``-aff %s``

res_file: (a pathlike object or string representing a file)
  The affine transformed floating image
  argument: ``-res %s``

omp_core_val: (an integer (int or long), nipype default value: 1)
  Number of openmp thread to use
  argument: ``-omp %i``

args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a value
  of class 'str', nipype default value: {})
  Environment variables

Outputs:
### 71.2.2 RegF3D

**Link to code**

Wraps the executable command `reg_f3d`.

Interface for executable `reg_f3d` from NiftyReg platform.

Fast Free-Form Deformation (F3D) algorithm for non-rigid registration. Initially based on Modat et al., “Fast Free-Form Deformation using graphics processing units”, CMPB, 2010

**Source code**

**Examples**

```python
>>> from nipype.interfaces import niftyreg
>>> node = niftyreg.RegF3D()
>>> node.inputs.ref_file = 'im1.nii'
>>> node.inputs.flo_file = 'im2.nii'
>>> node.inputs.rmask_file = 'mask.nii'
>>> node.inputs.omp_core_val = 4
>>> node.cmdline
'reg_f3d -cpp im2_cpp.nii.gz -flo im2.nii -omp 4 -ref im1.nii -res im2_res.nii.gz -rmask mask.nii'
```

**Inputs:**

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mandatory</strong></td>
<td></td>
</tr>
<tr>
<td>ref_file</td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>flo_file</td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td><strong>Optional</strong></td>
<td></td>
</tr>
<tr>
<td>aff_file</td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>incpp_file</td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>rmask_file</td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>ref_smooth_val</td>
<td>(a float)</td>
</tr>
<tr>
<td>flo_smooth_val</td>
<td>(a float)</td>
</tr>
</tbody>
</table>
rlwth_thr_val: (a float)
    Lower threshold for reference image
    argument: `--rlWTh %f`
rupth_thr_val: (a float)
    Upper threshold for reference image
    argument: `--rUpTh %f`
flwth_thr_val: (a float)
    Lower threshold for floating image
    argument: `--fLwTh %f`
fupth_thr_val: (a float)
    Upper threshold for floating image
    argument: `--fUpTh %f`
rlwth2_thr_val: (a tuple of the form: (a long integer >= 0, a float))
    Lower threshold for reference image at the specified time point
    argument: `--rlWTh %d %f`
rupth2_thr_val: (a tuple of the form: (a long integer >= 0, a float))
    Upper threshold for reference image at the specified time point
    argument: `--rUpTh %d %f`
flwth2_thr_val: (a tuple of the form: (a long integer >= 0, a float))
    Lower threshold for floating image at the specified time point
    argument: `--fLwTh %d %f`
fupth2_thr_val: (a tuple of the form: (a long integer >= 0, a float))
    Upper threshold for floating image at the specified time point
    argument: `--fUpTh %d %f`
sx_val: (a float)
    Final grid spacing along the x axes
    argument: `--sx %f`
sy_val: (a float)
    Final grid spacing along the y axes
    argument: `--sy %f`
sz_val: (a float)
    Final grid spacing along the z axes
    argument: `--sz %f`
be_val: (a float)
    Bending energy value
    argument: `--be %f`
le_val: (a float)
    Linear elasticity penalty term
    argument: `--le %f`
jl_val: (a float)
    Log of jacobian of deformation penalty value
    argument: `--jl %f`
no_app_jl_flag: (a boolean)
    Do not approximate the log of jacobian penalty at control points
    only
    argument: `--noAppJL`
nmi_flag: (a boolean)
    use NMI even when other options are specified
    argument: `--nmi`
rbn_val: (a long integer >= 0)
    Number of bins in the histogram for reference image
    argument: `--rbn %d`
fbn_val: (a long integer >= 0)
    Number of bins in the histogram for reference image
    argument: `--fbn %d`
rbn2_val: (a tuple of the form: (a long integer >= 0, a long integer >= 0))
Number of bins in the histogram for reference image for given time point
argument: `--rbn %d %d`

\( fbn2\_val \): (a tuple of the form: (a long integer \(\geq 0\), a long integer
\(\geq 0\))

Number of bins in the histogram for reference image for given time point
argument: `--fbn %d %d`

\( lncc\_val \): (a float)
SD of the Gaussian for computing LNCC
argument: `--\_lncc %f`

\( lncc2\_val \): (a tuple of the form: (a long integer \(\geq 0\), a float))
SD of the Gaussian for computing LNCC for a given time point
argument: `--\_lncc %d %f`

\( ssd\_flag \): (a boolean)
Use SSD as the similarity measure
argument: `--ssd`

\( ssd2\_flag \): (a long integer \(\geq 0\))
Use SSD as the similarity measure for a given time point
argument: `--ssd %d`

\( kld\_flag \): (a boolean)
Use KL divergence as the similarity measure
argument: `--kld`

\( kld2\_flag \): (a long integer \(\geq 0\))
Use KL divergence as the similarity measure for a given time point
argument: `--kld %d`

\( amc\_flag \): (a boolean)
Use additive NMI
argument: `--amc`

\( nox\_flag \): (a boolean)
Don't optimise in x direction
argument: `--nox`

\( noy\_flag \): (a boolean)
Don't optimise in y direction
argument: `--noy`

\( noz\_flag \): (a boolean)
Don't optimise in z direction
argument: `--noz`

\( maxit\_val \): (a long integer \(\geq 0\))
Maximum number of iterations per level
argument: `--maxit %d`

\( ln\_val \): (a long integer \(\geq 0\))
Number of resolution levels to create
argument: `--ln %d`

\( lp\_val \): (a long integer \(\geq 0\))
Number of resolution levels to perform
argument: `--lp %d`

\( nopy\_flag \): (a boolean)
Do not use the multiresolution approach
argument: `--nopy`

\( noconj\_flag \): (a boolean)
Use simple GD optimization
argument: `--noConj`

\( pert\_val \): (a long integer \(\geq 0\))
Add perturbation steps after each optimization step
argument: `--pert %d`
Use velocity field integration
argument: `'-vel'`

fmask_file: (a pathlike object or string representing an existing file)
Floating image mask
argument: `'-fmask %s'`

smooth_grad_val: (a float)
Kernel width for smoothing the metric gradient
argument: `'-smoothGrad %f'`

pad_val: (a float)
Padding value
argument: `'-pad %f'`

verbosity_off_flag: (a boolean)
Turn off verbose output
argument: `'-voff'`

cpp_file: (a pathlike object or string representing a file)
The output CPP file
argument: `'-cpp %s'`

res_file: (a pathlike object or string representing a file)
The output resampled image
argument: `'-res %s'`

omp_core_val: (an integer (int or long), nipype default value: 1)
Number of openmp thread to use
argument: `'-omp %i'`

args: (a unicode string)
Additional parameters to the command
argument: `'%s'`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

cpp_file: (a pathlike object or string representing a file)
The output CPP file
res_file: (a pathlike object or string representing a file)
The output resampled image
invcpp_file: (a pathlike object or string representing a file)
The output inverse CPP file
invres_file: (a pathlike object or string representing a file)
The output inverse res file
avg_output: (a string)
Output string in the format for reg_average

71.3 interfaces.niftyreg.regutils

71.3.1 RegAverage

Link to code
Wraps the executable command reg_average.
Interface for executable reg_average from NiftyReg platform.
Compute average matrix or image from a list of matrices or image. The tool can be use to resample images given input transformation parametrisation as well as to demean transformations in Euclidean or log-Euclidean space.
This interface is different than the others in the way that the options will be written in a command file that is
given as a parameter.

Source code

Examples

```python
>>> from nipype.interfaces import niftyreg
>>> node = niftyreg.RegAverage()
>>> one_file = 'im1.nii'
>>> two_file = 'im2.nii'
>>> three_file = 'im3.nii'
>>> node.inputs.avg_files = [one_file, two_file, three_file]
>>> node.cmdline
# doctest: +ELLIPSIS
'reg_average --cmd_file .../reg_average_cmd'
```

Inputs:

[Optional]

avg_files: (a list of items which are a pathlike object or string representing a file)

  Averaging of images/affine transformations
  argument: `--avg %s`, position: 1
  mutually_exclusive: avg_lts_files, avg_ref_file, demean1_ref_file, demean2_ref_file, demean3_ref_file, warp_files

avg_lts_files: (a list of items which are a pathlike object or string representing a file)

  Robust average of affine transformations
  argument: `--avg_lts %s`, position: 1
  mutually_exclusive: avg_files, avg_ref_file, demean1_ref_file, demean2_ref_file, demean3_ref_file, warp_files

avg_ref_file: (a pathlike object or string representing a file)

  All input images are resampled into the space of <reference image> and averaged. A cubic spline interpolation scheme is used for resampling
  argument: `--avg_tran %s`, position: 1
  mutually_exclusive: avg_files, avg_lts_files, demean1_ref_file, demean2_ref_file, demean3_ref_file
  requires: warp_files

demean1_ref_file: (a pathlike object or string representing a file)

  Average images and demean average image that have affine transformations to a common space
  argument: `--demean1 %s`, position: 1
  mutually_exclusive: avg_files, avg_lts_files, avg_ref_file, demean2_ref_file, demean3_ref_file
  requires: warp_files

demean2_ref_file: (a pathlike object or string representing a file)

  Average images and demean average image that have non-rigid transformations to a common space
  argument: `--demean2 %s`, position: 1
  mutually_exclusive: avg_files, avg_lts_files, avg_ref_file, demean1_ref_file, demean3_ref_file
  requires: warp_files

demean3_ref_file: (a pathlike object or string representing a file)

  Average images and demean average image that have linear and non-rigid transformations to a common space
  argument: `--demean3 %s`, position: 1
  mutually_exclusive: avg_files, avg_lts_files, avg_ref_file, demean1_ref_file, demean2_ref_file
  requires: warp_files

(continues on next page)
warp_files: (a list of items which are a pathlike object or string representing a file) transformation files and floating image pairs/triplets to the reference space argument: ``%s``, position: -1 mutually_exclusive: avg_files, avg_lts_files

out_file: (a pathlike object or string representing a file) Output file name argument: ``%s``, position: 0

omp_core_val: (an integer (int or long), nipype default value: 1) Number of openmp thread to use argument: ``-omp %i``

args: (a unicode string) Additional parameters to the command argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

out_file: (a pathlike object or string representing a file) Output file name

71.3.2 RegJacobian

Link to code
Wraps the executable command reg_jacobian. Interface for executable reg_resample from NiftyReg platform. Tool to generate Jacobian determinant maps from transformation parametrisation generated by reg_f3d

Source code

Examples

```python
>>> from nipype.interfaces import niftyreg
>>> node = niftyreg.RegJacobian()
>>> node.inputs.ref_file = 'im1.nii'
>>> node.inputs.trans_file = 'warpfield.nii'
>>> node.inputs.omp_core_val = 4
>>> node.cmdline
'reg_jacobian -omp 4 -ref im1.nii -trans warpfield.nii -jac warpfield_jac.nii.gz'
```

Inputs:

[Mandatory]
trans_file: (a pathlike object or string representing an existing file) The input non-rigid transformation argument: ``-trans %s``

[Optional]
ref_file: (a pathlike object or string representing an existing file) Reference/target file (required if specifying CPP transformations. argument: ``-ref %s``
type: ('jac' or 'jacL' or 'jacM', nipype default value: jac)
71.3.3 RegMeasure

Link to code
Wraps the executable command reg_measure.
Interface for executable reg_measure from NiftyReg platform.
Given two input images, compute the specified measure(s) of similarity

Source code

Examples

```python
>>> from nipype.interfaces import niftyreg
>>> node = niftyreg.RegMeasure()
>>> node.inputs.ref_file = 'im1.nii'
>>> node.inputs.flo_file = 'im2.nii'
>>> node.inputs.measure_type = 'lncc'
>>> node.inputs.omp_core_val = 4
>>> node.cmdline
'reg_measure -flo im2.nii -lncc -omp 4 -out im2_lncc.txt -ref im1.nii'
```

Inputs:

[Mandatory]
ref_file: (a pathlike object or string representing an existing file)
The input reference/target image
argument: ``-ref %s``
flo_file: (a pathlike object or string representing an existing file)
The input floating/source image
argument: ``-flo %s``
measure_type: ('ncc' or 'lncc' or 'nmi' or 'ssd')
Measure of similarity to compute
argument: ``-%s``

[Optional]
out_file: (a pathlike object or string representing a file)
The output text file containing the measure
71.3.4 RegResample

Link to code
Wraps the executable command reg_resample. Interface for executable reg_resample from NiftyReg platform. Tool to resample floating image in the space of a defined reference image given a transformation parametrisation generated by reg_aladin, reg_f3d or reg_transform.

Source code

Examples

```python
>>> from nipype.interfaces import niftyreg
>>> node = niftyreg.RegResample()
>>> node.inputs.ref_file = 'im1.nii'
>>> node.inputs.flo_file = 'im2.nii'
>>> node.inputs.trans_file = 'warpfield.nii'
>>> node.inputs.inter_val = 'LIN'
>>> node.inputs.omp_core_val = 4
>>> node.cmdline
'reg_resample -flo im2.nii -inter 1 -omp 4 -ref im1.nii -trans warpfield.nii -res \n->im2_res.nii.gz'
```

Inputs:

<table>
<thead>
<tr>
<th>Mandatory</th>
</tr>
</thead>
<tbody>
<tr>
<td>ref_file: (a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>The input reference/target image</td>
</tr>
<tr>
<td>argument: <code>-ref %s</code></td>
</tr>
</tbody>
</table>

| flo_file: (a pathlike object or string representing an existing file) |
| The input floating/source image |
| argument: `-flo %s` |

<table>
<thead>
<tr>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>trans_file: (a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>The input transformation file</td>
</tr>
<tr>
<td>argument: <code>-trans %s</code></td>
</tr>
</tbody>
</table>

| type: ('res' or 'blank', nipype default value: res) |
| Type of output |

Outputs:

| out_file: (a pathlike object or string representing a file) |
| The output text file containing the measure |
argument: `--%s`, position: -2
out_file: (a pathlike object or string representing a file)
    The output filename of the transformed image
argument: `-%s`, position: -1
inter_val: ("NN" or 'LIN' or 'CUB' or 'SINC')
    Interpolation type
argument: `--inter %d`
pad_val: (a float)
    Padding value
argument: `--pad %f`
tensor_flag: (a boolean)
    Resample Tensor Map
argument: `--tensor`
verbosity_off_flag: (a boolean)
    Turn off verbose output
argument: `--voff`
psf_flag: (a boolean)
    Perform the resampling in two steps to resample an image to a lower resolution
argument: `--psf`
psf_alg: (0 or 1)
    Minimise the matrix metric (0) or the determinant (1) when estimating the PSF [0]
argument: `--psf_alg %d`
omp_core_val: (an integer (int or long), nipype default value: 1)
    Number of openmp thread to use
argument: `--omp %i`
args: (a unicode string)
    Additional parameters to the command
argument: `--%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
    Environment variables

Outputs:

out_file: (a pathlike object or string representing a file)
    The output filename of the transformed image

71.3.5 RegTools

Link to code
Wraps the executable command reg_tools.
Interface for executable reg_tools from NiftyReg platform.
Tool delivering various actions related to registration such as resampling the input image to a chosen resolution or remove the nan and inf in the input image by a specified value.
Source code

Examples

>>> from nipype.interfaces import niftyreg
>>> node = niftyreg.RegTools()
>>> node.inputs.in_file = 'im1.nii'
>>> node.inputs.mul_val = 4
>>> node.inputs.omp_core_val = 4
>>> node.cmdline
'reg_tools -in im1.nii -mul 4.0 -omp 4 -out im1_tools.nii.gz'

Inputs:

<table>
<thead>
<tr>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_file:</td>
<td>The input image file path</td>
</tr>
<tr>
<td>out_file:</td>
<td>The output file name</td>
</tr>
<tr>
<td>iso_flag:</td>
<td>Make output image isotropic</td>
</tr>
<tr>
<td>noscl_flag:</td>
<td>Set scale, slope to 0 and 1</td>
</tr>
<tr>
<td>mask_file:</td>
<td>Values outside the mask are set to NaN</td>
</tr>
<tr>
<td>thr_val:</td>
<td>Binarise the input image with the given threshold</td>
</tr>
<tr>
<td>bin_flag:</td>
<td>Binarise the input image</td>
</tr>
<tr>
<td>rms_val:</td>
<td>Compute the mean RMS between the images</td>
</tr>
<tr>
<td>div_val:</td>
<td>Divide the input by image or value</td>
</tr>
<tr>
<td>mul_val:</td>
<td>Multiply the input by image or value</td>
</tr>
<tr>
<td>add_val:</td>
<td>Add to the input image or value</td>
</tr>
<tr>
<td>sub_val:</td>
<td>Add to the input image or value</td>
</tr>
<tr>
<td>down_flag:</td>
<td>Downsample the image by a factor of 2</td>
</tr>
<tr>
<td>smo_s_val:</td>
<td>Smooth the input image using a cubic spline kernel</td>
</tr>
<tr>
<td>chg_res_val:</td>
<td>Modify the input image resolution</td>
</tr>
</tbody>
</table>

(continues on next page)
Change the resolution of the input image
argument: `--chgres %f %f %f`

smo_g_val: (a tuple of the form: (a float, a float, a float))
Smooth the input image using a Gaussian kernel
argument: `--smoG %f %f %f`

inter_val: ('NN' or 'LIN' or 'CUB' or 'SINC')
Interpolation order to use to warp the floating image
argument: `--interp %d`

omp_core_val: (an integer (int or long), nipype default value: 1)
Number of openmp thread to use
argument: `--omp %i`

args: (a unicode string)
Additional parameters to the command
argument: `--%s`

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
The output file

71.3.6 RegTransform

Link to code
Wraps the executable command reg_transform.
Interface for executable reg_transform from NiftyReg platform.
Tools to convert transformation parametrization from one type to another as well as to compose, inverse or half
transformations.
Source code

Examples

```python
>>> from nipype.interfaces import niftyreg
>>> node = niftyreg.RegTransform()
>>> node.inputs.def_input = 'warpfield.nii'
>>> node.inputs.omp_core_val = 4
>>> node.cmdline
'doctest: +ELLIPSIS
reg_transform -omp 4 -def warpfield.nii ...
.../warpfield_trans.nii.gz'
```

Inputs:

[Optional]
ref1_file: (a pathlike object or string representing an existing file)
The input reference/target image
argument: `--ref %s`, position: 0

ref2_file: (a pathlike object or string representing an existing file)
The input second reference/target image
argument: `--ref2 %s`, position: 1
requires: ref1_file

def_input: (a pathlike object or string representing an existing file)
Compute deformation field from transformation argument: `'-def %s'`, position: -2
mutually_exclusive: disp_input, flow_input, comp_input,
 upd_s_form_input, inv_aff_input, inv_nrr_input, half_input,
 make_aff_input, aff_2_rig_input, flirt_2_nr_input
disp_input: (a pathlike object or string representing an existing file)
Compute displacement field from transformation argument: `'-disp %s'`, position: -2
mutually_exclusive: def_input, flow_input, comp_input,
 upd_s_form_input, inv_aff_input, inv_nrr_input, half_input,
 make_aff_input, aff_2_rig_input, flirt_2_nr_input
flow_input: (a pathlike object or string representing an existing file)
Compute flow field from spline SVF argument: `'-flow %s'`, position: -2
mutually_exclusive: def_input, disp_input, comp_input,
 upd_s_form_input, inv_aff_input, inv_nrr_input, half_input,
 make_aff_input, aff_2_rig_input, flirt_2_nr_input
comp_input: (a pathlike object or string representing an existing file)
compose two transformations argument: `'-comp %s'`, position: -3
mutually_exclusive: def_input, disp_input, flow_input,
 upd_s_form_input, inv_aff_input, inv_nrr_input, half_input,
 make_aff_input, aff_2_rig_input, flirt_2_nr_input
requires: comp_input2
comp_input2: (a pathlike object or string representing an existing file)
compose two transformations argument: `'-s'`, position: -2
upd_s_form_input: (a pathlike object or string representing an existing file)
Update s-form using the affine transformation argument: `'-updSform %s'`, position: -3
mutually_exclusive: def_input, disp_input, flow_input, comp_input,
 inv_aff_input, inv_nrr_input, half_input, make_aff_input,
 aff_2_rig_input, flirt_2_nr_input
requires: upd_s_form_input2
upd_s_form_input2: (a pathlike object or string representing an existing file)
Update s-form using the affine transformation argument: `'-s'`, position: -2
requires: upd_s_form_input
inv_aff_input: (a pathlike object or string representing an existing file)
Invert an affine transformation argument: `'-invAff %s'`, position: -2
mutually_exclusive: def_input, disp_input, flow_input, comp_input,
 upd_s_form_input, inv_nrr_input, half_input, make_aff_input,
 aff_2_rig_input, flirt_2_nr_input
inv_nrr_input: (a tuple of the form: (a pathlike object or string representing an existing file, a pathlike object or string representing an existing file))
Invert a non-linear transformation argument: `'-invNrr %s %s'`, position: -2
mutually_exclusive: def_input, disp_input, flow_input, comp_input,
upd_s_form_input, inv_aff_input, half_input, make_aff_input,
aff_2_rig_input, flirt_2_nr_input

half_input: (a pathlike object or string representing an existing file)

Half way to the input transformation
argument: `''-half %s''`, position: -2
mutually_exclusive: def_input, disp_input, flow_input, comp_input,
upd_s_form_input, inv_aff_input, inv_nrr_input, make_aff_input,
aff_2_rig_input, flirt_2_nr_input

make_aff_input: (a tuple of the form: (a float, a float, a float, a float, a float, a float, a float, a float, a float, a float, a float, a float))

Make an affine transformation matrix
argument: `''-makeAff %f %f %f %f %f %f %f %f %f %f %f %f''`,
position: -2
mutually_exclusive: def_input, disp_input, flow_input, comp_input,
upd_s_form_input, inv_aff_input, inv_nrr_input, half_input,
aff_2_rig_input, flirt_2_nr_input

aff_2_rig_input: (a pathlike object or string representing an existing file)

Extract the rigid component from affine transformation
argument: `''-aff2rig %s''`, position: -2
mutually_exclusive: def_input, disp_input, flow_input, comp_input,
upd_s_form_input, inv_aff_input, inv_nrr_input, half_input,
make_aff_input, flirt_2_nr_input

flirt_2_nr_input: (a tuple of the form: (a pathlike object or string representing an existing file, a pathlike object or string representing an existing file, a pathlike object or string representing an existing file))

Convert a FLIRT affine transformation to niftyreg affine transformation
argument: `''-flirtAff2NR %s %s %s''`, position: -2
mutually_exclusive: def_input, disp_input, flow_input, comp_input,
upd_s_form_input, inv_aff_input, inv_nrr_input, half_input,
make_aff_input, aff_2_rig_input

out_file: (a pathlike object or string representing a file)

transformation file to write
argument: `''%s''`, position: -1

omp_core_val: (an integer (int or long), nipype default value: 1)

Number of openmp thread to use
argument: `''-omp %i''`

args: (a unicode string)

Additional parameters to the command
argument: `''%s''`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
72.1 interfaces.niftyseg.base

72.1.1 NiftySegCommand

Link to code
Base support interface for NiftySeg commands.
Inputs:

```
[Optional]
args: (a unicode string)
  Additional parameters to the command
  argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
Environment variables
```

Outputs:

None

72.2 interfaces.niftyseg.em

72.2.1 EM

Link to code
Wraps the executable command seg_EM.
Interface for executable seg_EM from NiftySeg platform.
seg_EM is a general purpose intensity based image segmentation tool. In it’s simplest form, it takes in one 2D or 3D image and segments it in n classes.
Source code | Documentation
Examples

```python
>>> from nipype.interfaces import niftyseg
>>> node = niftyseg.EM()
>>> node.inputs.in_file = 'im1.nii'
>>> node.inputs.no_prior = 4
>>> node.cmdline
'seg_EM -in im1.nii -bc_order 3 -bc_thresh 0 -max_iter 100 -min_iter 0 -nopriors -'
->4 -bc_out im1_bc_em.nii.gz -out im1_em.nii.gz -out_outlier im1_outlier_em.nii.gz -'
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  Input image to segment
  argument: ```-in %s```, position: 4
no_prior: (an integer (int or long))
  Number of classes to use without prior
  argument: ```-nopriors %s```
  mutually_exclusive: prior_4D, priors
prior_4D: (a pathlike object or string representing an existing file)
  4D file containing the priors
  argument: ```-prior4D %s```
  mutually_exclusive: no_prior, priors
priors: (a list of items which are any value)
  list of priors filepaths.
  argument: ```%s```
  mutually_exclusive: no_prior, prior_4D

[Optional]
mask_file: (a pathlike object or string representing an existing file)
  Filename of the ROI for label fusion
  argument: ```-mask %s```
max_iter: (an integer (int or long), nipype default value: 100)
  Maximum number of iterations
  argument: ```-max_iter %s```
min_iter: (an integer (int or long), nipype default value: 0)
  Minimum number of iterations
  argument: ```-min_iter %s```
bc_order_val: (an integer (int or long), nipype default value: 3)
  Polynomial order for the bias field
  argument: ```-bc_order %s```
mrf_beta_val: (a float)
  Weight of the Markov Random Field
  argument: ```-mrf_beta %s```
bc_thresh_val: (a float, nipype default value: 0)
  Bias field correction will run only if the ratio of improvement is
  below bc_thresh. (default=0 [OFF])
  argument: ```-bc_thresh %s```
reg_val: (a float)
  Amount of regularization over the diagonal of the covariance matrix
  [above 1]
  argument: ```-reg %s```
outlier_val: (a tuple of the form: (a float, a float))
  Outlier detection as in (Van Leemput TMI 2003). <fl1> is the
  Mahalanobis threshold [recommended between 3 and 7] <fl2> is a
convergence ratio below which the outlier detection is going to be done [recommended 0.01]
```
argument: `-%s`
```
relax_priors: (a tuple of the form: (a float, a float))
Relax Priors [relaxation factor: 0<rf<1 (recommended=0.5), gaussian regularization: gstd>0 (recommended=2.0)] /only 3D/
```
argument: `-%s`
```
out_file: (a pathlike object or string representing a file)
Output segmentation
```
argument: `-%s`
```
out_bc_file: (a pathlike object or string representing a file)
Output bias corrected image
```
argument: `-%s`
```
out_outlier_file: (a pathlike object or string representing a file)
Output outlierness image
```
argument: `-%s`
```
args: (a unicode string)
Additional parameters to the command
```
argument: `-%s`
```
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:
```
out_file: (a pathlike object or string representing a file) Output segmentation
out_bc_file: (a pathlike object or string representing a file) Output bias corrected image
out_outlier_file: (a pathlike object or string representing a file) Output outlierness image
```

72.3 interfaces.niftyseg.label_fusion

72.3.1 CalcTopNCC

Link to code
Wraps the executable command `seg_CalcTopNCC`.
Interface for executable `seg_CalcTopNCC` from NiftySeg platform.

Examples
```
>>> from nipype.interfaces import niftyseg
>>> node = niftyseg.CalcTopNCC()
>>> node.inputs.in_file = 'im1.nii'
>>> node.inputs.num_templates = 2
>>> node.inputs.in_templates = ['im2.nii', 'im3.nii']
>>> node.inputs.top_templates = 1
>>> node.cmdline
'seg_CalcTopNCC -target im1.nii -templates 2 im2.nii im3.nii -n 1'
```

Inputs:
72.3.2 LabelFusion

Link to code
Wraps the executable command seg_LabFusion.

Examples

```python
>>> from nipype.interfaces import niftyseg
>>> node = niftyseg.LabelFusion()
>>> node.inputs.in_file = 'im1.nii'
>>> node.inputs.kernel_size = 2.0
>>> node.inputs.file_to_seg = 'im2.nii'
```
>>> node.inputs.template_file = 'im3.nii'
>>> node.inputs.template_num = 2
>>> node.inputs.classifier_type = 'STEPS'
>>> node.cmdline
'seg_LabFusion -in im1.nii -STEPS 2.000000 2 im2.nii im3.nii -out im1_steps.nii'

**Inputs:**

[Mandatory]

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_file</td>
<td>(a pathlike object or string representing an existing file)</td>
<td>Filename of the 4D integer label image.</td>
</tr>
<tr>
<td>file_to_seg</td>
<td>(a pathlike object or string representing an existing file)</td>
<td>Original image to segment (3D Image)</td>
</tr>
<tr>
<td>classifier_type</td>
<td>('STEPS' or 'STAPLE' or 'MV' or 'SBA')</td>
<td>Type of Classifier Fusion.</td>
</tr>
</tbody>
</table>

[Optional]

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>template_file</td>
<td>(a pathlike object or string representing an existing file)</td>
<td>Registered templates (4D Image)</td>
</tr>
<tr>
<td>mask_file</td>
<td>(a pathlike object or string representing an existing file)</td>
<td>Filename of the ROI for label fusion</td>
</tr>
<tr>
<td>out_file</td>
<td>(a pathlike object or string representing a file)</td>
<td>Output consensus segmentation</td>
</tr>
<tr>
<td>prob_flag</td>
<td>(a boolean)</td>
<td>Probabilistic/Fuzzy segmented image</td>
</tr>
<tr>
<td>verbose</td>
<td>('0' or '1' or '2')</td>
<td>Verbose level [0 = off, 1 = on, 2 = debug] (default = 0)</td>
</tr>
<tr>
<td>unc</td>
<td>(a boolean)</td>
<td>Only consider non-consensus voxels to calculate statistics</td>
</tr>
<tr>
<td>kernel_size</td>
<td>(a float)</td>
<td>Gaussian kernel size in mm to compute the local similarity</td>
</tr>
<tr>
<td>template_num</td>
<td>(an integer (int or long))</td>
<td>Number of labels to use</td>
</tr>
<tr>
<td>sm_ranking</td>
<td>('ALL' or 'GNCC' or 'ROINCC' or 'LNCC', nipype default value: ALL)</td>
<td>Ranking for STAPLE and MV</td>
</tr>
<tr>
<td>dilation_roi</td>
<td>(an integer (int or long))</td>
<td>Dilation of the ROI ( &lt;int&gt; d&gt;=1 )</td>
</tr>
<tr>
<td>proportion</td>
<td>(a float)</td>
<td>Proportion of the label (only for single labels).</td>
</tr>
<tr>
<td>prob_update_flag</td>
<td>(a boolean)</td>
<td>Update label proportions at each iteration</td>
</tr>
<tr>
<td>set_pq</td>
<td>(a tuple of the form: (a float, a float))</td>
<td>Value of P and Q [ 0 &lt; (P,Q) &lt; 1 ] (default = 0.99 0.99)</td>
</tr>
</tbody>
</table>

(continues on next page)
argument: `\'-setPQ %f %f'`
mrf_value: (a float)
    MRF prior strength (between 0 and 5)
argument: `\'-MRF_beta %f'`
max_iter: (an integer (int or long))
    Maximum number of iterations (default = 15).
argument: `\'-max_iter %d'`
unc_thres: (a float)
    If <float> percent of labels agree, then area is not uncertain.
argument: `\'-uncthres %f'`
conv: (a float)
    Ratio for convergence (default epsilon = 10^-5).
argument: `\'-conv %f'`
args: (a unicode string)
    Additional parameters to the command
argument: `\'-%s'`
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
    image written after calculations

72.4 interfaces.niftyseg.lesions

72.4.1 FillLesions

Link to code
Wraps the executable command seg_FillLesions.
Interface for executable seg_FillLesions from NiftySeg platform.
Fill all the masked lesions with WM intensity average.
Source code | Documentation

Examples

```python
>>> from nipype.interfaces import niftyseg
>>> node = niftyseg.FillLesions()
>>> node.inputs.in_file = 'im1.nii'
>>> node.inputs.lesion_mask = 'im2.nii'
>>> node.cmdline
'seg_FillLesions -i im1.nii -l im2.nii -o im1_lesions_filled.nii.gz'
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
    Input image to fill lesions
argument: `\'-i %s'`, position: 1
lesion_mask: (a pathlike object or string representing an existing
    file)
    Lesion mask
argument: `\'-l %s'`, position: 2
[Optional]

out_file: (a pathlike object or string representing a file)
The output filename of the fill lesions results
argument: `''-o %s''`, position: 3

in_dilation: (an integer (int or long))
Dilate the mask <int> times (in voxels, by default 0)
argument: `''-dil %d''`

match: (a float)
Percentage of minimum number of voxels between patches <float> (by
default 0.5).
argument: `''-match %f''`

search: (a float)
Minimum percentage of valid voxels in target patch <float> (by
default 0).
argument: `''-search %f''`

smooth: (a float)
Smoothing by <float> (in minimal 6-neighbourhood voxels (by default
0.1)).
argument: `''-smo %f''`

size: (an integer (int or long))
Search regions size respect biggest patch size (by default 4).
argument: `''-size %d''`

cwf: (a float)
Patch cardinality weighting factor (by default 2).
argument: `''-cwf %f''`

bin_mask: (a pathlike object or string representing a file)
Give a binary mask with the valid search areas.
argument: `''-mask %s''`

other: (a boolean)
Guizard et al. (FIN 2015) method, it doesn't include the
multiresolution/hierarchical inpainting part, this part needs to be
done with some external software such as reg_tools and reg_resample
from NiftyReg. By default it uses the method presented in Prados et
al. (Neuroimage 2016).
argument: `''-other''`

use_2d: (a boolean)
Uses 2D patches in the Z axis, by default 3D.
argument: `''-2D''`

dbagt: (a boolean)
Save all intermediate files (by default OFF).
argument: `''-debug''`

out_datatype: (a string)
Set output <datatype> (char, short, int, uchar, ushort, uint, float,
double).
argument: `''-odt %s''`

verbose: (a boolean)
Verbose (by default OFF).
argument: `''-v''`

args: (a unicode string)
Additional parameters to the command
argument: `''%s''`

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:
out_file: (a pathlike object or string representing a file)

72.5 interfaces.niftyseg.maths

72.5.1 BinaryMaths

Link to code
Wraps the executable command seg_maths.
Interface for executable seg_maths from NiftySeg platform.
Interface to use any binary mathematical operations that can be performed
with the seg_maths command-line program.
See below for those operations:
mul - <float/file> - Multiply image <float> value or by other image.
div - <float/file> - Divide image by <float> or by other image.
add - <float/file> - Add image by <float> or by other image.
sub - <float/file> - Subtract image by <float> or by other image.
pow - <float> - Image to the power of <float>,
thr - <float> - Threshold the image below <float>,
uthr - <float> - Threshold image above <float>,
smo - <float> - Gaussian smoothing by std <float> (in voxels and up to 4-D),
edge - <float> - Calculate the edges of the image using a threshold <float>,
sobel3 - <float> - Calculate the edges of all timepoints using a Sobel filter with a 3x3x3 kernel and applying
<float> gaussian smoothing.
sobel5 - <float> - Calculate the edges of all timepoints using a Sobel filter with a 5x5x5 kernel and applying
<float> gaussian smoothing.
min - <file> - Get the min per voxel between <current> and <file>,
smol - <float> - Gaussian smoothing of a 3D label image.
geo - <float/file> - Geodesic distance according to the speed function <float/file>
llsnorm <file_norm> - Linear LS normalisation between current and <file_norm>
masknan <file_norm> - Assign everything outside the mask (mask==0) with NaNs
hdr_copy <file> - Copy header from working image to <file> and save in <output>.
splitinter <x/y/z> - Split interleaved slices in direction <x/y/z> into separate time points

Source code | Documentation

Examples

```python
>>> import copy
>>> from nipype.interfaces import niftyseg
>>> binary = niftyseg.BinaryMaths()
>>> binary.inputs.in_file = 'im1.nii'
>>> binary.inputs.output_datatype = 'float'
>>> # Test sub operation
>>> binary_sub = copy.deepcopy(binary)
>>> binary_sub.inputs.operation = 'sub'
>>> binary_sub.inputs.operand_file = 'im2.nii'
>>> binary_sub.cmdline
'seg_maths im1.nii -sub im2.nii -odt float im1_sub.nii'
>>> binary_sub.run()  # doctest: +SKIP
>>> # Test mul operation
>>> binary_mul = copy.deepcopy(binary)
>>> binary_mul.inputs.operation = 'mul'
>>> binary_mul.inputs.operand_value = 2.0
```

(continues on next page)
>>> binary_mul.cmdline
'seg_maths im1.nii -mul 2.00000000 -odt float im1_mul.nii'
>>> binary_mul.run()  # doctest: +SKIP

# Test llsnorm operation
>>> binary_llsnorm = copy.deepcopy(binary)
>>> binary_llsnorm.inputs.operation = 'llsnorm'
>>> binary_llsnorm.inputs.operand_file = 'im2.nii'
>>> binary_llsnorm.cmdline
'seg_maths im1.nii -llsnorm im2.nii -odt float im1_llsnorm.nii'
>>> binary_llsnorm.run()  # doctest: +SKIP

# Test splitinter operation
>>> binary_splitinter = copy.deepcopy(binary)
>>> binary_splitinter.inputs.operation = 'splitinter'
>>> binary_splitinter.inputs.operand_str = 'z'
>>> binary_splitinter.cmdline
'seg_maths im1.nii -splitinter z -odt float im1_splitinter.nii'
>>> binary_splitinter.run()  # doctest: +SKIP

Inputs:

[Optional]
out_file: (a pathlike object or string representing a file)
image to write
argument: ``%-s``, position: -2

output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
or 'input')
datatype to use for output (default uses input type)
argument: ``-odt %s``', position: -3

args: (a unicode string)
Additional parameters to the command
argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a

(continues on next page)
Outputs:

<table>
<thead>
<tr>
<th>out_file: (a pathlike object or string representing a file)</th>
</tr>
</thead>
<tbody>
<tr>
<td>image written after calculations</td>
</tr>
</tbody>
</table>

## 72.5.2 BinaryMathsInteger

Link to code

Wraps the executable command `seg_maths`. Interface for executable `seg_maths` from NiftySeg platform. Interface to use any integer mathematical operations that can be performed with the `seg_maths` command-line program.

See below for those operations: (requiring integer values)

equal - `<int>` - Get voxels equal to `<int>`
dil - `<int>` - Dilate the image `<int>` times (in voxels).
ero - `<int>` - Erode the image `<int>` times (in voxels).
arp - `<int>` - Extract time point `<int>`
crop - `<int>` - Crop `<int>` voxels around each 3D volume.
pad - `<int>` - Pad `<int>` voxels with NaN value around each 3D volume.

Source code | Documentation

### Examples

```python
>>> import copy
>>> from nipype.interfaces.niftyseg import BinaryMathsInteger
>>> binaryi = BinaryMathsInteger()
>>> binaryi.inputs.in_file = 'im1.nii'
>>> binaryi.inputs.output_datatype = 'float'
>>> # Test dil operation
>>> binaryi_dil = copy.deepcopy(binaryi)
>>> binaryi_dil.inputs.operation = 'dil'
>>> binaryi_dil.inputs.operand_value = 2
>>> binaryi_dil.cmdline
'seg_maths im1.nii -dil 2 -odt float im1_dil.nii'
>>> binaryi_dil.run()  # doctest: +SKIP

>>> # Test erode operation
>>> binaryi_ero = copy.deepcopy(binaryi)
>>> binaryi_ero.inputs.operation = 'ero'
>>> binaryi_ero.inputs.operand_value = 1
>>> binaryi_ero.cmdline
'seg_maths im1.nii -ero 1 -odt float im1_ero.nii'
>>> binaryi_ero.run()  # doctest: +SKIP

>>> # Test pad operation
>>> binaryi_pad = copy.deepcopy(binaryi)
>>> binaryi_pad.inputs.operation = 'pad'
>>> binaryi_pad.inputs.operand_value = 4
>>> binaryi_pad.cmdline
'seg_maths im1.nii -pad 4 -odt float im1_pad.nii'
>>> binaryi_pad.run()  # doctest: +SKIP
```

Inputs:
operation: ('dil' or 'ero' or 'tp' or 'equal' or 'pad' or 'crop')
  operation to perform
  argument: `-%s`, position: 4
operand_value: (an integer (int or long))
  int value to perform operation with
  argument: `%d`, position: 5
in_file: (a pathlike object or string representing an existing file)
  image to operate on
  argument: `-%s`, position: 2

[Optional]
out_file: (a pathlike object or string representing a file)
  image to write
  argument: `-%s`, position: -2
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double' or 'input')
  datatype to use for output (default uses input type)
  argument: `-%odt %s`, position: -3
args: (a unicode string)
  Additional parameters to the command
  argument: `-%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) 
Environment variables

Outputs:

out_file: (a pathlike object or string representing a file)
  image written after calculations

72.5.3 MathsCommand

Link to code
Wraps the executable command seg_maths.
Base Command Interface for seg_maths interfaces.
The executable seg_maths enables the sequential execution of arithmetic operations, like multiplication (-mul), division (-div) or addition (-add), binarisation (-bin) or thresholding (-thr) operations and convolution by a Gaussian kernel (-smo). It also allows mathematical morphology based operations like dilation (-dil), erosion (-ero), connected components (-lconcomp) and hole filling (-fill), Euclidean (-euc) and geodesic (-geo) distance transforms, local image similarity metric calculation (-lncc and -lssd). Finally, it allows multiple operations over the dimensionality of the image, from merging 3D images together as a 4D image (-merge) or splitting (-split or -tp) 4D images into several 3D images, to estimating the maximum, minimum and average over all time-points, etc.

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
  image to operate on
  argument: `-%s`, position: 2

[Optional]
out_file: (a pathlike object or string representing a file)
  image to write
  argument: `-%s`, position: -2

(continues on next page)
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double' or 'input')

datatype to use for output (default uses input type)

argument: `--odt %s`, position: -3

args: (a unicode string)

Additional parameters to the command

argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

Environment variables

Outputs:

out_file: (a pathlike object or string representing a file)

image written after calculations

72.5.4 Merge

Link to code

Wraps the executable command `seg_maths`.

Interface for executable `seg_maths` from NiftySeg platform.

Interface to use the merge operation that can be performed with the `seg_maths` command-line program.

See below for this option:

merge <i> <d> <files> Merge <i> images and the working image in the <d> dimension

Source code | Documentation

Examples

```python
>>> from nipype.interfaces import niftyseg
>>> node = niftyseg.Merge()
>>> node.inputs.in_file = 'im1.nii'
>>> files = ['im2.nii', 'im3.nii']
>>> node.inputs.merge_files = files
>>> node.inputs.dimension = 2
>>> node.inputs.output_datatype = 'float'
>>> node.cmdline
'seg_maths im1.nii -merge 2 2 im2.nii im3.nii -odt float im1_merged.nii'
```

Inputs:

[Mandatory]

dimension: (an integer (int or long))

Dimension to merge the images.

merge_files: (a list of items which are a pathlike object or string representing an existing file)

List of images to merge to the working image <input>.

argument: `%s`, position: 4

in_file: (a pathlike object or string representing an existing file)

image to operate on

argument: `%s`, position: 2

[Optional]

out_file: (a pathlike object or string representing a file)
image to write
argument: ``\`\$s``', position: -2
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
or 'input')
datatype to use for output (default uses input type)
argument: `\`-odt \$s``', position: -3
args: (a unicode string)
Additional parameters to the command
argument: `\`\$s``
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: { })
Environment variables

Outputs:
out_file: (a pathlike object or string representing a file)
image written after calculations

### 72.5.5 TupleMaths

Link to code
Wraps the executable command seg_maths.
Interface for executable seg_maths from NiftySeg platform.
Interface to use any tuple mathematical operations that can be performed
with the seg_maths command-line program.
See below for those operations:
lncc <file> <std> Local CC between current img and <file> on a kernel with <std>
lssd <file> <std> Local SSD between current img and <file> on a kernel with <std>
lltsnorm <file_norm> <float> Linear LTS normalisation assuming <float> percent outliers

Source code | Documentation

#### Examples

```python
>>> import copy
>>> from nipype.interfaces import niftyseg

>>> tuple = niftyseg.TupleMaths()

>>> tuple.inputs.in_file = 'im1.nii'
>>>tuple.inputs.output_datatype = 'float'

>>> # Test lncc operation
>>> tuple_lncc = copy.deepcopy(tuple)
>>> tuple_lncc.inputs.operation = 'lncc'
>>> tuple_lncc.inputs.operand_file1 = 'im2.nii'
>>> tuple_lncc.inputs.operand_value2 = 2.0
>>> tuple_lncc.cmdline
'seg_maths im1.nii -lncc im2.nii 2.00000000 -odt float im1_lncc.nii'

>>> tuple_lncc.run()  # doctest: +SKIP

>>> # Test lssd operation
>>> tuple_lssd = copy.deepcopy(tuple)
>>> tuple_lssd.inputs.operation = 'lssd'
>>> tuple_lssd.inputs.operand_file1 = 'im2.nii'
>>> tuple_lssd.inputs.operand_value2 = 1.0
```
>>> tuple_lssd.cmdline
'seg_maths im1.nii -lssd im2.nii 1.00000000 -odt float im1_lssd.nii'
>>> tuple_lssd.run()  # doctest: +SKIP

>>> # Test lltsnorm operation
>>> tuple_lltsnorm = copy.deepcopy(tuple)
>>> tuple_lltsnorm.inputs.operation = 'lltsnorm'
>>> tuple_lltsnorm.inputs.operand_file1 = 'im2.nii'
>>> tuple_lltsnorm.inputs.operand_value2 = 0.01
>>> tuple_lltsnorm.cmdline
'seg_maths im1.nii -lltsnorm im2.nii 0.01000000 -odt float im1_lltsnorm.nii'
>>> tuple_lltsnorm.run()  # doctest: +SKIP

Inputs:

[Mandatory]
operation: ('lncc' or 'lssd' or 'lltsnorm')
  operation to perform
    argument: ``%-s``, position: 4
operand_file1: (a pathlike object or string representing an existing file)
  image to perform operation 1 with
    argument: ``%-s``, position: 5
    mutually_exclusive: operand_value1
operand_value1: (a float)
  float value to perform operation 1 with
    argument: ``%.8f``, position: 5
    mutually_exclusive: operand_file1
operand_file2: (a pathlike object or string representing an existing file)
  image to perform operation 2 with
    argument: ``%-s``, position: 6
    mutually_exclusive: operand_value2
operand_value2: (a float)
  float value to perform operation 2 with
    argument: ``%.8f``, position: 6
    mutually_exclusive: operand_file2
in_file: (a pathlike object or string representing an existing file)
  image to operate on
    argument: ``%-s``, position: 2

[Optional]
out_file: (a pathlike object or string representing a file)
  image to write
    argument: ``%-s``, position: -2
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double' or 'input')
  datatype to use for output (default uses input type)
    argument: ``-odt %s``, position: -3
args: (a unicode string)
  Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  Environment variables

Outputs:
out_file: (a pathlike object or string representing a file)
image written after calculations

72.5.6 UnaryMaths

Link to code
Wraps the executable command `seg Maths`.
Interface for executable `seg Maths` from NiftySeg platform.
Interface to use any unary mathematical operations that can be performed
with the `seg Maths` command-line program.
See below for those operations:
sqrt - Square root of the image.
exp - Exponential root of the image.
log - Log of the image.
recip - Reciprocal (1/I) of the image.
abs - Absolute value of the image.
bin - Binarise the image.
otsu - Otsu thresholding of the current image.
lconcomp - Take the largest connected component
concomp6 - Label the different connected components with a 6NN kernel
concomp26 - Label the different connected components with a 26NN kernel
fill - Fill holes in binary object (e.g. fill ventricle in brain mask).
euc - Euclidean distance transform
tpmax - Get the time point with the highest value (binarise 4D probabilities)
tmean - Mean value of all time points.
tmax - Max value of all time points.
tmin - Mean value of all time points.
splitlab - Split the integer labels into multiple timepoints
removenan - Remove all NaNs and replace then with 0
isnan - Binary image equal to 1 if the value is NaN and 0 otherwise
subsamp2 - Subsample the image by 2 using NN sampling (qform and sform scaled)
scl - Reset scale and slope info.
4to5 - Flip the 4th and 5th dimension.
rang - Reset the image range to the min max.

Source code | Documentation

Examples

```python
>>> import copy
>>> from nipype.interfaces import niftyseg
>>> unary = niftyseg.UnaryMaths()
>>> unary.inputs.output_datatype = 'float'
>>> unary.inputs.in_file = 'im1.nii'
>>> # Test sqrt operation
>>> unary_sqrt = copy.deepcopy(unary)
>>> unary_sqrt.inputs.operation = 'sqrt'
>>> unary_sqrt.cmdline
'seg Maths im1.nii -sqrt -odt float im1_sqrt.nii'
>>> unary_sqrt.run()  # doctest: +SKIP
>>> # Test sqrt operation
>>> unary_abs = copy.deepcopy(unary)
>>> unary_abs.inputs.operation = 'abs'
>>> unary_abs.cmdline
```
(continues on next page)
'seg_maths im1.nii -abs -odt float im1_abs.nii'
>>> unary_abs.run()  # doctest: +SKIP
>>> # Test bin operation
>>> unary_bin = copy.deepcopy(unary)
>>> unary_bin.inputs.operation = 'bin'
>>> unary_bin.cmdline
'seg_maths im1.nii -bin -odt float im1_bin.nii'
>>> unary_bin.run()  # doctest: +SKIP
>>> # Test otsu operation
>>> unary_otsu = copy.deepcopy(unary)
>>> unary_otsu.inputs.operation = 'otsu'
>>> unary_otsu.cmdline
'seg_maths im1.nii -otsu -odt float im1_otsu.nii'
>>> unary_otsu.run()  # doctest: +SKIP
>>> # Test isnan operation
>>> unary_isnan = copy.deepcopy(unary)
>>> unary_isnan.inputs.operation = 'isnan'
>>> unary_isnan.cmdline
'seg_maths im1.nii -isnan -odt float im1_isnan.nii'
>>> unary_isnan.run()  # doctest: +SKIP

Inputs:

[Mandatory]
operation: ('sqrt' or 'exp' or 'log' or 'recip' or 'abs' or 'bin' or 'otsu' or 'lconcomp' or 'concomp6' or 'concomp26' or 'fill' or 'euc' or 'tpmax' or 'tmean' or 'tmax' or 'tmin' or 'splitlab' or 'removenan' or 'isnan' or 'subsamp2' or 'scl' or '4to5' or 'range')
operation to perform
argument: ``-%s``, position: 4
in_file: (a pathlike object or string representing an existing file)
image to operate on
argument: ``%s``, position: 2

[Optional]
out_file: (a pathlike object or string representing a file)
image to write
argument: ``-%s``, position: -2
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double' or 'input')
datatype to use for output (default uses input type)
argument: ``-odt %s``, position: -3
args: (a unicode string)
Additional parameters to the command
argument: ``-%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

out_file: (a pathlike object or string representing a file)
image written after calculations
72.6 interfaces.niftyseg.patchmatch

72.6.1 PatchMatch

Link to code
Wraps the executable command `seg_PatchMatch`.

Interface for executable `seg_PatchMatch` from NiftySeg platform.
The database file is a text file and in each line we have a template file, a mask with the search region to consider and a file with the label to propagate.

Input image, input mask, template images from database and masks from database must have the same 4D resolution (same number of XxYxZ voxels, modalities and/or time-points). Label files from database must have the same 3D resolution (XxYxZ voxels) than input image but can have different number of volumes than the input image allowing to propagate multiple labels in the same execution.

Source code | Documentation

Examples

```python
>>> from nipype.interfaces import niftyseg
>>> node = niftyseg.PatchMatch()
>>> node.inputs.in_file = 'im1.nii'
>>> node.inputs.mask_file = 'im2.nii'
>>> node.inputs.database_file = 'db.xml'
>>> node.cmdline
'seg_PatchMatch -i im1.nii -m im2.nii -db db.xml -o im1_pm.nii.gz'
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
    Input image to segment
    argument: ``-i %s``, position: 1
mask_file: (a pathlike object or string representing an existing file)
    Input mask for the area where applies PatchMatch
    argument: ``-m %s``, position: 2
database_file: (a pathlike object or string representing an existing file)
    Database with the segmentations
    argument: ``-db %s``, position: 3

[Optional]
out_file: (a pathlike object or string representing a file)
    The output filename of the patchmatch results
    argument: ``-o %s``, position: 4
patch_size: (an integer (int or long))
    Patch size, #voxels
    argument: ``-size %i``
cs_size: (an integer (int or long))
    Constrained search area size, number of times bigger than the patchsize
    argument: ``-cs %i``
match_num: (an integer (int or long))
    Number of better matching
    argument: ``-match %i``
pm_num: (an integer (int or long))
    Number of patchmatch executions
    argument: ``-pm %i``
it_num: (an integer (int or long))
   Number of iterations for the patchmatch algorithm
   argument: `\`-it %i\``
args: (a unicode string)
   Additional parameters to the command
   argument: `\`%s\``
environ: (a dictionary with keys which are a bytes or None or a value
   of class 'str' and with values which are a bytes or None or a
   value of class 'str', nipype default value: {})
   Environment variables

Outputs:

out_file: (a pathlike object or string representing a file)
   Output segmentation

72.7 interfaces.niftyseg.stats

72.7.1 BinaryStats

Link to code
Wraps the executable command seg_stats.
Interface for executable seg_stats from NiftySeg platform.
Interface to use any binary statistical operations that can be performed
with the seg_stats command-line program.
See below for those operations:
p - <float> - The <float>th percentile of all voxels intensity (float=[0,100])
sa - <ax> - Average of all voxels
ss - <ax> - Standard deviation of all voxels
svp - <ax> - Volume of all probabilistic voxels (sum(in) * <volume per voxel>)
al - <in2> - Average value in <in> for each label in <in2>
d - <in2> - Calculate the Dice score between all classes in <in> and <in2>
ncc - <in2> - Normalized cross correlation between <in> and <in2>
nmi - <in2> - Normalized Mutual Information between <in> and <in2>
vl - <csv> - Volume of each integer label <in>. Save to <csv> file.
nl - <csv> - Count of each label <in>. Save to <csv> file.
Source code | Documentation

Examples

```python
>>> import copy
>>> from nipype.interfaces import niftyseg
>>> binary = niftyseg.BinaryStats()
>>> binary.inputs.in_file = 'im1.nii'
>>> # Test sa operation
>>> binary_sa = copy.deepcopy(binary)
>>> binary_sa.inputs.operation = 'sa'
>>> binary_sa.inputs.operand_value = 2.0
>>> binary_sa.cmdline
'seg_stats im1.nii -sa 2.00000000'
>>> binary_sa.run()  # doctest: +SKIP
>>> # Test ncc operation
>>> binary_ncc = copy.deepcopy(binary)
```
>>> binary_ncc.inputs.operation = 'ncc'
>>> binary_ncc.inputs.operand_file = 'im2.nii'
>>> binary_ncc.cmdline
'seg_stats im1.nii -ncc im2.nii'
>>> binary_ncc.run()  # doctest: +SKIP

>>> # Test Nl operation
>>> binary_nl = copy.deepcopy(binary)

>>> binary_nl.inputs.operation = 'Nl'
>>> binary_nl.inputs.operand_file = 'output.csv'
>>> binary_nl.cmdline
'seg_stats im1.nii -Nl output.csv'
>>> binary_nl.run()  # doctest: +SKIP

Inputs:

[Mandatory]
operation: ('p' or 'sa' or 'ss' or 'svp' or 'al' or 'd' or 'ncc' or
    'nmi' or 'Vl' or 'Nl')
    operation to perform
    argument: ``-%s``, position: 4
operand_file: (a pathlike object or string representing an existing
    file)
    second image to perform operation with
    argument: ``%s``, position: 5
    mutually_exclusive: operand_value
operand_value: (a float)
    value to perform operation with
    argument: ``%.8f``, position: 5
    mutually_exclusive: operand_file
in_file: (a pathlike object or string representing an existing file)
    image to operate on
    argument: ``%s``, position: 2

[Optional]
mask_file: (a pathlike object or string representing an existing
    file)
    statistics within the masked area
    argument: ``-m %s``, position: -2
larger_voxel: (a float)
    Only estimate statistics if voxel is larger than <float>
    argument: ``-t %f``, position: -3
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables

Outputs:

output: (an array)
    Output array from seg_stats

72.7.2 StatsCommand

Link to code
Wraps the executable command seg_stats.
Base Command Interface for `seg_stats` interfaces.
The executable `seg_stats` enables the estimation of image statistics on continuous voxel intensities (average, standard deviation, min/max, robust range, percentiles, sum, probabilistic volume, entropy, etc) either over the full image or on a per slice basis (slice axis can be specified), statistics over voxel coordinates (location of max, min and centre of mass, bounding box, etc) and statistics over categorical images (e.g. per region volume, count, average, Dice scores, etc). These statistics are robust to the presence of NaNs, and can be constrained by a mask and/or thresholded at a certain level.

Inputs:

<table>
<thead>
<tr>
<th>[Mandatory]</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in_file</code>: (a pathlike object or string representing an existing file)</td>
<td>image to operate on</td>
</tr>
<tr>
<td></td>
<td>argument: <code>%s</code>, position: 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[Optional]</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mask_file</code>: (a pathlike object or string representing an existing file)</td>
<td>statistics within the masked area</td>
</tr>
<tr>
<td></td>
<td>argument: <code>&quot;-m %s&quot;</code>, position: -2</td>
</tr>
<tr>
<td><code>larger_voxel</code>: (a float)</td>
<td>Only estimate statistics if voxel is larger than &lt;float&gt;</td>
</tr>
<tr>
<td></td>
<td>argument: <code>&quot;-t %f&quot;</code>, position: -3</td>
</tr>
<tr>
<td><code>args</code>: (a unicode string)</td>
<td>Additional parameters to the command</td>
</tr>
<tr>
<td></td>
<td>argument: <code>%s</code></td>
</tr>
<tr>
<td><code>environ</code>: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})</td>
<td>Environment variables</td>
</tr>
</tbody>
</table>

Outputs:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>output</code>: (an array)</td>
<td>Output array from <code>seg_stats</code></td>
</tr>
</tbody>
</table>

### 72.7.3 UnaryStats

Link to code

Wraps the executable command `seg_stats`.

Interface for executable `seg_stats` from NiftySeg platform.

Interface to use any unary statistical operations that can be performed with the `seg_stats` command-line program.

See below for those operations:

- `r` - The range <min max> of all voxels.
- `R` - The robust range (assuming 2% outliers on both sides) of all voxels.
- `a` - Average of all voxels.
- `s` - Standard deviation of all voxels.
- `v` - Volume of all voxels above 0 (<# voxels> * <volume per voxel>)
- `vl` - Volume of each integer label (<# voxels per label> * <volume per voxel>)
- `vp` - Volume of all probabilistic voxels (sum(<in>) * <volume per voxel>)
- `n` - Count of all voxels above 0 (<# voxels>)
- `np` - Sum of all fuzzy voxels (sum(<in>))
- `e` - Entropy of all voxels.
- `ne` - Normalized entropy of all voxels.
- `x` - Location (i j k x y z) of the smallest value in the image.
- `X` - Location (i j k x y z) of the largest value in the image.
c - Location (i j k x y z) of the centre of mass of the object
B - Bounding box of all nonzero voxels [ xmin xsize ymin ysize zmin zsize ]
xvox - Output the number of voxels in the x direction. Replace x with y/z for other directions.
xdim - Output the voxel dimension in the x direction. Replace x with y/z for other directions.

Examples

```python
>>> import copy
>>> from nipype.interfaces import niftyseg

>>> unary = niftyseg.UnaryStats()
>>> unary.inputs.in_file = 'im1.nii'

>>> # Test v operation
>>> unary_v = copy.deepcopy(unary)
>>> unary_v.inputs.operation = 'v'
>>> unary_v.cmdline
'seg_stats im1.nii -v'
>>> unary_v.run()  # doctest: +SKIP

>>> # Test vl operation
>>> unary_vl = copy.deepcopy(unary)
>>> unary_vl.inputs.operation = 'vl'
>>> unary_vl.cmdline
'seg_stats im1.nii -vl'
>>> unary_vl.run()  # doctest: +SKIP

>>> # Test x operation
>>> unary_x = copy.deepcopy(unary)
>>> unary_x.inputs.operation = 'x'
>>> unary_x.cmdline
'seg_stats im1.nii -x'
>>> unary_x.run()  # doctest: +SKIP
```

Inputs:

```plaintext
[Mandatory]
operation: ('r' or 'R' or 'a' or 's' or 'v' or 'vl' or 'vp' or 'n' or 'np' or 'e' or 'ne' or 'x' or 'X' or 'c' or 'B' or 'xvox' or 'xdim')
  operation to perform
  argument: ``-%s``, position: 4
in_file: (a pathlike object or string representing an existing file)
  image to operate on
  argument: ``%s``, position: 2

[Optional]
mask_file: (a pathlike object or string representing an existing file)
  statistics within the masked area
  argument: ``-m %s``, position: -2
larger_voxel: (a float)
  Only estimate statistics if voxel is larger than <float>
  argument: ``-t %f``, position: -3
args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables
```
## Outputs:

<table>
<thead>
<tr>
<th>output: (an array)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output array from seg_stats</td>
</tr>
</tbody>
</table>
CHAPTER 73

73.1 interfaces.nipy

73.1.1 NipyBaseInterface

Link to code

Inputs:

None

Outputs:

None

73.2 interfaces.nipy.model

73.2.1 EstimateContrast

Link to code

Estimate contrast of a fitted model.

Inputs:

[Mandatory]

contrasts: (a list of items which are a tuple of the form: (a unicode string, 'T', a list of items which are a unicode string, a list of items which are a float) or a tuple of the form: (a unicode string, 'T', a list of items which are a unicode string, a list of items which are a float) or a tuple of the form: (a unicode string, 'F', a list of items which are a tuple of the form: (a unicode string, 'T', a list of items which are a tuple of the form: (a unicode string, 'T', a list of items which are a unicode string, a list of items which are a float) or a tuple of the form: (a unicode string, 'T', a list of items which are a unicode string, a list of items which are a float)))

List of contrasts with each contrast being a list of the form: ['name', 'stat', [condition list], [weight list], [session] (continues on next page)
### 73.2.2 FitGLM

**Link to code**

Fit GLM model based on the specified design. Supports only single or concatenated runs.

**Inputs:**

- **session_info:** (a list of from 1 to 1 items which are any value)
  - Session specific information generated by `modelgen.SpecifyModel`, FitGLM does not support multiple runs unless they are concatenated (see SpecifyModel options)
  - TR: (a float)

- **hrf_model:** ('Canonical' or 'Canonical With Derivative' or 'FIR', nipype default value: Canonical)
  - that specifies the hemodynamic response function it can be 'Canonical', 'Canonical With Derivative' or 'FIR'

- **drift_model:** ('Cosine' or 'Polynomial' or 'Blank', nipype default value: Cosine)
  - string that specifies the desired drift model, to be chosen among 'Polynomial', 'Cosine', 'Blank'

- **model:** ('ar1' or 'spherical', nipype default value: ar1)
  - autoregressive mode is available only for the kalman method

- **method:** ('kalman' or 'ols', nipype default value: kalman)
  - method to fit the model, ols or kalma; kalman is more time consuming but it supports autoregressive model

- **mask:** (a pathlike object or string representing an existing file)
  - restrict the fitting only to the region defined by this mask

**Outputs:**

- **stat_maps:** (a list of items which are a pathlike object or string representing an existing file)
- **z_maps:** (a list of items which are a pathlike object or string representing an existing file)
- **p_maps:** (a list of items which are a pathlike object or string representing an existing file)
normalize_design_matrix: (a boolean, nipype default value: False)

normalize (zscore) the regressors before fitting

save_residuals: (a boolean, nipype default value: False)

plot_design_matrix: (a boolean, nipype default value: False)

Outputs:

beta: (a pathlike object or string representing an existing file)

nvbeta: (any value)

s2: (a pathlike object or string representing an existing file)

dof: (any value)

constants: (any value)

axis: (any value)

reg_names: (a list of items which are any value)

residuals: (a pathlike object or string representing a file)

a: (a pathlike object or string representing an existing file)

73.3 interfaces.nipy.preprocess

73.3.1 ComputeMask

Link to code

Inputs:

[Mandatory]

mean_volume: (a pathlike object or string representing an existing file)

    mean EPI image, used to compute the threshold for the mask

[Optional]

reference_volume: (a pathlike object or string representing an existing file)

    reference volume used to compute the mask. If none is give, the mean
    volume is used.

m: (a float)

    lower fraction of the histogram to be discarded

M: (a float)

    upper fraction of the histogram to be discarded

cc: (a boolean)

    Keep only the largest connected component

Outputs:

brain_mask: (a pathlike object or string representing an existing file)

73.3.2 SpaceTimeRealigner

Link to code

Simultaneous motion and slice timing correction algorithm

If slice_times is not specified, this algorithm performs spatial motion correction

This interface wraps nipy’s SpaceTimeRealign algorithm [Roche2011] or simply the SpatialRealign algorithm when timing info is not provided.
Examples

```python
>>> from nipype.interfaces.nipy import SpaceTimeRealigner
>>> # Run spatial realignment only
>>> realigner = SpaceTimeRealigner()
>>> realigner.inputs.in_file = ['functional.nii']
>>> res = realigner.run() # doctest: +SKIP

>>> realigner = SpaceTimeRealigner()
>>> realigner.inputs.in_file = ['functional.nii']
>>> realigner.inputs.tr = 2
>>> realigner.inputs.slice_times = list(range(0, 3, 67))
>>> realigner.inputs.slice_info = 2
>>> res = realigner.run() # doctest: +SKIP
```

References

Inputs:

[Mandatory]

- **in_file**: (a list of items which are a pathlike object or string representing an existing file)
  - File to realign

[Optional]

- **tr**: (a float)
  - TR in seconds
  - requires: slice_times
- **slice_times**: (a list of items which are a float or 'asc_alt_2' or 'asc_alt_2_1' or 'asc_alt_half' or 'asc_alt_siemens' or 'ascending' or 'desc_alt_2' or 'desc_alt_half' or 'descending')
  - Actual slice acquisition times.
- **slice_info**: (an integer (int or long) or a list of items which are any value)
  - Single integer or length 2 sequence If int, the axis in `images` that is the slice axis. In a 4D image, this will often be axis = 2. If a 2 sequence, then elements are `''(slice_axis, slice_direction)'''`, where `''slice_axis''` is the slice axis in the image as above, and `''slice_direction''` is 1 if the slices were acquired slice 0 first, slice -1 last, or -1 if acquired slice -1 first, slice 0 last. If `slice_info` is an int, assume `''slice_direction''` == 1.
  - requires: slice_times

Outputs:

- **out_file**: (a list of items which are a pathlike object or string representing an existing file)
  - Realigned files
- **par_file**: (a list of items which are a pathlike object or string representing an existing file)
  - Motion parameter files. Angles are not euler angles

73.3.3 Trim

Link to code

Simple interface to trim a few volumes from a 4d fmri nifti file
Examples

```python
>>> from nipype.interfaces.nipy.preprocess import Trim
>>> trim = Trim()
>>> trim.inputs.in_file = 'functional.nii'
>>> trim.inputs.begin_index = 3  # remove 3 first volumes
>>> res = trim.run()  # doctest: +SKIP
```

Inputs:

- **in_file**: (a pathlike object or string representing an existing file) EPI image to trim
- **begin_index**: (an integer (int or long), nipype default value: 0) first volume
- **end_index**: (an integer (int or long), nipype default value: 0) last volume indexed as in python (and 0 for last)
- **out_file**: (a pathlike object or string representing a file) output filename
- **suffix**: (a unicode string, nipype default value: _trim) suffix for out_file to use if no out_file provided

Outputs:

- **out_file**: (a pathlike object or string representing an existing file)

### 73.4 interfaces.nipy.utils

#### 73.4.1 Similarity

**Link to code**

Calculates similarity between two 3D volumes. Both volumes have to be in the same coordinate system, same space within that coordinate system and with the same voxel dimensions.

Deprecated since version 0.10.0: Use `nipype.algorithms.metrics.Similarity` instead.

**Example**

```python
>>> from nipype.interfaces.nipy.utils import Similarity
>>> similarity = Similarity()
>>> similarity.inputs.volume1 = 'rc1s1.nii'
>>> similarity.inputs.volume2 = 'rc1s2.nii'
>>> similarity.inputs.mask1 = 'mask.nii'
>>> similarity.inputs.mask2 = 'mask.nii'
>>> similarity.inputs.metric = 'cr'
>>> res = similarity.run()  # doctest: +SKIP
```

Inputs:

- **volume1**: (a pathlike object or string representing an existing file) 3D volume
- **volume2**: (a pathlike object or string representing an existing file) 3D volume

(continues on next page)
[Optional]
mask1: (a pathlike object \textbf{or} string representing an existing file)
3D volume
mask2: (a pathlike object \textbf{or} string representing an existing file)
3D volume
metric: ('cc' \textbf{or} 'cr' \textbf{or} 'crl1' \textbf{or} 'mi' \textbf{or} 'nmi' \textbf{or} 'slr' \textbf{or} a
callable value, nipype default value: \texttt{None})
\texttt{str} \textbf{or} callable
Cost-function \textbf{for} assessing image similarity. If a string,
one of 'cc': correlation coefficient, 'cr': correlation
ratio, 'crl1': L1-norm based correlation ratio, 'mi': mutual
information, 'nmi': normalized mutual information, 'slr':
supervised log-likelihood ratio. If a callable, it should
take a two-dimensional array representing the image joint
histogram \textbf{as} an input \textbf{and return} a float.

Outputs:

\texttt{similarity: (a float)}
\hspace{1em} Similarity between volume 1 \textbf{and} 2
74.1 interfaces.nitime

74.1.1 CoherenceAnalyzer

Inputs:

[Optional]
in_file: (a pathlike object or string representing an existing file)
    csv file with ROIs on the columns and time-points on the rows. ROI
    names at the top row
    requires: TR
TR: (a float)
    The TR used to collect the data in your csv file <in_file>
in_TS: (any value)
    a nitime TimeSeries object
NFFT: (a long integer >= 32, nipype default value: 64)
    This is the size of the window used for the spectral estimation. Use
    values between 32 and the number of samples in your time-series.
    (Defaults to 64.)
n_overlap: (a long integer >= 0, nipype default value: 0)
    The number of samples which overlap between subsequent
    windows. (Defaults to 0)
frequency_range: (a list of from 2 to 2 items which are any value,
    nipype default value: [0.02, 0.15])
    The range of frequencies over which the analysis will
    average. [low, high] (Default [0.02, 0.15])
output_csv_file: (a pathlike object or string representing a file)
    File to write outputs (coherence, time-delay) with file-names:
    file_name_{coherence, timedelay}
output_figure_file: (a pathlike object or string representing a file)
    File to write output figures (coherence, time-delay) with file-names:
    file_name_{coherence, timedelay}. Possible formats: .png, .svg, .pdf, .jpg,...
figure_type: ('matrix' or 'network', nipype default value: matrix)
    The type of plot to generate, where 'matrix' denotes a matrix image
    and 'network' denotes a graph representation. Default: 'matrix'
### 74.2 interfaces.nitime.base

#### 74.2.1 NitimeBaseInterface

**Inputs:**

None

**Outputs:**

None
75.1 interfaces.semtools.brains.classify

75.1.1 BRAINSPosteriorToContinuousClass

Link to code
Wraps the executable command ‘‘ BRAINSPosteriorToContinuousClass ‘‘.

title: Tissue Classification
category: BRAINS.Classify
description: This program will generate an 8-bit continuous tissue classified image based on BRAINSABC posterior images.
version: 3.0
license: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt
contributor: Vincent A. Magnotta
acknowledgements: Funding for this work was provided by NIH/NINDS award NS050568

Inputs:

```
[Optional]
inputWhiteVolume: (a pathlike object or string representing an existing file)
   White Matter Posterior Volume
   argument: `--inputWhiteVolume %s`
inputBasalGmVolume: (a pathlike object or string representing an existing file)
   Basal Grey Matter Posterior Volume
   argument: `--inputBasalGmVolume %s`
inputSurfaceGmVolume: (a pathlike object or string representing an existing file)
   Surface Grey Matter Posterior Volume
   argument: `--inputSurfaceGmVolume %s`
inputCsfVolume: (a pathlike object or string representing an existing file)
   CSF Posterior Volume
   argument: `--inputCsfVolume %s`
inputVbVolume: (a pathlike object or string representing an existing file)
```

(continues on next page)
Venous Blood Posterior Volume
argument: `--inputVbVolume %s`

inputCrblGmVolume: (a pathlike object or string representing an existing file)
Cerebellum Grey Matter Posterior Volume
argument: `--inputCrblGmVolume %s`

inputCrblWmVolume: (a pathlike object or string representing an existing file)
Cerebellum White Matter Posterior Volume
argument: `--inputCrblWmVolume %s`

outputVolume: (a boolean or a pathlike object or string representing a file)
Output Continuous Tissue Classified Image
argument: `--outputVolume %s`

args: (a unicode string)
Additional parameters to the command
argument: `--%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing file)
Output Continuous Tissue Classified Image

75.2 interfaces.semtools.brains.segmentation

75.2.1 BRAINSTalairach

Link to code
Wraps the executable command "BRAINSTalairach".

title: BRAINS Talairach
category: BRAINS.Segmentation
description: This program creates a VTK structured grid defining the Talairach coordinate system based on four points: AC, PC, IRP, and SLA. The resulting structured grid can be written as either a classic VTK file or the new VTK XML file format. Two representations of the resulting grid can be written. The first is a bounding box representation that also contains the location of the AC and PC points. The second representation is the full Talairach grid representation that includes the additional rows of boxes added to the inferior allowing full coverage of the cerebellum.

version: 0.1
license: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt
contributor: Steven Dunn and Vincent Magnotta
acknowledgements: Funding for this work was provided by NIH/NINDS award NS050568

Inputs:

[Optional]
AC: (a list of items which are a float)
Location of AC Point
argument: `--AC %s`
ACisIndex: (a boolean)
   AC Point is Index
   argument: `--ACisIndex`
PC: (a list of items which are a float)
   Location of PC Point
   argument: `--PC %s`
PCisIndex: (a boolean)
   PC Point is Index
   argument: `--PCisIndex`
SLA: (a list of items which are a float)
   Location of SLA Point
   argument: `--SLA %s`
SLAisIndex: (a boolean)
   SLA Point is Index
   argument: `--SLAisIndex`
IRP: (a list of items which are a float)
   Location of IRP Point
   argument: `--IRP %s`
IRPisIndex: (a boolean)
   IRP Point is Index
   argument: `--IRPisIndex`
inputVolume: (a pathlike object or string representing an existing file)
   Input image used to define physical space of images
   argument: `--inputVolume %s`
outputBox: (a boolean or a pathlike object or string representing a file)
   Name of the resulting Talairach Bounding Box file
   argument: `--outputBox %s`
outputGrid: (a boolean or a pathlike object or string representing a file)
   Name of the resulting Talairach Grid file
   argument: `--outputGrid %s`
args: (a unicode string)
   Additional parameters to the command
   argument: `%%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
   Environment variables

Outputs:

outputBox: (a pathlike object or string representing an existing file)
   Name of the resulting Talairach Bounding Box file
outputGrid: (a pathlike object or string representing an existing file)
   Name of the resulting Talairach Grid file

75.2.2 BRAINSTalairachMask

Link to code
Wraps the executable command ' BRAINSTalairachMask '.
title: Talairach Mask
category: BRAINS.Segmentation
description: This program creates a binary image representing the specified Talairach region. The input is an
example image to define the physical space for the resulting image, the Talairach grid representation in VTK format, and the file containing the Talairach box definitions to be generated. These can be combined in BRAINS to create a label map using the procedure Brains::WorkupUtils::CreateLabelMapFromBinaryImages.

version: 0.1

license: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt
contributor: Steven Dunn and Vincent Magnotta
acknowledgements: Funding for this work was provided by NIH/NINDS award NS050568

Inputs:

```
[Optional]
inputVolume: (a pathlike object or string representing an existing
    file)
    Input image used to define physical space of resulting mask
    argument: `--inputVolume %s`

talairachParameters: (a pathlike object or string representing an
    existing file)
    Name of the Talairach parameter file.
    argument: `--talairachParameters %s`

talairachBox: (a pathlike object or string representing an existing
    file)
    Name of the Talairach box file.
    argument: `--talairachBox %s`

hemisphereMode: ('left' or 'right' or 'both')
    Mode for box creation: left, right, both
    argument: `--hemisphereMode %s`

expand: (a boolean)
    Expand exterior box to include surface CSF
    argument: `--expand`

outputVolume: (a boolean or a pathlike object or string representing
    a file)
    Output filename for the resulting binary image
    argument: `--outputVolume %s`

args: (a unicode string)
    Additional parameters to the command
    argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: { })
    Environment variables
```

Outputs:

```
outputVolume: (a pathlike object or string representing an existing
    file)
    Output filename for the resulting binary image
```

### 75.2.3 SimilarityIndex

Link to code

Wraps the executable command `SimilarityIndex`.  

title: BRAINSCut:SimilarityIndexComputation

category: BRAINS.Segmentation

description: Automatic analysis of BRAINSCut Output

version: 1.0

license: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt

contributor: Eunyoung Regin Kim
Inputs:

[Optional]
outputCSVFilename: (a pathlike object or string representing an existing file)
   output CSV Filename
   argument: `'--outputCSVFilename %s'`
ANNContinuousVolume: (a pathlike object or string representing an existing file)
   ANN Continuous volume to be compared to the manual volume
   argument: `'--ANNContinuousVolume %s'`
inputManualVolume: (a pathlike object or string representing an existing file)
   input manual(reference) volume
   argument: `'--inputManualVolume %s'`
thresholdInterval: (a float)
   Threshold interval to compute similarity index between zero and one
   argument: `'--thresholdInterval %f'`
args: (a unicode string)
   Additional parameters to the command
   argument: `' %s'`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

None

75.3 interfaces.semtools.brains.utilities

75.3.1 GenerateEdgeMapImage

Link to code
Wraps the executable command `` GenerateEdgeMapImage ''. title: GenerateEdgeMapImage
category: BRAINS.Utilities
description: Automatic edgemap generation for edge-guided super-resolution reconstruction
version: 1.0
contributor: Ali Ghayoor

Inputs:

[Optional]
inputMRVolumes: (a list of items which are a pathlike object or string representing an existing file)
   List of input structural MR volumes to create the maximum edgemap
   argument: `--inputMRVolumes %s...`
inputMask: (a pathlike object or string representing an existing file)
   Input mask file name. If set, image histogram percentiles will be calculated within the mask
   argument: `--inputMask %s`
minimumOutputRange: (an integer (int or long))
   Map lower quantile and below to minimum output range. It should be a small number greater than zero. Default is 1
   argument: `--minimumOutputRange %d`
maximumOutputRange: (an integer (int or long))
  Map upper quantile and above to maximum output range. Default is 255
  that is the maximum range of unsigned char
  argument: `--maximumOutputRange %d`
lowerPercentileMatching: (a float)
  Map lower quantile and below to minOutputRange. It should be a value
  between zero and one
  argument: `--lowerPercentileMatching %f`
upperPercentileMatching: (a float)
  Map upper quantile and above to maxOutputRange. It should be a value
  between zero and one
  argument: `--upperPercentileMatching %f`
outputEdgeMap: (a boolean or a pathlike object or string representing
  a file)
  output edgemap file name
  argument: `--outputEdgeMap %s`
outputMaximumGradientImage: (a boolean or a pathlike object or string
  representing a file)
  output gradient image file name
  argument: `--outputMaximumGradientImage %s`
numberOfThreads: (an integer (int or long))
  Explicitly specify the maximum number of threads to use.
  argument: `--numberOfThreads %d`
args: (a unicode string)
  Additional parameters to the command
  argument: `%%s`
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a value
  of class 'str', nipype default value: {}) 
Environment variables

Outputs:

outputEdgeMap: (a pathlike object or string representing an existing
  file)
  (required) output file name
outputMaximumGradientImage: (a pathlike object or string representing
  an existing file)
  output gradient image file name

75.3.2 GeneratePurePlugMask

Link to code
Wraps the executable command `' GeneratePurePlugMask `.

title: GeneratePurePlugMask
category: BRAINS.Utilities
description: This program gets several modality image files and returns a binary mask that defines the pure plugs
version: 1.0
contributor: Ali Ghayoor
 Inputs:

[Optional]
inputImageModalities: (a list of items which are a pathlike object or
  string representing an existing file)
  List of input image file names to create pure plugs mask
  argument: `--inputImageModalities %s...`
threshold: (a float)
  threshold value to define class membership
  argument: `--threshold %f`

numberOfSubSamples: (a list of items which are an integer (int or long))
  Number of continuous index samples taken at each direction of lattice space for each plug volume
  argument: `--numberOfSubSamples %s`

outputMaskFile: (a boolean or a pathlike object or string representing a file)
  Output binary mask file name
  argument: `--outputMaskFile %s`

args: (a unicode string)
  Additional parameters to the command
  argument: `--%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

outputMaskFile: (a pathlike object or string representing an existing file)
  (required) Output binary mask file name

### 75.3.3 HistogramMatchingFilter

**Link to code**
Wraps the executable command `'HistogramMatchingFilter`.'

**title:** Write Out Image Intensities
**category:** BRAINS.Utilities
**description:** For Analysis
**version:** 0.1
**contributor:** University of Iowa Department of Psychiatry, http://www.psychiatry.uiowa.edu

**Inputs:**

```
[Optional]
inputVolume: (a pathlike object or string representing an existing file)
  The Input image to be computed for statistics
  argument: `--inputVolume %s`

referenceVolume: (a pathlike object or string representing an existing file)
  The Input image to be computed for statistics
  argument: `--referenceVolume %s`

outputVolume: (a boolean or a pathlike object or string representing a file)
  Output Image File Name
  argument: `--outputVolume %s`

referenceBinaryVolume: (a pathlike object or string representing an existing file)
  referenceBinaryVolume
  argument: `--referenceBinaryVolume %s`

inputBinaryVolume: (a pathlike object or string representing an existing file)
```
inputBinaryVolume
argument: `--inputBinaryVolume %s`

numberOfMatchPoints: (an integer (int or long))
number of histogram matching points
argument: `--numberOfMatchPoints %d`

numberOfHistogramBins: (an integer (int or long))
number of histogram bin
argument: `--numberOfHistogramBins %d`

writeHistogram: (a unicode string)
decline if histogram data would be written with prefixe of the file
name
argument: `--writeHistogram %s`

histogramAlgorithm: ('OtsuHistogramMatching')
histogram algrithm selection
argument: `--histogramAlgorithm %s`

verbose: (a boolean)
verbose mode running for debbuging
argument: `--verbose`

args: (a unicode string)
Additional parameters to the command
argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})

Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing
file)
Output Image File Name

75.4 interfaces.semtools.converters

75.4.1 DWICompare

Link to code
Wraps the executable command ‘’ DWICompare ‘’.

title: Nrrd DWI comparison
category: Converters
description: Compares two nrrd format DWI images and verifies that gradient magnitudes, gradient directions,
measurement frame, and max B0 value are identicle. Used for testing DWIComvert.

version: 0.1.0.$Revision: 916 $(alpha)
license: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt
contributor: Mark Scully (UIowa)
acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded
by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.
Additional support for DTI data produced on Philips scanners was contributed by Vincent Magnotta and Hans
Johnson at the University of Iowa.

Inputs:

[Optional]
inputVolume1: (a pathlike object or string representing an existing
file)
First input volume (.nhdr or .nrrd)
argument: ``--inputVolume1 %s``

Second input volume (.nhdr or .nrrd)
argument: ``--inputVolume2 %s``

Additional parameters to the command
argument: ``%s``

Environment variables

Outputs:

None

### 75.4.2 DWISimpleCompare

Wraps the executable command `" DWISimpleCompare "`.

title: Nrrd DWI comparison
category: Converters
description: Compares two nrrd format DWI images and verifies that gradient magnitudes, gradient directions, measurement frame, and max B0 value are identical. Used for testing DWIConvert.
version: 0.1.0.$Revision: 916 $(alpha)
contributor: Mark Scully (UIowa)
acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149. Additional support for DTI data produced on Philips scanners was contributed by Vincent Magnotta and Hans Johnson at the University of Iowa.

Inputs:

[Optional]
inputVolume1: (a pathlike object or string representing an existing file)
argument: ``--inputVolume1 %s``

inputVolume2: (a pathlike object or string representing an existing file)
argument: ``--inputVolume2 %s``

checkDWIData: (a boolean)
check for existence of DWI data, and if present, compare it
argument: ``--checkDWIData``

args: (a unicode string)
Additional parameters to the command
argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

Environment variables
75.5 interfaces.semtools.diffusion.diffusion

75.5.1 DWIConvert

Link to code
Wraps the executable command "DWIConvert".

title: DWIConverter
category: Diffusion.Diffusion Data Conversion
description: Converts diffusion weighted MR images in dicom series into Nrrd format for analysis in Slicer. This program has been tested on only a limited subset of DTI dicom formats available from Siemens, GE, and Phillips scanners. Work in progress to support dicom multi-frame data. The program parses dicom header to extract necessary information about measurement frame, diffusion weighting directions, b-values, etc, and write out a nrrd image. For non-diffusion weighted dicom images, it loads in an entire dicom series and writes out a single dicom volume in a .nhdr/.raw pair.

version: Version 1.0
license: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt
contributor: Vince Magnotta (UIowa), Hans Johnson (UIowa), Joy Matsui (UIowa), Kent Williams (UIowa), Mark Scully (UIowa), Xiaodong Tao (GE)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149. Additional support for DTI data produced on Philips scanners was contributed by Vincent Magnotta and Hans Johnson at the University of Iowa.

Inputs:

[Optional]
cconversionMode: ('DicomToNrrd' or 'DicomToFSL' or 'NrrdToFSL' or 'FSLToNrrd')
Determine which conversion to perform. DicomToNrrd (default):
Convert Dicom series to NRRD DicomToFSL: Convert DICOM series to NIfTI File + gradient/bvalue text files NrrdToFSL: Convert DWI NRRD file to NIfTI File + gradient/bvalue text files FSLToNrrd: Convert NIfTI File + gradient/bvalue text files to NRRD file.
argument: '--conversionMode %s'

inputVolume: (a pathlike object or string representing an existing file)
Input DWI volume -- not used for DicomToNrrd mode.
argument: '--inputVolume %s'

outputVolume: (a boolean or a pathlike object or string representing a file)
Output filename (.nhdr or .nrrd)
argument: '--outputVolume %s'

inputDicomDirectory: (a pathlike object or string representing an existing directory)
Directory holding Dicom series
argument: '--inputDicomDirectory %s'

fslNIFTIFile: (a pathlike object or string representing an existing file)
4D NIfTI file containing gradient volumes
argument: '--fslNIFTIFile %s'

inputBValues: (a pathlike object or string representing an existing file)

The B Values are stored in FSL .bval text file format
argument: ```--inputBValues %s```

inputBVectors: (a pathlike object or string representing an existing file)
The Gradient Vectors are stored in FSL .bvec text file format
argument: ```--inputBVectors %s```

outputBValues: (a boolean or a pathlike object or string representing a file)
The B Values are stored in FSL .bval text file format (defaults to `<outputVolume>.bval`)
argument: ```--outputBValues %s```

outputBVectors: (a boolean or a pathlike object or string representing a file)
The Gradient Vectors are stored in FSL .bvec text file format (defaults to `<outputVolume>.bvec`)
argument: ```--outputBVectors %s```

fMRI: (a boolean)
Output a NRRD file, but without gradients
argument: ```--fMRI```

writeProtocolGradientsFile: (a boolean)
Write the protocol gradients to a file suffixed by '.txt' as they were specified in the procol by multiplying each diffusion gradient direction by the measurement frame. This file is for debugging purposes only, the format is not fixed, and will likely change as debugging of new dicom formats is necessary.
argument: ```--writeProtocolGradientsFile```

useIdentityMeasurementFrame: (a boolean)
Adjust all the gradients so that the measurement frame is an identity matrix.
argument: ```--useIdentityMeasurementFrame```

useBMATRIXGradientDirections: (a boolean)
Fill the nhdr header with the gradient directions and bvalues computed out of the BMATRIX. Only changes behavior for Siemens data. In some cases the standard public gradients are not properly computed. The gradients can emperically computed from the private BMATRIX fields. In some cases the private BMATRIX is consistent with the public gradients, but not in all cases, when it exists BMATRIX is usually most robust.
argument: ```--useBMATRIXGradientDirections```

outputDirectory: (a boolean or a pathlike object or string representing a directory)
Directory holding the output NRRD file
argument: ```--outputDirectory %s```

gradientVectorFile: (a boolean or a pathlike object or string representing a file)
Text file giving gradient vectors
argument: ```--gradientVectorFile %s```

smallGradientThreshold: (a float)
If a gradient magnitude is greater than 0 and less than smallGradientThreshold, then DWIConvert will display an error message and quit, unless the useBMATRIXGradientDirections option is set.
argument: ```--smallGradientThreshold %f```

allowLossyConversion: (a boolean)
The only supported output type is 'short'. Conversion from images of a different type may cause data loss due to rounding or truncation.
Use with caution!

argument: `''--allowLossyConversion ``

transposeInputBVectors: (a boolean)

FSL input BVectors are expected to be encoded in the input file as
one vector per line. If it is not the case, use this option to
transpose the file as it is read.

argument: `''--transposeInputBVectors ``

args: (a unicode string)

Additional parameters to the command

argument: `''%s``

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})

Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing
file)

Output filename (.nhdr or .nrrd)

outputBValues: (a pathlike object or string representing an existing
file)

The B Values are stored in FSL .bval text file format (defaults to
<outputVolume>.bval)

outputBVectors: (a pathlike object or string representing an existing
file)

The Gradient Vectors are stored in FSL .bvec text file format
(defaults to <outputVolume>.bvec)

outputDirectory: (a pathlike object or string representing an
existing directory)

Directory holding the output NRRD file

gradientVectorFile: (a pathlike object or string representing an
existing file)

Text file giving gradient vectors

### 75.5.2 dtiaverage

Link to code

Wraps the executable command " dtiaverage ".

title: DTIAverage (DTIProcess)

category: Diffusion.Diffusion Tensor Images.CommandLineOnly

description: dtiaverage is a program that allows to compute the average of an arbitrary number of tensor fields (listed after the --input option).

Several average method can be used (specified by the --method option): euclidian, log-euclidian and pga.

The default being euclidian.

version: 1.0.0


license: Copyright (c) Casey Goodlett. All rights reserved. See http://www.ia.unc.edu/dev/Copyright.htm for details. This software is distributed WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the above copyright notices for more information.

contributor: Casey Goodlett

Inputs:

[Optional]

inputs: {a list of items which are a pathlike object or string}
representing an existing file)
List of all the tensor fields to be averaged
argument: `--inputs %s...`
tensor_output: (a boolean or a pathlike object or string representing
a file)
Averaged tensor volume
argument: `--tensor_output %s`
DTI_double: (a boolean)
Tensor components are saved as doubles (cannot be visualized in
Slicer)
argument: `--DTI_double`
verbose: (a boolean)
produce verbose output
argument: `--verbose`
args: (a unicode string)
Additional parameters to the command
argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})

Outputs:

tensor_output: (a pathlike object
or string representing an existing
file)
Averaged tensor volume

75.5.3 dtiestim

Link to code
Wraps the executable command `` dtiestim ``.
title: DTIEstim (DTIProcess)
category: Diffusion.Diffusion Weighted Images
description: dtiestim is a tool that takes in a set of DWIs (with –dwi_image option) in nrrd format and estimates
a tensor field out of it. The output tensor file name is specified with the –tensor_output option. There are several
methods to estimate the tensors which you can specify with the option –method lls|wls|nls|ml. Here is a short
description of the different methods:
lls Linear least squares. Standard estimation technique that recovers the tensor parameters by multiplying the
log of the normalized signal intensities by the pseudo-inverse of the gradient matrix. Default option.
wls Weighted least squares. This method is similar to the linear least squares method except that the gradient
matrix is weighted by the original lls estimate. (See Salvador, R., Pena, A., Menon, D. K., Carpenter, T.
A., Pickard, J. D., and Bullmore, E. T. Formal characterization and extension of the linearized diffusion
tensor model. Human Brain Mapping 24, 2 (Feb. 2005), 144-155. for more information on this method).
This method is recommended for most applications. The weight for each iteration can be specified with
the –weight_iterations. It is not currently the default due to occasional matrix singularities.
nls Non-linear least squares. This method does not take the log of the signal and requires an optimization based
on levenberg-marquadt to optimize the parameters of the signal. The lls estimate is used as an initialization.
For this method the step size can be specified with the –step option.
ml Maximum likelihood estimation. This method is experimental and is not currently recommended. For this
ml method the sigma can be specified with the option –sigma and the step size can be specified with the
–step option.
You can set a threshold (–threshold) to have the tensor estimated to only a subset of voxels. All the baseline
voxel value higher than the threshold define the voxels where the tensors are computed. If not specified the
threshold is calculated using an OTSU threshold on the baseline image. The masked generated by the -t option
or by the otsu value can be saved with the \(-B0\) mask output option.
dtistim also can extract a few scalar images out of the DWI set of images:
- the average baseline image \((-B0)\) which is the average of all the B0s.
- the IDWI \((-idwi)\) which is the geometric mean of the diffusion images.

You can also load a mask if you want to compute the tensors only where the voxels are non-zero \((-brain\_mask)\) or a negative mask and the tensors will be estimated where the negative mask has zero values \((-bad\_region\_mask)\)

Inputs:

[Optional]

\texttt{dwi\_image}: (a pathlike object or string representing an existing file)

- DWI image volume (required)
- `\(--\texttt{dwi\_image} \%s\)`

\texttt{tensor\_output}: (a boolean or a pathlike object or string representing a file)

- Tensor OutputImage
- `\(--\texttt{tensor\_output} \%s\)`

\texttt{B0}: (a boolean or a pathlike object or string representing a file)

- Baseline image, average of all baseline images
- `\(--\texttt{B0} \%s\)`

\texttt{idwi}: (a boolean or a pathlike object or string representing a file)

- idwi output image. Image with isotropic diffusion-weighted information = geometric mean of diffusion images
- `\(--\texttt{idwi} \%s\)`

\texttt{B0\_mask\_output}: (a boolean or a pathlike object or string representing a file)

- B0 mask used for the estimation. B0 thresholded either with the \(-t\) option value or the automatic OTSU value
- `\(--\texttt{B0\_mask\_output} \%s\)`

\texttt{brain\_mask}: (a pathlike object or string representing an existing file)

- Brain mask. Image where for every voxel == 0 the tensors are not estimated. Be aware that in addition a threshold based masking will be performed by default. If such an additional threshold masking is NOT desired, then use option \(-t\) 0.
- `\(--\texttt{brain\_mask} \%s\)`

\texttt{bad\_region\_mask}: (a pathlike object or string representing an existing file)

- Bad region mask. Image where for every voxel > 0 the tensors are not estimated
- `\(--\texttt{bad\_region\_mask} \%s\)`

\texttt{method}: ('lls' or 'wls' or 'nls' or 'ml')

- Estimation method (lls:linear least squares, wls:weighted least squares, nls:non-linear least squares, ml:maximum likelihood)
- `\(--\texttt{method} \%s\)`

(continues on next page)
correction: ('none' or 'zero' or 'abs' or 'nearest')
  Correct the tensors if computed tensor is not semi-definite positive
  argument: `--correction %s`

threshold: (an integer (int or long))
  Baseline threshold for estimation. If not specified calculated using
  an OTSU threshold on the baseline image.
  argument: `--threshold %d`

weight_iterations: (an integer (int or long))
  Number of iterations to recalculate weightings from tensor estimate
  argument: `--weight_iterations %d`

step: (a float)
  Gradient descent step size (for nls and ml methods)
  argument: `--step %f`

sigma: (a float)
  argument: `--sigma %f`

DTI_double: (a boolean)
  Tensor components are saved as doubles (cannot be visualized in
  Slicer)
  argument: `--DTI_double`

verbose: (a boolean)
  produce verbose output
  argument: `--verbose`

defaultTensor: (a list of items which are a float)
  Default tensor used if estimated tensor is below a given threshold
  argument: `--defaultTensor %s`

shiftNeg: (a boolean)
  Shift eigenvalues so all are positive (accounts for bad tensors
  related to noise or acquisition error). This is the same option as
  the one available in DWIToDTIEstimation in Slicer (but instead of
  just adding the minimum eigenvalue to all the eigenvalues if it is
  smaller than 0, we use a coefficient to have strictly positive
  eigenvalues
  argument: `--shiftNeg`

shiftNegCoeff: (a float)
  Shift eigenvalues so all are positive (accounts for bad tensors
  related to noise or acquisition error). Instead of just adding the
  minimum eigenvalue to all the eigenvalues if it is smaller than 0,
  we use a coefficient to have strictly positive eigenvalues.
  Coefficient must be between 1.0 and 1.001 (included).
  argument: `--shiftNegCoeff %f`

args: (a unicode string)
  Additional parameters to the command
  argument: `-%s`

environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
  Environment variables

Outputs:

tensor_output: (a pathlike object or string representing an existing
  file)
  Tensor OutputImage

B0: (a pathlike object or string representing an existing file)
  Baseline image, average of all baseline images

idwi: (a pathlike object or string representing an existing file)
  idwi output image. Image with isotropic diffusion-weighted
information = geometric mean of diffusion images

B0_mask_output: (a pathlike object or string representing an existing file)
  B0 mask used for the estimation. B0 thresholded either with the -t option value or the automatic OTSU value

### 75.5.4 dtiprocess

**Link to code**

Wraps the executable command `dtiprocess`.

title: DTIProcess (DTIPProcess)
category: Diffusion.Diffusion Tensor Images
description: dtiprocess is a tool that handles tensor fields. It takes as an input a tensor field in nrrd format. It can generate diffusion scalar properties out of the tensor field such as: FA (–fa_output), Gradient FA image (–fa_gradient_output), color FA (–color_fa_output), MD (–md_output), Frobenius norm (–frobenius_norm_output), lbd1, lbd2, lbd3 (–lambda{1,2,3}_output), binary map of voxel where if any of the eigenvalue is negative, the voxel is set to 1 (–negative_eigenvector_output)

It also creates 4D images out of the tensor field such as: Highest eigenvector map (highest eigenvector at each voxel) (–principal_eigenvector_output)

**Masking capabilities:** For any of the processing done with dtiprocess, it’s possible to apply it on a masked region of the tensor field. You need to use the –mask option for any of the option to be applied on that tensor field subregion only. If you want to save the masked tensor field use the option –outmask and specify the new masked tensor field file name. dtiprocess also allows a range of transformations on the tensor fields. The transformed tensor field file name is specified with the option –deformation_output. There are 3 resampling interpolation methods specified with the tag –interpolation followed by the type to use (nearestneighbor, linear, cubic) Then you have several transformations possible to apply:

- Affine transformations using as an input
- itk affine transformation file (based on the itkAffineTransform class)
- Affine transformations using rview (details and download at [http://www.doc.ic.ac.uk/~dr/software/](http://www.doc.ic.ac.uk/~dr/software/)). There are 2 versions of rview both creating transformation files called dof files. The old version of rview outputs text files containing the transformation parameters. It can be read in with the –dof_file option. The new version outputs binary dof files. These dof files can be transformed into human readable file with the dof2mat tool which is part of the rview package. So you need to save the output of dof2mat into a text file which can then be used with the –newdof_file option. Usage example: dof2mat mynewdoffile.dof >> mynewdoffile.txt dtiprocess –dti_image mytensorfield.nhdr –newdof_file mynewdoffile.txt –rot_output myaffinetensorfield.nhdr

Non linear transformations as an input: The default transformation file type is d-field (displacement field) in nrrd format. The option to use is –forward with the name of the file. If the transformation file is a h-field you have to add the option –hField.

version: 1.0.1
license: Copyright (c) Casey Goodlett. All rights reserved.
  See [http://www.ia.unc.edu/dev/Copyright.htm](http://www.ia.unc.edu/dev/Copyright.htm) for details. This software is distributed WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the above copyright notices for more information.

contributor: Casey Goodlett

**Inputs:**

```
[Optional]
dti_image: (a pathlike object or string representing an existing file)
  DTI tensor volume
  argument: ‘"--dti_image %s"’

fa_output: (a boolean or a pathlike object or string representing a
```
file)
Fractional Anisotropy output file
argument: `-fa_output %s`

md_output: (a boolean or a pathlike object or string representing a file)
Mean Diffusivity output file
argument: `-md_output %s`

sigma: (a float)
Scale of gradients
argument: `-sigma %f`

fa_gradient_output: (a boolean or a pathlike object or string representing a file)
Fractional Anisotropy Gradient output file
argument: `-fa_gradient_output %s`

fa_gradmag_output: (a boolean or a pathlike object or string representing a file)
Fractional Anisotropy Gradient Magnitude output file
argument: `-fa_gradmag_output %s`

color_fa_output: (a boolean or a pathlike object or string representing a file)
Color Fractional Anisotropy output file
argument: `-color_fa_output %s`

principal_eigenvector_output: (a boolean or a pathlike object or string representing a file)
Principal Eigenvectors Output
argument: `-principal_eigenvector_output %s`

negative_eigenvector_output: (a boolean or a pathlike object or string representing a file)
Negative Eigenvectors Output: create a binary image where if any of the eigen value is below zero, the voxel is set to 1, otherwise 0.
argument: `-negative_eigenvector_output %s`

frobenius_norm_output: (a boolean or a pathlike object or string representing a file)
Frobenius Norm Output
argument: `-frobenius_norm_output %s`

lambda1_output: (a boolean or a pathlike object or string representing a file)
Axial Diffusivity - Lambda 1 (largest eigenvalue) output
argument: `-lambda1_output %s`

lambda2_output: (a boolean or a pathlike object or string representing a file)
Lambda 2 (middle eigenvalue) output
argument: `-lambda2_output %s`

lambda3_output: (a boolean or a pathlike object or string representing a file)
Lambda 3 (smallest eigenvalue) output
argument: `-lambda3_output %s`

RD_output: (a boolean or a pathlike object or string representing a file)
RD (Radial Diffusivity 1/2*(lambda2+lambda3)) output
argument: `-RD_output %s`

rot_output: (a boolean or a pathlike object or string representing a file)
Rotated tensor output file. Must also specify the dof file.
argument: `-rot_output %s`

affineitk_file: (a pathlike object or string representing an existing file)
Transformation file for affine transformation. ITK format.
argument: `''--affineitk_file %s''`

dof_file: (a pathlike object or string representing an existing file)
Transformation file for affine transformation. This can be ITK format (or the outdated RView).
argument: `''--dof_file %s''`

newdof_file: (a pathlike object or string representing an existing file)
Transformation file for affine transformation. RView NEW format.
txt file output of dof2mat
argument: `''--newdof_file %s''`

mask: (a pathlike object or string representing an existing file)
Mask tensors. Specify --outmask if you want to save the masked tensor field, otherwise the mask is applied just for the current processing
argument: `''--mask %s''`

outmask: (a boolean or a pathlike object or string representing a file)
Name of the masked tensor field.
argument: `''--outmask %s''`

hField: (a boolean)
forward and inverse transformations are h-fields instead of displacement fields
argument: `''--hField ''`

forward: (a pathlike object or string representing an existing file)
Forward transformation. Assumed to be a deformation field in world coordinates, unless the --h-field option is specified.
argument: `''--forward %s''`

deformation_output: (a boolean or a pathlike object or string representing a file)
Warped tensor field based on a deformation field. This option requires the --forward,-F transformation to be specified.
argument: `''--deformation_output %s''`

interpolation: ('nearestneighbor' or 'linear' or 'cubic')
Interpolation type (nearestneighbor, linear, cubic)
argument: `''--interpolation %s''`

reorientation: ('fs' or 'ppd')
Reorientation type (fs, ppd)
argument: `''--reorientation %s''`

correction: ('none' or 'zero' or 'abs' or 'nearest')
Correct the tensors if computed tensor is not semi-definite positive
argument: `''--correction %s''`

scalar_float: (a boolean)
Write scalar [FA,MD] as unscaled float (with their actual values, otherwise scaled by 10 000). Also causes FA to be unscaled [0..1].
argument: `''--scalar_float ''`

DTI_double: (a boolean)
Tensor components are saved as doubles (cannot be visualized in Slicer)
argument: `''--DTI_double ''`

verbose: (a boolean)
produce verbose output
argument: `''--verbose ''`

args: (a unicode string)
Additional parameters to the command
argument: `''%s''`

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}'.

Environment variables

Outputs:

fa_output: (a pathlike object or string representing an existing file)
    Fractional Anisotropy output file
md_output: (a pathlike object or string representing an existing file)
    Mean Diffusivity output file
fa_gradient_output: (a pathlike object or string representing an existing file)
    Fractional Anisotropy Gradient output file
fa_gradmag_output: (a pathlike object or string representing an existing file)
    Fractional Anisotropy Gradient Magnitude output file
color_fa_output: (a pathlike object or string representing an existing file)
    Color Fractional Anisotropy output file
principal_eigenvector_output: (a pathlike object or string representing an existing file)
    Principal Eigenvectors Output
negative_eigenvector_output: (a pathlike object or string representing an existing file)
    Negative Eigenvectors Output: create a binary image where if any of the eigen value is below zero, the voxel is set to 1, otherwise 0.
frobenius_norm_output: (a pathlike object or string representing an existing file)
    Frobenius Norm Output
lambda1_output: (a pathlike object or string representing an existing file)
    Axial Diffusivity - Lambda 1 (largest eigenvalue) output
lambda2_output: (a pathlike object or string representing an existing file)
    Lambda 2 (middle eigenvalue) output
lambda3_output: (a pathlike object or string representing an existing file)
    Lambda 3 (smallest eigenvalue) output
RD_output: (a pathlike object or string representing an existing file)
    RD (Radial Diffusivity 1/2*(lambda2+lambda3)) output
rot_output: (a pathlike object or string representing an existing file)
    Rotated tensor output file. Must also specify the dof file.
outmask: (a pathlike object or string representing an existing file)
    Name of the masked tensor field.
deformation_output: (a pathlike object or string representing an existing file)
    Warped tensor field based on a deformation field. This option requires the --forward,-F transformation to be specified.
75.6 interfaces.semtools.diffusion.gtract

75.6.1 compareTractInclusion

Link to code
Wraps the executable command "compareTractInclusion".

title: Compare Tracts
category: Diffusion.GTRACT
description: This program will halt with a status code indicating whether a test tract is nearly enough included in a standard tract in the sense that every fiber in the test tract has a low enough sum of squares distance to some fiber in the standard tract modulo spline resampling of every fiber to a fixed number of points.
version: 4.0.0
license: http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt
contributor: This tool was developed by Vincent Magnotta and Greg Harris.
acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

| [Optional] | testFiber: (a pathlike object or string representing an existing file) |
| Required: test fiber tract file name |
| argument: '--testFiber %s' |
| standardFiber: (a pathlike object or string representing an existing file) |
| Required: standard fiber tract file name |
| argument: '--standardFiber %s' |
| closeness: (a float) |
| Closeness of every test fiber to some fiber in the standard tract, computed as a sum of squares of spatial differences of standard points |
| argument: '--closeness %f' |
| numberOfPoints: (an integer (int or long)) |
| Number of points in comparison fiber pairs |
| argument: '--numberOfPoints %d' |
| testForBijection: (a boolean) |
| Flag to apply the closeness criterion both ways |
| argument: '--testForBijection ' |
| testForFiberCardinality: (a boolean) |
| Flag to require the same number of fibers in both tracts |
| argument: '--testForFiberCardinality ' |
| writeXMLPolyDataFile: (a boolean) |
| Flag to make use of XML files when reading and writing vtkPolyData. |
| argument: '--writeXMLPolyDataFile ' |
| numberOfThreads: (an integer (int or long)) |
| Explicitly specify the maximum number of threads to use. |
| argument: '--numberOfThreads %d' |
| args: (a unicode string) |
| Additional parameters to the command |
| argument: '%$s' |
| environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) |

Environment variables

Outputs:
75.6.2 extractNrrdVectorIndex

Link to code
Wraps the executable command `extractNrrdVectorIndex`.

title: Extract Nrrd Index
category: Diffusion.GTRACT
description: This program will extract a 3D image (single vector) from a vector 3D image at a given vector index.
version: 4.0.0
license: http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt
contributor: This tool was developed by Vincent Magnotta and Greg Harris.
aknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

```python
[Optional]
inputVolume: (a pathlike object or string representing an existing file)
    Required: input file containing the vector that will be extracted
    argument: '--inputVolume %s'
vectorIndex: (an integer (int or long))
    Index in the vector image to extract
    argument: '--vectorIndex %d'
setImageOrientation: ('AsAcquired' or 'Axial' or 'Coronal' or 'Sagittal')
    Sets the image orientation of the extracted vector (Axial, Coronal, Sagittal)
    argument: '--setImageOrientation %s'
outputVolume: (a boolean or a pathlike object or string representing a file)
    Required: name of output NRRD file containing the vector image at the given index
    argument: '--outputVolume %s'
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: '--numberOfThreads %d'
args: (a unicode string)
    Additional parameters to the command
    argument: '%s'
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})  
    Environment variables
```

Outputs:

```python
outputVolume: (a pathlike object or string representing an existing file)
    Required: name of output NRRD file containing the vector image at the given index
```
75.6.3 gtractAnisotropyMap

Link to code
Wraps the executable command ‘gtractAnisotropyMap’.

title: Anisotropy Map
category: Diffusion.GTRACT
description: This program will generate a scalar map of anisotropy, given a tensor representation. Anisotropy images are used for fiber tracking, but the anisotropy scalars are not defined along the path. Instead, the tensor representation is included as point data allowing all of these metrics to be computed using only the fiber tract point data. The images can be saved in any ITK supported format, but it is suggested that you use an image format that supports the definition of the image origin. This includes NRRD, NifTI, and Meta formats. These images can also be used for scalar analysis including regional anisotropy measures or VBM style analysis.

version: 4.0.0
license: http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt
contributor: This tool was developed by Vincent Magnotta and Greg Harris.
acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputTensorVolume</td>
<td>(a pathlike object or string representing an existing file) Required: input file containing the diffusion tensor image</td>
</tr>
<tr>
<td>anisotropyType</td>
<td>('ADC' or 'FA' or 'RA' or 'VR' or 'AD' or 'RD' or 'LI') Anisotropy Mapping Type: ADC, FA, RA, VR, AD, RD, LI</td>
</tr>
<tr>
<td>outputVolume</td>
<td>(a boolean or a pathlike object or string representing a file) Required: name of output NRRD file containing the selected kind of anisotropy scalar.</td>
</tr>
<tr>
<td>numberOfThreads</td>
<td>(an integer (int or long)) Explicitly specify the maximum number of threads to use.</td>
</tr>
<tr>
<td>args</td>
<td>(a unicode string) Additional parameters to the command argument: %s</td>
</tr>
<tr>
<td>environ</td>
<td>(a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables</td>
</tr>
</tbody>
</table>

Outputs:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outputVolume</td>
<td>(a pathlike object or string representing an existing file) Required: name of output NRRD file containing the selected kind of anisotropy scalar.</td>
</tr>
</tbody>
</table>

75.6.4 gtractAverageBvalues

Link to code
Wraps the executable command ‘gtractAverageBvalues’.

title: Average B-Values
category: Diffusion.GTRACT

description: This program will directly average together the baseline gradients (b value equals 0) within a DWI scan. This is usually used after gtractCoregBvalues.

version: 4.0.0


license: http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt

contributor: This tool was developed by Vincent Magnotta and Greg Harris.

acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

[Optional]

inputVolume: (a pathlike object or string representing an existing file)
  Required: input image file name containing multiple baseline gradients to average
  argument: `--inputVolume %s`

outputVolume: (a boolean or a pathlike object or string representing a file)
  Required: name of output NRRD file containing directly averaged baseline images
  argument: `--outputVolume %s`

directionsTolerance: (a float)
  Tolerance for matching identical gradient direction pairs
  argument: `--directionsTolerance %f`

averageB0only: (a boolean)
  Average only baseline gradients. All other gradient directions are not averaged, but retained in the outputVolume
  argument: `--averageB0only`

numberOfThreads: (an integer (int or long))
  Explicitly specify the maximum number of threads to use.
  argument: `--numberOfThreads %d`

args: (a unicode string)
  Additional parameters to the command
  argument: `%s`

eviron: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing file)
  Required: name of output NRRD file containing directly averaged baseline images

75.6.5 gtractClipAnisotropy

Link to code

Wraps the executable command " gtractClipAnisotropy ".

title: Clip Anisotropy

category: Diffusion.GTRACT

description: This program will zero the first and/or last slice of an anisotropy image, creating a clipped anisotropy image.

version: 4.0.0

Inputs:

[Optional]

inputVolume: (a pathlike object or string representing an existing file)
  Required: input image file name
  argument: `--inputVolume %s`

outputVolume: (a boolean or a pathlike object or string representing a file)
  Required: name of output NRRD file containing the clipped anisotropy image
  argument: `--outputVolume %s`

clipFirstSlice: (a boolean)
  Clip the first slice of the anisotropy image
  argument: `--clipFirstSlice```

clipLastSlice: (a boolean)
  Clip the last slice of the anisotropy image
  argument: `--clipLastSlice```

numberOfThreads: (an integer (int or long))
  Explicitly specify the maximum number of threads to use.
  argument: `--numberOfThreads %d`

args: (a unicode string)
  Additional parameters to the command
  argument: `%%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})  
  Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing file)
  Required: name of output NRRD file containing the clipped anisotropy image

### 75.6.6 gtractCoRegAnatomy

**Link to code**

Wraps the executable command `'' gtractCoRegAnatomy ''`.

title: Coregister B0 to Anatomy B-Spline
category: Diffusion.GTRACT
description: This program will register a Nrrd diffusion weighted 4D vector image to a fixed anatomical image. Two registration methods are supported for alignment with anatomical images: Rigid and B-Spline. The rigid registration performs a rigid body registration with the anatomical images and should be done as well to initialize the B-Spline transform. The B-Spline transform is the deformable transform, where the user can control the amount of deformation based on the number of control points as well as the maximum distance that these points can move. The B-Spline registration places a low dimensional grid in the image, which is deformed. This allows for some susceptibility related distortions to be removed from the diffusion weighted images. In general the amount of motion in the slice selection and read-out directions direction should be kept low. The distortion is in the phase encoding direction in the images. It is recommended that skull stripped (i.e. image containing only brain with skull removed) images shoud be used for image co-registration with the B-Spline transform.

version: 4.0.0
Inputs:

```python
[Optional]
inputVolume: (a pathlike object or string representing an existing file)
  Required: input vector image file name. It is recommended that the
  input volume is the skull stripped baseline image of the DWI scan.
  argument: "--inputVolume %s"

inputAnatomicalVolume: (a pathlike object or string representing an existing file)
  Required: input anatomical image file name. It is recommended that
  that the input anatomical image has been skull stripped and has the
  same orientation as the DWI scan.
  argument: "--inputAnatomicalVolume %s"

vectorIndex: (an integer (int or long))
  Vector image index in the moving image (within the DWI) to be used
  for registration.
  argument: "--vectorIndex %d"

inputRigidTransform: (a pathlike object or string representing an existing file)
  Required (for B-Spline type co-registration): input rigid transform
  file name. Used as a starting point for the anatomical B-Spline
  registration.
  argument: "--inputRigidTransform %s"

outputTransformName: (a boolean or a pathlike object or string representing a file)
  Required: filename for the fit transform.
  argument: "--outputTransformName %s"

transformType: ('Rigid' or 'Bspline')
  Transform Type: Rigid|Bspline
  argument: "--transformType %s"

numberOfIterations: (an integer (int or long))
  Number of iterations in the selected 3D fit
  argument: "--numberOfIterations %d"

gridSize: (a list of items which are an integer (int or long))
  Number of grid subdivisions in all 3 directions
  argument: "--gridSize %s"

borderSize: (an integer (int or long))
  Size of border
  argument: "--borderSize %d"

numberOfHistogramBins: (an integer (int or long))
  Number of histogram bins
  argument: "--numberOfHistogramBins %d"

spatialScale: (an integer (int or long))
  Scales the number of voxels in the image by this value to specify
  the number of voxels used in the registration
  argument: "--spatialScale %d"

convergence: (a float)
  Convergence Factor
  argument: "--convergence %f"

gradientTolerance: (a float)
  Gradient Tolerance
  argument: "--gradientTolerance %f"
```

(continues on next page)
maxBSplineDisplacement: (a float)
   Sets the maximum allowed displacements in image physical
   coordinates for BSpline control grid along each axis. A value of 0.0
   indicates that the problem should be unbounded. NOTE: This only
   constrains the BSpline portion, and does not limit the displacement
   from the associated bulk transform. This can lead to a substantial
   reduction in computation time in the BSpline optimizer.
   argument: `--maxBSplineDisplacement %f`

maximumStepSize: (a float)
   Maximum permitted step size to move in the selected 3D fit
   argument: `--maximumStepSize %f`

minimumStepSize: (a float)
   Minimum required step size to move in the selected 3D fit without
   converging -- decrease this to make the fit more exacting
   argument: `--minimumStepSize %f`

translationScale: (a float)
   How much to scale up changes in position compared to unit rotational
   changes in radians -- decrease this to put more translation in the
   fit
   argument: `--translationScale %f`

relaxationFactor: (a float)
   Fraction of gradient from Jacobian to attempt to move in the
   selected 3D fit
   argument: `--relaxationFactor %f`

numberOfSamples: (an integer (int or long))
   The number of voxels sampled for mutual information computation.
   Increase this for a slower, more careful fit. NOTE that it is
   suggested to use samplingPercentage instead of this option. However,
   if set, it overwrites the samplingPercentage option.
   argument: `--numberOfSamples %d`

samplingPercentage: (a float)
   This is a number in (0.0,1.0] interval that shows the percentage of
   the input fixed image voxels that are sampled for mutual information
   computation. Increase this for a slower, more careful fit. You can
   also limit the sampling focus with ROI masks and ROIAUTO mask
   generation. The default is to use approximately 5% of voxels (for
   backwards compatibility 5% ~= 500000/(256*256*256)). Typical values
   range from 1% for low detail images to 20% for high detail images.
   argument: `--samplingPercentage %f`

useMomentsAlign: (a boolean)
   MomentsAlign assumes that the center of mass of the images represent
   similar structures. Perform a MomentsAlign registration as part of
   the sequential registration steps. This option MUST come first, and
   CAN NOT be used with either CenterOfHeadLAlign, GeometryAlign, or
   initialTransform file. This family of options supercedes the use of
   transformType if any of them are set.
   argument: `--useMomentsAlign`

useGeometryAlign: (a boolean)
   GeometryAlign on assumes that the center of the voxel lattice of the
   images represent similar structures. Perform a GeometryCenterAlign
   registration as part of the sequential registration steps. This
   option MUST come first, and CAN NOT be used with either
   MomentsAlign, CenterOfHeadAlign, or initialTransform file. This
   family of options supercedes the use of transformType if any of them
   are set.
   argument: `--useGeometryAlign`

useCenterOfHeadAlign: (a boolean)
CenterOfHeadAlign attempts to find a hemisphere full of foreground voxels from the superior direction as an estimate of where the center of a head shape would be to drive a center of mass estimate. Perform a CenterOfHeadAlign registration as part of the sequential registration steps. This option MUST come first, and CAN NOT be used with either MomentsAlign, GeometryAlign, or initialTransform file. This family of options supercedes the use of transformType if any of them are set.

argument: `--useCenterOfHeadAlign`

numberOfThreads: (an integer (int or long))
Explicitly specify the maximum number of threads to use.
argument: `--numberOfThreads %d`

args: (a unicode string)
Additional parameters to the command
argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

outputTransformName: (a pathlike object or string representing an existing file)
Required: filename for the fit transform.

75.6.7 gtractConcatDwi

Link to code
Wraps the executable command ' gtractConcatDwi .'

title: Concat DWI Images
category: Diffusion.GTRACT
description: This program will concatenate two DTI runs together.
version: 4.0.0
license: http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt
contributor: This tool was developed by Vincent Magnotta and Greg Harris.
acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

[Optional]

inputVolume: (a list of items which are a pathlike object or string representing an existing file)
Required: input file containing the first diffusion weighted image
argument: `--inputVolume %s...`

ignoreOrigins: (a boolean)
If image origins are different force all images to origin of first image
argument: `--ignoreOrigins`

outputVolume: (a boolean or a pathlike object or string representing a file)
Required: name of output NRRD file containing the combined diffusion weighted images.
argument: `--outputVolume %s`

numberOfThreads: (an integer (int or long))

(continues on next page)
Explicitly specify the maximum number of threads to use.
argument: `'--numberOfThreads %d'`

args: (a unicode string)
    Additional parameters to the command
argument: `'%s'`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

### Outputs:

<table>
<thead>
<tr>
<th>outputVolume: (a pathlike object or string representing an existing file)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required: name of output NRRD file containing the combined diffusion weighted images.</td>
</tr>
</tbody>
</table>

### 75.6.8 gtractCopyImageOrientation

**Link to code**
Wraps the executable command `'gtractCopyImageOrientation'`.

**Title:** Copy Image Orientation

**Category:** Diffusion.GTRACT

**Description:**
This program will copy the orientation from the reference image into the moving image. Currently, the registration process requires that the diffusion weighted images and the anatomical images have the same image orientation (i.e. Axial, Coronal, Sagittal). It is suggested that you copy the image orientation from the diffusion weighted images and apply this to the anatomical image. This image can be subsequently removed after the registration step is complete. We anticipate that this limitation will be removed in future versions of the registration programs.

**Version:** 4.0.0


**License:** [http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt](http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt)

**Contributor:**
This tool was developed by Vincent Magnotta and Greg Harris.

**Acknowledgements:**
Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

### Inputs:

[Optional]

<table>
<thead>
<tr>
<th>inputVolume: (a pathlike object or string representing an existing file)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required: input file containing the signed short image to reorient without resampling.</td>
</tr>
<tr>
<td>argument: <code>'--inputVolume %s'</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>inputReferenceVolume: (a pathlike object or string representing an existing file)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required: input file containing orientation that will be cloned.</td>
</tr>
<tr>
<td>argument: <code>'--inputReferenceVolume %s'</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>outputVolume: (a boolean or a pathlike object or string representing a file)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required: name of output NRRD or Nifti file containing the reoriented image in reference image space.</td>
</tr>
<tr>
<td>argument: <code>'--outputVolume %s'</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>numberOfThreads: (an integer (int or long))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explicitly specify the maximum number of threads to use.</td>
</tr>
<tr>
<td>argument: <code>'--numberOfThreads %d'</code></td>
</tr>
</tbody>
</table>
75.6.9 \texttt{gtractCoregBvalues}

\textbf{Link to code}

Wraps the executable command ``gtractCoregBvalues``.

\textbf{title: Coregister B-Values}

\textbf{category: Diffusion.GTRACT}

\textbf{description:} This step should be performed after converting DWI scans from DICOM to NRRD format. This program will register all gradients in a NRRD diffusion weighted 4D vector image (moving image) to a specified index in a fixed image. It also supports co-registration with a T2 weighted image or field map in the same plane as the DWI data. The fixed image for the registration should be a b0 image. A mutual information metric cost function is used for the registration because of the differences in signal intensity as a result of the diffusion gradients. The full affine allows the registration procedure to correct for eddy current distortions that may exist in the data. If the eddyCurrentCorrection is enabled, relaxationFactor (0.25) and maximumStepSize (0.1) should be adjusted.

\textbf{version: 4.0.0}


\textbf{license:} http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt

\textbf{contributor:} This tool was developed by Vincent Magnotta and Greg Harris.

\textbf{acknowledgements:} Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

\textbf{Inputs:}

\begin{verbatim}
[Optional]
movingVolume: (a pathlike object or string representing an existing file)
    Required: input moving image file name. In order to register gradients within a scan to its first gradient, set the movingVolume and fixedVolume as the same image.
    argument: `--movingVolume %s`

fixedVolume: (a pathlike object or string representing an existing file)
    Required: input fixed image file name. It is recommended that this image should either contain or be a b0 image.
    argument: `--fixedVolume %s`

fixedVolumeIndex: (an integer (int or long))
    Index in the fixed image for registration. It is recommended that this image should be a b0 image.
    argument: `--fixedVolumeIndex %d`

outputVolume: (a boolean or a pathlike object or string representing a file)
    
\end{verbatim}

\textbf{Outputs:}

\begin{verbatim}
outputVolume: (a pathlike object or string representing an existing file)
    Required: name of output NRRD or Nifti file containing the reoriented image in reference image space.
\end{verbatim}
Required: name of output NRRD file containing moving images
individually resampled and fit to the specified fixed image index.
argument: `--outputVolume %s`

outputTransform: (a boolean or a pathlike object or string
representing a file)
Registration 3D transforms concatenated in a single output file.
There are no tools that can use this, but can be used for debugging
purposes.
argument: `--outputTransform %s`

eddyCurrentCorrection: (a boolean)
Flag to perform eddy current corection in addition to motion
correction (recommended)
argument: `--eddyCurrentCorrection`

numberOfIterations: (an integer (int or long))
Number of iterations in each 3D fit
argument: `--numberOfIterations %d`

numberOfSpatialSamples: (an integer (int or long))
The number of voxels sampled for mutual information computation.
Increase this for a slower, more careful fit. NOTE that it is
suggested to use samplingPercentage instead of this option. However,
if set, it overwrites the samplingPercentage option.
argument: `--numberOfSpatialSamples %d`

samplingPercentage: (a float)
This is a number in \((0.0,1.0]\) interval that shows the percentage of
the input fixed image voxels that are sampled for mutual information
computation. Increase this for a slower, more careful fit. You can
also limit the sampling focus with ROI masks and ROIAUTO mask
generation. The default is to use approximately 5% of voxels (for
backwards compatibility 5% \(\approx 500000/(256*256*256)\)). Typical values
range from 1% for low detail images to 20% for high detail images.
argument: `--samplingPercentage %f`

relaxationFactor: (a float)
Fraction of gradient from Jacobian to attempt to move in each 3D fit
step (adjust when eddyCurrentCorrection is enabled; suggested value
= 0.25)
argument: `--relaxationFactor %f`

maximumStepSize: (a float)
Maximum permitted step size to move in each 3D fit step (adjust when
eddyCurrentCorrection is enabled; suggested value = 0.1)
argument: `--maximumStepSize %f`

minimumStepSize: (a float)
Minimum required step size to move in each 3D fit step without
converging -- decrease this to make the fit more exacting
argument: `--minimumStepSize %f`

spatialScale: (a float)
How much to scale up changes in position compared to unit rotational
changes in radians -- decrease this to put more rotation in the fit
argument: `--spatialScale %f`

registerB0Only: (a boolean)
Register the B0 images only
argument: `--registerB0Only`

debugLevel: (an integer (int or long))
Display debug messages, and produce debug intermediate results.
0=OFF, 1=Minimal, 10=Maximum debugging.
argument: `--debugLevel %d`

numberOfThreads: (an integer (int or long))
Explicitly specify the maximum number of threads to use.
argument: `''--numberOfThreads %d''`
args: (a unicode string)
   Additional parameters to the command
   argument: `''%s''`
environ: (a dictionary with keys which are a bytes or None or a value
   of class 'str' and with values which are a bytes or None or a
   value of class 'str', nipype default value: { })
   Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing
   file)
   Required: name of output NRRD file containing moving images
   individually resampled and fit to the specified fixed image index.
outputTransform: (a pathlike object or string representing an
   existing file)
   Registration 3D transforms concatenated in a single output file.
   There are no tools that can use this, but can be used for debugging
   purposes.

75.6.10 gtractCostFastMarching

Link to code

Wraps the executable command `'' gtractCostFastMarching ''`.
title: Cost Fast Marching
category: Diffusion.GTRACT
description: This program will use a fast marching fiber tracking algorithm to identify fiber tracts from a tensor
   image. This program is the first portion of the algorithm. The user must first run gtractFastMarchingTracking
to generate the actual fiber tracts. This algorithm is roughly based on the work by G. Parker et al. from IEEE
Transactions On Medical Imaging, 21(5): 505-512, 2002. An additional feature of including anisotropy into the
vcl_cost function calculation is included.
version: 4.0.0
license: http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt
contributor: This tool was developed by Vincent Magnotta and Greg Harris. The original code here was devel-
   oped by Daisy Espino.
acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1
Inputs:

[Optional]
inputTensorVolume: (a pathlike object or string representing an
   existing file)
   Required: input tensor image file name
   argument: `''--inputTensorVolume %s''`
inputAnisotropyVolume: (a pathlike object or string representing an
   existing file)
   Required: input anisotropy image file name
   argument: `''--inputAnisotropyVolume %s''`
inputStartingSeedsLabelMapVolume: (a pathlike object or string
   representing an existing file)
   Required: input starting seeds LabelMap image file name
   argument: `''--inputStartingSeedsLabelMapVolume %s''`
startingSeedsLabel: (an integer (int or long))
Label value for Starting Seeds
argument: `--startingSeedsLabel %d`

outputCostVolume: (a boolean or a pathlike object or string representing a file)
Output vcl_cost image
argument: `--outputCostVolume %s`

outputSpeedVolume: (a boolean or a pathlike object or string representing a file)
Output speed image
argument: `--outputSpeedVolume %s`

anisotropyWeight: (a float)
Anisotropy weight used for vcl_cost function calculations
argument: `--anisotropyWeight %f`

stoppingValue: (a float)
Terminating value for vcl_cost function estimation
argument: `--stoppingValue %f`

seedThreshold: (a float)
Anisotropy threshold used for seed selection
argument: `--seedThreshold %f`

numberOfThreads: (an integer (int or long))
Explicitly specify the maximum number of threads to use.
argument: `--numberOfThreads %d`

args: (a unicode string)
Additional parameters to the command
argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

outputCostVolume: (a pathlike object or string representing an existing file)
Output vcl_cost image

outputSpeedVolume: (a pathlike object or string representing an existing file)
Output speed image

75.6.11 gtractCreateGuideFiber

Link to code
Wraps the executable command `gtractCreateGuideFiber`.

title: Create Guide Fiber
category: Diffusion.GTRACT
description: This program will create a guide fiber by averaging fibers from a previously generated tract.
version: 4.0.0
license: http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt
contributor: This tool was developed by Vincent Magnotta and Greg Harris.
acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

[Optional]
inputFiber: (a pathlike object or string representing an existing file)
Required: input fiber tract file name
argument: ``--inputFiber %s``

numberOfPoints: (an integer (int or long))
Number of points in output guide fiber
argument: ``--numberOfPoints %d``

outputFiber: (a boolean or a pathlike object or string representing a file)
Required: output guide fiber file name
argument: ``--outputFiber %s``

writeXMLPolyDataFile: (a boolean)
Flag to make use of XML files when reading and writing vtkPolyData.
argument: ``--writeXMLPolyDataFile ``

numberOfThreads: (an integer (int or long))
Explicitly specify the maximum number of threads to use.
argument: ``--numberOfThreads %d``

args: (a unicode string)
Additional parameters to the command
argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}))
Environment variables

**Outputs:**

outputFiber: (a pathlike object or string representing an existing file)
Required: output guide fiber file name

### 75.6.12 gtractFastMarchingTracking

**Link to code**
Wraps the executable command `gtractFastMarchingTracking`.

**Title:** Fast Marching Tracking

**Category:** Diffusion.GTRACT

**Description:** This program will use a fast marching fiber tracking algorithm to identify fiber tracts from a tensor image. This program is the second portion of the algorithm. The user must first run `gtractCostFastMarching` to generate the vcl_cost image. The second step of the algorithm implemented here is a gradient descent solution from the defined ending region back to the seed points specified in `gtractCostFastMarching`. This algorithm is roughly based on the work by G. Parker et al. from IEEE Transactions On Medical Imaging, 21(5): 505-512, 2002. An additional feature of including anisotropy into the vcl_cost function calculation is included.

**Version:** 4.0.0


**License:** [http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt](http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt)

**Contributor:** This tool was developed by Vincent Magnotta and Greg Harris. The original code here was developed by Daisy Espino.

**Acknowledgements:** Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

**Inputs:**

[Optional]

inputTensorVolume: (a pathlike object or string representing an existing file)
Required: input tensor image file name

(continues on next page)
argument: `--inputTensorVolume %s`
inputAnisotropyVolume: (a pathlike object or string representing an existing file)
Required: input anisotropy image file name
argument: `--inputAnisotropyVolume %s`
inputCostVolume: (a pathlike object or string representing an existing file)
Required: input vcl_cost image file name
argument: `--inputCostVolume %s`
inputStartingSeedsLabelMapVolume: (a pathlike object or string representing an existing file)
Required: input starting seeds LabelMap image file name
argument: `--inputStartingSeedsLabelMapVolume %s`
startingSeedsLabel: (an integer (int or long))
Label value for Starting Seeds
argument: `--startingSeedsLabel %d`
outputTract: (a boolean or a pathlike object or string representing a file)
Required: name of output vtkPolydata file containing tract lines and the point data collected along them.
argument: `--outputTract %s`
writeXMLPolyDataFile: (a boolean)
Flag to make use of the XML format for vtkPolyData fiber tracts.
argument: `--writeXMLPolyDataFile`
numberOfIterations: (an integer (int or long))
Number of iterations used for the optimization
argument: `--numberOfIterations %d`
seedThreshold: (a float)
Anisotropy threshold used for seed selection
argument: `--seedThreshold %f`
trackingThreshold: (a float)
Anisotropy threshold used for fiber tracking
argument: `--trackingThreshold %f`
costStepSize: (a float)
Cost image sub-voxel sampling
argument: `--costStepSize %f`
maximumStepSize: (a float)
Maximum step size to move when tracking
argument: `--maximumStepSize %f`
minimumStepSize: (a float)
Minimum step size to move when tracking
argument: `--minimumStepSize %f`
numberOfThreads: (an integer (int or long))
Explicitly specify the maximum number of threads to use.
argument: `--numberOfThreads %d`
args: (a unicode string)
Additional parameters to the command
argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

outputTract: (a pathlike object or string representing an existing file)
Required: name of output vtkPolydata file containing tract lines and the point data collected along them.

75.6.13 gtractFiberTracking

Link to code

Wraps the executable command ‘’ gtractFiberTracking ‘’.

title: Fiber Tracking
category: Diffusion.GTRACT
description: This program implements four fiber tracking methods (Free, Streamline, GraphSearch, Guided). The output of the fiber tracking is vtkPolyData (i.e. Polylines) that can be loaded into Slicer3 for visualization. The poly data can be saved in either old VTK format files (.vtk) or in the new VTK XML format (.xml). The polylines contain point data that defines their Tensor at each point along the fiber tract. This can then be used to rendered as glyphs in Slicer3 and can be used to define several scalar measures without referencing back to the anisotropy images. (1) Free tracking is a basic streamlines algorithm. This is a direct implementation of the method original proposed by Basser et al. The tracking follows the primary eigenvector. The tracking begins with seed points in the starting region. Only those voxels above the specified anisotropy threshold in the starting region are used as seed points. Tracking terminates either as a result of maximum fiber length, low anisotropy, or large curvature. This is a great way to explore your data. (2) The streamlines algorithm is a direct implementation of the method originally proposed by Basser et al. The tracking follows the primary eigenvector. The tracking begins with seed points in the starting region. Only those voxels above the specified anisotropy threshold in the starting region are used as seed points. Tracking terminates either by reaching the ending region or reaching some stopping criteria. Stopping criteria are specified using the following parameters: tracking threshold, curvature threshold, and max length. Only paths terminating in the ending region are kept in this method. The TEND algorithm proposed by Lazar et al. (Human Brain Mapping 18:306-321, 2003) has been instrumented. This can be enabled using the –useTend option while performing Streamlines tracking. This utilizes the entire diffusion tensor to deflect the incoming vector instead of simply following the primary eigenvector. The TEND parameters are set using the –tendF and –tendG options. (3) Graph Search tracking is the first step in the full GTRACT algorithm developed by Cheng et al. (NeuroImage 31(3): 1075-1085, 2006) for finding the tracks in a tensor image. This method was developed to generate fibers in a Tensor representation where crossing fibers occur. The graph search algorithm follows the primary eigenvector in non-ambiguous regions and utilizes branching and a graph search algorithm in ambiguous regions. Ambiguous tracking regions are defined based on two criteria: Branching Al Threshold (anisotropy values below this value and above the tracking threshold) and Curvature Major Eigen (angles of the primary eigenvector direction and the current tracking direction). In regions that meet this criteria, two or three tracking paths are considered. The first is the standard primary eigenvector direction. The second is the secondary eigenvector direction. This is based on the assumption that these regions may be prolate regions. If the Random Walk option is selected then a third direction is also considered. This direction is defined by a cone pointing from the current position to the centroid of the ending region. The interior angle of the cone is specified by the user with the Branch/Guide Angle parameter. A vector contained inside of the cone is selected at random and used as the third direction. This method can also utilize the TEND option where the primary tracking direction is that specified by the TEND method instead of the primary eigenvector. The parameter ‘–maximumBranchPoints’ allows the tracking to have this number of branches being considered at a time. If this number of branch points is exceeded at any time, then the algorithm will revert back to a streamline algorithm until the number of branches is reduced. This allows the user to constrain the computational complexity of the algorithm. (4) The second phase of the GTRACT algorithm is Guided Tracking. This method incorporates anatomical information about the track orientation using an initial guess of the fiber track. In the originally proposed GTRACT method, this would be created from the fibers resulting from the Graph Search tracking. However, in practice this can be created using any method and could be defined manually. To create the guide fiber the program gtractCreateGuideFiber can be used. This program will load a fiber tract that has been generated and create a centerline representation of the fiber tract (i.e. a single fiber). In this method, the fiber tracking follows the primary eigenvector direction unless it deviates from the guide fiber track by a angle greater than that specified by the ‘–guidedCurvatureThreshold’ parameter.
The user must specify the guide fiber when running this program.

version: 4.0.0
license: http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt
contributor: This tool was developed by Vincent Magnotta, Greg Harris and Yongqiang Zhao.
acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

```python
Optional
inputTensorVolume: (a pathlike object or string representing an existing file)
    Required (for Free, Streamline, GraphSearch, and Guided fiber tracking methods): input tensor image file name
    argument: `--inputTensorVolume %s`

inputAnisotropyVolume: (a pathlike object or string representing an existing file)
    Required (for Free, Streamline, GraphSearch, and Guided fiber tracking methods): input anisotropy image file name
    argument: `--inputAnisotropyVolume %s`

inputStartingSeedsLabelMapVolume: (a pathlike object or string representing an existing file)
    Required (for Free, Streamline, GraphSearch, and Guided fiber tracking methods): input starting seeds LabelMap image file name
    argument: `--inputStartingSeedsLabelMapVolume %s`

startingSeedsLabel: (an integer (int or long))
    Label value for Starting Seeds (required if Label number used to create seed point in Slicer was not 1)
    argument: `--startingSeedsLabel %d`

inputEndingSeedsLabelMapVolume: (a pathlike object or string representing an existing file)
    Required (for Streamline, GraphSearch, and Guided fiber tracking methods): input ending seeds LabelMap image file name
    argument: `--inputEndingSeedsLabelMapVolume %s`

endingSeedsLabel: (an integer (int or long))
    Label value for Ending Seeds (required if Label number used to create seed point in Slicer was not 1)
    argument: `--endingSeedsLabel %d`

inputTract: (a pathlike object or string representing an existing file)
    Required (for Guided fiber tracking method): guide fiber in vtkPolydata file containing one tract line.
    argument: `--inputTract %s`

outputTract: (a boolean or a pathlike object or string representing a file)
    Required (for Free, Streamline, GraphSearch, and Guided fiber tracking methods): name of output vtkPolydata file containing tract lines and the point data collected along them.
    argument: `--outputTract %s`

writeXMLPolyDataFile: (a boolean)
    Flag to make use of the XML format for vtkPolyData fiber tracts.
    argument: `--writeXMLPolyDataFile`

trackingMethod: ('Guided' or 'Free' or 'Streamline' or 'GraphSearch')
    Fiber tracking Filter Type: Guided|Free|Streamline|GraphSearch
    argument: `--trackingMethod %s`

guidedCurvatureThreshold: (a float)
    Guided Curvature Threshold (Degrees)
```

(continues on next page)
argument: `--guidedCurvatureThreshold %f`
maximumGuideDistance: (a float)
    Maximum distance for using the guide fiber direction
    argument: `--maximumGuideDistance %f`
seedThreshold: (a float)
    Anisotropy threshold for seed selection (recommended for Free fiber tracking)
    argument: `--seedThreshold %f`
trackingThreshold: (a float)
    Anisotropy threshold for fiber tracking (anisotropy values of the next point along the path)
    argument: `--trackingThreshold %f`
curvatureThreshold: (a float)
    Curvature threshold in degrees (recommended for Free fiber tracking)
    argument: `--curvatureThreshold %f`
branchingThreshold: (a float)
    Anisotropy Branching threshold (recommended for GraphSearch fiber tracking method)
    argument: `--branchingThreshold %f`
maximumBranchPoints: (an integer (int or long))
    Maximum branch points (recommended for GraphSearch fiber tracking method)
    argument: `--maximumBranchPoints %d`
useRandomWalk: (a boolean)
    Flag to use random walk.
    argument: `--useRandomWalk`
randomSeed: (an integer (int or long))
    Random number generator seed
    argument: `--randomSeed %d`
branchingAngle: (a float)
    Branching angle in degrees (recommended for GraphSearch fiber tracking method)
    argument: `--branchingAngle %f`
minimumLength: (a float)
    Minimum fiber length. Helpful for filtering invalid tracts.
    argument: `--minimumLength %f`
maximumLength: (a float)
    Maximum fiber length (voxels)
    argument: `--maximumLength %f`
stepSize: (a float)
    Fiber tracking step size
    argument: `--stepSize %f`
useLoopDetection: (a boolean)
    Flag to make use of loop detection.
    argument: `--useLoopDetection`
useTend: (a boolean)
    Flag to make use of Tend F and Tend G parameters.
    argument: `--useTend`
tendF: (a float)
    Tend F parameter
    argument: `--tendF %f`
tendG: (a float)
    Tend G parameter
    argument: `--tendG %f`
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: `--numberOfThreads %d`
args: (a unicode string)
    Additional parameters to the command
    argument: `--args %s`
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})  
    Environment variables

Outputs:

outputTract: (a pathlike object or string representing an existing
    file)
    Required (for Free, Streamline, GraphSearch, and Guided fiber
    tracking methods): name of output vtkPolydata file containing tract
    lines and the point data collected along them.

75.6.14 gtractImageConformity

Link to code

Wraps the executable command `gtractImageConformity`.  

title: Image Conformity

category: Diffusion.GTRACT

description: This program will straighten out the Direction and Origin to match the Reference Image.

version: 4.0.0


license: http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt

contributor: This tool was developed by Vincent Magnotta and Greg Harris.

acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS
    R01NS050568-01A2S1

Inputs:

[Optional]

inputVolume: (a pathlike object or string representing an existing
    file)
    Required: input file containing the signed short image to reorient
    without resampling.
    argument: `--inputVolume %s`

inputReferenceVolume: (a pathlike object or string representing an
    existing file)
    Required: input file containing the standard image to clone the
    characteristics of.
    argument: `--inputReferenceVolume %s`

outputVolume: (a boolean or a pathlike object or string representing
    a file)
    Required: name of output Nrrd or Nifti file containing the
    reoriented image in reference image space.
    argument: `--outputVolume %s`

numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: `--numberOfThreads %d`

args: (a unicode string)
    Additional parameters to the command
    argument: `--args %s`

environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})
Environment variables

Outputs:

title: B-Spline Transform Inversion
category: Diffusion.GTRACT
description: This program will invert a B-Spline transform using a thin-plate spline approximation.
version: 4.0.0
license: http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt
contributor: This tool was developed by Vincent Magnotta and Greg Harris.
acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

[Optional]

Output arguments:

inputReferenceVolume: (a pathlike object or string representing an existing file)
    Required: input image file name to exemplify the anatomical space to interpolate over.
    argument: "--inputReferenceVolume %s"

inputTransform: (a pathlike object or string representing an existing file)
    Required: input B-Spline transform file name
    argument: "--inputTransform %s"

outputTransform: (a boolean or a pathlike object or string representing a file)
    Required: output transform file name
    argument: "--outputTransform %s"

landmarkDensity: (a list of items which are an integer (int or long))
    Number of landmark subdivisions in all 3 directions
    argument: "--landmarkDensity %s"

numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: "--numberOfThreads %d"

args: (a unicode string)
    Additional parameters to the command
    argument: "%s"

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
75.6.16 gtractInvertDisplacementField

Link to code
Wraps the executable command "gtractInvertDisplacementField".

title: Invert Displacement Field
category: Diffusion.GTRACT
description: This program will invert a deformation field. The size of the deformation field is defined by an example image provided by the user
version: 4.0.0
license: http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt
contributor: This tool was developed by Vincent Magnotta.

acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

<table>
<thead>
<tr>
<th>Optional</th>
<th>baseImage: (a pathlike object or string representing an existing file)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Required: base image used to define the size of the inverse field</td>
</tr>
<tr>
<td></td>
<td>argument: '--baseImage %s'</td>
</tr>
<tr>
<td></td>
<td>deformationImage: (a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td></td>
<td>Required: Displacement field image</td>
</tr>
<tr>
<td></td>
<td>argument: '--deformationImage %s'</td>
</tr>
<tr>
<td></td>
<td>outputVolume: (a boolean or a pathlike object or string representing a file)</td>
</tr>
<tr>
<td></td>
<td>Required: Output deformation field</td>
</tr>
<tr>
<td></td>
<td>argument: '--outputVolume %s'</td>
</tr>
<tr>
<td></td>
<td>subsamplingFactor: (an integer (int or long))</td>
</tr>
<tr>
<td></td>
<td>Subsampling factor for the deformation field</td>
</tr>
<tr>
<td></td>
<td>argument: '--subsamplingFactor %d'</td>
</tr>
<tr>
<td></td>
<td>numberOfThreads: (an integer (int or long))</td>
</tr>
<tr>
<td></td>
<td>Explicitly specify the maximum number of threads to use.</td>
</tr>
<tr>
<td></td>
<td>argument: '--numberOfThreads %d'</td>
</tr>
<tr>
<td></td>
<td>args: (a unicode string)</td>
</tr>
<tr>
<td></td>
<td>Additional parameters to the command</td>
</tr>
<tr>
<td></td>
<td>argument: '%s'</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optional</th>
<th>environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Environment variables</td>
</tr>
</tbody>
</table>

Outputs:

<table>
<thead>
<tr>
<th>Required</th>
<th>outputVolume: (a pathlike object or string representing an existing file)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Required: Output deformation field</td>
</tr>
</tbody>
</table>

75.6.17 gtractInvertRigidTransform

Link to code
Wraps the executable command "gtractInvertRigidTransform".
title: Rigid Transform Inversion
category: Diffusion.GTRACT
description: This program will invert a Rigid transform.
version: 4.0.0
license: http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt
contributor: This tool was developed by Vincent Magnotta and Greg Harris.
acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

[Optional]
inputTransform: (a pathlike object or string representing an existing file)
   Required: input rigid transform file name
   argument: '--inputTransform %s'
outputTransform: (a boolean or a pathlike object or string representing a file)
   Required: output transform file name
   argument: '--outputTransform %s'
numberOfThreads: (an integer (int or long))
   Explicitly specify the maximum number of threads to use.
   argument: '--numberOfThreads %d'
args: (a unicode string)
   Additional parameters to the command
   argument: '%s'
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
   Environment variables

Outputs:

outputTransform: (a pathlike object or string representing an existing file)
   Required: output transform file name

75.6.18 gtractResampleAnisotropy

Link to code
Wraps the executable command ‘' gtractResampleAnisotropy ''. title: Resample Anisotropy
category: Diffusion.GTRACT
description: This program will resample a floating point image using either the Rigid or B-Spline transform. You may want to save the aligned B0 image after each of the anisotropy map co-registration steps with the anatomical image to check the registration quality with another tool.
version: 4.0.0
license: http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt
contributor: This tool was developed by Vincent Magnotta and Greg Harris.
acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

[Optional]
inputAnisotropyVolume: (a pathlike object or string representing an
   (continues on next page)
existing file)
Required: input file containing the anisotropy image
argument: `--inputAnisotropyVolume %s`

inputAnatomicalVolume: (a pathlike object or string representing an existing file)
Required: input file containing the anatomical image whose characteristics will be cloned.
argument: `--inputAnatomicalVolume %s`

inputTransform: (a pathlike object or string representing an existing file)
Required: input Rigid OR Bspline transform file name
argument: `--inputTransform %s`

transformType: ('Rigid' or 'B-Spline')
Transform type: Rigid, B-Spline
argument: `--transformType %s`

outputVolume: (a boolean or a pathlike object or string representing a file)
Required: name of output NRRD file containing the resampled transformed anisotropy image.
argument: `--outputVolume %s`

numberOfThreads: (an integer (int or long))
Explicitly specify the maximum number of threads to use.
argument: `--numberOfThreads %d`

args: (a unicode string)
Additional parameters to the command
argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing file)
Required: name of output NRRD file containing the resampled transformed anisotropy image.

75.6.19 gtractResampleB0

Link to code
Wraps the executable command `gtractResampleB0`.

title: Resample B0
category: Diffusion.GTRACT
description: This program will resample a signed short image using either a Rigid or B-Spline transform. The user must specify a template image that will be used to define the origin, orientation, spacing, and size of the resampled image.

version: 4.0.0
license: http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt
contributor: This tool was developed by Vincent Magnotta and Greg Harris.
acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:
inputVolume: (a pathlike object or string representing an existing file)
   Required: input file containing the 4D image
   argument: `--inputVolume %s`

inputAnatomicalVolume: (a pathlike object or string representing an existing file)
   Required: input file containing the anatomical image defining the origin, spacing and size of the resampled image (template)
   argument: `--inputAnatomicalVolume %s`

inputTransform: (a pathlike object or string representing an existing file)
   Required: input Rigid OR Bspline transform file name
   argument: `--inputTransform %s`

vectorIndex: (an integer (int or long))
   Index in the diffusion weighted image set for the B0 image
   argument: `--vectorIndex %d`

transformType: ('Rigid' or 'B-Spline')
   Transform type: Rigid, B-Spline
   argument: `--transformType %s`

outputVolume: (a boolean or a pathlike object or string representing a file)
   Required: name of output NRRD file containing the resampled input image.
   argument: `--outputVolume %s`

numberOfThreads: (an integer (int or long))
   Explicitly specify the maximum number of threads to use.
   argument: `--numberOfThreads %d`

args: (a unicode string)
   Additional parameters to the command
   argument: `%%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Optional

inputCodeVolume: (a pathlike object or string representing an existing file)
   Required: input file containing the code image
   argument: `--inputCodeVolume %s`

inputReferenceVolume: (a pathlike object or string representing an existing file)
   Required: input file containing the standard image to clone the characteristics of.
   argument: `--inputReferenceVolume %s`

inputTransform: (a pathlike object or string representing an existing file)
   Required: input Rigid or Inverse-B-Spline transform file name
   argument: `--inputTransform %s`

transformType: ('Rigid' or 'Affine' or 'B-Spline' or 'Inverse-B-Spline' or 'None')
   Transform type: Rigid or Inverse-B-Spline
   argument: `--transformType %s`

outputVolume: (a boolean or a pathlike object or string representing a file)
   Required: name of output NRRD file containing the resampled code image in acquisition space.
   argument: `--outputVolume %s`

numberOfThreads: (an integer (int or long))
   Explicitly specify the maximum number of threads to use.
   argument: `--numberOfThreads %d`

args: (a unicode string)
   Additional parameters to the command
   argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
   Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing file)
   Required: name of output NRRD file containing the resampled code image in acquisition space.

75.6.21 gtractResampleDWIInPlace

Link to code
Wraps the executable command "gtractResampleDWIInPlace".

title: Resample DWI In Place
category: Diffusion.GTRACT
description: Resamples DWI image to structural image.
version: 4.0.0
license: http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt
contributor: This tool was developed by Vincent Magnotta, Greg Harris, Hans Johnson, and Joy Matsui.
acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:
[Optional]

**inputVolume**: (a pathlike object or string representing an existing file)

Required: input image is a 4D NRRD image.
argument: `--inputVolume %s`

**referenceVolume**: (a pathlike object or string representing an existing file)

If provided, resample to the final space of the referenceVolume 3D data set.
argument: `--referenceVolume %s`

**outputResampledB0**: (a boolean or a pathlike object or string representing a file)

Convenience function for extracting the first index location (assumed to be the B0)
argument: `--outputResampledB0 %s`

**inputTransform**: (a pathlike object or string representing an existing file)

Required: transform file derived from rigid registration of b0 image to reference structural image.
argument: `--inputTransform %s`

**warpDWITransform**: (a pathlike object or string representing an existing file)

Optional: transform file to warp gradient volumes.
argument: `--warpDWITransform %s`

**debugLevel**: (an integer (int or long))

Display debug messages, and produce debug intermediate results.
0=OFF, 1=Minimal, 10=Maximum debugging.
argument: `--debugLevel %d`

**imageOutputSize**: (a list of items which are an integer (int or long))

The voxel lattice for the output image, padding is added if necessary. NOTE: if 0,0,0, then the inputVolume size is used.
argument: `--imageOutputSize %s`

**outputVolume**: (a boolean or a pathlike object or string representing a file)

Required: output image (NRRD file) that has been rigidly transformed into the space of the structural image and padded if image padding was changed from 0,0,0 default.
argument: `--outputVolume %s`

**numberOfThreads**: (an integer (int or long))

Explicitly specify the maximum number of threads to use.
argument: `--numberOfThreads %d`

**args**: (a unicode string)

Additional parameters to the command
argument: `%s`

**environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

Environment variables

**Outputs:**

**outputResampledB0**: (a pathlike object or string representing an existing file)

Convenience function for extracting the first index location (assumed to be the B0)

**outputVolume**: (a pathlike object or string representing an existing file)

Required: output image (NRRD file) that has been rigidly transformed into the space of the structural image and padded if image padding was changed from 0,0,0 default.
into the space of the structural image and padded if image padding was changed from 0,0,0 default.

### 75.6.22 gtractResampleFibers

**Link to code**
Wraps the executable command `'gtractResampleFibers`.

title: Resample Fibers
category: Diffusion.GTRACT
description: This program will resample a fiber tract with respect to a pair of deformation fields that represent the forward and reverse deformation fields.

version: 4.0.0
license: http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt
contributor: This tool was developed by Vincent Magnotta and Greg Harris.

acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

**Inputs:**

```python
[Optional]
inputForwardDeformationFieldVolume: (a pathlike object or string representing an existing file)
    Required: input forward deformation field image file name
    argument: `--inputForwardDeformationFieldVolume %s`
inputReverseDeformationFieldVolume: (a pathlike object or string representing an existing file)
    Required: input reverse deformation field image file name
    argument: `--inputReverseDeformationFieldVolume %s`
inputTract: (a pathlike object or string representing an existing file)
    Required: name of input vtkPolydata file containing tract lines.
    argument: `--inputTract %s`
outputTract: (a boolean or a pathlike object or string representing a file)
    Required: name of output vtkPolydata file containing tract lines and the point data collected along them.
    argument: `--outputTract %s`
writeXMLPolyDataFile: (a boolean)
    Flag to make use of the XML format for vtkPolyData fiber tracts.
    argument: `--writeXMLPolyDataFile`'`
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: `--numberOfThreads %d`
args: (a unicode string)
    Additional parameters to the command
    argument: `%%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
```

**Outputs:**

```python
outputTract: (a pathlike object or string representing an existing file)
```

(continues on next page)
**75.6.23 gtractTensor**

**Link to code**
Wraps the executable command `gtractTensor`.

title: Tensor Estimation
category: Diffusion.GTRACT
description: This step will convert a b-value averaged diffusion tensor image to a 3x3 tensor voxel image. This step takes the diffusion tensor image data and generates a tensor representation of the data based on the signal intensity decay, b values applied, and the diffusion directions. The apparent diffusion coefficient for a given orientation is computed on a pixel-by-pixel basis by fitting the image data (voxel intensities) to the Stejskal-Tanner equation. If at least 6 diffusion directions are used, then the diffusion tensor can be computed. This program uses itk::DiffusionTensor3DReconstructionImageFilter. The user can adjust background threshold, median filter, and isotropic resampling.

version: 4.0.0
license: [http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt](http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt)
contributor: This tool was developed by Vincent Magnotta and Greg Harris.

acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

**Inputs:**

- **inputVolume**: (a pathlike object or string representing an existing file)
  - Required: input image 4D NRRD image. Must contain data based on at least 6 distinct diffusion directions. The inputVolume is allowed to have multiple b0 and gradient direction images. Averaging of the b0 image is done internally in this step. Prior averaging of the DWIs is not required.
  - argument: ``--inputVolume %s``

- **outputVolume**: (a boolean or a pathlike object or string representing a file)
  - Required: name of output NRRD file containing the Tensor vector image
  - argument: ``--outputVolume %s``

- **medianFilterSize**: (a list of items which are an integer (int or long))
  - Median filter radius in all 3 directions
  - argument: ``--medianFilterSize %s``

- **maskProcessingMode**: ('NOMASK' or 'ROIAUTO' or 'ROI')
  - ROIAUTO: mask is implicitly defined using a otsu foreground and hole filling algorithm. ROI: Uses the masks to define what parts of the image should be used for computing the transform. NOMASK: no mask used
  - argument: ``--maskProcessingMode %s``

- **maskVolume**: (a pathlike object or string representing an existing file)
  - Mask Image, if maskProcessingMode is ROI
  - argument: ``--maskVolume %s``

- **backgroundSuppressingThreshold**: (an integer (int or long))
  - Image threshold to suppress background. This sets a threshold used
on the b0 image to remove background voxels from processing. Typically, values of 100 and 500 work well for Siemens and GE DTI data, respectively. Check your data particularly in the globus pallidus to make sure the brain tissue is not being eliminated with this threshold.

arg: `--backgroundSuppressingThreshold %d`

resampleIsotropic: (a boolean)
Flag to resample to isotropic voxels. Enabling this feature is recommended if fiber tracking will be performed.

arg: `--resampleIsotropic`

size: (a float)
Isotropic voxel size to resample to

arg: `--size %f`

b0Index: (an integer (int or long))
Index in input vector index to extract

arg: `--b0Index %d`

applyMeasurementFrame: (a boolean)
Flag to apply the measurement frame to the gradient directions

arg: `--applyMeasurementFrame`

ignoreIndex: (a list of items which are an integer (int or long))
Ignore diffusion gradient index. Used to remove specific gradient directions with artifacts.

arg: `--ignoreIndex %s`

numberOfThreads: (an integer (int or long))
Explicitly specify the maximum number of threads to use.

arg: `--numberOfThreads %d`

args: (a unicode string)
Additional parameters to the command

arg: `%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing file)
Required: name of output NRRD file containing the Tensor vector image

75.6.24 gtractTransformToDisplacementField

Link to code
Wraps the executable command '" gtractTransformToDisplacementField '".

title: Create Displacement Field
category: Diffusion.GTRACT
description: This program will compute forward deformation from the given Transform. The size of the DF is equal to MNI space

version: 4.0.0
license: http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt
contributor: This tool was developed by Vincent Magnotta, Madhura Inghalikar, and Greg Harris
acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:
[Optional]
inputTransform: (a pathlike object or string representing an existing file)
    Input Transform File Name
    argument: `'--inputTransform %s'`
inputReferenceVolume: (a pathlike object or string representing an existing file)
    Required: input image file name to exemplify the anatomical space over which to vcl_express the transform as a displacement field.
    argument: `'--inputReferenceVolume %s'`
outputDeformationFieldVolume: (a pathlike object or string representing a file)
    Output deformation field
    argument: `'--outputDeformationFieldVolume %s'`
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: `'--numberOfThreads %d'`
args: (a unicode string)
    Additional parameters to the command
    argument: `'%s'`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})  
Environment variables

Outputs:

outputDeformationFieldVolume: (a pathlike object or string representing an existing file)
    Output deformation field

75.7 interfaces.semtools.diffusion.maxcurvature

75.7.1 maxcurvature

Link to code
Wraps the executable command `'maxcurvature`.

title: MaxCurvature-Hessian (DTIProcess)
category: Diffusion
description: This program computes the Hessian of the FA image (`image`). We use this scalar image as a registration input when doing DTI atlas building. For most adult FA we use a sigma of 2 whereas for neonate or primate images and sigma of 1 or 1.5 is more appropriate. For really noisy images, 2.5 - 4 can be considered. The final image (---output) shows the main feature of the input image.

version: 1.1.0
license: Copyright (c) Casey Goodlett. All rights reserved.
            See http://www.ia.unc.edu/dev/Copyright.htm for details. This software is distributed WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the above copyright notices for more information.
contributor: Casey Goodlett
acknowledgements: Hans Johnson(1,3,4); Kent Williams(1); (1=University of Iowa Department of Psychiatry, 3=University of Iowa Department of Biomedical Engineering, 4=University of Iowa Department of Electrical and Computer Engineering) provided conversions to make DTIProcess compatible with Slicer execution, and simplified the stand-alone build requirements by removing the dependancies on boost and a fortran compiler.
Inputs:
image: (a pathlike object or string representing an existing file)
    FA Image
    argument: `--image %s`
output: (a boolean or a pathlike object or string representing a file)
    Output File
    argument: `--output %s`
sigma: (a float)
    Scale of Gradients
    argument: `--sigma %f`
verbose: (a boolean)
    produce verbose output
    argument: `--verbose`
args: (a unicode string)
    Additional parameters to the command
    argument: `%%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
    Environment variables

Outputs:

output: (a pathlike object or string representing an existing file)
    Output File

75.8 interfaces.semtools.diffusion.tractography.commandlineonly

75.8.1 fiberstats

Link to code
Wraps the executable command "fiberstats".

title: FiberStats (DTIProcess)
category: Diffusion.Tractography.CommandLineOnly
description: Obsolete tool - Not used anymore
version: 1.1.0
license: Copyright (c) Casey Goodlett. All rights reserved.
    See http://www.ia.unc.edu/dev/Copyright.htm for details. This software is distributed WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the above copyright notices for more information.
contributor: Casey Goodlett
acknowledgements: Hans Johnson(1,3,4); Kent Williams(1); (1=University of Iowa Department of Psychiatry, 3=University of Iowa Department of Biomedical Engineering, 4=University of Iowa Department of Electrical and Computer Engineering) provided conversions to make DTIProcess compatible with Slicer execution, and simplified the stand-alone build requirements by removing the dependancies on boost and a fortran compiler.

Inputs:

fiber_file: (a pathlike object or string representing an existing file)
    DTI Fiber File
    argument: `--fiber_file %s`
verbose: (a boolean)
produce verbose output
argument: ``--verbose``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {}) 
Environment variables

Outputs:
None

75.9 interfaces.semtools.diffusion.tractography.fiberprocess

75.9.1 fiberprocess

Link to code
Wraps the executable command `fiberprocess`.

title: FiberProcess (DTIProcess)
category: Diffusion.Tractography
description: fiberprocess is a tool that manage fiber files extracted from the fibertrack tool or any fiber tracking algorithm. It takes as an input .fib and .vtk files (`fiber_file`) and saves the changed fibers (`fiber_output`) into the 2 same formats. The main purpose of this tool is to deform the fiber file with a transformation field as an input (`displacement_field` or `h_field` depending if you deal with dfield or hfield). To use that option you need to specify the tensor field from which the fiber file was extracted with the option `tensor_volume`. The transformation applied on the fiber file is the inverse of the one input. If the transformation is from one case to an atlas, fiberprocess assumes that the fiber file is in the atlas space and you want it in the original case space, so it’s the inverse of the transformation which has been computed. You have 2 options for fiber modification. You can either deform the fibers (their geometry) into the space OR you can keep the same geometry but map the diffusion properties (fa, md, lbd’s…) of the original tensor field along the fibers at the corresponding locations. This is triggered by the `--no_warp` option. To use the previous example: when you have a tensor field in the original space and the deformed tensor field in the atlas space, you want to track the fibers in the atlas space, keeping this geometry but with the original case diffusion properties. Then you can specify the transformations field (from original case -> atlas) and the original tensor field with the `tensor_volume` option. With fiberprocess you can also binarize a fiber file. Using the `voxelize` option will create an image where each voxel through which a fiber is passing is set to 1. The output is going to be a binary image with the values 0 or 1 by default but the 1 value voxel can be set to any number with the `voxel_label` option. Finally you can create an image where the value at the voxel is the number of fiber passing through. (`voxelize_count_fibers`)

version: 1.0.0
license: Copyright (c) Casey Goodlett. All rights reserved. See http://www.ia.unc.edu/... for details. This software is distributed WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the above copyright notices for more information.
contributor: Casey Goodlett

Inputs:

[Optional]
fiber_file: (a pathlike object or string representing an existing file)
    DTI fiber file

(continues on next page)
argument: `--fiber_file %s`
fiber_output: (a boolean or a pathlike object or string representing
a file)
Output fiber file. May be warped or updated with new data depending
on other options used.
argument: `--fiber_output %s`
tensor_volume: (a pathlike object or string representing an existing
file)
Interpolate tensor values from the given field
argument: `--tensor_volume %s`
h_field: (a pathlike object or string representing an existing file)
HField for warp and statistics lookup. If this option is used
tensor-volume must also be specified.
argument: `--h_field %s`
displacement_field: (a pathlike object or string representing an
existing file)
Displacement Field for warp and statistics lookup. If this option is
used tensor-volume must also be specified.
argument: `--displacement_field %s`
saveProperties: (a boolean)
save the tensor property as scalar data into the vtk (only works for
vtk fiber files).
argument: `--saveProperties`
no_warp: (a boolean)
Do not warp the geometry of the tensors only obtain the new
statistics.
argument: `--no_warp`
fiber_radius: (a float)
set radius of all fibers to this value
argument: `--fiber_radius %f`
index_space: (a boolean)
Use index-space for fiber output coordinates, otherwise us world
space for fiber output coordinates (from tensor file).
argument: `--index_space`
voxelize: (a boolean or a pathlike object or string representing a
file)
Voxelize fiber into a label map (the labelmap filename is the
argument of -V). The tensor file must be specified using -T for
information about the size, origin, spacing of the image. The
deformation is applied before the voxelization
argument: `--voxelize %s`
voxelize_count_fibers: (a boolean)
Count number of fibers per-voxel instead of just setting to 1
argument: `--voxelize_count_fibers`
voxel_label: (an integer (int or long))
Label for voxelized fiber
argument: `--voxel_label %d`
verbose: (a boolean)
produce verbose output
argument: `--verbose`
noDataChange: (a boolean)
Do not change data ???
argument: `--noDataChange`
args: (a unicode string)
Additional parameters to the command
argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value
(continues on next page)
Environment variables

Outputs:

fiber_output: (a pathlike object or string representing an existing file)
Output fiber file. May be warped or updated with new data depending on other options used.

voxelize: (a pathlike object or string representing an existing file)
Voxelize fiber into a label map (the labelmap filename is the argument of -V). The tensor file must be specified using -T for information about the size, origin, spacing of the image. The deformation is applied before the voxelization.

75.10 interfaces.semtools.diffusion.tractography.fibertrack

75.10.1 fibertrack

Link to code
Wraps the executable command ‘fibertrack’.

Title: FiberTrack (DTIProcess)
Category: Diffusion.Tractography
Description: This program implements a simple streamline tractography method based on the principal eigenvector of the tensor field. A fourth order Runge-Kutta integration rule used to advance the streamlines. As a first parameter you have to input the tensor field (with the -input_tensor_file option). Then the region of interest image file is set with the -input_roi_file. Next you want to set the output fiber file name after the -output_fiber_file option. You can specify the label value in the input_roi_file with the -target_label, -source_label and -forbidden_label options. By default target label is 1, source label is 2 and forbidden label is 0. The source label is where the streamlines are seeded, the target label defines the voxels through which the fibers must pass by to be kept in the final fiber file and the forbidden label defines the voxels where the streamlines are stopped if they pass through it. There is also a -whole_brain option which, if enabled, consider both target and source labels of the roi image as target labels and all the voxels of the image are considered as sources. During the tractography, the -fa_min parameter is used as the minimum value needed at different voxel for the tracking to keep going along a streamline. The -step_size parameter is used for each iteration of the tracking algorithm and defines the length of each step. The -max_angle option defines the maximum angle allowed between two successive segments along the tracked fiber.

Version: 1.1.0

License: Copyright (c) Casey Goodlett. All rights reserved.

See http://www.ia.unc.edu/dev/Copyright.htm for details. This software is distributed WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the above copyright notices for more information.

Contributor: Casey Goodlett
Acknowledgements: Hans Johnson(1,3,4); Kent Williams(1); (1=University of Iowa Department of Psychiatry, 3=University of Iowa Department of Biomedical Engineering, 4=University of Iowa Department of Electrical and Computer Engineering) provided conversions to make DTIProcess compatible with Slicer execution, and simplified the stand-alone build requirements by removing the dependencies on boost and a fortran compiler.

Inputs:

[Optional]
input_tensor_file: (a pathlike object or string representing an existing file or string representing an existing file)
existing file)
Tensor Image
argument: `--input_tensor_file %s`

input_roi_file: (a pathlike object or string representing an existing file)
The filename of the image which contains the labels used for seeding and constraining the algorithm.
argument: `--input_roi_file %s`

output_fiber_file: (a boolean or a pathlike object or string representing a file)
The filename for the fiber file produced by the algorithm. This file must end in a .fib or .vtk extension for ITK spatial object and vtkPolyData formats respectively.
argument: `--output_fiber_file %s`

source_label: (an integer (int or long))
The label of voxels in the labelfile to use for seeding tractography. One tract is seeded from the center of each voxel with this label
argument: `--source_label %d`

target_label: (an integer (int or long))
The label of voxels in the labelfile used to constrain tractography. Tracts that do not pass through a voxel with this label are rejected. Set this keep all tracts.
argument: `--target_label %d`

forbidden_label: (an integer (int or long))
Forbidden label
argument: `--forbidden_label %d`

whole_brain: (a boolean)
If this option is enabled all voxels in the image are used to seed tractography. When this option is enabled both source and target labels function as target labels
argument: `--whole_brain`

max_angle: (a float)
Maximum angle of change in radians
argument: `--max_angle %f`

step_size: (a float)
Step size in mm for the tracking algorithm
argument: `--step_size %f`

min_fa: (a float)
The minimum FA threshold to continue tractography
argument: `--min_fa %f`

force: (a boolean)
Ignore sanity checks.
argument: `--force`

verbose: (a boolean)
produce verbose output
argument: `--verbose`

really_verbose: (a boolean)
Follow detail of fiber tracking algorithm
argument: `--really_verbose`

args: (a unicode string)
Additional parameters to the command
argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Outputs:

```
output_fiber_file: (a pathlike object or string representing an existing file)
    The filename for the fiber file produced by the algorithm. This file must end in a .fib or .vtk extension for ITK spatial object and vtkPolyData formats respectively.
```

75.11 interfaces.semtools.diffusion.tractography.ukftractography

75.11.1 UKFTractography

Link to code
Wraps the executable command "UKFTractography".

title: UKF Tractography
category: Diffusion.Tractography
description: This module traces fibers in a DWI Volume using the multiple tensor unscented Kalman Filter methodology. For more informations check the documentation.

version: 1.0

contributor: Yogesh Rathi, Stefan Liebhard, Yinpeng Li, Martin Styner, Ipek Oguz, Yundi Shi, Christian Baumgartner, Kent Williams, Hans Johnson, Peter Savadjiev, Carl-Fredrik Westin.

acknowledgements: The development of this module was supported by NIH grants R01 MH097979 (PI Rathi), R01 MH092862 (PIs Westin and Verma), U01 NS083223 (PI Westin), R01 MH074794 (PI Westin) and P41 EB015902 (PI Kikinis).

Inputs:

```
[Optional]
dwiFile: (a pathlike object or string representing an existing file)
    Input DWI volume
    argument: `--dwiFile %s`

seedsFile: (a pathlike object or string representing an existing file)
    Seeds for diffusion. If not specified, full brain tractography will be performed, and the algorithm will start from every voxel in the brain mask where the Generalized Anisotropy is bigger than 0.18
    argument: `--seedsFile %s`

labels: (a list of items which are an integer (int or long))
    A vector of the ROI labels to be used
    argument: `--labels %s`

maskFile: (a pathlike object or string representing an existing file)
    Mask for diffusion tractography
    argument: `--maskFile %s`

tracts: (a boolean or a pathlike object or string representing a file)
    Tracts generated, with first tensor output
    argument: `--tracts %s`

writeAsciiTracts: (a boolean)
    Write tract file as a VTK binary data file
    argument: `--writeAsciiTracts`

writeUncompressedTracts: (a boolean)
    Write tract file as a VTK uncompressed data file
    argument: `--writeUncompressedTracts`

seedsPerVoxel: (an integer (int or long))
    Each seed generates a fiber, thus using more seeds generates more
```
fibers. In general use 1 or 2 seeds, and for a more thorough result use 5 or 10 (depending on your machine this may take up to 2 days to run).

```
--seedsPerVoxel %d
```

**numTensor:** ('1' or '2')
Number of tensors used
```
--numTensor %s
```

**freeWater:** (a boolean)
Adds a term for free water diffusion to the model. (Note for experts: if checked, the 1T simple model is forced)
```
--freeWater
```

**recordFA:** (a boolean)
Whether to store FA. Attaches field 'FA', and 'FA2' for 2-tensor case to fiber.
```
--recordFA
```

**recordFreeWater:** (a boolean)
Whether to store the fraction of free water. Attaches field 'FreeWater' to fiber.
```
--recordFreeWater
```

**recordTrace:** (a boolean)
Whether to store Trace. Attaches field 'Trace', and 'Trace2' for 2-tensor case to fiber.
```
--recordTrace
```

**recordTensors:** (a boolean)
Recording the tensors enables Slicer to color the fiber bundles by FA, orientation, and so on. The fields will be called 'TensorN', where N is the tensor number.
```
--recordTensors
```

**recordNMSE:** (a boolean)
Whether to store NMSE. Attaches field 'NMSE' to fiber.
```
--recordNMSE
```

**recordState:** (a boolean)
Whether to attach the states to the fiber. Will generate field 'state'.
```
--recordState
```

**recordCovariance:** (a boolean)
Whether to store the covariance. Will generate field 'covariance' in fiber.
```
--recordCovariance
```

**recordLength:** (a float)
Record length of tractography, in millimeters
```
--recordLength %f
```

**minFA:** (a float)
Abort the tractography when the Fractional Anisotropy is less than this value
```
--minFA %f
```

**minGA:** (a float)
Abort the tractography when the Generalized Anisotropy is less than this value
```
--minGA %f
```

**fullTensorModel:** (a boolean)
Whether to use the full tensor model. If unchecked, use the default simple tensor model
```
--fullTensorModel
```

**numThreads:** (an integer (int or long))
Number of threads used during computation. Set to the number of cores on your workstation for optimal speed. If left undefined the...
number of cores detected will be used.
argument: `--numThreads %d`

stepLength: (a float)
Step length of tractography, in millimeters
argument: `--stepLength %f`

maxHalfFiberLength: (a float)
The max length limit of the half fibers generated during tractography. Here the fiber is 'half' because the tractography goes in only one direction from one seed point at a time
argument: `--maxHalfFiberLength %f`

seedFALimit: (a float)
Seed points whose FA are below this value are excluded
argument: `--seedFALimit %f`

Qm: (a float)
Process noise for angles/direction
argument: `--Qm %f`

Ql: (a float)
Process noise for eigenvalues
argument: `--Ql %f`

Qw: (a float)
Process noise for free water weights, ignored if no free water estimation
argument: `--Qw %f`

Rs: (a float)
Measurement noise
argument: `--Rs %f`

maxBranchingAngle: (a float)
Maximum branching angle, in degrees. When using multiple tensors, a new branch will be created when the tensors' major directions form an angle between (minBranchingAngle, maxBranchingAngle). Branching is suppressed when this maxBranchingAngle is set to 0.0
argument: `--maxBranchingAngle %f`

minBranchingAngle: (a float)
Minimum branching angle, in degrees. When using multiple tensors, a new branch will be created when the tensors' major directions form an angle between (minBranchingAngle, maxBranchingAngle)
argument: `--minBranchingAngle %f`

tractsWithSecondTensor: (a boolean or a pathlike object or string representing a file)
Tracts generated, with second tensor output (if there is one)
argument: `--tractsWithSecondTensor %s`

storeGlyphs: (a boolean)
Store tensors' main directions as two-point lines in a separate file named glyphs_(tracts). When using multiple tensors, only the major tensors' main directions are stored
argument: `--storeGlyphs`

args: (a unicode string)
Additional parameters to the command
argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

tracts: (a pathlike object or string representing an existing file)
Tracts generated, with first tensor output
tractsWithSecondTensor: (a pathlike object or string representing an existing file)
Tracts generated, with second tensor output (if there is one)

75.12 interfaces.semtools.featurecreator

75.12.1 GenerateCsfClippedFromClassifiedImage

Link to code
Wraps the executable command ‘‘ GenerateCsfClippedFromClassifiedImage ‘‘.

title: GenerateCsfClippedFromClassifiedImage
category: FeatureCreator
description: Get the distance from a voxel to the nearest voxel of a given tissue type.
version: 0.1.0.$Revision: 1 $(alpha)
documentation-url: http://www.na-mic.org/
license: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt
contributor: This tool was written by Hans J. Johnson.

Inputs:

[Optional]
inputCassifiedVolume: (a pathlike object or string representing an existing file)
   Required: input tissue label image
   argument: ‘‘--inputCassifiedVolume %s’’
outputVolume: (a boolean or a pathlike object or string representing a file)
   Required: output image
   argument: ‘‘--outputVolume %s’’
args: (a unicode string)
   Additional parameters to the command
   argument: ‘‘%s’’
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
   Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing file)
   Required: output image

75.13 interfaces.semtools.filtering.denoising

75.13.1 UnbiasedNonLocalMeans

Link to code
Wraps the executable command ‘‘ UnbiasedNonLocalMeans ‘‘.

title: Unbiased NLM for MRI
category: Filtering.Denoising
description: This module implements a fast version of the popular Non-Local Means filter for image denoising. This algorithm computes the weights based on the similarity of each neighbor with the voxel to be denoised. In the original formulation a patch with a certain radius is centered in each of the voxels, and the Mean Squared Error between each pair of corresponding voxels is computed. In this implementation, only the
mean value and gradient components are compared. This, together with an efficient memory management, can attain a speed-up of nearly 20x. Besides, the filtering is more accurate than the original with poor SNR. This code is intended for its use with MRI (or any other Rician-distributed modality): the second order moment is estimated, then we subtract twice the squared power of noise, and finally we take the square root of the result to remove the Rician bias. The original implementation of the NLM filter may be found in: A. Buades, B. Coll, J. Morel, “A review of image denoising algorithms, with a new one”, Multiscale Modelling and Simulation 4(2): 490-530. 2005. The correction of the Rician bias is described in the following reference (among others): S. Aja-Fernandez, K. Krissian, “An unbiased Non-Local Means scheme for DWI filtering”, in: Proceedings of the MICCAI Workshop on Computational Diffusion MRI, 2008, pp. 277-284. The whole description of this version may be found in the following paper (please, cite it if you are willing to use this software): A. Tristan-Vega, V. Garcia Perez, S. Aja-Fenandez, and C.-F. Westin, “Efficient and Robust Nonlocal Means Denoising of MR Data Based on Salient Features Matching”, Computer Methods and Programs in Biomedicine. (Accepted for publication) 2011.

version: 0.0.1.$Revision: 1 $(beta)
contributor: Antonio Tristan Vega, Veronica Garcia-Perez, Santiago Aja-Fernandez, Carl-Fredrik Westin
acknowledgements: Supported by grant number FMECD-2010/71131616E from the Spanish Ministry of Education/Fulbright Committee
Inputs:

<table>
<thead>
<tr>
<th>Optional</th>
<th>sigma: (a float)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The root power of noise (sigma) in the complex Gaussian process the Rician comes from. If it is underestimated, the algorithm fails to remove the noise. If it is overestimated, over-blurring is likely to occur.</td>
</tr>
<tr>
<td></td>
<td>argument: <code>--sigma %f</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rs: (a list of items which are an integer (int or long))</th>
</tr>
</thead>
<tbody>
<tr>
<td>The algorithm search for similar voxels in a neighborhood of this radius (radii larger than 5,5,5 are very slow, and the results can be only marginally better. Small radii may fail to effectively remove the noise).</td>
</tr>
<tr>
<td>argument: <code>--rs %s</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rc: (a list of items which are an integer (int or long))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Similarity between blocks is computed as the difference between mean values and gradients. These parameters are computed fitting a hyperplane with LS inside a neighborhood of this size</td>
</tr>
<tr>
<td>argument: <code>--rc %s</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>hp: (a float)</th>
</tr>
</thead>
<tbody>
<tr>
<td>This parameter is related to noise; the larger the parameter, the more aggressive the filtering. Should be near 1, and only values between 0.8 and 1.2 are allowed</td>
</tr>
<tr>
<td>argument: <code>--hp %f</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ps: (a float)</th>
</tr>
</thead>
<tbody>
<tr>
<td>To accelerate computations, preselection is used: if the normalized difference is above this threshold, the voxel will be discarded (non used for average)</td>
</tr>
<tr>
<td>argument: <code>--ps %f</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>inputVolume: (a pathlike object or string representing an existing file)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input MRI volume.</td>
</tr>
<tr>
<td>argument: <code>%s</code>, position: -2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>outputVolume: (a boolean or a pathlike object or string representing a file)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output (filtered) MRI volume.</td>
</tr>
</tbody>
</table>

(continues on next page)
75.14 interfaces.semtools.filtering.featuredetection

75.14.1 CannyEdge

Link to code
Wraps the executable command `"CannyEdge"`

title: Canny Edge Detection
category: Filtering.FeatureDetection
description: Get the distance from a voxel to the nearest voxel of a given tissue type.
version: 0.1.0 (alpha)
documentation-url: http://www.na-mic.org/
license: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt
contributor: This tool was written by Hans J. Johnson.

Inputs:

```
[Optional]
inputVolume: (a pathlike object or string representing an existing file)
  Required: input tissue label image
  argument: `\"--inputVolume \s\"`

variance: (a float)
  Variance and Maximum error are used in the Gaussian smoothing of the input image. See itkDiscreteGaussianImageFilter for information on these parameters.
  argument: `\"--variance \f\"

upperThreshold: (a float)
  Threshold is the lowest allowed value in the output image. Its data type is the same as the data type of the output image. Any values below the Threshold level will be replaced with the OutsideValue parameter value, whose default is zero.
  argument: `\"--upperThreshold \f\"

lowerThreshold: (a float)
  Threshold is the lowest allowed value in the output image. Its data type is the same as the data type of the output image. Any values below the Threshold level will be replaced with the OutsideValue parameter value, whose default is zero.
  argument: `\"--lowerThreshold \f\"
```

Outputs:

```
outputVolume: (a pathlike object or string representing an existing file)
  Output (filtered) MRI volume.
```
argument: `--outputVolume %s`

args: (a unicode string)
Additional parameters to the command
argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing
file)
Required: output image

### 75.14.2 CannySegmentationLevelSetImageFilter

**Link to code**
Wraps the executable command `CannySegmentationLevelSetImageFilter`.

**title:** Canny Level Set Image Filter

**category:** Filtering.FeatureDetection

**description:** The CannySegmentationLevelSet is commonly used to refine a manually generated manual mask.

**version:** 0.3.0

**license:** CC

**contributor:** Regina Kim

**acknowledgements:** This command module was derived from Insight/Examples/Segmentation/CannySegmentationLevelSetImageFilter.cxx

**Inputs:**

[Optional]
inputVolume: (a pathlike object or string representing an existing
file)
argument: `--inputVolume %s`

initialModel: (a pathlike object or string representing an existing
file)
argument: `--initialModel %s`

outputVolume: (a boolean or a pathlike object or string representing
a file)
argument: `--outputVolume %s`

outputSpeedVolume: (a boolean or a pathlike object or string
representing a file)
argument: `--outputSpeedVolume %s`

cannyThreshold: (a float)
Canny Threshold Value
argument: `--cannyThreshold %f`
cannyVariance: (a float)
Canny variance
argument: `--cannyVariance %f`
advectionWeight: (a float)
Controls the smoothness of the resulting mask, small number are more
smooth, large numbers allow more sharp corners.
argument: `--advectionWeight %f`

initialModelIsovalue: (a float)
The identification of the input model iso-surface. (for a binary
image with 0s and 1s use 0.5) (for a binary image with 0s and 255's

(continues on next page)
75.14.3 DilateImage

Link to code

Wraps the executable command "DilateImage".

title: Dilate Image
category: Filtering.FeatureDetection
description: Uses mathematical morphology to dilate the input images.
version: 0.1.0.$Revision: 1 $(alpha)
documentation-url: http://www.na-mic.org/
license: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt
contributor: This tool was developed by Mark Scully and Jeremy Bockholt.

Inputs:

[Optional]
inputVolume: (a pathlike object or string representing an existing file)
  Required: input image
  argument: '--inputVolume %s'
inputMaskVolume: (a pathlike object or string representing an existing file)
  Required: input brain mask image
  argument: '--inputMaskVolume %s'
inputRadius: (an integer (int or long))
  Required: input neighborhood radius
  argument: '--inputRadius %d'
outputVolume: (a boolean or a pathlike object or string representing a file)
  Required: output image
  argument: '--outputVolume %s'
args: (a unicode string)
  Additional parameters to the command
  argument: '%s'
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing file)
outputSpeedVolume: (a pathlike object or string representing an existing file)
Outputs:

```
outputVolume: (a pathlike object or string representing an existing file)
    Required: output image
```

### 75.14.4 DilateMask

**Link to code**
Wraps the executable command `DilateMask`.

**title:** Dilate Image  
**category:** Filtering.FeatureDetection  
**description:** Uses mathematical morphology to dilate the input images.

**version:** 0.1.0.$Revision: 1 $(alpha)

**documentation-url:** [http://www.na-mic.org/](http://www.na-mic.org/)

**license:** [https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt](https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt)

**contributor:** This tool was developed by Mark Scully and Jeremy Bockholt.

**Inputs:**

```
[Optional]
inputVolume: (a pathlike object or string representing an existing file)
    Required: input image
    argument: '``--inputVolume %s``'

inputBinaryVolume: (a pathlike object or string representing an existing file)
    Required: input brain mask image
    argument: '``--inputBinaryVolume %s``'

sizeStructuralElement: (an integer (int or long))
    size of structural element. sizeStructuralElement=1 means that 3x3x3 structuring element for 3D
    argument: '``--sizeStructuralElement %d``'

lowerThreshold: (a float)
    Required: lowerThreshold value
    argument: '``--lowerThreshold %f``'

outputVolume: (a boolean or a pathlike object or string representing a file)
    Required: output image
    argument: '``--outputVolume %s``'

args: (a unicode string)
    Additional parameters to the command
    argument: '``%s``'

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})  
    Environment variables
```

### 75.14.5 DistanceMaps

**Link to code**
Wraps the executable command `DistanceMaps`.

**Inputs:**

```
outputVolume: (a pathlike object or string representing an existing file)
    Required: output image
```

75.14. interfaces.semtools.filtering.featuredetection
title: Mauerer Distance  
category: Filtering.FeatureDetection  
description: Get the distance from a voxel to the nearest voxel of a given tissue type.  
version: 0.1.0.$Revision: 1 $(alpha)  
documentation-url: http:://www.na-mic.org/  
license: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt  
contributor: This tool was developed by Mark Scully and Jeremy Bockholt.

Inputs:

[Optional]
inputLabelVolume: (a pathlike object or string representing an existing file)
  Required: input tissue label image
  argument: ``--inputLabelVolume %s``

inputMaskVolume: (a pathlike object or string representing an existing file)
  Required: input brain mask image
  argument: ``--inputMaskVolume %s``

inputTissueLabel: (an integer (int or long))
  Required: input integer value of tissue type used to calculate distance
  argument: ``--inputTissueLabel %d``

outputVolume: (a boolean or a pathlike object or string representing a file)
  Required: output image
  argument: ``--outputVolume %s``

args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing file)
  Required: output image

75.14.6 DumpBinaryTrainingVectors

Link to code

Wraps the executable command `` DumpBinaryTrainingVectors ''. 

title: Erode Image  
category: Filtering.FeatureDetection  
description: Uses mathematical morphology to erode the input images.  
version: 0.1.0.$Revision: 1 $(alpha)  
documentation-url: http:://www.na-mic.org/  
license: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt  
contributor: This tool was developed by Mark Scully and Jeremy Bockholt.

Inputs:

[Optional]
inputHeaderFilename: (a pathlike object or string representing an existing file)
  Required: input header file name

(continues on next page)
## 75.14.7 ErodeImage

**Link to code**
Wraps the executable command `"ErodeImage"`.

**title**: Erode Image  
**category**: Filtering.FeatureDetection  
**description**: Uses mathematical morphology to erode the input images.  
**version**: 0.1.0. $(Revision: 1 $(alpha))  
**documentation-url**: http://www.na-mic.org/  
**license**: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt  
**contributor**: This tool was developed by Mark Scully and Jeremy Bockholt.

### Inputs:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--inputVolume %s</code></td>
<td>Input image</td>
</tr>
<tr>
<td><code>--inputMaskVolume %s</code></td>
<td>Input brain mask image</td>
</tr>
<tr>
<td><code>--inputRadius %d</code></td>
<td>Input neighborhood radius</td>
</tr>
<tr>
<td><code>--outputVolume %s</code></td>
<td>Output image</td>
</tr>
</tbody>
</table>

### Outputs:

None
75.14.8 FlippedDifference

Link to code
Wraps the executable command “FlippedDifference”.
title: Flip Image
category: Filtering.FeatureDetection
description: Difference between an image and the axially flipped version of that image.
version: 0.1.0.$Revision: 1 $(alpha)
documentation-url: http://www.na-mic.org/
license: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt
contributor: This tool was developed by Mark Scully and Jeremy Bockholt.
Inputs:

<table>
<thead>
<tr>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outputVolume</td>
<td>(a pathlike object or string representing an existing file) Required: output image</td>
</tr>
<tr>
<td>inputVolume</td>
<td>(a pathlike object or string representing an existing file) Required: input image</td>
</tr>
<tr>
<td>inputMaskVolume</td>
<td>(a pathlike object or string representing an existing file) Required: input brain mask image</td>
</tr>
<tr>
<td>outputVolume</td>
<td>(a boolean or a pathlike object or string representing a file) Required: output image</td>
</tr>
<tr>
<td>args</td>
<td>(a unicode string) Additional parameters to the command argument: <code>%s</code></td>
</tr>
<tr>
<td>environ</td>
<td>(a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})</td>
</tr>
</tbody>
</table>

Outputs:

<table>
<thead>
<tr>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outputVolume</td>
<td>(a pathlike object or string representing an existing file) Required: output image</td>
</tr>
</tbody>
</table>

75.14.9 GenerateBrainClippedImage

Link to code
Wraps the executable command “GenerateBrainClippedImage”.
title: GenerateBrainClippedImage
category: Filtering.FeatureDetection
description: Automatic FeatureImages using neural networks
version: 1.0
license: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt
contributor: Eun Young Kim
Inputs:
### 75.14.10 GenerateSummedGradientImage

**Link to code**

Wraps the executable command ``GenerateSummedGradientImage``.

**title:** GenerateSummedGradient

**category:** Filtering.FeatureDetection

**description:** Automatic FeatureImages using neural networks

**version:** 1.0

**license:** [https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt](https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt)

**contributor:** Greg Harris, Eun Young Kim

**Inputs:**

```python
[Optional]
inputVolume1: (a pathlike object or string representing an existing
file)
    input volume 1, usually t1 image
    argument: '--inputVolume1 %s'
inputVolume2: (a pathlike object or string representing an existing
file)
    input volume 2, usually t2 image
    argument: '--inputVolume2 %s'
outputFileName: (a boolean or a pathlike object or string
    representing a file)
    (required) output file name
    argument: '--outputFileName %s'
maximumGradient: (a boolean)
    If set this flag, it will compute maximum gradient between two input
    volumes instead of sum of it.
    argument: '--MaximumGradient '
```

(continues on next page)
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: `--numberOfThreads %d`

args: (a unicode string)
    Additional parameters to the command
    argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})
    Environment variables

Outputs:

outputFileName: (a pathlike object or string representing an existing file)
    (required) output file name

75.14.11 GenerateTestImage

Link to code
Wraps the executable command `'GenerateTestImage`.

title: DownSampleImage
category: Filtering.FeatureDetection
description: Down sample image for testing
version: 1.0
license: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt
contributor: Eun Young Kim

Inputs:

[Optional]
inputVolume: (a pathlike object or string representing an existing file)
    input volume 1, usually t1 image
    argument: `--inputVolume %s`

outputVolume: (a boolean or a pathlike object or string representing a file)
    (required) output file name
    argument: `--outputVolume %s`

lowerBoundOfOutputVolume: (a float)
    argument: `--lowerBoundOfOutputVolume %f`

upperBoundOfOutputVolume: (a float)
    argument: `--upperBoundOfOutputVolume %f`

outputVolumeSize: (a float)
    output Volume Size
    argument: `--outputVolumeSize %f`

args: (a unicode string)
    Additional parameters to the command
    argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})
    Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing file)
75.14.12 GradientAnisotropicDiffusionImageFilter

Link to code
Wraps the executable command `GradientAnisotropicDiffusionImageFilter`.

title: GradientAnisotropicDiffusionFilter
category: Filtering.FeatureDetection
description: Image Smoothing using Gradient Anisotropic Diffusion Filter
ccontributor: This tool was developed by Eun Young Kim by modifying ITK Example

Inputs:

- **inputVolume**: (a pathlike object or string representing an existing file)
  - Required: input image
  - argument: `--inputVolume %s`
- **numberOfIterations**: (an integer (int or long))
  - Optional value for number of Iterations
  - argument: `--numberOfIterations %d`
- **timeStep**: (a float)
  - Time step for diffusion process
  - argument: `--timeStep %f`
- **conductance**: (a float)
  - Conductance for diffusion process
  - argument: `--conductance %f`
- **outputVolume**: (a boolean or a pathlike object or string representing a file)
  - Required: output image
  - argument: `--outputVolume %s`
- **args**: (a unicode string)
  - Additional parameters to the command
  - argument: `%s`
- **environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: `{}`)
  - Environment variables

Outputs:

- **outputVolume**: (a pathlike object or string representing an existing file)
  - Required: output image

75.14.13 HammerAttributeCreator

Link to code
Wraps the executable command `HammerAttributeCreator`.

title: HAMMER Feature Vectors
category: Filtering.FeatureDetection
description: Create the feature vectors used by HAMMER.

version: 0.1.0.$Revision: 1 $(alpha)
documentation-url: http:://www.na-mic.org/
license: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt
contributor: This was extracted from the Hammer Registration source code, and wrapped up by Hans J. Johnson.

Inputs:

```plaintext
[Optional]
Scale: (an integer (int or long))
   Determine Scale of Ball
   argument: `--Scale %d`
Strength: (a float)
   Determine Strength of Edges
   argument: `--Strength %f`
inputGMVolume: (a pathlike object or string representing an existing file)
   Required: input grey matter posterior image
   argument: `--inputGMVolume %s`
inputWMVolume: (a pathlike object or string representing an existing file)
   Required: input white matter posterior image
   argument: `--inputWMVolume %s`
inputCSFVolume: (a pathlike object or string representing an existing file)
   Required: input CSF posterior image
   argument: `--inputCSFVolume %s`
outputVolumeBase: (a unicode string)
   Required: output image base name to be appended for each feature vector.
   argument: `--outputVolumeBase %s`
args: (a unicode string)
   Additional parameters to the command
   argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) 
```

Environment variables

Outputs:

None

75.14.14 NeighborhoodMean

Link to code
Wraps the executable command `" NeighborhoodMean "`.

title: Neighborhood Mean
category: Filtering.FeatureDetection
description: Calculates the mean, for the given neighborhood size, at each voxel of the T1, T2, and FLAIR.
version: 0.1.0.$Revision: 1 $(alpha)
documentation-url: http://www.na-mic.org/
license: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt
contributor: This tool was developed by Mark Scully and Jeremy Bockholt.

Inputs:

```plaintext
[Optional]
inputVolume: (a pathlike object or string representing an existing file)
   Required: input image
   argument: `--inputVolume %s`
inputMaskVolume: (a pathlike object or string representing an
(continues on next page)
existing file)
Required: input brain mask image
argument: `--inputMaskVolume %s`
inputRadius: (an integer (int or long))
Required: input neighborhood radius
argument: `--inputRadius %d`
outputVolume: (a boolean or a pathlike object or string representing a file)
Required: output image
argument: `--outputVolume %s`
args: (a unicode string)
Additional parameters to the command
argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing file)
Required: output image

### 75.14.15 NeighborhoodMedian

**Link to code**
Wraps the executable command "NeighborhoodMedian".

**title:** Neighborhood Median
**category:** Filtering.FeatureDetection
**description:** Calculates the median, for the given neighborhood size, at each voxel of the input image.
**version:** 0.1.0.0
**documentation-url:** http://www.na-mic.org/
**license:** https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt
**contributor:** This tool was developed by Mark Scully and Jeremy Bockholt.

**Inputs:**

[Optional]
inputVolume: (a pathlike object or string representing an existing file)
Required: input image
argument: `--inputVolume %s`
inputMaskVolume: (a pathlike object or string representing an existing file)
Required: input brain mask image
argument: `--inputMaskVolume %s`
inputRadius: (an integer (int or long))
Required: input neighborhood radius
argument: `--inputRadius %d`
outputVolume: (a boolean or a pathlike object or string representing a file)
Required: output image
argument: `--outputVolume %s`
args: (a unicode string)
Additional parameters to the command
argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing file)
    Required: output image

75.14.16 STAPLEAnalysis

Link to code
Wraps the executable command "STAPLEAnalysis".

title: Dilate Image
category: Filtering.FeatureDetection
description: Uses mathematical morphology to dilate the input images.
version: 0.1.0.$Revision: 1 $(alpha)
documentation-url: http://www.na-mic.org/
license: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt
contributor: This tool was developed by Mark Scully and Jeremy Bockholt.

Inputs:

[Optional]
inputDimension: (an integer (int or long))
    Required: input image Dimension 2 or 3
    argument: "--inputDimension %d"
inputLabelVolume: (a list of items which are a pathlike object or string representing an existing file)
    Required: input label volume
    argument: "--inputLabelVolume %s..."
outputVolume: (a boolean or a pathlike object or string representing a file)
    Required: output image
    argument: "--outputVolume %s"
args: (a unicode string)
    Additional parameters to the command
    argument: "%s"
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing file)
    Required: output image

75.14.17 TextureFromNoiseImageFilter

Link to code
Wraps the executable command "TextureFromNoiseImageFilter".

title: TextureFromNoiseImageFilter
category: Filtering.FeatureDetection
description: Calculate the local noise in an image.
version: 0.1.0.$Revision: 1 $(alpha)
documentation-url: http://www.na-mic.org/
license: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt
contributor: This tool was developed by Eunyoung Regina Kim

Inputs:

```
inputVolume: (a pathlike object or string representing an existing file)
  Required: input image
  argument: `--inputVolume %s`
inputRadius: (an integer (int or long))
  Required: input neighborhood radius
  argument: `--inputRadius %d`
outputVolume: (a boolean or a pathlike object or string representing a file)
  Required: output image
  argument: `--outputVolume %s`
args: (a unicode string)
  Additional parameters to the command
  argument: `%'s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
```

Outputs:

```
outputVolume: (a pathlike object or string representing an existing file)
  Required: output image
```

75.14.18 TextureMeasureFilter

Link to code
Wraps the executable command ‘ TextureMeasureFilter ‘.
title: Canny Level Set Image Filter
category: Filtering.FeatureDetection
description: The CannySegmentationLevelSet is commonly used to refine a manually generated manual mask.
version: 0.3.0
license: CC
ccontributor: Regina Kim

Inputs:

```
inputVolume: (a pathlike object or string representing an existing file)
  argument: `--inputVolume %s`
inputMaskVolume: (a pathlike object or string representing an existing file)
  argument: `--inputMaskVolume %s`
distance: (an integer (int or long))
  argument: `--distance %d`
```
insideROIValue: (a float)
    argument: `--insideROIValue %f`
outputFilename: (a boolean or a pathlike object or string representing a file)
    argument: `--outputFilename %s`
args: (a unicode string)
    Additional parameters to the command
    argument: `-%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
    Environment variables

Outputs:

outputFilename: (a pathlike object or string representing an existing file)

75.15 interfaces.semtools.legacy.registration

75.15.1 scalartransform

Link to code
Wraps the executable command `'scalartransform`.

Title: ScalarTransform (DTIProcess)

Category: Legacy.Registration

Version: 1.0.0


License: Copyright (c) Casey Goodlett. All rights reserved.

See http://www.ia.unc.edu/dev/Copyright.htm for details. This software is distributed WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the above copyright notices for more information.

Contributor: Casey Goodlett

Inputs:

[Optional]
input_image: (a pathlike object or string representing an existing file)
    Image to transform
    argument: `--input_image %s`
output_image: (a boolean or a pathlike object or string representing a file)
    The transformed image
    argument: `--output_image %s`
transformation: (a boolean or a pathlike object or string representing a file)
    Output file for transformation parameters
    argument: `--transformation %s`
invert: (a boolean)
    Invert transform before applying.
    argument: `--invert`
deformation: (a pathlike object or string representing an existing file)
    Deformation field.
    argument: `--deformation %s`
h_field: (a boolean)
The deformation is an h-field.
argument: `--h_field`

interpolation: ('nearestneighbor' or 'linear' or 'cubic')
Interpolation type (nearestneighbor, linear, cubic)
argument: `--interpolation %s`

args: (a unicode string)
Additional parameters to the command
argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

output_image: (a pathlike object or string representing an existing
file)
The transformed image

transformation: (a pathlike object or string representing an existing
file)
Output file for transformation parameters

75.16 interfaces.semtools.registration.brainsfit

75.16.1 BRAINSFit

Link to code
Wraps the executable command `BRAINSFit`.
title: General Registration (BRAINS)
category: Registration
description: Register a three-dimensional volume to a reference volume (Mattes Mutual Information by
Modules/BRAINSFit. Method described in BRAINSFit: Mutual Information Registrations of Whole-Brain
http://hdl.handle.net/1926/1291
version: 3.0.0
license: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt
acknowledgements: Hans Johnson(1,3,4); Kent Williams(1); Gregory Harris(1), Vincent Magnotta(1,2,3); An-
driy Fedorov(5) 1=University of Iowa Department of Psychiatry, 2=University of Iowa Department of Radiol-
yogy, 3=University of Iowa Department of Biomedical Engineering, 4=University of Iowa Department of Elec-
trical and Computer Engineering, 5=Surgical Planning Lab, Harvard

Inputs:

[Optional]
fixedVolume: (a pathlike object or string representing an existing
file)
Input fixed image (the moving image will be transformed into this
image space).
argument: `--fixedVolume %s`

movingVolume: (a pathlike object or string representing an existing
file)
Input moving image (this image will be transformed into the fixed image space).
argument: `''--movingVolume %s'`

samplingPercentage: (a float)
Fraction of voxels of the fixed image that will be used for registration. The number has to be larger than zero and less or equal to one. Higher values increase the computation time but may give more accurate results. You can also limit the sampling focus with ROI masks and ROI AUTO mask generation. The default is 0.002 (use approximately 0.2% of voxels, resulting in 100000 samples in a 512x512x192 volume) to provide a very fast registration in most cases. Typical values range from 0.01 (1%) for low detail images to 0.2 (20%) for high detail images.
argument: `''--samplingPercentage %f'`

splineGridSize: (a list of items which are an integer (int or long))
Number of BSpline grid subdivisions along each axis of the fixed image, centered on the image space. Values must be 3 or higher for the BSpline to be correctly computed.
argument: `''--splineGridSize %s'`

linearTransform: (a boolean or a pathlike object or string representing a file)
(optional) Output estimated transform - in case the computed transform is not BSpline. NOTE: You must set at least one output object (transform and/or output volume).
argument: `''--linearTransform %s'`

bsplineTransform: (a boolean or a pathlike object or string representing a file)
(optional) Output estimated transform - in case the computed transform is BSpline. NOTE: You must set at least one output object (transform and/or output volume).
argument: `''--bsplineTransform %s'`

outputVolume: (a boolean or a pathlike object or string representing a file)
(optional) Output image: the moving image warped to the fixed image space. NOTE: You must set at least one output object (transform and/or output volume).
argument: `''--outputVolume %s'`

initialTransform: (a pathlike object or string representing an existing file)
Transform to be applied to the moving image to initialize the registration. This can only be used if Initialize Transform Mode is Off.
argument: `''--initialTransform %s'`

initializeTransformMode: ('Off' or 'useMomentsAlign' or 'useCenterOfHeadAlign' or 'useGeometryAlign' or 'useCenterOfROIAlign')
Determine how to initialize the transform center. useMomentsAlign assumes that the center of mass of the images represent similar structures. useCenterOfHeadAlign attempts to use the top of head and shape of neck to drive a center of mass estimate. useGeometryAlign on assumes that the center of the voxel lattice of the images represent similar structures. Off assumes that the physical space of the images are close. This flag is mutually exclusive with the Initialization transform.
argument: `''--initializeTransformMode %s'`

useRigid: (a boolean)
Perform a rigid registration as part of the sequential registration
steps. This family of options overrides the use of transformType if any of them are set.
argument: `--useRigid`

useScaleVersor3D: (a boolean)
Perform a ScaleVersor3D registration as part of the sequential registration steps. This family of options overrides the use of transformType if any of them are set.
argument: `--useScaleVersor3D`

useScaleSkewVersor3D: (a boolean)
Perform a ScaleSkewVersor3D registration as part of the sequential registration steps. This family of options overrides the use of transformType if any of them are set.
argument: `--useScaleSkewVersor3D`

useAffine: (a boolean)
Perform an Affine registration as part of the sequential registration steps. This family of options overrides the use of transformType if any of them are set.
argument: `--useAffine`

useBSpline: (a boolean)
Perform a BSpline registration as part of the sequential registration steps. This family of options overrides the use of transformType if any of them are set.
argument: `--useBSpline`

useSyN: (a boolean)
Perform a SyN registration as part of the sequential registration steps. This family of options overrides the use of transformType if any of them are set.
argument: `--useSyN`

useComposite: (a boolean)
Perform a Composite registration as part of the sequential registration steps. This family of options overrides the use of transformType if any of them are set.
argument: `--useComposite`

maskProcessingMode: ('NOMASK' or 'ROIAUTO' or 'ROI')
Specifies a mask to only consider a certain image region for the registration. If ROIAUTO is chosen, then the mask is computed using Otsu thresholding and hole filling. If ROI is chosen then the mask has to be specified as in input.
argument: `--maskProcessingMode %s`

fixedBinaryVolume: (a pathlike object or string representing an existing file)
Fixed Image binary mask volume, required if Masking Option is ROI. Image areas where the mask volume has zero value are ignored during the registration.
argument: `--fixedBinaryVolume %s`

movingBinaryVolume: (a pathlike object or string representing an existing file)
Moving Image binary mask volume, required if Masking Option is ROI. Image areas where the mask volume has zero value are ignored during the registration.
argument: `--movingBinaryVolume %s`

outputFixedVolumeROI: (a boolean or a pathlike object or string representing a file)
ROI that is automatically computed from the fixed image. Only available if Masking Option is ROIAUTO. Image areas where the mask volume has zero value are ignored during the registration.
argument: `--outputFixedVolumeROI %s`
outputMovingVolumeROI: (a boolean or a pathlike object or string representing a file)
  ROI that is automatically computed from the moving image. Only available if Masking Option is ROIAUTO. Image areas where the mask volume has zero value are ignored during the registration.
  argument: `--outputMovingVolumeROI %s`

useROIBSpline: (a boolean)
  If enabled then the bounding box of the input ROIs defines the BSpline grid support region. Otherwise the Spline grid support region is the whole fixed image.
  argument: `--useROIBSpline`

histogramMatch: (a boolean)
  Apply histogram matching operation for the input images to make them more similar. This is suitable for images of the same modality that may have different brightness or contrast, but the same overall intensity profile. Do NOT use if registering images from different modalities.
  argument: `--histogramMatch`

medianFilterSize: (a list of items which are an integer (int or long))
  Apply median filtering to reduce noise in the input volumes. The 3 values specify the radius for the optional MedianImageFilter preprocessing in all 3 directions (in voxels).
  argument: `--medianFilterSize %s`

removeIntensityOutliers: (a float)
  Remove very high and very low intensity voxels from the input volumes. The parameter specifies the half percentage to decide outliers of image intensities. The default value is zero, which means no outlier removal. If the value of 0.005 is given, the 0.005% of both tails will be thrown away, so 0.01% of intensities in total would be ignored in the statistic calculation.
  argument: `--removeIntensityOutliers %f`

fixedVolume2: (a pathlike object or string representing an existing file)
  Input fixed image that will be used for multimodal registration. (the moving image will be transformed into this image space).
  argument: `--fixedVolume2 %s`

movingVolume2: (a pathlike object or string representing an existing file)
  Input moving image that will be used for multimodal registration (this image will be transformed into the fixed image space).
  argument: `--movingVolume2 %s`

outputVolumePixelType: ('float' or 'short' or 'ushort' or 'int' or 'uint' or 'uchar')
  Data type for representing a voxel of the Output Volume.
  argument: `--outputVolumePixelType %s`

backgroundFillValue: (a float)
  This value will be used for filling those areas of the output image that have no corresponding voxels in the input moving image.
  argument: `--backgroundFillValue %f`

scaleOutputValues: (a boolean)
  If true, and the voxel values do not fit within the minimum and maximum values of the desired outputVolumePixelType, then linearly scale the min/max output image voxel values to fit within the min/max range of the outputVolumePixelType.
  argument: `--scaleOutputValues`
interpolationMode: ('NearestNeighbor' or 'Linear' or 'ResampleInPlace' or 'BSpline' or 'WindowedSinc' or 'Hamming' or 'Cosine' or 'Welch' or 'Lanczos' or 'Blackman')
Type of interpolation to be used when applying transform to moving volume. Options are Linear, NearestNeighbor, BSpline, WindowedSinc, Hamming, Cosine, Welch, Lanczos, or ResampleInPlace. The ResampleInPlace option will create an image with the same discrete voxel values and will adjust the origin and direction of the physical space interpretation.
argument: `--interpolationMode %s`

numberOfIterations: (a list of items which are an integer (int or long))
The maximum number of iterations to try before stopping the optimization. When using a lower value (500-1000) then the registration is forced to terminate earlier but there is a higher risk of stopping before an optimal solution is reached.
argument: `--numberOfIterations %s`

maximumStepLength: (a float)
Starting step length of the optimizer. In general, higher values allow for recovering larger initial misalignments but there is an increased chance that the registration will not converge.
argument: `--maximumStepLength %f`

minimumStepLength: (a list of items which are a float)
Each step in the optimization takes steps at least this big. When none are possible, registration is complete. Smaller values allows the optimizer to make smaller adjustments, but the registration time may increase.
argument: `--minimumStepLength %s`

relaxationFactor: (a float)
Specifies how quickly the optimization step length is decreased during registration. The value must be larger than 0 and smaller than 1. Larger values result in slower step size decrease, which allow for recovering larger initial misalignments but it increases the registration time and the chance that the registration will not converge.
argument: `--relaxationFactor %f`

translationScale: (a float)
How much to scale up changes in position (in mm) compared to unit rotational changes (in radians) -- decrease this to allow for more rotation in the search pattern.
argument: `--translationScale %f`

reproportionScale: (a float)
ScaleVersor3D 'Scale' compensation factor. Increase this to allow for more rescaling in a ScaleVersor3D or ScaleSkewVersor3D search pattern. 1.0 works well with a translationScale of 1000.0
argument: `--reproportionScale %f`

skewScale: (a float)
ScaleSkewVersor3D Skew compensation factor. Increase this to allow for more skew in a ScaleSkewVersor3D search pattern. 1.0 works well with a translationScale of 1000.0
argument: `--skewScale %f`

maxBSplineDisplacement: (a float)
Maximum allowed displacements in image physical coordinates (mm) for BSpline control grid along each axis. A value of 0.0 indicates that the problem should be unbounded. NOTE: This only constrains the BSpline portion, and does not limit the displacement from the associated bulk transform. This can lead to a substantial reduction
in computation time in the BSpline optimizer.
argument: '--maxBSplineDisplacement %f'

fixedVolumeTimeIndex: (an integer (int or long))
The index in the time series for the 3D fixed image to fit. Only allowed if the fixed input volume is 4-dimensional.
argument: '--fixedVolumeTimeIndex %d'

movingVolumeTimeIndex: (an integer (int or long))
The index in the time series for the 3D moving image to fit. Only allowed if the moving input volume is 4-dimensional.
argument: '--movingVolumeTimeIndex %d'

numberOfHistogramBins: (an integer (int or long))
The number of histogram levels used for mutual information metric estimation.
argument: '--numberOfHistogramBins %d'

numberOfMatchPoints: (an integer (int or long))
The number of histogram match points used for mutual information metric estimation.
argument: '--numberOfMatchPoints %d'

costMetric: ('MMI' or 'MSE' or 'NC' or 'MIH')
The cost metric to be used during fitting. Defaults to MMI. Options are MMI (Mattes Mutual Information), MSE (Mean Square Error), NC (Normalized Correlation), MC (Match Cardinality for binary images).
argument: '--costMetric %s'

maskInferiorCutOffFromCenter: (a float)
If Initialize Transform Mode is set to useCenterOfHeadAlign or Masking Option is ROIAUTO then this value defines the how much is cut off from the inferior part of the image. The cut-off distance is specified in millimeters, relative to the image center. If the value is 1000 or larger then no cut-off performed.
argument: '--maskInferiorCutOffFromCenter %f'

ROIAutoDilateSize: (a float)
This flag is only relevant when using ROIAUTO mode for initializing masks. It defines the final dilation size to capture a bit of background outside the tissue region. A setting of 10mm has been shown to help regularize a BSpline registration type so that there is some background constraints to match the edges of the head better.
argument: '--ROIAutoDilateSize %f'

ROIAutoClosingSize: (a float)
This flag is only relevant when using ROIAUTO mode for initializing masks. It defines the hole closing size in mm. It is rounded up to the nearest whole pixel size in each direction. The default is to use a closing size of 9mm. For mouse data this value may need to be reset to 0.9 or smaller.
argument: '--ROIAutoClosingSize %f'

numberOfSamples: (an integer (int or long))
The number of voxels sampled for mutual information computation. Increase this for higher accuracy, at the cost of longer computation time., NOTE that it is suggested to use samplingPercentage instead of this option. However, if set to non-zero, numberOfSamples overwrites the samplingPercentage option.
argument: '--numberOfSamples %d'

strippedOutputTransform: (a boolean or a pathlike object or string representing a file)
Rigid component of the estimated affine transform. Can be used to rigidly register the moving image to the fixed image. NOTE: This value is overridden if either bsplineTransform or linearTransform is
set.
argument: `--strippedOutputTransform %s`

transformType: (a list of items which are a unicode string)
Specifies a list of registration types to be used. The valid types are, Rigid, ScaleVersor3D, ScaleSkewVersor3D, Affine, BSpline and SyN. Specifying more than one in a comma separated list will initialize the next stage with the previous results. If registrationClass flag is used, it overrides this parameter setting.
argument: `--transformType %s`

outputTransform: (a boolean or a pathlike object or string representing a file)
(optional) Filename to which save the (optional) estimated transform. NOTE: You must select either the outputTransform or the outputVolume option.
argument: `--outputTransform %s`

initializeRegistrationByCurrentGenericTransform: (a boolean)
If this flag is ON, the current generic composite transform, resulted from the linear registration stages, is set to initialize the follow nonlinear registration process. However, by the default behaviour, the moving image is first warped based on the existant transform before it is passed to the BSpline registration filter. It is done to speed up the BSpline registration by reducing the computations of composite transform Jacobian.
argument: `--initializeRegistrationByCurrentGenericTransform`

failureExitCode: (an integer (int or long))
If the fit fails, exit with this status code. (It can be used to force a successful exit status of (0) if the registration fails due to reaching the maximum number of iterations.
argument: `--failureExitCode %d`

writeTransformOnFailure: (a boolean)
Flag to save the final transform even if the numberOfIterations are reached without convergence. (Intended for use when --failureExitCode 0 )
argument: `--writeTransformOnFailure`

numberOfThreads: (an integer (int or long))
Explicitly specify the maximum number of threads to use. (default is auto-detected)
argument: `--numberOfThreads %d`

debugLevel: (an integer (int or long))
Display debug messages, and produce debug intermediate results. 0=OFF, 1=Minimal, 10=Maximum debugging.
argument: `--debugLevel %d`

costFunctionConvergenceFactor: (a float)
From itkLBFGSBOptimizer.h: Set/Get the CostFunctionConvergenceFactor. Algorithm terminates when the reduction in cost function is less than (factor * epsmcj) where epsmch is the machine precision. Typical values for factor: 1e+12 for low accuracy; 1e+7 for moderate accuracy and 1e+1 for extremely high accuracy. 1e+9 seems to work well.,
argument: `--costFunctionConvergenceFactor %f`

projectedGradientTolerance: (a float)
From itkLBFGSBOptimizer.h: Set/Get the ProjectedGradientTolerance. Algorithm terminates when the project gradient is below the tolerance. Default lbfgsb value is 1e-5, but 1e-4 seems to work well.,
argument: `--projectedGradientTolerance %f`

maximumNumberOfEvaluations: (an integer (int or long))
(continues on next page)
Maximum number of evaluations for line search in lbfgsb optimizer.
argument: `--maximumNumberOfEvaluations %d`

maximumNumberOfCorrections: (an integer (int or long))
Maximum number of corrections in lbfgsb optimizer.
argument: `--maximumNumberOfCorrections %d`

**gui:** (a boolean)
Display intermediate image volumes for debugging. NOTE: This is not part of the standard build system, and probably does nothing on your installation.
argument: `--gui`

**promptUser:** (a boolean)
Prompt the user to hit enter each time an image is sent to the DebugImageViewer
argument: `--promptUser`

**metricSamplingStrategy:** ('Random')
It defines the method that registration filter uses to sample the input fixed image. Only Random is supported for now.
argument: `--metricSamplingStrategy %s`

**logFileReport:** (a boolean or a pathlike object or string representing a file)
A file to write out final information report in CSV file: MetricName,MetricValue,FixedImageName,FixedMaskName,MovingImageName,MovingMaskName
argument: `--logFileReport %s`

**writeOutputTransformInFloat:** (a boolean)
By default, the output registration transforms (either the output composite transform or each transform component) are written to the disk in double precision. If this flag is ON, the output transforms will be written in single (float) precision. It is especially important if the output transform is a displacement field transform, or it is a composite transform that includes several displacement fields.
argument: `--writeOutputTransformInFloat`

**args:** (a unicode string)
Additional parameters to the command
argument: `%s`

**environ:** (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

Environment variables

**Outputs:**

**linearTransform:** (a pathlike object or string representing an existing file)
(optional) Output estimated transform - in case the computed transform is not BSpline. NOTE: You must set at least one output object (transform and/or output volume).

**bsplineTransform:** (a pathlike object or string representing an existing file)
(optional) Output estimated transform - in case the computed transform is BSpline. NOTE: You must set at least one output object (transform and/or output volume).

**outputVolume:** (a pathlike object or string representing an existing file)
(optional) Output image: the moving image warped to the fixed image space. NOTE: You must set at least one output object (transform
and/or output volume).

outputFixedVolumeROI: (a pathlike object or string representing an existing file)
ROI that is automatically computed from the fixed image. Only available if Masking Option is ROIAUTO. Image areas where the mask volume has zero value are ignored during the registration.

outputMovingVolumeROI: (a pathlike object or string representing an existing file)
ROI that is automatically computed from the moving image. Only available if Masking Option is ROIAUTO. Image areas where the mask volume has zero value are ignored during the registration.

strippedOutputTransform: (a pathlike object or string representing an existing file)
Rigid component of the estimated affine transform. Can be used to rigidly register the moving image to the fixed image. NOTE: This value is overridden if either bsplineTransform or linearTransform is set.

outputTransform: (a pathlike object or string representing an existing file)
(optional) Filename to which save the (optional) estimated transform. NOTE: You must select either the outputTransform or the outputVolume option.

logFileReport: (a pathlike object or string representing an existing file)
A file to write out final information report in CSV file: MetricName, MetricValue, FixedImageName, FixedMaskName, MovingImageName, MovingMaskName

75.17 interfaces.semtools.registration.brainsresample

75.17.1 BRAINSResample

Link to code
Wraps the executable command ‘‘ BRAINSResample ‘‘.

title: Resample Image (BRAINS)
category: Registration
description: This program collects together three common image processing tasks that all involve resampling an image volume: Resampling to a new resolution and spacing, applying a transformation (using an ITK transform IO mechanisms) and Warping (using a vector image deformation field). Full documentation available here: http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/BRAINSResample.

version: 3.0.0
license: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt
contributor: This tool was developed by Vincent Magnotta, Greg Harris, and Hans Johnson.
acknowledgements: The development of this tool was supported by funding from grants NS050568 and NS40068 from the National Institute of Neurological Disorders and Stroke and grants MH31593, MH40856, from the National Institute of Mental Health.

Inputs:

[Optional]
inputVolume: (a pathlike object or string representing an existing file)
Image To Warp
argument: `--inputVolume %s`

referenceVolume: (a pathlike object or string representing an existing file)
Reference image used only to define the output space. If not specified, the warping is done in the same space as the image to warp.
argument: `--referenceVolume %s`

outputVolume: (a boolean or a pathlike object or string representing a file)
Resulting deformed image
argument: `--outputVolume %s`

pixelType: ("float" or 'short' or 'ushort' or 'int' or 'uint' or 'uchar' or 'binary')
Specifies the pixel type for the input/output images. The 'binary' pixel type uses a modified algorithm whereby the image is read in as unsigned char, a signed distance map is created, signed distance map is resampled, and then a thresholded image of type unsigned char is written to disk.
argument: `--pixelType %s`

defformationVolume: (a pathlike object or string representing an existing file)
Displacement Field to be used to warp the image (ITKv3 or earlier)
argument: `--deformationVolume %s`

desiredTransform: (a pathlike object or string representing an existing file)
Filename for the BRAINSFit transform (ITKv3 or earlier)
argument: `--desiredTransform %s`

interpolationMode: ("NearestNeighbor" or 'Linear' or 'ResampleInPlace' or 'BSpline' or 'WindowedSinc' or 'Hamming' or 'Cosine' or 'Welch' or 'Lanczos' or 'Blackman')
Type of interpolation to be used when applying transform to moving volume. Options are Linear, ResampleInPlace, NearestNeighbor, BSpline, or WindowedSinc
argument: `--interpolationMode %s`

inverseTransform: (a boolean)
True/False is to compute inverse of given transformation. Default is false
argument: `--inverseTransform`
defaultValue: (a float)
Default voxel value
argument: `--defaultValue %f`

girdSpacing: (a list of items which are an integer (int or long))
Add warped grid to output image to help show the deformation that occurred with specified spacing. A spacing of 0 in a dimension indicates that grid lines should be rendered to fall exactly (i.e. do not allow displacements off that plane). This is useful for making a 2D image of grid lines from the 3D space
argument: `--gridSpacing %s`

numberOfThreads: (an integer (int or long))
Explicitly specify the maximum number of threads to use.
argument: `--numberOfThreads %d`

args: (a unicode string)
Additional parameters to the command
argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str')
75.18 interfaces.semtools.registration.brainsresize

75.18.1 BRAINSResize

Link to code
Wraps the executable command "BRAINSResize".

title: Resize Image (BRAINS)
category: Registration
description: This program is useful for downsampling an image by a constant scale factor.
version: 3.0.0
license: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt
credits: This tool was developed by Hans Johnson.
acknowledgements: The development of this tool was supported by funding from grants NS050568 and NS40068 from the National Institute of Neurological Disorders and Stroke and grants MH31593, MH40856, from the National Institute of Mental Health.

Inputs:

- [Optional]
  - inputVolume: (a pathlike object or string representing an existing file)
    - Image To Scale
    - argument: '--inputVolume %s'
  - outputVolume: (a boolean or a pathlike object or string representing a file)
    - Resulting scaled image
    - argument: '--outputVolume %s'
  - pixelType: ('float' or 'short' or 'ushort' or 'int' or 'uint' or 'uchar' or 'binary')
    - Specifies the pixel type for the input/output images. The 'binary' pixel type uses a modified algorithm whereby the image is read in as unsigned char, a signed distance map is created, signed distance map is resampled, and then a thresholded image of type unsigned char is written to disk.
    - argument: '--pixelType %s'
  - scaleFactor: (a float)
    - The scale factor for the image spacing.
    - argument: '--scaleFactor %f'
  - args: (a unicode string)
    - Additional parameters to the command
    - argument: '%s'

Outputs:

- outputVolume: (a pathlike object or string representing an existing file)
  - Resulting deformed image
  - argument: %s
75.19 interfaces.semtools.registration.specialized

75.19.1 BRAINSDemonWarp

Link to code
Wraps the executable command ‘‘BRAINSDemonWarp ‘‘.

title: Demon Registration (BRAINS)
category: Registration.Specialized
description: This program finds a deformation field to warp a moving image onto a fixed image. The images must be of the same signal kind, and contain an image of the same kind of object. This program uses the Thirion Demons warp software in ITK, the Insight Toolkit. Additional information is available at: http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/BRAINSDemonWarp.

version: 3.0.0
license: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt
ccontributor: This tool was developed by Hans J. Johnson and Greg Harris.

acknowledgements: The development of this tool was supported by funding from grants NS050568 and NS40068 from the National Institute of Neurological Disorders and Stroke and grants MH31593, MH40856, from the National Institute of Mental Health.

Inputs:

movingVolume: (a pathlike object or string representing an existing file)
  Required: input moving image
  argument: ‘‘--movingVolume %s’’

fixedVolume: (a pathlike object or string representing an existing file)
  Required: input fixed (target) image
  argument: ‘‘--fixedVolume %s’’

inputPixelType: (‘float’ or ‘short’ or ‘ushort’ or ‘int’ or ‘uchar’)
  Input volumes will be typecast to this format:
  float|short|ushort|int|uchar
  argument: ‘‘--inputPixelType %s’’

outputVolume: (a boolean or a pathlike object or string representing a file)
  Required: output resampled moving image (will have the same physical space as the fixedVolume).
  argument: ‘‘--outputVolume %s’’

outputDisplacementFieldVolume: (a boolean or a pathlike object or string representing a file)
  Output deformation field vector image (will have the same physical space as the fixedVolume).
  argument: ‘‘--outputDisplacementFieldVolume %s’’

outputPixelType: (‘float’ or ‘short’ or ‘ushort’ or ‘int’ or ‘uchar’)
  outputVolume will be typecast to this format:
  float|short|ushort|int|uchar
  argument: ‘‘--outputPixelType %s’’

interpolationMode: (‘NearestNeighbor’ or ‘Linear’ or...
Type of interpolation to be used when applying transform to moving volume. Options are Linear, ResampleInPlace, NearestNeighbor, BSpline, or WindowedSinc

argument: `--interpolationMode %s`

registrationFilterType: ('Demons' or 'FastSymmetricForces' or 'Diffeomorphic')

Registration Filter Type: Demons|FastSymmetricForces|Diffeomorphic

argument: `--registrationFilterType %s`

smoothDisplacementFieldSigma: (a float)

A gaussian smoothing value to be applied to the deformation field at each iteration.

argument: `--smoothDisplacementFieldSigma %f`

numberOfPyramidLevels: (an integer (int or long))

Number of image pyramid levels to use in the multi-resolution registration.

argument: `--numberOfPyramidLevels %d`

minimumFixedPyramid: (a list of items which are an integer (int or long))

The shrink factor for the first level of the fixed image pyramid.
(i.e. start at 1/16 scale, then 1/8, then 1/4, then 1/2, and finally full scale)

argument: `--minimumFixedPyramid %s`

minimumMovingPyramid: (a list of items which are an integer (int or long))

The shrink factor for the first level of the moving image pyramid.
(i.e. start at 1/16 scale, then 1/8, then 1/4, then 1/2, and finally full scale)

argument: `--minimumMovingPyramid %s`

arrayOfPyramidLevelIterations: (a list of items which are an integer (int or long))

The number of iterations for each pyramid level

argument: `--arrayOfPyramidLevelIterations %s`

histogramMatch: (a boolean)

Histogram Match the input images. This is suitable for images of the same modality that may have different absolute scales, but the same overall intensity profile.

argument: `--histogramMatch`

numberOfHistogramBins: (an integer (int or long))

The number of histogram levels

argument: `--numberOfHistogramBins %d`

numberOfMatchPoints: (an integer (int or long))

The number of match points for histogramMatch

argument: `--numberOfMatchPoints %d`

medianFilterSize: (a list of items which are an integer (int or long))

Median filter radius in all 3 directions. When images have a lot of salt and pepper noise, this step can improve the registration.

argument: `--medianFilterSize %s`

initializeWithDisplacementField: (a pathlike object or string representing an existing file)

Initial deformation field vector image file name

argument: `--initializeWithDisplacementField %s`

initializeWithTransform: (a pathlike object or string representing an existing file)

Initial Transform filename
<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>argument: <code>--initializeWithTransform %s</code></td>
<td></td>
</tr>
<tr>
<td>maskProcessingMode: ('NOMASK' or 'ROIAUTO' or 'ROI' or 'BOBF')</td>
<td>What mode to use for using the masks: NOMASK</td>
</tr>
<tr>
<td>fixedBinaryVolume: (a pathlike object or string representing an existing file)</td>
<td>Mask filename for desired region of interest in the Fixed image.</td>
</tr>
<tr>
<td>movingBinaryVolume: (a pathlike object or string representing an existing file)</td>
<td>Mask filename for desired region of interest in the Moving image.</td>
</tr>
<tr>
<td>lowerThresholdForBOBF: (an integer (int or long))</td>
<td>Lower threshold for performing BOBF</td>
</tr>
<tr>
<td>upperThresholdForBOBF: (an integer (int or long))</td>
<td>Upper threshold for performing BOBF</td>
</tr>
<tr>
<td>backgroundFillValue: (an integer (int or long))</td>
<td>Replacement value to overwrite background when performing BOBF</td>
</tr>
<tr>
<td>seedForBOBF: (a list of items which are an integer (int or long))</td>
<td>coordinates in all 3 directions for Seed when performing BOBF</td>
</tr>
<tr>
<td>neighborhoodForBOBF: (a list of items which are an integer (int or long))</td>
<td>neighborhood in all 3 directions to be included when performing BOBF</td>
</tr>
<tr>
<td>outputDisplacementFieldPrefix: (a unicode string)</td>
<td>Displacement field filename prefix for writing separate x, y, and z component images</td>
</tr>
<tr>
<td>outputCheckerboardVolume: (a boolean or a pathlike object or string representing a file)</td>
<td>Generate a checkerboard image volume between the fixedVolume and the deformed movingVolume.</td>
</tr>
<tr>
<td>checkerboardPatternSubdivisions: (a list of items which are an integer (int or long))</td>
<td>Number of Checkerboard subdivisions in all 3 directions</td>
</tr>
<tr>
<td>outputNormalized: (a boolean)</td>
<td>Flag to warp and write the normalized images to output. In normalized images the image values are fit-scaled to be between 0 and the maximum storage type value.</td>
</tr>
<tr>
<td>outputDebug: (a boolean)</td>
<td>Flag to write debugging images after each step.</td>
</tr>
<tr>
<td>gradient_type: ('0' or '1' or '2')</td>
<td>Type of gradient used for computing the demons force (0 is</td>
</tr>
</tbody>
</table>
symmetrized, 1 is fixed image, 2 is moving image)
argument: `--gradient_type %s`

upFieldSmoothing: (a float)
Smoothing sigma for the update field at each iteration
argument: `--upFieldSmoothing %f`

max_step_length: (a float)
Maximum length of an update vector (0: no restriction)
argument: `--max_step_length %f`

use_vanilla_dem: (a boolean)
Run vanilla demons algorithm
argument: `--use_vanilla_dem`

gui: (a boolean)
Display intermediate image volumes for debugging
argument: `--gui`

promptUser: (a boolean)
Prompt the user to hit enter each time an image is sent to the
DebugImageViewer
argument: `--promptUser`

numberOfBCHApproximationTerms: (an integer (int or long))
Number of terms in the BCH expansion
argument: `--numberOfBCHApproximationTerms %d`

numberOfThreads: (an integer (int or long))
Explicitly specify the maximum number of threads to use.
argument: `--numberOfThreads %d`

args: (a unicode string)
Additional parameters to the command
argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})

Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing
file)
Required: output resampled moving image (will have the same physical
space as the fixedVolume).

outputDisplacementFieldVolume: (a pathlike object or string
representing an existing file)
Output deformation field vector image (will have the same physical
space as the fixedVolume).

outputCheckerboardVolume: (a pathlike object or string representing
an existing file)
Generate a checkerboard image volume between the fixedVolume and the
dehomed movingVolume.

75.19.2 BRAINSTransformFromFiducials

Link to code
Wraps the executable command "BRAINSTransformFromFiducials".
title: Fiducial Registration (BRAINS)
category: Registration.Specialized
description: Computes a rigid, similarity or affine transform from a matched list of fiducials
version: 0.1.0.$Revision$
contributor: Casey B Goodlett
acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

[Optional]
fixedLandmarks: (a list of items which are a list of from 3 to 3 items which are a float)
    Ordered list of landmarks in the fixed image
    argument: "--fixedLandmarks %s..."
movingLandmarks: (a list of items which are a list of from 3 to 3 items which are a float)
    Ordered list of landmarks in the moving image
    argument: "--movingLandmarks %s..."
saveTransform: (a boolean or a pathlike object or string representing a file)
    Save the transform that results from registration
    argument: "--saveTransform %s"
transformType: ('Translation' or 'Rigid' or 'Similarity')
    Type of transform to produce
    argument: "--transformType %s"
fixedLandmarksFile: (a pathlike object or string representing an existing file)
    An fcsv formatted file with a list of landmark points.
    argument: "--fixedLandmarksFile %s"
movingLandmarksFile: (a pathlike object or string representing an existing file)
    An fcsv formatted file with a list of landmark points.
    argument: "--movingLandmarksFile %s"
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: "--numberOfThreads %d"
args: (a unicode string)
    Additional parameters to the command
    argument: "%s"
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
BRAINSDemonWarp

license: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt

ccontributor: This tool was developed by Hans J. Johnson and Greg Harris.

acknowledgements: The development of this tool was supported by funding from grants NS050568 and NS40068 from the National Institute of Neurological Disorders and Stroke and grants MH31593, MH40856, from the National Institute of Mental Health.

Inputs:

[Optional]
movingVolume: (a list of items which are a pathlike object or string representing an existing file)
  Required: input moving image
  argument: "--movingVolume %s..."

fixedVolume: (a list of items which are a pathlike object or string representing an existing file)
  Required: input fixed (target) image
  argument: "--fixedVolume %s..."

inputPixelType: ('float' or 'short' or 'ushort' or 'int' or 'uchar')
  Input volumes will be typecast to this format:
  float|short|ushort|int|uchar
  argument: "--inputPixelType %s"

outputVolume: (a boolean or a pathlike object or string representing a file)
  Required: output resampled moving image (will have the same physical space as the fixedVolume).
  argument: "--outputVolume %s"

outputDisplacementFieldVolume: (a boolean or a pathlike object or string representing a file)
  Output deformation field vector image (will have the same physical space as the fixedVolume).
  argument: "--outputDisplacementFieldVolume %s"

outputPixelType: ('float' or 'short' or 'ushort' or 'int' or 'uchar')
  outputVolume will be typecast to this format:
  float|short|ushort|int|uchar
  argument: "--outputPixelType %s"

interpolationMode: ('NearestNeighbor' or 'Linear' or 'ResampleInPlace' or 'BSpline' or 'WindowedSinc' or 'Hamming' or 'Cosine' or 'Welch' or 'Lanczos' or 'Blackman')
  Type of interpolation to be used when applying transform to moving volume. Options are Linear, ResampleInPlace, NearestNeighbor, BSpline, or WindowedSinc.
  argument: "--interpolationMode %s"

registrationFilterType: ('Demons' or 'FastSymmetricForces' or 'Diffeomorphic' or 'LogDemons' or 'SymmetricLogDemons')
  Registration Filter Type: Demons|FastSymmetricForces|Diffeomorphic|LogDemons|SymmetricLogDemons
  argument: "--registrationFilterType %s"

smoothDisplacementFieldSigma: (a float)
  A gaussian smoothing value to be applied to the deformation field at each iteration.
  argument: "--smoothDisplacementFieldSigma %f"

numberOfPyramidLevels: (an integer (int or long))
  Number of image pyramid levels to use in the multi-resolution registration.
  argument: "--numberOfPyramidLevels %d"

minimumFixedPyramid: (a list of items which are an integer (int or long))

(continues on next page)
The shrink factor for the first level of the fixed image pyramid.
(i.e. start at 1/16 scale, then 1/8, then 1/4, then 1/2, and finally
full scale)
argument: ``--minimumFixedPyramid %s``

minimumMovingPyramid: (a list of items which are an integer (int or
long))
The shrink factor for the first level of the moving image pyramid.
(i.e. start at 1/16 scale, then 1/8, then 1/4, then 1/2, and finally
full scale)
argument: ``--minimumMovingPyramid %s``

arrayOfPyramidLevelIterations: (a list of items which are an integer
(int or long))
The number of iterations for each pyramid level
argument: ``--arrayOfPyramidLevelIterations %s``

histogramMatch: (a boolean)
Histogram Match the input images. This is suitable for images of the
same modality that may have different absolute scales, but the same
overall intensity profile.
argument: ``--histogramMatch``
	numberOfHistogramBins: (an integer (int or long))
The number of histogram levels
argument: ``--numberOfHistogramBins %d``

numberOfMatchPoints: (an integer (int or long))
The number of match points for histogramMatch
argument: ``--numberOfMatchPoints %d``

medianFilterSize: (a list of items which are an integer (int or
long))
Median filter radius in all 3 directions. When images have a lot of
salt and pepper noise, this step can improve the registration.
argument: ``--medianFilterSize %s``

initializeWithDisplacementField: (a pathlike object or string
representing an existing file)
Initial deformation field vector image file name
argument: ``--initializeWithDisplacementField %s``

initializeWithTransform: (a pathlike object or string representing an
existing file)
Initial Transform filename
argument: ``--initializeWithTransform %s``

makeBOBF: (a boolean)
Flag to make Brain-Only Background-Filled versions of the input and
target volumes.
argument: ``--makeBOBF``

fixedBinaryVolume: (a pathlike object or string representing an
existing file)
Mask filename for desired region of interest in the Fixed image.
argument: ``--fixedBinaryVolume %s``

movingBinaryVolume: (a pathlike object or string representing an
existing file)
Mask filename for desired region of interest in the Moving image.
argument: ``--movingBinaryVolume %s``

lowerThresholdForBOBF: (an integer (int or long))
Lower threshold for performing BOBF
argument: ``--lowerThresholdForBOBF %d``

upperThresholdForBOBF: (an integer (int or long))
Upper threshold for performing BOBF
argument: ``--upperThresholdForBOBF %d``

backgroundFillValue: (an integer (int or long))
Replacement value to overwrite background when performing BOBF
argument: `--backgroundFillValue %d`

seedForBOBF: (a list of items which are an integer (int or long))
coordinates in all 3 directions for Seed when performing BOBF
argument: `--seedForBOBF %s`

neighborhoodForBOBF: (a list of items which are an integer (int or long))
neighborhood in all 3 directions to be included when performing BOBF
argument: `--neighborhoodForBOBF %s`

outputDisplacementFieldPrefix: (a unicode string)
Displacement field filename prefix for writing separate x, y, and z
component images
argument: `--outputDisplacementFieldPrefix %s`

outputCheckerboardVolume: (a boolean or a pathlike object or string)
representing a file
Generate a checkerboard image volume between the fixedVolume and the
deformed movingVolume.
argument: `--outputCheckerboardVolume %s`

checkerboardPatternSubdivisions: (a list of items which are an
integer (int or long))
Number of Checkerboard subdivisions in all 3 directions
argument: `--checkerboardPatternSubdivisions %s`

outputNormalized: (a boolean)
Flag to warp and write the normalized images to output. In
normalized images the image values are fit-scaled to be between 0
and the maximum storage type value.
argument: `--outputNormalized`

outputDebug: (a boolean)
Flag to write debugging images after each step.
argument: `--outputDebug`

weightFactors: (a list of items which are a float)
Weight factors for each input images
argument: `--weightFactors %s`

gradient_type: ('0' or '1' or '2')
Type of gradient used for computing the demons force (0 is
symmetrized, 1 is fixed image, 2 is moving image)
argument: `--gradient_type %s`

upFieldSmoothing: (a float)
Smoothing sigma for the update field at each iteration
argument: `--upFieldSmoothing %f`

max_step_length: (a float)
Maximum length of an update vector (0: no restriction)
argument: `--max_step_length %f`

use_vanilla_dem: (a boolean)
Run vanilla demons algorithm
argument: `--use_vanilla_dem`

gui: (a boolean)
Display intermediate image volumes for debugging
argument: `--gui`

promptUser: (a boolean)
Prompt the user to hit enter each time an image is sent to the
DebugImageViewer
argument: `--promptUser`

numberOfBCHApproximationTerms: (an integer (int or long))
Number of terms in the BCH expansion
argument: `--numberOfBCHApproximationTerms %d`

numberOfThreads: (an integer (int or long))
Explicitly specify the maximum number of threads to use.
argument: `--numberofThreads %d`

args: (a unicode string)
Additional parameters to the command
argument: `--%s`

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing
file)
Required: output resampled moving image (will have the same physical
space as the fixedVolume).

outputDisplacementFieldVolume: (a pathlike object or string
representing an existing file)
Output deformation field vector image (will have the same physical
space as the fixedVolume).

outputCheckerboardVolume: (a pathlike object or string representing
an existing file)
Genete a checkerboard image volume between the fixedVolume and the
deformed movingVolume.

75.20 interfaces.semtools.segmentation.specialized

75.20.1 BRAINSABC

Link to code
Wraps the executable command `'' BRAINSABC ''.``

title: Intra-subject registration, bias Correction, and tissue classification (BRAINS)
category: Segmentation.Specialized
description: Atlas-based tissue segmentation method. This is an algorithmic extension of work done by XXXX
at UNC and Utah XXXX need more description here.

Inputs:

[Optional]
inputVolumes: (a list of items which are a pathlike object or string
representing an existing file)
The list of input image files to be segmented.
argument: `''--inputVolumes %s...''`
atlasDefinition: (a pathlike object or string representing an
existing file)
Contains all parameters for Atlas
argument: `''--atlasDefinition %s''`
restoreState: (a pathlike object or string representing an existing
file)
The initial state for the registration process
argument: `''--restoreState %s''`
saveState: (a boolean or a pathlike object or string representing a
file)
(optional) Filename to which save the final state of the
registration
argument: `''--saveState %s''`
```
inputVolumeTypes: (a list of items which are a unicode string)
The list of input image types corresponding to the inputVolumes.
argument: `'--inputVolumeTypes %s'`
outputDir: (a boolean or a pathlike object or string representing a
directory)
Output directory
argument: `'--outputDir %s'`

atlasToSubjectTransformType: ('Identity' or 'Rigid' or 'Affine' or
'Bspline' or 'SyN')
What type of linear transform type do you want to use to register
the atlas to the reference subject image.
argument: `'--atlasToSubjectTransformType %s'`

atlasToSubjectTransform: (a boolean or a pathlike object or string
representing a file)
The transform from atlas to the subject
argument: `'--atlasToSubjectTransform %s'`

atlasToSubjectInitialTransform: (a boolean or a pathlike object or
string representing a file)
The initial transform from atlas to the subject
argument: `'--atlasToSubjectInitialTransform %s'`

subjectIntermodeTransformType: ('Identity' or 'Rigid' or 'Affine' or
'Bspline')
What type of linear transform type do you want to use to register
the atlas to the reference subject image.
argument: `'--subjectIntermodeTransformType %s'`

outputVolumes: (a boolean or a list of items which are a pathlike
object or string representing a file)
Corrected Output Images: should specify the same number of images as
inputVolume, if only one element is given, then it is used as a file
pattern where %s is replaced by the imageVolumeType, and %d by the
index list location.
argument: `'--outputVolumes %s...'`

outputLabels: (a boolean or a pathlike object or string representing
a file)
Output Label Image
argument: `'--outputLabels %s'`

outputDirtyLabels: (a boolean or a pathlike object or string
representing a file)
Output Dirty Label Image
argument: `'--outputDirtyLabels %s'`

posteriorTemplate: (a unicode string)
filename template for Posterior output files
argument: `'--posteriorTemplate %s'`

outputFormat: ('NIFTI' or 'Meta' or 'Nrrd')
Output format
argument: `'--outputFormat %s'`

interpolationMode: ('Bspline' or 'NearestNeighbor' or 'WindowedSinc'
or 'Linear' or 'ResampleInPlace' or 'Hamming' or 'Cosine' or
'Welch' or 'Lanczos' or 'Blackman')
Type of interpolation to be used when applying transform to moving
volume. Options are Linear, NearestNeighbor, B spline, WindowedSinc,
or ResampleInPlace. The ResampleInPlace option will create an image
with the same discrete voxel values and will adjust the origin and
direction of the physical space interpretation.
argument: `'--interpolationMode %s'`

maxIterations: (an integer (int or long))
Filter iterations
```
argument: `--maxIterations %d`

medianFilterSize: (a list of items which are an integer (int or long))

The radius for the optional MedianImageFilter preprocessing in all 3 directions.

argument: `--medianFilterSize %s`

filterIteration: (an integer (int or long))

Filter iterations

argument: `--filterIteration %d`

filterTimeStep: (a float)

Filter time step should be less than (PixelSpacing/(1^(DIM+1))), value is set to negative, then allow automatic setting of this value.

argument: `--filterTimeStep %f`

filterMethod: ('None' or 'CurvatureFlow' or 'GradientAnisotropicDiffusion' or 'Median')

Filter method for preprocessing of registration

argument: `--filterMethod %s`

maxBiasDegree: (an integer (int or long))

Maximum bias degree

argument: `--maxBiasDegree %d`

useKNN: (a boolean)

Use the KNN stage of estimating posteriors.

argument: `--useKNN`

purePlugsThreshold: (a float)

If this threshold value is greater than zero, only pure samples are used to compute the distributions in EM classification, and only pure samples are used for KNN training. The default value is set to 0, that means not using pure plugs. However, a value of 0.2 is suggested if you want to activate using pure plugs option.

argument: `--purePlugsThreshold %f`

numberOfSubSamplesInEachPlugArea: (a list of items which are an integer (int or long))

Number of continous index samples taken at each direction of lattice space for each plug volume.

argument: `--numberOfSubSamplesInEachPlugArea %s`

atlasWarpingOff: (a boolean)

Deformable registration of atlas to subject

argument: `--atlasWarpingOff`

gridSize: (a list of items which are an integer (int or long))

Grid size for atlas warping with BSplines

argument: `--gridSize %s`

defaultSuffix: (a unicode string)

argument: `--defaultSuffix %s`

implicitOutputs: (a boolean or a list of items which are a pathlike object or string representing a file)

Outputs to be made available to NiPype. Needed because not all BRAINSABC outputs have command line arguments.

argument: `--implicitOutputs %s...`

debuglevel: (an integer (int or long))

Display debug messages, and produce debug intermediate results.

0=OFF, 1=Minimal, 10=Maximum debugging.

argument: `--debuglevel %d`

writeLess: (a boolean)

Does not write posteriors and filtered, bias corrected images

argument: `--writeLess`

numberOfThreads: (an integer (int or long))
Explicitly specify the maximum number of threads to use.
argument: `--numberofThreads %d`
args: (a unicode string)
Additional parameters to the command
argument: `\%s`
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a value
of class 'str', nipype default value: {})
Environment variables

Outputs:

- saveState: (a pathlike object or string representing an existing file)
  (optional) Filename to which save the final state of the registration
- outputDir: (a pathlike object or string representing an existing directory)
  Output directory
- atlasToSubjectTransform: (a pathlike object or string representing an existing file)
  The transform from atlas to the subject
- atlasToSubjectInitialTransform: (a pathlike object or string representing an existing file)
  The initial transform from atlas to the subject
- outputVolumes: (a list of items which are a pathlike object or string representing an existing file)
  Corrected Output Images: should specify the same number of images as inputVolume, if only one element is given, then it is used as a file pattern where %s is replaced by the imageVolumeType, and %d by the index list location.
- outputLabels: (a pathlike object or string representing an existing file)
  Output Label Image
- outputDirtyLabels: (a pathlike object or string representing an existing file)
  Output Dirty Label Image
- implicitOutputs: (a list of items which are a pathlike object or string representing an existing file)
  Outputs to be made available to NiPype. Needed because not all BRAINSABC outputs have command line arguments.

### 75.20.2 BRAINSConstellationDetector

**Link to code**

Wraps the executable command `BRAINSConstellationDetector`.

**title:** Brain Landmark Constellation Detector (BRAINS)

**category:** Segmentation.Specialized

**description:** This program will find the mid-sagittal plane, a constellation of landmarks in a volume, and create an AC/PC aligned data set with the AC point at the center of the voxel lattice (labeled at the origin of the image physical space.) Part of this work is an extention of the algorithms originally described by Dr. Babak A. Ardekani, Alvin H. Bachman, Model-based automatic detection of the anterior and posterior commissures on MRI scans, NeuroImage, Volume 46, Issue 3, 1 July 2009, Pages 677-682, ISSN 1053-8119, DOI: 10.1016/j.neuroimage.2009.02.030. (http://www.sciencedirect.com/science/article/B6WNP-4VRP25C-4/2/8207b962a38aa83c822c6379bc43fe4c)

**version:** 1.0
Inputs:

[Optional]

houghEyeDetectorMode: (an integer (int or long))
, This flag controls the mode of Hough eye detector. By default,
value of 1 is for T1W images, while the value of 0 is for T2W and PD
images.,
argument: `--houghEyeDetectorMode %d`

inputTemplateModel: (a pathlike object or string representing an
existing file)
User-specified template model.,
argument: `--inputTemplateModel %s`

LLSModel: (a pathlike object or string representing an existing file)
Linear least squares model filename in HD5 format
argument: `--LLSModel %s`

inputVolume: (a pathlike object or string representing an existing
file)
Input image in which to find ACPC points
argument: `--inputVolume %s`

outputVolume: (a boolean or a pathlike object or string representing
a file)
ACPC-aligned output image with the same voxels, but updated origin,
and direction cosign so that the AC point would fall at the physical
location (0.0,0.0,0.0), and the mid-sagital plane is the plane where
physical L/R coordinate is 0.0.
argument: `--outputVolume %s`

outputResampledVolume: (a boolean or a pathlike object or string
representing a file)
ACPC-aligned output image in a resampled unifor space. Currently
this is a 1mm, 256^3, Identity direction image.
argument: `--outputResampledVolume %s`

outputTransform: (a boolean or a pathlike object or string
representing a file)
The filename for the original space to ACPC alignment to be written
(in .h5 format).
argument: `--outputTransform %s`

outputLandmarksInInputSpace: (a boolean or a pathlike object or
string representing a file)
The filename for the new subject-specific landmark definition file
in the same format produced by Slicer3 (.fcsv) with the landmarks in
the original image space (the detected RP, AC, PC, and VN4) in it to
be written.
argument: `--outputLandmarksInInputSpace %s`

outputLandmarksInACPCAlignedSpace: (a boolean or a pathlike object or
string representing a file)
The filename for the new subject-specific landmark definition file
in the same format produced by Slicer3 (.fcsv) with the landmarks in
the output image space (the detected RP, AC, PC, and VN4) in it to
be written.
argument: `--outputLandmarksInACPCAlignedSpace %s`

outputMRML: (a boolean or a pathlike object or string representing a
file)
The filename for the new subject-specific scene definition file in
the same format produced by Slicer3 (in .mrml format). Only the
components that were specified by the user on command line would be
generated. Compatible components include inputVolume, outputVolume,
outputLandmarksInInputSpace, outputLandmarksInACPCAlignedSpace, and
outputTransform,
argument: `--outputMRML %s`

outputVerificationScript: (a boolean or a pathlike object or string representing a file)

The filename for the Slicer3 script that verifies the aligned landmarks against the aligned image file. This will happen only in conjunction with saveOutputLandmarks and an outputVolume.,
argument: `--outputVerificationScript %s`

mspQualityLevel: (an integer (int or long))

Flag controls how aggressive the MSP is estimated. 0=quick estimate (9 seconds), 1=normal estimate (11 seconds), 2=great estimate (22 seconds), 3=best estimate (58 seconds), NOTE: -1= Prealigned so no estimate!,
argument: `--mspQualityLevel %d`

otsuPercentileThreshold: (a float)

This is a parameter to FindLargestForegroundFilledMask, which is employed when acLowerBound is set and an outputUntransformedClippedVolume is requested.,
argument: `--otsuPercentileThreshold %f`

acLowerBound: (a float)

When generating a resampled output image, replace the image with the BackgroundFillValue everywhere below the plane This Far in physical units (millimeters) below (inferior to) the AC point (as found by the model.) The oversize default was chosen to have no effect. Based on visualizing a thousand masks in the IPIG study, we recommend a limit no smaller than 80.0 mm.,
argument: `--acLowerBound %f`

cutOutHeadInOutputVolume: (a boolean)

Flag to cut out just the head tissue when producing an (un)transformed clipped volume.,
argument: `--cutOutHeadInOutputVolume`

outputUntransformedClippedVolume: (a boolean or a pathlike object or string representing a file)

Output image in which to store neck-clipped input image, with the use of --acLowerBound and maybe --cutOutHeadInUntransformedVolume.

argument: `--outputUntransformedClippedVolume %s`

rescaleIntensities: (a boolean)

Flag to turn on rescaling image intensities on input.,
argument: `--rescaleIntensities`

trimRescaledIntensities: (a float)

Turn on clipping the rescaled image one-tailed on input. Units of standard deviations above the mean. Very large values are very permissive. Non-positive value turns clipping off. Defaults to removing 0.00001 of a normal tail above the mean.,
argument: `--trimRescaledIntensities %f`

rescaleIntensitiesOutputRange: (a list of items which are an integer (int or long))

This pair of integers gives the lower and upper bounds on the signal portion of the output image. Out-of-field voxels are taken from BackgroundFillValue.,
argument: `--rescaleIntensitiesOutputRange %s`

BackgroundFillValue: (a unicode string)

Fill the background of image with specified short int value. Enter number or use BIGNEG for a large negative number.
argument: `--BackgroundFillValue %s`

interpolationMode: ('NearestNeighbor' or 'Linear' or 'ResampleInPlace' or 'BSpline' or 'WindowedSinc' or 'Hamming' or 'WindowedSinc' or 'Hamming' or..."
Interpolation to be used when applying transform to moving volume. Options are Linear, ResampleInPlace, NearestNeighbor, BSpline, or WindowedSinc.

- **forceACPoint**: (a list of items which are a float)
  - Use this flag to manually specify the AC point from the original image on the command line.
  - **argument**: `--forceACPoint %s`

- **forcePCPoint**: (a list of items which are a float)
  - Use this flag to manually specify the PC point from the original image on the command line.
  - **argument**: `--forcePCPoint %s`

- **forceVN4Point**: (a list of items which are a float)
  - Use this flag to manually specify the VN4 point from the original image on the command line.
  - **argument**: `--forceVN4Point %s`

- **forceRPPoint**: (a list of items which are a float)
  - Use this flag to manually specify the RP point from the original image on the command line.
  - **argument**: `--forceRPPoint %s`

- **inputLandmarksEMSP**: (a pathlike object or string representing an existing file)
  - The filename for the new subject-specific landmark definition file in the same format produced by Slicer3 (in .fcsv) with the landmarks in the estimated MSP aligned space to be loaded. The detector will only process landmarks not enlisted on the file.
  - **argument**: `--inputLandmarksEMSP %s`

- **forceHoughEyeDetectorReportFailure**: (a boolean)
  - Flag indicates whether the Hough eye detector should report failure.
  - **argument**: `--forceHoughEyeDetectorReportFailure`

- **rmpj**: (a float)
  - Search radius for MPJ in unit of mm.
  - **argument**: `--rmpj %f`

- **rac**: (a float)
  - Search radius for AC in unit of mm.
  - **argument**: `--rac %f`

- **rpc**: (a float)
  - Search radius for PC in unit of mm.
  - **argument**: `--rpc %f`

- **rVN4**: (a float)
  - Search radius for VN4 in unit of mm.
  - **argument**: `--rVN4 %f`

- **debug**: (a boolean)
  - Show internal debugging information.
  - **argument**: `--debug`

- **verbose**: (a boolean)
  - Show more verbose output.
  - **argument**: `--verbose`

- **writeBranded2DImage**: (a boolean or a pathlike object or string representing a file)
  - The filename for the 2D .png branded midline debugging image. This will happen only in conjunction with requesting an outputVolume.
  - **argument**: `--writeBranded2DImage %s`

- **resultsDir**: (a boolean or a pathlike object or string representing a directory)
The directory for the debugging images to be written.
argument: `--resultsDir %s`

writeDebuggingImagesLevel: (an integer (int or long))
This flag controls if debugging images are produced. By default
value of 0 is no images. Anything greater than zero will be
increasing level of debugging images.
argument: `--writeDebuggingImagesLevel %d`

numberOfThreads: (an integer (int or long))
Explicitly specify the maximum number of threads to use.
argument: `--numberOfThreads %d`

atlasVolume: (a pathlike object or string representing an existing
file)
Atlas volume image to be used for BRAINSFit registration
argument: `--atlasVolume %s`

atlasLandmarks: (a pathlike object or string representing an existing
file)
Atlas landmarks to be used for BRAINSFit registration
initialization,
argument: `--atlasLandmarks %s`

atlasLandmarkWeights: (a pathlike object or string representing an
existing file)
Weights associated with atlas landmarks to be used for BRAINSFit
registration initialization,
argument: `--atlasLandmarkWeights %s`

args: (a unicode string)
Additional parameters to the command
argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing
file)
ACPC-aligned output image with the same voxels, but updated origin,
and direction cosines so that the AC point would fall at the physical
location (0.0, 0.0, 0.0), and the mid-sagittal plane is the plane where
physical L/R coordinate is 0.0.

outputResampledVolume: (a pathlike object or string representing an
existing file)
ACPC-aligned output image in a resampled uniform space. Currently
this is a 1mm, 256^3, Identity direction image.

outputTransform: (a pathlike object or string representing an
existing file)
The filename for the original space to ACPC alignment to be written
in .h5 format.,

outputLandmarksInInputSpace: (a pathlike object or string
representing an existing file)
The filename for the new subject-specific landmark definition file
in the same format produced by Slicer3 (.fcsv) with the landmarks in
the original image space (the detected RP, AC, PC, and VN4) in it to
be written.,

outputLandmarksInACPCAlignedSpace: (a pathlike object or string
representing an existing file)
The filename for the new subject-specific landmark definition file

in the same format produced by Slicer3 (.fcsv) with the landmarks in
the output image space (the detected RP, AC, PC, and VN4) in it to
be written.,
outputMRML: (a pathlike object or string representing an existing
file)

, The filename for the new subject-specific scene definition file in
the same format produced by Slicer3 (in .mrml format). Only the
components that were specified by the user on command line would be
generated. Compatible components include inputVolume, outputVolume,
outputLandmarksInInputSpace, outputLandmarksInACPCAlignedSpace, and
outputTransform.,
outputVerificationScript: (a pathlike object or string representing
an existing file)

, The filename for the Slicer3 script that verifies the aligned
landmarks against the aligned image file. This will happen only in conjunction with saveOutputLandmarks and an outputVolume.,
outputUntransformedClippedVolume: (a pathlike object or string
representing an existing file)

Output image in which to store neck-clipped input image, with the
use of --acLowerBound and maybe --cutOutHeadInUntransformedVolume.
writeBranded2DImage: (a pathlike object or string representing an
existing file)

, The filename for the 2D .png branded midline debugging image. This
will happen only in conjunction with requesting an outputVolume.,
resultsDir: (a pathlike object or string representing an existing
directory)

, The directory for the debugging images to be written.,

75.20.3 BRAINSCreateLabelMapFromProbabilityMaps

Link to code
Wraps the executable command “BRAINSCreateLabelMapFromProbabilityMaps”.

title: Create Label Map From Probability Maps (BRAINS)
category: Segmentation.Specialized
description: Given A list of Probability Maps, generate a LabelMap.

Inputs:

[Optional]
inputProbabilityVolume: (a list of items which are a pathlike object
or string representing an existing file)
The list of probability images.
argument: "--inputProbabilityVolume %s..."
priorLabelCodes: (a list of items which are an integer (int or long))
A list of PriorLabelCode values used for coding the output label images
argument: "--priorLabelCodes %s"
foregroundPriors: (a list of items which are an integer (int or long))
A list: For each Prior Label, 1 if foreground, 0 if background
argument: "--foregroundPriors %s"
nonAirRegionMask: (a pathlike object or string representing an
existing file)
a mask representing the 'NonAirRegion' -- Just force pixels in this
region to zero
argument: "--nonAirRegionMask %s"
inclusionThreshold: (a float)
tolerance for inclusion
argument: `--inclusionThreshold %f`
dirtyLabelVolume: (a boolean or a pathlike object or string representing a file)
the labels prior to cleaning
argument: `--dirtyLabelVolume %s`
cleanLabelVolume: (a boolean or a pathlike object or string representing a file)
the foreground labels volume
argument: `--cleanLabelVolume %s`
args: (a unicode string)
Additional parameters to the command
argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

dirtyLabelVolume: (a pathlike object or string representing an existing file)
the labels prior to cleaning
cleanLabelVolume: (a pathlike object or string representing an existing file)
the foreground labels volume

75.20.4 BRAINSCut

Link to code
Wraps the executable command `BRAINSCut`.

title: BRAINSCut (BRAINS)
category: Segmentation.Specialized
description: Automatic Segmentation using neural networks
version: 1.0
license: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt
contributor: Vince Magnotta, Hans Johnson, Greg Harris, Kent Williams, Eunyoung Regina Kim

Inputs:

[Optional]
netConfiguration: (a pathlike object or string representing an existing file)
XML File defining BRAINSCut parameters. OLD NAME. PLEASE USE modelConfigurationFilename instead.
argument: `--netConfiguration %s`
modelConfigurationFilename: (a pathlike object or string representing an existing file)
XML File defining BRAINSCut parameters
argument: `--modelConfigurationFilename %s`
trainModelStartIndex: (an integer (int or long))
Starting iteration for training
argument: `--trainModelStartIndex %d`
verbose: (an integer (int or long))
print out some debugging information
argument: `--verbose %d`
multiStructureThreshold: (a boolean)

(continues on next page)
multiStructureThreshold module to deal with overlapping area
argument: `--multiStructureThreshold`

histogramEqualization: (a boolean)
A Histogram Equalization process could be added to the
creating/applying process from Subject To Atlas. Default is false,
which generate input vectors without Histogram Equalization.
argument: `--histogramEqualization`

computeSSEOn: (a boolean)
compute Sum of Square Error (SSE) along the trained model until the
number of iteration given in the modelConfigurationFilename file
argument: `--computeSSEOn`

generateProbability: (a boolean)
Generate probability map
argument: `--generateProbability`

createVectors: (a boolean)
create vectors for training neural net
argument: `--createVectors`

trainModel: (a boolean)
train the neural net
argument: `--trainModel`

NoTrainingVectorShuffling: (a boolean)
If this flag is on, there will be no shuffling.
argument: `--NoTrainingVectorShuffling`

applyModel: (a boolean)
apply the neural net
argument: `--applyModel`

validate: (a boolean)
validate data set. Just need for the first time run (This is for
validation of xml file and not working yet)
argument: `--validate`

method: ("RandomForest" or 'ANN')
argument: `--method %s`

numberOfTrees: (an integer (int or long))
Random tree: number of trees. This is to be used when only one
model with specified depth wish to be created.
argument: `--numberOfTrees %d`

randomTreeDepth: (an integer (int or long))
Random tree depth. This is to be used when only one model with
specified depth wish to be created.
argument: `--randomTreeDepth %d`

modelFilename: (a unicode string)
model file name given from user (not by xml configuration file)
argument: `--modelFilename %s`

args: (a unicode string)
Additional parameters to the command
argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
**75.20.5 BRAINSMultiSTAPLE**

**Link to code**

Wraps the executable command "BRAINSMultiSTAPLE".

**Title**: Create best representative label map

**Category**: Segmentation.Specialized

**Description**: given a list of label map images, create a representative/average label map.

**Inputs:**

```
[Optional]
inputCompositeT1Volume: (a pathlike object or string representing an existing file)
    Composite T1, all label maps transformed into the space for this image.
    argument: '--inputCompositeT1Volume %s'
inputLabelVolume: (a list of items which are a pathlike object or string representing an existing file)
    The list of probability images.
    argument: '--inputLabelVolume %s...'
inputTransform: (a list of items which are a pathlike object or string representing an existing file)
    transforms to apply to label volumes
    argument: '--inputTransform %s...'
labelForUndecidedPixels: (an integer (int or long))
    Label for undecided pixels
    argument: '--labelForUndecidedPixels %d'
resampledVolumePrefix: (a unicode string)
    if given, write out resampled volumes with this prefix
    argument: '--resampledVolumePrefix %s'
skipResampling: (a boolean)
    Omit resampling images into reference space
    argument: '--skipResampling '
outputMultiSTAPLE: (a boolean or a pathlike object or string representing a file)
    the MultiSTAPLE average of input label volumes
    argument: '--outputMultiSTAPLE %s'
outputConfusionMatrix: (a boolean or a pathlike object or string representing a file)
    Confusion Matrix
    argument: '--outputConfusionMatrix %s'
args: (a unicode string)
    Additional parameters to the command
    argument: '%s'
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
```

**Outputs:**

```
outputMultiSTAPLE: (a pathlike object or string representing an existing file)
    the MultiSTAPLE average of input label volumes
outputConfusionMatrix: (a pathlike object or string representing an existing file)
    Confusion Matrix
```
75.20.6 BRAINSROIAuto

Link to code

Wraps the executable command "BRAINSROIAuto".

title: Foreground masking (BRAINS)
category: Segmentation.Specialized
description: This program is used to create a mask over the most prominent foreground region in an image. This is accomplished via a combination of otsu thresholding and a closing operation. More documentation is available here: http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/ForegroundMasking.

version: 2.4.1
license: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt
acknowledgements: Hans Johnson(1,3,4); Kent Williams(1); Gregory Harris(1), Vincent Magnotta(1,2,3); Andriy Fedorov(5), fedorov -at- bwh.harvard.edu (Slicer integration); (1=University of Iowa Department of Psychiatry, 2=University of Iowa Department of Radiology, 3=University of Iowa Department of Biomedical Engineering, 4=University of Iowa Department of Electrical and Computer Engineering, 5=Surgical Planning Lab, Harvard)

Inputs:

[Optional]

inputVolume: (a pathlike object or string representing an existing file)
The input image for finding the largest region filled mask.
argument: `--inputVolume %s`

outputROIMaskVolume: (a boolean or a pathlike object or string representing a file)
The ROI automatically found from the input image.
argument: `--outputROIMaskVolume %s`

outputVolume: (a boolean or a pathlike object or string representing a file)
The inputVolume with optional [maskOutput|cropOutput] to the region of the brain mask.
argument: `--outputVolume %s`

maskOutput: (a boolean)
The inputVolume multiplied by the ROI mask.
argument: `--maskOutput`

cropOutput: (a boolean)
The inputVolume cropped to the region of the ROI mask.
argument: `--cropOutput`

otsuPercentileThreshold: (a float)
Parameter to the Otsu threshold algorithm.
argument: `--otsuPercentileThreshold %f`

thresholdCorrectionFactor: (a float)
A factor to scale the Otsu algorithm's result threshold, in case clipping mangles the image.
argument: `--thresholdCorrectionFactor %f`

closingSize: (a float)
The Closing Size (in millimeters) for largest connected filled mask.
This value is divided by image spacing and rounded to the next largest voxel number.
argument: `--closingSize %f`

ROIAutoDilateSize: (a float)
This flag is only relevant when using ROIAUTO mode for initializing masks. It defines the final dilation size to capture a bit of background outside the tissue region. At setting of 10mm has been shown to help regularize a BSpline registration type so that there is some background constraints to match the edges of the head

(continues on next page)
better.

**argument:** `--ROIAutoDilateSize %f`

**outputVolumePixelType:** `('float' or 'short' or 'ushort' or 'int' or 'uint' or 'uchar')`

The output image Pixel Type is the scalar datatype for representation of the Output Volume.

**argument:** `--outputVolumePixelType %s`

**numberOfThreads:** `(an integer (int or long))`

Explicitly specify the maximum number of threads to use.

**argument:** `--numberOfThreads %d`

**args:** `(a unicode string)`

Additional parameters to the command

**argument:** `"%s"`

**environ:** `(a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})`

Environment variables

**Outputs:**

**outputROIMaskVolume:** `(a pathlike object or string representing an existing file)`

The ROI automatically found from the input image.

**outputVolume:** `(a pathlike object or string representing an existing file)`

The inputVolume with optional [maskOutput|cropOutput] to the region of the brain mask.

### 75.20.7 BinaryMaskEditorBasedOnLandmarks

**Link to code**

Wraps the executable command “ BinaryMaskEditorBasedOnLandmarks “.

**title:** BRAINS Binary Mask Editor Based On Landmarks(BRAINS)

**category:** Segmentation.Specialized

**version:** 1.0


**Inputs:**

[Optional]

**inputBinaryVolume:** `(a pathlike object or string representing an existing file)`

Input binary image in which to be edited

**argument:** `"--inputBinaryVolume %s"`

**outputBinaryVolume:** `(a boolean or a pathlike object or string representing a file)`

Output binary image in which to be edited

**argument:** `"--outputBinaryVolume %s"`

**inputLandmarksFilename:** `(a pathlike object or string representing an existing file)`

The filename for the landmark definition file in the same format produced by Slicer3 (.fcsv).

**argument:** `"--inputLandmarksFilename %s"`

**inputLandmarkNames:** `(a list of items which are a unicode string)`

A target input landmark name to be edited. This should be listed in the inputLandmarkFilename Given.

**argument:** `"--inputLandmarkNames %s"`
setCutDirectionForLandmark: (a list of items which are a unicode string)

Setting the cutting out direction of the input binary image to the one of anterior, posterior, left, right, superior or posterior.

(ENUMERATION: ANTERIOR, POSTERIOR, LEFT, RIGHT, SUPERIOR, POSTERIOR)

argument: ``--setCutDirectionForLandmark %s``

setCutDirectionForObliquePlane: (a list of items which are a unicode string)

If this is true, the mask will be thresholded out to the direction of inferior, posterior, and/or left. Default behavior is that cutting out to the direction of superior, anterior and/or right.

argument: ``--setCutDirectionForObliquePlane %s``

inputLandmarkNamesForObliquePlane: (a list of items which are a unicode string)

Three subset landmark names of inputLandmarksFilename for a oblique plane computation. The plane computed for binary volume editing.

argument: ``--inputLandmarkNamesForObliquePlane %s``

args: (a unicode string)

Additional parameters to the command

argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

Environment variables

Outputs:

outputBinaryVolume: (a pathlike object or string representing an existing file)

Output binary image in which to be edited

75.20.8 ESLR

Link to code

Wraps the executable command “ESLR“.

title: Clean Contiguous Label Map (BRAINS)
category: Segmentation.Specialized
description: From a range of label map values, extract the largest contiguous region of those labels

Inputs:

[Optional]

inputVolume: (a pathlike object or string representing an existing file)

Input Label Volume

argument: ``--inputVolume %s``

outputVolume: (a boolean or a pathlike object or string representing a file)

Output Label Volume

argument: ``--outputVolume %s``

low: (an integer (int or long))

The lower bound of the labels to be used.

argument: ``--low %d``

high: (an integer (int or long))

The higher bound of the labels to be used.

argument: ``--high %d``

closingSize: (an integer (int or long))
The closing size for hole filling.
argument: `--closingSize %d`

openingSize: (an integer (int or long))
The opening size for hole filling.
argument: `--openingSize %d`

safetySize: (an integer (int or long))
The safetySize size for the clipping region.
argument: `--safetySize %d`

preserveOutside: (a boolean)
For values outside the specified range, preserve those values.
argument: `--preserveOutside`

numberOfThreads: (an integer (int or long))
Explicitly specify the maximum number of threads to use.
argument: `--numberOfThreads %d`

args: (a unicode string)
Additional parameters to the command
argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing
file)
Output Label Volume

75.21 interfaces.semtools.utilities.brains

75.21.1 BRAINSAlignMSP

Link to code
Wraps the executable command ‘BRAINSAlignMSP’.
title: Align Mid Saggital Brain (BRAINS)
category: Utilities.BRAINS
description: Resample an image into ACPC alignement ACPCDetect
Inputs:

[Optional]
inputVolume: (a pathlike object or string representing an existing
file)
, The Image to be resampled,
argument: `--inputVolume %s`

OutputresampleMSP: (a boolean or a pathlike object or string
representing a file)
, The image to be output.,
argument: `--OutputresampleMSP %s`

verbose: (a boolean)
, Show more verbose output,
argument: `--verbose`

resultsDir: (a boolean or a pathlike object or string representing a
directory)
, The directory for the results to be written.,
argument: `--resultsDir %s`
writedebuggingImagesLevel: (an integer (int or long))

  This flag controls if debugging images are produced. By default value of 0 is no images. Anything greater than zero will be increasing level of debugging images.

  argument: `--writedebuggingImagesLevel %d`

mspQualityLevel: (an integer (int or long))

  Flag controls how aggressive the MSP is estimated. 0=quick estimate (9 seconds), 1=normal estimate (11 seconds), 2=good estimate (22 seconds), 3=best estimate (58 seconds).

  argument: `--mspQualityLevel %d`

rescaleIntensities: (a boolean)

  Flag to turn on rescaling image intensities on input.

  argument: `--rescaleIntensities`

trimRescaledIntensities: (a float)

  Turn on clipping the rescaled image one-tailed on input. Units of standard deviations above the mean. Very large values are very permissive. Non-positive value turns clipping off. Defaults to removing 0.00001 of a normal tail above the mean.

  argument: `--trimRescaledIntensities %f`

rescaleIntensitiesOutputRange: (a list of items which are an integer (int or long))

  This pair of integers gives the lower and upper bounds on the signal portion of the output image. Out-of-field voxels are taken from BackgroundFillValue.

  argument: `--rescaleIntensitiesOutputRange %s`

BackgroundFillValue: (a unicode string)

  Fill the background of image with specified short int value. Enter number or use BIGNEG for a large negative number.

  argument: `--BackgroundFillValue %s`

interpolationMode: ('NearestNeighbor' or 'Linear' or 'ResampleInPlace' or 'BSpline' or 'WindowedSinc' or 'Hamming' or 'Cosine' or 'Welch' or 'Lanczos' or 'Blackman')

  Type of interpolation to be used when applying transform to moving volume. Options are Linear, ResampleInPlace, NearestNeighbor, BSpline, or WindowedSinc.

  argument: `--interpolationMode %s`

numberOfThreads: (an integer (int or long))

  Explicitly specify the maximum number of threads to use.

  argument: `--numberOfThreads %d`

args: (a unicode string)

  Additional parameters to the command

  argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

  Environment variables

**Outputs:**

Output resampleMSP: (a pathlike object or string representing an existing file)

  The image to be output.

resultsDir: (a pathlike object or string representing an existing directory)

  The directory for the results to be written.
75.21.2 BRAINSClip Inferior

Link to code
Wraps the executable command “BRAINSClip Inferior”.

title: Clip Inferior of Center of Brain (BRAINS)
category: Utilities.BRAINS
description: This program will read the inputVolue as a short int image, write the BackgroundFillValue everywhere inferior to the lower bound, and write the resulting clipped short int image in the outputVolue.
version: 1.0

Inputs:

<table>
<thead>
<tr>
<th>Optional</th>
<th>inputVolue: (a pathlike object or string representing an existing file)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Input image to make a clipped short int copy from.</td>
</tr>
<tr>
<td></td>
<td>argument: <code>--inputVolue %s</code></td>
</tr>
<tr>
<td>Optional</td>
<td>outputVolue: (a boolean or a pathlike object or string representing a file)</td>
</tr>
<tr>
<td></td>
<td>Output image, a short int copy of the upper portion of the input image, filled with BackgroundFillValue.</td>
</tr>
<tr>
<td></td>
<td>argument: <code>--outputVolue %s</code></td>
</tr>
<tr>
<td></td>
<td>acLowerBound: (a float)</td>
</tr>
<tr>
<td></td>
<td>When the input image to the output image, replace the image with the BackgroundFillValue everywhere below the plane This Far in physical units (millimeters) below (inferior to) the AC point (assumed to be the voxel field middle.) The oversize default was chosen to have no effect. Based on visualizing a thousand masks in the IPIG study, we recommend a limit no smaller than 80.0 mm.</td>
</tr>
<tr>
<td></td>
<td>argument: <code>--acLowerBound %f</code></td>
</tr>
<tr>
<td></td>
<td>BackgroundFillValue: (a unicode string)</td>
</tr>
<tr>
<td></td>
<td>Fill the background of image with specified short int value. Enter number or use BIGNEG for a large negative number.</td>
</tr>
<tr>
<td></td>
<td>argument: <code>--BackgroundFillValue %s</code></td>
</tr>
<tr>
<td></td>
<td>numberOfThreads: (an integer (int or long))</td>
</tr>
<tr>
<td></td>
<td>Explicitly specify the maximum number of threads to use.</td>
</tr>
<tr>
<td></td>
<td>argument: <code>--numberOfThreads %d</code></td>
</tr>
<tr>
<td></td>
<td>args: (a unicode string)</td>
</tr>
<tr>
<td></td>
<td>Additional parameters to the command</td>
</tr>
<tr>
<td></td>
<td>argument: <code>%s</code></td>
</tr>
<tr>
<td></td>
<td>environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})</td>
</tr>
<tr>
<td></td>
<td>Environment variables</td>
</tr>
</tbody>
</table>

Outputs:

| outputVolue: (a pathlike object or string representing an existing file) |
| Output image, a short int copy of the upper portion of the input image, filled with BackgroundFillValue. |

75.21.3 BRAINSConstellationModeler

Link to code
Wraps the executable command “BRAINSConstellationModeler”.

title: Generate Landmarks Model (BRAINS)
category: Utilities.BRAINS
description: Train up a model for BRAINSConstellationDetector
Inputs:

[Optional]

- **verbose**: (a boolean)
  - Show more verbose output.
  - Argument: `--verbose`

- **inputTrainingList**: (a pathlike object or string representing an existing file)
  - Setup file, giving all parameters for training up a template model for each landmark.
  - Argument: `--inputTrainingList %s`

- **outputModel**: (a boolean or a pathlike object or string representing a file)
  - The full filename of the output model file.
  - Argument: `--outputModel %s`

- **saveOptimizedLandmarks**: (a boolean)
  - Flag to make a new subject-specific landmark definition file in the same format produced by Slicer3 with the optimized landmark (the detected RP, AC, and PC) in it. Useful to tighten the variances in the ConstellationModeler.
  - Argument: `--saveOptimizedLandmarks`

- **optimizedLandmarksFilenameExtender**: (a unicode string)
  - If the trainingList is (indexFullPathName) and contains landmark data filenames [path]/[filename].fcsv, make the optimized landmarks filenames out of [path]/[filename](thisExtender) and the optimized version of the input trainingList out of (indexFullPathName)(thisExtender), when you rewrite all the landmarks according to the saveOptimizedLandmarks flag.
  - Argument: `--optimizedLandmarksFilenameExtender %s`

- **resultsDir**: (a boolean or a pathlike object or string representing a directory)
  - The directory for the results to be written.
  - Argument: `--resultsDir %s`

- **mspQualityLevel**: (an integer (int or long))
  - Flag controls how aggressive the MSP is estimated. 0=quick estimate (9 seconds), 1=normal estimate (11 seconds), 2=great estimate (22 seconds), 3=best estimate (58 seconds).
  - Argument: `--mspQualityLevel %d`

- **rescaleIntensities**: (a boolean)
  - Flag to turn on rescaling image intensities on input.
  - Argument: `--rescaleIntensities`

- **trimRescaledIntensities**: (a float)
  - Turn on clipping the rescaled image one-tailed on input. Units of standard deviations above the mean. Very large values are very permissive. Non-positive value turns clipping off. Defaults to removing 0.00001 of a normal tail above the mean.
  - Argument: `--trimRescaledIntensities %f`

- **rescaleIntensitiesOutputRange**: (a list of items which are an integer (int or long))
  - This pair of integers gives the lower and upper bounds on the signal portion of the output image. Out-of-field voxels are taken from BackgroundFillValue.
  - Argument: `--rescaleIntensitiesOutputRange %s`

- **BackgroundFillValue**: (a unicode string)
  - Fill the background of image with specified short int value. Enter number or use BIGNEG for a large negative number.
  - Argument: `--BackgroundFillValue %s`

(continues on next page)
This flag controls if debugging images are produced. By default value of 0 is no images. Anything greater than zero will be increasing level of debugging images.

**numberOfThreads**: (an integer (int or long))

Explicitly specify the maximum number of threads to use.

**args**: (a unicode string)

Additional parameters to the command

**environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

Environment variables

**Outputs:**

**outputModel**: (a pathlike object or string representing an existing file)

The full filename of the output model file.

**resultsDir**: (a pathlike object or string representing an existing directory)

The directory for the results to be written.

### 75.21.4 BRAINSEyeDetector

**Link to code**

Wraps the executable command ``BRAINSEyeDetector``

**title**: Eye Detector (BRAINS)

**category**: Utilities.BRAINS

**version**: 1.0


**Inputs:**

[Optional]

**numberOfThreads**: (an integer (int or long))

Explicitly specify the maximum number of threads to use.

**inputVolume**: (a pathlike object or string representing an existing file)

The input volume

**outputVolume**: (a boolean or a pathlike object or string representing a file)

The output volume

**debugDir**: (a unicode string)

A place for debug information

**args**: (a unicode string)

Additional parameters to the command

**environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

Environment variables
Outputs:

outputVolume: (a pathlike object or string representing an existing file)
  The output volume

### 75.21.5 BRAINSInitializedControlPoints

**Link to code**
Wraps the executable command `" BRAINSInitializedControlPoints "`.

**title:** Initialized Control Points (BRAINS)

**category:** Utilities.BRAINS

**description:** Outputs bspline control points as landmarks

**version:** 0.1.0.$Revision: 916 $(alpha)

**license:** [https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt](https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt)

**contributor:** Mark Scully

**acknowledgements:** This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149. Additional support for Mark Scully and Hans Johnson at the University of Iowa.

**Inputs:**

[Optional]

- **inputVolume:** (a pathlike object or string representing an existing file)
  
  Input Volume
  
  argument: `"--inputVolume %s"`

- **outputVolume:** (a boolean or a pathlike object or string representing a file)
  
  Output Volume
  
  argument: `"--outputVolume %s"`

- **splineGridSize:** (a list of items which are an integer (int or long))
  
  The number of subdivisions of the BSpline Grid to be centered on the image space. Each dimension must have at least 3 subdivisions for the BSpline to be correctly computed.
  
  argument: `"--splineGridSize %s"`

- **permuteOrder:** (a list of items which are an integer (int or long))
  
  The permutation order for the images. The default is 0,1,2 (i.e. no permutation)
  
  argument: `"--permuteOrder %s"`

- **outputLandmarksFile:** (a unicode string)
  
  Output filename
  
  argument: `"--outputLandmarksFile %s"`

- **numberOfThreads:** (an integer (int or long))
  
  Explicitly specify the maximum number of threads to use.
  
  argument: `"--numberOfThreads %d"`

- **args:** (a unicode string)
  
  Additional parameters to the command
  
  argument: `"%s"`

- **environ:** (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) 
  
  Environment variables

**Outputs:**

outputVolume: (a pathlike object or string representing an existing file)
75.21.6 BRAINSLandmarkInitializer

Link to code
Wraps the executable command `BRAINSLandmarkInitializer`.

title: BRAINSLandmarkInitializer
category: Utilities.BRAINS
description: Create transformation file (.mat) from a pair of landmarks (.fcsv) files.
version: 1.0
license: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt
contributor: Eunyoung Regina Kim

Inputs:

- inputFixedLandmarkFilename: (a pathlike object or string representing an existing file)
  - input fixed landmark. *.fcsv
  - argument: `--inputFixedLandmarkFilename %s`

- inputMovingLandmarkFilename: (a pathlike object or string representing an existing file)
  - input moving landmark. *.fcsv
  - argument: `--inputMovingLandmarkFilename %s`

- inputWeightFilename: (a pathlike object or string representing an existing file)
  - Input weight file name for landmarks. Higher weighted landmark will be considered more heavily. Weights are proportional, that is the magnitude of weights will be normalized by its minimum and maximum value.
  - argument: `--inputWeightFilename %s`

- outputTransformFilename: (a boolean or a pathlike object or string representing a file)
  - output transform file name (ex: ./outputTransform.mat)
  - argument: `--outputTransformFilename %s`

- args: (a unicode string)
  - Additional parameters to the command
  - argument: `%s`

- environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  - Environment variables

Outputs:

- outputTransformFilename: (a pathlike object or string representing an existing file)
  - output transform file name (ex: ./outputTransform.mat)

75.21.7 BRAINSLinearModelerEPCA

Link to code
Wraps the executable command `BRAINSLinearModelerEPCA`.

title: Landmark Linear Modeler (BRAINS)
category: Utilities.BRAINS
description: Training linear model using EPCA. Implementation based on my MS thesis, "A METHOD..."
FOR AUTOMATED LANDMARK CONSTELLATION DETECTION USING EVOLUTIONARY PRINCIPAL COMPONENTS AND STATISTICAL SHAPE MODELS''
version: 1.0
documentation-url: http://www.nitrc.org/projects/brainscdetector/

Inputs:

[Optional]
inputTrainingList: (a pathlike object or string representing an existing file)
  Input Training Landmark List Filename,
  argument: ``--inputTrainingList %s''
numberOfThreads: (an integer (int or long))
  Explicitly specify the maximum number of threads to use.
  argument: ``--numberOfThreads %d''
args: (a unicode string)
  Additional parameters to the command
  argument: ``%s''
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

Environment variables

Outputs:

None

75.21.8 BRAINSLmkTransform

Link to code
Wraps the executable command 'BRAINSLmkTransform'.
title: Landmark Transform (BRAINS)
category: Utilities.BRAINS
description: This utility program estimates the affine transform to align the fixed landmarks to the moving landmarks, and then generate the resampled moving image to the same physical space as that of the reference image.
version: 1.0
documentation-url: http://www.nitrc.org/projects/brainscdetector/

Inputs:

[Optional]
inputMovingLandmarks: (a pathlike object or string representing an existing file)
  Input Moving Landmark list file in fcsv,
  argument: ``--inputMovingLandmarks %s''
inputFixedLandmarks: (a pathlike object or string representing an existing file)
  Input Fixed Landmark list file in fcsv,
  argument: ``--inputFixedLandmarks %s''
outputAffineTransform: (a boolean or a pathlike object or string representing a file)
  The filename for the estimated affine transform,
  argument: ``--outputAffineTransform %s''
inputMovingVolume: (a pathlike object or string representing an existing file)
  The filename of input moving volume
  argument: ``--inputMovingVolume %s''
inputReferenceVolume: (a pathlike object or string representing an

(continues on next page)
existing file)
The filename of the reference volume
argument: `'--inputReferenceVolume %s'`
outputResampledVolume: (a boolean or a pathlike object or string
representing a file)
The filename of the output resampled volume
argument: `'--outputResampledVolume %s'`
numberOfThreads: (an integer (int or long))
Explicitly specify the maximum number of threads to use.
argument: `'--numberOfThreads %d'`
args: (a unicode string)
Additional parameters to the command
argument: `'\%s'`
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a value
of class 'str', nipype default value: { })
Environment variables

Outputs:

outputAffineTransform: (a pathlike object or string representing an
existing file)
The filename for the estimated affine transform,
outputResampledVolume: (a pathlike object or string representing an
existing file)
The filename of the output resampled volume

75.21.9 BRAINSMush

Link to code
Wraps the executable command "BRAINSMush".
title: Brain Extraction from T1/T2 image (BRAINS)
category: Utilities.BRAINS
description: This program: 1) generates a weighted mixture image optimizing the mean and variance and 2)
produces a mask of the brain volume
version: 0.1.0.$Revision: 1.4 $ (alpha)
documentation-url: http://mri.radiology.uiowa.edu
license: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt
contributor: This tool is a modification by Steven Dunn of a program developed by Greg Harris and Ron Pierson.
acknowledgements: This work was developed by the University of Iowa Departments of Radiology and Psychiatry. This software was supported in part of NIH/NINDS award NS050568.

Inputs:

[Optional]
inputFirstVolume: (a pathlike object or string representing an
existing file)
Input image (1) for mixture optimization
argument: `'--inputFirstVolume %s'`
inputSecondVolume: (a pathlike object or string representing an
existing file)
Input image (2) for mixture optimization
argument: `'--inputSecondVolume %s'`
inputMaskVolume: (a pathlike object or string representing an
existing file)
Input label image for mixture optimization

(continues on next page)
argument: `"--inputMaskVolume %s"`
outputWeightsFile: (a boolean or a pathlike object or string representing a file)
Output Weights File
argument: `"--outputWeightsFile %s"
outputVolume: (a boolean or a pathlike object or string representing a file)
The MUSH image produced from the T1 and T2 weighted images
argument: `"--outputVolume %s"
outputMask: (a boolean or a pathlike object or string representing a file)
The brain volume mask generated from the MUSH image
argument: `"--outputMask %s"
seed: (a list of items which are an integer (int or long))
Seed Point for Brain Region Filling
argument: `"--seed %s"
desiredMean: (a float)
Desired mean within the mask for weighted sum of both images.
argument: `"--desiredMean %f"
desiredVariance: (a float)
Desired variance within the mask for weighted sum of both images.
argument: `"--desiredVariance %f"
lowerThresholdFactorPre: (a float)
Lower threshold factor for finding an initial brain mask
argument: `"--lowerThresholdFactorPre %f"
upperThresholdFactorPre: (a float)
Upper threshold factor for finding an initial brain mask
argument: `"--upperThresholdFactorPre %f"
lowerThresholdFactor: (a float)
Lower threshold factor for defining the brain mask
argument: `"--lowerThresholdFactor %f"
upperThresholdFactor: (a float)
Upper threshold factor for defining the brain mask
argument: `"--upperThresholdFactor %f"
boundingBoxSize: (a list of items which are an integer (int or long))
Size of the cubic bounding box mask used when no brain mask is present
argument: `"--boundingBoxSize %s"
boundingBoxStart: (a list of items which are an integer (int or long))
XYZ point-coordinate for the start of the cubic bounding box mask used when no brain mask is present
argument: `"--boundingBoxStart %s"
numberOfThreads: (an integer (int or long))
Explicitly specify the maximum number of threads to use.
argument: `"--numberOfThreads %d"
args: (a unicode string)
Additional parameters to the command
argument: `"%s"
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

outputWeightsFile: (a pathlike object or string representing an
75.21.10 BRAINSSnapShotWriter

Link to code

Wraps the executable command "BRAINSSnapShotWriter".

Title: BRAINSSnapShotWriter
Category: Utilities.BRAINS
Description: Create 2D snapshot of input images. Mask images are color-coded
Version: 1.0
License: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt
Contributor: Eunyoung Regina Kim

Inputs:

[Optional]

inputVolumes: (a list of items which are a pathlike object or string representing an existing file)
  Input image volume list to be extracted as 2D image. Multiple input is possible. At least one input is required.
  argument: `''--inputVolumes %s...''`
inputBinaryVolumes: (a list of items which are a pathlike object or string representing an existing file)
  Input mask (binary) volume list to be extracted as 2D image. Multiple input is possible.
  argument: `''--inputBinaryVolumes %s...''`
inputSliceToExtractInPhysicalPoint: (a list of items which are a float)
  2D slice number of input images. For autoWorkUp output, which AC-PC aligned, 0,0,0 will be the center.
  argument: `''--inputSliceToExtractInPhysicalPoint %s''`
inputSliceToExtractInIndex: (a list of items which are an integer (int or long))
  2D slice number of input images. For size of 256*256*256 image, 128 is usually used.
  argument: `''--inputSliceToExtractInIndex %s''`
inputSliceToExtractInPercent: (a list of items which are an integer (int or long))
  2D slice number of input images. Percentage input from 0%-100%. (ex. --inputSliceToExtractInPercent 50,50,50)
  argument: `''--inputSliceToExtractInPercent %s''`
inputPlaneDirection: (a list of items which are an integer (int or long))
  Plane to display. In general, 0=sagittal, 1=coronal, and 2=axial plane.
  argument: `''--inputPlaneDirection %s''`
outputFilename: (a boolean or a pathlike object or string representing a file)
  2D file name of input images. Required.
argument: `--outputFilename %s`
args: (a unicode string)
  Additional parameters to the command
  argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
  Environment variables

<table>
<thead>
<tr>
<th>Outputs:</th>
</tr>
</thead>
</table>

  outputFilename: (a pathlike object or string representing an existing
  file)
  2D file name of input images. Required.

### 75.21.11 BRAINSTransformConvert

**Link to code**
Wraps the executable command `BRAINSTransformConvert`.

**title:** BRAINS Transform Convert

**category:** Utilities.BRAINS

**description:** Convert ITK transforms to higher order transforms

**version:** 1.0

**documentation-url:** A utility to convert between transform file formats.

**license:** [https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt](https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt)

**contributor:** Hans J. Johnson, Kent Williams, Ali Ghayoor

**Inputs:**

[Optional]

inputTransform: (a pathlike object or string representing an existing
  file)
  argument: `--inputTransform %s`

referenceVolume: (a pathlike object or string representing an
  existing file)
  argument: `--referenceVolume %s`

outputTransformType: ('Affine' or 'VersorRigid' or 'ScaleVersor' or
  'ScaleSkewVersor' or 'DisplacementField' or 'Same')
  The target transformation type. Must be conversion-compatible with
  the input transform type
  argument: `--outputTransformType %s`

outputPrecisionType: ('double' or 'float')
  Precision type of the output transform. It can be either single
  precision or double precision
  argument: `--outputPrecisionType %s`

displacementVolume: (a boolean or a pathlike object or string
  representing a file)
  argument: `--displacementVolume %s`

outputTransform: (a boolean or a pathlike object or string
  representing a file)
  argument: `--outputTransform %s`

args: (a unicode string)
  Additional parameters to the command
  argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})

(continues on next page)
Environment variables

Outputs:

<table>
<thead>
<tr>
<th>displacementVolume</th>
<th>(a pathlike object or string representing an existing file)</th>
</tr>
</thead>
<tbody>
<tr>
<td>outputTransform</td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
</tbody>
</table>

75.21.12 BRAINSTrimForegroundInDirection

Link to code
Wraps the executable command `BRAINSTrimForegroundInDirection`.

title: Trim Foreground In Direction (BRAINS)
category: Utilities.BRAINS
description: This program will trim off the neck and also air-filling noise from the inputImage.
version: 0.1
documentation-url: http://www.nitrc.org/projects/art/

Inputs:

[Optional]

<table>
<thead>
<tr>
<th>inputVolume</th>
<th>(a pathlike object or string representing an existing file)</th>
</tr>
</thead>
<tbody>
<tr>
<td>argument: <code>--inputVolume %s</code></td>
<td></td>
</tr>
<tr>
<td>outputVolume</td>
<td>(a boolean or a pathlike object or string representing a file)</td>
</tr>
<tr>
<td>argument: <code>--outputVolume %s</code></td>
<td></td>
</tr>
<tr>
<td>directionCode</td>
<td>(an integer (int or long))</td>
</tr>
<tr>
<td>argument: <code>--directionCode %d</code></td>
<td></td>
</tr>
<tr>
<td>otsuPercentileThreshold</td>
<td>(a float)</td>
</tr>
<tr>
<td>argument: <code>--otsuPercentileThreshold %f</code></td>
<td></td>
</tr>
<tr>
<td>closingSize</td>
<td>(an integer (int or long))</td>
</tr>
<tr>
<td>argument: <code>--closingSize %d</code></td>
<td></td>
</tr>
<tr>
<td>headSizeLimit</td>
<td>(a float)</td>
</tr>
<tr>
<td>argument: <code>--headSizeLimit %f</code></td>
<td></td>
</tr>
<tr>
<td>BackgroundFillValue</td>
<td>(a unicode string)</td>
</tr>
<tr>
<td>argument: <code>--BackgroundFillValue %s</code></td>
<td></td>
</tr>
<tr>
<td>numberOfThreads</td>
<td>(an integer (int or long))</td>
</tr>
<tr>
<td>argument: <code>--numberOfThreads %d</code></td>
<td></td>
</tr>
<tr>
<td>args</td>
<td>(a unicode string)</td>
</tr>
<tr>
<td>argument: <code>%s</code></td>
<td></td>
</tr>
</tbody>
</table>
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing file)
    Output image with neck and air-filling noise trimmed isotropic image with AC at center of image.

75.21.13 CleanUpOverlapLabels

Link to code
Wraps the executable command "CleanUpOverlapLabels".

title: Clean Up Overla Labels
category: Utilities.BRAINS
description: Take a series of input binary images and clean up for those overlapped area. Binary volumes given first always wins out
version: 0.1.0
contributor: Eun Young Kim

Inputs:

[Optional]
inputBinaryVolumes: (a list of items which are a pathlike object or string representing an existing file)
The list of binary images to be checked and cleaned up. Order is important. Binary volume given first always wins out.
argument: ``--inputBinaryVolumes %s...``

outputBinaryVolumes: (a boolean or a list of items which are a pathlike object or string representing a file)
The output label map images, with integer values in it. Each label value specified in the inputLabels is combined into this output label map volume
argument: ``--outputBinaryVolumes %s...``

args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

outputBinaryVolumes: (a list of items which are a pathlike object or string representing an existing file)
The output label map images, with integer values in it. Each label value specified in the inputLabels is combined into this output label map volume

75.21.14 FindCenterOfBrain

Link to code
Wraps the executable command "FindCenterOfBrain".
title: Center Of Brain (BRAINS)
category: Utilities.BRAINS
description: Finds the center point of a brain
version: 3.0.0
license: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt
acknowledgements: Hans Johnson(1,3,4); Kent Williams(1); (1=University of Iowa Department of Psychiatry, 3=University of Iowa Department of Biomedical Engineering, 4=University of Iowa Department of Electrical and Computer Engineering
Inputs:

[Optional]

inputVolume: (a pathlike object or string representing an existing file)
The image in which to find the center.
argument: `--inputVolume %s`

imageMask: (a pathlike object or string representing an existing file)
argument: `--imageMask %s`

croppedImageMask: (a boolean or a pathlike object or string representing a file)
argument: `--croppedImageMask %s`

maximize: (a boolean)
argument: `--maximize`

axis: (an integer (int or long))
argument: `--axis %d`

otsuPercentileThreshold: (a float)
argument: `--otsuPercentileThreshold %f`

closingSize: (an integer (int or long))
argument: `--closingSize %d`

headSizeLimit: (a float)
argument: `--headSizeLimit %f`

headSizeEstimate: (a float)
argument: `--headSizeEstimate %f`

backgroundValue: (an integer (int or long))
argument: `--backgroundValue %d`

generateDebugImages: (a boolean)
argument: `--generateDebugImages`

distTransformImage: (a boolean or a pathlike object or string representing a file)
argument: `--distTransformImage %s`

gridImage: (a boolean or a pathlike object or string representing a file)
argument: `--gridImage %s`

debugAfterGridComputationsForegroundImage: (a boolean or a pathlike object or string representing a file)
argument: `--debugAfterGridComputationsForegroundImage %s`

croppedImageMask: (a boolean or a pathlike object or string representing a file)
argument: `--croppedImageMask %s`

distTransformImage: (a boolean or a pathlike object or string representing a file)
argument: `--distTransformImage %s`

additionalArgs: (a unicode string)
Additional parameters to the command
argument: `${}`

environ: (a dictionary with keys which are a bytes or None or a value
(continues on next page)
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})

Environment variables

Outputs:

clippedImageMask: (a pathlike object or string representing an
existing file)
debugDistanceImage: (a pathlike object or string representing an
existing file)
debugGridImage: (a pathlike object or string representing an existing
file)
debugAfterGridComputationsForegroundImage: (a pathlike object or
string representing an existing file)
debugClippedImageMask: (a pathlike object or string representing an
existing file)
debugTrimmedImage: (a pathlike object or string representing an
existing file)

75.21.15 GenerateLabelMapFromProbabilityMap

Link to code
Wraps the executable command `GenerateLabelMapFromProbabilityMap`.

title: Label Map from Probability Images
category: Utilities.BRAINS
description: Given a list of probability maps for labels, create a discrete label map where only the highest
probability region is used for the labeling.
version: 0.1
contributor: University of Iowa Department of Psychiatry, http://www.psychiatry.uiowa.edu

Inputs:

[Optional]
inputVolumes: (a list of items which are a pathlike object or string
representing an existing file)
The Input probability images to be computed for lable maps
argument: `--inputVolumes %s...`
outputLabelVolume: (a boolean or a pathlike object or string
representing a file)
The Input binary image for region of interest
argument: `--outputLabelVolume %s`
numberOfThreads: (an integer (int or long))
Explicitly specify the maximum number of threads to use.
argument: `--numberOfThreads %d`
args: (a unicode string)
Additional parameters to the command
argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})

Environment variables

Outputs:

outputLabelVolume: (a pathlike object or string representing an
existing file)
The Input binary image for region of interest
### 75.21.16 ImageRegionPlotter

**Link to code**

Wraps the executable command "ImageRegionPlotter".

**title:** Write Out Image Intensities  
**category:** Utilities.BRAINS  
**description:** For Analysis  
**version:** 0.1  
**contributor:** University of Iowa Department of Psychiatry, [http://www.psychiatry.uiowa.edu](http://www.psychiatry.uiowa.edu)

**Inputs:**

<table>
<thead>
<tr>
<th>Input</th>
<th>Type</th>
<th>Description</th>
<th>Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputVolume1</td>
<td>(a pathlike object or string representing an existing file)</td>
<td>The Input image to be computed for statistics</td>
<td><code>--inputVolume1 %s</code></td>
</tr>
<tr>
<td>inputVolume2</td>
<td>(a pathlike object or string representing an existing file)</td>
<td>The Input image to be computed for statistics</td>
<td><code>--inputVolume2 %s</code></td>
</tr>
<tr>
<td>inputBinaryROIVolume</td>
<td>(a pathlike object or string representing an existing file)</td>
<td>The Input binary image for region of interest</td>
<td><code>--inputBinaryROIVolume %s</code></td>
</tr>
<tr>
<td>inputLabelVolume</td>
<td>(a pathlike object or string representing an existing file)</td>
<td>The Label Image</td>
<td><code>--inputLabelVolume %s</code></td>
</tr>
<tr>
<td>numberOfHistogramBins</td>
<td>(an integer (int or long))</td>
<td>the number of histogram levels</td>
<td><code>--numberOfHistogramBins %d</code></td>
</tr>
<tr>
<td>outputJointHistogramData</td>
<td>(a unicode string)</td>
<td>output data file name</td>
<td><code>--outputJointHistogramData %s</code></td>
</tr>
<tr>
<td>useROIAUTO</td>
<td>(a boolean)</td>
<td>Use ROIAUTO to compute region of interest. This cannot be used with inputLabelVolume</td>
<td><code>--useROIAUTO </code></td>
</tr>
<tr>
<td>useIntensityForHistogram</td>
<td>(a boolean)</td>
<td>Create Intensity Joint Histogram instead of Quantile Joint Histogram</td>
<td><code>--useIntensityForHistogram </code></td>
</tr>
<tr>
<td>verbose</td>
<td>(a boolean)</td>
<td>print debugging information</td>
<td><code>--verbose </code></td>
</tr>
<tr>
<td>args</td>
<td>(a unicode string)</td>
<td>Additional parameters to the command</td>
<td><code>%s</code></td>
</tr>
<tr>
<td>environ</td>
<td>(a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})</td>
<td>Environment variables</td>
<td></td>
</tr>
</tbody>
</table>

**Outputs:**

None
75.21.17 JointHistogram

Link to code
Wraps the executable command "JointHistogram".
title: Write Out Image Intensities
category: Utilities.BRAINS
description: For Analysis
version: 0.1
ccontributor: University of Iowa Department of Psychiatry, http://www.psychiatry.uiowa.edu

Inputs:

- `inputVolumeInXAxis`: (a pathlike object or string representing an existing file)
  - The input image to be computed for statistics
  - argument: ``--inputVolumeInXAxis %s``

- `inputVolumeInYAxis`: (a pathlike object or string representing an existing file)
  - The input image to be computed for statistics
  - argument: ``--inputVolumeInYAxis %s``

- `inputMaskVolumeInXAxis`: (a pathlike object or string representing an existing file)
  - Input mask volume for inputVolumeInXAxis. Histogram will be computed just for the masked region
  - argument: ``--inputMaskVolumeInXAxis %s``

- `inputMaskVolumeInYAxis`: (a pathlike object or string representing an existing file)
  - Input mask volume for inputVolumeInYAxis. Histogram will be computed just for the masked region
  - argument: ``--inputMaskVolumeInYAxis %s``

- `outputJointHistogramImage`: (a unicode string)
  - output joint histogram image file name. Histogram is usually 2D image.
  - argument: ``--outputJointHistogramImage %s``

- `verbose`: (a boolean)
  - print debugging information,
  - argument: ``--verbose``

- `args`: (a unicode string)
  - Additional parameters to the command
  - argument: ``%s``

- `environ`: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

None

75.21.18 ShuffleVectorsModule

Link to code
Wraps the executable command "ShuffleVectorsModule".
title: ShuffleVectors
category: Utilities.BRAINS
description: Automatic Segmentation using neural networks
version: 1.0
license: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt
Contributor: Hans Johnson

Inputs:

[Optional]
inputVectorFileBaseName: (a pathlike object or string representing an existing file)

input vector file name prefix. Usually end with .txt and header file has proot fix of .txt.hdr
argument: ```--inputVectorFileBaseName %s```

outputVectorFileBaseName: (a boolean or a pathlike object or string representing a file)

output vector file name prefix. Usually end with .txt and header file has proot fix of .txt.hdr
argument: ```--outputVectorFileBaseName %s```

resampleProportion: (a float)

downsample size of 1 will be the same size as the input images,
downsample size of 3 will throw 2/3 the vectors away.
argument: ```--resampleProportion %f```

args: (a unicode string)

Additional parameters to the command
argument: ```%s```

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

Environment variables

Outputs:

outputVectorFileBaseName: (a pathlike object or string representing an existing file)

output vector file name prefix. Usually end with .txt and header file has proot fix of .txt.hdr

### 75.21.19 fcsv_to_hdf5

Link to code

Wraps the executable command ``fcsv_to_hdf5``.

title: fcsv_to_hdf5 (BRAINS)
category: Utilities.BRAINS
description: Convert a collection of fcsv files to a HDF5 format file

Inputs:

[Optional]
versionID: (a unicode string)

Current version ID. It should be match with the version of BCD that will be using the output model file,
argument: ```--versionID %s```

landmarksInformationFile: (a boolean or a pathlike object or string representing a file)

name of HDF5 file to write matrices into,
argument: ```--landmarksInformationFile %s```

landmarkTypesList: (a pathlike object or string representing an existing file)

file containing list of landmark types,
argument: ```--landmarkTypesList %s```

modelFile: (a boolean or a pathlike object or string representing a file)

(continues on next page)
, name of HDF5 file containing BRAINSConstellationDetector Model file (LLSMatrices, LLSMeans and LLSSearchRadii),
argument: `--modelFile %s`
landmarkGlobPattern: (a unicode string)
Glob pattern to select fcsv files
argument: `--landmarkGlobPattern %s`
numberOfThreads: (an integer (int or long))
Explicitly specify the maximum number of threads to use.
argument: `--numberOfThreads %d`
args: (a unicode string)
Additional parameters to the command
argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

Outputs:

landmarksInformationFile: (a pathlike object or string representing an existing file)
, name of HDF5 file to write matrices into,
modelFile: (a pathlike object or string representing an existing file)
, name of HDF5 file containing BRAINSConstellationDetector Model file (LLSMatrices, LLSMeans and LLSSearchRadii),

75.21.20 insertMidACPCpoint

Link to code
Wraps the executable command `insertMidACPCpoint`.
title: MidACPC Landmark Insertion
category: Utilities.BRAINS
description: This program gets a landmark fcsv file and adds a new landmark as the midpoint between AC and PC points to the output landmark fcsv file
contributor: Ali Ghayoor

Inputs:

[Optional]
inputLandmarkFile: (a pathlike object or string representing an existing file)
, Input landmark file (.fcsv)
argument: `--inputLandmarkFile %s`
outputLandmarkFile: (a boolean or a pathlike object or string representing a file)
, Output landmark file (.fcsv)
argument: `--outputLandmarkFile %s`
args: (a unicode string)
, Additional parameters to the command
argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

Outputs:
75.21.21 landmarksConstellationAligner

Link to code
Wraps the executable command "landmarksConstellationAligner".

title: MidACPC Landmark Insertion
category: Utilities.BRAINS
description: This program converts the original landmark files to the acpc-aligned landmark files
contributor: Ali Ghayoor

Inputs:

[Optional]
inputLandmarksPaired: (a pathlike object or string representing an existing file)
  Input landmark file (.fcsv)
  argument: `--inputLandmarksPaired %s`
outputLandmarksPaired: (a boolean or a pathlike object or string representing a file)
  Output landmark file (.fcsv)
  argument: `--outputLandmarksPaired %s`
args: (a unicode string)
  Additional parameters to the command
  argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})  
  Environment variables

Outputs:

outputLandmarksPaired: (a pathlike object or string representing an existing file)
  Output landmark file (.fcsv)

75.21.22 landmarksConstellationWeights

Link to code
Wraps the executable command "landmarksConstellationWeights".

title: Generate Landmarks Weights (BRAINS)
category: Utilities.BRAINS
description: Train up a list of Weights for the Landmarks in BRAINSConstellationDetector

Inputs:

[Optional]
inputTrainingList: (a pathlike object or string representing an existing file)
  , Setup file, giving all parameters for training up a Weight list for landmark.,
  argument: `--inputTrainingList %s`
inputTemplateModel: (a pathlike object or string representing an existing file)
  User-specified template model.,
  argument: `--inputTemplateModel %s`

(continues on next page)
LLSModel: (a pathlike object or string representing an existing file)
   Linear least squares model filename in HD5 format
   argument: `--LLSModel %s`
outputWeightsList: (a boolean or a pathlike object or string
   representing a file)
   , The filename of a csv file which is a list of landmarks and their
   corresponding weights.,
   argument: `--outputWeightsList %s`
args: (a unicode string)
   Additional parameters to the command
   argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value
   of class 'str' and with values which are a bytes or None or a
   value of class 'str', nipype default value: {}) Environment variables

Outputs:

outputWeightsList: (a pathlike object or string representing an
existing file)
   , The filename of a csv file which is a list of landmarks and their
   corresponding weights.,
76.1 interfaces.slicer.base

76.1.1 SlicerCommandLine

Link to code

Inputs:

[Optional]
args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})  
  Environment variables

Outputs:

None

76.2 interfaces.slicer.converters

76.2.1 DicomToNrrdConverter

Link to code

Wraps the executable command ``DicomToNrrdConverter``.

title: DICOM to NRRD Converter
category: Converters
description: Converts diffusion weighted MR images in dicom series into Nrrd format for analysis in Slicer. This program has been tested on only a limited subset of DTI dicom formats available from Siemens, GE, and Phillips scanners. Work in progress to support dicom multi-frame data. The program parses dicom header to extract necessary information about measurement frame, diffusion weighting directions, b-values, etc, and write out a nrrd image. For non-diffusion weighted dicom images, it loads in an entire dicom series and writes out a single dicom volume in a .nhdr/.raw pair.
version: 0.2.0.$Revision: 916 $(alpha)
Inputs:

- **inputDicomDirectory**: (a pathlike object or string representing an existing directory)
  - Directory holding Dicom series
  - argument: ``--inputDicomDirectory %s``
- **outputDirectory**: (a boolean or a pathlike object or string representing a directory)
  - Directory holding the output NRRD format
  - argument: ``--outputDirectory %s``
- **outputVolume**: (a unicode string)
  - Output filename (.nhdr or .nrrd)
  - argument: ``--outputVolume %s``
- **smallGradientThreshold**: (a float)
  - If a gradient magnitude is greater than 0 and less than `smallGradientThreshold`, then DicomToNrrdConverter will display an error message and quit, unless the `useBMatrixGradientDirections` option is set.
  - argument: ``--smallGradientThreshold %f``
- **writeProtocolGradientsFile**: (a boolean)
  - Write the protocol gradients to a file suffixed by '.txt' as they were specified in the procoll by multiplying each diffusion gradient direction by the measurement frame. This file is for debugging purposes only, the format is not fixed, and will likely change as debugging of new dicom formats is necessary.
  - argument: ``--writeProtocolGradientsFile``
- **useIdentityMeasurementFrame**: (a boolean)
  - Adjust all the gradients so that the measurement frame is an identity matrix.
  - argument: ``--useIdentityMeasurementFrame``
- **useBMatrixGradientDirections**: (a boolean)
  - Fill the nhdr header with the gradient directions and bvalues computed out of the BMatrix. Only changes behavior for Siemens data.
  - argument: ``--useBMatrixGradientDirections``
- **args**: (a unicode string)
  - Additional parameters to the command
  - argument: ``%s``
- **environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
76.2.2 OrientScalarVolume

Link to code
Wraps the executable command “OrientScalarVolume“.
title: Orient Scalar Volume
category: Converters
description: Orients an output volume. Rearranges the slices in a volume according to the selected orientation. The slices are not interpolated. They are just reordered and/or permuted. The resulting volume will cover the original volume. NOTE: since Slicer takes into account the orientation of a volume, the re-oriented volume will not show any difference from the original volume. To see the difference, save the volume and display it with a system that either ignores the orientation of the image (e.g. Paraview) or displays individual images.
version: 0.1.0.$Revision: 19608 $(alpha)
contributor: Bill Lorensen (GE)
acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

[Optional]
inputVolume1: (a pathlike object or string representing an existing file)
    Input volume 1
    argument: `--%s`, position: -2
outputVolume: (a boolean or a pathlike object or string representing a file)
    The oriented volume
    argument: `--%s`, position: -1
orientation: ('Axial' or 'Coronal' or 'Sagittal' or 'RIP' or 'LIP' or 'RSP' or 'LSP' or 'RIA' or 'LIA' or 'RSA' or 'LSA' or 'IRP' or 'ILP' or 'SRP' or 'SLP' or 'IRA' or 'ILA' or 'RSA' or 'SLA' or 'RPI' or 'LPI' or 'RAI' or 'LAI' or 'RPS' or 'LPS' or 'RAS' or 'LAS' or 'PRI' or 'PLI' or 'ARI' or 'ALI' or 'PRS' or 'PLS' or 'ARS' or 'ALS' or 'IPR' or 'SPI' or 'SPR' or 'IR' or 'SAR' or 'IPL' or 'SPL' or 'IAL' or 'SAL' or 'PRI' or 'PSR' or 'AIR' or 'ASR' or 'PIL' or 'PSL' or 'AIL' or 'ASL')
    Orientation choices
    argument: `--orientation %s`
args: (a unicode string)
    Additional parameters to the command
    argument: `--%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
    Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing file)
    The oriented volume

76.3 interfaces.slicer.diffusion.diffusion

76.3.1 DTIexport

Link to code
Wraps the executable command “DTIexport“.
title: DTIexport
category: Diffusion.Diffusion Data Conversion
description: Export DTI data to various file formats
version: 1.0
ccontributor: Sonia Pujol (SPL, BWH)
acknowledgements: This work is part of the National Alliance for Medical Image Computing (NA-MIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

<table>
<thead>
<tr>
<th>[Optional]</th>
<th>inputTensor: (a pathlike object or string representing an existing file)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Input DTI volume</td>
</tr>
<tr>
<td></td>
<td>argument: <code>$s</code>, position: -2</td>
</tr>
<tr>
<td>outputFile: (a boolean or a pathlike object or string representing a file)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Output DTI file</td>
</tr>
<tr>
<td></td>
<td>argument: <code>$s</code>, position: -1</td>
</tr>
<tr>
<td>args: (a unicode string)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Additional parameters to the command</td>
</tr>
<tr>
<td></td>
<td>argument: <code>$s</code></td>
</tr>
<tr>
<td>environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Environment variables</td>
</tr>
</tbody>
</table>

Outputs:

<table>
<thead>
<tr>
<th>outputFile: (a pathlike object or string representing an existing file)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output DTI file</td>
</tr>
</tbody>
</table>

76.3.2 DTIimport

Link to code
Wraps the executable command "DTIimport".

title: DTIimport
category: Diffusion.Diffusion Data Conversion
description: Import tensor datasets from various formats, including the NifTi file format
version: 1.0
ccontributor: Sonia Pujol (SPL, BWH)
acknowledgements: This work is part of the National Alliance for Medical Image Computing (NA-MIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

<table>
<thead>
<tr>
<th>[Optional]</th>
<th>inputFile: (a pathlike object or string representing an existing file)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Input DTI file</td>
</tr>
<tr>
<td></td>
<td>argument: <code>$s</code>, position: -2</td>
</tr>
<tr>
<td>outputTensor: (a boolean or a pathlike object or string representing a file)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Output DTI volume</td>
</tr>
<tr>
<td></td>
<td>argument: <code>$s</code>, position: -1</td>
</tr>
</tbody>
</table>

(continues on next page)
testingmode: (a boolean)

Enable testing mode. Sample helix file (helix-DTI.nhdr) will be
loaded into Slicer and converted in Nifti.
argument: ``--testingmode``
args: (a unicode string)
Additional parameters to the command
argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

outputTensor: (a pathlike object or string representing an existing
file)
Output DTI volume

76.3.3 DWIJointRicianLMMSEFilter

Link to code
Wraps the executable command `"DWIJointRicianLMMSEFilter"`.

title: DWI Joint Rician LMMSE Filter
category: Diffusion.Diffusion Weighted Images
description: This module reduces Rician noise (or unwanted detail) on a set of diffusion weighted images. For this, it filters the image in the mean squared error sense using a Rician noise model. The N closest gradient directions to the direction being processed are filtered together to improve the results: the noise-free signal is seen as an n-dimensional vector which has to be estimated with the LMMSE method from a set of corrupted measurements. To that end, the covariance matrix of the noise-free vector and the cross covariance between this signal and the noise have to be estimated, which is done taking into account the image formation process. The noise parameter is automatically estimated from a rough segmentation of the background of the image. In this area the signal is simply 0, so that Rician statistics reduce to Rayleigh and the noise power can be easily estimated from the mode of the histogram. A complete description of the algorithm may be found in: Antonio Tristan-Vega and Santiago Aja-Fernandez, DWI filtering using joint information for DTI and HARDI, Medical Image Analysis, Volume 14, Issue 2, Pages 205-218. 2010.
version: 0.1.1.$Revision: 1 $(alpha)
contributor: Antonio Tristan Vega (UVa), Santiago Aja Fernandez (UVa)
acknowledgements: Partially founded by grant number TEC2007-67073/TCM from the Comision Interministerial de Ciencia y Tecnologia (Spain).

Inputs:

[Optional]
re: (a list of items which are an integer (int or long))
  Estimation radius.
  argument: `--re %s`
rf: (a list of items which are an integer (int or long))
  Filtering radius.
  argument: `--rf %s`
ng: (an integer (int or long))
  The number of the closest gradients that are used to jointly filter
  a given gradient direction (0 to use all).
  argument: `--ng %d`
inputVolume: (a pathlike object or string representing an existing
  (continues on next page)
Input DWI volume.
argument: ``%s``, position: -2

Output DWI volume.
argument: ``%s``, position: -1

Compress the data of the compressed file using gzip
argument: ``--compressOutput``

Additional parameters to the command
argument: ``%s``

Environment variables

Output DWI volume.

76.3.4 DWIRicianLMMSEFilter

Link to code
Wraps the executable command "DWIRicianLMMSEFilter".

title: DWI Rician LMMSE Filter
category: Diffusion.Diffusion Weighted Images
description: This module reduces noise (or unwanted detail) on a set of diffusion weighted images. For this, it filters the image in the mean squared error sense using a Rician noise model. Images corresponding to each gradient direction, including baseline, are processed individually. The noise parameter is automatically estimated (noise estimation improved but slower). Note that this is a general purpose filter for MRI images. The module jointLMMSE has been specifically designed for DWI volumes and shows a better performance, so its use is recommended instead. A complete description of the algorithm in this module can be found in: S. Aja-Fernandez, M. Niethammer, M. Kubicki, M. Shenton, and C.-F. Westin. Restoration of DWI data using a Rician LMMSE estimator. IEEE Transactions on Medical Imaging, 27(10): pp. 1389-1403, Oct. 2008.
version: 0.1.1.$Revision: 1 $(alpha)
contributor: Antonio Tristan Vega (UVa), Santiago Aja Fernandez (UVa), Marc Niethammer (UNC)
acknowledgements: Partially founded by grant number TEC2007-67073/TCM from the Comision Interministerial de Ciencia y Tecnologia (Spain).

Inputs:

[Optional]
iter: (an integer (int or long))
argument: ``--iter %d``

re: (a list of items which are an integer (int or long))
argument: ``--re %s``

rf: (a list of items which are an integer (int or long))
argument: ``--rf %s``
nipype Documentation, Release 1.2.1

(continued from previous page)

argument: ``--rf %s``
mnvf: (an integer (int or long))
  Minimum number of voxels in kernel used for filtering.
  argument: ``--mnvf %d``
mnve: (an integer (int or long))
  Minimum number of voxels in kernel used for estimation.
  argument: ``--mnve %d``
minnstd: (an integer (int or long))
  Minimum allowed noise standard deviation.
  argument: ``--minnstd %d``
maxnstd: (an integer (int or long))
  Maximum allowed noise standard deviation.
  argument: ``--maxnstd %d``
hrf: (a float)
  How many histogram bins per unit interval.
  argument: ``--hrf %f``
uav: (a boolean)
  Use absolute value in case of negative square.
  argument: ``--uav ``

inputVolume: (a pathlike object or string representing an existing file)
  Input DWI volume.
  argument: ``%s``, position: -2
outputVolume: (a boolean or a pathlike object or string representing a file)
  Output DWI volume.
  argument: ``%s``, position: -1
compressOutput: (a boolean)
  Compress the data of the compressed file using gzip
  argument: ``--compressOutput ```
args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
  Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing file)
  Output DWI volume.

76.3.5 DWIToDTIESTIMATION

Link to code

Wraps the executable command ``DWIToDTIESTIMATION``.

title: DWI to DTI Estimation
category: Diffusion.Diffusion Weighted Images
description: Performs a tensor model estimation from diffusion weighted images.
There are three estimation methods available: least squares, weighted least squares and non-linear estimation.
The first method is the traditional method for tensor estimation and the fastest one. Weighted least squares takes
into account the noise characteristics of the MRI images to weight the DWI samples used in the estimation based
on its intensity magnitude. The last method is the more complex.
version: 0.1.0.$Revision: 1892 $(alpha)
DiffusionTensorEstimation

license: slicer3

contributor: Raul San Jose (SPL, BWH)

acknowledgements: This command module is based on the estimation functionality provided by the Teem library. This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

[Optional]

inputVolume: (a pathlike object or string representing an existing file)

Input DWI volume

argument: ``%s``, position: -3

mask: (a pathlike object or string representing an existing file)

Mask where the tensors will be computed

argument: ``--mask %s``

outputTensor: (a boolean or a pathlike object or string representing a file)

Estimated DTI volume

argument: ``%s``, position: -2

outputBaseline: (a boolean or a pathlike object or string representing a file)

Estimated baseline volume

argument: ``%s``, position: -1

enumeration: ('LS' or 'WLS')

LS: Least Squares, WLS: Weighted Least Squares

argument: ``--enumeration %s``

shiftNeg: (a boolean)

Shift eigenvalues so all are positive (accounts for bad tensors related to noise or acquisition error)

argument: ``--shiftNeg``

args: (a unicode string)

Additional parameters to the command

argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})  

Environment variables

Outputs:

outputTensor: (a pathlike object or string representing an existing file)

Estimated DTI volume

outputBaseline: (a pathlike object or string representing an existing file)

Estimated baseline volume

76.3.6 DiffusionTensorScalarMeasurements

Link to code

Wraps the executable command “DiffusionTensorScalarMeasurements “.

title: Diffusion Tensor Scalar Measurements

category: Diffusion.Diffusion Tensor Images

description: Compute a set of different scalar measurements from a tensor field, specially oriented for Diffusion Tensors where some rotationally invariant measurements, like Fractional Anisotropy, are highly used to describe the anisotropic behaviour of the tensor.
76.3.7 DiffusionWeightedVolumeMasking

Link to code
Wraps the executable command "DiffusionWeightedVolumeMasking".

title: Diffusion Weighted Volume Masking
category: Diffusion.Diffusion Weighted Images
description: Performs a mask calculation from a diffusion weighted (DW) image. Starting from a dw image, this module computes the baseline image averaging all the images without diffusion weighting and then applies the otsu segmentation algorithm in order to produce a mask. this mask can then be used when estimating the diffusion tensor (dt) image, not to estimate tensors all over the volume.

version: 0.1.0.$Revision: 1892 $(alpha)
license: slicer3
contributor: Demian Wassermann (SPL, BWH)

Inputs:

outputScalar: (a pathlike object or string representing an existing file)
    Scalar volume derived from tensor

Outputs:

outputScalar: (a pathlike object or string representing an existing file)
    Scalar volume derived from tensor

Scalar volume derived from tensor argument: ``%s``, position: -1
enumeration: ('Trace' or 'Determinant' or 'RelativeAnisotropy' or 'FractionalAnisotropy' or 'Mode' or 'LinearMeasure' or 'PlanarMeasure' or 'SphericalMeasure' or 'MinEigenvalue' or 'MidEigenvalue' or 'MaxEigenvalue' or 'MaxEigenvalueProjectionX' or 'MaxEigenvalueProjectionY' or 'MaxEigenvalueProjectionZ' or 'RAIMaxEigenvecX' or 'RAIMaxEigenvecY' or 'RAIMaxEigenvecZ' or 'MaxEigenvecX' or 'MaxEigenvecY' or 'MaxEigenvecZ' or 'D11' or 'D22' or 'D33' or 'ParallelDiffusivity' or 'PerpendicularDiffusivity')
    An enumeration of strings argument: `--enumeration %s`

args: (a unicode string)
    Additional parameters to the command argument: `%(args)"

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
    Environment variables
76.3.8 ResampleDTIVolume

Link to code
Wraps the executable command "ResampleDTIVolume".

title: Resample DTI Volume
category: Diffusion.Diffusion Tensor Images
description: Resampling an image is a very important task in image analysis. It is especially important in the frame of image registration. This module implements DT image resampling through the use of itk Transforms. The resampling is controlled by the Output Spacing. “Resampling” is performed in space coordinates, not pixel/grid coordinates. It is quite important to ensure that image spacing is properly set on the images involved. The interpolator is required since the mapping from one space to the other will often require evaluation of the intensity of the image at non-grid positions.

version: 0.1
dontribution: slicer

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149. Information on the National Centers for Biomedical Computing can be obtained from http://nihroadmap.nih.gov/bioinformatics

Inputs:

[Optional]
inputVolume: (a pathlike object or string representing an existing file)
  Input DWI volume
  argument: ``%s``, position: -4
outputBaseline: (a boolean or a pathlike object or string representing a file)
  Estimated baseline volume
  argument: ``%s``, position: -2
thresholdMask: (a boolean or a pathlike object or string representing a file)
  Otsu Threshold Mask
  argument: ``%s``, position: -1
otsuomegathreshold: (a float)
  Control the sharpness of the threshold in the Otsu computation. 0: lower threshold, 1: higher threshold
  argument: ``--otsuomegathreshold %f``
removeislands: (a boolean)
  Remove Islands in Threshold Mask?
  argument: ``--removeislands ``
args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  Environment variables

Outputs:

outputBaseline: (a pathlike object or string representing an existing file)
  Estimated baseline volume
thresholdMask: (a pathlike object or string representing an existing file)
  Otsu Threshold Mask
inputVolume: (a pathlike object or string representing an existing file)
   Input volume to be resampled
   argument: "-s", position: -2
outputVolume: (a boolean or a pathlike object or string representing a file)
   Resampled Volume
   argument: "-s", position: -1
Reference: (a pathlike object or string representing an existing file)
   Reference Volume (spacing, size, orientation, origin)
   argument: "--Reference %s"
transformationFile: (a pathlike object or string representing an existing file)
   argument: "--transformationFile %s"
defField: (a pathlike object or string representing an existing file)
   File containing the deformation field (3D vector image containing vectors with 3 components)
   argument: "--defField %s"
hfieldtype: ('displacement' or 'h-Field')
   Set if the deformation field is an -Field
   argument: "--hfieldtype %s"
interpolation: ('linear' or 'nn' or 'ws' or 'bs')
   Sampling algorithm (linear, nn (nearest neighborhor), ws (WindowedSinc), bs (BSpline))
   argument: "--interpolation %s"
correction: ('zero' or 'none' or 'abs' or 'nearest')
   Correct the tensors if computed tensor is not semi-definite positive
   argument: "--correction %s"
transform_tensor_method: ('PPD' or 'FS')
   Chooses between 2 methods to transform the tensors: Finite Strain (FS), faster but less accurate, or Preservation of the Principal Direction (PPD)
   argument: "--transform_tensor_method %s"
transform_order: ('input-to-output' or 'output-to-input')
   Select in what order the transforms are read
   argument: "--transform_order %s"
notbulk: (a boolean)
   The transform following the BSpline transform is not set as a bulk transform for the BSpline transform
   argument: "--notbulk"
spaceChange: (a boolean)
   Space Orientation between transform and image is different (RAS/LPS)
   (warning: if the transform is a Transform Node in Slicer3, do not select)
   argument: "--spaceChange"
rotation_point: (a list of items which are any value)
   Center of rotation (only for rigid and affine transforms)
   argument: "--rotation_point %s"
centered_transform: (a boolean)
   Set the center of the transformation to the center of the input image (only for rigid and affine transforms)
   argument: "--centered_transform"
image_center: ('input' or 'output')
   Image to use to center the transform (used only if 'Centered Transform' is selected)
argument: `--image_center %s`

Inverse_ITK_Transformation: (a boolean)
Inverse the transformation before applying it from output image to
input image (only for rigid and affine transforms)
argument: `--Inverse_ITK_Transformation`

spacing: (a list of items which are a float)
Spacing along each dimension (0 means use input spacing)
argument: `--spacing %s`

type: (a list of items which are a float)
Size along each dimension (0 means use input size)
argument: `--size %s`

origin: (a list of items which are any value)
Origin of the output Image
argument: `--origin %s`

direction_matrix: (a list of items which are a float)
9 parameters of the direction matrix by rows (ijk to LPS if LPS
transform, ijk to RAS if RAS transform)
argument: `--direction_matrix %s`

number_of_thread: (an integer (int or long))
Number of thread used to compute the output image
argument: `--number_of_thread %d`

default_pixel_value: (a float)
Default pixel value for samples falling outside of the input region
argument: `--default_pixel_value %f`

window_function: (`h` or `c` or `w` or `l` or `b`)
Window Function , h = Hamming , c = Cosine , w = Welch , l = Lanczos
, b = Blackman
argument: `--window_function %s`

spline_order: (an integer (int or long))
Spline Order (Spline order may be from 0 to 5)
argument: `--spline_order %d`

transform_matrix: (a list of items which are a float)
12 parameters of the transform matrix by rows ( --last 3 being
translation-- )
argument: `--transform_matrix %s`

type: (`rt` or `a`)
Transform algorithm, rt = Rigid Transform, a = Affine Transform
argument: `--transform %s`

args: (a unicode string)
Additional parameters to the command
argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value
class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: { })
Environment variables

Output:

outputVolume: (a pathlike object or string representing an existing
file)
Resampled Volume

76.3.9 TractographyLabelMapSeeding

Link to code
Wraps the executable command "TractographyLabelMapSeeding".
title: Tractography Label Map Seeding
category: Diffusion.Diffusion Tensor Images
description: Seed tracts on a Diffusion Tensor Image (DT) from a label map
version: 0.1.0.$Revision: 1892 $(alpha)
license: slicer3
contributor: Raul San Jose (SPL, BWH), Demian Wassermann (SPL, BWH)
aknowledgements: Laboratory of Mathematics in Imaging. This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

[Optional]
InputVolume: (a pathlike object or string representing an existing file)
Input DTI volume
argument: `--inputvolume %s`, position: -2
inputroi: (a pathlike object or string representing an existing file)
Label map with seeding ROIs
argument: `--inputroi %s`

OutputFibers: (a boolean or a pathlike object or string representing a file)
Tractography result
argument: `--outputfibers %s`, position: -1

useindexspace: (a boolean)
Seed at IJK voxel grid
argument: `--useindexspace`

seedspace: (a float)
Spacing (in mm) between seed points, only matters if use Use Index Space is off
argument: `--seedspace %f`

randomgrid: (a boolean)
Enable random placing of seeds
argument: `--randomgrid`

clthreshold: (a float)
Minimum Linear Measure for the seeding to start.
argument: `--clthreshold %f`

minimumlength: (a float)
Minimum length of the fibers (in mm)
argument: `--minimumlength %f`

maximumlength: (a float)
Maximum length of fibers (in mm)
argument: `--maximumlength %f`

stoppingmode: ('LinearMeasure' or 'FractionalAnisotropy')
Tensor measurement used to stop the tractography
argument: `--stoppingmode %s`

stoppingvalue: (a float)
Tractography will stop when the stopping measurement drops below this value
argument: `--stoppingvalue %f`

stoppingcurvature: (a float)
Tractography will stop if radius of curvature becomes smaller than this number units are degrees per mm
argument: `--stoppingcurvature %f`

integrationsteplength: (a float)
Distance between points on the same fiber in mm
argument: `--integrationsteplength %f`

label: (an integer (int or long))

(continues on next page)
Label value that defines seeding region.
argument: `--label %d`

writetofile: (a boolean)
Write fibers to disk or create in the scene?
argument: `--writetofile`

outputdirectory: (a boolean or a pathlike object or string representing a directory)
Directory in which to save fiber(s)
argument: `--outputdirectory %s`

name: (a unicode string)
Name to use for fiber files
argument: `--name %s`

args: (a unicode string)
Additional parameters to the command
argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

OutputFibers: (a pathlike object or string representing an existing file)
Tractography result

outputdirectory: (a pathlike object or string representing an existing directory)
Directory in which to save fiber(s)

76.4 interfaces.slicer.filtering.arithmetic

76.4.1 AddScalarVolumes

Link to code
Wraps the executable command "AddScalarVolumes".

title: Add Scalar Volumes
category: Filtering.Arithmetic
description: Adds two images. Although all image types are supported on input, only signed types are produced. The two images do not have to have the same dimensions.
version: 0.1.0.$Revision: 19608 $(alpha)
ccontributor: Bill Lorensen (GE)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

[Optional]
inputVolume1: (a pathlike object or string representing an existing file)
Input volume 1
argument: `%s`, position: -3

inputVolume2: (a pathlike object or string representing an existing file)
Input volume 2
argument: `%s`, position: -2
outputVolume: (a boolean or a pathlike object or string representing a file)
    Volume1 + Volume2
argument: `\"%s\"`, position: -1
order: (`'0'` or `'1'` or `'2'` or `'3'`)
    Interpolation order if two images are in different coordinate frames or have different sampling.
argument: `\"--order %s\"`
args: (a unicode string)
    Additional parameters to the command
argument: `\"%s\``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: `{}`)
    Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing file)
    Volume1 + Volume2

76.4.2 CastScalarVolume

Link to code
Wraps the executable command `CastScalarVolume`.

title: Cast Scalar Volume
category: Filtering.Arithmetic
description: Cast a volume to a given data type. Use at your own risk when casting an input volume into a lower precision type! Allows casting to the same type as the input volume.
version: 0.1.0.$Revision: 2104 $(alpha)
contributor: Nicole Aucoin (SPL, BWH), Ron Kikinis (SPL, BWH)
acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

[Optional]
InputVolume: (a pathlike object or string representing an existing file)
    Input volume, the volume to cast.
argument: `\"%s\```, position: -2
OutputVolume: (a boolean or a pathlike object or string representing a file)
    Output volume, cast to the new type.
argument: `\"%s\```, position: -1
type: (`'Char'` or `'UnsignedChar'` or `'Short'` or `'UnsignedShort'` or `'Int'` or `'UnsignedInt'` or `'Float'` or `'Double'`)
    Type for the new output volume.
argument: `\"--type %s\``
args: (a unicode string)
    Additional parameters to the command
argument: `\"%s\``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: `{}`)
76.4.3 MaskScalarVolume

Link to code

Wraps the executable command "MaskScalarVolume".

title: Mask Scalar Volume
category: Filtering.Arithmetic
description: Masks two images. The output image is set to 0 everywhere except where the chosen label from the mask volume is present, at which point it will retain its original values. Although all image types are supported on input, only signed types are produced. The two images do not have to have the same dimensions.

version: 0.1.0.$Revision: 8595 $(alpha)
contributor: Nicole Aucoin (SPL, BWH), Ron Kikinis (SPL, BWH)
acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

[Optional]
InputVolume: (a pathlike object or string representing an existing file)
  Input volume to be masked
  argument: ``%s``, position: -3

MaskVolume: (a pathlike object or string representing an existing file)
  Label volume containing the mask
  argument: ``%s``, position: -2

OutputVolume: (a boolean or a pathlike object or string representing a file)
  Output volume: Input Volume masked by label value from Mask Volume
  argument: ``%s``, position: -1

label: (an integer (int or long))
  Label value in the Mask Volume to use as the mask
  argument: ``--label %d``

replace: (an integer (int or long))
  Value to use for the output volume outside of the mask
  argument: ``--replace %d``

args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  Environment variables

Outputs:

OutputVolume: (a pathlike object or string representing an existing file)
  Output volume: Input Volume masked by label value from Mask Volume
76.4.4 MultiplyScalarVolumes

Link to code
Wraps the executable command “MultiplyScalarVolumes “.

title: Multiply Scalar Volumes
category: Filtering.Arithmetic
description: Multiplies two images. Although all image types are supported on input, only signed types are produced. The two images do not have to have the same dimensions.
version: 0.1.0.$Revision: 8595 $(alpha)
contributor: Bill Lorensen (GE)
acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

<table>
<thead>
<tr>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputVolume1: (a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>Input volume 1</td>
</tr>
<tr>
<td>argument: <code>%-s</code>, position: -3</td>
</tr>
<tr>
<td>inputVolume2: (a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>Input volume 2</td>
</tr>
<tr>
<td>argument: <code>%-s</code>, position: -2</td>
</tr>
<tr>
<td>outputVolume: (a boolean or a pathlike object or string representing a file)</td>
</tr>
<tr>
<td>Volume1 * Volume2</td>
</tr>
<tr>
<td>argument: <code>%-s</code>, position: -1</td>
</tr>
<tr>
<td>order: (`0' or '1' or '2' or '3')</td>
</tr>
<tr>
<td>Interpolation order if two images are in different coordinate frames or have different sampling.</td>
</tr>
<tr>
<td>argument: <code>--order %s</code></td>
</tr>
<tr>
<td>args: (a unicode string)</td>
</tr>
<tr>
<td>Additional parameters to the command</td>
</tr>
<tr>
<td>argument: <code>%-s</code></td>
</tr>
<tr>
<td>environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}}</td>
</tr>
<tr>
<td>Environment variables</td>
</tr>
</tbody>
</table>

Outputs:

| outputVolume: (a pathlike object or string representing an existing file) |
| Volume1 * Volume2 |

76.4.5 SubtractScalarVolumes

Link to code
Wraps the executable command “SubtractScalarVolumes “.

title: Subtract Scalar Volumes
category: Filtering.Arithmetic
description: Subtracts two images. Although all image types are supported on input, only signed types are produced. The two images do not have to have the same dimensions.
version: 0.1.0.$Revision: 19608 $(alpha)
contributor: Bill Lorensen (GE)
acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```
[Optional]
inputVolume1: (a pathlike object or string representing an existing file)
    Input volume 1
    argument: ``%s``, position: -3
inputVolume2: (a pathlike object or string representing an existing file)
    Input volume 2
    argument: ``%s``, position: -2
outputVolume: (a boolean or a pathlike object or string representing a file)
    Volume1 - Volume2
    argument: ``%s``, position: -1
order: ('0' or '1' or '2' or '3')
    Interpolation order if two images are in different coordinate frames or have different sampling.
    argument: ``--order %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
    Environment variables
```

Outputs:

```
outputVolume: (a pathlike object or string representing an existing file)
    Volume1 - Volume2
```

### 76.5 interfaces.slicer.filtering.checkerboardfilter

#### 76.5.1 CheckerBoardFilter

Link to code
Wraps the executable command "CheckerBoardFilter".

title: CheckerBoard Filter
category: Filtering
description: Create a checkerboard volume of two volumes. The output volume will show the two inputs alternating according to the user supplied checkerPattern. This filter is often used to compare the results of image registration. Note that the second input is resampled to the same origin, spacing and direction before it is composed with the first input. The scalar type of the output volume will be the same as the input image scalar type.
version: 0.1.0.$Revision: 19608 $(alpha)
ccontributor: Bill Lorensen (GE)
acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.
[Optional]

checkerPattern: (a list of items which are an integer (int or long))
The pattern of input 1 and input 2 in the output image. The user can specify the number of checkers in each dimension. A checkerPattern of 2,2,1 means that images will alternate in every other checker in the first two dimensions. The same pattern will be used in the 3rd dimension.
argument: `--checkerPattern %s`

inputVolume1: (a pathlike object or string representing an existing file)
First Input volume
argument: `-%s`, position: -3

inputVolume2: (a pathlike object or string representing an existing file)
Second Input volume
argument: `-%s`, position: -2

outputVolume: (a boolean or a pathlike object or string representing a file)
Output filtered
argument: `-%s`, position: -1

args: (a unicode string)
Additional parameters to the command
argument: `-%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing file)
Output filtered

---

### 76.6 `interfaces.slicer.filtering.denoising`

#### 76.6.1 CurvatureAnisotropicDiffusion

Link to code
Wraps the executable command "CurvatureAnisotropicDiffusion".

title: Curvature Anisotropic Diffusion
category: Filtering.Denoising
description: Performs anisotropic diffusion on an image using a modified curvature diffusion equation (MCDE).
MCDE does not exhibit the edge enhancing properties of classic anisotropic diffusion, which can under certain conditions undergo a ‘negative’ diffusion, which enhances the contrast of edges. Equations of the form of MCDE always undergo positive diffusion, with the conductance term only varying the strength of that diffusion.
Qualitatively, MCDE compares well with other non-linear diffusion techniques. It is less sensitive to contrast than classic Perona-Malik style diffusion, and preserves finer detailed structures in images.
There is a potential speed trade-off for using this function in place of Gradient Anisotropic Diffusion.
Each iteration of the solution takes roughly twice as long. Fewer iterations, however, may be required to reach an acceptable solution.

version: 0.1.0 $Revision: 19608 $(alpha)
contributor: Bill Lorensen (GE)
acknowledgements: This command module was derived from Insight/Examples (copyright) Insight Software
Consortium

Inputs:

[Optional]
condu%40ntance: (a float)
Conductance controls the sensitivity of the conductance term. As a
general rule, the lower the value, the more strongly the filter
preserves edges. A high value will cause diffusion (smoothing)
across edges. Note that the number of iterations controls how much
smoothing is done within regions bounded by edges.
argument: `--conductance %f`

iterations: (an integer (int or long))
The more iterations, the more smoothing. Each iteration takes the
same amount of time. If it takes 10 seconds for one iteration, then
it will take 100 seconds for 10 iterations. Note that the
conductance controls how much each iteration smooths across edges.
argument: `--iterations %d`

timeStep: (a float)
The time step depends on the dimensionality of the image. In Slicer
the images are 3D and the default (.0625) time step will provide a
stable solution.
argument: `--timeStep %f`

inputVolume: (a pathlike object or string representing an existing
file)
Input volume to be filtered
argument: `%s`, position: -2

outputVolume: (a boolean or a pathlike object or string representing
a file)
Output filtered
argument: `%s`, position: -1

args: (a unicode string)
Additional parameters to the command
argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing
file)
Output filtered

76.6.2 GaussianBlurImageFilter

Link to code
Wraps the executable command "GaussianBlurImageFilter".
title: Gaussian Blur Image Filter
category: Filtering.Denoising
description: Apply a gaussian blur to an image
version: 0.1.0.$Revision: 1.1 $(alpha)
GaussianBlurImageFilter
contributor: Julien Jomier (Kitware), Stephen Aylward (Kitware)
acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded
by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.
Inputs:
76.6.3 GradientAnisotropicDiffusion

Link to code
Wraps the executable command ‘GradientAnisotropicDiffusion’.

title: Gradient Anisotropic Diffusion
category: Filtering.Denoising
description: Runs gradient anisotropic diffusion on a volume.
Anisotropic diffusion methods reduce noise (or unwanted detail) in images while preserving specific image features, like edges. For many applications, there is an assumption that light-dark transitions (edges) are interesting. Standard isotropic diffusion methods move and blur light-dark boundaries. Anisotropic diffusion methods are formulated to specifically preserve edges. The conductance term for this implementation is a function of the gradient magnitude of the image at each point, reducing the strength of diffusion at edges. The numerical implementation of this equation is similar to that described in the Perona-Malik paper, but uses a more robust technique for gradient magnitude estimation and has been generalized to N-dimensions.

version: 0.1.0.


contributor: Bill Lorensen (GE)

acknowledgements: This command module was derived from Insight/Examples (copyright) Insight Software Consortium

Inputs:

[Optional]
conductance: (a float)
Conductance controls the sensitivity of the conductance term. As a general rule, the lower the value, the more strongly the filter preserves edges. A high value will cause diffusion (smoothing) across edges. Note that the number of iterations controls how much smoothing is done within regions bounded by edges.
argument: `--conductance %f`
iterations: (an integer (int or long))
The more iterations, the more smoothing. Each iteration takes the same amount of time. If it takes 10 seconds for one iteration, then it will take 100 seconds for 10 iterations. Note that the conductance controls how much each iteration smooths across edges.
argument: `--iterations %d`

timeStep: (a float)
The time step depends on the dimensionality of the image. In Slicer the images are 3D and the default (0.0625) time step will provide a stable solution.
argument: `--timeStep %f`

inputVolume: (a pathlike object or string representing an existing file)
Input volume to be filtered
argument: `%s`, position: -2

outputVolume: (a boolean or a pathlike object or string representing a file)
Output filtered
argument: `%s`, position: -1

args: (a unicode string)
Additional parameters to the command
argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing file)
Output filtered

76.6.4 MedianImageFilter

Link to code
Wraps the executable command "MedianImageFilter".

title: Median Image Filter
category: Filtering.Denoising
description: The MedianImageFilter is commonly used as a robust approach for noise reduction. This filter is particularly efficient against "salt-and-pepper" noise. In other words, it is robust to the presence of gray-level outliers. MedianImageFilter computes the value of each output pixel as the statistical median of the neighborhood of values around the corresponding input pixel.
version: 0.1.0.$Revision: 19608 $(alpha)
contributor: Bill Lorensen (GE)
acknowledgements: This command module was derived from Insight/Examples/Filtering/MedianImageFilter (copyright) Insight Software Consortium

Inputs:

[Optional]
neighborhood: (a list of items which are an integer (int or long))
The size of the neighborhood in each dimension
argument: `--neighborhood %s`

inputVolume: (a pathlike object or string representing an existing file)
Input volume to be filtered
argument: `"%s"`, position: -2
outputVolume: (a boolean or a pathlike object or string representing
a file)
Output filtered
argument: `"%s"`, position: -1
args: (a unicode string)
Additional parameters to the command
argument: `"%s``
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {}))
Environment variables

Outputs:

Output filtered

76.7 interfaces.slicer.filtering.extractskeleton

76.7.1 ExtractSkeleton

Link to code
Wraps the executable command "ExtractSkeleton".
title: Extract Skeleton
category: Filtering
description: Extract the skeleton of a binary object. The skeleton can be limited to being a 1D curve or allowed
to be a full 2D manifold. The branches of the skeleton can be pruned so that only the maximal center skeleton
is returned.
version: 0.1.0.$Revision: 2104 $(alpha)
contributor: Pierre Seroul (UNC), Martin Styner (UNC), Guido Gerig (UNC), Stephen Aylward (Kitware)
acknowledgements: The original implementation of this method was provided by ETH Zurich, Image Analysis
Laboratory of Profs Olaf Kuebler, Gabor Szekely and Guido Gerig. Martin Styner at UNC, Chapel Hill made
enhancements. Wrapping for Slicer was provided by Pierre Seroul and Stephen Aylward at Kitware, Inc.
Inputs:

[Optional]
InputImageFileName: (a pathlike object or string representing an
existing file)
Input image
argument: `"%s"`, position: -2
OutputImageFileName: (a boolean or a pathlike object or string
representing a file)
Skeleton of the input image
argument: `"%s"`, position: -1
type: ('1D' or '2D')
Type of skeleton to create
argument: `"--type %s``
dontPrune: (a boolean)
Return the full skeleton, not just the maximal skeleton
argument: `"--dontPrune ``

(continues on next page)
numPoints: (an integer (int or long))
    Number of points used to represent the skeleton
    argument: `'--numPoints %d'`
pointsFile: (a unicode string)
    Name of the file to store the coordinates of the central (1D)
    skeleton points
    argument: `'--pointsFile %s'`
args: (a unicode string)
    Additional parameters to the command
    argument: `''`
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipype default value: {})  
    Environment variables

Outputs:

OutputImageFileName: (a pathlike object or string representing an
    existing file)
    Skeleton of the input image

76.8 interfaces.slicer.filtering.histogrammatching

76.8.1 HistogramMatching

Link to code
Wraps the executable command `"HistogramMatching"`.
title: Histogram Matching
category: Filtering
description: Normalizes the grayscale values of a source image based on the grayscale values of a reference
    image. This filter uses a histogram matching technique where the histograms of the two images are matched
    only at a specified number of quantile values.
The filter was originally designed to normalize MR images of the same MR protocol and same body part. The
algorithm works best if background pixels are excluded from both the source and reference histograms. A
simple background exclusion method is to exclude all pixels whose grayscale values are smaller than the mean
grayscale value. ThresholdAtMeanIntensity switches on this simple background exclusion method.
Number of match points governs the number of quantile values to be matched.
The filter assumes that both the source and reference are of the same type and that the input and output image
    type have the same number of dimension and have scalar pixel types.
version: 0.1.0.$Revision: 19608 $(alpha)
    HistogramMatching
contributor: Bill Lorensen (GE)
acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded
    by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.
Inputs:

[Optional]
numberOfHistogramLevels: (an integer (int or long))
    The number of histogram levels to use
    argument: `'--numberOfHistogramLevels %d'`
numberOfMatchPoints: (an integer (int or long))
    The number of match points to use
    argument: `'--numberOfMatchPoints %d'`
threshold: (a boolean)
  If on, only pixels above the mean in each volume are thresholded.
  argument: `--threshold`

inputVolume: (a pathlike object or string representing an existing file)
  Input volume to be filtered
  argument: `%s`, position: -3

referenceVolume: (a pathlike object or string representing an existing file)
  Input volume whose histogram will be matched
  argument: `%s`, position: -2

outputVolume: (a boolean or a pathlike object or string representing a file)
  Output volume. This is the input volume with intensities matched to the reference volume.
  argument: `%s`, position: -1

args: (a unicode string)
  Additional parameters to the command
  argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing file)
  Output volume. This is the input volume with intensities matched to the reference volume.

76.9 interfaces.slicer.filtering.imagelabelcombine

76.9.1 ImageLabelCombine

Link to code
Wraps the executable command "ImageLabelCombine".

title: Image Label Combine

category: Filtering
description: Combine two label maps into one

version: 0.1.0

contributor: Alex Yarmarkovich (SPL, BWH)

Inputs:

[Optional]
InputLabelMap_A: (a pathlike object or string representing an existing file)
  Label map image
  argument: ` `%s``, position: -3

InputLabelMap_B: (a pathlike object or string representing an existing file)
  Label map image
  argument: ` `%s``, position: -2

(continues on next page)
OutputLabelMap: (a boolean or a pathlike object or string representing a file)
Resulting Label map image
argument: `--%s`, position: -1
first_overwrites: (a boolean)
Use first or second label when both are present
argument: `--first_overwrites`
args: (a unicode string)
Additional parameters to the command
argument: `--%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

OutputLabelMap: (a pathlike object or string representing an existing file)
Resulting Label map image

## 76.10 interfaces.slicer.filtering.morphology

### 76.10.1 GrayscaleFillHoleImageFilter

Link to code

Wraps the executable command "GrayscaleFillHoleImageFilter".

title: Grayscale Fill Hole Image Filter
category: Filtering.Morphology
description: GrayscaleFillholeImageFilter fills holes in a grayscale image. Holes are local minima in the grayscale topography that are not connected to boundaries of the image. Gray level values adjacent to a hole are extrapolated across the hole.
This filter is used to smooth over local minima without affecting the values of local maxima. If you take the difference between the output of this filter and the original image (and perhaps threshold the difference above a small value), you'll obtain a map of the local minima.
This filter uses the itkGrayscaleGeodesicErodeImageFilter. It provides its own input as the "mask" input to the geodesic erosion. The "marker" image for the geodesic erosion is constructed such that boundary pixels match the boundary pixels of the input image and the interior pixels are set to the maximum pixel value in the input image.


A companion filter, Grayscale Grind Peak, removes peaks in grayscale images.

version: 0.1.0.$Revision: 19608 $


ccontributor: Bill Lorensen (GE)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

[Optional]
inputVolume: (a pathlike object or string representing an existing file)
Input volume to be filtered
argument: ``%s``", position: -2
outputVolume: (a boolean or a pathlike object or string representing
a file)
Output filtered
argument: ``%s``", position: -1
args: (a unicode string)
Additional parameters to the command
argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {}))
Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing
file)
Output filtered

76.10.2 GrayscaleGrindPeakImageFilter

Link to code
Wraps the executable command 'GrayscaleGrindPeakImageFilter'.
title: Grayscale Grind Peak Image Filter
category: Filtering.Morphology
description: GrayscaleGrindPeakImageFilter removes peaks in a grayscale image. Peaks are local maxima in
the grayscale topography that are not connected to boundaries of the image. Gray level values adjacent to a peak
are extrapolated through the peak.
This filter is used to smooth over local maxima without affecting the values of local minima. If you take the
difference between the output of this filter and the original image (and perhaps threshold the difference above a
small value), you’ll obtain a map of the local maxima.
This filter uses the GrayscaleGeodesicDilateImageFilter. It provides its own input as the “mask” input to the
geodesic erosion. The “marker” image for the geodesic erosion is constructed such that boundary pixels match
the boundary pixels of the input image and the interior pixels are set to the minimum pixel value in the input
image.
This filter is the dual to the GrayscaleFillholeImageFilter which implements the Fillhole algorithm. Since it is a
dual, it is somewhat superfluous but is provided as a convenience.
Geodesic morphology and the Fillhole algorithm is described in Chapter 6 of Pierre Soille’s book “Morpholog-
A companion filter, Grayscale Fill Hole, fills holes in grayscale images.
version: 0.1.0.$Revision: 19608 $(alpha)
GrayscaleGrindPeakImageFilter
ccontributor: Bill Lorensen (GE)
acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded
by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.
Inputs:

[Optional]
inputVolume: (a pathlike object or string representing an existing
file)
Input volume to be filtered
argument: ``%s``", position: -2
outputVolume: (a boolean or a pathlike object or string representing
(continues on next page)
76.11 interfaces.slicer.filtering.n4itkbiasfieldcorrection

76.11.1 N4ITKBiasFieldCorrection

Link to code
Wraps the executable command "N4ITKBiasFieldCorrection".
title: N4ITK MRI Bias correction
category: Filtering
description: Performs image bias correction using N4 algorithm. This module is based on the ITK filters contributed in the following publication: Tustison N, Gee J “N4ITK: Nick’s N3 ITK Implementation For MRI Bias Field Correction”, The Insight Journal 2009 January-June, http://hdl.handle.net/10380/3053
version: 9
contributor: Nick Tustison (UPenn), Andrey Fedorov (SPL, BWH), Ron Kikinis (SPL, BWH)
acknowledgements: The development of this module was partially supported by NIH grants R01 AA016748-01, R01 CA111288 and U01 CA151261 as well as by NA-MIC, NAC, NCIGT and the Slicer community.

Inputs:

[Optional]
inputimage: (a pathlike object or string representing an existing file)
   Input image where you observe signal inhomogeneity
   argument: '--inputimage %s'
maskimage: (a pathlike object or string representing an existing file)
   Binary mask that defines the structure of your interest. NOTE: This parameter is OPTIONAL. If the mask is not specified, the module will use internally Otsu thresholding to define this mask. Better processing results can often be obtained when a meaningful mask is defined.
   argument: '--maskimage %s'
outputimage: (a boolean or a pathlike object or string representing a file)
   Result of processing
   argument: '--outputimage %s'
outputbiasfield: (a boolean or a pathlike object or string representing a file)

Outputs:

outputVolume: (a pathlike object or string representing an existing file)
   Output filtered
Recovered bias field (OPTIONAL)
argument: `--outputbiasfield %s`

iterations: (a list of items which are an integer (int or long))
Maximum number of iterations at each level of resolution. Larger
values will increase execution time, but may lead to better results.
argument: `--iterations %s`

convergencethreshold: (a float)
Stopping criterion for the iterative bias estimation. Larger values
will lead to smaller execution time.
argument: `--convergencethreshold %f`

meshresolution: (a list of items which are a float)
Resolution of the initial bspline grid defined as a sequence of
three numbers. The actual resolution will be defined by adding the
bspline order (default is 3) to the resolution in each dimension
specified here. For example, 1,1,1 will result in a 4x4x4 grid of
control points. This parameter may need to be adjusted based on your
input image. In the multi-resolution N4 framework, the resolution of
the bspline grid at subsequent iterations will be doubled. The
number of resolutions is implicitly defined by Number of iterations
parameter (the size of this list is the number of resolutions)
argument: `--meshresolution %s`

splinedistance: (a float)
An alternative means to define the spline grid, by setting the
distance between the control points. This parameter is used only if
the grid resolution is not specified.
argument: `--splinedistance %f`

shrinkfactor: (an integer (int or long))
Defines how much the image should be upsampled before estimating the
inhomogeneity field. Increase if you want to reduce the execution
time. 1 corresponds to the original resolution. Larger values will
significantly reduce the computation time.
argument: `--shrinkfactor %d`

bsplineorder: (an integer (int or long))
Order of B-spline used in the approximation. Larger values will lead
to longer execution times, may result in overfitting and poor
result.
argument: `--bsplineorder %d`

weightimage: (a pathlike object or string representing an existing
file)
Weight Image
argument: `--weightimage %s`

histogramsharpening: (a list of items which are a float)
A vector of up to three values. Non-zero values correspond to Bias
Field Full Width at Half Maximum, Wiener filter noise, and Number of
histogram bins.
argument: `--histogramsharpening %s`

args: (a unicode string)
Additional parameters to the command
argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

outputimage: (a pathlike object or string representing an existing
file)
Result of processing outputbiasfield: (a pathlike object or string representing an existing file) Recovered bias field (OPTIONAL)

76.12 interfaces.slicer.filtering.resamplescalarvectordwivolume

76.12.1 ResampleScalarVectorDWIVolume

Link to code Wraps the executable command "ResampleScalarVectorDWIVolume".

title: Resample Scalar/Vector/DWI Volume
category: Filtering
description: This module implements image and vector-image resampling through the use of itk Transforms. It can also handle diffusion weighted MRI image resampling. “Resampling” is performed in space coordinates, not pixel/grid coordinates. It is quite important to ensure that image spacing is properly set on the images involved. The interpolator is required since the mapping from one space to the other will often require evaluation of the intensity of the image at non-grid positions.
Warning: To resample DWMR Images, use nrrd input and output files.
Warning: Do not use to resample Diffusion Tensor Images, tensors would not be reoriented

version: 0.1
contributor: Francois Budin (UNC)
acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149. Information on the National Centers for Biomedical Computing can be obtained from http://nihroadmap.nih.gov/bioinformatics

Inputs:

[Optional]
inputVolume: (a pathlike object or string representing an existing file) Input Volume to be resampled argument: "%s", position: -2
outputVolume: (a boolean or a pathlike object or string representing a file) Resampled Volume argument: "%s", position: -1
Reference: (a pathlike object or string representing an existing file) Reference Volume (spacing, size, orientation, origin) argument: "--Reference %s"
transformationFile: (a pathlike object or string representing an existing file) argument: "--transformationFile %s"
defField: (a pathlike object or string representing an existing file) File containing the deformation field (3D vector image containing vectors with 3 components) argument: "--defField %s"
hfieldtype: ("displacement" or "h-Field") Set if the deformation field is an h-Field

(continues on next page)
argument: `--hfieldtype %s`
interpolation: ("linear" or "nn" or "ws" or "bs")
Sampling algorithm (linear or nn (nearest neighborhoor), ws
(WindowedSinc), bs (BSpline))
argument: `--interpolation %s`
transform_order: ("input-to-output" or "output-to-input")
Select in what order the transforms are read
argument: `--transform_order %s`
notbulk: (a boolean)
The transform following the BSpline transform is not set as a bulk
transform for the BSpline transform
argument: `--notbulk`
spaceChange: (a boolean)
Space Orientation between transform and image is different (RAS/LPS)
(warning: if the transform is a Transform Node in Slicer3, do not
select)
argument: `--spaceChange`
rotation_point: (a list of items which are any value)
Rotation Point in case of rotation around a point (otherwise
useless)
argument: `--rotation_point %s`
centered_transform: (a boolean)
Set the center of the transformation to the center of the input
image
argument: `--centered_transform`
image_center: ("input" or "output")
Image to use to center the transform (used only if "Centered
Transform" is selected)
argument: `--image_center %s`
Inverse_ITK_Transformation: (a boolean)
Inverse the transformation before applying it from output image to
input image
argument: `--Inverse_ITK_Transformation`
spacing: (a list of items which are a float)
Spacing along each dimension (0 means use input spacing)
argument: `--spacing %s`
size: (a list of items which are a float)
Size along each dimension (0 means use input size)
argument: `--size %s`
origin: (a list of items which are any value)
Origin of the output Image
argument: `--origin %s`
direction_matrix: (a list of items which are a float)
9 parameters of the direction matrix by rows (ijk to LPS if LPS
transform, ijk to RAS if RAS transform)
argument: `--direction_matrix %s`
number_of_thread: (an integer (int or long))
Number of thread used to compute the output image
argument: `--number_of_thread %d`
default_pixel_value: (a float)
Default pixel value for samples falling outside of the input region
argument: `--default_pixel_value %f`
window_function: ("h" or "c" or "w" or "l" or "b")
Window Function, h = Hamming, c = Cosine, w = Welch, l = Lanczos
, b = Blackman
argument: `--window_function %s`
spline_order: (an integer (int or long))
Spline Order
argument: `--spline_order %d`

transform_matrix: (a list of items which are a float)
12 parameters of the transform matrix by rows ( --last 3 being translation-- )
argument: `--transform_matrix %s`

transform: ('rt' or 'a')
Transform algorithm, rt = Rigid Transform, a = Affine Transform
argument: `--transform %s`

args: (a unicode string)
Additional parameters to the command
argument: `--%s`

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a value
of class 'str', nipype default value: {})
Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing file)
  Resampled Volume

76.13 interfaces.slicer.filtering.thresholdscalarvolume

76.13.1 ThresholdScalarVolume

Link to code
Wraps the executable command "ThresholdScalarVolume".
title: Threshold Scalar Volume
category: Filtering
description: <p>Threshold an image.</p><p>Set image values to a user-specified outside value if they are below, above, or between simple threshold values.</p><p>ThresholdAbove: The values greater than or equal to the threshold value are set to OutsideValue.</p><p>ThresholdBelow: The values less than or equal to the threshold value are set to OutsideValue.</p><p>ThresholdOutside: The values outside the range Lower-Upper are set to OutsideValue.</p><p>Although all image types are supported on input, only signed types are produced.</p><p>version: 0.1.0.$Revision: 2104 $(alpha)
ccontributor: Nicole Aucoin (SPL, BWH), Ron Kikinis (SPL, BWH)
acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

[Optional]
InputVolume: (a pathlike object or string representing an existing file)
  Input volume
  argument: `--%s`, position: -2
OutputVolume: (a boolean or a pathlike object or string representing a file)
  Thresholded input volume
  argument: `--%s`, position: -1
threshold: (an integer (int or long))
Threshold value
argument: ``--threshold %d``

lower: (an integer (int or long))
Lower threshold value
argument: ``--lower %d``

upper: (an integer (int or long))
Upper threshold value
argument: ``--upper %d``

outsidevalue: (an integer (int or long))
Set the voxels to this value if they fall outside the threshold range
argument: ``--outsidevalue %d``

thresholdtype: ('Below' or 'Above' or 'Outside')
What kind of threshold to perform. If Outside is selected, uses Upper and Lower values. If Below is selected, uses the ThresholdValue, if Above is selected, uses the ThresholdValue.
argument: ``--thresholdtype %s``

args: (a unicode string)
Additional parameters to the command
argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

OutputVolume: (a pathlike object or string representing an existing file)
Thresholded input volume

76.14 interfaces.slicer.filtering.votingbinaryholefillingimagefilter

76.14.1 VotingBinaryHoleFillingImageFilter

Link to code
Wraps the executable command VotingBinaryHoleFillingImageFilter.
title: Voting Binary Hole Filling Image Filter
category: Filtering
description: Applies a voting operation in order to fill-in cavities. This can be used for smoothing contours and for filling holes in binary images. This technique is used frequently when segmenting complete organs that may have ducts or vasculature that may not have been included in the initial segmentation, e.g. lungs, kidneys, liver.
version: 0.1.0.$Revision: 19608 $ (alpha)
contributor: Bill Lorensen (GE)
acknowledgements: This command module was derived from Insight/Examples/Filtering/VotingBinaryHoleFillingImageFilter
(copyright) Insight Software Consortium

[Optional]

radius: (a list of items which are an integer (int or long))
The radius of a hole to be filled
argument: ``--radius %s``

majorityThreshold: (an integer (int or long))

(continues on next page)
The number of pixels over 50% that will decide whether an OFF pixel will become ON or not. For example, if the neighborhood of a pixel has 124 pixels (excluding itself), the 50% will be 62, and if you set a Majority threshold of 5, that means that the filter will require 67 or more neighbor pixels to be ON in order to switch the current OFF pixel to ON.

argument: ``--majorityThreshold %d``

background: (an integer (int or long))
The value associated with the background (not object)
argument: ``--background %d``

foreground: (an integer (int or long))
The value associated with the foreground (object)
argument: ``--foreground %d``

inputVolume: (a pathlike object or string representing an existing file)
Input volume to be filtered
argument: ``%s``, position: -2

outputVolume: (a boolean or a pathlike object or string representing a file)
Output filtered
argument: ``%s``, position: -1

args: (a unicode string)
Additional parameters to the command
argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing file)
Output filtered

76.15 interfaces.slicer.legacy.converters

76.15.1 BSplineToDeformationField

Link to code
Wraps the executable command "BSplineToDeformationField".

title: BSpline to deformation field
category: Legacy.Converters
description: Create a dense deformation field from a bspline+bulk transform.
version: 0.1.0.$Revision: 2104 $(alpha)
contributor: Andrey Fedorov (SPL, BWH)
acknowledgements: This work is funded by NIH grants R01 CA111288 and U01 CA151261.

Inputs:

[Optional]
tfm: (a pathlike object or string representing an existing file)
argument: ``--tfm %s``

refImage: (a pathlike object or string representing an existing file)
argument: ``--refImage %s``
defImage: (a boolean or a pathlike object or string representing a file)
    argument: ``--defImage %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
    Environment variables

Outputs:
defImage: (a pathlike object or string representing an existing file)

76.16 interfaces.slicer.legacy.diffusion.denoising

76.16.1 DWIUnbiasedNonLocalMeansFilter

Link to code
Wraps the executable command ‘DWIUnbiasedNonLocalMeansFilter’.
title: DWI Unbiased Non Local Means Filter
category: Legacy.Diffusion.Denoising
description: This module reduces noise (or unwanted detail) on a set of diffusion weighted images. For this, it filters the images using a Unbiased Non Local Means for Rician noise algorithm. It exploits not only the spatial redundancy, but the redundancy in similar gradient directions as well; it takes into account the N closest gradient directions to the direction being processed (a maximum of 5 gradient directions is allowed to keep a reasonable computational load, since we do not use neither similarity maps nor block-wise implementation). The noise parameter is automatically estimated in the same way as in the jointLMMSE module. A complete description of the algorithm may be found in: Antonio Tristan-Vega and Santiago Aja-Fernandez, DWI filtering using joint information for DTI and HARDI, Medical Image Analysis, Volume 14, Issue 2, Pages 205-218. 2010. Please, note that the execution of this filter is extremely slow, so only very conservative parameters (block size and search size as small as possible) should be used. Even so, its execution may take several hours. The advantage of this filter over joint LMMSE is its better preservation of edges and fine structures.
version: 0.0.1.$Revision: 1 $(alpha)
contributor: Antonio Tristan Vega (UVa), Santiago Aja Fernandez (UVa)
acknowledgements: Partially founded by grant number TEC2007-67073/TCM from the Comision Interministerial de Ciencia y Tecnologia (Spain).

Inputs:

[Optional]
rs: (a list of items which are an integer (int or long))
    The algorithm search for similar voxels in a neighborhood of this size (larger sizes than the default one are extremely slow).
    argument: ``--rs %s``
rc: (a list of items which are an integer (int or long))
    Similarity between blocks is measured using windows of this size.
    argument: ``--rc %s``
hp: (a float)
    This parameter is related to noise; the larger the parameter, the more aggressive the filtering. Should be near 1, and only values

(continues on next page)
between 0.8 and 1.2 are allowed
argument: `--hp %f`

ng: (an integer (int or long))
The number of the closest gradients that are used to jointly filter
a given gradient direction (a maximum of 5 is allowed).
argument: `--ng %d`

re: (a list of items which are an integer (int or long))
A neighborhood of this size is used to compute the statistics for
noise estimation.
argument: `--re %s`

inputVolume: (a pathlike object or string representing an existing
file)
Input DWI volume.
argument: `%s`, position: -2

outputVolume: (a boolean or a pathlike object or string representing
a file)
Output DWI volume.
argument: `%s`, position: -1

args: (a unicode string)
Additional parameters to the command
argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing
file)
Output DWI volume.

76.17 interfaces.slicer.legacy.filtering

76.17.1 OtsuThresholdImageFilter

Link to code
Wraps the executable command `OtsuThresholdImageFilter`.

title: Otsu Threshold Image Filter
category: Legacy.Filtering
description: This filter creates a binary thresholded image that separates an image into foreground and back-
ground components. The filter calculates the optimum threshold separating those two classes so that their
combined spread (intra-class variance) is minimal (see http://en.wikipedia.org/wiki/Otsu%27s_method). Then
the filter applies that threshold to the input image using the itkBinaryThresholdImageFilter. The numberOfHis-
togram bins can be set for the Otsu Calculator. The insideValue and outsideValue can be set for the Binary-
ThresholdImageFilter. The filter produces a labeled volume.
The original reference is:
1979.
version: 0.1.0.$Revision: 19608 $(alpha)
OtsuThresholdImageFilter
ccontributor: Bill Lorensen (GE)
acknowledgements: This command module was derived from Insight/Examples (copyright) Insight Software
Consortium
Inputs:

```markdown
[Optional]
insideValue: (an integer (int or long))
   The value assigned to pixels that are inside the computed threshold
   argument: `--insideValue %d`
outsideValue: (an integer (int or long))
   The value assigned to pixels that are outside the computed threshold
   argument: `--outsideValue %d`
numberOfBins: (an integer (int or long))
   This is an advanced parameter. The number of bins in the histogram
   used to model the probability mass function of the two intensity
   distributions. Small numbers of bins may result in a more
   conservative threshold. The default should suffice for most
   applications. Experimentation is the only way to see the effect of
   varying this parameter.
   argument: `--numberOfBins %d`
inputVolume: (a pathlike object or string representing an existing
   file)
   Input volume to be filtered
   argument: `%%s`, position: -2
outputVolume: (a boolean or a pathlike object or string representing
   a file)
   Output filtered
   argument: `%%s`, position: -1
args: (a unicode string)
   Additional parameters to the command
   argument: `%%s`
environ: (a dictionary with keys which are a bytes or None or a value
   of class 'str' and with values which are a bytes or None or a
   value of class 'str', nipype default value: {})
   Environment variables
```

Outputs:

```markdown
outputVolume: (a pathlike object or string representing an existing
   file)
   Output filtered
```

### 76.17.2 ResampleScalarVolume

**Link to code**

Wraps the executable command "ResampleScalarVolume".

title: Resample Scalar Volume
category: Legacy.Filtering
description: Resampling an image is an important task in image analysis. It is especially important in the frame
of image registration. This module implements image resampling through the use of itk Transforms. This
module uses an Identity Transform. The resampling is controlled by the Output Spacing. “Resampling” is
performed in space coordinates, not pixel/grid coordinates. It is quite important to ensure that image spacing
is properly set on the images involved. The interpolator is required since the mapping from one space to the
other will often require evaluation of the intensity of the image at non-grid positions. Several interpolators are
available: linear, nearest neighbor, bspline and five flavors of sinc. The sinc interpolators, although more precise,
are much slower than the linear and nearest neighbor interpolator. To resample label volumes, nearest neighbor
interpolation should be used exclusively.
version: 0.1.0.$Revision: 20594 $(alpha)
ccontributor: Bill Lorensen (GE)
acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

[Optional]
spacing: (a list of items which are a float)
Spacing along each dimension (0 means use input spacing)
argument: `--spacing %s`

interpolation: ('linear' or 'nearestNeighbor' or 'bspline' or 'hamming' or 'cosine' or 'welch' or 'lanczos' or 'blackman')
Sampling algorithm (linear, nearest neighbor, bspline(cubic) or windowed sinc). There are several sinc algorithms available as described in the following publication: Erik H. W. Meijering, Wiro J. Niessen, Josien P. W. Pluim, Max A. Viergever: Quantitative Comparison of Sinc-Approximating Kernels for Medical Image Interpolation. MICCAI 1999, pp. 210-217. Each window has a radius of 3;
argument: `--interpolation %s`

InputVolume: (a pathlike object or string representing an existing file)
Input volume to be resampled
argument: `%s`, position: -2

OutputVolume: (a boolean or a pathlike object or string representing a file)
Resampled Volume
argument: `%s`, position: -1

args: (a unicode string)
Additional parameters to the command
argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

OutputVolume: (a pathlike object or string representing an existing file)
Resampled Volume

76.18 interfaces.slicer.legacy.registration

76.18.1 AffineRegistration

Link to code
Wraps the executable command "AffineRegistration".

title: Affine Registration
category: Legacy.Registration
description: Registers two images together using an affine transform and mutual information. This module is often used to align images of different subjects or images of the same subject from different modalities.
This module can smooth images prior to registration to mitigate noise and improve convergence. Many of the registration parameters require a working knowledge of the algorithm although the default parameters are sufficient for many registration tasks.
version: 0.1.0.$Revision: 19608 $(alpha)
ccontributor: Daniel Blezek (GE)
acknowledgements: This module was developed by Daniel Blezek while at GE Research with contributions from Jim Miller.
This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

[Optional]

fixedsmoothingfactor: (an integer (int or long))
Amount of smoothing applied to fixed image prior to registration.
Default is 0 (none). Range is 0-5 (unitless). Consider smoothing the input data if there is considerable amounts of noise or the noise pattern in the fixed and moving images is very different.
argument: `'--fixedsmoothingfactor %d'`

movingsmoothingfactor: (an integer (int or long))
Amount of smoothing applied to moving image prior to registration.
Default is 0 (none). Range is 0-5 (unitless). Consider smoothing the input data if there is considerable amounts of noise or the noise pattern in the fixed and moving images is very different.
argument: `'--movingsmoothingfactor %d'`

histogrambins: (an integer (int or long))
Number of histogram bins to use for Mattes Mutual Information.
Reduce the number of bins if a registration fails. If the number of bins is too large, the estimated PDFs will be a field of impulses and will inhibit reliable registration estimation.
argument: `'--histogrambins %d'`

spatialsamples: (an integer (int or long))
Number of spatial samples to use in estimating Mattes Mutual Information. Larger values yield more accurate PDFs and improved registration quality.
argument: `'--spatialsamples %d'`

iterations: (an integer (int or long))
Number of iterations
argument: `'--iterations %d'`

translationscale: (a float)
Relative scale of translations to rotations, i.e. a value of 100 means 10mm = 1 degree. (Actual scale used is 1/(TranslationScale^2)). This parameter is used to 'weight' or 'standardized' the transform parameters and their effect on the registration objective function.
argument: `'--translationscale %f'`

initialtransform: (a pathlike object or string representing an existing file)
Initial transform for aligning the fixed and moving image. Maps positions in the fixed coordinate frame to positions in the moving coordinate frame. Optional.
argument: `'--initialtransform %s'`

FixedImageFileName: (a pathlike object or string representing an existing file)
Fixed image to which to register
argument: `'%s'`, position: -2

MovingImageFileName: (a pathlike object or string representing an existing file)
Moving image
argument: `'%s'`, position: -1

outputtransform: (a boolean or a pathlike object or string representing a file)
Transform calculated that aligns the fixed and moving image. Maps positions in the fixed coordinate frame to the moving coordinate

(continues on next page)
frame. Optional (specify an output transform or an output volume or both).
argument: `--outputtransform %s`
resampledmovingfilename: (a boolean or a pathlike object or string representing a file)
Resampled moving image to the fixed image coordinate frame. Optional (specify an output transform or an output volume or both).
argument: `--resampledmovingfilename %s`
args: (a unicode string)
Additional parameters to the command
argument: `%%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:
outputtransform: (a pathlike object or string representing an existing file)
Transform calculated that aligns the fixed and moving image. Maps positions in the fixed coordinate frame to the moving coordinate frame. Optional (specify an output transform or an output volume or both).
resampledmovingfilename: (a pathlike object or string representing an existing file)
Resampled moving image to the fixed image coordinate frame. Optional (specify an output transform or an output volume or both).

76.18.2 BSplineDeformableRegistration

Link to code
Wraps the executable command `"BSplineDeformableRegistration"`.
title: BSpline Deformable Registration
category: Legacy.Registration
description: Registers two images together using BSpline transform and mutual information.
version: 0.1.0.$Revision: 19608 $(alpha)
contributor: Bill Lorensen (GE)
acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.
Inputs:

[Optional]
iterations: (an integer (int or long))
Number of iterations
argument: `--iterations %d`
gridSize: (an integer (int or long))
Number of grid points on interior of the fixed image. Larger grid sizes allow for finer registrations.
argument: `--gridSize %d`
histogrambins: (an integer (int or long))
Number of histogram bins to use for Mattes Mutual Information. Reduce the number of bins if a deformable registration fails. If the number of bins is too large, the estimated PDFs will be a field of
impulses and will inhibit reliable registration estimation.
argument: ```--histogrambins %d```  
spatial_samples: (an integer (int or long))
Number of spatial samples to use in estimating Mattes Mutual
Information. Larger values yield more accurate PDFs and improved
registration quality.
argument: ```--spatial_samples %d```  
constrain: (a boolean)
Constrain the deformation to the amount specified in Maximum
Deformation
argument: ```--constrain ```  
maximum_deformation: (a float)
If Constrain Deformation is checked, limit the deformation to this
amount.
argument: ```--maximum_deformation %f```  
default: (an integer (int or long))
Default pixel value used if resampling a pixel outside of the
volume.
argument: ```--default %d```  
initial_transform: (a pathlike object or string representing an
existing file)
Initial transform for aligning the fixed and moving image. Maps
positions in the fixed coordinate frame to positions in the moving
coordinate frame. This transform should be an affine or rigid
transform. It is used an a bulk transform for the BSpline. Optional.
argument: ```--initial_transform %s```  
fixed_image_filename: (a pathlike object or string representing an
existing file)
Fixed image to which to register
argument: ```%s```, position: -2  
moving_image_filename: (a pathlike object or string representing an
existing file)
Moving image
argument: ```%s```, position: -1  
output_transform: (a boolean or a pathlike object or string
representing a file)
Transform calculated that aligns the fixed and moving image. Maps
positions from the fixed coordinate frame to the moving coordinate
frame. Optional (specify an output transform or an output volume or
both).
argument: ```--output_transform %s```  
output_warp: (a boolean or a pathlike object or string representing a
file)
Vector field that applies an equivalent warp as the BSpline. Maps
positions from the fixed coordinate frame to the moving coordinate
frame. Optional.
argument: ```--output_warp %s```  
resampled_moving_filename: (a boolean or a pathlike object or string
representing a file)
Resampled moving image to fixed image coordinate frame. Optional
(specify an output transform or an output volume or both).
argument: ```--resampled_moving_filename %s```  
args: (a unicode string)
Additional parameters to the command
argument: ```%s```  
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {}))
Environment variables

Outputs:

**outputtransform**: (a pathlike object or string representing an existing file)
Transform calculated that aligns the fixed and moving image. Maps positions from the fixed coordinate frame to the moving coordinate frame. Optional (specify an output transform or an output volume or both).

**outputwarp**: (a pathlike object or string representing an existing file)
Vector field that applies an equivalent warp as the BSpline. Maps positions from the fixed coordinate frame to the moving coordinate frame. Optional.

**resampledmovingfilename**: (a pathlike object or string representing an existing file)
Resampled moving image to fixed image coordinate frame. Optional (specify an output transform or an output volume or both).

---

### 76.18.3 ExpertAutomatedRegistration

**Link to code**
Wraps the executable command ``ExpertAutomatedRegistration``.

title: Expert Automated Registration
category: Legacy.Registration
description: Provides rigid, affine, and BSpline registration methods via a simple GUI
version: 0.1.0.$Revision: 2104 $(alpha)
contributor: Stephen R Aylward (Kitware), Casey B Goodlett (Kitware)
acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

**Inputs:**

- **fixedImage**: (a pathlike object or string representing an existing file)
  Image which defines the space into which the moving image is registered
  argument: `"%s"`, position: -2

- **movingImage**: (a pathlike object or string representing an existing file)
  The transform goes from the fixed image's space into the moving image's space
  argument: `"%s"`, position: -1

- **resampledImage**: (a boolean or a pathlike object or string representing a file)
  Registration results
  argument: `"--resampledImage %s`'

- **loadTransform**: (a pathlike object or string representing an existing file)
  Load a transform that is immediately applied to the moving image
  argument: `"--loadTransform %s`'

- **saveTransform**: (a boolean or a pathlike object or string representing an existing file)
  Save a transform to file
  argument: `"--saveTransform %s`'
a file)
Save the transform that results from registration
argument: `--saveTransform %s`

initialization: ('None' or 'Landmarks' or 'ImageCenters' or
'CentersOfMass' or 'SecondMoments')
Method to prime the registration process
argument: `--initialization %s`

registration: ('None' or 'Initial' or 'Rigid' or 'Affine' or
'B spline' or 'PipelineRigid' or 'PipelineAffine' or
'PipelineBSpline')
Method for the registration process
argument: `--registration %s`

metric: ('MattesMI' or 'NormCorr' or 'MeanSqr d')
Method to quantify image match
argument: `--metric %s`

expectedOffset: (a float)
Expected misalignment after initialization
argument: `--expectedOffset %f`

expectedRotation: (a float)
Expected misalignment after initialization
argument: `--expectedRotation %f`

expectedScale: (a float)
Expected misalignment after initialization
argument: `--expectedScale %f`

expectedSkew: (a float)
Expected misalignment after initialization
argument: `--expectedSkew %f`

verbosityLevel: ('Silent' or 'Standard' or 'Verbose')
level of detail of reporting progress
argument: `--verbosityLevel %s`

sampleFromOverlap: (a boolean)
Limit metric evaluation to the fixed image region overlapped by the
moving image
argument: `--sampleFromOverlap`

fixedImageMask: (a pathlike object or string representing an existing
file)
Image which defines a mask for the fixed image
argument: `--fixedImageMask %s`

randomNumberSeed: (an integer (int or long))
Seed to generate a consistent random number sequence
argument: `--randomNumberSeed %d`

numberOfThreads: (an integer (int or long))
Number of CPU threads to use
argument: `--numberOfThreads %d`

minimizeMemory: (a boolean)
Reduce the amount of memory required at the cost of increased
computation time
argument: `--minimizeMemory`

interpolation: ('NearestNeighbor' or 'Linear' or 'BSpline')
Method for interpolation within the optimization process
argument: `--interpolation %s`

fixedLandmarks: (a list of items which are a list of from 3 to 3
items which are a float)
Ordered list of landmarks in the fixed image
argument: `--fixedLandmarks %s...

movingLandmarks: (a list of items which are a list of from 3 to 3
items which are a float)
Ordered list of landmarks in the moving image
argument: `--movingLandmarks %s...`

rigidMaxIterations: (an integer (int or long))
Maximum number of rigid optimization iterations
argument: `--rigidMaxIterations %d`

rigidSamplingRatio: (a float)
Portion of the image to use in computing the metric during rigid
registration
argument: `--rigidSamplingRatio %f`

affineMaxIterations: (an integer (int or long))
Maximum number of affine optimization iterations
argument: `--affineMaxIterations %d`

affineSamplingRatio: (a float)
Portion of the image to use in computing the metric during affine
registration
argument: `--affineSamplingRatio %f`

bsplineMaxIterations: (an integer (int or long))
Maximum number of bspline optimization iterations
argument: `--bsplineMaxIterations %d`

bsplineSamplingRatio: (a float)
Portion of the image to use in computing the metric during BSpline
registration
argument: `--bsplineSamplingRatio %f`

c控制PointSpacing: (an integer (int or long))
Number of pixels between control points
argument: `--controlPointSpacing %d`

args: (a unicode string)
Additional parameters to the command
argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})

Environment variables

Outputs:

**resampledImage**: (a pathlike object or string representing an existing file)
Registration results

**saveTransform**: (a pathlike object or string representing an existing file)
Save the transform that results from registration

### 76.18.4 LinearRegistration

Link to code
Wraps the executable command "LinearRegistration".
title: Linear Registration
category: Legacy.Registration
description: Registers two images together using a rigid transform and mutual information.
version: 0.1.0
creditor: Daniel Blezek (GE)
aknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded
by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:
fixedsmoothingfactor: (an integer (int or long))
Amount of smoothing applied to fixed image prior to registration.
Default is 0 (none). Range is 0-5 (unitless). Consider smoothing the
input data if there is considerable amounts of noise or the noise
pattern in the fixed and moving images is very different.
argument: `--fixedsmoothingfactor %d`
movingsmoothingfactor: (an integer (int or long))
Amount of smoothing applied to moving image prior to registration.
Default is 0 (none). Range is 0-5 (unitless). Consider smoothing the
input data if there is considerable amounts of noise or the noise
pattern in the fixed and moving images is very different.
argument: `--movingsmoothingfactor %d`
histogrambins: (an integer (int or long))
Number of histogram bins to use for Mattes Mutual Information.
Reduce the number of bins if a registration fails. If the number of
bins is too large, the estimated PDFs will be a field of impulses
and will inhibit reliable registration estimation.
argument: `--histogrambins %d`
spatialsamples: (an integer (int or long))
Number of spatial samples to use in estimating Mattes Mutual
Information. Larger values yield more accurate PDFs and improved
registration quality.
argument: `--spatialsamples %d`
iterations: (a list of items which are an integer (int or long))
Comma separated list of iterations. Must have the same number of
elements as the learning rate.
argument: `--iterations %s`
learningrate: (a list of items which are a float)
Comma separated list of learning rates. Learning rate is a scale
factor on the gradient of the registration objective function
(gradients with respect to the parameters of the transformation) used
to update the parameters of the transformation during optimization.
Smaller values cause the optimizer to take smaller steps through the
parameter space. Larger values are typically used early in the
registration process to take large jumps in parameter space followed
by smaller values to home in on the optimum value of the
registration objective function. Default is: 0.01, 0.005, 0.0005,
0.0002. Must have the same number of elements as iterations.
argument: `--learningrate %s`
translationscale: (a float)
Relative scale of translations to rotations, i.e. a value of 100
means 10mm = 1 degree. (Actual scale used 1/(TranslationScale^2)).
This parameter is used to 'weight' or 'standardized' the transform
parameters and their effect on the registration objective function.
argument: `--translationscale %f`
initialtransform: (a pathlike object or string representing an
existing file)
Initial transform for aligning the fixed and moving image. Maps
positions in the fixed coordinate frame to positions in the moving
coordinate frame. Optional.
argument: `--initialtransform %s`
FixedImageFileName: (a pathlike object or string representing an
existing file)
Fixed image to which to register
argument: `%s`, position: -2
MovingImageFileName: (a pathlike object or string representing an
existing file)
Moving image
argument: `\%s`, position: -1
outputtransform: (a boolean or a pathlike object or string
representing a file)
Transform calculated that aligns the fixed and moving image. Maps
positions in the fixed coordinate frame to the moving coordinate
frame. Optional (specify an output transform or an output volume or
both).
argument: `--outputtransform %s`
resampledmovingfilename: (a boolean or a pathlike object or string
representing a file)
Resampled moving image to the fixed image coordinate frame. Optional
(specify an output transform or an output volume or both).
argument: `--resampledmovingfilename %s`
args: (a unicode string)
Additional parameters to the command
argument: `\%s`
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})}
Environment variables

Outputs:

outputtransform: (a pathlike object or string representing an
existing file)
Transform calculated that aligns the fixed and moving image. Maps
positions in the fixed coordinate frame to the moving coordinate
frame. Optional (specify an output transform or an output volume or
both).
resampledmovingfilename: (a pathlike object or string representing an
existing file)
Resampled moving image to the fixed image coordinate frame. Optional
(specify an output transform or an output volume or both).

76.18.5 MultiResolutionAffineRegistration

Link to code
Wraps the executable command `"MultiResolutionAffineRegistration"`.
title: Robust Multiresolution Affine Registration
category: Legacy.Registration
description: Provides affine registration using multiple resolution levels and decomposed affine transforms.
version: 0.1.0.$Revision: 2104 $(alpha)
documentation-url: http://www.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/MultiResolutionAffineRegistration
contributor: Casey B Goodlett (Utah)
acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded
by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.
Inputs:

[Optional]
fixedImage: (a pathlike object or string representing an existing
file)
Image which defines the space into which the moving image is
registered
argument: ``%s``, position: -2
movingImage: (a pathlike object or string representing an existing file)
The transform goes from the fixed image's space into the moving image's space
argument: ``%s``, position: -1
resampledImage: (a boolean or a pathlike object or string representing a file)
Registration results
argument: ``--resampledImage %s``
saveTransform: (a boolean or a pathlike object or string representing a file)
Save the output transform from the registration
argument: ``--saveTransform %s``
fixedImageMask: (a pathlike object or string representing an existing file)
Label image which defines a mask of interest for the fixed image
argument: ``--fixedImageMask %s``
fixedImageROI: (a list of items which are any value)
Label image which defines a ROI of interest for the fixed image
argument: ``--fixedImageROI %s``
numIterations: (an integer (int or long))
Number of iterations to run at each resolution level.
argument: ``--numIterations %d``
numLineIterations: (an integer (int or long))
Number of iterations to run at each resolution level.
argument: ``--numLineIterations %d``
stepSize: (a float)
The maximum step size of the optimizer in voxels
argument: ``--stepSize %f``
stepTolerance: (a float)
The maximum step size of the optimizer in voxels
argument: ``--stepTolerance %f``
metricTolerance: (a float)
argument: ``--metricTolerance %f``
args: (a unicode string)
Additional parameters to the command
argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

resampledImage: (a pathlike object or string representing an existing file)
Registration results
saveTransform: (a pathlike object or string representing an existing file)
Save the output transform from the registration

76.18.6 RigidRegistration

Link to code
Wraps the executable command ``RigidRegistration``.
title: Rigid Registration
category: Legacy.Registration
description: Registers two images together using a rigid transform and mutual information. This module was originally distributed as “Linear registration” but has been renamed to eliminate confusion with the “Affine registration” module. This module is often used to align images of different subjects or images of the same subject from different modalities. This module can smooth images prior to registration to mitigate noise and improve convergence. Many of the registration parameters require a working knowledge of the algorithm although the default parameters are sufficient for many registration tasks.

version: 0.1.0.


ccontributor: Daniel Blezek (GE)

acknowledgements: This module was developed by Daniel Blezek while at GE Research with contributions from Jim Miller.

This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

[Optional]

fixedsmoothingfactor: (an integer (int or long))
Amount of smoothing applied to fixed image prior to registration.
Default is 0 (none). Range is 0-5 (unitless). Consider smoothing the input data if there is considerable amounts of noise or the noise pattern in the fixed and moving images is very different.
argument: `--fixedsmoothingfactor %d`

movingsmoothingfactor: (an integer (int or long))
Amount of smoothing applied to moving image prior to registration.
Default is 0 (none). Range is 0-5 (unitless). Consider smoothing the input data if there is considerable amounts of noise or the noise pattern in the fixed and moving images is very different.
argument: `--movingsmoothingfactor %d`

testingmode: (a boolean)
Enable testing mode. Input transform will be used to construct floating image. The floating image will be ignored if passed.
argument: `--testingmode`

histogrambins: (an integer (int or long))
Number of histogram bins to use for Mattes Mutual Information. Reduce the number of bins if a registration fails. If the number of bins is too large, the estimated PDFs will be a field of impulses and will inhibit reliable registration estimation.
argument: `--histogrambins %d`

spatialsamples: (an integer (int or long))
Number of spatial samples to use in estimating Mattes Mutual Information. Larger values yield more accurate PDFs and improved registration quality.
argument: `--spatialsamples %d`

iterations: (a list of items which are an integer (int or long))
Comma separated list of iterations. Must have the same number of elements as the learning rate.
argument: `--iterations %s`

learningrate: (a list of items which are a float)
Comma separated list of learning rates. Learning rate is a scale factor on the gradient of the registration objective function (gradient with respect to the parameters of the transformation) used to update the parameters of the transformation during optimization. Smaller values cause the optimizer to take smaller steps through the parameter space. Larger values are typically used early in the

(continues on next page)
registration process to take large jumps in parameter space followed by smaller values to home in on the optimum value of the registration objective function. Default is: 0.01, 0.005, 0.0005, 0.0002. Must have the same number of elements as iterations.

argument: `--learningrate %s`

translationscale: (a float)
Relative scale of translations to rotations, i.e. a value of 100 means 10mm = 1 degree. (Actual scale used 1/(TranslationScale^2)). This parameter is used to 'weight' or 'standardized' the transform parameters and their effect on the registration objective function.

argument: `--translationscale %f`

initialtransform: (a pathlike object or string representing an existing file)
Initial transform for aligning the fixed and moving image. Maps positions in the fixed coordinate frame to positions in the moving coordinate frame. Optional.

argument: `--initialtransform %s`

FixedImageFileName: (a pathlike object or string representing an existing file)
Fixed image to which to register

argument: `--FixedImageFileName %s`, position: -2

MovingImageFileName: (a pathlike object or string representing an existing file)
Moving image

argument: `--MovingImageFileName %s`, position: -1

outputtransform: (a boolean or a pathlike object or string representing a file)
Transform calculated that aligns the fixed and moving image. Maps positions in the fixed coordinate frame to the moving coordinate frame. Optional (specify an output transform or an output volume or both).

argument: `--outputtransform %s`

resampledmovingfilename: (a boolean or a pathlike object or string representing a file)
Resampled moving image to the fixed image coordinate frame. Optional (specify an output transform or an output volume or both).

argument: `--resampledmovingfilename %s`

args: (a unicode string)
Additional parameters to the command

argument: `--args %s`

eviron: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

Environment variables

Outputs:

outputtransform: (a pathlike object or string representing an existing file)
Transform calculated that aligns the fixed and moving image. Maps positions in the fixed coordinate frame to the moving coordinate frame. Optional (specify an output transform or an output volume or both).

resampledmovingfilename: (a pathlike object or string representing an existing file)
Resampled moving image to the fixed image coordinate frame. Optional (specify an output transform or an output volume or both).
76.19 interfaces.slicer.legacy.segmentation

76.19.1 OtsuThresholdSegmentation

Wraps the executable command ``OtsuThresholdSegmentation``.

title: Otsu Threshold Segmentation
category: Legacy.Segmentation
description: This filter creates a labeled image from a grayscale image. First, it calculates an optimal threshold that separates the image into foreground and background. This threshold separates those two classes so that their intra-class variance is minimal (see http://en.wikipedia.org/wiki/Otsu%27s_method). Then the filter runs a connected component algorithm to generate unique labels for each connected region of the foreground. Finally, the resulting image is relabeled to provide consecutive numbering.

version: 1.0

contributor: Bill Lorensen (GE)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```
[Optional]
brightObjects: (a boolean)
    Segmenting bright objects on a dark background or dark objects on a
    bright background.
    argument: ``--brightObjects``

numberOfBins: (an integer (int or long))
    This is an advanced parameter. The number of bins in the histogram
    used to model the probability mass function of the two intensity
    distributions. Small numbers of bins may result in a more
    conservative threshold. The default should suffice for most
    applications. Experimentation is the only way to see the effect of
    varying this parameter.
    argument: ``--numberOfBins %d``

faceConnected: (a boolean)
    This is an advanced parameter. Adjacent voxels are face connected.
    This affects the connected component algorithm. If this parameter is
    false, more regions are likely to be identified.
    argument: ``--faceConnected``

minimumObjectSize: (an integer (int or long))
    Minimum size of object to retain. This parameter can be used to get
    rid of small regions in noisy images.
    argument: ``--minimumObjectSize %d``

inputVolume: (a pathlike object or string representing an existing
    file)
    Input volume to be segmented
    argument: ``-%s``, position: -2

outputVolume: (a boolean or a pathlike object or string representing
    a file)
    Output filtered
    argument: ``-%s``, position: -1

args: (a unicode string)
    Additional parameters to the command
    argument: ``-%s``

environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
```
Environment variables

<table>
<thead>
<tr>
<th>Value of class 'str', nipype default value: {}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outputs:</td>
</tr>
<tr>
<td>outputVolume: (a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td>Output filtered</td>
</tr>
</tbody>
</table>

### 76.20 interfaces.slicer.quantification.changequantification

#### 76.20.1 IntensityDifferenceMetric

**Link to code**
Wraps the executable command `"IntensityDifferenceMetric"`.

**Title:** Intensity Difference Change Detection (FAST)

**Category:** Quantification.ChangeQuantification

**Description:** Quantifies the changes between two spatially aligned images based on the pixel-wise difference of image intensities.

**Version:** 0.1

**Contributor:** Andrey Fedorov

**Acknowledgements:**

**Inputs:**

- sensitivityThreshold: (a float)
  This parameter should be between 0 and 1, and defines how sensitive the metric should be to the intensity changes.
  - argument: `"--sensitivityThreshold %f"`

- changingBandSize: (an integer (int or long))
  How far (in mm) from the boundary of the segmentation should the intensity changes be considered.
  - argument: `"--changingBandSize %d"`

- baselineVolume: (a pathlike object or string representing an existing file)
  Baseline volume to be compared to
  - argument: `"%s"`, position: -4

- baselineSegmentationVolume: (a pathlike object or string representing an existing file)
  Label volume that contains segmentation of the structure of interest in the baseline volume.
  - argument: `"%s"`, position: -3

- followupVolume: (a pathlike object or string representing an existing file)
  Followup volume to be compare to the baseline
  - argument: `"%s"`, position: -2

- outputVolume: (a boolean or a pathlike object or string representing a file)
  Output volume to keep the results of change quantification.
  - argument: `"%s"`, position: -1

- reportFileName: (a boolean or a pathlike object or string representing a file)
  Report file name
  - argument: `"--reportFileName %s"`

**Args:**

(continues on next page)
Additional parameters to the command
argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing
file)
Output volume to keep the results of change quantification.
reportFileName: (a pathlike object or string representing an existing
file)
Report file name

76.21 interfaces.slicer.quantification.petstandarduptakevaluecomputation

76.21.1 PETStandardUptakeValueComputation

Link to code
Wraps the executable command `"PETStandardUptakeValueComputation"`.
title: PET Standard Uptake Value Computation
category: Quantification
description: Computes the standardized uptake value based on body weight. Takes an input PET image in
DICOM and NRRD format (DICOM header must contain Radiopharmaceutical parameters). Produces a CSV
file that contains patientID, studyDate, dose, labelID, suvmin, suvmax, suvmean, labelName for each volume
of interest. It also displays some of the information as output strings in the GUI, the CSV file is optional in that
case. The CSV file is appended to on each execution of the CLI.
version: 0.1.0.$Revision: 8595 $(alpha)
ComputeSUVBodyWeight
contributor: Wendy Plesniak (SPL, BWH), Nicole Aucoin (SPL, BWH), Ron Kikinis (SPL, BWH)
acknowledgements: This work is funded by the Harvard Catalyst, and the National Alliance for Medical Im-
age Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical
Research, Grant U54 EB005149.
Inputs:

[Optional]

petDICOMPath: (a pathlike object or string representing an existing
directory)
Input path to a directory containing a PET volume containing DICOM
header information for SUV computation
argument: `"--petDICOMPath %s"`

petVolume: (a pathlike object or string representing an existing
file)
Input PET volume for SUVbw computation (must be the same volume as
pointed to by the DICOM path!).
argument: `"--petVolume %s"`

labelMap: (a pathlike object or string representing an existing file)
Input label volume containing the volumes of interest
argument: `"--labelMap %s"`

[Optional]

color: (a pathlike object or string representing an existing file)
Color table to to map labels to colors and names

(continues on next page)
argument: `--color %s`
csvFile: (a boolean or a pathlike object or string representing a file)
  A file holding the output SUV values in comma separated lines, one per label. Optional.
  argument: `--csvFile %s`
OutputLabel: (a unicode string)
  List of labels for which SUV values were computed
  argument: `--OutputLabel %s`
OutputLabelValue: (a unicode string)
  List of label values for which SUV values were computed
  argument: `--OutputLabelValue %s`
SUVMax: (a unicode string)
  SUV max for each label
  argument: `--SUVMax %s`
SUVMean: (a unicode string)
  SUV mean for each label
  argument: `--SUVMean %s`
SUVMin: (a unicode string)
  SUV minimum for each label
  argument: `--SUVMin %s`
args: (a unicode string)
  Additional parameters to the command
  argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  Environment variables

Outputs:
csvFile: (a pathlike object or string representing an existing file)
  A file holding the output SUV values in comma separated lines, one per label. Optional.

---

### 76.22 interfaces.slicer.registration.brainsfit

#### 76.22.1 BRAINSFit

Link to code

Wraps the executable command “BRAINSFit”.
title: General Registration (BRAINS)
category: Registration
version: 3.0.0
license: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt
acknowledgements: Hans Johnson(1,3,4); Kent Williams(1); Gregory Harris(1), Vincent Magnotta(1,2,3); Andriy Fedorov(5) 1=University of Iowa Department of Psychiatry, 2=University of Iowa Department of Radiology, 3=University of Iowa Department of Biomedical Engineering, 4=University of Iowa Department of Electrical and Computer Engineering, 5=Surgical Planning Lab, Harvard
Inputs:
[Optional]

**fixedVolume**: (a pathlike object or string representing an existing file)
The fixed image for registration by mutual information optimization.
argument: ``--fixedVolume %s``

**movingVolume**: (a pathlike object or string representing an existing file)
The moving image for registration by mutual information optimization.
argument: ``--movingVolume %s``

**bsplineTransform**: (a boolean or a pathlike object or string representing a file)
(optional) Filename to which save the estimated transform. NOTE: You must set at least one output object (either a deformed image or a transform. NOTE: USE THIS ONLY IF THE FINAL TRANSFORM IS BSpline
argument: ``--bsplineTransform %s``

**linearTransform**: (a boolean or a pathlike object or string representing a file)
(optional) Filename to which save the estimated transform. NOTE: You must set at least one output object (either a deformed image or a transform. NOTE: USE THIS ONLY IF THE FINAL TRANSFORM IS ---NOT--- BSpline
argument: ``--linearTransform %s``

**outputVolume**: (a boolean or a pathlike object or string representing a file)
(optional) Output image for registration. NOTE: You must select either the outputTransform or the outputVolume option.
argument: ``--outputVolume %s``

**initialTransform**: (a pathlike object or string representing an existing file)
Filename of transform used to initialize the registration. This CAN NOT be used with either CenterOfHeadLAlign, MomentsAlign, GeometryAlign, or initialTransform file.
argument: ``--initialTransform %s``

**initializeTransformMode**: ('Off' or 'useMomentsAlign' or 'useCenterOfHeadAlign' or 'useGeometryAlign' or 'useCenterOfROIAlign')
Determine how to initialize the transform center. GeometryAlign on assumes that the center of the voxel lattice of the images represent similar structures. MomentsAlign assumes that the center of mass of the images represent similar structures. useCenterOfHeadAlign attempts to use the top of head and shape of neck to drive a center of mass estimate. Off assumes that the physical space of the images are close, and that centering in terms of the image Origins is a good starting point. This flag is mutually exclusive with the initialTransform flag.
argument: ``--initializeTransformMode %s``

**useRigid**: (a boolean)
Perform a rigid registration as part of the sequential registration steps. This family of options superceeds the use of transformType if any of them are set.
argument: ``--useRigid``

**useScaleVersor3D**: (a boolean)
Perform a ScaleVersor3D registration as part of the sequential registration steps. This family of options supercedes the use of transformType if any of them are set.
argument: ``--useScaleVersor3D``

(continues on next page)
**useScaleSkewVersor3D**: (a boolean)
- Perform a ScaleSkewVersor3D registration as part of the sequential registration steps. This family of options supersedes the use of `transformType` if any of them are set.
- argument: `--useScaleSkewVersor3D`

**useAffine**: (a boolean)
- Perform an Affine registration as part of the sequential registration steps. This family of options supersedes the use of `transformType` if any of them are set.
- argument: `--useAffine`

**useBSpline**: (a boolean)
- Perform a BSpline registration as part of the sequential registration steps. This family of options supersedes the use of `transformType` if any of them are set.
- argument: `--useBSpline`

**numberOfSamples**: (an integer (int or long))
- The number of voxels sampled for mutual information computation. Increase this for a slower, more careful fit. You can also limit the sampling focus with ROI masks and ROIAUTO mask generation.
- argument: `--numberOfSamples %d`

**splineGridSize**: (a list of items which are an integer (int or long))
- The number of subdivisions of the BSpline Grid to be centered on the image space. Each dimension must have at least 3 subdivisions for the BSpline to be correctly computed.
- argument: `--splineGridSize %s`

**numberOfIterations**: (a list of items which are an integer (int or long))
- The maximum number of iterations to try before failing to converge. Use an explicit limit like 500 or 1000 to manage risk of divergence.
- argument: `--numberOfIterations %s`

**maskProcessingMode**: ('NOMASK' or 'ROIAUTO' or 'ROI')
- What mode to use for using the masks. If ROIAUTO is choosen, then the mask is implicitly defined using a otsu forground and hole filling algorithm. The Region Of Interest mode (choose ROI) uses the masks to define what parts of the image should be used for computing the transform.
- argument: `--maskProcessingMode %s`

**fixedBinaryVolume**: (a pathlike object or string representing an existing file)
- Fixed Image binary mask volume, ONLY FOR MANUAL ROI mode.
- argument: `--fixedBinaryVolume %s`

**movingBinaryVolume**: (a pathlike object or string representing an existing file)
- Moving Image binary mask volume, ONLY FOR MANUAL ROI mode.
- argument: `--movingBinaryVolume %s`

**outputFixedVolumeROI**: (a boolean or a pathlike object or string representing a file)
- The ROI automatically found in fixed image, ONLY FOR ROIAUTO mode.
- argument: `--outputFixedVolumeROI %s`

**outputMovingVolumeROI**: (a boolean or a pathlike object or string representing a file)
- The ROI automatically found in moving image, ONLY FOR ROIAUTO mode.
- argument: `--outputMovingVolumeROI %s`

**outputVolumePixelType**: ('float' or 'short' or 'ushort' or 'int' or 'uint' or 'uchar')
- The output image Pixel Type is the scalar datatype for representation of the Output Volume.
argument: `--outputVolumePixelType %s`

backgroundFillValue: (a float)
Background fill value for output image.
argument: `--backgroundFillValue %f`

maskInferiorCutOffFromCenter: (a float)
For use with --useCenterOfHeadAlign (and --maskProcessingMode ROIAUTO): the cut-off below the image centers, in millimeters,
argument: `--maskInferiorCutOffFromCenter %f`

scaleOutputValues: (a boolean)
If true, and the voxel values do not fit within the minimum and
maximum values of the desired outputVolumePixelType, then linearly
scale the min/max output image voxel values to fit within the
min/max range of the outputVolumePixelType.
argument: `--scaleOutputValues`

interpolationMode: ('NearestNeighbor' or 'Linear' or
'ResampleInPlace' or 'BSpline' or 'WindowedSinc' or 'Hamming' or
'Cosine' or 'Welch' or 'Lanczos' or 'Blackman')
Type of interpolation to be used when applying transform to moving
volume. Options are Linear, NearestNeighbor, BSpline, WindowedSinc,
or ResampleInPlace. The ResampleInPlace option will create an image
with the same discrete voxel values and will adjust the origin and
direction of the physical space interpretation.
argument: `--interpolationMode %s`

minimumStepLength: (a list of items which are a float)
Each step in the optimization takes steps at least this big. When
none are possible, registration is complete.
argument: `--minimumStepLength %s`

translationScale: (a float)
How much to scale up changes in position compared to unit rotational
changes in radians -- decrease this to put more rotation in the
search pattern.
argument: `--translationScale %f`

reproportionScale: (a float)
ScaleVersor3D 'Scale' compensation factor. Increase this to put more
rescaling in a ScaleVersor3D or ScaleSkewVersor3D search pattern.
1.0 works well with a translationScale of 1000.0
argument: `--reproportionScale %f`

skewScale: (a float)
ScaleSkewVersor3D Skew compensation factor. Increase this to put
more skew in a ScaleSkewVersor3D search pattern. 1.0 works well with
a translationScale of 1000.0
argument: `--skewScale %f`

maxBSplineDisplacement: (a float)
Sets the maximum allowed displacements in image physical
coordinates for BSpline control grid along each axis. A value of 0.0
indicates that the problem should be unbounded. NOTE: This only
constrains the BSpline portion, and does not limit the displacement
from the associated bulk transform. This can lead to a substantial
reduction in computation time in the BSpline optimizer.,
argument: `--maxBSplineDisplacement %f`

histogramMatch: (a boolean)
Histogram Match the input images. This is suitable for images of the
same modality that may have different absolute scales, but the same
overall intensity profile. Do NOT use if registering images from
different modalities.
argument: `--histogramMatch`

numberOfHistogramBins: (an integer (int or long))
The number of histogram levels
argument: `--numberOfHistogramBins %d`

numberOfMatchPoints: (an integer (int or long))
the number of match points
argument: `--numberOfMatchPoints %d`

strippedOutputTransform: (a boolean or a pathlike object or string representing a file)
File name for the rigid component of the estimated affine transform. Can be used to rigidly register the moving image to the fixed image. NOTE: This value is overwritten if either bsplineTransform or linearTransform is set.
argument: `--strippedOutputTransform %s`

transformType: (a list of items which are a unicode string)
Specifies a list of registration types to be used. The valid types are, Rigid, ScaleVersor3D, ScaleSkewVersor3D, Affine, and BSpline. Specifying more than one in a comma separated list will initialize the next stage with the previous results. If registrationClass flag is used, it overrides this parameter setting.
argument: `--transformType %s`

outputTransform: (a boolean or a pathlike object or string representing a file)
(optional) Filename to which save the (optional) estimated transform. NOTE: You must select either the outputTransform or the outputVolume option.
argument: `--outputTransform %s`

fixedVolumeTimeIndex: (an integer (int or long))
The index in the time series for the 3D fixed image to fit, if 4-dimensional.
argument: `--fixedVolumeTimeIndex %d`

movingVolumeTimeIndex: (an integer (int or long))
The index in the time series for the 3D moving image to fit, if 4-dimensional.
argument: `--movingVolumeTimeIndex %d`

medianFilterSize: (a list of items which are an integer (int or long))
The radius for the optional MedianImageFilter preprocessing in all 3 directions.
argument: `--medianFilterSize %s`

removeIntensityOutliers: (a float)
The half percentage to decide outliers of image intensities. The default value is zero, which means no outlier removal. If the value of 0.005 is given, the module will throw away 0.005 % of both tails, so 0.01% of intensities in total would be ignored in its statistic calculation.
argument: `--removeIntensityOutliers %f`

useCachingOfBSplineWeightsMode: ('ON' or 'OFF')
This is a 5x speed advantage at the expense of requiring much more memory. Only relevant when transformType is BSpline.
argument: `--useCachingOfBSplineWeightsMode %s`

useExplicitPDFDerivativesMode: ('AUTO' or 'ON' or 'OFF')
Using mode AUTO means OFF for BSplineDeformableTransforms and ON for the linear transforms. The ON alternative uses more memory to sometimes do a better job.
argument: `--useExplicitPDFDerivativesMode %s`

ROIAutoDilateSize: (a float)
This flag is only relavent when using ROIAUTO mode for initializing masks. It defines the final dilation size to capture a bit of
background outside the tissue region. At setting of 10mm has been shown to help regularize a BSpline registration type so that there is some background constraints to match the edges of the head better.

argument: `--ROIAutoDilateSize %f`

**ROIAutoClosingSize**: (a float)

This flag is only relevant when using ROIAUTO mode for initializing masks. It defines the hole closing size in mm. It is rounded up to the nearest whole pixel size in each direction. The default is to use a closing size of 9mm. For mouse data this value may need to be reset to 0.9 or smaller.

argument: `--ROIAutoClosingSize %f`

**relaxationFactor**: (a float)

Internal debugging parameter, and should probably never be used from the command line. This will be removed in the future.

argument: `--relaxationFactor %f`

**maximumStepLength**: (a float)

Internal debugging parameter, and should probably never be used from the command line. This will be removed in the future.

argument: `--maximumStepLength %f`

**failureExitCode**: (an integer (int or long))

If the fit fails, exit with this status code. (It can be used to force a successful exit status of (0) if the registration fails due to reaching the maximum number of iterations.

argument: `--failureExitCode %d`

**writeTransformOnFailure**: (a boolean)

Flag to save the final transform even if the numberOfIterations are reached without convergence. (Intended for use when --failureExitCode 0)

argument: `--writeTransformOnFailure`

**numberOfThreads**: (an integer (int or long))

Explicitly specify the maximum number of threads to use. (default is auto-detected)

argument: `--numberOfThreads %d`

**forceMINumberOfThreads**: (an integer (int or long))

Force the the maximum number of threads to use for non thread safe MI metric. CAUTION: Inconsistent results my arise!

argument: `--forceMINumberOfThreads %d`

**debugLevel**: (an integer (int or long))

Display debug messages, and produce debug intermediate results. 0=OFF, 1=Minimal, 10=Maximum debugging.

argument: `--debugLevel %d`

**costFunctionConvergenceFactor**: (a float)

From itkLBFGSOptimizer.h: Set/Get the CostFunctionConvergenceFactor. Algorithm terminates when the reduction in cost function is less than (factor * epsmcj) where epsmch is the machine precision. Typical values for factor: 1e+12 for low accuracy; 1e+7 for moderate accuracy and 1e+1 for extremely high accuracy. 1e+9 seems to work well.,

argument: `--costFunctionConvergenceFactor %f`

**projectedGradientTolerance**: (a float)

From itkLBFGSOptimizer.h: Set/Get the ProjectedGradientTolerance. Algorithm terminates when the project gradient is below the tolerance. Default lbfgsb value is 1e-5, but 1e-4 seems to work well.,

argument: `--projectedGradientTolerance %f`

**gui**: (a boolean)

(continues on next page)
Display intermediate image volumes for debugging. NOTE: This is not part of the standard build system, and probably does nothing on your installation.

argument: `--gui`

promptUser: (a boolean)

Prompt the user to hit enter each time an image is sent to the DebugImageViewer

argument: `--promptUser`

NEVER_USE_THIS_FLAG_IT_IS_OUTDATED_00: (a boolean)

DO NOT USE THIS FLAG

argument: `--NEVER_USE_THIS_FLAG_IT_IS_OUTDATED_00`

NEVER_USE_THIS_FLAG_IT_IS_OUTDATED_01: (a boolean)

DO NOT USE THIS FLAG

argument: `--NEVER_USE_THIS_FLAG_IT_IS_OUTDATED_01`

NEVER_USE_THIS_FLAG_IT_IS_OUTDATED_02: (a boolean)

DO NOT USE THIS FLAG

argument: `--NEVER_USE_THIS_FLAG_IT_IS_OUTDATED_02`

permitParameterVariation: (a list of items which are an integer (int or long))

A bit vector to permit linear transform parameters to vary under optimization. The vector order corresponds with transform parameters, and beyond the end ones fill in as a default. For instance, you can choose to rotate only in x (pitch) with 1,0,0; this is mostly for expert use in turning on and off individual degrees of freedom in rotation, translation or scaling without multiplying the number of transform representations; this trick is probably meaningless when tried with the general affine transform.

argument: `--permitParameterVariation %s`

costMetric: ('MMI' or 'MSE' or 'NC' or 'MC')

The cost metric to be used during fitting. Defaults to MMI. Options are MMI (Mattes Mutual Information), MSE (Mean Square Error), NC (Normalized Correlation), MC (Match Cardinality for binary images)

argument: `--costMetric %s`

writeOutputTransformInFloat: (a boolean)

By default, the output registration transforms (either the output composite transform or each transform component) are written to the disk in double precision. If this flag is ON, the output transforms will be written in single (float) precision. It is especially important if the output transform is a displacement field transform, or it is a composite transform that includes several displacement fields.

argument: `--writeOutputTransformInFloat`

args: (a unicode string)

Additional parameters to the command

argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

Environment variables

Outputs:

bsplineTransform: (a pathlike object or string representing an existing file)

(optional) Filename to which save the estimated transform. NOTE: You must set at least one output object (either a deformed image or a transform. NOTE: USE THIS ONLY IF THE FINAL TRANSFORM IS BSpline
linearTransform: (a pathlike object or string representing an existing file)
  (optional) Filename to which save the estimated transform. NOTE: You must set at least one output object (either a deformed image or a transform. NOTE: USE THIS ONLY IF THE FINAL TRANSFORM IS ---NOT--- BSpline
outputVolume: (a pathlike object or string representing an existing file)
  (optional) Output image for registration. NOTE: You must select either the outputTransform or the outputVolume option.
outputFixedVolumeROI: (a pathlike object or string representing an existing file)
  The ROI automatically found in fixed image, ONLY FOR ROIAUTO mode.
outputMovingVolumeROI: (a pathlike object or string representing an existing file)
  The ROI automatically found in moving image, ONLY FOR ROIAUTO mode.
strippedOutputTransform: (a pathlike object or string representing an existing file)
  File name for the rigid component of the estimated affine transform. Can be used to rigidly register the moving image to the fixed image. NOTE: This value is overwritten if either bsplineTransform or linearTransform is set.
outputTransform: (a pathlike object or string representing an existing file)
  (optional) Filename to which save the (optional) estimated transform. NOTE: You must select either the outputTransform or the outputVolume option.

76.23 interfaces.slicer.registration.brainsresample

76.23.1 BRAINSResample

Link to code
Wraps the executable command "BRAINSResample".
title: Resample Image (BRAINS)
category: Registration
description: This program resamples an image image using a deformation field or a transform (BSpline, Affine, Rigid, etc.).
version: 3.0.0
license: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt
contributor: This tool was developed by Vincent Magnotta, Greg Harris, and Hans Johnson.
acknowledgements: The development of this tool was supported by funding from grants NS050568 and NS40068 from the National Institute of Neurological Disorders and Stroke and grants MH31593, MH40856, from the National Institute of Mental Health.

Inputs:

[Optional]
inputVolume: (a pathlike object or string representing an existing file)
  Image To Warp
  argument: `--inputVolume %s`
referenceVolume: (a pathlike object or string representing an existing file)

(continues on next page)
Reference image used only to define the output space. If not specified, the warping is done in the same space as the image to warp.

argument: `--referenceVolume %s`

outputVolume: (a boolean or a pathlike object or string representing a file)

Resulting deformed image

argument: `--outputVolume %s`

pixelType: ('float' or 'short' or 'ushort' or 'int' or 'uint' or 'uchar' or 'binary')

Specifies the pixel type for the input/output images. The 'binary' pixel type uses a modified algorithm whereby the image is read in as unsigned char, a signed distance map is created, signed distance map is resampled, and then a thresholded image of type unsigned char is written to disk.

argument: `--pixelType %s`

deforrmationVolume: (a pathlike object or string representing an existing file)

Displacement Field to be used to warp the image

argument: `--deformationVolume %s`

warpTransform: (a pathlike object or string representing an existing file)

Filename for the BRAINSFit transform used in place of the deformation field

argument: `--warpTransform %s`

interpolationMode: ('NearestNeighbor' or 'Linear' or 'ResampleInPlace' or 'BSpline' or 'WindowedSinc' or 'Hamming' or 'Cosine' or 'Welch' or 'Lanczos' or 'Blackman')

Type of interpolation to be used when applying transform to moving volume. Options are Linear, ResampleInPlace, NearestNeighbor, BSpline, or WindowedSinc

argument: `--interpolationMode %s`

inverseTransform: (a boolean)

True/False is to compute inverse of given transformation. Default is false

argument: `--inverseTransform`

defaultValue: (a float)

Default voxel value

argument: `--defaultValue %f`

gridSpacing: (a list of items which are an integer (int or long))

Add warped grid to output image to help show the deformation that occured with specified spacing. A spacing of 0 in a dimenasion indicates that grid lines should be rendered to fall exactly (i.e. do not allow displacements off that plane). This is useful for making a 2D image of grid lines from the 3D space

argument: `--gridSpacing %s`

numberOfThreads: (an integer (int or long))

Explicitly specify the maximum number of threads to use.

argument: `--numberOfThreads %d`

args: (a unicode string)

Additional parameters to the command

argument: `%%`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

Environment variables

Outputs:
outputVolume: (a pathlike object or string representing an existing file)  
Resulting deformed image

### 76.24 interfaces.slicer.registration.specialized

#### 76.24.1 ACPCTransform

**Link to code**
Wraps the executable command `ACPCTransform`.

title: ACPC Transform
category: Registration.Specialized
description: Calculate a transformation from two lists of fiducial points.
The ACPC line is two fiducial points, one at the anterior commissure and one at the posterior commissure. The resulting transform will bring the line connecting them to horizontal to the AP axis. The midline is a series of points defining the division between the hemispheres of the brain (the mid sagittal plane). The resulting transform will put the output volume with the mid sagittal plane lined up with the AS plane. Use the Filtering module to Resample Scalar/Vector/DWI Volume to apply the transformation to a volume.

version: 1.0
license: slicer3
contributor: Nicole Aucoin (SPL, BWH), Ron Kikinis (SPL, BWH)
acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

**Inputs:**

- acpc: (a list of items which are a list of from 3 to 3 items which are a float)  
  ACPC line, two fiducial points, one at the anterior commissure and one at the posterior commissure.  
  argument: ```--acpc %s...```  
- midline: (a list of items which are a list of from 3 to 3 items which are a float)  
  The midline is a series of points defining the division between the hemispheres of the brain (the mid sagittal plane).  
  argument: ```--midline %s...```  
- outputTransform: (a boolean or a pathlike object or string representing a file)  
  A transform filled in from the ACPC and Midline registration calculation  
  argument: ```--outputTransform %s```  
- debugSwitch: (a boolean)  
  Click if wish to see debugging output  
  argument: ```--debugSwitch```  
- args: (a unicode string)  
  Additional parameters to the command  
  argument: ```%s```  
- environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})  
  Environment variables

**Outputs:**
76.24.2 BRAINSDemonWarp

Link to code
Wraps the executable command "BRAINSDemonWarp".

title: Demon Registration (BRAINS)
category: Registration.Specialized
description: This program finds a deformation field to warp a moving image onto a fixed image. The images must be of the same signal kind, and contain an image of the same kind of object. This program uses the Thirion Demons warp software in ITK, the Insight Toolkit. Additional information is available at: http://www.nitrc.org/projects/brainsdemonwarp.

version: 3.0.0
license: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt
contributor: This tool was developed by Hans J. Johnson and Greg Harris.
acknowledgements: The development of this tool was supported by funding from grants NS050568 and NS40068 from the National Institute of Neurological Disorders and Stroke and grants MH31593, MH40856, from the National Institute of Mental Health.

Inputs:

[Optional]
movingVolume: (a pathlike object or string representing an existing file)
  Required: input moving image
  argument: `--movingVolume %s`

fixedVolume: (a pathlike object or string representing an existing file)
  Required: input fixed (target) image
  argument: `--fixedVolume %s`

inputPixelType: ('float' or 'short' or 'ushort' or 'int' or 'uchar')
  Input volumes will be typecast to this format:
  float|short|ushort|int|uchar
  argument: `--inputPixelType %s`

outputVolume: (a boolean or a pathlike object or string representing a file)
  Required: output resampled moving image (will have the same physical space as the fixedVolume).
  argument: `--outputVolume %s`

outputDisplacementFieldVolume: (a boolean or a pathlike object or string representing a file)
  Output deformation field vector image (will have the same physical space as the fixedVolume).
  argument: `--outputDisplacementFieldVolume %s`

outputPixelType: ('float' or 'short' or 'ushort' or 'int' or 'uchar')
  Output volume will be typecast to this format:
  float|short|ushort|int|uchar
  argument: `--outputPixelType %s`

interpolationMode: ('NearestNeighbor' or 'Linear' or 'ResampleInPlace' or 'BSpline' or 'WindowedSinc' or 'Hamming' or 'Cosine' or 'Welch' or 'Lanczos' or 'Blackman')
  Type of interpolation to be used when applying transform to moving
volume. Options are Linear, ResampleInPlace, NearestNeighbor, BSpline, or WindowedSinc
argument: ``--interpolationMode %s``

registrationFilterType: ('Demons' or 'FastSymmetricForces' or 'Diffeomorphic')
Registration Filter Type: Demons|FastSymmetricForces|Diffeomorphic
argument: ``--registrationFilterType %s``

smoothDisplacementFieldSigma: (a float)
A gaussian smoothing value to be applied to the deformation field at each iteration.
argument: ``--smoothDisplacementFieldSigma %f``

numberOfPyramidLevels: (an integer (int or long))
Number of image pyramid levels to use in the multi-resolution registration.
argument: ``--numberOfPyramidLevels %d``

minimumFixedPyramid: (a list of items which are an integer (int or long))
The shrink factor for the first level of the fixed image pyramid.
(i.e. start at 1/16 scale, then 1/8, then 1/4, then 1/2, and finally full scale)
argument: ``--minimumFixedPyramid %s``

minimumMovingPyramid: (a list of items which are an integer (int or long))
The shrink factor for the first level of the moving image pyramid.
(i.e. start at 1/16 scale, then 1/8, then 1/4, then 1/2, and finally full scale)
argument: ``--minimumMovingPyramid %s``

arrayOfPyramidLevelIterations: (a list of items which are an integer (int or long))
The number of iterations for each pyramid level
argument: ``--arrayOfPyramidLevelIterations %s``

histogramMatch: (a boolean)
Histogram Match the input images. This is suitable for images of the same modality that may have different absolute scales, but the same overall intensity profile.
argument: ``--histogramMatch``

numberOfHistogramBins: (an integer (int or long))
The number of histogram levels
argument: ``--numberOfHistogramBins %d``

numberOfMatchPoints: (an integer (int or long))
The number of match points for histogramMatch
argument: ``--numberOfMatchPoints %d``

medianFilterSize: (a list of items which are an integer (int or long))
Median filter radius in all 3 directions. When images have a lot of salt and pepper noise, this step can improve the registration.
argument: ``--medianFilterSize %s``

initializeWithDisplacementField: (a pathlike object or string representing an existing file)
Initial deformation field vector image file name
argument: ``--initializeWithDisplacementField %s``

initializeWithTransform: (a pathlike object or string representing an existing file)
Initial Transform filename
argument: ``--initializeWithTransform %s``

maskProcessingMode: ('NOMASK' or 'ROIAUTO' or 'ROI' or 'BOBF')
What mode to use for using the masks: NOMASK|ROIAUTO|ROI|BOBF, If
ROIAUTO is choosen, then the mask is implicitly defined using a otsu foreground and hole filling algorithm. Where the Region Of Interest mode uses the masks to define what parts of the image should be used for computing the deformation field. Brain Only Background Fill uses the masks to pre-process the input images by clipping and filling in the background with a predefined value.

argument: `--maskProcessingMode %s`

fixedBinaryVolume: (a pathlike object or string representing an existing file)
Mask filename for desired region of interest in the Fixed image.
argument: `--fixedBinaryVolume %s`

movingBinaryVolume: (a pathlike object or string representing an existing file)
Mask filename for desired region of interest in the Moving image.
argument: `--movingBinaryVolume %s`

lowerThresholdForBOBF: (an integer (int or long))
Lower threshold for performing BOBF
argument: `--lowerThresholdForBOBF %d`

upperThresholdForBOBF: (an integer (int or long))
Upper threshold for performing BOBF
argument: `--upperThresholdForBOBF %d`

backgroundFillValue: (an integer (int or long))
Replacement value to overwrite background when performing BOBF
argument: `--backgroundFillValue %d`

seedForBOBF: (a list of items which are an integer (int or long))
Coordinates in all 3 directions for Seed when performing BOBF
argument: `--seedForBOBF %s`

neighborhoodForBOBF: (a list of items which are an integer (int or long))
Neighborhood in all 3 directions to be included when performing BOBF
argument: `--neighborhoodForBOBF %s`

outputDisplacementFieldPrefix: (a unicode string)
Displacement field filename prefix for writing separate x, y, and z component images
argument: `--outputDisplacementFieldPrefix %s`

outputCheckerboardVolume: (a boolean or a pathlike object or string representing a file)
Generate a checkerboard image volume between the fixedVolume and the deformed movingVolume.
argument: `--outputCheckerboardVolume %s`

checkerboardPatternSubdivisions: (a list of items which are an integer (int or long))
Number of Checkerboard subdivisions in all 3 directions
argument: `--checkerboardPatternSubdivisions %s`

outputNormalized: (a boolean)
Flag to warp and write the normalized images to output. In normalized images the image values are fit-scaled to be between 0 and the maximum storage type value.
argument: `--outputNormalized`

outputDebug: (a boolean)
Flag to write debugging images after each step.
argument: `--outputDebug`

gradient_type: ('0' or '1' or '2')
Type of gradient used for computing the demons force (0 is symmetrized, 1 is fixed image, 2 is moving image)
argument: `--gradient_type %s`

upFieldSmoothing: (a float)
Smoothing sigma for the update field at each iteration
argument: ``--upFieldSmoothing %f``

max_step_length: (a float)
    Maximum length of an update vector (0: no restriction)
argument: ``--max_step_length %f``

use_vanilla_dem: (a boolean)
    Run vanilla demons algorithm
argument: ``--use_vanilla_dem``

gui: (a boolean)
    Display intermediate image volumes for debugging
argument: ``--gui``

promptUser: (a boolean)
    Prompt the user to hit enter each time an image is sent to the DebugImageViewer
argument: ``--promptUser``

numberOfBCHApproximationTerms: (an integer (int or long))
    Number of terms in the BCH expansion
argument: ``--numberOfBCHApproximationTerms %d``

numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
argument: ``--numberOfThreads %d``

args: (a unicode string)
    Additional parameters to the command
argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})  
Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing file)
    Required: output resampled moving image (will have the same physical space as the fixedVolume).

outputDisplacementFieldVolume: (a pathlike object or string representing an existing file)
    Output deformation field vector image (will have the same physical space as the fixedVolume).

outputCheckerboardVolume: (a pathlike object or string representing an existing file)
    Generate a checkerboard image volume between the fixedVolume and the deformed movingVolume.

76.24.3 FiducialRegistration

Link to code
Wraps the executable command “FiducialRegistration “.

title: Fiducial Registration
category: Registration.Specialized
description: Computes a rigid, similarity or affine transform from a matched list of fiducials
version: 0.1.0.$Revision$
contributor: Casey B Goodlett (Kitware), Dominik Meier (SPL, BWH)
acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded
by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```plaintext
[Optional]
fixedLandmarks: (a list of items which are a list of from 3 to 3
items which are a float)
    Ordered list of landmarks in the fixed image
    argument: '``--fixedLandmarks %s...``'
movingLandmarks: (a list of items which are a list of from 3 to 3
items which are a float)
    Ordered list of landmarks in the moving image
    argument: '``--movingLandmarks %s...``'
saveTransform: (a boolean or a pathlike object or string representing
    a file)
    Save the transform that results from registration
    argument: '``--saveTransform %s``'
transformType: ('Translation' or 'Rigid' or 'Similarity')
    Type of transform to produce
    argument: '``--transformType %s``'
rms: (a float)
    Display RMS Error.
    argument: '``--rms %f``'
outputMessage: (a unicode string)
    Provides more information on the output
    argument: '``--outputMessage %s``'
args: (a unicode string)
    Additional parameters to the command
    argument: '``%s``'
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})
    Environment variables
```

Outputs:

```plaintext
saveTransform: (a pathlike object or string representing an existing
    file)
    Save the transform that results from registration
```

76.24.4 VBRAINSDemonWarp

Link to code

Wraps the executable command "VBRAINSDemonWarp".

title: Vector Demon Registration (BRAINS)
category: Registration.Specialized
description: This program finds a deformation field to warp a moving image onto a fixed image. The images
    must be of the same signal kind, and contain an image of the same kind of object. This program uses
the Thirion Demons warp software in ITK, the Insight Toolkit. Additional information is available at:

version: 3.0.0
license: https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt
contributor: This tool was developed by Hans J. Johnson and Greg Harris.
acknowledgements: The development of this tool was supported by funding from grants NS050568 and
NS40068 from the National Institute of Neurological Disorders and Stroke and grants MH31593, MH40856,
from the National Institute of Mental Health.

Inputs:
movingVolume: (a list of items which are a pathlike object or string representing an existing file)
  Required: input moving image
  argument: '--movingVolume %s...'

fixedVolume: (a list of items which are a pathlike object or string representing an existing file)
  Required: input fixed (target) image
  argument: '--fixedVolume %s...'

inputPixelType: ('float' or 'short' or 'ushort' or 'int' or 'uchar')
  Input volumes will be typecast to this format:
  float|short|ushort|int|uchar
  argument: '--inputPixelType %s'

outputVolume: (a boolean or a pathlike object or string representing a file)
  Required: output resampled moving image (will have the same physical space as the fixedVolume).
  argument: '--outputVolume %s'

outputDisplacementFieldVolume: (a boolean or a pathlike object or string representing a file)
  Output deformation field vector image (will have the same physical space as the fixedVolume).
  argument: '--outputDisplacementFieldVolume %s'

outputPixelType: ('float' or 'short' or 'ushort' or 'int' or 'uchar')
  outputVolume will be typecast to this format:
  float|short|ushort|int|uchar
  argument: '--outputPixelType %s'

interpolationMode: ('NearestNeighbor' or 'Linear' or 'ResampleInPlace' or 'BSpline' or 'WindowedSinc' or 'Hamming' or 'Cosine' or 'Welch' or 'Lanczos' or 'Blackman')
  Type of interpolation to be used when applying transform to moving volume. Options are Linear, ResampleInPlace, NearestNeighbor, BSpline, or WindowedSinc
  argument: '--interpolationMode %s'

registrationFilterType: ('Demons' or 'FastSymmetricForces' or 'Diffeomorphic' or 'LogDemons' or 'SymmetricLogDemons')
  Registration Filter Type: Demons|FastSymmetricForces|Diffeomorphic|LogDemons|SymmetricLogDemons
  argument: '--registrationFilterType %s'

smoothDisplacementFieldSigma: (a float)
  A gaussian smoothing value to be applied to the deformation field at each iteration.
  argument: '--smoothDisplacementFieldSigma %f'

numberOfPyramidLevels: (an integer (int or long))
  Number of image pyramid levels to use in the multi-resolution registration.
  argument: '--numberOfPyramidLevels %d'

minimumFixedPyramid: (a list of items which are an integer (int or long))
  The shrink factor for the first level of the fixed image pyramid.
  (i.e. start at 1/16 scale, then 1/8, then 1/4, then 1/2, and finally full scale)
  argument: '--minimumFixedPyramid %s'

minimumMovingPyramid: (a list of items which are an integer (int or long))
  The shrink factor for the first level of the moving image pyramid.
  (i.e. start at 1/16 scale, then 1/8, then 1/4, then 1/2, and finally full scale)
  argument: '--minimumMovingPyramid %s'
full scale
argument: `--minimumMovingPyramid %s`
arrayOfPyramidLevelIterations: (a list of items which are an integer
(int or long))
The number of iterations for each pyramid level
argument: `--arrayOfPyramidLevelIterations %s`
histogramMatch: (a boolean)
Histogram Match the input images. This is suitable for images of the
same modality that may have different absolute scales, but the same
overall intensity profile.
argument: `--histogramMatch`
numberOfHistogramBins: (an integer (int or long))
The number of histogram levels
argument: `--numberOfHistogramBins %d`
numberOfMatchPoints: (an integer (int or long))
The number of match points for histogramMatch
argument: `--numberOfMatchPoints %d`
medianFilterSize: (a list of items which are an integer (int or
long))
Median filter radius in all 3 directions. When images have a lot of
salt and pepper noise, this step can improve the registration.
argument: `--medianFilterSize %s`
initializeWithDisplacementField: (a pathlike object or string
representing an existing file)
Initial deformation field vector image file name
argument: `--initializeWithDisplacementField %s`
initializeWithTransform: (a pathlike object or string representing an
existing file)
Initial Transform filename
argument: `--initializeWithTransform %s`
makeBOBF: (a boolean)
Flag to make Brain-Only Background-Filled versions of the input and
target volumes.
argument: `--makeBOBF`
fixedBinaryVolume: (a pathlike object or string representing an
existing file)
Mask filename for desired region of interest in the Fixed image.
argument: `--fixedBinaryVolume %s`
movingBinaryVolume: (a pathlike object or string representing an
existing file)
Mask filename for desired region of interest in the Moving image.
argument: `--movingBinaryVolume %s`
lowerThresholdForBOBF: (an integer (int or long))
Lower threshold for performing BOBF
argument: `--lowerThresholdForBOBF %d`
upperThresholdForBOBF: (an integer (int or long))
Upper threshold for performing BOBF
argument: `--upperThresholdForBOBF %d`
backgroundFillValue: (an integer (int or long))
Replacement value to overwrite background when performing BOBF
argument: `--backgroundFillValue %d`
seedForBOBF: (a list of items which are an integer (int or long))
coordinates in all 3 directions for Seed when performing BOBF
argument: `--seedForBOBF %s`
neighborhoodForBOBF: (a list of items which are an integer (int or
long))
neighborhood in all 3 directions to be included when performing BOBF
argument: ``--neighborhoodForBOBF %s``

outputDisplacementFieldPrefix: (a unicode string)
Displacement field filename prefix for writing separate x, y, and z component images
argument: ``--outputDisplacementFieldPrefix %s``

outputCheckerboardVolume: (a boolean or a pathlike object or string representing a file)
Generate a checkerboard image volume between the fixedVolume and the deformed movingVolume.
argument: ``--outputCheckerboardVolume %s``

checkerboardPatternSubdivisions: (a list of items which are an integer (int or long))
Number of Checkerboard subdivisions in all 3 directions
argument: ``--checkerboardPatternSubdivisions %s``

outputNormalized: (a boolean)
Flag to warp and write the normalized images to output. In normalized images the image values are fit-scaled to be between 0 and the maximum storage type value.
argument: ``--outputNormalized``

outputDebug: (a boolean)
Flag to write debugging images after each step.
argument: ``--outputDebug``

weightFactors: (a list of items which are a float)
Weight factors for each input images
argument: ``--weightFactors %s``

gradient_type: ('0' or '1' or '2')
Type of gradient used for computing the demons force (0 is symmetrized, 1 is fixed image, 2 is moving image)
argument: ``--gradient_type %s``

upFieldSmoothing: (a float)
Smoothing sigma for the update field at each iteration
argument: ``--upFieldSmoothing %f``

max_step_length: (a float)
Maximum length of an update vector (0: no restriction)
argument: ``--max_step_length %f``

use_vanilla_dem: (a boolean)
Run vanilla demons algorithm
argument: ``--use_vanilla_dem``

gui: (a boolean)
Display intermediate image volumes for debugging
argument: ``--gui``

promptUser: (a boolean)
Prompt the user to hit enter each time an image is sent to the DebugImageViewer
argument: ``--promptUser``

numberOfBCHApproximationTerms: (an integer (int or long))
Number of terms in the BCH expansion
argument: ``--numberOfBCHApproximationTerms %d``

numberOfThreads: (an integer (int or long))
Explicitly specify the maximum number of threads to use.
argument: ``--numberOfThreads %d``

args: (a unicode string)
Additional parameters to the command
argument: ``%s``

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

<table>
<thead>
<tr>
<th>Output Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outputVolume: (a pathlike object or string representing an existing file)</td>
<td>Required: output resampled moving image (will have the same physical space as the fixedVolume).</td>
</tr>
<tr>
<td>outputDisplacementFieldVolume: (a pathlike object or string representing an existing file)</td>
<td>Output deformation field vector image (will have the same physical space as the fixedVolume).</td>
</tr>
<tr>
<td>outputCheckerboardVolume: (a pathlike object or string representing an existing file)</td>
<td>Generate a checkerboard image volume between the fixedVolume and the deformed movingVolume.</td>
</tr>
</tbody>
</table>

76.25 interfaces.slicer.segmentation.simpleregiongrowingsegmentation

76.25.1 SimpleRegionGrowingSegmentation

Link to code
Wraps the executable command "SimpleRegionGrowingSegmentation".

title: Simple Region Growing Segmentation
category: Segmentation
description: A simple region growing segmentation algorithm based on intensity statistics. To create a list of fiducials (Seeds) for this algorithm, click on the tool bar icon of an arrow pointing to a starburst fiducial to enter the 'place a new object mode' and then use the fiducials module. This module uses the Slicer Command Line Interface (CLI) and the ITK filters CurvatureFlowImageFilter and ConfidenceConnectedImageFilter.
version: 0.1.0.$Revision: 19904 $(alpha)
contributor: Jim Miller (GE)
acknowledgements: This command module was derived from Insight/Examples (copyright) Insight Software Consortium

Inputs:

<table>
<thead>
<tr>
<th>Optional Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>smoothingIterations: (an integer (int or long))</td>
<td>Number of smoothing iterations</td>
</tr>
<tr>
<td>argument: '--smoothingIterations %d'</td>
<td></td>
</tr>
<tr>
<td>timestep: (a float)</td>
<td>Timestep for curvature flow</td>
</tr>
<tr>
<td>argument: '--timestep %f'</td>
<td></td>
</tr>
<tr>
<td>iterations: (an integer (int or long))</td>
<td>Number of iterations of region growing</td>
</tr>
<tr>
<td>argument: '--iterations %d'</td>
<td></td>
</tr>
<tr>
<td>multiplier: (a float)</td>
<td>Number of standard deviations to include in intensity model</td>
</tr>
<tr>
<td>argument: '--multiplier %f'</td>
<td></td>
</tr>
<tr>
<td>neighborhood: (an integer (int or long))</td>
<td>The radius of the neighborhood over which to calculate intensity model</td>
</tr>
<tr>
<td>argument: '--neighborhood %d'</td>
<td></td>
</tr>
</tbody>
</table>
labelvalue: (an integer (int or long))
    The integer value (0-255) to use for the segmentation results. This
    will determine the color of the segmentation that will be generated
    by the Region growing algorithm
    argument: `--labelvalue %d`
seed: (a list of items which are a list of from 3 to 3 items which
    are a float)
    Seed point(s) for region growing
    argument: `--seed %s...`
inputVolume: (a pathlike object or string representing an existing
    file)
    Input volume to be filtered
    argument: `-%s`, position: -2
outputVolume: (a boolean or a pathlike object or string representing
    a file)
    Output filtered
    argument: `-%s`, position: -1
args: (a unicode string)
    Additional parameters to the command
    argument: `-%s`
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})
The input image for finding the largest region filled mask.
argument: `--inputVolume %s`

outputROIMaskVolume: (a boolean or a pathlike object or string representing a file)
The ROI automatically found from the input image.
argument: `--outputROI_maskVolume %s`

outputClippedVolumeROI: (a boolean or a pathlike object or string representing a file)
The inputVolume clipped to the region of the brain mask.
argument: `--outputClippedVolumeROI %s`

otsuPercentileThreshold: (a float)
Parameter to the Otsu threshold algorithm.
argument: `--otsuPercentileThreshold %f`

thresholdCorrectionFactor: (a float)
A factor to scale the Otsu algorithm's result threshold, in case clipping mangles the image.
argument: `--thresholdCorrectionFactor %f`

closingSize: (a float)
The Closing Size (in millimeters) for largest connected filled mask. This value is divided by image spacing and rounded to the next largest voxel number.
argument: `--closingSize %f`

ROIAutoDilateSize: (a float)
This flag is only relevant when using ROIAUTO mode for initializing masks. It defines the final dilation size to capture a bit of background outside the tissue region. At setting of 10mm has been shown to help regularize a BSpline registration type so that there is some background constraints to match the edges of the head better.
argument: `--ROIAutoDilateSize %f`

outputVolumePixelType: ('float' or 'short' or 'ushort' or 'int' or 'uint' or 'uchar')
The output image Pixel Type is the scalar datatype for representation of the Output Volume.
argument: `--outputVolumePixelType %s`

numberOfThreads: (an integer (int or long))
Explicitly specify the maximum number of threads to use.
argument: `--numberOfThreads %d`

args: (a unicode string)
Additional parameters to the command
argument: `%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: { })

Environment variables

Outputs:

outputROI_maskVolume: (a pathlike object or string representing an existing file)
The ROI automatically found from the input image.
outputClippedVolumeROI: (a pathlike object or string representing an existing file)
The inputVolume clipped to the region of the brain mask.
76.26.2 EMSegmentCommandLine

Link to code
Wraps the executable command "EMSegmentCommandLine".

title: EMSegment Command-line
category: Segmentation.Specialized
description: This module is used to simplify the process of segmenting large collections of images by providing a command line interface to the EMSegment algorithm for script and batch processing.

contributor: Sebastien Barre, Brad Davis, Kilian Pohl, Polina Golland, Yumin Yuan, Daniel Haehn
acknowledgements: Many people and organizations have contributed to the funding, design, and development of the EMSegment algorithm and its various implementations.

Inputs:

[Optional]

mrmlSceneFileName: (a pathlike object or string representing an existing file)
  Active MRML scene that contains EMSegment algorithm parameters.
  argument: '--mrmlSceneFileName %s'

resultVolumeFileName: (a boolean or a pathlike object or string representing a file)
  The file name that the segmentation result volume will be written to.
  argument: '--resultVolumeFileName %s'

targetVolumeFileNames: (a list of items which are a pathlike object or string representing an existing file)
  File names of target volumes (to be segmented). The number of target images must be equal to the number of target images specified in the parameter set, and these images must be spatially aligned.
  argument: '--targetVolumeFileNames %s...'

intermediateResultsDirectory: (a pathlike object or string representing an existing directory)
  Directory where EMSegmenter will write intermediate data (e.g., aligned atlas data).
  argument: '--intermediateResultsDirectory %s'

parametersMRMLNodeName: (a unicode string)
  The name of the EMSegment parameters node within the active MRML scene. Leave blank for default.
  argument: '--parametersMRMLNodeName %s'

disableMultithreading: (an integer (int or long))
  Disable multithreading for the EMSegmenter algorithm only!
  Preprocessing might still run in multi-threaded mode. -1: Do not overwrite default value. 0: Disable. 1: Enable.
  argument: '--disableMultithreading %d'

dontUpdateIntermediateData: (an integer (int or long))
  Disable update of intermediate results. -1: Do not overwrite default value. 0: Disable. 1: Enable.
  argument: '--dontUpdateIntermediateData %d'

verbose: (a boolean)
  Enable verbose output.
  argument: '--verbose '

loadTargetCentered: (a boolean)
  Read target files centered.
  argument: '--loadTargetCentered '

loadAtlasNonCentered: (a boolean)
  Read atlas files non-centered.
  argument: '--loadAtlasNonCentered '

(continues on next page)
argument: '--loadAtlasNonCentered`

taskPreProcessingSetting: (a unicode string)
    Specifies the different task parameter. Leave blank for default.
    argument: '--taskPreProcessingSetting %s'
keepTempFiles: (a boolean)
    If flag is set then at the end of command the temporary files are
    not removed
    argument: '--keepTempFiles`
resultStandardVolumeFileName: (a pathlike object or string
    representing an existing file)
    Used for testing. Compare segmentation results to this image and
    return EXIT_FAILURE if they do not match.
    argument: '--resultStandardVolumeFileName %s'
dontWriteResults: (a boolean)
    Used for testing. Don't actually write the resulting labelmap to
    disk.
    argument: '--dontWriteResults`
generateEmptyMRMLSceneAndQuit: (a boolean or a pathlike object or
    string representing a file)
    Used for testing. Only write a scene with default mrml parameters.
    argument: '--generateEmptyMRMLSceneAndQuit %s'
resultMRMLSceneFileName: (a boolean or a pathlike object or string
    representing a file)
    Write out the MRML scene after command line substitutions have been
    made.
    argument: '--resultMRMLSceneFileName %s'
disableCompression: (a boolean)
    Don't use compression when writing result image to disk.
    argument: '--disableCompression'
atlasVolumeFileNames: (a list of items which are a pathlike object or
    string representing an existing file)
    Use an alternative atlas to the one that is specified by the mrml
    file - note the order matters!
    argument: '--atlasVolumeFileNames %s...'
registrationPackage: (a unicode string)
    specify the registration package for preprocessing (CMTK or BRAINS
    or PLASTIMATCH or DEMONS)
    argument: '--registrationPackage %s'
registrationAffineType: (an integer (int or long))
    specify the accuracy of the affine registration. -2: Do not
    overwrite default, -1: Test, 0: Disable, 1: Fast, 2: Accurate
    argument: '--registrationAffineType %d'
registrationDeformableType: (an integer (int or long))
    specify the accuracy of the deformable registration. -2: Do not
    overwrite default, -1: Test, 0: Disable, 1: Fast, 2: Accurate
    argument: '--registrationDeformableType %d'
args: (a unicode string)
    Additional parameters to the command
    argument: '%s'
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})
    Environment variables

Outputs:

resultVolumeFileName: (a pathlike object or string representing an
existing file)
The file name that the segmentation result volume will be written
to.
generateEmptyMRMLSceneAndQuit: (a pathlike object
representing an existing file)
Used for testing. Only write a scene with default mrml parameters.
resultMRMLSceneFileName: (a pathlike object or string
representing an
existing file)
Write out the MRML scene after command line substitutions have been
made.

76.26.3 RobustStatisticsSegmenter

Link to code
Wraps the executable command ‘‘RobustStatisticsSegmenter’’.
title: Robust Statistics Segments
category: Segmentation.Specialized
description: Active contour segmentation using robust statistic.
version: 1.0
RobustStatisticsSegmenter
contributor: Yi Gao (gatech), Allen Tannenbaum (gatech), Ron Kikinis (SPL, BWH)
acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded
by the National Institutes of Health

Inputs:
[Optional]
expectedVolume: (a float)
The approximate volume of the object, in mL.
argument: ‘‘--expectedVolume %f’’
intensityHomogeneity: (a float)
What is the homogeneity of intensity within the object? Given
constant intensity at 1.0 score and extreme fluctuating intensity at
0.
argument: ‘‘--intensityHomogeneity %f’’
curvatureWeight: (a float)
Given sphere 1.0 score and extreme rough bounday/surface 0 score,
what is the expected smoothness of the object?
argument: ‘‘--curvatureWeight %f’’
labelValue: (an integer (int or long))
Label value of the output image
argument: ‘‘--labelValue %d’’
maxRunningTime: (a float)
The program will stop if this time is reached.
argument: ‘‘--maxRunningTime %f’’
originalImageFileName: (a pathlike object or string representing an
existing file)
Original image to be segmented
argument: ‘‘%s’’, position: -3
labelImageFileName: (a pathlike object or string representing an
existing file)
Label image for initialization
argument: ‘‘%s’’, position: -2
segmentedImageFileName: (a boolean or a pathlike object or string
representing a file)
Segmented image argument: ``%s``, position: -1
args: (a unicode string)
   Additional parameters to the command argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

segmentedImageFileName: (a pathlike object or string representing an existing file)
   Segmented image

76.27 interfaces.slicer.surface

76.27.1 GrayscaleModelMaker

Link to code
Wraps the executable command "GrayscaleModelMaker".

title: Grayscale Model Maker
category: Surface Models
description: Create 3D surface models from grayscale data. This module uses Marching Cubes to create an isosurface at a given threshold. The resulting surface consists of triangles that separate a volume into regions below and above the threshold. The resulting surface can be smoothed and decimated. This model works on continuous data while the module Model Maker works on labeled (or discrete) data.

version: 3.0
license: slicer3
contributor: Nicole Aucoin (SPL, BWH), Bill Lorensen (GE)
aknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

[Optional]
InputVolume: (a pathlike object or string representing an existing file)
   Volume containing the input grayscale data.
   argument: ``%s``, position: -2
OutputGeometry: (a boolean or a pathlike object or string representing a file)
   Output that contains geometry model.
   argument: ``%s``, position: -1
threshold: (a float)
   Grayscale threshold of isosurface. The resulting surface of triangles separates the volume into voxels that lie above (inside) and below (outside) the threshold.
   argument: ``--threshold %f``
name: (a unicode string)
   Name to use for this model.
   argument: ``--name %s``
smooth: (an integer (int or long))
Number of smoothing iterations. If 0, no smoothing will be done.
argument: ```--smooth %d```

decimate: (a float)
Target reduction during decimation, as a decimal percentage reduction in the number of polygons. If 0, no decimation will be done.
argument: ```--decimate %f```

splitnormals: (a boolean)
Splitting normals is useful for visualizing sharp features. However it creates holes in surfaces which affect measurements
argument: ```--splitnormals```

pointnormals: (a boolean)
Calculate the point normals? Calculated point normals make the surface appear smooth. Without point normals, the surface will appear faceted.
argument: ```--pointnormals```

args: (a unicode string)
Additional parameters to the command
argument: ```%s```

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

OutputGeometry: (a pathlike object or string representing an existing file)
Output that contains geometry model.

76.27.2 LabelMapSmoothing

Link to code
Wraps the executable command ```LabelMapSmoothing```.

title: Label Map Smoothing
category: Surface Models
description: This filter smoothes a binary label map. With a label map as input, this filter runs an anti-alising algorithm followed by a Gaussian smoothing algorithm. The output is a smoothed label map.
version: 1.0

contributor: Dirk Padfield (GE), Josh Cates (Utah), Ross Whitaker (Utah)
acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149. This filter is based on work developed at the University of Utah, and implemented at GE Research.

Inputs:

[Optional]
labelToSmooth: (an integer (int or long))
The label to smooth. All others will be ignored. If no label is selected by the user, the maximum label in the image is chosen by default.
argument: ```--labelToSmooth %d```

numberOfIterations: (an integer (int or long))
The number of iterations of the level set AntiAliasing algorithm
argument: `--numberOfIterations %d`

maxRMSError: (a float)
The maximum RMS error.
argument: `--maxRMSError %f`

gaussianSigma: (a float)
The standard deviation of the Gaussian kernel
argument: `--gaussianSigma %f`

inputVolume: (a pathlike object or string representing an existing file)
Input label map to smooth
argument: `%%s`, position: -2

outputVolume: (a boolean or a pathlike object or string representing a file)
Smoothed label map
argument: `%%s`, position: -1

args: (a unicode string)
Additional parameters to the command
argument: `%%s`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:

outputVolume: (a pathlike object or string representing an existing file)
Smoothed label map

76.27.3 MergeModels

Link to code
Wraps the executable command "MergeModels ".

title: Merge Models
category: Surface Models
description: Merge the polydata from two input models and output a new model with the added polydata. Uses the vtkAppendPolyData filter. Works on .vtp and .vtk surface files.
version: $Revision$
contributor: Nicole Aucoin (SPL, BWH), Ron Kikinis (SPL, BWH), Daniel Haehn (SPL, BWH)
acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

[Optional]
Model1: (a pathlike object or string representing an existing file)
Model
argument: `%%s`, position: -3

Model2: (a pathlike object or string representing an existing file)
Model
argument: `%%s`, position: -2

ModelOutput: (a boolean or a pathlike object or string representing a file)
Model
argument: `%%s`, position: -1

(continues on next page)
**76.27.4 ModelMaker**

**Link to code**
Wraps the executable command `"ModelMaker "`.

title: Model Maker
category: Surface Models
description: Create 3D surface models from segmented data.

Models are imported into Slicer under a model hierarchy node in a MRML scene. The model colors are set by the color table associated with the input volume (these colours will only be visible if you load the model scene file).

If you specify a list of Labels, it will override any start/end label settings.

If you click `Generate All` it will override the list of labels and any start/end label settings.

You can set the number of smoothing iterations, target reduction in number of polygons (decimal percentage). Use 0 and 1 if you wish no smoothing nor decimation.

You can set the flags to split normals or generate point normals in this pane as well.

You can save a copy of the models after intermediate steps (marching cubes, smoothing, and decimation if not joint smoothing, otherwise just after decimation); these models are not saved in the mrml file, turn off deleting temporary files first in the python window:

```
<ip>slicer.modules.modelmaker.cliModuleLogic().DeleteTemporaryFilesOff()</ip>
```

version: 4.1
license: slicer4
contributor: Nicole Aucoin (SPL, BWH), Ron Kikinis (SPL, BWH), Bill Lorensen (GE)
aknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

**Inputs:**

- **InputVolume**: (a pathlike object or string representing an existing file)
  - Input label map. The Input Volume drop down menu is populated with the label map volumes that are present in the scene, select one from which to generate models.
  - argument: `"%s"`, position: -1

- **color**: (a pathlike object or string representing an existing file)
  - Color table to make labels to colors and objects
  - argument: `"--color %s"`

- **modelSceneFile**: (a boolean or a list of items which are a pathlike object or string representing a file)
  - Generated models, under a model hierarchy node. Models are imported into Slicer under a model hierarchy node, and their colors are set by the color table associated with the input label map volume. The model hierarchy node must be created before running the model maker.
by selecting Create New ModelHierarchy from the Models drop down menu. If you're running from the command line, a model hierarchy node in a new mrml scene will be created for you.

argument: `--modelSceneFile %s`

name: (a unicode string)

Name to use for this model. Any text entered in the entry box will be the starting string for the created model file names. The label number and the color name will also be part of the file name. If making multiple models, use this as a prefix to the label and color name.

argument: `--name %s`

generateAll: (a boolean)

Generate models for all labels in the input volume. select this option if you want to create all models that correspond to all values in a labelmap volume (using the Joint Smoothing option below is useful with this option). Ignores Labels, Start Label, End Label settings. Skips label 0.

argument: `--generateAll`

labels: (a list of items which are an integer (int or long))

A comma separated list of label values from which to make models. If you specify a list of Labels, it will override any start/end label settings. If you click Generate All Models it will override the list of labels and any start/end label settings.

argument: `--labels %s`

start: (an integer (int or long))

If you want to specify a continuous range of labels from which to generate models, enter the lower label here. Voxel value from which to start making models. Used instead of the label list to specify a range (make sure the label list is empty or it will override this).

argument: `--start %d`

decimate: (a float)

(continues on previous page)
Chose the target reduction in number of polygons as a decimal percentage (between 0 and 1) of the number of polygons. Specifies the percentage of triangles to be removed. For example, 0.1 means 10% reduction and 0.9 means 90% reduction.

```
argument: `--decimate %f`
```

**splitnormals**: (a boolean)

Splitting normals is useful for visualizing sharp features. However it creates holes in surfaces which affects measurements.

```
argument: `--splitnormals`
```

**pointnormals**: (a boolean)

Turn this flag on if you wish to calculate the normal vectors for the points.

```
argument: `--pointnormals`
```

**pad**: (a boolean)

Pad the input volume with zero value voxels on all 6 faces in order to ensure the production of closed surfaces. Sets the origin translation and extent translation so that the models still line up with the unpadded input volume.

```
argument: `--pad`
```

**saveIntermediateModels**: (a boolean)

You can save a copy of the models after each of the intermediate steps (marching cubes, smoothing, and decimation if not joint smoothing, otherwise just after decimation). These intermediate models are not saved in the mrml file, you have to load them manually after turning off deleting temporary files in they python console (View ->Python Interactor) using the following command slice r.modules.modelmaker.cliModuleLogic().DeleteTemporaryFilesOff().

```
argument: `--saveIntermediateModels`
```

**debug**: (a boolean)

Turn this flag on in order to see debugging output (look in the Error Log window that is accessed via the View menu)

```
argument: `--debug`
```

**args**: (a unicode string)

Additional parameters to the command

```
argument: `%s`
```

**environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

Environment variables

Outputs:

**modelSceneFile**: (a list of items which are a pathlike object or string representing an existing file)

Generated models, under a model hierarchy node. Models are imported into Slicer under a model hierarchy node, and their colors are set by the color table associated with the input label map volume. The model hierarchy node must be created before running the model maker, by selecting Create New ModelHierarchy from the Models drop down menu. If you're running from the command line, a model hierarchy node in a new mrml scene will be created for you.

---

**76.27.5 ModelToLabelMap**

*Link to code*

Wraps the executable command “ModelToLabelMap “.

title: Model To Label Map
category: Surface Models
description: Intersects an input model with an reference volume and produces an output label map.
version: 0.1.0.$Revision: 8643 $(alpha)
documentation-url: http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/PolyDataToLabelMap
contributor: Nicole Aucoin (SPL, BWH), Xiaodong Tao (GE)
acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

[Optional]
distance: (a float)
   Sample distance
   argument: `--distance %f`
InputVolume: (a pathlike object or string representing an existing file)
   Input volume
   argument: `--%s`, position: -3
surface: (a pathlike object or string representing an existing file)
   Model
   argument: `--%s`, position: -2
OutputVolume: (a boolean or a pathlike object or string representing a file)
   The label volume
   argument: `--%s`, position: -1
args: (a unicode string)
   Additional parameters to the command
   argument: `--%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

OutputVolume: (a pathlike object or string representing an existing file)
   The label volume

76.27.6 ProbeVolumeWithModel

Link to code
Wraps the executable command "ProbeVolumeWithModel".
title: Probe Volume With Model
category: Surface Models
description: Paint a model by a volume (using vtkProbeFilter).
version: 0.1.0.$Revision: 1892 $(alpha)
contributor: Lauren O’Donnell (SPL, BWH)
acknowledgements: BWH, NCIGT/LMI

Inputs:

[Optional]
InputVolume: (a pathlike object or string representing an existing file)
Volume to use to 'paint' the model
argument: ``%s``, position: -3
InputModel: (a pathlike object or string representing an existing file)
    Input model
    argument: ``%s``, position: -2
OutputModel: (a boolean or a pathlike object or string representing a file)
    Output 'painted' model
    argument: ``%s``, position: -1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
envir: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables

Outputs:
OutputModel: (a pathlike object or string representing an existing file)
    Output 'painted' model

76.28 interfaces.slicer.utilities

76.28.1 EMSegmentTransformToNewFormat

Link to code
Wraps the executable command "EMSegmentTransformToNewFormat".

title: Transform MRML Files to New EMSegmenter Standard
category: Utilities
description: Transform MRML Files to New EMSegmenter Standard

Inputs:

[Optional]
inputMRMLFileName: (a pathlike object or string representing an existing file)
    Active MRML scene that contains EMSegment algorithm parameters in the format before 3.6.3 - please include absolute file name in path.
    argument: ``--inputMRMLFileName %s``
outputMRMLFileName: (a boolean or a pathlike object or string representing a file)
    Write out the MRML scene after transformation to format 3.6.3 has been made. - has to be in the same directory as the input MRML file due to Slicer Core bug - please include absolute file name in path
    argument: ``--outputMRMLFileName %s``
templateFlag: (a boolean)
    Set to true if the transformed mrml file should be used as template file
    argument: ``--templateFlag``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
envir: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
Environment variables
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {()}

Environment variables

Outputs:

outputMRMLFileName: (a pathlike object or string representing an
existing file)
Write out the MRML scene after transformation to format 3.6.3 has
been made. - has to be in the same directory as the input MRML file
due to Slicer Core bug - please include absolute file name in path
77.1 interfaces.spm.model

77.1.1 EstimateContrast

Link to code
Use spm_contrasts to estimate contrasts of interest

Examples

```python
>>> import nipype.interfaces.spm as spm
>>> est = spm.EsteimateContrast()
>>> est.inputs.spm_mat_file = 'SPM.mat'
>>> cont1 = ('Task>Baseline','T', ['Task-Odd','Task-Even'],[0.5,0.5])
>>> cont2 = ('Task-Odd>Task-Even','T', ['Task-Odd','Task-Even'],[1,-1])
>>> contrasts = [cont1,cont2]
>>> est.inputs.contrasts = contrasts
>>> est.run() # doctest: +SKIP
```

Inputs:

- **spm_mat_file**: (a pathlike object or string representing an existing file)
  Absolute path to SPM.mat
- **contrasts**: (a list of items which are a tuple of the form: (a unicode string, 'T', a list of items which are a unicode string, a list of items which are a float) or a tuple of the form: (a unicode string, 'T', a list of items which are a unicode string, a list of items which are a float) or a tuple of the form: (a unicode string, 'T', a list of items which are a unicode string, a list of items which are a float))
  List of contrasts with each contrast being a list of the form:
nipype Documentation, Release 1.2.1

([{'name', 'stat'], [condition list], [weight list], [session list]])

If session list is None or not provided, all sessions are used. For F contrasts, the condition list should contain previously defined T-contrasts.

beta_images: (a list of items which are a pathlike object or string representing an existing file)
Parameter estimates of the design matrix
residual_image: (a pathlike object or string representing an existing file)
Mean-squared image of the residuals

[Optional]
use_derivs: (a boolean)
use derivatives for estimation
mutually_exclusive: group_contrast

mutually_exclusive: use_derivs

mutually_exclusive: group_contrast
higher level contrast

matlab_cmd: (a unicode string)
matlab command to use

paths: (a list of items which are a pathlike object or string representing a directory)
Paths to add to matlabpath

mfile: (a boolean, nipype default value: True)
Run m-code using m-file

use_mcr: (a boolean)
Run m-code using SPM MCR
use_v8struct: (a boolean, nipype default value: True)
Generate SPM8 and higher compatible jobs

Outputs:

con_images: (a list of items which are a pathlike object or string representing an existing file)
contrast images from a t-contrast

spmT_images: (a list of items which are a pathlike object or string representing an existing file)
stat images from a t-contrast

ess_images: (a list of items which are a pathlike object or string representing an existing file)
contrast images from an F-contrast

spmF_images: (a list of items which are a pathlike object or string representing an existing file)
stat images from an F-contrast

spm_mat_file: (a pathlike object or string representing an existing file)
Updated SPM mat file

References:
None

77.1.2 EstimateModel

Link to code

Use spm_spm to estimate the parameters of a model
http://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf#page=69
Examples

```python
>>> est = EstimateModel()
>>> est.inputs.spm_mat_file = 'SPM.mat'
>>> est.inputs.estimation_method = {'Classical': 1}
>>> est.run()  # doctest: +SKIP
```

Inputs:

- **Mandatory**
  - `spm_mat_file`: (a pathlike object or string representing an existing file)
    - Absolute path to SPM.mat
  - `estimation_method`: (a dictionary with keys which are 'Classical' or 'Bayesian2' or 'Bayesian' and with values which are any value)
    - Dictionary of either Classical: 1, Bayesian: 1, or Bayesian2: 1 (dict)

- **Optional**
  - `write_residuals`: (a boolean)
    - Write individual residual images
  - `flags`: (a dictionary with keys which are any value and with values which are any value)
    - Additional arguments
  - `matlab_cmd`: (a unicode string)
    - matlab command to use
  - `paths`: (a list of items which are a pathlike object or string representing a directory)
    - Paths to add to matlabpath
  - `mfile`: (a boolean, nipype default value: True)
    - Run m-code using m-file
  - `use_mcr`: (a boolean)
    - Run m-code using SPM MCR
  - `use_v8struct`: (a boolean, nipype default value: True)
    - Generate SPM8 and higher compatible jobs

Outputs:

- `mask_image`: (a pathlike object or string representing an existing file)
  - binary mask to constrain estimation
- `beta_images`: (a list of items which are a pathlike object or string representing an existing file)
  - design parameter estimates
- `residual_image`: (a pathlike object or string representing an existing file)
  - Mean-squared image of the residuals
- `residual_images`: (a list of items which are a pathlike object or string representing an existing file)
  - individual residual images (requires `write_residuals`)
- `RPVImage`: (a pathlike object or string representing an existing file)
  - Resels per voxel image
- `spm_mat_file`: (a pathlike object or string representing an existing file)
  - Updated SPM mat file
- `labels`: (a pathlike object or string representing an existing file)
  - label file
- `SDerror`: (a list of items which are a pathlike object or string representing an existing file)
Images of the standard deviation of the error
ARcoef: (a list of items which are a pathlike object or string representing an existing file)
Images of the AR coefficient
Cbetas: (a list of items which are a pathlike object or string representing an existing file)
Images of the parameter posteriors
SDbetas: (a list of items which are a pathlike object or string representing an existing file)
Images of the standard deviation of parameter posteriors

References:
None

77.1.3 FactorialDesign

Link to code
Base class for factorial designs
http://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf#page=77

Inputs:

[Optional]

- spm_mat_dir: (a pathlike object or string representing an existing directory)
  directory to store SPM.mat file (opt)
- covariates: (a list of items which are a dictionary with keys which are 'vector' or 'name' or 'interaction' or 'centering' and with values which are any value)
  covariate dictionary {vector, name, interaction, centering}
- threshold_mask_none: (a boolean)
  do not use threshold masking
  mutually_exclusive: threshold_mask_absolute, threshold_mask_relative
- threshold_mask_absolute: (a float)
  use an absolute threshold
  mutually_exclusive: threshold_mask_none, threshold_mask_relative
- threshold_mask_relative: (a float)
  threshold using a proportion of the global value
  mutually_exclusive: threshold_mask_absolute, threshold_mask_none
- use_implicit_threshold: (a boolean)
  use implicit mask NaNs or zeros to threshold
- explicit_mask_file: (a pathlike object or string representing a file)
  use an implicit mask file to threshold
- global_calc_omit: (a boolean)
  omit global calculation
  mutually_exclusive: global_calc_mean, global_calc_values
- global_calc_mean: (a boolean)
  use mean for global calculation
  mutually_exclusive: global_calc_omit, global_calc_values
- global_calc_values: (a list of items which are a float)
  omit global calculation
  mutually_exclusive: global_calc_mean, global_calc_omit
- no_grand_mean_scaling: (a boolean)
  do not perform grand mean scaling
- global_normalization: (1 or 2 or 3)
  global normalization None-1, Proportional-2, ANCOVA-3
matlab_cmd: (a unicode string)
matlab command to use
paths: (a list of items which are a pathlike object or string representing a directory)
Paths to add to matlabpath
mfile: (a boolean, nipype default value: True)
Run m-code using m-file
use_mcr: (a boolean)
Run m-code using SPM MCR
use_v8struct: (a boolean, nipipe default value: True)
Generate SPM8 and higher compatible jobs

Outputs:

spm_mat_file: (a pathlike object or string representing an existing file)
SPM mat file

References:
None

77.1.4 Level1Design

Link to code
Generate an SPM design matrix
http://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf#page=59

Examples

```python
>>> level1design = Level1Design()
>>> level1design.inputs.timing_units = 'secs'
>>> level1design.inputs.interscan_interval = 2.5
>>> level1design.inputs.bases = {'hrf': {'derivs': [0, 0]}}
>>> level1design.inputs.session_info = 'session_info.npz'
>>> level1design.inputs.flags = {'mthresh': 0.4}
>>> level1design.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
timing_units: ('secs' or 'scans')
units for specification of onsets
interscan_interval: (a float)
Interscan interval in secs
session_info: (any value)
Session specific information generated by `.modelgen.SpecifyModel`
bases: (a dictionary with keys which are 'hrf' or 'fourier' or
'fourier_han' or 'gamma' or 'fir' and with values which are any value)
dict {'name': ['basesparam1': val,...]}
name : string
Name of basis function (hrf, fourier, fourier_han, gamma, fir)
hrf :
derivs : 2-element list
Model HRF Derivatives. No derivatives: [0,0],
Time derivatives: [1,0], Time and Dispersion
derivatives: [1,1]
fourier, fourier_han, gamma, fir:
length : int
Post-stimulus window length (in seconds)
order : int
Number of basis functions

[Optional]

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spm_mat_dir</td>
<td>(a pathlike object or string representing an existing directory)</td>
</tr>
<tr>
<td></td>
<td>directory to store SPM.mat file (opt)</td>
</tr>
<tr>
<td>microtime_resolution</td>
<td>(an integer (int or long))</td>
</tr>
<tr>
<td></td>
<td>Number of time-bins per scan in secs (opt)</td>
</tr>
<tr>
<td>microtime_onset</td>
<td>(a float)</td>
</tr>
<tr>
<td></td>
<td>The onset/time-bin in seconds for alignment (opt)</td>
</tr>
<tr>
<td>factor_info</td>
<td>(a list of items which are a dictionary with keys which are 'name' or 'levels' and with values which are any value)</td>
</tr>
<tr>
<td></td>
<td>Factor specific information file (opt)</td>
</tr>
<tr>
<td>volterra_expansion_order</td>
<td>(1 or 2)</td>
</tr>
<tr>
<td></td>
<td>Model interactions - yes:1, no:2</td>
</tr>
<tr>
<td>global_intensity_normalization</td>
<td>('none' or 'scaling')</td>
</tr>
<tr>
<td></td>
<td>Global intensity normalization - scaling or none</td>
</tr>
<tr>
<td>mask_image</td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td></td>
<td>Image for explicitly masking the analysis</td>
</tr>
<tr>
<td>mask_threshold</td>
<td>('-Inf' or a float, nipype default value: -Inf)</td>
</tr>
<tr>
<td></td>
<td>Thresholding for the mask</td>
</tr>
<tr>
<td>model_serial_correlations</td>
<td>('AR(1)' or 'FAST' or 'none')</td>
</tr>
<tr>
<td></td>
<td>Model serial correlations AR(1), FAST or none. FAST is available in SPM12</td>
</tr>
<tr>
<td>flags</td>
<td>(a dictionary with keys which are any value and with values which are any value)</td>
</tr>
<tr>
<td></td>
<td>Additional arguments to the job, e.g., a common SPM operation is to modify the default masking threshold (mthresh)</td>
</tr>
<tr>
<td>matlab_cmd</td>
<td>(a unicode string)</td>
</tr>
<tr>
<td></td>
<td>Matlab command to use</td>
</tr>
<tr>
<td>paths</td>
<td>(a list of items which are a pathlike object or string representing a directory)</td>
</tr>
<tr>
<td></td>
<td>Paths to add to matlabpath</td>
</tr>
<tr>
<td>mfile</td>
<td>(a boolean, nipype default value: True)</td>
</tr>
<tr>
<td></td>
<td>Run m-code using m-file</td>
</tr>
<tr>
<td>use_mcr</td>
<td>(a boolean)</td>
</tr>
<tr>
<td></td>
<td>Run m-code using SPM MCR</td>
</tr>
<tr>
<td>use_v8struct</td>
<td>(a boolean, nipype default value: True)</td>
</tr>
<tr>
<td></td>
<td>Generate SPMS and higher compatible jobs</td>
</tr>
</tbody>
</table>

Outputs:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spm_mat_file</td>
<td>(a pathlike object or string representing an existing file)</td>
</tr>
<tr>
<td></td>
<td>SPM mat file</td>
</tr>
</tbody>
</table>

References:

None
77.1.5 MultipleRegressionDesign

Link to code
Create SPM design for multiple regression

Examples

```python
>>> mreg = MultipleRegressionDesign()
>>> mreg.inputs.in_files = ['cont1.nii', 'cont2.nii']
>>> mreg.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
- `in_files`: (a list of at least 2 items which are a pathlike object or string representing an existing file) List of files

[Optional]
- `include_intercept`: (a boolean, nipype default value: True) Include intercept in design
- `user_covariates`: (a list of items which are a dictionary with keys which are 'vector' or 'name' or 'centering' and with values which are any value) covariate dictionary {vector, name, centering}
- `spm_mat_dir`: (a pathlike object or string representing an existing directory) directory to store SPM.mat file (opt)
- `covariates`: (a list of items which are a dictionary with keys which are 'vector' or 'name' or 'interaction' or 'centering' and with values which are any value) covariate dictionary {vector, name, interaction, centering}
- `threshold_mask_none`: (a boolean) do not use threshold masking
- `threshold_mask_absolute`: (a float) use an absolute threshold
- `threshold_mask_relative`: (a float) threshold using a proportion of the global value
- `use_implicit_threshold`: (a boolean) use implicit mask NaNs or zeros to threshold
- `explicit_mask_file`: (a pathlike object or string representing a file) use an implicit mask file to threshold
- `global_calc_omit`: (a boolean) omit global calculation
- `global_calc_mean`: (a boolean) use mean for global calculation
- `global_calc_values`: (a list of items which are a float) omit global calculation
- `no_grand_mean_scaling`: (a boolean) do not perform grand mean scaling
- `global_normalization`: (1 or 2 or 3) global normalization None-1, Proportional-2, ANCOVA-3

(continues on next page)
matlab_cmd: (a unicode string)
    matlab command to use
paths: (a list of items which are a pathlike object or string
    representing a directory)
    Paths to add to matlabbapth
mfile: (a boolean, nipype default value: True)
    Run m-code using m-file
use_mcr: (a boolean)
    Run m-code using SPM MCR
use_v8struct: (a boolean, nipype default value: True)
    Generate SPM8 and higher compatible jobs

Outputs:

spm_mat_file: (a pathlike object or string representing an existing
    file)
    SPM mat file

References:

None

77.1.6 OneSampleTTestDesign

Link to code
Create SPM design for one sample t-test

Examples

>>> ttest = OneSampleTTestDesign()
>>> ttest.inputs.in_files = ['cont1.nii', 'cont2.nii']
>>> ttest.run() # doctest: +SKIP

Inputs:

[Mandatory]
in_files: (a list of at least 2 items which are a pathlike object or
    string representing an existing file)
    input files

[Optional]
spm_mat_dir: (a pathlike object or string representing an existing
    directory)
    directory to store SPM.mat file (opt)
covariates: (a list of items which are a dictionary with keys which
    are 'vector' or 'name' or 'interaction' or 'centering' and with
    values which are any value)
    covariate dictionary {vector, name, interaction, centering}
threshold_mask_none: (a boolean)
    do not use threshold masking
    mutually_exclusive: threshold_mask_absolute, threshold_mask_relative
threshold_mask_absolute: (a float)
    use an absolute threshold
    mutually_exclusive: threshold_mask_none, threshold_mask_relative
threshold_mask_relative: (a float)
    threshold using a proportion of the global value
mutually_exclusive: threshold_mask_absolute, threshold_mask_none
use_implicit_threshold: (a boolean)
  use implicit mask NaNs or zeros to threshold
explicit_mask_file: (a pathlike object or string representing a file)
  use an implicit mask file to threshold
global_calc_omit: (a boolean)
  omit global calculation
  mutually_exclusive: global_calc_mean, global_calc_values
global_calc_mean: (a boolean)
  use mean for global calculation
  mutually_exclusive: global_calc_omit, global_calc_values
global_calc_values: (a list of items which are a float)
  omit global calculation
  mutually_exclusive: global_calc_mean, global_calc_omit
no_grand_mean_scaling: (a boolean)
  do not perform grand mean scaling
global_normalization: (1 or 2 or 3)
  global normalization None-1, Proportional-2, ANCOVA-3
matlab_cmd: (a unicode string)
  matlab command to use
paths: (a list of items which are a pathlike object or string representing a directory)
  Paths to add to matlabpath
mfile: (a boolean, nipype default value: True)
  Run m-code using m-file
use_mcr: (a boolean)
  Run m-code using SPM MCR
use_v8struct: (a boolean, nipype default value: True)
  Generate SPM8 and higher compatible jobs

Outputs:

spm_mat_file: (a pathlike object or string representing an existing file)
  SPM mat file

References:
None

77.1.7 PairedTTestDesign

Link to code

Create SPM design for paired t-test

Examples

```python
>>> pttest = PairedTTestDesign()
>>> pttest.inputs.paired_files = [['cont1.nii','cont1a.nii'],['cont2.nii','cont2a.nii']]  
>>> pttest.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
paired_files: (a list of at least 2 items which are a list of from 2}
to 2 items which are a pathlike object or string representing an existing file

List of paired files

[Optional]
grand_mean_scaling: (a boolean)
    Perform grand mean scaling
ancova: (a boolean)
    Specify ancova-by-factor regressors
spm_mat_dir: (a pathlike object or string representing an existing directory)
    directory to store SPM.mat file (opt)
covariates: (a list of items which are a dictionary with keys which
    are 'vector' or 'name' or 'interaction' or 'centering' and with
    values which are any value)
covariate dictionary {vector, name, interaction, centering}
threshold_mask_none: (a boolean)
    do not use threshold masking
mutually_exclusive: threshold_mask_absolute, threshold_mask_relative
threshold_mask_absolute: (a float)
    use an absolute threshold
mutually_exclusive: threshold_mask_none, threshold_mask_relative
threshold_mask_relative: (a float)
    threshold using a proportion of the global value
mutually_exclusive: threshold_mask_absolute, threshold_mask_none
use_implicit_threshold: (a boolean)
    use implicit mask NaNs or zeros to threshold
explicit_mask_file: (a pathlike object or string representing a file)
    use an implicit mask file to threshold
global_calc_omit: (a boolean)
    omit global calculation
mutually_exclusive: global_calc_mean, global_calc_values
global_calc_mean: (a boolean)
    use mean for global calculation
mutually_exclusive: global_calc_omit, global_calc_values
global_calc_values: (a list of items which are a float)
    omit global calculation
mutually_exclusive: global_calc_mean, global_calc_omit
no_grand_mean_scaling: (a boolean)
    do not perform grand mean scaling
mutually_exclusive: global_calc_mean, global_calc_values

Lastly, it is important to note that...

Outputs:

spm_mat_file: (a pathlike object or string representing an existing file)
References:

None

77.1.8 Threshold

Link to code

Topological FDR thresholding based on cluster extent/size. Smoothness is estimated from GLM residuals but is assumed to be the same for all of the voxels.

Examples

```python
>>> thresh = Threshold()
>>> thresh.inputs.spm_mat_file = 'SPM.mat'
>>> thresh.inputs.stat_image = 'spmT_0001.img'
>>> thresh.inputs.contrast_index = 1
>>> thresh.inputs.extent_fdr_p_threshold = 0.05
>>> thresh.run() # doctest: +SKIP
```

Inputs:

- [Mandatory]
  - `spm_mat_file`: (a pathlike object or string representing an existing file)
  - `stat_image`: (a pathlike object or string representing an existing file)
  - `contrast_index`: (an integer (int or long))
    - which contrast in the SPM.mat to use

- [Optional]
  - `use_fwe_correction`: (a boolean, nipype default value: True)
    - whether to use FWE (Bonferroni) correction for initial threshold
      - (height_threshold_type has to be set to p-value)
  - `use_topo_fdr`: (a boolean, nipype default value: True)
    - whether to use FDR over cluster extent probabilities
  - `height_threshold`: (a float, nipype default value: 0.05)
    - value for initial thresholding (defining clusters)
  - `height_threshold_type`: ('p-value' or 'stat', nipype default value: 'p-value')
    - Is the cluster forming threshold a stat value or p-value?
  - `extent_fdr_p_threshold`: (a float, nipype default value: 0.05)
    - p threshold on FDR corrected cluster size probabilities
  - `extent_threshold`: (an integer (int or long), nipype default value: 0)
    - Minimum cluster size in voxels
  - `force_activation`: (a boolean, nipype default value: False)
    - In case no clusters survive the topological inference step this will pick a cluster with the highest sum of t-values. Use with care.
  - `matlab_cmd`: (a unicode string)
    - matlab command to use
  - `paths`: (a list of items which are a pathlike object or string)
representing a directory)
Paths to add to matlabpath
mfile: (a boolean, nipype default value: True)
Run m-code using m-file
use_mcr: (a boolean)
Run m-code using SPM MCR
use_v8struct: (a boolean, nipype default value: True)
Generate SPM8 and higher compatible jobs

Outputs:

thresholded_map: (a pathlike object or string representing an existing file)
n_clusters: (an integer (int or long))
pre_topo_fdr_map: (a pathlike object or string representing an existing file)
pre_topo_n_clusters: (an integer (int or long))
activation_forced: (a boolean)
cluster_forming_thr: (a float)

References:
None

77.1.9 ThresholdStatistics

Link to code
Given height and cluster size threshold calculate theoretical probabilities concerning false positives

Examples

```python
>>> thresh = ThresholdStatistics()
>>> thresh.inputs.spm_mat_file = 'SPM.mat'
>>> thresh.inputs.stat_image = 'spmT_0001.img'
>>> thresh.inputs.contrast_index = 1
>>> thresh.inputs.height_threshold = 4.56
>>> thresh.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
- spm_mat_file: (a pathlike object or string representing an existing file)
  absolute path to SPM.mat
- stat_image: (a pathlike object or string representing an existing file)
  stat image
- contrast_index: (an integer (int or long))
  which contrast in the SPM.mat to use
- height_threshold: (a float)
  stat value for initial thresholding (defining clusters)

[Optional]
- extent_threshold: (an integer (int or long), nipype default value: 0)
  Minimum cluster size in voxels
- matlab_cmd: (a unicode string)
matlab command to use
paths: (a list of items which are a pathlike object or string representing a directory)
   Paths to add to matlabpath
mfile: (a boolean, nipype default value: True)
   Run m-code using m-file
use_mcr: (a boolean)
   Run m-code using SPM MCR
use_v8struct: (a boolean, nipype default value: True)
   Generate SPM8 and higher compatible jobs

Outputs:

voxelwise_P_Bonf: (a float)
voxelwise_P_RF: (a float)
voxelwise_P_uncor: (a float)
voxelwise_P_FDR: (a float)
clusterwise_P_RF: (a float)
clusterwise_P_FDR: (a float)

References:
None

77.1.10 TwoSampleTTestDesign

Link to code
Create SPM design for two sample t-test

Examples

```python
>>> ttest = TwoSampleTTestDesign()
>>> ttest.inputs.group1_files = ['cont1.nii', 'cont2.nii']
>>> ttest.inputs.group2_files = ['cont1a.nii', 'cont2a.nii']
>>> ttest.run() # doctest: +SKIP
```

Inputs:

[Mandatory]

group1_files: (a list of at least 2 items which are a pathlike object or string representing an existing file)
   Group 1 input files
group2_files: (a list of at least 2 items which are a pathlike object or string representing an existing file)
   Group 2 input files

[Optional]
dependent: (a boolean)
   Are the measurements dependent between levels
unequal_variance: (a boolean)
   Are the variances equal or unequal between groups
spm_mat_dir: (a pathlike object or string representing an existing directory)
   directory to store SPM.mat file (opt)
covariates: (a list of items which are a dictionary with keys which are 'vector' or 'name' or 'interaction' or 'centering' and with
values which are any value)
covariate dictionary {vector, name, interaction, centering}
threshold_mask_none: (a boolean)
do not use threshold masking
mutually_exclusive: threshold_mask_absolute, threshold_mask_relative
threshold_mask_absolute: (a float)
use an absolute threshold
mutually_exclusive: threshold_mask_none, threshold_mask_relative
threshold_mask_relative: (a float)
threshold using a proportion of the global value
mutually_exclusive: threshold_mask_absolute, threshold_mask_none
use Implicit_threshold: (a boolean)
use implicit mask NaNs or zeros to threshold
explicit_mask_file: (a pathlike object or string representing a file)
use an implicit mask file to threshold
global_calc_omit: (a boolean)
omit global calculation
mutually_exclusive: global_calc_mean, global_calc_values
global_calc_mean: (a boolean)
use mean for global calculation
mutually_exclusive: global_calc_omit, global_calc_values
global_calc_values: (a list of items which are a float)
omit global calculation
mutually_exclusive: global_calc_mean, global_calc_omit
no_grand_mean_scaling: (a boolean)
do not perform grand mean scaling
global_normalization: (1 or 2 or 3)
    global normalization None−1, Proportional−2, ANCOVA−3
matlab_cmd: (a unicode string)
matlab command to use
paths: (a list of items which are a pathlike object or string
    representing a directory)
Paths to add to matlabpath
mfile: (a boolean, nipype default value: True)
Run m-code using m-file
use_mcr: (a boolean)
Run m-code using SPM MCR
use_v8struct: (a boolean, nipype default value: True)
Generate SPMS and higher compatible jobs

Outputs:

| spm_mat_file: (a pathlike object or string representing an existing file) |
| SPM mat file |

References:
None

77.2 interfaces.spm.preprocess

77.2.1 ApplyDeformations

Link to code
Inputs:
Mandatory

in_files: (a list of items which are a pathlike object or string representing an existing file)
deformation_field: (a pathlike object or string representing an existing file)
reference_volume: (a pathlike object or string representing an existing file)

Optional

interp: (0 <= a long integer <= 7)
    degree of b-spline used for interpolation
matlab_cmd: (a unicode string)
    matlab command to use
paths: (a list of items which are a pathlike object or string representing a directory)
    Paths to add to matlabpath
mfile: (a boolean, nipype default value: True)
    Run m-code using m-file
use_mcr: (a boolean)
    Run m-code using SPM MCR
use_v8struct: (a boolean, nipype default value: True)
    Generate SPMS and higher compatible jobs

Outputs:

out_files: (a list of items which are a pathlike object or string representing an existing file)

References:

None

77.2.2 Coregister

Link to code
Use spm_coreg for estimating cross-modality rigid body alignment
http://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf#page=39

Examples

```python
>>> import nipype.interfaces.spm as spm
>>> coreg = spm.Coregister()
>>> coreg.inputs.target = 'functional.nii'
>>> coreg.inputs.source = 'structural.nii'
>>> coreg.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
target: (a pathlike object or string representing an existing file)
    reference file to register to
source: (a list of items which are a pathlike object or string representing an existing file)
    file to register to target

(Optional)

jobtype: ('estwrite' or 'estimate' or 'write', nipype default value: 'estwrite')
estwrite)
    one of: estimate, write, estwrite
apply_to_files: (a list of items which are a pathlike object or
    string representing an existing file)
    files to apply transformation to
cost_function: ('mi' or 'nmi' or 'ecc' or 'ncc')
    cost function, one of:
    'mi' - Mutual Information,
    'nmi' - Normalised Mutual Information,
    'ecc' - Entropy Correlation Coefficient,
    'ncc' - Normalised Cross Correlation
fwhm: (a list of from 2 to 2 items which are a float)
    gaussian smoothing kernel width (mm)
separation: (a list of items which are a float)
    sampling separation in mm
tolerance: (a list of items which are a float)
    acceptable tolerance for each of 12 params
write_interp: (0 <= a long integer <= 7)
    degree of b-spline used for interpolation
write_wrap: (a list of from 3 to 3 items which are an integer (int or
    long))
    Check if interpolation should wrap in [x,y,z]
write_mask: (a boolean)
    True/False mask output image
out_prefix: (a string, nipype default value: r)
    coregistered output prefix
matlab_cmd: (a unicode string)
    matlab command to use
paths: (a list of items which are a pathlike object or string
    representing a directory)
    Paths to add to matlabpath
mfile: (a boolean, nipype default value: True)
    Run m-code using m-file
use_mcr: (a boolean)
    Run m-code using SPM MCR
use_v8struct: (a boolean, nipipe default value: True)
    Generate SPM8 and higher compatible jobs

Outputs:
coregistered_source: (a list of items which are a pathlike object or
    string representing an existing file)
    Coregistered source files
coregistered_files: (a list of items which are a pathlike object or
    string representing an existing file)
    Coregistered other files

References:
None

77.2.3 CreateWarped

Link to code
Apply a flow field estimated by DARTEL to create warped images
http://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf#page=190
Examples

```python
>>> import nipype.interfaces.spm as spm

>>> create_warped = spm.CreateWarped()

>>> create_warped.inputs.image_files = ['rc1s1.nii', 'rc1s2.nii']

>>> create_warped.inputs.flowfield_files = ['u_rc1s1_Template.nii', 'u_rc1s2_Template.nii']

>>> create_warped.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
- **image_files**: (a list of items which are a pathlike object or string representing an existing file)
  
  A list of files to be warped

- **flowfield_files**: (a list of items which are a pathlike object or string representing an existing file)
  
  DARTEL flow fields u_rc1*

[Optional]
- **iterations**: (0 <= a long integer <= 9)
  
  The number of iterations: log2(number of time steps)

- **interp**: (0 <= a long integer <= 7)
  
  degree of b-spline used for interpolation

- **modulate**: (a boolean)
  
  Modulate images

- **matlab_cmd**: (a unicode string)
  
  matlab command to use

- **paths**: (a list of items which are a pathlike object or string representing a directory)
  
  Paths to add to matlabpath

- **mfile**: (a boolean, nipype default value: True)
  
  Run m-code using m-file

- **use_mcr**: (a boolean)
  
  Run m-code using SPM MCR

- **use_v8struct**: (a boolean, nipype default value: True)
  
  Generate SPMS and higher compatible jobs

Outputs:

- **warped_files**: (a list of items which are a pathlike object or string representing an existing file)

References:

None

77.2.4 DARTEL

Link to code

Use spm DARTEL to create a template and flow fields

http://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf#page=185

Examples

```python
>>> import nipype.interfaces.spm as spm

>>> dartel = spm.DARTEL()
```

(continues on next page)
>>> dartel.inputs.image_files = [['rc1s1.nii', 'rc1s2.nii'], ['rc2s1.nii', 'rc2s2.nii']]
>>> dartel.run()  # doctest: +SKIP

## Inputs:

- **image_files**: A list of files to be segmented
  - List of items which are a list of items which are a pathlike object or string representing an existing file

## Optional Inputs:

- **template_prefix**: Prefix for template
- **regularization_form**: Form of regularization energy term
  - Linear or Membrane or Bending
- **iteration_parameters**: List of tuples for each iteration
  - Inner iterations
  - Regularization parameters
  - Time points for deformation model
  - Smoothing parameter
  - From 3 to 12 items which are a tuple of the form: (1 <= a long integer <= 10, a tuple of the form: (a float, a float, a float), 1 or 2 or 4 or 8 or 16 or 32 or 64 or 128 or 256 or 512, 0 or 0.5 or 1 or 2 or 4 or 8 or 16 or 32)

## Optimization Parameters:

- **optimization_parameters**: Optimization settings a tuple
  - LM regularization
  - Cycles of multigrid solver
  - Relaxation iterations
  - A tuple of the form: (a float, 1 <= a long integer <= 8, 1 <= a long integer <= 8)

## MATLAB Command:

- **matlab_cmd**: MATLAB command to use

## Paths:

- **paths**: Paths to add to matlabpath
  - A list of items which are a pathlike object or string representing a directory

## Output Flags:

- **mfile**: Run m-code using m-file
  - A boolean, default value: True
- **use_mcr**: Run m-code using SPM MCR
  - A boolean, default value: True
- **use_v8struct**: Generate SPm8 and higher compatible jobs
  - A boolean, default value: True

## Outputs:

- **final_template_file**: Final DARTEL template
  - A pathlike object or string representing an existing file
- **template_files**: Templates from different stages of iteration
  - A list of items which are a pathlike object or string representing an existing file
- **dartel_flow_fields**: DARTEL flow fields
  - A list of items which are a pathlike object or string representing an existing file

## References:

None
77.2.5 DARTELNorm2MNI

Link to code
Use spm DARTEL to normalize data to MNI space
http://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf#page=188

Examples

```python
>>> import nipype.interfaces.spm as spm
>>> nm = spm.DARTELNorm2MNI()
>>> nm.inputs.template_file = 'Template_6.nii'
>>> nm.inputs.flowfield_files = ['u_rc1s1_Template.nii', 'u_rc1s3_Template.nii']
>>> nm.inputs.apply_to_files = ['c1s1.nii', 'c1s3.nii']
>>> nm.inputs.modulate = True
>>> nm.run()  # doctest: +SKIP
```

Inputs:

- `template_file`: (a pathlike object or string representing an existing file) DARTEL template
- `flowfield_files`: (a list of items which are a pathlike object or string representing an existing file) DARTEL flow fields $u_{rc1}$
- `apply_to_files`: (a list of items which are a pathlike object or string representing an existing file) Files to apply the transform to

[Optional]
- `voxel_size`: (a tuple of the form: (a float, a float, a float)) Voxel sizes for output file
- `bounding_box`: (a tuple of the form: (a float, a float, a float, a float, a float, a float)) Voxel sizes for output file
- `modulate`: (a boolean) Modulate out images - no modulation preserves concentrations
- `fwhm`: (a list of from 3 to 3 items which are a float or a float) 3-list of fwhm for each dimension
- `matlab_cmd`: (a unicode string) matlab command to use
- `paths`: (a list of items which are a pathlike object or string representing a directory) Paths to add to matlabpath
- `mfile`: (a boolean, nipype default value: True) Run m-code using m-file
- `use_mcr`: (a boolean) Run m-code using SPM MCR
- `use_v8struct`: (a boolean, nipype default value: True) Generate SPM8 and higher compatible jobs

Outputs:

- `normalized_files`: (a list of items which are a pathlike object or string representing an existing file) Normalized files in MNI space
- `normalization_parameter_file`: (a pathlike object or string representing an existing file) Transform parameters to MNI space
References:
None

77.2.6 FieldMap

Link to code
Use the fieldmap toolbox from spm to calculate the voxel displacement map (VDM).
http://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf#page=173

To do
Deal with real/imag magnitude images and with the two phase files case.

Examples

```python
>>> from nipype.interfaces.spm import FieldMap
>>> fm = FieldMap()
>>> fm.inputs.phase_file = 'phase.nii'
>>> fm.inputs.magnitude_file = 'magnitude.nii'
>>> fm.inputs.echo_times = (5.19, 7.65)
>>> fm.inputs.blip_direction = 1
>>> fm.inputs.total_readout_time = 15.6
>>> fm.inputs.epi_file = 'epi.nii'
>>> fm.run()  # doctest: +SKIP
```

Inputs:

[Optional]
jobtype: ("calculatevdm" or "applyvdm", nipype default value: calculatevdm)
    one of: calculatevdm, applyvdm
maskbrain: (a boolean, nipype default value: True)
    masking or no masking of the brain
epifm: (a boolean, nipype default value: False)
    epi-based field map
jacobian_modulation: (a boolean, nipype default value: False)
    jacobian modulation
method: ("Mark3D" or "Mark2D" or "Huttonish", nipype default value: Mark3D)
    One of: Mark3D, Mark2D, Huttonish
unwarp_fwhm: (a long integer >= 0, nipype default value: 10)

(continues on next page)
gaussian smoothing kernel width
pad: (a long integer $\geq 0$, nipype default value: $0$)
padding kernel width
ws: (a boolean, nipype default value: True)
weighted smoothing
template: (a pathlike object or string representing an existing file)
template image for brain masking
mask_fwhm: (a long integer $\geq 0$, nipype default value: $5$)
gaussian smoothing kernel width
nerode: (a long integer $\geq 0$, nipype default value: $2$)
number of erosions
ndilate: (a long integer $\geq 0$, nipype default value: $4$)
number of erosions
thresh: (a float, nipype default value: $0.5$)
threshold used to create brain mask from segmented data
reg: (a float, nipype default value: $0.02$)
regularization value used in the segmentation
matchvdm: (a boolean, nipype default value: True)
match VDM to EPI
sessname: (a unicode string, nipype default value: _run-)
VDM filename extension
writeunwarped: (a boolean, nipype default value: False)
write unwarped EPI
anat_file: (a pathlike object or string representing an existing file)
anatomical image for comparison
matchanat: (a boolean, nipype default value: True)
match anatomical image to EPI
matlab_cmd: (a unicode string)
matlab command to use
paths: (a list of items which are a pathlike object or string representing a directory)
Paths to add to matlabpath
mfile: (a boolean, nipype default value: True)
Run m-code using m-file
use_mcr: (a boolean)
Run m-code using SPM MCR
use_v8struct: (a boolean, nipype default value: True)
Generate SPMS and higher compatible jobs

Outputs:

vdm: (a pathlike object or string representing an existing file)
voxel difference map

References:

None

77.2.7 NewSegment

Link to code
Use spm_preproc8 (New Segment) to separate structural images into different tissue classes. Supports multiple modalities.
NOTE: This interface currently supports single channel input only
http://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf#page=43
Examples

```python
>>> import nipype.interfaces.spm as spm

>>> seg = spm.NewSegment()
>>> seg.inputs.channel_files = 'structural.nii'
>>> seg.inputs.channel_info = (0.0001, 60, (True, True))
>>> seg.run()  # doctest: +SKIP
```

For VBM pre-processing [http://www.fil.ion.ucl.ac.uk/~john/misc/VBMclass10.pdf], TPM.nii should be replaced by /path/to/spm8/toolbox/Seg/TPM.nii

```python
>>> seg = NewSegment()
>>> seg.inputs.channel_files = 'structural.nii'
>>> tissue1 = (('TPM.nii', 1), 2, (True,True), (False, False))
>>> tissue2 = (('TPM.nii', 2), 2, (True,True), (False, False))
>>> tissue3 = (('TPM.nii', 3), 2, (True,False), (False, False))
>>> tissue4 = (('TPM.nii', 4), 2, (False,False), (False, False))
>>> tissue5 = (('TPM.nii', 5), 2, (False,False), (False, False))
>>> seg.inputs.tissues = [tissue1, tissue2, tissue3, tissue4, tissue5]
>>> seg.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
channel_files: (a list of items which are a pathlike object or string representing an existing file)

A list of files to be segmented

[Optional]
channel_info: (a tuple of the form: (a float, a float, a tuple of the form: (a boolean, a boolean)))

A tuple with the following fields:
- bias reguralisation (0-10)
- FWHM of Gaussian smoothness of bias
- which maps to save (Field, Corrected) - a tuple of two boolean values
tissues: (a list of items which are a tuple of the form: (a tuple of the form: (a pathlike object or string representing an existing file, an integer (int or long)), an integer (int or long), a tuple of the form: (a boolean, a boolean), a tuple of the form: (a boolean, a boolean)))

A list of tuples (one per tissue) with the following fields:
- tissue probability map (4D), 1-based index to frame
- number of gaussians
- which maps to save [Native, DARTEL] - a tuple of two boolean values
- which maps to save [Unmodulated, Modulated] - a tuple of two boolean values

affine_regularization: ('mni' or 'eastern' or 'subj' or 'none')

mni, eastern, subj, none

warping_regularization: (a list of from 5 to 5 items which are a float or a float)

Warping regularization parameter(s). Accepts float or list of floats (the latter is required by SPM12)
sampling_distance: (a float)

Sampling distance on data for parameter estimation

write_deformation_fields: (a list of from 2 to 2 items which are a boolean)

(continues on next page)
Which deformation fields to write: [Inverse, Forward]
matlab_cmd: (a unicode string)
  matlab command to use
paths: (a list of items which are a pathlike object or string
  representing a directory)
  Paths to add to matlabpath
mfile: (a boolean, nipype default value: True)
  Run m-code using m-file
use_mcr: (a boolean)
  Run m-code using SPM MCR
use_v8struct: (a boolean, nipype default value: True)
  Generate SPM8 and higher compatible jobs

Outputs:

native_class_images: (a list of items which are a list of items which
  are a pathlike object or string representing an existing file)
native space probability maps
dartel_input_images: (a list of items which are a list of items which
  are a pathlike object or string representing an existing file)
dartel imported class images
normalized_class_images: (a list of items which are a list of items
  which are a pathlike object or string representing an existing
  file)
normalized class images
modulated_class_images: (a list of items which are a list of items
  which are a pathlike object or string representing an existing
  file)
modulated=normalized class images
transformation_mat: (a list of items which are a pathlike object or
  string representing an existing file)
Normalization transformation
bias_corrected_images: (a list of items which are a pathlike object
  or string representing an existing file)
bias corrected images
bias_field_images: (a list of items which are a pathlike object or
  string representing an existing file)
bias field images
forward_deformation_field: (a list of items which are a pathlike
  object or string representing an existing file)
inverse_deformation_field: (a list of items which are a pathlike
  object or string representing an existing file)

References:
None

77.2.8 Normalize

Link to code
use spm_normalise for warping an image to a template
http://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf#page=203
Examples

```python
>>> import nipype.interfaces.spm as spm
>>> norm = spm.Normalize()
>>> norm.inputs.source = 'functional.nii'
>>> norm.run() # doctest: +SKIP

Inputs:

[Mandatory]

template: (a pathlike object or string representing an existing file)
  template file to normalize to
  mutually_exclusive: parameter_file

source: (a list of items which are a pathlike object or string
  representing an existing file)
  file to normalize to template
  mutually_exclusive: parameter_file

parameter_file: (a pathlike object or string representing a file)
  normalization parameter file*_sn.mat
  mutually_exclusive: source, template

[Optional]

jobtype: ('estwrite' or 'est' or 'write', nipype default value: estwrite)
  Estimate, Write or do both

apply_to_files: (a list of items which are a pathlike object or
  string representing an existing file or a list of items which are
  a pathlike object or string representing an existing file)
  files to apply transformation to

source_weight: (a pathlike object or string representing a file)
  name of weighting image for source

template_weight: (a pathlike object or string representing a file)
  name of weighting image for template

source_image_smoothing: (a float)
  source smoothing

template_image_smoothing: (a float)
  template smoothing

affine_regularization_type: ('mni' or 'size' or 'none')
  mni, size, none

DCT_period_cutoff: (a float)
  Cutoff of for DCT bases

nonlinear_iterations: (an integer (int or long))
  Number of iterations of nonlinear warping

nonlinear_regularization: (a float)
  the amount of the regularization for the nonlinear part of the
  normalization

write_preserve: (a boolean)
  True/False warped images are modulated

write_bounding_box: (a list of from 2 to 2 items which are a list of
  from 3 to 3 items which are a float)
  3x2-element list of lists

write_voxel_sizes: (a list of from 3 to 3 items which are a float)
  3-element list

write_interp: (0 <= a long integer <= 7)
  degree of b-spline used for interpolation

write_wrap: (a list of items which are an integer (int or long))
  Check if interpolation should wrap in [x,y,z] - list of bools

out_prefix: (a string, nipype default value: w)
```
normalized output prefix
matlab_cmd: (a unicode string)
matlab command to use
paths: (a list of items which are a pathlike object or string representing a directory)
Paths to add to matlabpath
mfile: (a boolean, nipype default value: True)
Run m-code using m-file
use_mcr: (a boolean)
Run m-code using SPM MCR
use_v8struct: (a boolean, nipype default value: True)
Generate SPM8 and higher compatible jobs

Outputs:

normalization_parameters: (a list of items which are a pathlike object or string representing an existing file)
MAT files containing the normalization parameters
normalized_source: (a list of items which are a pathlike object or string representing an existing file)
Normalized source files
normalized_files: (a list of items which are a pathlike object or string representing an existing file)
Normalized other files

References:

None

77.2.9 Normalize12

Link to code
uses SPM12’s new Normalise routine for warping an image to a template. Spatial normalisation is now done via the segmentation routine (which was known as New Segment in SPM8). Note that the normalisation in SPM12 is done towards a file containing multiple tissue probability maps, which was not the case in SPM8.

http://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf#page=49

Examples

```python
>>> import nipype.interfaces.spm as spm
>>> norm12 = spm.Normalize12()
>>> norm12.inputs.image_to_align = 'structural.nii'
>>> norm12.inputs.apply_to_files = 'functional.nii'
>>> norm12.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
image_to_align: (a pathlike object or string representing an existing file)
  file to estimate normalization parameters with
mutually_exclusive: deformation_file
deformation_file: (a pathlike object or string representing a file)
  file y_*.nii containing 3 deformation fields for the deformation in
  x, y and z dimension
  mutually_exclusive: image_to_align, tpm
[Optional]
apply_to_files: (a list of items which are a pathlike object or string representing an existing file or a list of items which are a pathlike object or string representing an existing file) files to apply transformation to

jobtype: ('estwrite' or 'est' or 'write', nipype default value: estwrite) Estimate, Write or do Both

bias_regularization: (0 or 1e-05 or 0.0001 or 0.001 or 0.01 or 0.1 or 1 or 10)
no(0) - extremely heavy (10)
bias_fwhm: (30 or 40 or 50 or 60 or 70 or 80 or 90 or 100 or 110 or 120 or 130 or 140 or 150 or 'Inf') FWHM of Gaussian smoothness of bias
tpm: (a pathlike object or string representing an existing file) template in form of tissue probablitiy maps to normalize to mutually_exclusive: deformation_file

affine_regularization_type: ('mni' or 'size' or 'none') mni, size, none
warping_regularization: (a list of from 5 to 5 items which are a float) controls balance between parameters and data

smoothness: (a float) value (in mm) to smooth the data before normalization
sampling_distance: (a float) Sampling distance on data for parameter estimation

write_bounding_box: (a list of from 2 to 2 items which are a list of from 3 to 3 items which are a float) 3x2-element list of lists representing the bounding box (in mm) to be written
write_voxel_sizes: (a list of from 3 to 3 items which are a float) 3-element list representing the voxel sizes (in mm) of the written normalised images

write_interp: (0 <= a long integer <= 7) degree of b-spline used for interpolation
out_prefix: (a string, nipype default value: w) Normalized output prefix
matlab_cmd: (a unicode string) matlab command to use
paths: (a list of items which are a pathlike object or string representing a directory) Paths to add to matlabpath

mfile: (a boolean, nipype default value: True) Run m-code using m-file
use_mcr: (a boolean) Run m-code using SPM MCR
use_v8struct: (a boolean, nipype default value: True) Generate SPMB and higher compatible jobs

Outputs:

defformation_field: (a list of items which are a pathlike object or string representing an existing file) NIfTI file containing 3 deformation fields for the deformation in x, y and z dimension
normalized_image: (a list of items which are a pathlike object or string representing a directory)
string representing an existing file)

Normalized file that needed to be aligned

normalized_files: (a list of items which are a pathlike object or
string representing an existing file)

Normalized other files

References:
None

77.2.10 Realign

Link to code
Use spm_realign for estimating within modality rigid body alignment
http://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf#page=25

Examples

```python
>>> import nipype.interfaces.spm as spm
>>> realign = spm.Realign()
>>> realign.inputs.in_files = 'functional.nii'
>>> realign.inputs.register_to_mean = True
>>> realign.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_files: (a list of items which are a pathlike object or string
representing an existing file or a list of items which are a
pathlike object or string representing an existing file)

list of filenames to realign

[Optional]

type: ('estwrite' or 'estimate' or 'write', nipype default value:
estwrite)

one of: estimate, write, estwrite

quality: (0.0 <= a floating point number <= 1.0)

0.1 = fast, 1.0 = precise

fwhm: (a floating point number >= 0.0)

gaussian smoothing kernel width

separation: (a floating point number >= 0.0)

sampling separation in mm

register_to_mean: (a boolean)

Indicate whether realignment is done to the mean image

weight_img: (a pathlike object or string representing an existing
file)

filename of weighting image

interp: (0 <= a long integer <= 7)

degree of b-spline used for interpolation

wrap: (a list of from 3 to 3 items which are an integer (int or
long))

Check if interpolation should wrap in [x,y,z]

write_which: (a list of items which are a value of class 'int',
nipype default value: [2, 1])

determines which images to reslice
write_interp: (0 <= a long integer <= 7)
    degree of b-spline used for interpolation
write_wrap: (a list of from 3 to 3 items which are an integer (int or long))
    Check if interpolation should wrap in [x,y,z]
write_mask: (a boolean)
    True/False mask output image
out_prefix: (a string, nipype default value: r)
    realigned output prefix
matlab_cmd: (a unicode string)
    matlab command to use
paths: (a list of items which are a pathlike object or string representing a directory)
    Paths to add to matlabpath
mfile: (a boolean, nipype default value: True)
    Run m-code using m-file
use_mcr: (a boolean)
    Run m-code using SPM MCR
use_v8struct: (a boolean, nipype default value: True)
    Generate SPM8 and higher compatible jobs

Outputs:

mean_image: (a pathlike object or string representing an existing file)
    Mean image file from the realignment
modified_in_files: (a list of items which are a list of items which are a pathlike object or string representing an existing file or a pathlike object or string representing an existing file)
    Copies of all files passed to in_files. Headers will have been modified to align all images with the first, or optionally to first do that, extract a mean image, and re-align to that mean image.
realigned_files: (a list of items which are a list of items which are a pathlike object or string representing an existing file or a pathlike object or string representing an existing file)
    If jobtype is write or estwrite, these will be the resliced files. Otherwise, they will be copies of in_files that have had their headers rewritten.
realignment_parameters: (a list of items which are a pathlike object or string representing an existing file)
    Estimated translation and rotation parameters

References:

None

77.2.11 RealignUnwarp

Link to code
Use spm_uw_estimate for estimating within subject registration and unwarping of time series. Function accepts only one single field map. If in_files is a list of files they will be treated as separate sessions but associated to the same fieldmap.
http://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf#page=31
Examples

```python
>>> import nipype.interfaces.spm as spm
>>> realignUnwarp = spm.RealignUnwarp()
>>> realignUnwarp.inputs.in_files = ['functional.nii', 'functional2.nii']
>>> realignUnwarp.inputs.phase_map = 'voxeldisplacemap.vdm'
>>> realignUnwarp.inputs.register_to_mean = True
>>> realignUnwarp.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
in_files: (a list of items which are a pathlike object or string representing an existing file or a list of items which are a pathlike object or string representing an existing file)
list of filenames to realign and unwarp

[Optional]
phase_map: (a pathlike object or string representing a file)
Voxel displacement map to use in unwarping. Unlike SPM standard behaviour, the same map will be used for all sessions
quality: (0.0 <= a floating point number <= 1.0)
0.1 = fast, 1.0 = precise
fwhm: (a floating point number >= 0.0)
gaussian smoothing kernel width
separation: (a floating point number >= 0.0)
sampling separation in mm
register_to_mean: (a boolean)
Indicate whether realignment is done to the mean image
weight_img: (a pathlike object or string representing an existing file)
filename of weighting image
interp: (0 <= a long integer <= 7)
degree of b-spline used for interpolation
wrap: (a list of from 3 to 3 items which are an integer (int or long))
Check if interpolation should wrap in [x,y,z]
est_basis_func: (a list of from 2 to 2 items which are an integer (int or long))
Number of basis functions to use for each dimension
est_reg_order: (0 <= a long integer <= 3)
This parameter determines how to balance the compromise between likelihood maximization and smoothness maximization of the estimated field.
est_reg_factor: (a list of items which are a value of class 'int', nipype default value: [100000])
Regularisation factor. Default: 100000 (medium).
est_jacobian_deformations: (a boolean)
Jacobian deformations. In theory a good idea to include them, in practice a bad idea. Default: No.
est_first_order_effects: (a list of from 1 to 6 items which are an integer (int or long))
First order effects should only depend on pitch and roll, i.e. [4 5]
est_second_order_effects: (a list of from 1 to 6 items which are an integer (int or long))
List of second order terms to model second derivatives of.
est_unwarp_fwhm: (a floating point number >= 0.0)
gaussian smoothing kernel width for unwarp

(continues on next page)
est_re_est_mov_par: (a boolean)
   Re-estimate movement parameters at each unwarping iteration.
est_num_of_iterations: (a list of items which are a value of class
   'int', nipype default value: [5])
   Number of iterations.
est_taylor_expansion_point: (a string, nipype default value: Average)
   Point in position space to perform Taylor-expansion around.
reslice_which: (a list of items which are a value of class 'int',
   nipype default value: [2, 1])
   determines which images to reslice
reslice_interp: (0 <= a long integer <= 7)
   degree of b-spline used for interpolation
reslice_wrap: (a list of from 3 to 3 items which are an integer (int
or long))
   Check if interpolation should wrap in [x,y,z]
reslice_mask: (a boolean)
   True/False mask output image
out_prefix: (a string, nipype default value: u)
   realigned and unwarped output prefix
matlab_cmd: (a unicode string)
   matlab command to use
paths: (a list of items which are a pathlike object or string
   representing a directory)
   Paths to add to matlabpath
mfile: (a boolean, nipype default value: True)
   Run m-code using m-file
use_mcr: (a boolean)
   Run m-code using SPM MCR
use_v8struct: (a boolean, nipype default value: True)
   Generate SPMS and higher compatible jobs

Outputs:

mean_image: (a pathlike object or string representing an existing
file)
   Mean image file from the realignment & unwarping
modified_in_files: (a list of items which are a list of items which
   are a pathlike object or string representing an existing file or a
   pathlike object or string representing an existing file)
   Copies of all files passed to in_files. Headers will have been
   modified to align all images with the first, or optionally to first
   do that, extract a mean image, and re-align to that mean image.
realigned_unwarped_files: (a list of items which are a list of items
   which are a pathlike object or string representing an existing
   file or a pathlike object or string representing an existing file)
   Realigned and unwarped files written to disc.
realignment_parameters: (a list of items which are a pathlike object
   or string representing an existing file)
   Estimated translation and rotation parameters

References:

None

77.2.12 Segment

Link to code
use spm_segment to separate structural images into different tissue classes.
http://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf#page=209

Examples

```python
>>> import nipype.interfaces.spm as spm
>>> seg = spm.Segment()
>>> seg.inputs.data = 'structural.nii'
>>> seg.run()  # doctest: +SKIP
```

**Inputs:**

**[Mandatory]**

data: (a list of items which are a pathlike object or string representing an existing file)

one scan per subject

**[Optional]**

gm_output_type: (a list of from 3 to 3 items which are a boolean)

Options to produce grey matter images: c1*.img, wc1*.img and mwc1*.img.

- None: [False, False, False],
- Native Space: [False, False, True],
- Unmodulated Normalised: [False, True, False],
- Modulated Normalised: [True, False, False],
- Native + Unmodulated Normalised: [False, True, True],
- Native + Modulated Normalised: [True, False, True],
- Native + Modulated + Unmodulated: [True, True, True],
- Modulated + Unmodulated Normalised: [True, True, False]

wm_output_type: (a list of from 3 to 3 items which are a boolean)

Options to produce white matter images: c2*.img, wc2*.img and mwc2*.img.

- None: [False, False, False],
- Native Space: [False, False, True],
- Unmodulated Normalised: [False, True, False],
- Modulated Normalised: [True, False, False],
- Native + Unmodulated Normalised: [False, True, True],
- Native + Modulated Normalised: [True, False, True],
- Native + Modulated + Unmodulated: [True, True, True],
- Modulated + Unmodulated Normalised: [True, True, False]

csf_output_type: (a list of from 3 to 3 items which are a boolean)

Options to produce CSF images: c3*.img, wc3*.img and mwc3*.img.

- None: [False, False, False],
- Native Space: [False, False, True],
- Unmodulated Normalised: [False, True, False],
- Modulated Normalised: [True, False, False],
- Native + Unmodulated Normalised: [False, True, True],
- Native + Modulated Normalised: [True, False, True],
- Native + Modulated + Unmodulated: [True, True, True],
- Modulated + Unmodulated Normalised: [True, True, False]

save_bias_corrected: (a boolean)

True/False produce a bias corrected image

clean_masks: ('no' or 'light' or 'thorough')

clean using estimated brain mask ('no', 'light', 'thorough')

tissue_prob_maps: (a list of items which are a pathlike object or string representing an existing file)

- list of gray, white & csf prob. (opt.)

gaussians_per_class: (a list of items which are an integer (int or (continues on next page)
long)
num Gaussians capture intensity distribution
affine_regularization: ('mni' or 'eastern' or 'subj' or 'none' or '')
    Possible options: "mni", "eastern", "subj", "none" (no
    reguralisation), "" (no affine registration)
warping_regularization: (a float)
    Controls balance between parameters and data
warp_frequency_cutoff: (a float)
    Cutoff of DCT bases
bias_regularization: (0 or 1e-05 or 0.0001 or 0.001 or 0.01 or 0.1 or 1 or 10)
    no(0) - extremely heavy (10)
bias_fwhm: (30 or 40 or 50 or 60 or 70 or 80 or 90 or 100 or 110 or 120 or 130 or 'Inf')
    FWHM of Gaussian smoothness of bias
sampling_distance: (a float)
    Sampling distance on data for parameter estimation
mask_image: (a pathlike object or string representing an existing
    file)
    Binary image to restrict parameter estimation
matlab_cmd: (a unicode string)
    matlab command to use
paths: (a list of items which are a pathlike object or string
    representing a directory)
    Paths to add to matlabpath
mfile: (a boolean, nipype default value: True)
    Run m-code using m-file
use_mcr: (a boolean)
    Run m-code using SPM MCR
use_v8struct: (a boolean, nipype default value: True)
    Generate SPMS and higher compatible jobs

Outputs:

native_gm_image: (a pathlike object or string representing a file)
    native space grey probability map
normalized_gm_image: (a pathlike object or string representing a
    file)
    normalized grey probability map
modulated_gm_image: (a pathlike object or string representing a file)
    modulated, normalized grey probability map
native_wm_image: (a pathlike object or string representing a file)
    native space white probability map
normalized_wm_image: (a pathlike object or string representing a
    file)
    normalized white probability map
modulated_wm_image: (a pathlike object or string representing a file)
    modulated, normalized white probability map
native_csf_image: (a pathlike object or string representing a file)
    native space csf probability map
normalized_csf_image: (a pathlike object or string representing a
    file)
    normalized csf probability map
modulated_csf_image: (a pathlike object or string representing a
    file)
    modulated, normalized csf probability map
modulated_input_image: (a pathlike object or string representing a
References:

None

77.2.13 SliceTiming

Link to code
Use spm to perform slice timing correction.
http://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf#page=19

Examples

```python
>>> from nipype.interfaces.spm import SliceTiming
>>> st = SliceTiming()
>>> st.inputs.in_files = 'functional.nii'
>>> st.inputs.num_slices = 32
>>> st.inputs.time_repetition = 6.0
>>> st.inputs.time_acquisition = 6. - 6./32.
>>> st.inputs.slice_order = list(range(32,0,-1))
>>> st.inputs.ref_slice = 1
>>> st.run() # doctest: +SKIP
```

Inputs:

[Optional]
out_prefix: (a string, nipype default value: a)
slicetimed output prefix
matlab_cmd: (a unicode string)
    matlab command to use
paths: (a list of items which are a pathlike object or string representing a directory)
    Paths to add to matlabpath
mfile: (a boolean, nipype default value: True)
    Run m-code using m-file
use_mcr: (a boolean)
    Run m-code using SPM MCR
use_v8struct: (a boolean, nipype default value: True)
    Generate SPM8 and higher compatible jobs

Outputs:

timecorrected_files: (a list of items which are a list of items which are a pathlike object or string representing an existing file or a pathlike object or string representing an existing file)
    slice time corrected files

References:

None

77.2.14 Smooth

Link to code
Use spm_smooth for 3D Gaussian smoothing of image volumes.
http://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf#page=55

Examples

>>> import nipype.interfaces.spm as spm
>>> smooth = spm.Smooth()
>>> smooth.inputs.in_files = 'functional.nii'
>>> smooth.inputs.fwhm = [4, 4, 4]
>>> smooth.run() # doctest: +SKIP

Inputs:

[Mandatory]
in_files: (a list of items which are a pathlike object or string representing an existing file)
    list of files to smooth

[Optional]
fwhm: (a list of from 3 to 3 items which are a float or a float)
    3-list of fwhm for each dimension
data_type: (an integer (int or long))
    Data type of the output images
implicit_masking: (a boolean)
    A mask implied by a particularvoxel value
out_prefix: (a string, nipype default value: s)
    smoothed output prefix
matlab_cmd: (a unicode string)
    matlab command to use
paths: (a list of items which are a pathlike object or string representing a directory)
   Paths to add to matlabpath
mfile: (a boolean, nipype default value: True)
   Run m-code using m-file
use_mcr: (a boolean)
   Run m-code using SPM MCR
use_v8struct: (a boolean, nipype default value: True)
   Generate SPM8 and higher compatible jobs

Outputs:

smoothed_files: (a list of items which are a pathlike object or string representing an existing file)
  smoothed files

References:
None

77.2.15 VBMSegment

Link to code
Use VBM8 toolbox to separate structural images into different tissue classes.

Example

```python
>>> import nipype.interfaces.spm as spm
>>> seg = spm.VBMSegment()
>>> seg.inputs.tissues = 'TPM.nii'
>>> seg.inputs.dartel_template = 'Template_1_IXI550_MNI152.nii'
>>> seg.inputs.bias_corrected_native = True
>>> seg.inputs.gm_native = True
>>> seg.inputs.wm_native = True
>>> seg.inputs.csf_native = True
>>> seg.inputs.pve_label_native = True
>>> seg.inputs.deformation_field = (True, False)
>>> seg.run() # doctest: +SKIP
```

Inputs:

[Optional]
tissues: (a pathlike object or string representing an existing file)
   tissue probability map
gaussians_per_class: (a tuple of the form: (an integer (int or long),
   an integer (int or long), an integer (int or long), an integer (int or long),
   an integer (int or long), an integer (int or long), an integer (int or long)),
   nipype default value: (2, 2, 2, 3, 4, 2))
   number of gaussians for each tissue class
bias_regularization: (0 or 1e-05 or 0.0001 or 0.001 or 0.01 or 0.1 or
   1 or 10, nipype default value: 0.0001)
no(0) - extremely heavy (10)
bias_fwhm: (30 or 40 or 50 or 60 or 70 or 80 or 90 or 100 or 110 or 120 or 130 or 'Inf', nipype default value: 60)
FWHM of Gaussian smoothness of bias
sampling_distance: (a float, nipype default value: 3)
Sampling distance on data for parameter estimation
warping_regularization: (a float, nipype default value: 4)
Controls balance between parameters and data
spatial_normalization: ('high' or 'low', nipype default value: high)
dartel_template: (a pathlike object or string representing an existing file)
use_sanlm_denoising_filter: (0 <= a long integer <= 2, nipype default value: 2)
  0=No denoising, 1=denoising, 2=denoising multi-threaded
mrf_weighting: (a float, nipype default value: 0.15)
cleanup_partitions: (an integer (int or long), nipype default value: 1)
  0=None, 1=light, 2=thorough
display_results: (a boolean, nipype default value: True)
gm_native: (a boolean, nipype default value: False)
gm_normalized: (a boolean, nipype default value: False)
gm_modulated_normalized: (0 <= a long integer <= 2, nipype default value: 2)
  0=None, 1=affine+non-linear (SPM8 default), 2=non-linear only
gm_dartel: (0 <= a long integer <= 2, nipype default value: 0)
  0=None, 1=rigid (SPM8 default), 2=affine
wm_native: (a boolean, nipype default value: False)
wm_normalized: (a boolean, nipype default value: False)
wmpmodulated_normalized: (0 <= a long integer <= 2, nipype default value: 2)
  0=None, 1=affine+non-linear (SPM8 default), 2=non-linear only
wm_dartel: (0 <= a long integer <= 2, nipype default value: 0)
  0=None, 1=rigid (SPM8 default), 2=affine
csf_native: (a boolean, nipype default value: False)
csf_normalized: (a boolean, nipype default value: False)
csf_modulated_normalized: (0 <= a long integer <= 2, nipype default value: 2)
  0=None, 1=affine+non-linear (SPM8 default), 2=non-linear only
csf_dartel: (0 <= a long integer <= 2, nipype default value: 0)
  0=None, 1=rigid (SPM8 default), 2=affine
bias_corrected_native: (a boolean, nipype default value: False)
bias_corrected_normalized: (a boolean, nipype default value: True)
bias_corrected_affine: (a boolean, nipype default value: False)
pve_label_native: (a boolean, nipype default value: False)
pve_label_normalized: (a boolean, nipype default value: False)
pve_label_dartel: (0 <= a long integer <= 2, nipype default value: 0)
  0=None, 1=rigid (SPM8 default), 2=affine
jacobian_determinant: (a boolean, nipype default value: False)
deformation_field: (a tuple of the form: (a boolean, a boolean), nipype default value: (0, 0))
  forward and inverse field
matlab_cmd: (a unicode string)
  matlab command to use
paths: (a list of items which are a pathlike object or string representing a directory)
Paths to add to matlabpath
mfile: (a boolean, nipype default value: True)
Run m-code using m-file
use_mcr: (a boolean)
  Run m-code using SPM MCR
use_v8struct: (a boolean, nipype default value: True)
  Generate SPMS and higher compatible jobs

Outputs:

native_class_images: (a list of items which are a list of items which are a pathlike object or string representing an existing file)
  native space probability maps
dartel_input_images: (a list of items which are a list of items which are a pathlike object or string representing an existing file)
  dartel imported class images
normalized_class_images: (a list of items which are a list of items which are a pathlike object or string representing an existing file)
  normalized class images
modulated_class_images: (a list of items which are a list of items which are a pathlike object or string representing an existing file)
  modulated+normalized class images
transformation_mat: (a list of items which are a pathlike object or string representing an existing file)
  Normalization transformation
bias_corrected_images: (a list of items which are a pathlike object or string representing an existing file)
  bias corrected images
normalized_bias_corrected_images: (a list of items which are a pathlike object or string representing an existing file)
  bias corrected images
pve_label_native_images: (a list of items which are a pathlike object or string representing an existing file)
pve_label_normalized_images: (a list of items which are a pathlike object or string representing an existing file)
pve_label_registered_images: (a list of items which are a pathlike object or string representing an existing file)
forward_deformation_field: (a list of items which are a pathlike object or string representing an existing file)
inverse_deformation_field: (a list of items which are a pathlike object or string representing an existing file)
jacobian_determinant_images: (a list of items which are a pathlike object or string representing an existing file)

References:
None

77.3 interfaces.spm.utils

77.3.1 Analyze2nii

Link to code
Inputs:
[Mandatory]
analyze_file: (a pathlike object or string representing an existing file)

[Optional]
matlab_cmd: (a unicode string)
  matlab command to use
paths: (a list of items which are a pathlike object or string representing a directory)
  Paths to add to matlabpath
mfile: (a boolean, nipype default value: True)
  Run m-code using m-file
use_mcr: (a boolean)
  Run m-code using SPM MCR
use_v8struct: (a boolean, nipype default value: True)
  Generate SPM8 and higher compatible jobs

Outputs:
nifti_file: (a pathlike object or string representing an existing file)
matlab_cmd: (a unicode string)
  matlab command to use
paths: (a list of items which are a pathlike object or string representing a directory)
  Paths to add to matlabpath
mfile: (a boolean, nipype default value: True)
  Run m-code using m-file
use_mcr: (a boolean)
  Run m-code using SPM MCR
use_v8struct: (a boolean, nipype default value: True)
  Generate SPM8 and higher compatible jobs

References:
None

77.3.2 ApplyInverseDeformation

Link to code
Uses spm to apply inverse deformation stored in a .mat file or a deformation field to a given file

Examples

```python
>>> import nipype.interfaces.spm.utils as spmu
>>> inv = spmu.ApplyInverseDeformation()
>>> inv.inputs.in_files = 'functional.nii'
>>> inv.inputs.deformation = 'struct_to_func.mat'
>>> inv.inputs.target = 'structural.nii'
>>> inv.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
in_files: (a list of items which are a pathlike object or string representing an existing file)
  Files on which deformation is applied
[Optional]
target: (a pathlike object or string representing an existing file)
   File defining target space
deformation: (a pathlike object or string representing an existing file)
   SN SPM deformation file
   mutually_exclusive: deformation_field
deformation_field: (a pathlike object or string representing an existing file)
   SN SPM deformation file
   mutually_exclusive: deformation
interpolation: (0 <= a long integer <= 7)
   degree of b-spline used for interpolation
bounding_box: (a list of from 6 to 6 items which are a float)
   6-element list (opt)
voxel_sizes: (a list of from 3 to 3 items which are a float)
   3-element list (opt)
matlab_cmd: (a unicode string)
   matlab command to use
paths: (a list of items which are a pathlike object or string representing a directory)
   Paths to add to matlabpath
mfile: (a boolean, nipype default value: True)
   Run m-code using m-file
use_mcr: (a boolean)
   Run m-code using SPM MCR
use_v8struct: (a boolean, nipype default value: True)
   Generate SPM8 and higher compatible jobs

Outputs:

out_files: (a list of items which are a pathlike object or string
   representing an existing file)
   Transformed files

References:

None

77.3.3 ApplyTransform

Uses SPM to apply transform stored in a .mat file to given file

Examples

```python
>>> import nipype.interfaces.spm.utils as spmu
>>> applymat = spmu.ApplyTransform()
>>> applymat.inputs.in_file = 'functional.nii'
>>> applymat.inputs.mat = 'func_to_struct.mat'
>>> applymat.run()  # doctest: +SKIP
```
[Mandatory]
in_file: (a pathlike object or string representing an existing file)
    file to apply transform to, (only updates header)
mat: (a pathlike object or string representing an existing file)
    file holding transform to apply

[Optional]
out_file: (a pathlike object or string representing a file)
    output file name for transformed data
matlab_cmd: (a unicode string)
    matlab command to use
paths: (a list of items which are a pathlike object or string
        representing a directory)
    Paths to add to matlabpath
mfile: (a boolean, nipype default value: True)
    Run m-code using m-file
use_mcr: (a boolean)
    Run m-code using SPM MCR
use_v8struct: (a boolean, nipype default value: True)
    Generate SPM8 and higher compatible jobs

Outputs:

out_file: (a pathlike object or string representing an existing file)
    Transformed image file

References:

None

77.3.4 CalcCoregAffine

Link to code

Uses SPM (spm_coreg) to calculate the transform mapping moving to target. Saves Transform in mat (matlab binary file) Also saves inverse transform

Examples

>>> import nipype.interfaces.spm.utils as spmu
>>> coreg = spmu.CalcCoregAffine(matlab_cmd='matlab-spm8')
>>> coreg.inputs.target = 'structural.nii'
>>> coreg.inputs.moving = 'functional.nii'
>>> coreg.inputs.mat = 'func_to_struct.mat'
>>> coreg.run()  # doctest: +SKIP

Note:

• the output file mat is saves as a matlab binary file
• calculating the transforms does NOT change either input image it does not move the moving image, only calculates the transform that can be used to move it

Inputs:

[Mandatory]
target: (a pathlike object or string representing an existing file)
    target for generating affine transform

(continues on next page)
moving: (a pathlike object or string representing an existing file)
volume transform can be applied to register with target

[Optional]
mat: (a pathlike object or string representing a file)
Filename used to store affine matrix
invmat: (a pathlike object or string representing a file)
Filename used to store inverse affine matrix
matlab_cmd: (a unicode string)
matlab command to use
paths: (a list of items which are a pathlike object or string representing a directory)
Paths to add to matlabpath
mfile: (a boolean, nipype default value: True)
Run m-code using m-file
use_mcr: (a boolean)
Run m-code using SPM MCR
use_v8struct: (a boolean, nipype default value: True)
Generate SPMS and higher compatible jobs

Outputs:

mat: (a pathlike object or string representing an existing file)
Matlab file holding transform
invmat: (a pathlike object or string representing a file)
Matlab file holding inverse transform

References:
None

77.3.5 DicomImport

Link to code
Uses spm to convert DICOM files to nii or img+hdr.

Examples

```python
>>> import nipype.interfaces.spm.utils as spmu
>>> di = spmu.DicomImport()
>>> di.inputs.in_files = ['functional_1.dcm', 'functional_2.dcm']
>>> di.run() # doctest: +SKIP
```

Inputs:

[Mandatory]
in_files: (a list of items which are a pathlike object or string representing an existing file)
dicom files to be converted

[Optional]
output_dir_struct: ('flat' or 'series' or 'patname' or 'patid_date'
or 'patid' or 'date_time', nipype default value: flat)
directory structure for the output.
output_dir: (a unicode string, nipype default value: ./converted_dicom)
output directory.

format: ('nii' or 'img', nipype default value: nii)

output format.

icedims: (a boolean, nipype default value: False)

If image sorting fails, one can try using the additional SIEMENS ICEDims information to create unique filenames. Use this only if there would be multiple volumes with exactly the same file names.

matlab_cmd: (a unicode string)

matlab command to use

paths: (a list of items which are a pathlike object or string representing a directory)

Paths to add to matlabpath

mfile: (a boolean, nipype default value: True)

Run m-code using m-file

use_mcr: (a boolean)

Run m-code using SPM MCR

use_v8struct: (a boolean, nipype default value: True)

Generate SPM8 and higher compatible jobs

Outs:

out_files: (a list of items which are a pathlike object or string representing an existing file)

converted files

References:

None

77.3.6 Reslice

Link to code

uses spm_reslice to resample in_file into space of space_defining

Inputs:

[Mandatory]

in_file: (a pathlike object or string representing an existing file)

file to apply transform to, (only updates header)

space_defining: (a pathlike object or string representing an existing file)

Volume defining space to slice in_file into

[Optional]

interp: (0 <= a long integer <= 7, nipype default value: 0)

degree of b-spline used for interpolation0 is nearest neighbor (default)

out_file: (a pathlike object or string representing a file)

Optional file to save resliced volume

matlab_cmd: (a unicode string)

matlab command to use

paths: (a list of items which are a pathlike object or string representing a directory)

Paths to add to matlabpath

mfile: (a boolean, nipype default value: True)

Run m-code using m-file

use_mcr: (a boolean)
Run m-code using SPM MCR
use_v8struct: (a boolean, nipype default value: True)
    Generate SPMS and higher compatible jobs

Outputs:
out_file: (a pathlike object or string representing an existing file)
    resliced volume

References:
None

77.3.7 ResliceToReference

Link to code
Uses spm to reslice a volume to a target image space or to a provided voxel size and bounding box

Examples

```python
>>> import nipype.interfaces.spm.utils as spmu
>>> r2ref = spmu.ResliceToReference()
>>> r2ref.inputs.in_files = 'functional.nii'
>>> r2ref.inputs.target = 'structural.nii'
>>> r2ref.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_files: (a list of items which are a pathlike object or string representing an existing file)
    Files on which deformation is applied

[Optional]
target: (a pathlike object or string representing an existing file)
    File defining target space
interpolation: (0 <= a long integer <= 7)
    degree of b-spline used for interpolation
bounding_box: (a list of from 6 to 6 items which are a float)
    6-element list (opt)
voxel_sizes: (a list of from 3 to 3 items which are a float)
    3-element list (opt)
matlab_cmd: (a unicode string)
    matlab command to use
paths: (a list of items which are a pathlike object or string representing a directory)
    Paths to add to matlabpath
mfile: (a boolean, nipype default value: True)
    Run m-code using m-file
use_mcr: (a boolean)
    Run m-code using SPM MCR
use_v8struct: (a boolean, nipype default value: True)
    Generate SPMS and higher compatible jobs

Outputs:
out_files: (a list of items which are a pathlike object or string representing an existing file)
Transformed files

References:
None
78.1 interfaces.utility

78.1.1 AssertEqual

Link to code

Inputs:

[Mandatory]
volume1: (a pathlike object or string representing an existing file)
volume2: (a pathlike object or string representing an existing file)

Outputs:

None

78.1.2 IdentityInterface

Link to code

Basic interface class generates identity mappings

Examples

```python
>>> from nipype.interfaces.utility import IdentityInterface
>>> ii = IdentityInterface(fields=['a', 'b'], mandatory_inputs=False)
>>> ii.inputs.a
<undefined>

>>> ii.inputs.a = 'foo'
>>> out = ii._outputs()
>>> out.a
<undefined>

>>> out = ii.run()
>>> out.outputs.a
'foo'
```
>>> ii2 = IdentityInterface(fields=['a', 'b'], mandatory_inputs=True)
>>> ii2.inputs.a = 'foo'
>>> out = ii2.run() # doctest: +SKIP
ValueError: IdentityInterface requires a value for input 'b' because it was
listed in 'fields' Interface IdentityInterface failed to run.

Inputs:

None

Outputs:

None

### 78.1.3 Merge

Link to code

Basic interface class to merge inputs into a single list

`Merge(1)` will merge a list of lists

**Examples**

```python
>>> from nipype.interfaces.utility import Merge
>>> mi = Merge(3)
>>> mi.inputs.in1 = 1
>>> mi.inputs.in2 = [2, 5]
>>> mi.inputs.in3 = 3
>>> out = mi.run()
>>> out.outputs.out
[1, 2, 5, 3]
```

```python
>>> merge = Merge(1)
>>> merge.inputs.in1 = [1, [2, 5], 3]
>>> out = merge.run()
>>> out.outputs.out
[1, [2, 5], 3]
```

```python
>>> merge = Merge(1)
>>> merge.inputs.in1 = [1, [2, 5], 3]
>>> merge.inputs.ravel_inputs = True
>>> out = merge.run()
>>> out.outputs.out
[1, 2, 5, 3]
```

```python
>>> merge = Merge(1)
>>> merge.inputs.in1 = [1, [2, 5], 3]
>>> merge.inputs.no_flatten = True
>>> out = merge.run()
>>> out.outputs.out
[[1, [2, 5], 3]]
```

Inputs:

[Optional]

- **axis**: ('vstack' or 'hstack', nipype default value: vstack)
  - direction in which to merge, hstack requires same number of elements
in each input

- no_flatten: (a boolean, nipype default value: False)
  append to outlist instead of extending in vstack mode
- ravel_inputs: (a boolean, nipype default value: False)
  ravel inputs when no_flatten is False

Outputs:

- out: (a list of items which are any value)
  Merged output

### 78.1.4 Rename

**Link to code**

Change the name of a file based on a mapped format string.

To use additional inputs that will be defined at run-time, the class constructor must be called with the format template, and the fields identified will become inputs to the interface.

Additionally, you may set the parse_string input, which will be run over the input filename with a regular expressions search, and will fill in additional input fields from matched groups. Fields set with inputs have precedence over fields filled in with the regexp match.

**Examples**

```python
>>> from nipype.interfaces.utility import Rename
>>> rename1 = Rename()
>>> rename1.inputs.in_file = os.path.join(datadir, "zstat1.nii.gz")  # datadir is a directory with exemplary files, defined in conftest.py
>>> rename1.inputs.format_string = "Faces-Scenes.nii.gz"
>>> res = rename1.run()  # doctest: +SKIP
>>> res.outputs.out_file  # doctest: +SKIP
'Faces-Scenes.nii.gz'  # doctest: +SKIP
```

```python
>>> rename2 = Rename(format_string="%(subject_id)s_func_run%(run)02d")
>>> rename2.inputs.in_file = os.path.join(datadir, "functional.nii")
>>> rename2.inputs.keep_ext = True
>>> rename2.inputs.subject_id = "subj_201"
>>> rename2.inputs.run = 2
>>> res = rename2.run()  # doctest: +SKIP
>>> res.outputs.out_file  # doctest: +SKIP
'subj_201_func_run02.nii'  # doctest: +SKIP
```

```python
>>> rename3 = Rename(format_string="%(subject_id)s_%(seq)s_run%(run)02d.nii")
>>> rename3.inputs.in_file = os.path.join(datadir, "func_epi_1_1.nii")
>>> rename3.inputs.parse_string = "func_(?P<seq>\w*)_.*"
>>> rename3.inputs.subject_id = "subj_201"
>>> rename3.inputs.run = 2
>>> res = rename3.run()  # doctest: +SKIP
>>> res.outputs.out_file  # doctest: +SKIP
'subj_201_epi_run02.nii'  # doctest: +SKIP
```

**Inputs:**

- [Mandatory]
  - in_file: (a pathlike object or string representing an existing file)
    file to rename
format_string: (a unicode string)
   Python formatting string for output template

[Optional]
keep_ext: (a boolean)
   Keep in_file extension, replace non-extension component of name
parse_string: (a unicode string)
   Python regexp parse string to define replacement inputs
use_fullpath: (a boolean, nipype default value: False)
   Use full path as input to regex parser

Outputs:

out_file: (a pathlike object or string representing an existing file)
   softlink to original file with new name

78.1.5 Select

Link to code
Basic interface class to select specific elements from a list

Examples

```python
>>> from nipype.interfaces.utility import Select
>>> sl = Select()
>>> _ = sl.inputs.trait_set(inlist=[1, 2, 3, 4, 5], index=[3])
>>> out = sl.run()
>>> out.outputs.out
~

>>> _ = sl.inputs.trait_set(inlist=[1, 2, 3, 4, 5], index=[3, 4])
>>> out = sl.run()
>>> out.outputs.out
[4, 5]
```

Inputs:

[Mandatory]

inlist: (a list of items which are any value)
   list of values to choose from
index: (a list of items which are an integer (int or long))
   0-based indices of values to choose

Outputs:

out: (a list of items which are any value)
   list of selected values

78.1.6 Split

Link to code
Basic interface class to split lists into multiple outputs
Examples

```python
>>> from nipype.interfaces.utility import Split
>>> sp = Split()
>>> _ = sp.inputs.trait_set(inlist=[1, 2, 3], splits=[2, 1])
>>> out = sp.run()
>>> out.outputs.out1
[1, 2]
```

Inputs:

[Mandatory]
- `inlist`: (a list of items which are any value)
  - list of values to split
- `splits`: (a list of items which are an integer (int or long))
  - Number of outputs in each split - should add to number of inputs

[Optional]
- `squeeze`: (a boolean, nipype default value: False)
  - unfold one-element splits removing the list

Outputs:

None

78.2 interfaces.utility.csv

78.2.1 CSVReader

Examples

```python
>>> reader = CSVReader()  # doctest: +SKIP
>>> reader.inputs.in_file = 'noHeader.csv'  # doctest: +SKIP
>>> out = reader.run()  # doctest: +SKIP
>>> out.outputs.column_0 == ['foo', 'bar', 'baz']  # doctest: +SKIP
True
>>> out.outputs.column_1 == ['hello', 'world', 'goodbye']  # doctest: +SKIP
True
>>> out.outputs.column_2 == ['300.1', '5', '0.3']  # doctest: +SKIP
True
```

```python
>>> reader = CSVReader()  # doctest: +SKIP
>>> reader.inputs.in_file = 'header.csv'  # doctest: +SKIP
>>> reader.inputs.header = True  # doctest: +SKIP
>>> out = reader.run()  # doctest: +SKIP
>>> out.outputs.files == ['foo', 'bar', 'baz']  # doctest: +SKIP
True
>>> out.outputs.labels == ['hello', 'world', 'goodbye']  # doctest: +SKIP
True
>>> out.outputs.erosion == ['300.1', '5', '0.3']  # doctest: +SKIP
True
```

Inputs:
### 78.3 interfaces.utility.wrappers

#### 78.3.1 Function

Link to code

Runs arbitrary function as an interface

**Examples**

```python
>>> func = 'def func(arg1, arg2=5): return arg1 + arg2'
>>> fi = Function(input_names=['arg1', 'arg2'], output_names=['out'])
>>> fi.inputs.function_str = func
>>> res = fi.run(arg1=1)
>>> res.outputs.out
```

**Inputs:**

- **function_str**: (a Unicode string)
  - code for function

**Outputs:**

- **None**
interface_vista

79.1 interfaces.vista.vista

79.1.1 Vnifti2Image

Link to code
Wraps the executable command vnifti2image.
Convert a nifti file into a vista file.

Example

```python
>>> vimage = Vnifti2Image()
>>> vimage.inputs.in_file = 'image.nii'
>>> vimage.cmdline
'vnifti2image -in image.nii -out image.v'
>>> vimage.run() # doctest: +SKIP
```

Inputs:

- **in_file**: (a pathlike object or string representing an existing file)
  
in file
  argument: ```-in %s```, position: 1

- **attributes**: (a pathlike object or string representing an existing file)
  
attribute file
  argument: ```-attr %s```, position: 2

- **out_file**: (a pathlike object or string representing a file)
  
output data file
  argument: ```-out %s```, position: -1

- **args**: (a unicode string)
  
Additional parameters to the command
  argument: ```%s```

- **environ**: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value...
value of class 'str', nipype default value: {})
Environonent variables

Outputs:

t_out_file: (a pathlike object or string representing an existing file)
    Output vista file

79.1.2 VtoMat

Link to code
Wraps the executable command `vтомат`.
Convert a nifti file into a vista file.

Example

```python
>>> vimage = VtoMat()
>>> vimage.inputs.in_file = 'image.v'
>>> vimage.cmdline
'veтомат -in image.v -out image.mat'
>>> vimage.run()     # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
in file
    argument: `'\~in %s'`, position: 1

[Optional]
out_file: (a pathlike object or string representing a file)
    output mat file
    argument: `'\~out %s'`, position: -1
args: (a unicode string)
    Additional parameters to the command
    argument: `'\%s'`
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})
    Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
    Output mat file
80.1 interfaces.workbench.base

80.1.1 WBCommand

Link to code
Base support for workbench commands.

Inputs:

<table>
<thead>
<tr>
<th>Optional</th>
<th>(a unicode string)</th>
</tr>
</thead>
<tbody>
<tr>
<td>args</td>
<td>Additional parameters to the command</td>
</tr>
<tr>
<td></td>
<td>argument: ''%s''</td>
</tr>
<tr>
<td>environ</td>
<td>(a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})</td>
</tr>
</tbody>
</table>

Environment variables

Outputs:

None

80.2 interfaces.workbench.cifti

80.2.1 CiftiSmooth

Link to code
Wraps the executable command `wb_command -cifti-smoothing`.

Smooth a CIFTI file

The input cifti file must have a brain models mapping on the chosen dimension, columns for .dtseries, and either for .dconn. By default, data in different structures is smoothed independently (i.e., “parcel constrained” smoothing), so volume structures that touch do not smooth across this boundary. Specify `merged_volume` to ignore these boundaries. Surface smoothing uses the `GEO_GAUSS_AREA` smoothing method.

The `*_corrected_areas` options are intended for when it is unavoidable to smooth on group average surfaces, it is only an approximate correction for the reduction of structure in a group average surface. It is better to smooth the data on individuals before averaging, when feasible.
The `fix_zeros_*` options will treat values of zero as lack of data, and not use that value when generating the smoothed values, but will fill zeros with extrapolated values. The ROI should have a brain models mapping along columns, exactly matching the mapping of the chosen direction in the input file. Data outside the ROI is ignored.

```python
>>> from nipype.interfaces.workbench import CiftiSmooth
>>> smooth = CiftiSmooth()
>>> smooth.inputs.in_file = 'sub-01_task-rest.dtseries.nii'
>>> smooth.inputs.sigma_surf = 4
>>> smooth.inputs.sigma_vol = 4
>>> smooth.inputs.direction = 'COLUMN'
>>> smooth.inputs.right_surf = 'sub-01.R.midthickness.32k_fs_LR.surf.gii'
>>> smooth.inputs.left_surf = 'sub-01.L.midthickness.32k_fs_LR.surf.gii'
>>> smooth.cmdline
'wb_command -cifti-smoothing sub-01_task-rest.dtseries.nii 4.0 4.0 COLUMN
smoothed_sub-01_task-rest.dtseries.nii -left-surface sub-01.L.midthickness.32k_fs_LR.surf.gii -right-surface sub-01.R.midthickness.32k_fs_LR.surf.gii'
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
The input CIFTI file
  argument: '%s', position: 0
sigma_surf: (a float)
  the sigma for the gaussian surface smoothing kernel, in mm
  argument: '%s', position: 1
sigma_vol: (a float)
  the sigma for the gaussian volume smoothing kernel, in mm
  argument: '%s', position: 2
direction: ('ROW' or 'COLUMN')
  which dimension to smooth along, ROW or COLUMN
  argument: '%s', position: 3
left_surf: (a pathlike object or string representing an existing file)
  Specify the left surface to use
  argument: '-left-surface %s', position: 5
right_surf: (a pathlike object or string representing an existing file)
  Specify the right surface to use
  argument: '-right-surface %s', position: 7

(Optional)
out_file: (a pathlike object or string representing a file)
The output CIFTI
  argument: '%s', position: 4
left_corrected_areas: (a pathlike object or string representing an existing file)
  vertex areas (as a metric) to use instead of computing them from the left surface.
  argument: '-left-corrected-areas %s', position: 6
right_corrected_areas: (a pathlike object or string representing an existing file)
  vertex areas (as a metric) to use instead of computing them from the right surface.
  argument: '-right-corrected-areas %s', position: 8
cerebellum_surf: (a pathlike object or string representing an existing file)
specify the cerebellum surface to use
argument: `-cerebellum-surface %s`, position: 9
cerebellum_corrected_areas: (a pathlike object or string representing an existing file)
vertex areas (as a metric) to use instead of computing them from the cerebellum surface
argument: `-cerebellum-corrected-areas %s`, position: 10
requires: cerebellum_surf
cifti_roi: (a pathlike object or string representing an existing file)
CIFTI file for ROI smoothing
argument: `-cifti-roi %s`, position: 11
fix_zeros_vol: (a boolean)
treat values of zero in the volume as missing data
argument: `-fix-zeros-volume`, position: 12
fix_zeros_surf: (a boolean)
treat values of zero on the surface as missing data
argument: `-fix-zeros-surface`, position: 13
merged_volume: (a boolean)
smooth across subcortical structure boundaries
argument: `-merged-volume`, position: 14
args: (a unicode string)
Additional parameters to the command
argument: `%s`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
output CIFTI file

80.3 interfaces.workbench.metric

80.3.1 MetricResample

Link to code
Wraps the executable command `wb_command -metric-resample`.
Resample a metric file to a different mesh
Resamples a metric file, given two spherical surfaces that are in register. If ADAP_BARY_AREA is used, exactly one of `-area-surfs` or `-area-metrics` must be specified.
The ADAP_BARY_AREA method is recommended for ordinary metric data, because it should use all data while downsampling, unlike BARYCENTRIC. The recommended areas option for most data is individual midthicknesses for individual data, and averaged vertex area metrics from individual midthicknesses for group average data.
The `-current-roi` option only masks the input, the output may be slightly dilated in comparison, consider using `-metric-mask` on the output when using `-current-roi`.
The `-largest` option results in nearest vertex behavior when used with BARYCENTRIC. When resampling a binary metric, consider thresholding at 0.5 after resampling rather than using `-largest`.

```python
>>> from nipype.interfaces.workbench import MetricResample
>>> metres = MetricResample()
>>> metres.inputs.in_file = 'sub-01_task-rest_bold_space-fsaverage5_L.func.gii'
```
>>> metres.inputs.method = 'ADAP_BARY_AREA'
>>> metres.inputs.current_sphere = 'fsaverage5_std_sphere.L.10k_fsavg_L.surf.gii'
>>> metres.inputs.new_sphere = 'fs_LR-deformed_to-fsaverage.L.sphere.32k_fs_LR.surf.gii'
>>> metres.inputs.area_metrics = True
>>> metres.inputs.current_area = 'fsaverage5.L.midthickness_va_avg.10k_fsavg_L.shape.gii'
>>> metres.inputs.new_area = 'fs_LR.L.midthickness_va_avg.32k_fs_LR.shape.gii'
>>> metres.cmdline
'wb_command -metric-resample sub-01_task-rest_bold_space-fsaverage5.L.func.gii sub-01_task-rest_bold_space-fsaverage5_std_sphere.L.10k_fsavg_L.surf.gii fs_LR-deformed_to-fsaverage.L.sphere.32k_fs_LR.surf.gii ADAP_BARY_AREA fs_LR-deformed_to-fsaverage.L.sphere.32k_fs_LR.surf.out -area-metrics fsaverage5.L.midthickness_va_avg.10k_fsavg_L.shape.gii fs_LR.L.midthickness_va_avg.32k_fs_LR.shape.gii'

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
The metric file to resample
  argument: ```%s```, position: 0
current_sphere: (a pathlike object or string representing an existing file)
  A sphere surface with the mesh that the metric is currently on
  argument: ```%s```, position: 1
new_sphere: (a pathlike object or string representing an existing file)
  A sphere surface that is in register with <current-sphere> and has
  the desired output mesh
  argument: ```%s```, position: 2
method: ('ADAP_BARY_AREA' or 'BARYCENTRIC')
  The method name - ADAP_BARY_AREA method is recommended for ordinary
  metric data, because it should use all data while downsampling,
  unlike BARYCENTRIC. If ADAP_BARY_AREA is used, exactly one of
  area_surfs or area_metrics must be specified
  argument: ```%s```, position: 3

[Optional]
out_file: (a pathlike object or string representing a file)
The output metric
  argument: ```%s```, position: 4
area_surfs: (a boolean)
  Specify surfaces to do vertex area correction based on
  argument: ```--area-surfs```, position: 5
  mutually_exclusive: area_metrics
area_metrics: (a boolean)
  Specify vertex area metrics to do area correction based on
  argument: ```--area-metrics```, position: 5
  mutually_exclusive: area_surfs
current_area: (a pathlike object or string representing an existing file)
  A relevant anatomical surface with <current-sphere> mesh OR a metric
  file with vertex areas for <current-sphere> mesh
  argument: ```%s```, position: 6
new_area: (a pathlike object or string representing an existing file)
  A relevant anatomical surface with <current-sphere> mesh OR a metric
  file with vertex areas for <current-sphere> mesh
  (continues on next page)
argument: ``%s``
roi_metric: (a pathlike object or string representing an existing file)
    Input roi on the current mesh used to exclude non-data vertices
    argument: ``'--current-roi %s'``, position: 8
valid_roi_out: (a boolean)
    Output the ROI of vertices that got data from valid source vertices
    argument: ``'--valid-roi-out'``, position: 9
largest: (a boolean)
    Use only the value of the vertex with the largest weight
    argument: ``'--largest'``, position: 10
args: (a unicode string)
    Additional parameters to the command
    argument: ``'%s'``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
    Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
    the output metric
roi_file: (a pathlike object or string representing a file)
    ROI of vertices that got data from valid source vertices
81.1 Bru2

Link to code
Wraps the executable command Bru2.
Uses bru2nii’s Bru2 to convert Bruker files

81.1.1 Examples

```python
>>> from nipype.interfaces.bru2nii import Bru2
>>> converter = Bru2()
>>> converter.inputs.input_dir = "brukerdir"
>>> converter.cmdline # doctest: +ELLIPSIS
'Bru2 -o .../data/brukerdir brukerdir'
```

Inputs:

[Mandatory]
input_dir: (a pathlike object or string representing an existing directory)
   Input Directory
   argument: `"%s"`, position: -1

[Optional]
actual_size: (a boolean)
   Keep actual size - otherwise x10 scale so animals match human.
   argument: `"-a"`
force_conversion: (a boolean)
   Force conversion of localizers images (multiple slice orientations).
   argument: `"-f"`
compress: (a boolean)
   gz compress images (".nii.gz")
   argument: `"-z"`
append_protocol_name: (a boolean)
   Append protocol name to output filename.
   argument: `"-p"`
output_filename: (a unicode string)
Output filename (".nii" will be appended, or ".nii.gz" if the "-z" compress option is selected)
argument: `'-o %s'`
args: (a unicode string)
  Additional parameters to the command
  argument: `''%s''`
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})
  Environment variables

Outputs:

nii_file: (a pathlike object or string representing an existing file)
82.1 C3d

Link to code
Wraps the executable command `c3d`.
Convert3d is a command-line tool for converting 3D (or 4D) images between common file formats. The tool also includes a growing list of commands for image manipulation, such as thresholding and resampling. The tool can also be used to obtain information about image files. More information on Convert3d can be found at: https://sourceforge.net/p/c3d/git/ci/master/tree/doc/c3d.md

82.1.1 Example

```python
>>> from nipype.interfaces.c3 import C3d
>>> c3 = C3d()
>>> c3.inputs.in_file = "T1.nii"
>>> c3.inputs.pix_type = "short"
>>> c3.inputs.out_file = "T1.img"
>>> c3.cmdline
'c3d T1.nii -type short -o T1.img'
>>> c3.inputs.is_4d = True
>>> c3.inputs.in_file = "epi.nii"
>>> c3.inputs.out_file = "epi.img"
>>> c3.cmdline
'c4d epi.nii -type short -o epi.img'
```

Inputs:

[Mandatory]
in_file: (a list of items which are a pathlike object or string representing a file)
Input file (wildcard and multiple are supported).
argument: ``%s``, position: 1

[Optional]
out_file: (a pathlike object or string representing a file)
Output file of last image on the stack.
argument: ``-o %s``, position: -1
mutually_exclusive: out_files
out_files: (a list of items which are a pathlike object or string representing a file)
Write all images on the convert3d stack as multiple files. Supports both list of output files or a pattern for the output filenames (using %d substitution).
argument: ``-oo %s``, position: -1
mutually_exclusive: out_file
pix_type: ('float' or 'char' or 'uchar' or 'short' or 'ushort' or 'int' or 'uint' or 'double')
Specifies the pixel type for the output image. By default, images are written in floating point (float) format.
argument: ``-type %s``
scale: (an integer (int or long) or a float)
Multiplies the intensity of each voxel in the last image on the stack by the given factor.
argument: ``--scale %s``
shift: (an integer (int or long) or a float)
Adds the given constant to every voxel.
argument: ``--shift %s``
interp: ('Linear' or 'NearestNeighbor' or 'Cubic' or 'Sinc' or 'Gaussian')
Specifies the interpolation used with -resample and other commands. Default is Linear.
argument: ``-interpolation %s``
resample: (a unicode string)
Resamples the image, keeping the bounding box the same, but changing the number of voxels in the image. The dimensions can be specified as a percentage, for example to double the number of voxels in each direction. The -interpolation flag affects how sampling is performed.
argument: ``-resample %s``
smooth: (a unicode string)
Applies Gaussian smoothing to the image. The parameter vector specifies the standard deviation of the Gaussian kernel.
argument: ``-smooth %s``
multicomp_split: (a boolean, nipype default value: False)
Enable reading of multi-component images.
argument: ``-mcr``, position: 0
is_4d: (a boolean, nipype default value: False)
Changes command to support 4D file operations (default is false).
args: (a unicode string)
Additional parameters to the command
argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {}) Environment variables

Outputs:

out_files: (a list of items which are a pathlike object or string representing a file)
82.2 C3dAffineTool

Link to code
Wraps the executable command `c3d_affine_tool`,
Converts fsl-style Affine registration into ANTS compatible itk format

82.2.1 Example

```python
>>> from nipype.interfaces.c3 import C3dAffineTool
>>> c3 = C3dAffineTool()
>>> c3.inputs.source_file = 'cmatrix.mat'
>>> c3.inputs.itk_transform = 'affine.txt'
>>> c3.inputs.fsl2ras = True
>>> c3.cmdline
'c3d_affine_tool -src cmatrix.mat -fsl2ras -oitk affine.txt'
```

Inputs:

[Optional]
- `reference_file`: (a pathlike object or string representing an existing file)
  - argument: `'-%s'`, position: 1
- `source_file`: (a pathlike object or string representing an existing file)
  - argument: `'-%s'`, position: 2
- `transform_file`: (a pathlike object or string representing an existing file)
  - argument: `'-%s'`, position: 3
- `itk_transform`: (a boolean or a pathlike object or string representing a file)
  - Export ITK transform.
  - argument: `'-%otk %s'`, position: 5
- `fsl2ras`: (a boolean)
  - argument: `'-%fs2ras'`, position: 4
- `args`: (a unicode string)
  - Additional parameters to the command
  - argument: `'-%s'`
- `environ`: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: `{}`)
  - Environment variables

Outputs:

- `itk_transform`: (a pathlike object or string representing an existing file)
83.1 Dcm2nii

Link to code
Wraps the executable command dcm2nii. Uses MRIcron’s dcm2nii to convert dicom files

83.1.1 Examples

```python
>>> from nipype.interfaces.dcm2nii import Dcm2nii
>>> converter = Dcm2nii()
>>> converter.inputs.source_names = ['functional_1.dcm', 'functional_2.dcm']
>>> converter.inputs.gzip_output = True
>>> converter.inputs.output_dir = '.
>>> converter.cmdline
'dcm2nii -a y -c y -b config.ini -v y -d y -e y -f y -i n -n y -o . -p y -x n -f n functional_1.dcm'
```

Inputs:

[Mandatory]
- source_names: (a list of items which are a pathlike object or string representing an existing file)
  - argument: ``%s``, position: -1
  - mutually_exclusive: source_dir
- source_dir: (a pathlike object or string representing an existing directory)
  - argument: ``%s``, position: -1
  - mutually_exclusive: source_names

[Optional]
- anonymize: (a boolean, nipype default value: True)
  - Remove identifying information
  - argument: ``-a``
- config_file: (a pathlike object or string representing an existing file)
  - Load settings from specified inifile

(continues on next page)
argument: `'-b %s'`
collapse_folders: (a boolean, nipype default value: True)
    Collapse input folders
    argument: `'-c'`
date_in_filename: (a boolean, nipype default value: True)
    Date in filename
    argument: `'-d'`
events_in_filename: (a boolean, nipype default value: True)
    Events (series/acq) in filename
    argument: `'-e'`
source_in_filename: (a boolean, nipype default value: False)
    Source filename
    argument: `'-f'`
gzip_output: (a boolean, nipype default value: False)
    Gzip output (.gz)
    argument: `'-g'`
id_in_filename: (a boolean, nipype default value: False)
    ID in filename
    argument: `'-i'`
nii_output: (a boolean, nipype default value: True)
    Save as .nii - if no, create .hdr/.img pair
    argument: `'-n'`
output_dir: (a pathlike object or string representing an existing
directory)
    Output dir - if unspecified, source directory is used
    argument: `'-o %s'`
protocol_in_filename: (a boolean, nipype default value: True)
    Protocol in filename
    argument: `'-p'`
reorient: (a boolean)
    Reorient image to nearest orthogonal
    argument: `'-r'`
spm_analyze: (a boolean)
    SPM2/Analyze not SPM5/NIfTI
    argument: `'-s'`
mutually_exclusive: nii_output
convert_all_pars: (a boolean, nipype default value: True)
    Convert every image in directory
    argument: `'-v'`
reorient_and_crop: (a boolean, nipype default value: False)
    Reorient and crop 3D images
    argument: `'-x'`
args: (a unicode string)
    Additional parameters to the command
    argument: `'%s'`
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {})
    Environment variables

Outputs:

converted_files: (a list of items which are a pathlike object or
    string representing an existing file)
reoriented_files: (a list of items which are a pathlike object or
    string representing an existing file)
reoriented_and_cropped_files: (a list of items which are a pathlike
    (continues on next page)
**83.2 Dcm2niix**

Link to code

Wraps the executable command `dcm2niix`. Uses Chris Rorden’s `dcm2niix` to convert dicom files

### 83.2.1 Examples

```python
>>> from nipype.interfaces.dcm2niix import Dcm2niix
>>> converter = Dcm2niix()
>>> converter.inputs.source_dir = 'dicomdir'
>>> converter.inputs.compression = 5
>>> converter.inputs.output_dir = 'ds005'
>>> converter.cmdline
'dcm2niix -b y -z y -5 -x n -t n -m n -o ds005 -s n -v n dicomdir'
>>> converter.run()  # doctest: +SKIP
```

In the example below, we note that the current version of `dcm2niix` converts any files in the directory containing the files in the list. We also do not support nested filenames with this option. Thus all files must have a common root directory.

```python
>>> converter = Dcm2niix()
>>> converter.inputs.source_names = ['functional_1.dcm', 'functional_2.dcm']
>>> converter.inputs.compression = 5
>>> converter.inputs.output_dir = 'ds005'
>>> converter.cmdline
'dcm2niix -b y -z y -5 -x n -t n -m n -o ds005 -s n -v n .'
>>> converter.run()  # doctest: +SKIP
```

**Inputs:**

[Mandatory]

- **source_names**: (a list of items which are a pathlike object or string representing an existing file)
  - A set of filenames to be converted. Note that the current version (1.0.20180328) of `dcm2niix` converts any files in the directory. To only convert specific files they should be in an isolated directory argument: ```%s```, position: -1
  - mutually_exclusive: source_dir

- **source_dir**: (a pathlike object or string representing an existing directory)
  - A directory containing dicom files to be converted
  - argument: ```%s```, position: -1
  - mutually_exclusive: source_names

[Optional]

- **out_filename**: (a unicode string)
  - Output filename template (%a=antenna (coil) number, %c=comments, %d=description, %e=echo number, %f=folder name, %i=ID of patient, %s=site number, %t=tag number)

(continues on next page)
%j=seriesInstanceUID, %k=studyInstanceUID, %m=manufacturer, %n=name of patient, %p=protocol, %s=series number, %t=time, %u=acquisition number, %v=vendor, %x=study ID; %z=sequence name

argument: ```-f %s```

output_dir: (a pathlike object or string representing an existing directory, nipype default value: .)

Output directory

argument: ```-o %s```

bids_format: (a boolean, nipype default value: True)

Create a BIDS sidecar file

argument: ```-b```

anon_bids: (a boolean)

Anonymize BIDS

argument: ```-ba```

requires: bids_format

compress: ('y' or 'i' or 'n' or '3', nipype default value: y)

Gzip compress images - [y=pigz, i=internal, n=no, 3=no, 3D]

argument: ```-z %s```

merge_imgs: (a boolean, nipype default value: False)

merge 2D slices from same series

argument: ```-m```

single_file: (a boolean, nipype default value: False)

Single file mode

argument: ```-s```

verbose: (a boolean, nipype default value: False)

Verbose output

argument: ```-v```

crop: (a boolean, nipype default value: False)

Crop 3D T1 acquisitions

argument: ```-x```

has_private: (a boolean, nipype default value: False)

Flag if text notes include private patient details

argument: ```-t```

compression: (1 or 2 or 3 or 4 or 5 or 6 or 7 or 8 or 9)

Gz compression level (1=fastest, 9=smallest)

argument: ```-%d```

comment: (a unicode string)

Comment stored as NIfTI aux_file

argument: ```-c %s```

ignore deriv: (a boolean)

Ignore derived, localizer and 2D images

argument: ```-i```

series_numbers: (a list of items which are a unicode string)

Selectively convert by series number - can be used up to 16 times

argument: ```-n %s...```

philips_float: (a boolean)

Philips precise float (not display) scaling

argument: ```-p```

args: (a unicode string)

Additional parameters to the command

argument: ```%s```

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})

Environment variables

Outputs:
converted_files: (a list of items which are a pathlike object or string representing an existing file)
bvecs: (a list of items which are a pathlike object or string representing an existing file)
bvals: (a list of items which are a pathlike object or string representing an existing file)
bids: (a list of items which are a pathlike object or string representing an existing file)
84.1 CopyMeta

Link to code
Copy meta data from one Nifti file to another. Useful for preserving meta data after some processing steps.

Inputs:

[Mandatory]
src_file: (a pathlike object or string representing an existing file)
dest_file: (a pathlike object or string representing an existing file)

[Optional]
include_classes: (a list of items which are any value)
  list of specific meta data classifications to include. If not specified include everything.
exclude_classes: (a list of items which are any value)
  list of meta data classifications to exclude

Outputs:
dest_file: (a pathlike object or string representing an existing file)

84.2 DcmStack

Link to code
Create one Nifti file from a set of DICOM files. Can optionally embed meta data.

84.2.1 Example

```python
>>> from nipype.interfaces.dcmstack import DcmStack
>>> stacker = DcmStack()
>>> stacker.inputs.dicom_files = 'path/to/series/'
>>> stacker.run() # doctest: +SKIP
```
>>> result.outputs.out_file # doctest: +SKIP
'/path/to/cwd/sequence.nii.gz'

Inputs:

[Mandatory]
dicom_files: (a list of items which are a pathlike object or string representing an existing file or a pathlike object or string representing an existing directory or a unicode string)

[Optional]
embed_meta: (a boolean)
   Embed DICOM meta data into result
exclude_regexes: (a list of items which are any value)
   Meta data to exclude, suplementing any default exclude filters
include_regexes: (a list of items which are any value)
   Meta data to include, overriding any exclude filters
force_read: (a boolean, nipype default value: True)
   Force reading files without DICM marker
out_format: (a unicode string)
   String which can be formatted with meta data to create the output filename(s)
out_ext: (a unicode string, nipype default value: .nii.gz)
   Determines output file type
out_path: (a pathlike object or string representing a directory)
   output path, current working directory if not set

Outputs:

out_file: (a pathlike object or string representing an existing file)

84.3 GroupAndStack

Link to code
Create (potentially) multiple Nifti files for a set of DICOM files.

Inputs:

[Mandatory]
dicom_files: (a list of items which are a pathlike object or string representing an existing file or a pathlike object or string representing an existing directory or a unicode string)

[Optional]
embed_meta: (a boolean)
   Embed DICOM meta data into result
exclude_regexes: (a list of items which are any value)
   Meta data to exclude, suplementing any default exclude filters
include_regexes: (a list of items which are any value)
   Meta data to include, overriding any exclude filters
force_read: (a boolean, nipype default value: True)
   Force reading files without DICM marker
out_format: (a unicode string)
   String which can be formatted with meta data to create the output filename(s)
out_ext: (a unicode string, nipype default value: .nii.gz)
   Determines output file type
(continues on next page)
84.4 LookupMeta

Link to code
Lookup meta data values from a Nifti with embedded meta data.

84.4.1 Example

```python
>>> from nipype.interfaces import dcmstack
>>> lookup = dcmstack.LookupMeta()
>>> lookup.inputs.in_file = 'functional.nii'
>>> lookup.inputs.meta_keys = {'RepetitionTime': 'TR',
                               'EchoTime': 'TE'}
>>> result = lookup.run() # doctest: +SKIP
>>> result.outputs.TR # doctest: +SKIP
9500.0
>>> result.outputs.TE # doctest: +SKIP
95.0
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
   The input Nifti file
meta_keys: (a list of items which are any value or a dictionary with
   keys which are any value and with values which are any value)
   List of meta data keys to lookup, or a dict where keys specify the
   meta data keys to lookup and the values specify the output names

Outputs:

None

84.5 MergeNifti

Link to code
Merge multiple Nifti files into one. Merges together meta data extensions as well.

Inputs:

[Mandatory]
in_files: (a list of items which are any value)
   List of Nifti files to merge

[Optional]
sort_order: (a unicode string or a list of items which are any value)
   One or more meta data keys to sort files by.
merge_dim: (an integer (int or long))
Dimension to merge along. If not specified, the last singular or non-existant dimension is used.

out_format: (a unicode string)
- String which can be formatted with meta data to create the output filename(s)

out_ext: (a unicode string, nipype default value: .nii.gz)
- Determines output file type

out_path: (a pathlike object or string representing a directory)
- output path, current working directory if not set

Outputs:

out_file: (a pathlike object or string representing an existing file)
- Merged Nifti file

84.6 NiftiGeneratorBase

Link to code
Base class for interfaces that produce Nifti files, potentially with embedded meta data.
Inputs:

None

Outputs:

None

84.7 SplitNifti

Link to code
Split one Nifti file into many along the specified dimension. Each result has an updated meta data extension as well.
Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
- Nifti file to split

[Optional]
split_dim: (an integer (int or long))
- Dimension to split along. If not specified, the last dimension is used.

out_format: (a unicode string)
- String which can be formatted with meta data to create the output filename(s)

out_ext: (a unicode string, nipype default value: .nii.gz)
- Determines output file type

out_path: (a pathlike object or string representing a directory)
- output path, current working directory if not set

Outputs:

out_list: (a list of items which are a pathlike object or string representing an existing file)
- Split Nifti files
84.8 make_key_func()

Link to code

84.9 sanitize_path_comp()

Link to code
85.1 SlicerCommandLine

Link to code
Wraps the executable command Slicer3.
Experimental Slicer wrapper. Work in progress.

Inputs:

```
[Optional]
module: (a unicode string)
    name of the Slicer command line module you want to use
args: (a unicode string)
    Additional parameters to the command
    argument: ``\$s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipype default value: {}))
```

Environment variables

Outputs:

None
86.1 Reorient

Link to code
Conform an image to a given orientation
Flips and reorder the image data array so that the axes match the directions indicated in orientation. The default RAS orientation corresponds to the first axis being ordered from left to right, the second axis from posterior to anterior, and the third axis from inferior to superior.
For oblique images, the original orientation is considered to be the closest plumb orientation.
No resampling is performed, and thus the output image is not de-obliqued or registered to any other image or template.
The effective transform is calculated from the original affine matrix to the reoriented affine matrix.

86.1.1 Examples
If an image is not reoriented, the original file is not modified

```python
>>> import numpy as np
>>> from nipype.interfaces.image import Reorient
>>> reorient = Reorient(orientation='LPS')
>>> reorient.inputs.in_file = 'segmentation0.nii.gz'
>>> res = reorient.run()
>>> res.outputs.out_file
'segmentation0.nii.gz'
```

```python
>>> print_affine(np.loadtxt(res.outputs.transform))
1.  0.  0.  0.
0.  1.  0.  0.
0.  0.  1.  0.
0.  0.  0.  1.
```

```python
>>> reorient.inputs.orientation = 'RAS'
>>> res = reorient.run()
>>> res.outputs.out_file  # doctest: +ELLIPSIS
'.../segmentation0_ras.nii.gz'
```
>>> print_affine(np.loadtxt(res.outputs.transform))
-1. 0. 0. 60.
0. -1. 0. 72.
0. 0. 1. 0.
0. 0. 0. 1.

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
Input image

[Optional]
orientation: ('RAS' or 'RAI' or 'RPS' or 'RPI' or 'LAS' or 'LAI' or
'LPS' or 'LPI' or 'RSA' or 'RSP' or 'RIA' or 'RIP' or 'LSA' or
'LSP' or 'LIA' or 'LIP' or 'ARS' or 'ARI' or 'ALS' or 'ALI' or
'PRS' or 'PRI' or 'PLS' or 'PLI' or 'ASR' or 'ASL' or 'AIR' or
'AIL' or 'PSR' or 'PSL' or 'PIR' or 'PLI' or 'SRA' or 'SRP' or
'SLA' or 'SLP' or 'IRA' or 'IRP' or 'ILA' or 'ILP' or 'SAR' or
'SAL' or 'SPR' or 'SPL' or 'IAR' or 'IAL' or 'IPR' or 'IPL',
nipype default value: RAS)
Target axis orientation

Outputs:

out_file: (a pathlike object or string representing an existing file)
Reoriented image
transform: (a pathlike object or string representing an existing file)
Affine transform from input orientation to output

86.2 Rescale

Link to code
Rescale an image
Rescales the non-zero portion of in_file to match the bounds of the non-zero portion of ref_file. Reference values in the input and reference images are defined by the percentile parameter, and the reference values in each image are identified and the remaining values are scaled accordingly. In the case of percentile == 0, the reference values are the maxima and minima of each image. If the invert parameter is set, the input file is inverted prior to rescaling.

86.2.1 Examples

To use a high-resolution T1w image as a registration target for a T2* image, it may be useful to invert the T1w image and rescale to the T2* range. Using the 1st and 99th percentiles may reduce the impact of outlier voxels.

```python
>>> from nipype.interfaces.image import Rescale
>>> invert_t1w = Rescale(invert=True)
>>> invert_t1w.inputs.in_file = 'structural.nii'
>>> invert_t1w.inputs.ref_file = 'functional.nii'
>>> invert_t1w.inputs.percentile = 1.
>>> res = invert_t1w.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
in_file: (a pathlike object or string representing an existing file)
Skull-stripped image to rescale

ref_file: (a pathlike object or string representing an existing file)
  Skull-stripped reference image

[Optional]
invert: (a boolean)
  Invert contrast of rescaled image
percentile: (0.0 <= a floating point number <= 50.0, nipype default value: 0.0)
  Percentile to use for reference to allow for outliers - 1 indicates the 1st and 99th percentiles in the input file will be mapped to the 99th and 1st percentiles in the reference; 0 indicates minima and maxima will be mapped

Outputs:

g.s: (a pathlike object or string representing an existing file)
  Rescaled image
87.1 BIDSDataGrabber

Link to code
BIDS datagrabber module that wraps around pybids to allow arbitrary querying of BIDS datasets.

87.1.1 Examples

By default, the BIDSDataGrabber fetches anatomical and functional images from a project, and makes BIDS entities (e.g. subject) available for filtering outputs.

```python
>>> bg = BIDSDataGrabber()
>>> bg.inputs.base_dir = 'ds005/'
>>> bg.inputs.subject = '01'
>>> results = bg.run()  # doctest: +SKIP
```

Dynamically created, user-defined output fields can also be defined to return different types of outputs from the same project. All outputs are filtered on common entities, which can be explicitly defined as infields.

```python
>>> bg = BIDSDataGrabber(infields=['subject'])
>>> bg.inputs.base_dir = 'ds005/'
>>> bg.inputs.subject = '01'
>>> bg.inputs.output_query['dwi'] = dict(datatype='dwi')
>>> results = bg.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
base_dir: (a pathlike object or string representing an existing directory)
   Path to BIDS Directory.
index_derivatives: (a boolean, nipype default value: False)
   Index derivatives/ sub-directory

[Optional]
output_query: (a dictionary with keys which are a unicode string and with values which are a dictionary with keys which are any value and with values which are any value)
Queries for out-field outputs
raise_on_empty: (a boolean, nipype default value: True)
Generate exception if list is empty for a given field
extra_derivatives: (a list of items which are a pathlike object or
string representing an existing directory)
Additional derivative directories to index

Outputs:
None

87.2 DataFinder

Link to code
Search for paths that match a given regular expression. Allows a less proscriptive approach to gathering input
files compared to DataGrabber. Will recursively search any subdirectories by default. This can be limited
with the min/max depth options. Matched paths are available in the output ‘out_paths’. Any named groups of
captured text from the regular expression are also available as outputs of the same name.

87.2.1 Examples

```python
>>> from nipype.interfaces.io import DataFinder
>>> df = DataFinder()
>>> df.inputs.root_paths = '.
>>> df.inputs.match_regex = '.+/(?P<series_dir>.+(qT1|ep2d_fid_T1).+)/(?P<basename>.+).nii.gz'
>>> result = df.run() # doctest: +SKIP
>>> result.outputs.out_paths # doctest: +SKIP
['./027-ep2d_fid_T1_Gd4/acquisition.nii.gz',
'./018-ep2d_fid_T1_Gd2/acquisition.nii.gz',
'./016-ep2d_fid_T1_Gd1/acquisition.nii.gz',
'./013-ep2d_fid_T1_pre/acquisition.nii.gz']
>>> result.outputs.series_dir # doctest: +SKIP
['027-ep2d_fid_T1_Gd4',
'018-ep2d_fid_T1_Gd2',
'016-ep2d_fid_T1_Gd1',
'013-ep2d_fid_T1_pre']
>>> result.outputs.basename # doctest: +SKIP
['acquisition',
'acquisition',
'acquisition',
'acquisition']
```

Inputs:

[Mandatory]
root_paths: (a list of items which are any value or a unicode string)

(Optional)
match_regex: (a unicode string, nipype default value: (.+))
Regular expression for matching paths.
ignore_regexes: (a list of items which are any value)
List of regular expressions, if any match the path it will be ignored.
max_depth: (an integer (int or long))
The maximum depth to search beneath the root_paths
min_depth: (an integer (int or long))
The minimum depth to search beneath the root paths
unpack_single: (a boolean, nipype default value: False)
Unpack single results from list

Outputs:
None

87.3 DataGrabber

Link to code
Generic datagrabber module that wraps around glob in an intelligent way for neuroimaging tasks to grab files

Attention: Doesn’t support directories currently

87.3.1 Examples

```python
>>> from nipype.interfaces.io import DataGrabber

Pick all files from current directory
>>> dg = DataGrabber()
>>> dg.inputs.template = '*'

Pick file foo/foo.nii from current directory
>>> dg.inputs.template = '%s/%s.dcm'
>>> dg.inputs.template_args['outfiles']=[['dicomdir','123456-1-1.dcm']]  

Same thing but with dynamically created fields
>>> dg = DataGrabber(infields=['arg1','arg2'])
>>> dg.inputs.template = '%s/%s.nii'
>>> dg.inputs.arg1 = 'foo'
>>> dg.inputs.arg2 = 'foo'

however this latter form can be used with iterables and iterfield in a pipeline.
Dynamically created, user-defined input and output fields
>>> dg = DataGrabber(infields=['sid'], outfields=['func','struct','ref'])
>>> dg.inputs.base_directory = '.'
>>> dg.inputs.template = '%s/%s.nii'
>>> dg.inputs.template_args['func'] = [['sid','[f3','f5']]
>>> dg.inputs.template_args['struct'] = [['sid', 'struct']]
>>> dg.inputs.template_args['ref'] = [['sid','ref']]
>>> dg.inputs.sid = 's1'

Change the template only for output field struct. The rest use the general template
>>> dg.inputs.field_template = dict(struct='%s/struct.nii')
>>> dg.inputs.template_args['struct'] = [['sid']]
```

Inputs:
[Mandatory]

sort_filelist: (a boolean)
Sort the filelist that matches the template

template: (a unicode string)
layout used to get files. relative to base directory if defined

[Optional]

base_directory: (a pathlike object or string representing an existing directory)
Path to the base directory consisting of subject data.
raise_on_empty: (a boolean, nipype default value: True)
Generate exception if list is empty for a given field

drop_blank_outputs: (a boolean, nipype default value: False)
Remove `None` entries from output lists

template_args: (a dictionary with keys which are a unicode string and
with values which are a list of items which are a list of items
which are any value)
Information to plug into template

Outputs:

None

## 87.4 DataSink

**Link to code**

Generic datasink module to store structured outputs
Primarily for use within a workflow. This interface allows arbitrary creation of input attributes. The names of these attributes define the directory structure to create for storage of the files or directories.
The attributes take the following form:
string[[[@]][@]]string[[[@]][@]]string]…
where parts between [] are optional.
An attribute such as contrasts.@con will create a ‘contrasts’ directory to store the results linked to the attribute.
If the @ is left out, such as in ‘contrasts.con’, a subdirectory ‘con’ will be created under ‘contrasts’.
the general form of the output is:

'base_directory/container/parameterization/destloc/filename'

destloc = string[[[@]][@]]string[[[@]][@]]string] and
filename comes from the input to the connect statement.

**Warning:** This is not a thread-safe node because it can write to a common shared location. It will not complain when it overwrites a file.

**Note:** If both substitutions and regexp_substitutions are used, then substitutions are applied first followed by regexp_substitutions.
This interface **cannot** be used in a MapNode as the inputs are defined only when the connect statement is executed.
87.4.1 Examples

```python
>>> ds = DataSink()
>>> ds.inputs.base_directory = 'results_dir'
>>> ds.inputs.container = 'subject'
>>> ds.inputs.structural = 'structural.nii'
>>> setattr(ds.inputs, 'contrasts.@con', ['cont1.nii', 'cont2.nii'])
>>> setattr(ds.inputs, 'contrasts.alt', ['cont1a.nii', 'cont2a.nii'])
>>> ds.run()  # doctest: +SKIP
```

To use DataSink in a MapNode, its inputs have to be defined at the time the interface is created.

```python
>>> ds = DataSink(infields=['contrasts.@con'])
>>> ds.inputs.base_directory = 'results_dir'
>>> ds.inputs.container = 'subject'
>>> ds.inputs.structural = 'structural.nii'
>>> setattr(ds.inputs, 'contrasts.@con', ['cont1.nii', 'cont2.nii'])
>>> setattr(ds.inputs, 'contrasts.alt', ['cont1a.nii', 'cont2a.nii'])
>>> ds.run()  # doctest: +SKIP
```

Inputs:

- **base_directory**: (a pathlike object or string representing a directory) Path to the base directory for storing data.
- **container**: (a unicode string) Folder within base directory in which to store output parameterization: (a boolean, nipype default value: True) store output in parametrized structure
- **strip_dir**: (a pathlike object or string representing a directory) path to strip out of filename
- **substitutions**: (a list of items which are a tuple of the form: (a unicode string, a unicode string)) List of 2-tuples reflecting string to substitute and string to replace it with
- **regexp_substitutions**: (a list of items which are a tuple of the form: (a unicode string, a unicode string)) List of 2-tuples reflecting a pair of a Python regexp pattern and a replacement string. Invoked after string `substitutions`
- **_outputs**: (a dictionary with keys which are a unicode string and with values which are any value, nipype default value: {}) remove dest_dir: (a boolean, nipype default value: False) remove dest directory when copying dirs
- **creds_path**: (a unicode string) Filepath to AWS credentials file for S3 bucket access; if not specified, the credentials will be taken from the AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY environment variables
- **encrypt_bucket_keys**: (a boolean) Flag indicating whether to use S3 server-side AES-256 encryption
- **bucket**: (any value) Boto3 S3 bucket for manual override of bucket
- **local_copy**: (a unicode string) Copy files locally as well as to S3 bucket

Outputs:

- **out_file**: (any value) datasink output
87.5 FreeSurferSource

Link to code
Generates freesurfer subject info from their directories

87.5.1 Examples

```python
>>> from nipype.interfaces.io import FreeSurferSource
>>> fs = FreeSurferSource()
>>> #fs.inputs.subjects_dir = '.'
>>> fs.inputs.subject_id = 'PWS04'
>>> res = fs.run()  # doctest: +SKIP

>>> fs.inputs.hemi = 'lh'
>>> res = fs.run()  # doctest: +SKIP
```

Inputs:

[Mandatory]
subjects_dir: (a pathlike object or string representing an existing directory)
  Freesurfer subjects directory.
subject_id: (a unicode string)
  Subject name for whom to retrieve data

[Optional]
hemi: ('both' or 'lh' or 'rh', nipype default value: both)
  Selects hemisphere specific outputs

Outputs:

T1: (a pathlike object or string representing an existing file)
  Intensity normalized whole-head volume
aseg: (a pathlike object or string representing an existing file)
  Volumetric map of regions from automatic segmentation
brain: (a pathlike object or string representing an existing file)
  Intensity normalized brain-only volume
brainmask: (a pathlike object or string representing an existing file)
  Skull-stripped (brain-only) volume
filled: (a pathlike object or string representing an existing file)
  Subcortical mass volume
norm: (a pathlike object or string representing an existing file)
  Normalized skull-stripped volume
nu: (a pathlike object or string representing an existing file)
  Non-uniformity corrected whole-head volume
orig: (a pathlike object or string representing an existing file)
  Base image conformed to Freesurfer space
rawavg: (a pathlike object or string representing an existing file)
  Volume formed by averaging input images
ribbon: (a list of items which are a pathlike object or string representing an existing file)
  Volumetric maps of cortical ribbons
wm: (a pathlike object or string representing an existing file)
  Segmented white-matter volume
wmparc: (a pathlike object or string representing an existing file)
  Aparc parcellation projected into subcortical white matter

(continues on next page)
curv: (a list of items which are a pathlike object or string representing an existing file)
Maps of surface curvature
avg_curv: (a list of items which are a pathlike object or string representing an existing file)
Average atlas curvature, sampled to subject
inflated: (a list of items which are a pathlike object or string representing an existing file)
Inflated surface meshes
pial: (a list of items which are a pathlike object or string representing an existing file)
Gray matter/pia mater surface meshes
area_pial: (a list of items which are a pathlike object or string representing an existing file)
Mean area of triangles each vertex on the pial surface is associated with
curv_pial: (a list of items which are a pathlike object or string representing an existing file)
Curvature of pial surface
smoothwm: (a list of items which are a pathlike object or string representing an existing file)
Smoothed original surface meshes
sphere: (a list of items which are a pathlike object or string representing an existing file)
Spherical surface meshes
sulc: (a list of items which are a pathlike object or string representing an existing file)
Surface maps of sulcal depth
thickness: (a list of items which are a pathlike object or string representing an existing file)
Surface maps of cortical thickness
volume: (a list of items which are a pathlike object or string representing an existing file)
Surface maps of cortical volume
white: (a list of items which are a pathlike object or string representing an existing file)
White/gray matter surface meshes
jacobian_white: (a list of items which are a pathlike object or string representing an existing file)
Distortion required to register to spherical atlas
graymid: (a list of items which are a pathlike object or string representing an existing file)
Graymid/midthickness surface meshes
label: (a list of items which are a pathlike object or string representing an existing file)
Volume and surface label files
annot: (a list of items which are a pathlike object or string representing an existing file)
Surface annotation files
aparc_aseg: (a list of items which are a pathlike object or string representing an existing file)
Aparc parcellation projected into aseg volume
sphere_reg: (a list of items which are a pathlike object or string representing an existing file)
Spherical registration file
aseg_stats: (a list of items which are a pathlike object or string representing an existing file)
Automated segmentation statistics file
wmparc_stats: (a list of items which are a pathlike object or string representing an existing file)

White matter parcellation statistics file
aparc_stats: (a list of items which are a pathlike object or string representing an existing file)

Aparc parcellation statistics files
BA_stats: (a list of items which are a pathlike object or string representing an existing file)

Brodman Area statistics files
aparc_a2009s_stats: (a list of items which are a pathlike object or string representing an existing file)

Aparc a2009s parcellation statistics files
curv_stats: (a list of items which are a pathlike object or string representing an existing file)

Curvature statistics files
entorhinal_exvivo_stats: (a list of items which are a pathlike object or string representing an existing file)

Entorhinal exvivo statistics files

87.6 IOBase

Link to code
Inputs:
None

Outputs:
None

87.7 JSONFileGrabber

Link to code
Datagrabber interface that loads a json file and generates an output for every first-level object

87.7.1 Example

```python
>>> import pprint
>>> from nipype.interfaces.io import JSONFileGrabber
>>> jsonSource = JSONFileGrabber()
>>> jsonSource.inputs.defaults = {'param1': 'overrideMe', 'param3': 1.0}
>>> res = jsonSource.run()
>>> pprint.pprint(res.outputs.get())
{'param1': 'overrideMe', 'param3': 1.0}
>>> jsonSource.inputs.in_file = os.path.join(datadir, 'jsongrabber.txt')
>>> res = jsonSource.run()
>>> pprint.pprint(res.outputs.get())  # doctest: +ELLIPSIS
{'param1': 'exampleStr', 'param2': 4, 'param3': 1.0}
```
[Optional]
in_file: (a pathlike object or string representing an existing file)
JSON source file
defaults: (a dictionary with keys which are any value and with values
which are any value)
JSON dictionary that sets default output values, overridden by values
found in in_file

Outputs:
None

87.8 JSONFileSink

Link to code
Very simple frontend for storing values into a JSON file. Entries already existing in in_dict will be overridden
by matching entries dynamically added as inputs.

Warning: This is not a thread-safe node because it can write to a common shared location. It
will not complain when it overwrites a file.

```python
>>> jsonsink = JSONFileSink(input_names=['subject_id',
...                                          'some_measurement'])
>>> jsonsink.inputs.subject_id = 's1'
>>> jsonsink.inputs.some_measurement = 11.4
>>> jsonsink.run() # doctest: +SKIP
```

Using a dictionary as input:

```python
>>> dictsink = JSONFileSink()
>>> dictsink.inputs.in_dict = {'subject_id': 's1',
...                              'some_measurement': 11.4}
>>> dictsink.run() # doctest: +SKIP
```

Inputs:

```python
[Optional]
out_file: (a pathlike object or string representing a file)
JSON sink file
in_dict: (a dictionary with keys which are any value and with values
which are any value, nipype default value: {})
input JSON dictionary
_outputs: (a dictionary with keys which are any value and with values
which are any value, nipype default value: {})
```

Outputs:

```python
out_file: (a pathlike object or string representing a file)
JSON sink file
```

87.9 MySQLSink

Link to code
Very simple frontend for storing values into MySQL database.
87.9.1 Examples

```python
>>> sql = MySQLSink(input_names=['subject_id', 'some_measurement'])
>>> sql.inputs.database_name = 'my_database'
>>> sql.inputs.table_name = 'experiment_results'
>>> sql.inputs.username = 'root'
>>> sql.inputs.password = 'secret'
>>> sql.inputs.subject_id = 's1'
>>> sql.inputs.some_measurement = 11.4
>>> sql.run() # doctest: +SKIP
```

Inputs:

- **host**: (a unicode string, nipype default value: localhost)
  - mutually_exclusive: config
  - requires: username, password
- **config**: (a pathlike object or string representing a file)
  - MySQL Options File (same format as my.cnf)
  - mutually_exclusive: host
- **database_name**: (a unicode string)
  - Otherwise known as the schema name
- **table_name**: (a unicode string)

[Optional]

- **username**: (a unicode string)
- **password**: (a unicode string)

Outputs:

None

87.10 S3DataGrabber

Link to code

Generic datagrabber module that wraps around glob in an intelligent way for neuroimaging tasks to grab files from Amazon S3.

Works exactly like DataGrabber, except, you must specify an S3 “bucket” and “bucket_path” to search for your data and a “local_directory” to store the data. “local_directory” should be a location on HDFS for Spark jobs. Additionally, “template” uses regex style formatting, rather than the glob-style found in the original DataGrabber.

87.10.1 Examples

```python
>>> s3grab = S3DataGrabber(infields=['subj_id'], outfields=['func', 'anat'])
>>> s3grab.inputs.bucket = 'openneuro'
>>> s3grab.inputs.sort_filelist = True
>>> s3grab.inputs.template = '${a}
>>> s3grab.inputs.anon = True
>>> s3grab.inputs.bucket_path = 'ds000101/ds000101_R2.0.0/uncompressed/
>>> s3grab.inputs.local_directory = '/tmp'
>>> s3grab.inputs.field_template = {'anat': '${anat}/$s_T1w.nii.gz',
... 'func': '${func}/${s_task-simon_run-1_bold.nii.gz'}
>>> s3grab.inputs.template_args = {'anat': [['subj_id', 'subj_id']},
```
...                 'func': [['subj_id', 'subj_id']]}
>>> s3grab.inputs.subj_id = 'sub-01'
>>> s3grab.run()  # doctest: +SKIP

**Inputs:**

[Mandatory]
- bucket: (a unicode string)
  - Amazon S3 bucket where your data is stored
- sort_filelist: (a boolean)
  - Sort the filelist that matches the template
- template: (a unicode string)
  - Layout used to get files. Relative to bucket_path if defined. Uses regex rather than glob style formatting.

[Optional]
- anon: (a boolean, nipype default value: False)
  - Use anonymous connection to s3. If this is set to True, boto may print a urlopen error, but this does not prevent data from being downloaded.
- region: (a unicode string, nipype default value: us-east-1)
  - Region of s3 bucket
- bucket_path: (a unicode string, nipype default value: )
  - Location within your bucket for subject data.
- local_directory: (a pathlike object or string representing an existing directory)
  - Path to the local directory for subject data to be downloaded and accessed. Should be on HDFS for Spark jobs.
- raise_on_empty: (a boolean, nipype default value: True)
  - Generate exception if list is empty for a given field
- template_args: (a dictionary with keys which are a unicode string and with values which are a list of items which are a list of items which are any value)
  - Information to plug into template

**Outputs:**

None

### 87.11 SQLiteSink

**Link to code**

Very simple frontend for storing values into SQLite database.

**Warning:** This is not a thread-safe node because it can write to a common shared location. It will not complain when it overwrites a file.

### 87.11.1 Examples

```python
>>> sql = SQLiteSink(input_names=['subject_id', 'some_measurement'])
>>> sql.inputs.database_file = 'my_database.db'
>>> sql.inputs.table_name = 'experiment_results'
>>> sql.inputs.subject_id = 's1'

(continues on next page)```
>>> sql.inputs.some_measurement = 11.4
>>> sql.run() # doctest: +SKIP

Inputs:

[Mandatory]
database_file: (a pathlike object or string representing an existing file)
table_name: (a unicode string)

Outputs:

None

87.12 SSHDataGrabber

[Link to code]
Extension of DataGrabber module that downloads the file list and optionally the files from a SSH server. The SSH operation must not need user and password so an SSH agent must be active in where this module is being run.

Attention: Doesn't support directories currently

87.12.1 Examples

```python
>>> from nipype.interfaces.io import SSHDataGrabber
>>> dg = SSHDataGrabber()
>>> dg.inputs.hostname = 'test.rebex.net'
>>> dg.inputs.user = 'demo'
>>> dg.inputs.password = 'password'
>>> dg.inputs.base_directory = 'pub/example'
```

Pick all files from the base directory

```python
>>> dg.inputs.template = '*'
```

Pick all files starting with “s” and a number from current directory

```python
>>> dg.inputs.template_expression = 'regexp'
>>> dg.inputs.template = 'pop[0-9].*'  
```

Same thing but with dynamically created fields

```python
>>> dg = SSHDataGrabber(infields=['arg1','arg2'])
>>> dg.inputs.hostname = 'test.rebex.net'
>>> dg.inputs.user = 'demo'
>>> dg.inputs.password = 'password'
>>> dg.inputs.base_directory = 'pub'
>>> dg.inputs.template = '%s/$s.txt'
>>> dg.inputs.arg1 = 'example'
>>> dg.inputs.arg2 = 'foo'
```

however this latter form can be used with iterables and iterfield in a pipeline. Dynamically created, user-defined input and output fields
>>> dg = SSHDataGrabber(infields=['sid'], outfields=['func','struct','ref'])
>>> dg.inputs.hostname = 'myhost.com'
>>> dg.inputs.base_directory = '/main_folder/my_remote_dir'
>>> dg.inputs.template_args['func'] = [['sid'],['f3','f5']]
>>> dg.inputs.template_args['struct'] = [['sid'],['struct']]  
>>> dg.inputs.template_args['ref'] = [['sid'],['ref']]  
>>> dg.inputs.sid = 's1'

Change the template only for output field struct. The rest use the general template

>>> dg.inputs.field_template = dict(struct='%s/struct.nii')
>>> dg.inputs.template_args['struct'] = [['sid']]  

87.13 SelectFiles

Link to code

Flexibly collect data from disk to feed into workflows. This interface uses the {}-based string formatting syntax to plug values (possibly known only at workflow execution time) into string templates and collect files from persistent storage. These templates can also be combined with glob wildcards. The field names in the formatting template (i.e. the terms in braces) will become inputs fields on the interface, and the keys in the templates dictionary will form the output fields.
87.13.1 Examples

```python
>>> import pprint
>>> from nipype import SelectFiles, Node
>>> templates={"T1": "{subject_id}/struct/T1.nii",
...   "epi": "{subject_id}/func/f[0, 1].nii"}
>>> dg = Node(SelectFiles(templates), "selectfiles")
>>> dg.inputs.subject_id = "subj1"
>>> pprint.pprint(dg.outputs.get()) # doctest:
{'T1': <undefined>, 'epi': <undefined>}

The same thing with dynamic grabbing of specific files:

```python
>>> templates["epi"] = "{subject_id}/func/f{run!s}.nii"
>>> dg = Node(SelectFiles(templates), "selectfiles")
>>> dg.inputs.subject_id = "subj1"
>>> dg.inputs.run = [2, 4]

Input:

<table>
<thead>
<tr>
<th>[Optional]</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_directory: (a pathlike object or string representing an existing directory)</td>
</tr>
<tr>
<td>Root path common to templates.</td>
</tr>
<tr>
<td>sort_filelist: (a boolean, nipype default value: True)</td>
</tr>
<tr>
<td>When matching multiple files, return them in sorted order.</td>
</tr>
<tr>
<td>raise_on_empty: (a boolean, nipype default value: True)</td>
</tr>
<tr>
<td>Raise an exception if a template pattern matches no files.</td>
</tr>
<tr>
<td>force_lists: (a boolean or a list of items which are a unicode string, nipype default value: False)</td>
</tr>
<tr>
<td>Whether to return outputs as a list even when only one file matches the template. Either a boolean that applies to all output fields or a list of output field names to coerce to a list</td>
</tr>
</tbody>
</table>

Output:

None

87.14 XNATSink

Link to code

Generic datasink module that takes a directory containing a list of nifti files and provides a set of structured output fields.

Inputs:

<table>
<thead>
<tr>
<th>[Mandatory]</th>
</tr>
</thead>
<tbody>
<tr>
<td>server: (a unicode string)</td>
</tr>
<tr>
<td>mutually_exclusive: config</td>
</tr>
<tr>
<td>requires: user, pwd</td>
</tr>
<tr>
<td>config: (a pathlike object or string representing a file)</td>
</tr>
<tr>
<td>mutually_exclusive: server</td>
</tr>
<tr>
<td>project_id: (a unicode string)</td>
</tr>
<tr>
<td>Project in which to store the outputs</td>
</tr>
<tr>
<td>subject_id: (a unicode string)</td>
</tr>
<tr>
<td>Set to subject id</td>
</tr>
<tr>
<td>experiment_id: (a unicode string)</td>
</tr>
<tr>
<td>Set to workflow name</td>
</tr>
</tbody>
</table>

(continues on next page)
[Optional]

_outputs: (a dictionary with keys which are a unicode string and with values which are any value, nipype default value: {})

user: (a unicode string)

pwd: (a string)

cache_dir: (a pathlike object or string representing a directory)

_assessor_id: (a unicode string)

Option to customize outputs representation in XNAT - assessor level will be used with specified id

mutually_exclusive: reconstruction_id

_reconstruction_id: (a unicode string)

Option to customize outputs representation in XNAT - reconstruction level will be used with specified id

mutually_exclusive: assessor_id

share: (a boolean, nipype default value: False)

Option to share the subjects from the original project instead of creating new ones when possible - the created experiments are then shared back to the original project.

Outputs:

None

87.15 XNATSource

Link to code

Generic XNATSource module that wraps around the pyxnat module in an intelligent way for neuroimaging tasks to grab files and data from an XNAT server.

87.15.1 Examples

```python
>>> from nipype.interfaces.io import XNATSource

Pick all files from current directory

>>> dg = XNATSource()
>>> dg.inputs.template = '*'

>>> dg = XNATSource(infields=["project","subject","experiment","assessor","inout"], outfields="")
>>> dg.inputs.query_template = '/projects/%s/subjects/%s/experiments/%s' /

>>> dg.inputs.project = 'IMAGEN'
>>> dg.inputs.subject = 'IMAGEN_000000001274'
>>> dg.inputs.experiment = '*SessionA*'  
>>> dg.inputs.assessor = '*ADNI_MPRAGE_nii'
>>> dg.inputs.inout = 'out'

>>> dg = XNATSource(infields=['sid'], outfields=['struct','func'])
>>> dg.inputs.query_template = '/projects/IMAGEN/subjects/%s/experiments/*SessionA*/' /

>>> dg.inputs.query_template_args['struct'] = ['sid','ADNI_MPRAGE']
>>> dg.inputs.query_template_args['func'] = ['sid','EPI_faces']
>>> dg.inputs.sid = 'IMAGEN_000000001274'
```
**Inputs:**

[Mandatory]
- query_template: (a unicode string)
  Layout used to get files. Relative to base directory if defined
- server: (a unicode string)
  mutually_exclusive: config
  requires: user, pwd
- config: (a pathlike object or string representing a file)
  mutually_exclusive: server

[Optional]
- query_template_args: (a dictionary with keys which are a unicode string and with values which are a list of items which are a list of items which are any value, nipype default value: {'outfiles': []})
  Information to plug into template
- user: (a unicode string)
- pwd: (a string)
- cache_dir: (a pathlike object or string representing a directory)
  Cache directory

**Outputs:**

None

### 87.16 add_traits()

Link to code
Add traits to a traited class.
All traits are set to Undefined by default

### 87.17 copytree()

Link to code
Recursively copy a directory tree using nipype.utils.filemanip.copytree()
This is not a thread-safe routine. However, in the case of creating new directories, it checks to see if a particular directory has already been created by another process.

### 87.18 push_file()

Link to code

### 87.19 quote_id()

Link to code

### 87.20 unquote_id()

Link to code
88.1 MeshFix

Link to code
Wraps the executable command `meshfix`.
MeshFix v1.2-alpha - by Marco Attene, Mirko Windhoff, Axel Thielscher.
See also:
http://jmshlib.sourceforge.net  Sourceforge page
http://simnibs.de/installation/meshfixandgetfem  Ubuntu installation instructions
If MeshFix is used for research purposes, please cite the following paper: M. Attene - A lightweight approach
to repairing digitized polygon meshes. The Visual Computer, 2010. (c) Springer.
Accepted input formats are OFF, PLY and STL. Other formats (like .msh for gmsh) are supported only partially.

88.1.1 Example

```python
>>> import nipype.interfaces.meshfix as mf
>>> fix = mf.MeshFix()
>>> fix.inputs.in_file1 = 'lh-pial.stl'
>>> fix.inputs.in_file2 = 'rh-pial.stl'
>>> fix.run()  # doctest: +SKIP
>>> fix.cmdline
'meshfix lh-pial.stl rh-pial.stl -o lh-pial_fixed.off'
```

Inputs:

|Mandatory|
in_file1: (a pathlike object or string representing an existing file)
  argument: ``%s``, position: 1

|Optional|
number_of_biggest_shells: (an integer (int or long))
  Only the N biggest shells are kept
  argument: ``--shells %d``
epsilon_angle: (0.0 <= a floating point number <= 2.0)
  Epsilon angle in degrees (must be between 0 and 2)
  argument: ``-a %f``
join_overlapping_largest_components: (a boolean)
  Join 2 biggest components if they overlap, remove the rest.
  argument: `'--j'`
  mutually_exclusive: join_closest_components

join_closest_components: (a boolean)
  Join the closest pair of components.
  argument: `'--jc'`
  mutually_exclusive: join_closest_components

quiet_mode: (a boolean)
  Quiet mode, don't write much to stdout.
  argument: `'--q'`

dont_clean: (a boolean)
  Don't Clean
  argument: `'--no-clean'`

save_as_stl: (a boolean)
  Result is saved in stereolithographic format (.stl)
  argument: `'--stl'`
  mutually_exclusive: save_as_vrml, save_as_freesurfer_mesh

save_as_vrml: (a boolean)
  Result is saved in VRML1.0 format (.wrl)
  argument: `'--wrl'`
  mutually_exclusive: save_as_stl, save_as_freesurfer_mesh

save_as_freesurfer_mesh: (a boolean)
  Result is saved in freesurfer mesh format
  argument: `'--fsmesh'`
  mutually_exclusive: save_as_vrml, save_as_stl

remove_handles: (a boolean)
  Remove handles
  argument: `'--remove-handles'`

uniform_remeshing_steps: (an integer (int or long))
  Number of steps for uniform remeshing of the whole mesh
  argument: `'--u %d'`
  requires: uniform_remeshing_vertices

uniform_remeshing_vertices: (an integer (int or long))
  Constrains the number of vertices. Must be used with
  uniform_remeshing_steps
  argument: `'--vertices %d'`
  requires: uniform_remeshing_steps

laplacian_smoothing_steps: (an integer (int or long))
  The number of laplacian smoothing steps to apply
  argument: `'--smooth %d'`

x_shift: (an integer (int or long))
  Shifts the coordinates of the vertices when saving. Output must be
  in FreeSurfer format
  argument: `'--smooth %d'`

cut_outer: (an integer (int or long))
  Remove triangles of 1st that are outside of the 2nd shell.
  argument: `'--cut-outer %d'`

cut_inner: (an integer (int or long))
  Remove triangles of 1st that are inside of the 2nd shell. Dilate 2nd
  by N; Fill holes and keep only 1st afterwards.
  argument: `'--cut-inner %d'`

decouple_inin: (an integer (int or long))
  Treat 1st file as inner, 2nd file as outer component. Resolve
  overlaps by moving inners triangles inwards. Constrain the min
  distance between the components > d.
  argument: `'--decouple-inin %d'`
decouple_outin: (an integer (int or long))
Treat 1st file as outer, 2nd file as inner component. Resolve
overlaps by moving outers triangles inwards. Constrain the min
distance between the components > d.
argument: `--decouple-outin %d`

decouple_outout: (an integer (int or long))
Treat 1st file as outer, 2nd file as inner component. Resolve
overlaps by moving outers triangles outwards. Constrain the min
distance between the components > d.
argument: `--decouple-outout %d`

finetuning_inwards: (a boolean)
Used to fine-tune the minimal distance between surfaces.
argument: `--fineTuneIn`, position: -3
requires: finetuning_distance, finetuning_substeps

finetuning_outwards: (a boolean)
Similar to finetuning_inwards, but ensures minimal distance in the
other direction
argument: `--fineTuneOut`, position: -3
mutually_exclusive: finetuning_inwards
requires: finetuning_distance, finetuning_substeps

finetuning_distance: (a float)
Used to fine-tune the minimal distance between surfaces. A minimal
distance d is ensured, and reached in n substeps. When using the
surfaces for subsequent volume meshing by gmsh, this step prevent
too flat tetrahedra2)
argument: `--f %f`, position: -2
requires: finetuning_substeps

finetuning_substeps: (an integer (int or long))
Used to fine-tune the minimal distance between surfaces. A minimal
distance d is ensured, and reached in n substeps. When using the
surfaces for subsequent volume meshing by gmsh, this step prevent
too flat tetrahedra2)
argument: `--d %d`, position: -1
requires: finetuning_distance

dilation: (an integer (int or long))
Dilate the surface by d. d < 0 means shrinking.
argument: `--dilate %d`

set_intersections_to_one: (a boolean)
If the mesh contains intersections, return value = 1. If saved in
gmsh format, intersections will be highlighted.
argument: `--intersect`

in_file2: (a pathlike object or string representing an existing file)
argument: `-%s`, position: 2

output_type: ('stl' or 'msh' or 'wrl' or 'vrml' or 'fs' or 'off',
nipype default value: off)
The output type to save the file as.

out_filename: (a pathlike object or string representing a file)
The output filename for the fixed mesh file
argument: `-%o %s`

args: (a unicode string)
Additional parameters to the command
argument: `-%s`

environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipype default value: {})
Environment variables

Outputs:
mesh_file: (a pathlike object or string representing an existing file)

The output mesh file
CHAPTER 89

interfaces.nilearn

89.1 NilearnBaseInterface

Link to code

Inputs:

None

Outputs:

None

89.2 SignalExtraction

Link to code

Extracts signals over tissue classes or brain regions

```python
>>> seinterface = SignalExtraction()
>>> seinterface.inputs.in_file = 'functional.nii'
>>> seinterface.inputs.label_files = 'segmentation0.nii.gz'
>>> seinterface.inputs.out_file = 'means.tsv'
>>> segments = ['CSF', 'GrayMatter', 'WhiteMatter']
>>> seinterface.inputs.class_labels = segments
>>> seinterface.inputs.detrend = True
>>> seinterface.inputs.include_global = True
```

Inputs:

[Mandatory]

in_file: (a pathlike object or string representing an existing file)

4-D fMRI nii file

label_files: (a list of items which are a pathlike object or string

representing an existing file)

a 3-D label image, with 0 denoting background, or a list of 3-D

probability maps (one per label) or the equivalent 4D file.

class_labels: (a list of items which are any value)

Human-readable labels for each segment in the label file, in order.
The length of class_labels must be equal to the number of segments (background excluded). This list corresponds to the class labels in label_file in ascending order.

[Optional]

out_file: (a pathlike object or string representing a file, nipype default value: signals.tsv)
The name of the file to output to. signals.tsv by default.

incl_shared_variance: (a boolean, nipype default value: True)
By default (True), returns simple time series calculated from each region independently (e.g., for noise regression). If False, returns unique signals for each region, discarding shared variance (e.g., for connectivity. Only has effect with 4D probability maps.

include_global: (a boolean, nipype default value: False)
If True, include an extra column labeled "GlobalSignal", with values calculated from the entire brain (instead of just regions).

detrend: (a boolean, nipype default value: False)
If True, perform detrending using nilearn.

Outputs:

out_file: (a pathlike object or string representing an existing file)
    tsv file containing the computed signals, with as many columns as there are labels and as many rows as there are timepoints in in_file, plus a header row with values from class_labels.
90.1 PETPVC

Link to code
Wraps the executable command petpvc.
Use PETPVC for partial volume correction of PET images.
PETPVC is a software from the Nuclear Medicine Department of the UCL University Hospital, London, UK. Its source code is here: https://github.com/UCL/PETPVC
Its command line help shows this:

```
-i –input < filename > = PET image file
-o –output < filename > = Output file
[-m –mask < filename >] = Mask image file
-p –pvc < keyword > = Desired PVC method
-x < X > = The full-width at half maximum in mm along x-axis
-y < Y > = The full-width at half maximum in mm along y-axis
-z < Z > = The full-width at half maximum in mm along z-axis
[-d –debug ] = Prints debug information
[-n –iter [ Val ] ]
   = Number of iterations With: Val (Default = 10)
[-k [ Val ] ]
   = Number of deconvolution iterations With: Val (Default = 10)
[-a –alpha [ aval ] ]
   = Alpha value With: aval (Default = 1.5)
[-s –stop [ stopval ] ]
   = Stopping criterion With: stopval (Default = 0.01)
```

90.1.1 Technique - keyword

- Geometric transfer matrix - “GTM”
- Labbe approach - “LABBE”
- Richardson-Lucy - “RL”
- Van-Cittert - “VC”
• Region-based voxel-wise correction - “RBV”
• RBV with Labbe - “LABBE+RBV”
• RBV with Van-Cittert - “RBV+VC”
• RBV with Richardson-Lucy - “RBV+RL”
• RBV with Labbe and Van-Cittert - “LABBE+RBV+VC”
• RBV with Labbe and Richardson-Lucy- “LABBE+RBV+RL”
• Multi-target correction - “MTC”
• MTC with Labbe - “LABBE+MTC”
• MTC with Van-Cittert - “MTC+VC”
• MTC with Richardson-Lucy - “MTC+RL”
• MTC with Labbe and Van-Cittert - “LABBE+MTC+VC”
• MTC with Labbe and Richardson-Lucy- “LABBE+MTC+RL”
• Iterative Yang - “IY”
• Iterative Yang with Van-Cittert - “IY+VC”
• Iterative Yang with Richardson-Lucy - “IY+RL”
• Muller Gartner - “MG”
• Muller Gartner with Van-Cittert - “MG+VC”
• Muller Gartner with Richardson-Lucy - “MG+RL”

90.1.2 Examples

```python
>>> from ..testing import example_data
>>> #TODO get data for PETPVC
>>> pvc = PETPVC()
>>> pvc.inputs.in_file = 'pet.nii.gz'
>>> pvc.inputs.mask_file = 'tissues.nii.gz'
>>> pvc.inputs.out_file = 'pet_pvc_rbv.nii.gz'
>>> pvc.inputs.pvc = 'RBV'
>>> pvc.inputs.fwhm_x = 2.0
>>> pvc.inputs.fwhm_y = 2.0
>>> pvc.inputs.fwhm_z = 2.0
>>> outs = pvc.run() #doctest: +SKIP
```

Inputs:

[Mandatory]

in_file: (a pathlike object or string representing an existing file)
    PET image file
    argument: `'-i %s'`
mask_file: (a pathlike object or string representing an existing file)
    Mask image file
    argument: `'-m %s'`
pvc: {'GTM' or 'IY' or 'IY+RL' or 'IY+VC' or 'LABBE' or 'LABBE+MTC'
    or 'LABBE+MTC+RL' or 'LABBE+MTC+VC' or 'LABBE+RBV' or
    'LABBE+RBV+RL' or 'LABBE+RBV+VC' or 'MG' or 'MG+RL' or 'MG+VC' or
    'MTC' or 'MTC+RL' or 'MTC+VC' or 'RBV' or 'RBV+RL' or 'RBV+VC' or
    'RL' or 'VC'}
    Desired PVC method
    argument: `'-p %s'`
fwhm_x: (a float)
    The full-width at half maximum in mm along x-axis
    argument: `'-x %.4f'`
fwhm_y: (a float)
    The full-width at half maximum in mm along y-axis
    argument: `'-y %.4f'`

(continues on next page)
fwhm_z: (a float)
The full-width at half maximum in mm along z-axis
  argument: ``-z %.4f``

[Optional]
out_file: (a pathlike object or string representing a file)
  Output file
  argument: ``-o %s``
debug: (a boolean, nipype default value: False)
  Prints debug information
  argument: ``-d``
n_iter: (an integer (int or long), nipype default value: 10)
  Number of iterations
  argument: ``-n %d``
n_deconv: (an integer (int or long), nipype default value: 10)
  Number of deconvolution iterations
  argument: ``-k %d``
alpha: (a float, nipype default value: 1.5)
  Alpha value
  argument: ``-a %.4f``
stop_crit: (a float, nipype default value: 0.01)
  Stopping criterion
  argument: ``-s %.4f``
args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
  of class 'str' and with values which are a bytes or None or a
  value of class 'str', nipype default value: {})
91.1 Quickshear

Link to code
Wraps the executable command quickshear.
Quickshear is a simple geometric defacing algorithm
Given an anatomical image and a reasonable brainmask, Quickshear estimates a shearing plane with the brain mask on one side and the face on the other, zeroing out the face side.

```python
>>> from nipype.interfaces.quickshear import Quickshear
>>> qs = Quickshear(in_file='T1.nii', mask_file='brain_mask.nii')
>>> qs.cmdline
'quickshear T1.nii brain_mask.nii T1_defaced.nii'
```

In the absence of a precomputed mask, a simple pipeline can be generated with any tool that generates brain masks:

```python
>>> from nipype.pipeline import engine as pe
>>> from nipype.interfaces import utility as niu
>>> from nipype.interfaces.fsl import BET

>>> deface_wf = pe.Workflow('deface_wf')
>>> inputnode = pe.Node(niu.IdentityInterface(['in_file']),
...                     name='inputnode')
>>> outputnode = pe.Node(niu.IdentityInterface(['out_file']),
...                       name='outputnode')
>>> bet = pe.Node(BET(mask=True), name='bet')
>>> quickshear = pe.Node(Quickshear(), name='quickshear')

>>> deface_wf.connect([(inputnode, bet, [('in_file', 'in_file')]),
...                     (inputnode, quickshear, [('in_file', 'in_file')]),
...                     (bet, quickshear, [('mask_file', 'mask_file')]),
...                     (quickshear, outputnode, [('out_file', 'out_file')]),
...                     (inputnode, outputnode, [('in_file', 'in_file')])])
>>> inputnode.inputs.in_file = 'T1.nii'
>>> res = deface_wf.run()  # doctest: +SKIP
```

Inputs:
[Mandatory]
in_file: (a pathlike object or string representing an existing file)
    neuroimage to deface
    argument: `"%s"`, position: 1
mask_file: (a pathlike object or string representing an existing file)
    brain mask
    argument: `"%s"`, position: 2

[Optional]
out_file: (a pathlike object or string representing a file)
    defaced output image
    argument: `"%s"`, position: 3
buff: (an integer (int or long))
    buffer size (in voxels) between shearing plane and the brain
    argument: `"%d"`, position: 4
args: (a unicode string)
    Additional parameters to the command
    argument: `"%s"
environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipype default value: {})  
Environment variables

Outputs:

out_file: (a pathlike object or string representing an existing file)
    defaced output image

91.1.1 References:

None
 CHAPTER 92

interfaces.vtkbase

92.1 configure_input_data()

Link to code
Configure the input data for vtk pipeline object obj. Copied from latest version of mayavi

92.2 vtk_old()

Link to code
Checks if VTK uses the old-style pipeline (VTK<6.0)

92.3 vtk_output()

Link to code
Configure the input data for vtk pipeline object obj.

Developer Guides

• API
93.1 caching.memory

93.1.1 Module: caching.memory

Inheritance diagram for nipype.caching.memory:

- memory.PipeFunc
- memory.Memory

Using nipype with persistence and lazy recompu-
tation but without explicit name-steps pipeline: getting back scope in command-line based programming.

93.1.2 Classes

Memory
class nipype.caching.memory.Memory(base_dir)

Bases: object

Memory context to provide caching for interfaces

Parameters

base_dir: string The directory name of the location for the caching

Methods

cache(self, interface) Returns a callable that caches the output of an interface

clear_previous_runs(self[, warn]) Remove all the cache that were not used in the latest run of the memory object: i.e.

clear_previous_runs(self[, warn]) Remove all the cache that were not used in the latest run of the memory object: i.e.

__init__(self, base_dir)

Initialize self. See help(type(self)) for accurate signature.

cache(self, interface) Returns a callable that caches the output of an interface

Parameters

interface: nipype interface The nipype interface class to be wrapped and cached

Returns

pipe_func: a PipeFunc callable object An object that can be used as a function to apply the interface to arguments. Inputs of the interface are given as keyword arguments, bearing the same name as the name in the inputs specs of the interface.

Examples

>>> from tempfile import mkdtemp
>>> mem = Memory(mkdtemp())
>>> from nipype.interfaces import fsl

Here we create a callable that can be used to apply an fsl.Merge interface to files

>>> fsl_merge = mem.cache(fsl.Merge)

Now we apply it to a list of files. We need to specify the list of input files and the dimension along which the files should be merged.

>>> results = fsl_merge(in_files=['a.nii', 'b.nii'],
...                      dimension='t')  # doctest: +SKIP

We can retrieve the resulting file from the outputs: >>> results.outputs.merged_file # doctest: +SKIP

clear_previous_runs(self, warn=True)

Remove all the cache that were not used in the latest run of the memory object: i.e. since the corresponding Python object was created.

Parameters

warn: boolean, optional If true, echoes warning messages for all directory removed
clear_runs_since(self, day=None, month=None, year=None, warn=True)

Remove all the cache that were not used since the given date

Parameters

day, month, year: integers, optional

The integers specifying the latest day (in localtime) that a node should have been accessed to be kept. If not given, the current date is used.

warn: boolean, optional

If true, echoes warning messages for all directory removed

PipeFunc
class  nipype.caching.memory.PipeFunc(interface, base_dir, callback=None)

Bases: object

Callable interface to nipype.interface objects

Use this to wrap nipype.interface object and call them specifying their input with keyword arguments:

```python
fsi_merge = PipeFunc(fsl.Merge, base_dir='.
out = fsl_merge(in_files=files, dimension='t')
```

Methods

__call__(self, **kwargs) Call self as a function.

__init__(self, interface, base_dir, callback=None)

Parameters

interface: a nipype interface class

The interface class to wrap

base_dir: a string

The directory in which the computation will be stored

callback: a callable

An optional callable called each time after the function is called.

93.1.3 Functions

nipype.caching.memory.read_log(filename, run_dict=None)
nipype.caching.memory.rm_all_but(base_dir, dirs_to_keep, warn=False)

Remove all the sub-directories of base_dir, but those listed

Parameters

base_dir: string

The base directory

dirs_to_keep: set

The names of the directories to keep

93.2 conftest

93.2.1 Module: conftest

93.2.2 Functions

nipype.conftest.add_np(doctest_namespace)
nipype.conftest.pytest_unconfigure(config)

93.3 interfaces.matlab

93.3.1 Module: interfaces.matlab

Inheritance diagram for nipype.interfaces.matlab:
General matlab interface code

93.3.2 Classes

MatlabCommand

class nipype.interfaces.matlab.MatlabCommand(matlab_cmd=None, **inputs)
Bases: nipype.interfaces.base.core.CommandLine
Interface that runs matlab code

```python
>>> import nipype.interfaces.matlab as matlab
>>> mlab = matlab.MatlabCommand(mfile=False)  # don't write script file
>>> mlab.inputs.script = "which('who')"  
>>> out = mlab.run()  # doctest: +SKIP
```

Attributes

- **always_run** Should the interface be always run even if the inputs were not changed?
  Only applies to interfaces being run within a workflow context.
- **can_resume** Defines if the interface can reuse partial results after interruption.
- **cmd** sets base command, immutable
- **cmdline** command plus any arguments (args)
- **output_spec**
- **terminal_output**
- **version** interfaces should implement a version property

Methods

- **aggregate_outputs**(self[, runtime, ...]) Collate expected outputs and apply output traits validation.
- **help**(returnhelp) Prints class help
- **input_spec** alias of MatlabInputSpec
- **load_inputs_from_json**(self[, json_file[, ...]]) A convenient way to load pre-set inputs from a JSON file.
- **run**(self[, cwd, ignore_exception]) Execute this interface.
- **save_inputs_to_json**(self, json_file) A convenient way to save current inputs to a JSON file.
- **set_default_matlab_cmd**(matlab_cmd) Set the default MATLAB command line for MATLAB classes.
- **set_default_mfile**(mfile) Set the default MATLAB script file format for MATLAB classes.
- **set_default_paths**(paths) Set the default MATLAB paths for MATLAB classes.

Continued on next page
Table 3 – continued from previous page

| set_default_terminal_output | (output_type) Set the default terminal output for CommandLine Interfaces. |

raise_exception

version_from_command

__init__(self, matlab_cmd=None, **inputs)
initializes interface to matlab (default ‘matlab -nodesktop -nosplash’)

aggregate_outputs (self, runtime=None, needed_outputs=None)
Collate expected outputs and apply output traits validation.

always_run
Should the interface be always run even if the inputs were not changed? Only applies to interfaces being run within a workflow context.

can_resume
Defines if the interface can reuse partial results after interruption. Only applies to interfaces being run within a workflow context.

cmd
sets base command, immutable

cmdline
command plus any arguments (args) validates arguments and generates command line

classmethod help (returnhelp=False)
Prints class help

input_spec
alias of MatlabInputSpec

load_inputs_from_json (self, json_file, overwrite=True)
A convenient way to load pre-set inputs from a JSON file.

output_spec = None
raise_exception (self, runtime)

references_ = []

resource_monitor = True
run (self, cwd=None, ignore_exception=None, **inputs)
Execute this interface.
This interface will not raise an exception if runtime.returncode is non-zero.

Parameters

cwd [specify a folder where the interface should be run]

inputs [allows the interface settings to be updated]

Returns

results [an InterfaceResult object containing a copy of the instance]

that was executed, provenance information and, if successful, results

save_inputs_to_json (self, json_file)
A convenient way to save current inputs to a JSON file.

classmethod set_default_matlab_cmd(matlab_cmd)
Set the default MATLAB command line for MATLAB classes.
This method is used to set values for all MATLAB subclasses. However, setting this will not update the output type for any existing instances. For these, assign the <instance>.inputs.matlab_cmd.

classmethod set_default_mfile (mfile)
Set the default MATLAB script file format for MATLAB classes.
This method is used to set values for all MATLAB subclasses. However, setting this will not update the output type for any existing instances. For these, assign the <instance>.inputs.mfile.

classmethod set_default_paths (paths)
Set the default MATLAB paths for MATLAB classes.
This method is used to set values for all MATLAB subclasses. However, setting this will not update the output type for any existing instances. For these, assign the <instance>.inputs.paths.
```
classmethod set_default_terminal_output(output_type)
    Set the default terminal output for CommandLine Interfaces.
    This method is used to set default terminal output for CommandLine Interfaces. However, setting
    this will not update the output type for any existing instances. For these, assign the <instance>.terminal_output.

    terminal_output
    version
        interfaces should implement a version property

    version_from_command(self, flag='--v', cmd=None)
```

**MatlabInputSpec**

class nipype.interfaces.matlab.MatlabInputSpec(**kwargs)
    Bases: nipype.interfaces.base.specs.CommandLineInputSpec
    Basic expected inputs to Matlab interface

**Methods**

```
add_class_trait(name, \*\*trait) Adds a named trait attribute to this class.
add_trait(self, name, \*\*trait) Adds a trait attribute to this object.
add_trait_category(category) Adds a trait category to a class.
all_trait_names(self) Returns the list of all trait names, including implicitly defined traits.
base_trait(self, name) Returns the base trait definition for a trait attribute.
class_default_traits_view() Returns the name of the default traits view for the class.
class_editable_traits() Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current class.
class_trait_names(\*\*metadata) Returns a list of the names of all trait attributes whose definitions match the set of metadata criteria specified.
class_trait_view_elements() Returns the ViewElements object associated with the class.
class_traits(\*\*metadata) Returns a dictionary containing the definitions of all of the trait attributes of the class that match the set of metadata criteria.
class_visible_traits() Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current class, that should be GUI visible
clone_traits(self, \*\*metadata) Clones a new object from this one, optionally copying only a specified set of traits.
configure_traits(self, filename, view, \*\*metadata) Creates and displays a dialog box for editing values of trait attributes, as if it were a complete, self-contained GUI application.
copy_traits(self, other, \*\*metadata) Copies another object’s trait attributes into this one.
copyable_trait_names(self, \*\*metadata) Returns the list of trait names to copy or clone by default.
default_traits_view(self) Returns the name of the default traits view for the object’s class.
```
Table 4 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>edit_traits(self[, view, parent, kind, ...])</code></td>
<td>Displays a user interface window for editing trait attribute values.</td>
</tr>
<tr>
<td><code>editable_traits(self)</code></td>
<td>Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current object.</td>
</tr>
</tbody>
</table>
| `get(self, 
\^*kwargs)` | Returns traited class as a dict |
| `get_hashval(self[, hash_method])` | Return a dictionary of our items with hashes for each file. |
| `get_traitsfree(self, 
\^*kwargs)` | Returns traited class as a dict |
| `has_metadata(self, name, metadata[, value, ...])` | Return has_metadata for the requested trait name in this interface |
| `has_traits_interface(self, \*interfaces)` | Returns whether the object implements a specified traits interface. |
| `items(self)` | Name, trait generator for user modifiable traits |
| `on_trait_change(self, handler[, name, ...])` | Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association. |
| `on_trait_event(self, handler[, name, ...])` | Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association. |
| `print_traits(self[, show_help])` | Prints the values of all explicitly-defined, non-event trait attributes on the current object, in an easily readable format. |
| `remove_trait(self, name)` | Removes a trait attribute from this object. |
| `reset_traits(self[, traits])` | Resets some or all of an object’s trait attributes to their default values. |
| `set(self[, trait_change_notify])` | Shortcut for setting object trait attributes. |
| `set_trait_dispatch_handler(name, \klass[, ...])` | Sets a trait notification dispatch handler. |
| `sync_trait(self, trait_name, object[, ...])` | Synchronizes the value of a trait attribute on this object with a trait attribute on another object. |
| `trait(self, name[, force, copy])` | Returns the trait definition for the name trait attribute. |
| `trait_context(self)` | Returns the default context to use for editing or configuring traits. |
| `trait_get(self, \^*kwargs)` | Returns traited class as a dict |
| `trait_items_event(event_trait,name,items_event)` | Returns traited class as a dict |
| `trait_monitor(handler[, remove])` | Adds or removes the specified handler from the list of active monitors. |
| `trait_names(self, \^*metadata)` | Returns a list of the names of all trait attributes whose definitions match the set of metadata criteria specified. |
| `trait_property_changed(...)` | Shortcut for setting object trait attributes. |
| `trait_set(self[, trait_change_notify])` | Shortcut for setting object trait attributes. |
| `trait_setq(self, \^\*traits)` | Returns a list of the immediate (or all) subclasses of this class. |
| `trait_view(self[, name, view_element])` | Gets or sets a ViewElement associated with an object’s class. |
| `trait_view_elements(self)` | Returns the ViewElements object associated with the object’s class. |

Continued on next page
Table 4 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>trait_views(self[, klass])</code></td>
<td>Returns a list of the names of all view elements associated with the current object’s class.</td>
</tr>
<tr>
<td><code>traits(self, *metadata)</code></td>
<td>Returns a dictionary containing the definitions of all of the trait attributes of this object that match the set of metadata criteria.</td>
</tr>
<tr>
<td><code>traits_init()</code></td>
<td></td>
</tr>
<tr>
<td><code>traits_inited(True)</code></td>
<td></td>
</tr>
<tr>
<td><code>validate_trait(self, name, value)</code></td>
<td>Validates whether a value is legal for a trait.</td>
</tr>
<tr>
<td><code>visible_traits(self)</code></td>
<td>Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current object, that should be GUI visible</td>
</tr>
</tbody>
</table>

```
add_trait_listener
class_trait_view
remove_trait_listener
```

```
__init__(self, **kwargs)

Initialize handlers and inputs

classmethod add_class_trait(name, *trait)

Adds a named trait attribute to this class.

Parameters

name [str] Name of the attribute to add.
*trait : A trait or a value that can be converted to a trait using Trait() Trait definition of the attribute. It can be a single value or a list equivalent to an argument list for the Trait() function.

add_trait(self, name, *trait)

Adds a trait attribute to this object.

Parameters

name [str] Name of the attribute to add.
*trait : Trait or a value that can be converted to a trait by Trait(). Trait definition for name. If more than one value is specified, it is equivalent to passing the entire list of values to Trait().

classmethod add_trait_category(category)

Adds a trait category to a class.

classmethod add_trait_listener(self, object, prefix=’”)

classmethod all_trait_names(self)

Returns the list of all trait names, including implicitly defined traits.

base_trait(self, name)

Returns the base trait definition for a trait attribute.

Parameters

name [str] Name of the attribute whose trait definition is returned.

classmethod class_default_traits_view()

Returns the name of the default traits view for the class.

classmethod class_editable_traits()

Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current class.

classmethod class_trait_names(**metadata)

Returns a list of the names of all trait attributes whose definitions match the set of metadata criteria specified.

Parameters

**metadata : Criteria for selecting trait attributes.

classmethod class_trait_view(name=None, view_element=None)

classmethod class_trait_view_elements()
```
Returns the ViewElements object associated with the class. The returned object can be used to access all the view elements associated with the class.

```python
classmethod class_traits(**metadata)
```

Returns a dictionary containing the definitions of all of the trait attributes of the class that match the set of metadata criteria.

Parameters

**metadata** : Criteria for selecting trait attributes.

```python
classmethod class_visible_traits()
```

Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current class, that should be GUI visible

```python
clone_traits(self, traits=None, memo=None, copy=None, **metadata)
```

Clones a new object from this one, optionally copying only a specified set of traits.

Parameters

- **traits** [list of strings] The list of names of the trait attributes to copy.
- **memo** [dict] A dictionary of objects that have already been copied.
- **copy** [str] The type of copy deep or shallow to perform on any trait that does not have explicit ‘copy’ metadata. A value of None means ‘copy reference’.

Returns

new : The newly cloned object.

```python
configure_traits(self, filename=None, view=None, kind=None, edit=True, context=None, handler=None, id=' ', scrollable=None, **args)
```

Creates and displays a dialog box for editing values of trait attributes, as if it were a complete, self-contained GUI application.

Parameters

- **filename** [str] The name (including path) of a file that contains a pickled representation of the current object. When this parameter is specified, the method reads the corresponding file (if it exists) to restore the saved values of the object’s traits before displaying them. If the user confirms the dialog box (by clicking OK), the new values are written to the file. If this parameter is not specified, the values are loaded from the in-memory object, and are not persisted when the dialog box is closed.
- **view** [View or str] A View object (or its name) that defines a user interface for editing trait attribute values of the current object. If the view is defined as an attribute on this class, use the name of the attribute. Otherwise, use a reference to the view object. If this attribute is not specified, the View object returned by trait_view() is used.
- **kind** [str] The type of user interface window to create. See the traitsui.view.kind_trait trait for values and their meanings. If kind is unspecified or None, the kind attribute of the View object is used.
- **edit** [bool] Indicates whether to display a user interface. If filename specifies an existing file, setting edit to False loads the saved values from that file into the object without requiring user interaction.
- **context** [object or dictionary] A single object or a dictionary of string/object pairs, whose trait attributes are to be edited. If not specified, the current object is used
- **handler** [Handler] A handler object used for event handling in the dialog box. If None, the default handler for Traits UI is used.
- **id** [str] A unique ID for persisting preferences about this user interface, such as size and position. If not specified, no user preferences are saved.
- **scrollable** [bool] Indicates whether the dialog box should be scrollable. When set to True, scroll bars appear on the dialog box if it is not large enough to display all of the items in the view at one time.

Returns

True on success.

```python
copy_traits(self, other, traits=None, memo=None, copy=None, **metadata)
```

Copies another object’s trait attributes into this one.

Parameters
other  [object] The object whose trait attribute values should be copied.

traits  [list of strings] A list of names of trait attributes to copy. If None or unspecified, the set of names returned by trait_names() is used. If 'all' or an empty list, the set of names returned by all_trait_names() is used.

memo  [dict] A dictionary of objects that have already been copied.

copy  [None | ‘deep’ | ‘shallow’] The type of copy to perform on any trait that does not have explicit ‘copy’ metadata. A value of None means ‘copy reference’.

Returns

unassignable  [list of strings] A list of attributes that the method was unable to copy, which is empty if all the attributes were successfully copied.

copyable_trait_names  (self, **metadata)

Returns the list of trait names to copy or clone by default.

default_traits_view  (self)

Returns the name of the default traits view for the object’s class.

edit_traits  (self, view=None, parent=None, kind=None, context=None, handler=None, id=", scrollable=None, **args)

Displays a user interface window for editing trait attribute values.

Parameters

view  [View or string] A View object (or its name) that defines a user interface for editing trait attribute values of the current object. If the view is defined as an attribute on this class, use the name of the attribute. Otherwise, use a reference to the view object. If this attribute is not specified, the View object returned by trait_view() is used.

parent  [toolkit control] The reference to a user interface component to use as the parent window for the object’s UI window.

kind  [str] The type of user interface window to create. See the traitsui.view.kind_trait trait for values and their meanings. If kind is unspecified or None, the kind attribute of the View object is used.

context  [object or dictionary] A single object or a dictionary of string/object pairs, whose trait attributes are to be edited. If not specified, the current object is used.

handler  [Handler] A handler object used for event handling in the dialog box. If None, the default handler for Traits UI is used.

id  [str] A unique ID for persisting preferences about this user interface, such as size and position. If not specified, no user preferences are saved.

scrollable  [bool] Indicates whether the dialog box should be scrollable. When set to True, scroll bars appear on the dialog box if it is not large enough to display all of the items in the view at one time.

Returns

A UI object.

editable_traits  (self)

Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current object.

get  (self, **kwargs)

Returns traited class as a dict

Augments the trait get function to return a dictionary without notification handles

get_hashval  (self, hash_method=None)

Return a dictionary of our items with hashes for each file.

Searches through dictionary items and if an item is a file, it calculates the md5 hash of the file contents and stores the file name and hash value as the new key value. However, the overall bunch hash is calculated only on the hash value of a file. The path and name of the file are not used in the overall hash calculation.

Returns

list_withhash  [dict] Copy of our dictionary with the new file hashes included with each file.

hashvalue  [str] The md5 hash value of the traited spec
get_traitsfree (self, **kwargs)
Returns traited class as a dict
Augments the trait get function to return a dictionary without any traits. The dictionary does not contain any attributes that were Undefined

has_metadata (self, name, metadata, value=None, recursive=True)
Returns has_metadata for the requested trait name in this interface

has_traits_interface (self, *interfaces)
Returns whether the object implements a specified traits interface.
Parameters
*interfaces : One or more traits Interface (sub)classes.

items (self)
Name, trait generator for user modifiable traits

on_trait_change (self, handler, name=None, remove=False, dispatch='same', priority=False, deferred=False, target=None)
Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.
Parameters
handler [function] A trait notification function for the name trait attribute, with one of the signatures described below.
name [str] The name of the trait attribute whose value changes trigger the notification.
The name can specify complex patterns of trait changes using an extended name syntax, which is described below.
remove [bool] If True, removes the previously-set association between handler and name; if False (the default), creates the association.
dispatch [str] A string indicating the thread on which notifications must be run. Possible values are:

<table>
<thead>
<tr>
<th>value</th>
<th>dispatch</th>
</tr>
</thead>
<tbody>
<tr>
<td>same</td>
<td>Run notifications on the same thread as this one.</td>
</tr>
<tr>
<td>ui</td>
<td>Run notifications on the UI thread. If the current thread is the UI thread, the notifications are executed immediately; otherwise, they are placed on the UI event queue.</td>
</tr>
<tr>
<td>fast</td>
<td>Alias for ui.</td>
</tr>
<tr>
<td>new</td>
<td>Run notifications in a new thread.</td>
</tr>
</tbody>
</table>

on_trait_event (self, handler, name=None, remove=False, dispatch='same', priority=False, deferred=False, target=None)
Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.
Parameters
handler [function] A trait notification function for the name trait attribute, with one of the signatures described below.
name [str] The name of the trait attribute whose value changes trigger the notification.
The name can specify complex patterns of trait changes using an extended name syntax, which is described below.
remove [bool] If True, removes the previously-set association between handler and name; if False (the default), creates the association.
dispatch [str] A string indicating the thread on which notifications must be run. Possible values are:
### Trait Methods

<table>
<thead>
<tr>
<th>value</th>
<th>dispatch</th>
</tr>
</thead>
<tbody>
<tr>
<td>same</td>
<td>Run notifications on the same thread as this one.</td>
</tr>
<tr>
<td>ui</td>
<td>Run notifications on the UI thread. If the current thread is the UI thread, the notifications are executed immediately; otherwise, they are placed on the UI event queue.</td>
</tr>
<tr>
<td>fast</td>
<td>Alias for ui.</td>
</tr>
<tr>
<td>new</td>
<td>Run notifications in a new thread.</td>
</tr>
</tbody>
</table>

```python
package_version = <Version('1.2.2.dev0+g216db0d67')>
```

```python
print_traits(self, show_help=False, **metadata)
```

Prints the values of all explicitly-defined, non-event trait attributes on the current object, in an easily readable format.

**Parameters**
- **show_help** [bool] Indicates whether to display additional descriptive information.

```python
remove_trait(self, name)
```

Removes a trait attribute from this object.

**Parameters**
- **name** [str] Name of the attribute to remove.

**Returns**
- **result** [bool] True if the trait was successfully removed.

```python
remove_trait_listener(self, object, prefix="")
```

```python
reset_traits(self, traits=None, **metadata)
```

Resets some or all of an object’s trait attributes to their default values.

**Parameters**
- **traits** [list of strings] Names of trait attributes to reset.

**Returns**
- **unresetable** [list of strings] A list of attributes that the method was unable to reset, which is empty if all the attributes were successfully reset.

```python
set(self, trait_change_notify=True, **traits)
```

Shortcut for setting object trait attributes.

**Parameters**
- **trait_change_notify** [bool] If True (the default), then each value assigned may generate a trait change notification. If False, no trait change notifications will be generated. (see also: trait_setq)

**Returns**
- **self** The method returns this object, after setting attributes.

```python
classmethod set_trait_dispatch_handler(name, klass, override=False)
```

Sets a trait notification dispatch handler.

```python
classmethod sync_trait(self, trait_name, object, alias=None, mutual=True, remove=False)
```

Synchronizes the value of a trait attribute on this object with a trait attribute on another object.

**Parameters**
- **trait_name** [str] Name of the trait attribute on this object.
- **object** [object] The object with which to synchronize.
- **alias** [str] Name of the trait attribute on other; if None or omitted, same as name.
- **mutual** [bool or int] Indicates whether synchronization is mutual (True or non-zero) or one-way (False or zero)
- **remove** [bool or int] Indicates whether synchronization is being added (False or zero) or removed (True or non-zero)

```python
trait(self, name, force=False, copy=False)
```

Returns the trait definition for the name trait attribute.

**Parameters**
- **name** [str] Name of the attribute whose trait definition is to be returned.
force [bool] Indicates whether to return a trait definition if name is not explicitly defined.
copy [bool] Indicates whether to return the original trait definition or a copy.

trait_context (self)
Returns the default context to use for editing or configuring traits.

trait_get (self, **kwargs)
Returns traited class as a dict
Augments the trait get function to return a dictionary without notification handles

trait_items_event (event_trait, name, items_event)
classmethod trait_monitor (handler, remove=False)
Adds or removes the specified handler from the list of active monitors.

Parameters
- handler [function] The function to add or remove as a monitor.
- remove [bool] Flag indicating whether to remove (True) or add the specified handler as a
  monitor for this class.

trait_names (self, **metadata)
Returns a list of the names of all trait attributes whose definitions match the set of metadata criteria
specified.

Parameters
- **metadata : Criteria for selecting trait attributes.

trait_property_changed (name, old_value[, new_value ])

trait_set (self, trait_change_notify=True, **traits)
Shortcut for setting object trait attributes.

Parameters
- trait_change_notify [bool] If True (the default), then each value assigned may generate
  a trait change notification. If False, then no trait change notifications will be generated.
  (see also: trait_set)
- **traits : Key/value pairs, the trait attributes and their values to be set

Returns
- self : The method returns this object, after setting attributes.

trait_setq (self, **traits)
Shortcut for setting object trait attributes.

Parameters
- **traits : Key/value pairs, the trait attributes and their values to be set. No trait change
  notifications will be generated for any values assigned (see also: trait_set).

Returns
- self : The method returns this object, after setting attributes.

classmethod trait_subclasses (all=False)
Returns a list of the immediate (or all) subclasses of this class.

Parameters
- all [bool] Indicates whether to return all subclasses of this class. If False, only immediate
  subclasses are returned.

trait_view (self, name=None, view_element=None)
Gets or sets a ViewElement associated with an object’s class.

Parameters
- name [str] Name of a view element
- view_element [ViewElement] View element to associate

Returns
- A view element.

trait_view_elements (self)
Returns the ViewElements object associated with the object’s class.
The returned object can be used to access all the view elements associated with the class.

trait_views (self, klass=None)
Returns a list of the names of all view elements associated with the current object’s class.

Parameters

93.3. interfaces.matlab
klass [class] A class, such that all returned names must correspond to instances of this
class. Possible values include:
- Group
- Item
- View
- ViewElement
- ViewSubElement

traits (self, **metadata)
Returns a dictionary containing the definitions of all of the trait attributes of this object that match the
set of metadata criteria.

Parameters
**metadata : Criteria for selecting trait attributes.

traits_init()

traits_inited([True])

validate_trait (self, name, value)
Validates whether a value is legal for a trait.
Returns the validated value if it is valid.

visible_traits (self)
Returns an alphabetically sorted list of the names of non-event trait attributes associated with the
current object, that should be GUI visible

wrappers = {'extended': <class 'traits.trait_notifiers.ExtendedTraitChangeNotifyWrapper'>, 'fast_ui': <class 'traits.trait_notifiers.FastUITraitChangeNotifyWrapper'>}

93.3.3 Function
nipype.interfaces.matlab.get_matlab_command()

93.4 pipeline.engine.base

93.4.1 Module: pipeline.engine.base
Inheritance diagram for nipype.pipeline.engine.base:

base.EngineBase

Defines functionality for pipelined execution of inter-
terfaces.
The EngineBase class implements the more general view of a task.

93.4.2 EngineBase
class nipype.pipeline.engine.base.EngineBase (name=Name, base_dir=Name)

Bases: object
Defines common attributes and functions for workflows and nodes.

Attributes
- fullname
- inputs
iname Name for expanded iterable
name
outputs

Methods

clone(self, name) Clone an EngineBase object

__init__(self, name=None, base_dir=None)
Initialize base parameters of a workflow or node

Parameters
name [string (mandatory)] Name of this node. Name must be alphanumeric and not contain any special characters (e.g., ‘,’ ‘@’).
base_dir [string] base output directory (will be hashed before creations) default=None, which results in the use of mkdtemp

close (self, name)
Clone an EngineBase object

Parameters
name [string (mandatory)] A clone of node or workflow must have a new name

fullname
inputs
iname
   Name for expanded iterable
load (self, filename)
name
outputs
save (self, filename=None)

93.5 pipeline.engine.nodes

93.5.1 Module: pipeline.engine.nodes
Inheritance diagram for pipeline.engine.nodes:


Defines functionality for pipelined execution of interfaces
The Node class provides core functionality for batch processing.
93.5.2 Classes

JoinNode

class nipype.pipeline.engine.nodes.JoinNode(interface, name, joinsource, joinfield=None, unique=False, **kwargs)

Bases: nipype.pipeline.engine.nodes.Node

Wraps interface objects that join inputs into a list.

Examples

```python
>>> import nipype.pipeline.engine as pe
>>> from nipype import Node, JoinNode, Workflow
>>> from nipype.interfaces.utility import IdentityInterface
>>> from nipype.interfaces import (ants, dcm2nii, fsl)

>>> wf = Workflow(name='preprocess')
>>> inputspec = Node(IdentityInterface(fields=['image']),
...                   name='inputspec')
>>> inputspec.iterables = [('image',
...                         ['img1.nii', 'img2.nii', 'img3.nii'])]
>>> img2flt = Node(fsl.ImageMaths(out_data_type='float'),
...                name='img2flt')
>>> wf.connect(inputspec, 'image', img2flt, 'in_file')
>>> average = JoinNode(ants.AverageImages(), joinsource='inputspec',
...                     joinfield='images', name='average')
>>> wf.connect(img2flt, 'out_file', average, 'images')
>>> realign = Node(fsl.FLIRT(), name='realign')
>>> wf.connect(img2flt, 'out_file', realign, 'in_file')
>>> wf.connect(average, 'output_average_image', realign, 'reference')
>>> strip = Node(fsl.BET(), name='strip')
>>> wf.connect(realign, 'out_file', strip, 'in_file')
```

Attributes

- **fullname**
  - The JoinNode inputs include the join field overrides.
- **interface**
  - Return the underlying interface object
- **itername**
  - Name for expanded iterable
- **joinsource**
- **mem_gb**
  - Get estimated memory (GB)
- **n_procs**
  - Get the estimated number of processes/threads
- **name**
- **needed_outputs**
- **outputs**
  - Return the output fields of the underlying interface
- **result**
  - Get result from result file (do not hold it in memory)

Methods

- **clone**(self, name)
  - Clone an EngineBase object
- **get_output**(self, parameter)
  - Retrieve a particular output of the node
- **hash_exists**(self, updatehash)
  - Decorate the new is_cached method with hash updating to maintain backwards compatibility.
- **help**(self)
  - Print interface help
- **is_cached**(self, rm_outdated)
  - Check if the interface has been run previously, and whether cached results are up-to-date.
Table 6 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>output_dir(self)</td>
<td>Return the location of the output directory for the node</td>
</tr>
<tr>
<td>run(self[, updatehash])</td>
<td>Execute the node in its directory.</td>
</tr>
<tr>
<td>set_input(self, parameter, val)</td>
<td>Set interface input value</td>
</tr>
<tr>
<td>update(self, **opts)</td>
<td>Update inputs</td>
</tr>
</tbody>
</table>

```
__init__(self, interface, name, joinsource, joinfield=None, unique=False, **kwargs)
```

**Parameters**

- **interface** [interface object] node specific interface (fsl.Bet(), spm.Coregister())
- **name** [alphanumeric string] node specific name
- **joinsource** [node name] name of the join predecessor iterable node
- **joinfield** [string or list of strings] name(s) of list input fields that will be aggregated. The default is all of the join node input fields.
- **unique** [flag indicating whether to ignore duplicate input values]

See Node docstring for additional keyword arguments.

```
clone(self, name)
```

Clone an EngineBase object

**Parameters**

- **name** [string (mandatory)] A clone of node or workflow must have a new name

```
fullname
```

```
get_output (self, parameter)
```

Retrieve a particular output of the node

```
hash_exists (self, updatehash=False)
```

Decorate the new is_cached method with hash updating to maintain backwards compatibility.

```
help (self)
```

Print interface help

```
inputs
```

The JoinNode inputs include the join field overrides.

```
interface
```

Return the underlying interface object

```
is_cached (self, rm_outdated=False)
```

Check if the interface has been run previously, and whether cached results are up-to-date.

```
itername
```

Name for expanded iterable

```
joinfield = None
```

the fields to join

```
joinsource
```

the join predecessor iterable node

```
load (self, filename)
```

```
mem_gb
```

Get estimated memory (GB)

```
n_procs
```

Get the estimated number of processes/threads

```
name
```

```
needed_outputs
```

```
output_dir (self)
```

Return the location of the output directory for the node

```
outputs
```

Return the output fields of the underlying interface

```
result
```
Get result from result file (do not hold it in memory)

```
run(self, updatehash=False)
```

Execute the node in its directory.

**Parameters**

- **updatehash**: boolean  
  When the hash stored in the output directory as a result of a previous run does not match that calculated for this execution, updatehash=True only updates the hash without re-running.

```
save(self, filename=None)
set_input(self, parameter, val)
update(self, **opts)
```

Set interface input value

Update inputs

### MapNode

**class** nipype.pipeline.engine.nodes.MapNode(interface, iterfield, name, serial=False, nested=False, **kwargs)**

Bases: nipype.pipeline.engine.nodes.Node

Wraps interface objects that need to be iterated on a list of inputs.

#### Examples

```python
>>> from nipype import MapNode
>>> from nipype.interfaces import fsl

>>> realign = MapNode(fsl.MCFLIRT(), 'in_file', 'realign')
>>> realign.inputs.in_file = ['functional.nii', ...
... 'functional2.nii', ...
... 'functional3.nii']
```

```
>>> realign.run()  # doctest: +SKIP
```

**Attributes**

- **fullname**  
  Return the inputs of the underlying interface

- **interface**  
  Return the underlying interface object

- **itername**  
  Name for expanded iterable

- **mem_gb**  
  Get estimated memory (GB)

- **n_procs**  
  Get the estimated number of processes/threads

- **name**

- **needed_outputs**  
  Return the output fields of the underlying interface

- **result**  
  Get result from result file (do not hold it in memory)

**Methods**

- **clone(self, name)**  
  Clone an EngineBase object

- **get_output(self, parameter)**  
  Retrieve a particular output of the node

- **get_subnodes(self)**  
  Generate subnodes of a mapnode and write pre-execution report

- **hash_exists(self[, updatehash])**  
  Decorate the new *is_cached* method with hash updating to maintain backwards compatibility.

- **help(self)**  
  Print interface help

- **is_cached(self[, rm_outdated])**  
  Check if the interface has been run previously, and whether cached results are up-to-date.
Table 7 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>num_subnodes</code></td>
<td>Get the number of subnodes to iterate in this MapNode</td>
</tr>
<tr>
<td><code>output_dir</code></td>
<td>Return the location of the output directory for the node</td>
</tr>
<tr>
<td><code>run</code></td>
<td>Execute the node in its directory.</td>
</tr>
<tr>
<td><code>set_input</code></td>
<td>Set interface input value or nodewrapper attribute</td>
</tr>
<tr>
<td><code>update</code></td>
<td>Update inputs</td>
</tr>
</tbody>
</table>

**`__init__`** (self, interface, iterfield, name, serial=False, nested=False, **kwargs)

**Parameters**
- `interface` [interface object] node specific interface (fsl.Bet(), spm.Coregister())
- `iterfield` [string or list of strings] name(s) of input fields that will receive a list of whatever kind of input they take. The node will be run separately for each value in these lists. For more than one input, the values are paired (i.e. it does not compute a combinatorial product).
- `name` [alphanumeric string] node specific name
- `serial` [boolean] flag to enforce executing the jobs of the mapnode in a serial manner rather than parallel
- `nested` [boolean] support for nested lists. If set, the input list will be flattened before running and the nested list structure of the outputs will be restored.

See Node docstring for additional keyword arguments.

**`clone`** (self, name)

Clone an EngineBase object

**Parameters**
- `name` [string (mandatory)] A clone of node or workflow must have a new name

**`fullname`**

Retrieve a particular output of the node

**`get_subnodes`** (self)

Generate subnodes of a mapnode and write pre-execution report

**`hash_exists`** (self, updatehash=False)

Decorate the new `is_cached` method with hash updating to maintain backwards compatibility.

**`help`** (self)

Print interface help

**`inputs`**

Return the inputs of the underlying interface

**`interface`**

Return the underlying interface object

**`is_cached`** (self, rm_outdated=False)

Check if the interface has been run previously, and whether cached results are up-to-date.

**`itername`**

Name for expanded iterable

**`load`** (self, filename)

Get estimated memory (GB)

**`n_procs`**

Get the estimated number of processes/threads

**`needed_outputs`**
num_subnodes (self)
    Get the number of subnodes to iterate in this MapNode

output_dir (self)
    Return the location of the output directory for the node

outputs
    Return the output fields of the underlying interface

result
    Get result from result file (do not hold it in memory)

run (self, updatehash=False)
    Execute the node in its directory.
    Parameters
    updatehash: boolean When the hash stored in the output directory as a result of a previous run does not match that calculated for this execution, updatehash=True only updates the hash without re-running.

save (self, filename=None)

set_input (self, parameter, val)
    Set interface input value or nodewrapper attribute Priority goes to interface.

update (self, **opts)
    Update inputs

Node
class nipype.pipeline.engine.nodes.Node (interface, name, iterables=None, iter-source=None, synchronize=False, overwrite=None, needed_outputs=None, run_without_submitting=False, n_procs=None, mem_gb=0.2, **kwargs)

Bases: nipype.pipeline.engine.base.EngineBase
Wraps interface objects for use in pipeline
A Node creates a sandbox-like directory for executing the underlying interface. It will copy or link inputs into this directory to ensure that input data are not overwritten. A hash of the input state is used to determine if the Node inputs have changed and whether the node needs to be re-executed.

Examples

```python
>>> from nipype import Node
>>> from nipype.interfaces import spm
>>> realign = Node(spm.Realign(), 'realign')
>>> realign.inputs.in_files = 'functional.nii'
>>> realign.inputs.register_to_mean = True
>>> realign.run() # doctest: +SKIP
```

Attributes
 fullname
    inputs Return the inputs of the underlying interface
    interface Return the underlying interface object
    itername Name for expanded iterable
    mem_gb Get estimated memory (GB)
    n_procs Get the estimated number of processes/threads
    name
    needed_outputs
    outputs Return the output fields of the underlying interface
    result Get result from result file (do not hold it in memory)
nimpye Documentation, Release 1.2.1

Methods

- `clone(self, name)` Clone an EngineBase object
- `get_output(self, parameter)` Retrieve a particular output of the node
- `hash_exists(self[, updatehash])` Decorate the new `is_cached` method with hash updating to maintain backwards compatibility.
- `help(self)` Print interface help
- `is_cached(self[, rm_outdated])` Check if the interface has been run previously, and whether cached results are up-to-date.
- `output_dir(self)` Return the location of the output directory for the node
- `run(self[, updatehash])` Execute the node in its directory.
- `set_input(self, parameter, val)` Set interface input value
- `update(self, **opts)` Update inputs

```
load
save
```

```
__init__ (self, interface, name, iterables=None, itersource=None, synchronize=False, overwrite=None, needed_outputs=None, run_without_submitting=False, n_procs=None, mem_gb=0.2, **kwargs)
```

Parameters

- `interface` [interface object] node specific interface (fsl.Bet(), spm.Coregister())
- `name` [alphanumeric string] node specific name
- `iterables` [generator] Input field and list to iterate using the pipeline engine for example to iterate over different frac values in fsl.Bet() for a single field the input can be a tuple, otherwise a list of tuples

```python
node.iterables = ('frac', [0.5, 0.6, 0.7])
node.iterables = [('fwhm', [2, 4]), ('fieldx', [0.5, 0.6, 0.7])]
```

If this node has an itersource, then the iterables values is a dictionary which maps an iterable source field value to the target iterables field values, e.g.:

```python
inputspec.iterables = ('images', ['img1.nii', 'img2.nii'])
node.itersource = ('inputspec', ['frac'])
node.iterables = ('frac', {'img1.nii': [0.5, 0.6], 'img2.nii': [0.6, 0.7]})
```

If this node’s synchronize flag is set, then an alternate form of the iterables is a [fields, values] list, where fields is the list of iterated fields and values is the list of value tuples for the given fields, e.g.:

```python
node.synchronize = True
node.iterables = [['frac', 'threshold'], [(0.5, True), (0.6, False)]
```

- `itersource` [tuple] The (name, fields) itersource which specifies the name of the predecessor iterable node and the input fields to use from that source node. The output field values comprise the key to the iterables parameter value mapping dictionary.
- `synchronize` [boolean] Flag indicating whether iterables are synchronized. If the iterables are synchronized, then this iterable node is expanded once per iteration over all of the iterables values. Otherwise, this iterable node is expanded once per each permutation of the iterables values.
- `overwrite` [Boolean] Whether to overwrite contents of output directory if it already exists.
If directory exists and hash matches it assumes that process has been executed

**needed_outputs** [list of output_names] Force the node to keep only specific outputs. By default all outputs are kept. Setting this attribute will delete any output files and directories from the node’s working directory that are not part of the `needed_outputs`.

**run_without_submitting** [boolean] Run the node without submitting to a job engine or to a multiprocessing pool

**clone** *(self, name)*
Clone an EngineBase object

**Parameters**

**name** [string (mandatory)] A clone of node or workflow must have a new name

**fullname**
Retrieve a particular output of the node

**hash_exists** *(self, updatehash=False)*
Decorate the new `is_cached` method with hash updating to maintain backwards compatibility.

**help** *(self)*
Print interface help

**inputs**
Return the inputs of the underlying interface

**interface**
Return the underlying interface object

**is_cached** *(self, rm_outdated=False)*
Check if the interface has been run previously, and whether cached results are up-to-date.

**itername**
Name for expanded iterable

**load** *(self, filename)*
Get estimated memory (GB)

**mem_gb**
Get estimated memory (GB)

**n_procs**
Get the estimated number of processes/threads

**name**

**needed_outputs**

**output_dir** *(self)*
Return the location of the output directory for the node

**outputs**
Return the output fields of the underlying interface

**result**
Get result from result file (do not hold it in memory)

**run** *(self, updatehash=False)*
Execute the node in its directory.

**Parameters**

**updatehash** [boolean] When the hash stored in the output directory as a result of a previous run does not match that calculated for this execution, updatehash=True only updates the hash without re-running.

**save** *(self, filename=None)*

**set_input** *(self, parameter, val)*
Set interface input value

**update** *(self, **opts)*
Update inputs
93.6 pipeline.engine.utils

93.6.1 Module: pipeline.engine.utils

Utility routines for workflow graphs

93.6.2 Functions

nipype.pipeline.engine.utils.clean_working_directory(outputs, cwd, inputs, needed_outputs, config, files2keep=None, dirs2keep=None)

Removes all files not needed for further analysis from the directory

nipype.pipeline.engine.utils.count_iterables(iterables, synchronize=False)

Return the number of iterable expansion nodes.
If synchronize is True, then the count is the maximum number of iterables value lists. Otherwise, the count is the product of the iterables value list sizes.

nipype.pipeline.engine.utils.evaluate_connect_function(function_source, args, first_arg)

nipype.pipeline.engine.utils.expand_iterables(iterables, synchronize=False)

nipype.pipeline.engine.utils.export_graph(graph_in, base_dir=None, show=False, use_execgraph=False, show_connectinfo=False, dotfilename='graph.dot', format='png', simple_form=True)

Displays the graph layout of the pipeline
This function requires that pygraphviz and matplotlib are available on the system.

Parameters

- show [boolean]
  Indicate whether to generate pygraphviz output from networkx. default [False]
- use_execgraph [boolean]
  Indicates whether to use the specification graph or the execution graph. default [False]
- show_connectioninfo [boolean]
  Indicates whether to show the edge data on the graph. This makes the graph rather cluttered. default [False]

nipype.pipeline.engine.utils.format_dot(dotfilename, format='png')

Dump a directed graph (Linux only; install via brew on OSX)

nipype.pipeline.engine.utils.format_node(node, format='python', include_config=False)

Format a node in a given output syntax.

nipype.pipeline.engine.utils.generate_expanded_graph(graph_in)

Generates an expanded graph based on node parameterization
Parameterization is controlled using the iterables field of the pipeline elements. Thus if there are two nodes with iterables a=[1,2] and b=[3,4] this procedure will generate a graph with sub-graphs parameterized as (a=1,b=3), (a=1,b=4), (a=2,b=3) and (a=2,b=4).

nipype.pipeline.engine.utils.get_all_files(infile)

nipype.pipeline.engine.utils.get_levels(G)

nipype.pipeline.engine.utils.get_print_name(node, simple_form=True)

Get the name of the node
For example, a node containing an instance of interfaces.fsl.BET would be called nodename.BET.fsl

nipype.pipeline.engine.utils.load_resultfile(results_file, resolve=True)

Load InterfaceResult file from path.

Returns
result [InterfaceResult structure]
aggregate [boolean indicating whether node should aggregate_outputs]
attribute error [boolean indicating whether there was some mismatch in] versions of traits
used to store result and hence node needs to rerun

nipype.pipeline.engine.utils.merge_bundles(gl, g2)
nipype.pipeline.engine.utils.merge_dict(d1, d2, merge=<function <lambda> at 0x7fb12d4d8510>)

Merges two dictionaries, non-destructively, combining values on duplicate keys as defined by the optional merge function. The default behavior replaces the values in d1 with corresponding values in d2. (There is no other generally applicable merge strategy, but often you’ll have homogeneous types in your dicts, so specifying a merge technique can be valuable.)

Examples:

```python
>>> d1 = {'a': 1, 'c': 3, 'b': 2}
>>> d2 = merge_dict(d1, d1)
>>> len(d2)
3
>>> [d2[k] for k in ['a', 'b', 'c']]
[1, 2, 3]

>>> d3 = merge_dict(d1, d1, lambda x,y: x+y)
>>> len(d3)
3
>>> [d3[k] for k in ['a', 'b', 'c']]
[2, 4, 6]
```

nipype.pipeline.engine.utils.modify_paths(object, relative=True, basedir=None)

Convert paths in data structure to either full paths or relative paths

Supports combinations of lists, dicts, tuples, strs

Parameters
relative [boolean indicating whether paths should be set relative to the] current directory
basedir [default os.getcwd()] what base directory to use as default

nipype.pipeline.engine.utils.nodelist_runner(nodes, updatehash=False, stop_first=False)

A generator that iterates and over a list of nodes and executes them.

nipype.pipeline.engine.utils.save_hashfile(hashfile, hashed_inputs)

Save a hashfile

nipype.pipeline.engine.utils.save_resultfile(result, cwd, name, rebase=None)

Save a result pklz file to cwd.

nipype.pipeline.engine.utils.strip_temp(files, wd)

Remove temp from a list of file paths

nipype.pipeline.engine.utils.synchronize_iterables(iterables)

Synchronize the given iterables in item-wise order.

Return: the {field: value} dictionary list

Examples

```python
>>> from nipype.pipeline.engine.utils import synchronize_iterables
>>> iterables = dict(a=lambda: [1, 2], b=lambda: [3, 4])
>>> synced = synchronize_iterables(iterables)
>>> synced == [{'a': 1, 'b': 3}, {'a': 2, 'b': 4}]
True

>>> iterables = dict(a=lambda: [1, 2], b=lambda: [3], c=lambda: [4, 5, 6])
>>> synced = synchronize_iterables(iterables)
>>> synced == [{'a': 1, 'b': 3, 'c': 4}, {'a': 2, 'c': 5}, {'c': 6}]
True
```
nipype.pipeline.engine.utils.topological_sort(graph, depth_first=False)

Returns a depth first sorted order if depth_first is True

nipype.pipeline.engine.utils.walk(children, level=0, path=None, usename=True)

Generate all the full paths in a tree, as a dict.

Examples

```python
>>> from nipype.pipeline.engine.utils import walk
>>> iterables = [('a', lambda: [1, 2]), ('b', lambda: [3, 4])]
>>> [val['a'] for val in walk(iterables)]
[1, 2]
>>> [val['b'] for val in walk(iterables)]
[3, 4]
```

nipype.pipeline.engine.utils.walk_files(cwd)

Extract every file and directory from a python structure

nipype.pipeline.engine.utils.walk_outputs(object)

Extract every file and directory from a python structure

nipype.pipeline.engine.utils.write_report(node, report_type=None, is_mapnode=False)

Write a report file for a node

nipype.pipeline.engine.utils.write_workflow_prov(graph, filename=None, format='all')

Write W3C PROV Model JSON file

nipipe.pipeline.engine.utils.write_workflow_resources(graph, filename=None, append=None)

Generate a JSON file with profiling traces that can be loaded in a pandas DataFrame or processed with JavaScript like D3.js

93.7 pipeline.engine.workflows

93.7.1 Module: pipeline.engine.workflows

Inheritance diagram for nipype.pipeline.engine.workflows:

```
base.EngineBase  workflows.Workflow
```

Defines functionality for pipelined execution of interfaces

The Workflow class provides core functionality for batch processing.

93.7.2 Workflow

class nipype.pipeline.engine.workflows.Workflow(name, base_dir=None)

Bases: nipype.pipeline.engine.base.EngineBase

Controls the setup and execution of a pipeline of processes.

Attributes

fullname
inputs
  itername Name for expanded iterable
  name
outputs

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_nodes(self, nodes)</td>
<td>Add nodes to a workflow</td>
</tr>
<tr>
<td>clone(self, name)</td>
<td>Clone a workflow</td>
</tr>
<tr>
<td>connect(self, *args, **kwargs)</td>
<td>Connect nodes in the pipeline.</td>
</tr>
<tr>
<td>disconnect(self, *args)</td>
<td>Disconnect nodes See the docstring for connect for format.</td>
</tr>
<tr>
<td>export(self[, filename, prefix, format, ...])</td>
<td>Export object into a different format</td>
</tr>
<tr>
<td>get_node(self, name)</td>
<td>Return an internal node by name</td>
</tr>
<tr>
<td>list_node_names(self)</td>
<td>List names of all nodes in a workflow</td>
</tr>
<tr>
<td>remove_nodes(self, nodes)</td>
<td>Remove nodes from a workflow</td>
</tr>
<tr>
<td>run(self[, plugin, plugin_args, updatehash])</td>
<td>Execute the workflow</td>
</tr>
<tr>
<td>write_graph(self[, dotfilename, graph2use, ...])</td>
<td>Generates a graphviz dot file and a png file</td>
</tr>
</tbody>
</table>

__init__(self, name, base_dir=None)

Create a workflow object.

Parameters

- **name** [alphanumeric string] unique identifier for the workflow
- **base_dir** [string, optional] path to workflow storage

add_nodes(self, nodes)

Add nodes to a workflow

Parameters

- **nodes** [list] A list of EngineBase-based objects

clone(self, name)

Clone a workflow

Parameters

- **name** alphanumeric name unique name for the workflow

Note: Will reset attributes used for executing workflow. See __init_runtime_fields.

connect(self, *args, **kwargs)

Connect nodes in the pipeline.

This routine also checks if inputs and outputs are actually provided by the nodes that are being connected.

Creates edges in the directed graph using the nodes and edges specified in the connection_list. Uses the NetworkX method DiGraph.add_edges_from.

Parameters

- **args** [list or a set of four positional arguments] Four positional arguments of the form:

  ```python
  connect(source, sourceoutput, dest, destinput)
  ```

  - **source** : nodewrapper node sourceoutput : string (must be in source.outputs) dest :
  - nodewrapper node destinput : string (must be in dest.inputs)
  - A list of 3-tuples of the following form:
[(source, target, 
    [('sourceoutput/attribute', 'targetinput'),
    ...])],
...]

Or:
[
    (source, target, 
     [([('sourceoutput1', func, arg2, ...),
        'targetinput'), ...
    ]),
...]

sourceoutput1 will always be the first argument to func
and func will be evaluated and the results sent to targetinput

currently func needs to define all its needed imports within the
function as we use the inspect module to get at the source code
and execute it remotely

disconnect (self, *args)
    Disconnect nodes See the docstring for connect for format.
export (self, filename=None, prefix='output', format='python', include_config=False)
    Export object into a different format
    Parameters
        filename: string  file to save the code to; overrides prefix
        prefix: string    prefix to use for output file
        format: string    one of “python”
        include_config: boolean  whether to include node and workflow config values
fullname
get_node (self, name)
    Return an internal node by name
inputs
itername
    Name for expanded iterable
list_node_names (self)
    List names of all nodes in a workflow
load (self, filename)
    name
outputs
remove_nodes (self, nodes)
    Remove nodes from a workflow
    Parameters
        nodes: [list] A list of EngineBase-based objects
run (self, plugin=None, plugin_args=None, updatehash=False)
    Execute the workflow
    Parameters
        plugin: plugin name or object  Plugin to use for execution. You can create your own
        plugin_args: [dictionary containing arguments to be sent to plugin] constructor. see individual
            plugin doc strings for details.
save (self, filename=None)
write_graph (self, dotfilename='graph.dot', graph2use='hierarchical', format='png', simple_form=True)
    Generates a graphviz dot file and a png file
    Parameters
            level graph without expanding internal workflow nodes; flat - expands workflow nodes
            recursively; hierarchical - expands workflow nodes recursively with a notion on hierar-
... colored - expands workflow nodes recursively with a notion on hierarchy in color;
exec - expands workflows to depict iterables
format: 'png', 'svg'
simple_form: boolean (default: True) Determines if the node name used in the graph
should be of the form 'nodename (package)' when True or 'nodename.Class.package'
when False.

write_hierarchical_dotfile (self, dotfilename=None, colored=False, simple_form=True)

93.8 sphinxext.plot_workflow

93.8.1 Module: sphinxext.plot_workflow

Inheritance diagram for nipype.sphinxext.plot_workflow:

plot_workflow.ImageFile

plot_workflow.GraphError

nipype.sphinxext.plot_workflow – Workflow plotting extension

A directive for including a nipype workflow graph in a Sphinx document. This code is forked from the plot_figure sphinx extension of matplotlib.
By default, in HTML output, workflow will include a .png file with a link to a high-res .png. In LaTeX output, it will include a .pdf. The source code for the workflow may be included as inlined content to the directive workflow:

```plaintext
.. workflow ::
    :graph2use: flat
    :simple_form: no

    from nipype.workflows.dmri.camino.connectivity_mapping import create_
    connectivity_pipeline
```

(continues on next page)
For example, the following graph has been generated inserting the previous code block in this documentation:

```python
wf = create_connectivity_pipeline()
```

Options

The `workflow` directive supports the following options:

- **graph2use**:
  ```python
  [{'hierarchical', 'colored', 'flat', 'orig', 'exec'}]
  ```
  Specify the type of graph to be generated.

- **simple_form**:
  ```python
  bool
  ```
  Whether the graph will be in detailed or simple form.

- **format**:
  ```python
  [{'python', 'doctest'}]
  ```
  Specify the format of the input.

- **include-source**:
  ```python
  bool
  ```
  Whether to display the source code. The default can be changed using the `workflow_include_source` variable in `conf.py`.

- **encoding**:
  ```python
  [str]
  ```
  If this source file is in a non-UTF8 or non-ASCII encoding, the encoding must be specified using the `:encoding:` option. The encoding will not be inferred using the `-*- coding -*-` metacomment.

Additionally, this directive supports all of the options of the `image` directive, except for `target` (since `workflow` will add its own target). These include `alt`, `height`, `width`, `scale`, `align` and `class`.

Configuration options

The `workflow` directive has the following configuration options:

- **graph2use**
  Select a graph type to use

- **simple_form**
  Determines if the node name shown in the visualization is either of the form `nodename (package)` when set to True or `nodename.Class.package` when set to False.

- **wf_include_source**
  Default value for the include-source option

- **wf_html_show_source_link**
  Whether to show a link to the source in HTML.

- **wf_pre_code**
  Code that should be executed before each workflow.

- **wf_basedir**
  Base directory, to which `workflow::` file names are relative to. (If None or empty, file names are relative to the directory where the file containing the directive is.)

- **wf_formats**
  
  **File formats to generate. List of tuples or strings::**
  ```python
  [(suffix, dpi), suffix, ...]
  ```
  that determine the file format and the DPI. For entries whose DPI was omitted, sensible defaults are chosen. When passing from the command line through `sphinx_build` the list should be passed as `suffix;dpi;suffix;dpi;...`

- **wf_html_show_formats**
  Whether to show links to the files in HTML.
**wf_rcparams** A dictionary containing any non-standard rcParams that should be applied before each workflow.

**wf_apply_rcparams** By default, rcParams are applied when `context` option is not used in a workflow directive. This configuration option overrides this behavior and applies rcParams before each workflow.

**wf_working_directory** By default, the working directory will be changed to the directory of the example, so the code can get at its data files, if any. Also its path will be added to `sys.path` so it can import any helper modules sitting beside it. This configuration option can be used to specify a central directory (also added to `sys.path`) where data files and helper modules for all code are located.

**wf_template** Provide a customized template for preparing restructured text.

### 93.8.2 Classes

**GraphError**

class `nipype.sphinxext.plot_workflow.GraphError`

Bases: `RuntimeError`

**Attributes**
- `args`

**Methods**

`with_traceback()` ► Exception.with_traceback(tb) – set self._traceback_ to tb and return self.

```python
__init__(self, /, *args, **kwargs)
```

Initialize self. See help(type(self)) for accurate signature.

```python
args
```

```python
with_traceback()
```

Exception.with_traceback(tb) – set self._traceback_ to tb and return self.

**ImageFile**

class `nipype.sphinxext.plot_workflow.ImageFile`

Bases: `object`

**Methods**

```python
filename
```

```python
filenames
```

```python
__init__(self, basename, dirname)
```

Initialize self. See help(type(self)) for accurate signature.

```python
filename (self, fnt)
```

```python
filenames (self)
```

### 93.8.3 Functions

`nipype.sphinxext.plot_workflow.contains_doctest(text)`

`nipype.sphinxext.plot_workflow.get_wf_formats(config)`

`nipype.sphinxext.plot_workflow.mark_wf_labels(app, document)`

To make graphs referenceable, we need to move the reference from the “htmlonly” (or “latexonly”) node to the actual figure node itself.

`nipype.sphinxext.plot_workflow.out_of_date(original, derived)`

Returns True if derivative is out-of-date wrt original, both of which are full file paths.
nipype.sphinxext.plot_workflow.remove_coding(text)
Remove the coding comment, which exec doesn’t like.
nipype.sphinxext.plot_workflow.render_figures(code, code_path, output_dir, output_base, context, function_name, config, graph2use, simple_form, context_reset=False, close_figs=False)
Run a nipype workflow creation script and save the graph in output_dir. Save the images under output_dir with file names derived from output_base
nipype.sphinxext.plot_workflow.run(arguments, content, options, state_machine, state, lineno)
nipype.sphinxext.plot_workflow.run_code(code, code_path, ns=None, function_name=None)
Import a Python module from a path, and run the function given by name, if function_name is not None.
nipype.sphinxext.plot_workflow.setup(app)
nipype.sphinxext.plot_workflow.unescape_doctest(text)
Extract code from a piece of text, which contains either Python code or doctests.
nipype.sphinxext.plot_workflow.wf_directive(name, arguments, options, content, lineno, content_offset, block_text, state, state_machine)

• Developer Guide
Since nipype is part of the NIPY project, we follow the same conventions documented in the NIPY Developers Guide. For bleeding-edge version help see Nightly documentation.

## 94.1 Interface Specifications

### 94.1.1 Before you start

Nipype is maintained by an enthusiastic group of developers, and we’re excited to have you join us! In case of trouble, we encourage you to post on NeuroStars with the *nipype* tag. NeuroStars.org is a platform similar to StackOverflow but dedicated to neuroinformatics. You can also post on the nipype developers mailing list: http://mail.python.org/mailman/listinfo/neuroimaging. As we are sharing a mailing list with the nipy community, please add [nipy] to the message title. Alternatively, you’re welcome to chat with us in the Nipype Gitter channel or in the BrainHack Slack channel. (Click here to join the Slack workspace.)

### 94.1.2 Overview

We’re using the Traits (formerly known as Enthought Traits) package for all of our inputs and outputs. Traits allows us to validate user inputs and provides a mechanism to handle all the *special cases* in a simple and concise way though metadata. With the metadata, each input/output can have an optional set of metadata attributes (described in more detail below). The machinery for handling the metadata is located in the base classes, so all subclasses use the same code to handle these cases. This is in contrast to our previous code where every class defined it’s own _parse_inputs, run and aggregate_outputs methods to handle these cases. Which of course leads to a dozen different ways to solve the same problem.

Traits is a big package with a lot to learn in order to take full advantage of. But don’t be intimidated! To write a Nipype Trait Specification, you only need to learn a few of the basics of Traits. Here are a few starting points in the documentation:

- **What are Traits?** The *Introduction in the User Manual* gives a brief description of the functionality traits provides.
- **Traits and metadata.** The *second section of the User Manual* gives more details on traits and how to use them. Plus there a section describing metadata, including the metadata all traits have.
- **If your interested in more of a *big picture* overview,** Gael wrote a good tutorial that shows how to write a scientific application using traits for the benefit of the generated UI components. (For now, Nipype is not taking...
advantage of the generated UI feature of traits.)

**Traits version**

We’re using Traits version 4.x which can be installed from pypi

**More documentation**

Not everything is documented in the User Manual, in those cases the the API docs is your next place to look.

### 94.1.3 Nipype Interface Specifications

Each interface class defines two specifications: 1) an InputSpec and 2) an OutputSpec. Each of these are prefixed with the class name of the interfaces. For example, Bet has these specs:

- **BETInputSpec**
- **BETOutputSpec**

Each of these Specs are classes, derived from a base TraitedSpec class (more on these below). The InputSpec consists of attributes which correspond to different parameters for the tool they wrap/interface. In the case of a command-line tool like Bet, the InputSpec attributes correspond to the different command-line parameters that can be passed to Bet. When an interfaces class is instantiated, the InputSpec is bound to the `inputs` attribute of that object. Below is an example of how the `inputs` appear to a user for Bet:

```python
>>> from nipype.interfaces import fsl
>>> bet = fsl.BET()
>>> type(bet.inputs)
<class 'nipype.interfaces.fsl.preprocess.BETInputSpec'>
>>> bet.inputs.<TAB>
bet.inputs._class__ bet.inputs.center
bet.inputs._delattr__ bet.inputs.environ
bet.inputs._doc__ bet.inputs.frac
bet.inputs._getattribute__ bet.inputs.functional
bet.inputs._hash__ bet.inputs.hashval
bet.inputs._init__ bet.inputs.infile
bet.inputs._new__ bet.inputs.items
bet.inputs._reduce__ bet.inputs.mask
bet.inputs._reduce_ex__ bet.inputs.mesh
bet.inputs._repr__ bet.inputs.nooutput
bet.inputs._setattr__ bet.inputs.outfile
bet.inputs._str__ bet.inputs.outline
bet.inputs._generate_handlers bet.inputs.outputtype
bet.inputs._get_hashval bet.inputs.radius
bet.inputs._hash_infile bet.inputs.reduce_bias
bet.inputs._xor_inputs bet.inputs.skull
bet.inputs._xor_warn bet.inputs.threshold
bet.inputs.args bet.inputs.vertical_gradient
```

Each Spec inherits from a parent Spec. The parent Specs provide attribute(s) that are common to all child classes. For example, FSL InputSpecs inherit from interfaces.fsl.base.FSLTraitedSpec. FSLTraitedSpec defines an `outputtype` attribute, which stores the file type (NIFTI, NIFTI_PAIR, etc...) for all generated output files.

**InputSpec class hierarchy**

Below is the current class hierarchy for InputSpec classes (from base class down to subclasses):

- **TraitedSpec**: Nipype’s primary base class for all Specs. Provides initialization, some nipype-specific methods and any trait handlers we define. Inherits from traits.HasTraits.
- **BaseInterfaceInputSpec**: Defines inputs common to all Interfaces (ignore_exception). If in doubt inherit from this.
CommandLineInputSpec: Defines inputs common to all command-line classes (args and environ)

FSLTreatedSpec: Defines inputs common to all FSL classes (outputtype)

SPMCommandInputSpec: Defines inputs common to all SPM classes (matlab_cmd, path, and mfile)

FSTreatedSpec: Defines inputs common to all FreeSurfer classes (subjects_dir)

MatlabInputSpec: Defines inputs common to all Matlab classes (script, nodeltop, nosplash, logfile, single_comp_thread, mfile, script_file, and paths)

SlicerCommandLineInputSpec: Defines inputs common to all Slicer classes (module)

Most developers will only need to code at the the interface-level (i.e. implementing custom class inheriting from one of the above classes).

Output Specs

The OutputSpec defines the outputs that are generated, or possibly generated depending on inputs, by the tool. OutputSpecs inherit from interfaces.base.TraitedSpec directly.

94.1.4 Controlling outputs to terminal

It is very likely that the software wrapped within the interface writes to the standard output or the standard error of the terminal. Interfaces provide a means to access and retrieve these outputs, by using the terminal_output attribute:

```python
import nipype.interfaces.fsl as fsl
mybet = fsl.BET(from_file='bet-settings.json')
mybet.terminal_output = 'file_split'
```

In the example, the terminal_output = 'file_split' will redirect the standard output and the standard error to split files (called stdout.nipype and stderr.nipype respectively). The possible values for terminal_output are:

- **file**: Redirects both standard output and standard error to the same file called output.nipype. Messages from both streams will be overlapped as they arrive to the file.
- **file_split**: Redirects the output streams separately, to stdout.nipype and stderr.nipype respectively, as described in the example.
- **file_stdout**: Only the standard output will be redirected to stdout.nipype and the standard error will be discarded.
- **file_stderr**: Only the standard error will be redirected to stderr.nipype and the standard output will be discarded.
- **stream**: Both output streams are redirected to the current logger printing their messages interleaved and immediately to the terminal.
- **allatonce**: Both output streams will be forwarded to a buffer and stored separately in the runtime object that the run() method returns. No files are written nor streams printed out to terminal.

- **none**: Both outputs are discarded

In all cases, except for the 'none' setting of terminal_output, the run() method will return a “runtime” object that will contain the streams in the corresponding properties (runtime.stdout for the standard output, runtime.stderr for the standard error, and runtime.merged for both when streams are mixed, eg. when using the file option).

```python
import nipype.interfaces.fsl as fsl
mybet = fsl.BET(from_file='bet-settings.json')
mybet.terminal_output = 'file_split'
```
94.1.5 Traited Attributes

Each specification attribute is an instance of a Trait class. These classes encapsulate many standard Python types like Float and Int, but with additional behavior like type checking. (See the documentation on traits for more information on these trait types.) To handle unique behaviors of our attributes we use traits metadata. These are keyword arguments supplied in the initialization of the attributes. The base classes BaseInterface and CommandLine (defined in nipype.interfaces.base) check for the existence/or value of these metadata and handle the inputs/outputs accordingly. For example, all mandatory parameters will have the mandatory = True metadata:

```python
class BetInputSpec(FSLTraitedSpec):
    infile = File(exists=True,
                  desc = 'input file to skull strip',
                  argstr='%s', position=0, mandatory=True)
```

Common

**exists** For files, use nipype.interfaces.base.File as the trait type. If the file must exist for the tool to execute, specify exists = True in the initialization of File (as shown in BetInputSpec above). This will trigger the underlying traits code to confirm the file assigned to that input actually exists. If it does not exist, the user will be presented with an error message:

```python
>>> bet.inputs.infile = 'does_not_exist.nii'
```

```
Traceback (most recent call last):
  File "<ipython console>", line 1, in <module>
  File "/Users/cburns/local/lib/python2.5/site-packages/nipype/interfaces/˓
  →base.py", line 76, in validate
    self.error( object, name, value )
  File "/Users/cburns/local/lib/python2.5/site-packages/enthought/traits/˓
  →trait_handlers.py", line 175, in error
    value)
TraitError: The 'infile' trait of a BetInputSpec instance must be a file name, but a value of 'does_not_exist.nii' <type 'str'> was specified.
```

**hash_files** To be used with inputs that are defining output filenames. When this flag is set to false any Nipype will not try to hash any files described by this input. This is useful to avoid rerunning when the specified output file already exists and has changed.

**desc** All trait objects have a set of default metadata attributes. desc is one of those and is used as a simple, one-line docstring. The desc is printed when users use the help() methods.

Required: This metadata is required by all nipype interface classes.

**usedefault** Set this metadata to True when the default value for the trait type of this attribute is an acceptable value. All trait objects have a default value, traits.Int has a default of 0, traits.Float has a default of 0.0, etc... You can also define a default value when you define the class. For example, in the code below all objects of Foo will have a default value of 12 for x:

```python
>>> import enthought.traits.api as traits
>>> class Foo(traits.HasTraits):
...     x = traits.Int(12)
```

(continues on next page)
Nipype only passes inputs on to the underlying package if they have been defined (more on this later). So if you specify usedefault = True, you are telling the parser to pass the default value on to the underlying package. Let’s look at the InputSpec for SPM Realign:

```python
... y = traits.Int
...
>>> foo = Foo()
>>> foo.x
12
>>> foo.y
0
```

Here we’ve defined jobtype to be an enumerated trait type, Enum, which can be set to one of the following: estwrite, estimate, or write. In a container, the default is always the first element. So in this case, the default will be estwrite:

```python
>>> from nipype.interfaces import spm
>>> rlgn = spm.Realign()
>>> rlgn.inputs.infile
<undefined>
>>> rlgn.inputs.jobtype
'estwrite'
```

**xor and requires** Both of these accept a list of trait names. The xor metadata reflects mutually exclusive traits, while the requires metadata reflects traits that have to be set together. When a xor-ed trait is set, all other traits belonging to the list are set to Undefined. The function check_mandatory_inputs ensures that all requirements (both mandatory and via the requires metadata are satisfied). These are also reflected in the help function.

**copyfile** This is metadata for a File or Directory trait that is relevant only in the context of wrapping an interface in a Node and MapNode. copyfile can be set to either True or False. False indicates that contents should be symlinked, while True indicates that the contents should be copied over.

**min_ver and max_ver** These metadata determine if a particular trait will be available when a given version of the underlying interface runs. Note that this check is performed at runtime.:

```python
class RealignInputSpec(BaseInterfaceInputSpec):
    jobtype = traits.Enum('estwrite', 'estimate', 'write', min_ver='5',
                          usedefault=True)
```

**deprecated and new_name** This is metadata for removing or renaming an input field from a spec.:

```python
class RealignInputSpec(BaseInterfaceInputSpec):
    jobtype = traits.Enum('estwrite', 'estimate', 'write',
                          deprecated='0.8',
                          desc='one of: estimate, write, estwrite',
                          usedefault=True)
```

In the above example this means that the jobtype input is deprecated and will be removed in version 0.8. Deprecation should be set to two versions from current release. Raises TraitError after package version crosses the deprecation version.

For inputs that are being renamed, one can specify the new name of the field.:
deprecated='0.8', new_name='job_type',
desc='one of: estimate, write, estwrite',
usedefault=True)
job_type = traits.Enum('estwrite', 'estimate', 'write',
desc='one of: estimate, write, estwrite',
usedefault=True)

In the above example, the jobtype field is being renamed to job_type. When new_name is provided it must exist as a trait, otherwise an exception will be raised.

Note: The version information for min_ver, max_ver and deprecated has to be provided as a string. For example, min_ver='0.1'.

CommandLine

argstr The metadata keyword for specifying the format strings for the parameters. This was the value string in the opt_map dictionaries of Nipype 0.2 code. If we look at the FlirtInputSpec, the argstr for the reference file corresponds to the argument string I would need to provide with the command-line version of flirt:

class FlirtInputSpec(FSLTraitedSpec):
    reference = File(exists = True, argstr = '-ref $s', mandatory = True, position = 1, desc = 'reference file')

Required: This metadata is required by all command-line interface classes.

position This metadata is used to specify the position of arguments. Both positive and negative values are accepted. position = 0 will position this argument as the first parameter after the command name. position = -1 will position this argument as the last parameter, after all other parameters.

genfile If True, the genfile metadata specifies that a filename should be generated for this parameter if-and-only-if the user did not provide one. The nipype convention is to automatically generate output filenames when not specified by the user both as a convenience for the user and so the pipeline can easily gather the outputs. Requires _gen_filename() method to be implemented. This way should be used if the desired file name is dependent on some runtime variables (such as file name of one of the inputs, or current working directory). In case when it should be fixed it's recommended to just use usedefault.

sep For List traits the string with which elements of the list will be joined.

name_source Indicates the list of input fields from which the value of the current File output variable will be drawn. This input field must be the name of a File. Chaining is allowed, meaning that an input field can point to another as name_source, which also points as name_source to a third field. In this situation, the templates for substitutions are also accumulated.

name_template By default a %s_generated template is used to create the output filename. This metadata keyword allows overriding the generated name.

keep_extension Use this and set it True if you want the extension from the input to be kept.

SPM

field name of the structure refered by the SPM job manager

Required: This metadata is required by all SPM-mediated interface classes.

94.1.6 Defining an interface class

Common

When you define an interface class, you will define these attributes and methods:

• input_spec: the InputSpec
• output_spec: the OutputSpec
• `_list_outputs()`: Returns a dictionary containing names of generated files that are expected after package completes execution. This is used by `BaseInterface.aggregate_outputs` to gather all output files for the pipeline.

**CommandLine**

For command-line interfaces:
• `_cmd`: the command-line command
  If you used genfile:
• `_gen_filename(name)`: Generate filename, used for filenames that nipype generates as a convenience for users. This is for parameters that are required by the wrapped package, but we’re generating from some other parameter. For example, `BET.inputs.outfile` is required by BET but we can generate the name from `BET.inputs.infile`. Override this method in subclass to handle.
  And optionally:
• `_redirect_x`: If set to True it will make Nipype start Xvfb before running the interface and redirect X output to it. This is useful for commandlines that spawn a graphical user interface.
• `_format_arg(name, spec, value)`: For extra formatting of the input values before passing them to `generic_parse_inputs()` method.

For example this is the class definition for Flirt, minus the docstring:

```python
class FLIRTInputSpec(FSLCommandInputSpec):
in_file = File(exists=True, argstr='in $s', mandatory=True, position=0, desc='input file')
reference = File(exists=True, argstr='ref $s', mandatory=True, position=1, desc='reference file')
out_file = File(argstr='out $s', desc='registered output file', name_source=['in_file'], name_template='$_flirt', position=2, hash_files=False)
out_matrix_file = File(argstr='omat $s',
                      name_source=['in_file'], keep_extension=True,
                      name_template='$_flirt.mat',
                      desc='output affine matrix in 4x4 ascii format',
                      position=3, hash_files=False)
out_log = File(name_source=['in_file'], keep_extension=True,
               requires=['save_log'],
               name_template='$_flirt.log', desc='output log')
...

class FLIRTOutputSpec(TraitedSpec):
out_file = File(exists=True,
               desc='path/name of registered file (if generated)')
out_matrix_file = File(exists=True,
                      desc='path/name of calculated affine transform '
                      '(if generated)')
out_log = File(desc='path/name of output log (if generated)')

class Flirt(FSLCommand):
  _cmd = 'flirt'
  input_spec = FlirtInputSpec
  output_spec = FlirtOutputSpec
```

There are two possible output files `outfile` and `outmatrix`, both of which can be generated if not specified by the user.
Also notice the use of `self._gen_fname()` - a FSLCommand helper method for generating filenames (with extensions conforming with FSLOUTPUTTYPE).
See also *How to wrap a command line tool*. 

**94.1. Interface Specifications**
SPM

For SPM-mediated interfaces:

- `_jobtype` and `_jobname`: special names used used by the SPM job manager. You can find them by saving your batch job as an .m file and looking up the code.

And optionally:

- `_format_arg(name, spec, value)`: For extra formatting of the input values before passing them to generic `_parse_inputs()` method.

Matlab

See *How to wrap a MATLAB script*.

Python

See *How to wrap a Python script*.

94.1.7 Undefined inputs

All the inputs and outputs that were not explicitly set (And do not have a usedefault flag - see above) will have Undefined value. To check if something is defined you have to explicitly call `isdefined` function (comparing to None will not work).

94.1.8 Example of inputs

Below we have an example of using Bet. We can see from the help which inputs are mandatory and which are optional, along with the one-line description provided by the `desc` metadata:

```python
>>> from nipype.interfaces import fsl
>>> fsl.BET.help()

Inputs
------

Mandatory:
- `infile`: input file to skull strip

Optional:
- `args`: Additional parameters to the command
center: center of gravity in voxels
environ: Environment variables (default={})
frac: fractional intensity threshold
functional: apply to 4D fMRI data
mask: create binary mask image
mesh: generate a vtk mesh brain surface
nooutput: Don't generate segmented output
outfile: name of output skull stripped image
outline: create surface outline image
outputtype: None
radius: head radius
reduce_bias: bias field and neck cleanup
skull: create skull image
threshold: apply thresholding to segmented brain image and mask
vertical_gradient: vertical gradient in fractional intensity threshold (-1, 1)

Outputs
-------

maskfile: path/name of binary brain mask (if generated)
```

(continues on next page)
Here we create a bet object and specify the required input. We then check our inputs to see which are defined and which are not:

```python
>>> bet = fsl.BET(infile = 'f3.nii')
```

```
bet.inputs
```

```
args = <undefined>
center = <undefined>
environ = {'FSLOUTPUTTYPE': 'NIFTI_GZ'}
frac = <undefined>
functional = <undefined>
infile = f3.nii
mask = <undefined>
mesh = <undefined>
noutput = <undefined>
outfile = <undefined>
outline = <undefined>
outputtype = NIFTI_GZ
radius = <undefined>
reduce_bias = <undefined>
skull = <undefined>
threshold = <undefined>
vertical_gradient = <undefined>
```  

```python
>>> bet.cmdline
'bet f3.nii /Users/cburns/data/nipype/s1/f3_brain.nii.gz'
```

We also checked the command-line that will be generated when we run the command and can see the generated output filename `f3_brain.nii.gz`.

### 94.2 How to wrap a command line tool

The aim of this section is to describe how external programs and scripts can be wrapped for use in Nipype either as interactive interfaces or within the workflow/pipeline environment. Currently, there is support for command line executables/scripts and matlab scripts. One can also create pure Python interfaces. The key to defining interfaces is to provide a formal specification of inputs and outputs and determining what outputs are generated given a set of inputs.

#### 94.2.1 Defining inputs and outputs

In Nipype we use Enthought Traits to define inputs and outputs of the interfaces. This allows to introduce easy type checking. Inputs and outputs are grouped into separate classes (usually suffixed with InputSpec and OutputSpec). For example:

```python
class ExampleInputSpec(TraitedSpec):
    input_volume = File(desc = "Input volume", exists = True,
                        mandatory = True)
    parameter = traits.Int(desc = "some parameter")

class ExampleOutputSpec(TraitedSpec):
    output_volume = File(desc = "Output volume", exists = True)
```

For the Traits (and Nipype) to work correctly output and input spec has to be inherited from TraitedSpec (however, this does not have to be direct inheritance).
Traits (File, Int etc.) have different parameters (called metadata). In the above example we have used the `desc` metadata which holds human readable description of the input. The `mandatory` flag forces Nipype to throw an exception if the input was not set. `exists` is a special flag that works only for `File` traits and checks if the provided file exists. More details can be found at Interface Specifications.

The input and output specifications have to be connected to the our example interface class:

```python
class Example(Interface):
    input_spec = ExampleInputSpec
    output_spec = ExampleOutputSpec
```

Where the names of the classes grouping inputs and outputs were arbitrary the names of the fields within the interface they are assigned are not (it always has to be input_spec and output_spec). Of course this interface does not do much because we have not specified how to process the inputs and create the outputs. This can be done in many ways.

### 94.2.2 Command line executable

As with all interfaces command line wrappers need to have inputs defined. Command line input spec has to inherit from `CommandLineInputSpec` which adds two extra inputs: `environ` (a dictionary of environmental variables), and `args` (a string defining extra flags). In addition input spec can define the relation between the inputs and the generated command line. To achieve this we have added two metadata: `argstr` (string defining how the argument should be formatted) and `position` (number defining the order of the arguments). For example

```python
class ExampleInputSpec(CommandLineSpec):
    input_volume = File(desc = "Input volume", exists = True,
                        mandatory = True, position = 0, argstr="%s")
    parameter = traits.Int(desc = "some parameter", argstr = "--param %d")
```

As you probably noticed the `argstr` is a printf type string with formatting symbols. For an input defined in `InputSpec` to be included into the executed commandline `argstr` has to be included. Additionally inside the main interface class you need to specify the name of the executable by assigning it to the `_cmd` field. Also the main interface class needs to inherit from `CommandLine`:

```python
class Example(CommandLine):
    _cmd = 'my_command'
    input_spec = ExampleInputSpec
    output_spec = ExampleOutputSpec
```

There is one more thing we need to take care of. When the executable finishes processing it will presumably create some output files. We need to know which files to look for, check if they exist and expose them to whatever node would like to use them. This is done by implementing `_list_outputs` method in the main interface class. Basically what it does is assigning the expected output files to the fields of our output spec:

```python
def _list_outputs(self):
    outputs = self.output_spec().get()
    outputs['output_volume'] = os.path.abspath('name_of_the_file_this_cmd_made.nii')
    return outputs
```

Sometimes the inputs need extra parsing before turning into command line parameters. For example imagine a parameter selecting between three methods: "old", "standard" and "new". Imagine also that the command line accept this as a parameter "--method=" accepting 0, 1 or 2. Since we are aiming to make nipype scripts as informative as possible it’s better to define the inputs as following:

```python
class ExampleInputSpec(CommandLineSpec):
    method = traits.Enum("old", "standard", "new", desc = "method",
                        argstr="--method=%d")
```
nipype Documentation, Release 1.2.1

Here we’ve used the Enum trait which restricts input a few fixed options. If we would leave it as it is it would
not work since the argstr is expecting numbers. We need to do additional parsing by overloading the following
method in the main interface class:
def _format_arg(self, name, spec, value):
if name == 'method':
return spec.argstr%{"old":0, "standard":1, "new":2}[value]
return super(Example, self)._format_arg(name, spec, value)

Here is a minimalistic interface for the gzip command:
from nipype.interfaces.base import (
TraitedSpec,
CommandLineInputSpec,
CommandLine,
File
)
import os
class GZipInputSpec(CommandLineInputSpec):
input_file = File(desc="File", exists=True, mandatory=True, argstr="%s")
class GZipOutputSpec(TraitedSpec):
output_file = File(desc = "Zip file", exists = True)
class GZipTask(CommandLine):
input_spec = GZipInputSpec
output_spec = GZipOutputSpec
cmd = 'gzip'
def _list_outputs(self):
outputs = self.output_spec().get()
outputs['output_file'] = os.path.abspath(self.inputs.input_file + ".gz
")

˓→

return outputs
if __name__ == '__main__':
zipper = GZipTask(input_file='an_existing_file')
print zipper.cmdline
zipper.run()

94.2.3 Creating outputs on the fly
In many cases, command line executables will require specifying output file names as arguments on the
command line. We have simplified this procedure with three additional metadata terms: name_source,
name_template, keep_extension.
For example in the InvWarp class, the inverse_warp parameter is the name of the output file that is created
by the routine.
class InvWarpInputSpec(FSLCommandInputSpec):
...
inverse_warp = File(argstr='--out=%s', name_source=['warp'],
hash_files=False, name_template='%s_inverse',
...

we add several metadata to inputspec.
name_source indicates which field to draw from, this field must be the name of a File.

94.2. How to wrap a command line tool

1641


hash_files indicates that the input for this field if provided should not be used in computing the input hash for this interface.

name_template (optional) overrides the default _generated suffix

output_name (optional) name of the output (if this is not set same name as the input will be assumed)

keep_extension (optional) if you want the extension from the input or name_template to be kept. The name_template extension always overrides the input extension.

In addition one can add functionality to your class or base class, to allow changing extensions specific to package or interface. This overload function is trigged only if keep_extension is not defined.

```python
def self._overload_extension(self, value):
    return value  # do whatever you want here with the name
```

Finally, in the outputspec make sure the name matches that of the inputspec.

```python
class InvWarpOutputSpec(TraitedSpec):
    inverse_warp = File(exists=True,
            desc=('Name of output file, containing warps that ' +
            'are the "reverse" of those in --warp.'))
```

### 94.3 How to wrap a MATLAB script

#### 94.3.1 Example 1

This is a minimal script for wrapping MATLAB code. You should replace the MATLAB code template, and define appropriate inputs and outputs.

```python
from nipype.interfaces.matlab import MatlabCommand
from nipype.interfaces.base import TraitedSpec, BaseInterface, BaseInterfaceInputSpec, File
import os
from string import Template

class ConmapTxt2MatInputSpec(BaseInterfaceInputSpec):
    in_file = File(exists=True, mandatory=True)
    out_file = File('cmatrix.mat', usedefault=True)

class ConmapTxt2MatOutputSpec(TraitedSpec):
    out_file = File(exists=True)

class ConmapTxt2Mat(BaseInterface):
    input_spec = ConmapTxt2MatInputSpec
    output_spec = ConmapTxt2MatOutputSpec

    def _run_interface(self, runtime):
        d = dict(in_file=self.inputs.in_file,
                  out_file=self.inputs.out_file)
        # This is your MATLAB code template
        script = Template("""in_file = '$in_file';
        out_file = '$out_file';
        ConmapTxt2Mat(in_file, out_file);
        exit;
        """).substitute(d)
```

(continues on next page)
nipype Documentation, Release 1.2.1

(continued from previous page)

# mfile = True will create an .m file with your script and executed.
# Alternatively
# mfile can be set to False which will cause the matlab code to be
# passed
# as a commandline argument to the matlab executable
# (without creating any files).
# This, however, is less reliable and harder to debug
# (code will be reduced to
# a single line and stripped of any comments).
mlab = MatlabCommand(script=script, mfile=True)
result = mlab.run()
return result.runtime
def _list_outputs(self):
outputs = self._outputs().get()
outputs['out_file'] = os.path.abspath(self.inputs.out_file)
return outputs

Example source code
You can download the source code of this example.

94.3.2 Example 2
By subclassing nipype.interfaces.matlab.MatlabCommand for your main class, and nipype.
interfaces.matlab.MatlabInputSpec for your input and output spec, you gain access to some useful
MATLAB hooks
from nipype.interfaces.base import traits
from nipype.interfaces.base import TraitedSpec
from nipype.interfaces.matlab import MatlabCommand, MatlabInputSpec

class HelloWorldInputSpec(MatlabInputSpec):
name = traits.Str(mandatory=True,
desc='Name of person to say hello to')

class HelloWorldOutputSpec(TraitedSpec):
matlab_output = traits.Str()

class HelloWorld(MatlabCommand):
"""Basic Hello World that displays Hello <name> in MATLAB
Returns
------matlab_output : capture of matlab output which may be
parsed by user to get computation results
Examples
------->>> hello = HelloWorld()
(continues on next page)

94.3. How to wrap a MATLAB script

1643


>>> hello.inputs.name = 'hello_world'
>>> out = hello.run()
>>> print out.outputs.matlab_output

input_spec = HelloWorldInputSpec
output_spec = HelloWorldOutputSpec

def _my_script(self):
    """This is where you implement your script""
    script = ""
    disp('Hello %s Python')
    two = 1 + 1
    """ % (self.inputs.name)
    return script

def run(self, **inputs):
    # inject your script
    self.inputs.script = self._my_script()
    results = super(MatlabCommand, self).run(**inputs)
    stdout = results.runtime.stdout
    # attach stdout to outputs to access matlab results
    results.outputs.matlab_output = stdout
    return results

def _list_outputs(self):
    outputs = self._outputs().get()
    return outputs

Example source code
You can download the source code of this example.

94.4 How to wrap a Python script

This is a minimal pure python interface. As you can see all you need to do is to do is to define inputs, outputs, _run_interface() (not run()), and _list_outputs.

from nipype.interfaces.base import BaseInterface, 
    BaseInterfaceInputSpec, traits, File, TraitedSpec
from nipype.utils.filemanip import split_filename

import nibabel as nb
import numpy as np
import os

class SimpleThresholdInputSpec(BaseInterfaceInputSpec):
    volume = File(exists=True, desc='volume to be thresholded', mandatory=True)
    threshold = traits.Float(desc='everything below this value will be set to zero',
                              mandatory=True)

class SimpleThresholdOutputSpec(TraitedSpec):
    thresholded_volume = File(exists=True, desc="thresholded volume")
class SimpleThreshold(BaseInterface):
    input_spec = SimpleThresholdInputSpec
    output_spec = SimpleThresholdOutputSpec

    def _run_interface(self, runtime):
        fname = self.inputs.volume
        img = nb.load(fname)
        data = np.array(img.get_data())

        active_map = data > self.inputs.threshold
        thresholded_map = np.zeros(data.shape)
        thresholded_map[active_map] = data[active_map]

        new_img = nb.Nifti1Image(thresholded_map, img.affine, img.header)
        _, base, _ = split_filename(fname)
        nb.save(new_img, base + '_thresholded.nii')

        return runtime

    def _list_outputs(self):
        outputs = self._outputs().get()
        fname = self.inputs.volume
        _, base, _ = split_filename(fname)
        outputs['thresholded_volume'] = os.path.abspath(base + '_thresholded.nii')

        return outputs

94.5 Working with nipype source code

Contents:

94.5.1 Introduction

These pages describe a git and github workflow for the nipype project. There are several different workflows here, for different ways of working with nipype. This is not a comprehensive git reference, it’s just a workflow for our own project. It’s tailored to the github hosting service. You may well find better or quicker ways of getting stuff done with git, but these should get you started. For general resources for learning git see git resources.

94.5.2 Install git

Overview

<table>
<thead>
<tr>
<th>Platform</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debian / Ubuntu</td>
<td>sudo apt-get install git-core</td>
</tr>
<tr>
<td>Fedora</td>
<td>sudo yum install git-core</td>
</tr>
<tr>
<td>Windows</td>
<td>Download and install msysGit</td>
</tr>
<tr>
<td>OS X</td>
<td>Use the git-osx-installer</td>
</tr>
</tbody>
</table>

In detail

See the git page for the most recent information. Have a look at the github install help pages available from github help
There are good instructions here: http://book.git-scm.com/2_installing_git.html

94.5.3 Following the latest source

These are the instructions if you just want to follow the latest *nipype* source, but you don’t need to do any development for now.

The steps are:

- Install git
- get local copy of the git repository from github
- update local copy from time to time

**Get the local copy of the code**

From the command line:

```bash
git clone git://github.com/nipy/nipype.git
```

You now have a copy of the code tree in the new *nipype* directory.

**Updating the code**

From time to time you may want to pull down the latest code. Do this with:

```
cd nipype
git pull
```

The tree in *nipype* will now have the latest changes from the initial repository.

94.5.4 Making a patch

You’ve discovered a bug or something else you want to change in *nipype* .. — excellent!
You’ve worked out a way to fix it — even better!
You want to tell us about it — best of all!
The easiest way is to make a patch or set of patches. Here we explain how. Making a patch is the simplest and quickest, but if you’re going to be doing anything more than simple quick things, please consider following the *Git for development* model instead.

**Making patches**

**Overview**

```bash
# tell git who you are
git config --global user.email you@yourdomain.example.com
git config --global user.name "Your Name Comes Here"
# get the repository if you don't have it
git clone git://github.com/nipy/nipype.git
# make a branch for your patching
cd nipype
git branch the-fix-im-thinking-of
git checkout the-fix-im-thinking-of
# hack, hack, hack
# Tell git about any new files you've made
git add somewhere/tests/test_my_bug.py
# commit work in progress as you go
git commit -am 'BF - added tests for Funny bug'
# hack hack, hack

(continues on next page)
# make the patch files

git format-patch -M -C master

Then, send the generated patch files to the nipype mailing list — where we will thank you warmly.

In detail

1. Tell git who you are so it can label the commits you’ve made:

   ```
git config --global user.email you@yourdomain.example.com
git config --global user.name "Your Name Comes Here"
   ```

2. If you don’t already have one, clone a copy of the nipype repository:

   ```
git clone git://github.com/nipy/nipype.git
cd nipype
   ```

3. Make a ‘feature branch’. This will be where you work on your bug fix. It’s nice and safe and leaves you with access to an unmodified copy of the code in the main branch:

   ```
git branch the-fix-im-thinking-of

git checkout the-fix-im-thinking-of
   ```

4. Do some edits, and commit them as you go:

   ```
# hack, hack, hack
# Tell git about any new files you've made
git add somewhere/tests/test_my_bug.py
# commit work in progress as you go
git commit -am 'BF - added tests for Funny bug'
# hack hack, hack

git commit -am 'BF - added fix for Funny bug'
   ```

   Note the -am options to commit. The m flag just signals that you’re going to type a message on the command line. The a flag — you can just take on faith — or see why the -a flag?

5. When you have finished, check you have committed all your changes:

   ```
git status
   ```

6. Finally, make your commits into patches. You want all the commits since you branched from the master branch:

   ```
git format-patch -M -C master
   ```

You will now have several files named for the commits:

```
0001-BF-added-tests-for-Funny-bug.patch
0002-BF-added-fix-for-Funny-bug.patch
   ```

Send these files to the nipype mailing list.

When you are done, to switch back to the main copy of the code, just return to the master branch:

```
git checkout master
   ```

Moving from patching to development

If you find you have done some patches, and you have one or more feature branches, you will probably want to switch to development mode. You can do this with the repository you have.

Fork the nipype repository on github — *Making your own copy (fork) of nipype*. Then:
Then you can, if you want, follow the Development workflow.

## 94.5.5 Git for development

### Contents:

**Making your own copy (fork) of nipype**

You need to do this only once. The instructions here are very similar to the instructions at [http://help.github.com/forking/](http://help.github.com/forking/) — please see that page for more detail. We’re repeating some of it here just to give the specifics for the nipype project, and to suggest some default names.

### Set up and configure a github account

If you don’t have a github account, go to the github page, and make one. You then need to configure your account to allow write access — see the Generating SSH keys help on github help.

### Create your own forked copy of nipype

1. Log into your github account.
2. Go to the nipype github home at nipype github.
3. Click on the fork button:

   ![Fork button](image)

Now, after a short pause and some ‘Hardcore forking action’, you should find yourself at the home page for your own forked copy of nipype.

### Set up your fork

First you follow the instructions for Making your own copy (fork) of nipype.

### Overview

```
git clone git@github.com:your-user-name/nipype.git
cd nipype
git remote add upstream git://github.com/nipy/nipype.git
```
In detail

Clone your fork

1. Clone your fork to the local computer with `git clone git@github.com:your-user-name/nipype.git`
2. Investigate. Change directory to your new repo: `cd nipype`. Then `git branch -a` to show you all branches. You’ll get something like:

```
* master
remotes/origin/master
```

This tells you that you are currently on the `master` branch, and that you also have a remote connection to `origin/master`. What remote repository is `remote/origin`? Try `git remote -v` to see the URLs for the remote. They will point to your `github` fork.

Now you want to connect to the upstream `nipype github` repository, so you can merge in changes from trunk.

Linking your repository to the upstream repo

```
cd nipype
git remote add upstream git://github.com/nipy/nipype.git
```

`upstream` here is just the arbitrary name we’re using to refer to the main `nipype` repository at `nipype github`. Note that we’ve used `git://` for the URL rather than `git@`. The `git://` URL is read only. This means we that we can’t accidentally (or deliberately) write to the upstream repo, and we are only going to use it to merge into our own code.

Just for your own satisfaction, show yourself that you now have a new ‘remote’, with `git remote -v show`, giving you something like:

```
upstream git://github.com/nipy/nipype.git (fetch)
upstream git://github.com/nipy/nipype.git (push)
origin git@github.com:your-user-name/nipype.git (fetch)
origin git@github.com:your-user-name/nipype.git (push)
```

Configure git

Overview

Your personal `git` configurations are saved in the `.gitconfig` file in your home directory. Here is an example `.gitconfig` file:

```
[user]
    name = Your Name
    email = you@yourdomain.example.com

[alias]
    ci = commit -a
    co = checkout
    st = status -a
    stat = status -a
    br = branch
    wdiff = diff --color-words

[core]
    editor = vim
```

(continues on next page)
You can edit this file directly or you can use the `git config --global` command:

```
[merge]
    summary = true
```

```
[global]
    user.name = "Your Name"
    user.email = you@yourdomain.example.com
    alias.ci = "commit -a"
    alias.co = checkout
    alias.st = "status -a"
    alias.stat = "status -a"
    alias.br = branch
    alias.wdiff = "diff --color-words"
    core.editor = vim
    merge.summary = true
```

To set up on another computer, you can copy your `~/.gitconfig` file, or run the commands above.

**In detail**

**user.name and user.email**

It is good practice to tell `git` who you are, for labeling any changes you make to the code. The simplest way to do this is from the command line:

```
[global]
    user.name = "Your Name"
    user.email = you@yourdomain.example.com
```

This will write the settings into your git configuration file, which should now contain a user section with your name and email:

```
[user]
    name = Your Name
    email = you@yourdomain.example.com
```

Of course you'll need to replace `Your Name` and `you@yourdomain.example.com` with your actual name and email address.

**Aliases**

You might well benefit from some aliases to common commands. For example, you might well want to be able to shorten `git checkout` to `git co`. Or you may want to alias `git diff --color-words` (which gives a nicely formatted output of the diff) to `git wdiff`.

The following `git config --global` commands:

```
[global]
    alias.ci = "commit -a"
    alias.co = checkout
    alias.st = "status -a"
    alias.stat = "status -a"
    alias.br = branch
    alias.wdiff = "diff --color-words"
```

will create an alias section in your `.gitconfig` file with contents like this:

```
[alias]
    ci = commit -a
    co = checkout
```

(continues on next page)
Editor

You may also want to make sure that your editor of choice is used

```
git config --global core.editor vim
```

Merging

To enforce summaries when doing merges (~/.gitconfig file again):

```
[merge]
  log = true
```

Or from the command line:

```
git config --global merge.log true
```

Development workflow

You already have your own forked copy of the nipype repository, by following *Making your own copy (fork) of nipype, Set up your fork*, and you have configured git by following *Configure git*.

Workflow summary

- Keep your master branch clean of edits that have not been merged to the main nipype development repo. Your master then will follow the main nipype repository.
- Start a new feature branch for each set of edits that you do.
- If you can avoid it, try not to merge other branches into your feature branch while you are working.
- Ask for review!
  
  This way of working really helps to keep work well organized, and in keeping history as clear as possible.
  
  See — for example — linux git workflow.

Making a new feature branch

```
git branch my-new-feature

git checkout my-new-feature
```

Generally, you will want to keep this also on your public github fork of nipype. To do this, you *git push* this new branch up to your github repo. Generally (if you followed the instructions in these pages, and by default), git will have a link to your github repo, called origin. You push up to your own repo on github with:

```
git push origin my-new-feature
```

In git >1.7 you can ensure that the link is correctly set by using the *--set-upstream* option:

```
git push --set-upstream origin my-new-feature
```

From now on *git* will know that *my-new-feature* is related to the *my-new-feature* branch in the github repo.
The editing workflow

Overview

```
# hack hack
git add my_new_file
git commit -am 'NF - some message'
git push
```

In more detail

1. Make some changes
2. See which files have changed with `git status` (see `git status`). You’ll see a listing like this one:

```
# On branch ny-new-feature
# Changed but not updated:
# (use "git add <file>..." to update what will be committed)
# (use "git checkout -- <file>..." to discard changes in working directory)
#
# modified: README
#
# Untracked files:
# (use "git add <file>..." to include in what will be committed)
#
# INSTALL
no changes added to commit (use "git add" and/or "git commit -a")
```
3. Check what the actual changes are with `git diff` (see `git diff`).
4. Add any new files to version control `git add new_file_name` (see `git add`).
5. To commit all modified files into the local copy of your repo, do `git commit -am 'A commit message'`. Note the `-am` options to commit. The `m` flag just signals that you’re going to type a message on the command line. The `a` flag — you can just take on faith — or see why the `-a` flag? — and the helpful use-case description in the tangled working copy problem. The `git commit` manual page might also be useful.
6. To push the changes up to your forked repo on github, do a `git push` (see `git push`).

Asking for code review

1. Go to your repo URL — e.g. http://github.com/your-user-name/nipype.
2. Click on the `Branch list` button:

3. Click on the `Compare` button for your feature branch — here `my-new-feature`:

4. If asked, select the `base` and `comparison` branch names you want to compare. Usually these will be `master` and `my-new-feature` (where that is your feature branch name).
5. At this point you should get a nice summary of the changes. Copy the URL for this, and post it to the nipype mailing list, asking for review. The URL will look something like: http://github.com/your-user-name/nipype/compare/master...my-new-feature. There’s an example at http://github.com/matthew-brett/nipy/compare/master...find-install-data See: http://github.com/blog/612-introducing-github-compare-view for more detail.

The generated comparison, is between your feature branch my-new-feature, and the place in master from which you branched my-new-feature. In other words, you can keep updating master without interfering with the output from the comparison. More detail? Note the three dots in the URL above (master...my-new-feature).

Two vs three dots
Imagine a series of commits A, B, C, D... Imagine that there are two branches, topic and master. You branched topic off master when master was at commit ‘E’. The graph of the commits looks like this:

```
A---B---C topic
   /
D---E---F---G master
```

Then:

```
git diff master..topic
```

will output the difference from G to C (i.e. with effects of F and G), while:

```
git diff master...topic
```

would output just differences in the topic branch (i.e. only A, B, and C).¹

Asking for your changes to be merged with the main repo

When you are ready to ask for the merge of your code:

1. Go to the URL of your forked repo, say http://github.com/your-user-name/nipype.git.
2. Click on the ‘Pull request’ button:

```
Admin Unwatch Pull Request Downl
```

Enter a message; we suggest you select only nipype as the recipient. The message will go to the nipype mailing list. Please feel free to add others from the list as you like.

Merging from trunk

This updates your code from the upstream nipype github repo.

Overview

```
# go to your master branch
git checkout master
# pull changes from github
```

¹ Thanks to Yarik Halchenko for this explanation.
In detail

We suggest that you do this only for your `master` branch, and leave your ‘feature’ branches unmerged, to keep their history as clean as possible. This makes code review easier:

```bash
git checkout master
```

Make sure you have done *Linking your repository to the upstream repo.*

Merge the upstream code into your current development by first pulling the upstream repo to a copy on your local machine:

```bash
git fetch upstream
```

then merging into your current branch:

```bash
git merge upstream/master
```

Deleting a branch on github

```bash
git checkout master
# delete branch locally
git branch --delete my-unwanted-branch
# delete branch on github
git push origin :my-unwanted-branch
```

(Nota the colon `:` before `test-branch`. See also: http://github.com/guides/remove-a-remote-branch)

Several people sharing a single repository

If you want to work on some stuff with other people, where you are all committing into the same repository, or even the same branch, then just share it via `github`.

First fork nipype into your account, as from *Making your own copy (fork) of nipype.*

Then, go to your forked repository github page, say http://github.com/your-user-name/nipype

Click on the ‘Admin’ button, and add anyone else to the repo as a collaborator:

Now all those people can do:

```bash
git clone git@github.com:your-user-name/nipype.git
```

Remember that links starting with `git@` use the ssh protocol and are read-write; links starting with `git://` are read-only.

Your collaborators can then commit directly into that repo with the usual:
git commit -am 'ENH - much better code'
git push origin master # pushes directly into your repo

Exploring your repository

To see a graphical representation of the repository branches and commits:

gitk --all

To see a linear list of commits for this branch:

git log

You can also look at the network graph visualizer for your github repo.

94.5.6 git resources

Tutorials and summaries

- [github help](https://github.com) has an excellent series of how-to guides.
- [learn.github](https://learn.github.com) has an excellent series of tutorials
- The [pro git book](https://git-scm.com/book) is a good in-depth book on git.
- A [git cheat sheet](https://git-scm.com/docs/cheatsheet) is a page giving summaries of common commands.
- The [git user manual](https://git-scm.com/docs/user)
- The [git tutorial](https://git-scm.com/docs/tutorial)
- The [git community book](https://git-scm.com/docs/community)
- [git ready](https://git-scm.com/docs/ready) — a nice series of tutorials
- [git casts](https://git-scm.com/docs/casts) — video snippets giving git how-tos.
- [git magic](https://git-scm.com/docs/magic) — extended introduction with intermediate detail
- The [git parable](https://git-scm.com/docs/parable) is an easy read explaining the concepts behind git.
- [Our own git foundation](https://git-scm.com/docs/foundation) expands on the git parable.
- [Fernando Perez’ git page](https://git-scm.com/docs/perez) — Fernando’s git page — many links and tips
- A good but technical page on git concepts
- git svn crash course: git for those of us used to subversion

Advanced git workflow

There are many ways of working with git; here are some posts on the rules of thumb that other projects have come up with:
- Linus Torvalds on [git management](https://git-scm.com/docs/torvalds)
- Linus Torvalds on [linux git workflow](https://git-scm.com/docs/torvalds/torvalds). Summary: use the git tools to make the history of your edits as clean as possible; merge from upstream edits as little as possible in branches where you are doing active development.

Manual pages online

You can get these on your own machine with (e.g) `git help push` or (same thing) `git push --help`, but, for convenience, here are the online manual pages for some common commands:
- git add
- git branch
- git checkout
- git clone
- git commit
- git config
- git diff
- git log
- git pull

94.5. Working with nipype source code
94.6 Architecture (discussions from 2009)

This section reflects notes and discussion between developers during the start of the nipype project in 2009.

94.6.1 Design Guidelines

These are guidelines that the core nipype developers have agreed on:

Interfaces should keep all parameters affecting construction of the appropriate command in the “input” bunch.
The .run() method of an Interface should include all required inputs as explicitly named parameters, and they should take a default value of None.

Any Interface should at a minimum support cwd as a command-line argument to .run(). This may be accomplished by allowing cwd as an element of the input Bunch, or handled as a separate case.

Relatedly, any Interface should output all files to cwd if it is set, and otherwise to os.getcwd() (or equivalent).

We need to decide on a consistent policy towards the maintinence of paths to files. It seems like the best strategy might be to do absolute (os.realpath?) filenames by default, allowing for relative paths by explicitly including something that doesn’t start with a ‘/’. This could include ‘.’ in some sort of path-spec.

Class attributes should never be modified by an instance of that class. And probably not ever.

94.6.2 Providing for Provenance

The following is a specific discussion that should be thought out an more generally applied to the way we handle auto-generation / or “sourcing” of settings in an interface.

There are two possible sources (at a minimum) from which the interface instance could obtain “outputtype” - itself, or FSLInfo. Currently, the outputtype gets read from FSLInfo if self.outputtype (er, _outputtype?) is None.

In the case of other opt_map specifications, there are defaults that get specified if the value is None. For example output filenames are often auto-generated. If you look at the code for fsl.Bet for example, there is no way for the outfile to get picked up at the pipeline level, because it is a transient variable. This is OK, as the generation of the outfile name is contingent ONLY on inputs which ARE available to the pipeline machinery (i.e., via inspection of the Bet instance’s attributes).

However, with outputtype, we are in a situation in which “autogeneration” incorporates potentially transient information external to the instance itself. Thus, some care needs to be taken in always ensuring this information is hashable.

94.6.3 Design Principles

These are (currently) Dav Clark’s best guess at what the group might agree on:

It should be very easy to figure out what was done by the pipeline.

Code should support relocatability - this could be via URIs, relative paths or potentially other mechanisms.

Unless otherwise called for, code should be thread safe, just in case.

The pipeline should make it easy to change aspects of an analysis with minimal recomputation, downloading, etc. (This is not the case currently - any change will overwrite the old node). Also, the fact that multiple files get rolled into a single node is problematic for similar reasons. E.g. - node([file1 . . . file100]) will get recomputed if we add only one file!.

However, it should also be easy to identify and delete things you don’t need anymore.

Pipelines and bits of pipelines should be easy to share.

Things that are the same should be called the same thing in most places. For interfaces that have an obvious meaning for the terms, “infiles” and “outfile(s)”. If a file is in both the inputs and outputs, it should be called the same thing in both places. If it is produced by one interface and consumed by another, same thing should be used.
94.6.4 Discussions

Auto-generated filenames

In refactoring the inputs in the traitslet branch I’m working through the different ways that filenames are generated and want to make sure the interface is consistent. The notes below are all using fsl.Bet as that’s the first class we’re Trait. Other interface classes may handle this differently, but should agree on a convention and apply it across all Interfaces (if possible).

Current Rules

These rules are for fsl.Bet, but it appears they are the same for all fsl and spm Interfaces.

Bet has two mandatory parameters, infile and outfile. These are the rules for how they are handled in different use cases.

1. If infile or outfile are absolute paths, they are used as-is and never changed. This allows users to override any filename/path generation.
2. If outfile is not specified, a filename is generated.
3. Generated filenames (at least for outfile) are based on:
   • infile, the filename minus the extensions.
   • A suffix specified by the Interface. For example Bet uses _brain suffix.
   • The current working directory, os.getcwd(). Example:
     If infile == ‘foo.nii’ and the cwd is /home/cburns then generated outfile for Bet will be /home/cburns/foo_brain.nii.gz
4. If outfile is not an absolute path, for instance just a filename, the absolute path is generated using os.path.realpath. This absolute path is needed to make sure the packages (Bet in this case) write the output file to a location of our choosing. The generated absolute path is only used in the cmdline at runtime and does __not__ overwrite the class attr self.inputs.outfile. It is generated only when the cmdline is invoked.

Walking through some examples

In this example we assign infile directly but outfile is generated in Bet._parse_inputs based on infile. The generated outfile is only used in the cmdline at runtime and not stored in self.inputs.outfile. This seems correct.

```
In [15]: from nipype.interfaces import fsl
In [16]: mybet = fsl.Bet()
In [17]: mybet.inputs.infile = 'foo.nii'
In [18]: res = mybet.run()
In [19]: res.runtime.cmdline
Out[19]: 'bet foo.nii /Users/cburns/src/nipy-sf/nipype/trunk/nipype/interfaces/
˓→tests/foo_brain.nii.gz'
In [21]: mybet.inputs
Out[21]: Bunch(center=None, flags=None, frac=None, functional=None, 
  infile='foo.nii', mask=None, mesh=None, nooutput=None, outfile=None, 
  outline=None, radius=None, reduce_bias=None, skull=None, threshold=None, 
  verbose=None, vertical_gradient=None)
In [24]: mybet.cmdline
Out[24]: 'bet foo.nii /Users/cburns/src/nipy-sf/nipype/trunk/nipype/interfaces/
˓→tests/foo_brain.nii.gz'
```
In [25]: mybet.inputs.outfile
In [26]: mybet.inputs.infile
Out[26]: 'foo.nii'

We get the same behavior here when we assign `infile` at initialization:

In [28]: mybet = fsl.Bet(infile='foo.nii')
In [29]: mybet.cmdline
Out[29]: 'bet foo.nii /Users/cburns/src/nipy-sf/nipype/trunk/nipype/interfaces/
<tests/foo_brain.nii.gz'
In [30]: mybet.inputs
Out[30]: Bunch(center=None, flags=None, frac=None, functional=None,
infile='foo.nii', mask=None, mesh=None, nooutput=None, outfile=None,
outline=None, radius=None, reduce_bias=None, skull=None, threshold=None,
verbose=None, vertical_gradient=None)
In [31]: res = mybet.run()
In [32]: res.runtime.cmdline
Out[32]: 'bet foo.nii /Users/cburns/src/nipy-sf/nipype/trunk/nipype/interfaces/
<tests/foo_brain.nii.gz'

Here we specify absolute paths for both `infile` and `outfile`. The command line's look as expected:

In [53]: import os
In [54]: mybet = fsl.Bet()
In [55]: mybet.inputs.infile = os.path.join('/Users/cburns/tmp/junk', 'foo.nii')
In [56]: mybet.inputs.outfile = os.path.join('/Users/cburns/tmp/junk', 'bar.nii')
In [57]: mybet.cmdline
Out[57]: 'bet /Users/cburns/tmp/junk/foo.nii /Users/cburns/tmp/junk/bar.nii'
In [58]: res = mybet.run()
In [59]: res.runtime.cmdline

Here passing in a new `outfile` in the `run` method will update `mybet.inputs.outfile` to the passed in value. Should this be the case?

In [110]: mybet = fsl.Bet(infile='foo.nii', outfile='bar.nii')
In [111]: mybet.inputs.outfile
Out[111]: 'bar.nii'
In [112]: mybet.cmdline
Out[112]: 'bet foo.nii /Users/cburns/src/nipy-sf/nipype/trunk/nipype/interfaces/
<tests/bar.nii'
In [113]: res = mybet.run(outfile = os.path.join('/Users/cburns/tmp/junk', 'not_ 
<bar.nii'))
In [114]: mybet.inputs.outfile
Out[114]: '/Users/cburns/tmp/junk/not_bar.nii'

In [115]: mybet.cmdline
Out[115]: 'bet foo.nii /Users/cburns/tmp/junk/not_bar.nii'

In this case we provide outfile but not as an absolute path, so the absolute path is generated and used for the cmdline when run, but mybet.inputs.outfile is not updated with the absolute path.

In [74]: mybet = fsl.Bet(infile='foo.nii', outfile='bar.nii')

In [75]: mybet.inputs.outfile
Out[75]: 'bar.nii'

In [76]: mybet.cmdline
Out[76]: 'bet foo.nii /Users/cburns/src/nipy-sf/nipype/trunk/nipype/interfaces/˓
→tests/bar.nii'

In [77]: res = mybet.run()

In [78]: res.runtime.cmdline
Out[78]: 'bet foo.nii /Users/cburns/src/nipy-sf/nipype/trunk/nipype/interfaces/˓
→tests/bar.nii'

In [80]: res.interface.inputs.outfile
Out[80]: 'bar.nii'

94.7 W3C PROV support

94.7.1 Overview

We’re using the the W3C PROV data model to capture and represent provenance in Nipype. For an overview see: PROV-DM overview

Each interface writes out a provenance.json (currently prov-json) or provenance.rdf (if rdflib is available) file. The workflow engine can also write out a provenance of the workflow if instructed. This is very much an experimental feature as we continue to refine how exactly the provenance should be stored and how such information can be used for reporting or reconstituting workflows. By default provenance writing is disabled for the 0.9 release, to enable insert the following code at the top of your script:

```python
>>> from nipype import config
>>> config.enable_provenance()
```

94.8 Software using Nipype

94.8.1 Configurable Pipeline for the Analysis of Connectomes (C-PAC)

C-PAC is an open-source software pipeline for automated preprocessing and analysis of resting-state fMRI data. C-PAC builds upon a robust set of existing software packages including AFNI, FSL, and ANTS, and makes it easy for both novice users and experts to explore their data using a wide array of analytic tools. Users define analysis pipelines by specifying a combination of preprocessing options and analyses to be run on an arbitrary number of subjects. Results can then be compared across groups using the integrated group statistics feature. C-PAC makes extensive use of Nipype Workflows and Interfaces.
94.8.2 BRAINSTools

BRAINSTools is a suite of tools for medical image processing focused on brain analysis.

94.8.3 Brain Imaging Pipelines (BIPs)

BIPs is a set of predefined Nipype workflows coupled with a graphical interface and ability to save and share workflow configurations. It provides both Nipype Workflows and Interfaces.

94.8.4 BROCCOLI

BROCCOLI is a piece of software for fast fMRI analysis on many core CPUs and GPUs. It provides Nipype Interfaces.

94.8.5 Forward

Forward is set of tools simplifying the preparation of accurate electromagnetic head models for EEG forward modeling. It uses Nipype Workflows and Interfaces.

94.8.6 Limbo

Limbo is a toolbox for finding brain regions that are neither significantly active nor inactive, but rather “in limbo”. It was build using custom Nipype Interfaces and Workflows.

94.8.7 Lyman

Lyman is a high-level ecosystem for analyzing task based fMRI neuroimaging data using open-source software. It aims to support an analysis workflow that is powerful, flexible, and reproducible, while automating as much of the processing as possible. It is build upon Nipype Workflows and Interfaces.

94.8.8 Medimsight

Medimsight is a commercial service medical imaging cloud platform. It uses Nipype to interface with various neuroimaging software.

94.8.9 MIA

MIA MIA is a a toolkit for gray scale medical image analysis. It provides Nipype interfaces for easy integration with other software.

94.8.10 Mindboggle

Mindboggle software package automates shape analysis of anatomical labels and features extracted from human brain MR image data. Mindboggle can be run as a single command, and can be easily installed as a cross-platform virtual machine for convenience and reproducibility of results. Behind the scenes, open source Python and C++ code run within a Nipype pipeline framework.

94.8.11 OpenfMRI

OpenfMRI is a repository for task based fMRI datasets. It uses Nipype for automated analysis of the deposited data.

94.8.12 serial functional Diffusion Mapping (sfDM)

’sfDM <http://github.com/PIRCImagingTools/sfDM>‘ is a software package for looking at changes in diffusion profiles of different tissue types across time. It uses Nipype to process the data.
94.8.13 The Stanford CNI MRS Library (SMAL)

SMAL is a library providing algorithms and methods to read and analyze data from Magnetic Resonance Spectroscopy (MRS) experiments. It provides an API for fitting models of the spectral line-widths of several different molecular species, and quantify their relative abundance in human brain tissue. SMAL uses Nipype Workflows and Interfaces.

94.8.14 tract_querier

tract_querier is a White Matter Query Language tool. It provides Nipype interfaces.

94.9 Testing nipype

In order to ensure the stability of each release of Nipype, the project uses two continuous integration services: CircleCI and Travis CI. If both batteries of tests are passing, the following badges should be shown in green color:

---

94.9.1 Installation for developers

To check out the latest development version:

```
  git clone https://github.com/nipy/nipype.git
```

After cloning:

```
  cd nipype
  pip install -r requirements.txt
  pip install -e .[dev]
```

94.9.2 Test implementation

Nipype testing framework is built upon pytest. After installation in developer mode, the tests can be run with the following command at the root folder of the project:

```
  pytest -v --doctest-modules nipype
```

A successful test run should complete in 10-30 minutes and end with something like:

```
2445 passed, 41 skipped, 7 xfailed in 1277.66 seconds
```

No test should fail (unless you’re missing a dependency). If the SUBJECTS_DIR environment variable is not set, some FreeSurfer related tests will fail. If any of the tests failed, please report them on our bug tracker.

On Debian systems with a local copy of MATLAB installed, set the following environment variable before running tests:

```
  export MATLABCMD=$pathtomatlabdir/bin/$platform/MATLAB
```

where $pathtomatlabdir is the path to your matlab installation and $platform is the directory referring to x86 or x64 installations (typically glnxa64 on 64-bit installations).
**Skipped tests**

Nipype will skip some tests depending on the currently available software and data dependencies. Installing software dependencies and downloading the necessary data will reduce the number of skipped tests. A few tests in Nipype make use of some images distributed within the FSL course data. This reduced version of the package can be downloaded here. To enable the tests depending on these data, just unpack the tar.gz file and set the `FSL_COURSE_DATA` environment variable to point to that folder. Note, that the test execution time can increase significantly with these additional tests.

**Xfailed tests**

Some tests are expected to fail until the code will be changed or for other reasons.

### 94.9.3 Testing Nipype using Docker

Nipype is tested inside Docker containers and users can use nipype images to test local versions. First, install the [Docker Engine](https://www.docker.com). Nipype has one base docker image called `nipype/nipype:base`, that contains several useful tools (FreeSurfer, AFNI, FSL, ANTs, etc.), and additional test images for specific Python versions: py27 for Python 2.7 and py36 for Python 3.6.

Users can pull the nipype image for Python 3.6 as follows:

```bash
docker pull nipype/nipype:py36
```

In order to test a local version of nipype you can run test within container as follows:

```bash
docker run -it -v $PWD:/src/nipype --rm nipype/nipype:py36 py.test -v --doctest-modules /src/nipype/nipype
```

### 94.9.4 Additional comments

If the project is tested both on your local OS and within a Docker container, you might have to remove all `__pycache__` directories before switching between your OS and a container.
Bibliography


[Rohde04] Rohde et al., Comprehensive Approach for Correction of Motion and Distortion in Diffusion-Weighted MRI, MRM 51:103-114 (2004).


Python Module Index

nipype.algorithms.confounds, 363
nipype.algorithms.icc, 367
nipype.algorithms.misc, 390
nipype.algorithms.modelgen, 398
nipype.caching.memory, 1599
nipype.conftest, 1601
nipype.interfaces.base.specs, 607
nipype.interfaces.base.support, 607
nipype.interfaces.base.traits_extension, 607
nipype.interfaces.brainsuite.brainsuite, 637
nipype.interfaces.cmtk.cmtk, 709
nipype.interfaces.cmtk.nbs, 713
nipype.interfaces.cmtk.nx, 715
nipype.interfaces.cmtk.parcellation, 717
nipype.interfaces.dcmstack, 1560
nipype.interfaces.dipy.base, 732
nipype.interfaces.dipy.preprocess, 734
nipype.interfaces.freesurfer.utils, 910
nipype.interfaces.fsl.model, 998
nipype.interfaces.io, 1584
nipype.interfaces.matlab, 1602
nipype.interfaces.minc.base, 1065
nipype.interfaces.minc.testdata, 1112
nipype.interfaces.mrtrix.convert, 1145
nipype.interfaces.mrtrix.tensors, 1165
nipype.interfaces.niftyreg.base, 1225
nipype.interfaces.vtkbase, 1597
nipype.pipeline.engine.base, 1612
nipype.pipeline.engine.nodes, 1613
nipype.pipeline.engine.utils, 1621
nipype.pipeline.engine.workflows, 1623
nipype.workflows.data, 3
nipype.workflows.dmri.camino.group_connectivity, 5
nipype.workflows.dmri.camino.group_connectivity, 6
nipype.workflows.dmri.camino.diffusion, 6
nipype.workflows.dmri.cmri.connectivity.group_connectivity, 7
nipype.workflows.dmri.cmri.connectivity.group_connectivity, 8
nipype.workflows.dmri.cmri.connectivity.nx, 11
nipype.workflows.dmri.dipy.denoise, 13
nipype.workflows.dmri.dtik.tensor_registration, 14
nipype.workflows.dmri.dcmartifacts, 16
nipype.workflows.dmri.fsl.dti, 33
nipype.workflows.dmri.fsl.epi, 37
nipype.workflows.dmri.fsl.tbss, 49
nipype.workflows.dmri.fsl.utils, 60
nipype.workflows.dmri.mrtrix.connectivity_mapping, 66
nipype.workflows.dmri.mrtrix.diffusion, 67
nipype.workflows.dmri.mrtrix.group_connectivity, 69
nipype.workflows.fsl.epi, 71
nipype.workflows.fsl.preprocess, 75
nipype.workflows.fsl.spm.preprocess, 85
nipype.workflows.misc.utils, 91
nipype.workflows.rsfmri.fsl.resting, 93
nipype.workflows.smri.ants.ANTSBuildTemplate, 99
nipype.workflows.smri.ants.antsRegistrationBuildTemplate, 101
nipype.workflows.smri.freesurfer.autorecon1, 102
nipype.workflows.smri.freesurfer.autorecon2, 102
nipype.workflows.smri.freesurfer.bem, 102
nipype.workflows.smri.freesurfer.recon, 104
ApplyTOPUP, 934
ApplyTransform, 1525
ApplyTransforms, 567
ApplyTransformsToPoints, 569
ApplyVolTransform, 812
ApplyWarp, 772, 1000
ApplyXFM, 1002
AR1Image, 954
args (nipype.sphinxext.plot_workflow.GraphError attribute), 1628
ArtifactDetect, 401
AssertEqual, 1531
Atropos, 577
Autobox, 493
Automask, 427
AutoTcorrelate, 426
AutoTLRC, 424
Average, 1065
AverageAffineTransform, 594
AverageImages, 595
AverageNetworks, 713
AvScale, 1029
Axialize, 494
B
B0Calc, 999
Bandpass, 428
base_trait () (nipype.interfaces.matlab.MatlabInputSpec method), 1606
BaseInterface, 603
BBBox, 1068
BBRegister, 814
BDF, 609
Beast, 1069
BEDPOSTX5, 913
BestLinReg, 1071
BET, 1005
Bfc, 617
BIDSDataGrabber, 1569
BigAverage, 1072
Binarize, 780
BinaryMaskEditorBasedOnLandmarks, 1377
BinaryMaths, 955, 1248
BinaryMathsInteger, 1250
BinaryStats, 1258
BinThresh, 759
BinThreshTask, 760
Blob, 1074
Blur, 1074
BlurInMask, 430
BlurToFWMH, 432
BrainExtraction, 578
BrainMask, 1197
BRAINSABC, 1364
BRAINSAignMSP, 1379
BRAINSClipInferior, 1380
BRAINSConstellationDetector, 1367
BRAINSConstellationModeler, 1381
BRAINSCreateLabelMapFromProbabilityMaps, 1372
BRAINSCut, 1373
BRAINSDemonWarp, 1356, 1463
BRAINSEyeDetector, 1383
BRAINSFit, 1345, 1453
BRAINSLinitialControlPoints, 1384
BRAINSLandmarkInitializer, 1385
BRAINSLinearModelerEPCA, 1385
BRAINS_linearTransform, 1386
BRAINSMultiSTAPLE, 1374
BRAINSmush, 1387
BRAINTSPosteriorToContinuousClass, 1271
BRAINSSpatialResample, 1353, 1460
BRAINSSResize, 1355
BRAINSROIAuto, 1375, 1472
BRAINTSSnapShotWriter, 1389
BRAINTalairach, 1272
BRAINSTalairachMask, 1273
BRAINTransformFromFiducials, 1359
BRAINTsTrimForegroundInDirection, 1391
BrickStat, 495
bru2, 1545
Bse, 620
BSplineDeformableRegistration, 1440
BSplineToDeformationField, 1434
Bucket, 496
BuildConnectome, 1179
buildtemplateparallel, 550

C
C3d, 1547
C3dAffineTool, 1548
cache () (nipype.caching.memory.Memory method), 1600
CALabel, 816
Calc, 498, 1076
CalcCoregAffine, 1526
CalcTopNCC, 1243
CalculateMedian, 381
CalculateNormalizedMoments, 381
Camino2Trackvis, 701
can_resume (nipype.interfaces.matlab.MatlabCommand attribute), 1603
CannyEdge, 1330
CannySegmentationLevelSetImageFilter, 1331
CANormalize, 818
CAREgister, 819
count_iterables() (in module nipype.pipeline.engine.utils), 1621
CreateJacobianDeterminantImage, 596
CreateMatrix, 705
CreateNifti, 382
CreateNodes, 707
CreateTiledMosaic, 601
CreateWarped, 1502
CSD, 735
CSVReader, 1535
Curvature, 867
CurvatureAnisotropicDiffusion, 1419
CurvatureStats, 868

D
DARTEL, 1503
DARTELNorm2MNI, 1504
DataFinder, 1570
DataGrabber, 1571
DataSink, 1572
Dcm2ni, 1551
Dcm2niix, 1553
DcmStack, 1557
Deconvolve, 408
default_traits_view() (nipype.interfaces.matlab.MatlabInputSpec method), 1608
DegreeCentrality, 434
Denoise, 733
DenoiseImage, 584
Despike, 435
Detrend, 436
Dewisp, 624
Dfs, 625
DICOMConvert, 821
DicomImport, 1527
DicomToNrrdConverter, 1401
Diffgeo, 749
diffgeoScalarVol, 750
diffgeoScalarVolTask, 757
diffgeoSymTensor3DVol, 751
diffgeoSymTensor3DVolTask, 758
diffgeoTask, 752
DiffusionTensorScalarMeasurements, 1408
DiffusionTensorStreamlineTrack, 1165
DiffusionWeightedVolumeMasking, 1409
DilateImage, 957, 1332
DilateMask, 1333
DipyBaseInterface, 732
dipyDiffusionInterface, 732
Directions2Amplitude, 1160
disconnect() (nipype.pipeline.engine.workflows.Workflow method), 1625
Distance, 373, 382
DistanceMap, 918
DistanceMaps, 1333
Dot, 506
DT2NIfTI, 647
DTI, 739
dtiaverage, 1282
dtiestim, 1283
DTIexport, 1403
DTIFit, 661, 916
DTIimport, 1404
dti processes, 1286
DTIREcon, 719
DTItracker, 720
DTLUTGen, 662
DTMetric, 663
DualRegression, 977
Dump, 1081
dumpBinaryTrainingVectors, 1334
DWI2SphericalHarmonicsImage, 1159
DWI2Tensor, 1146
DWIBiasCorrect, 1183
DWICompare, 1278
DWIConvert, 1280
DWIDenoise, 1185
DWIEstimate, 1201
DWIJointRicianLMMSEFilter, 1405
DWIRicianLMMSEFilter, 1406
DWISimpleCompare, 1279
DWIToDTIEstimation, 1407
dwiTool, 1214
DWIUnbiasedNonLocalMeansFilter, 1435

E
ECM, 437
Eddy, 937
EddyCorrect, 940
EddyQuad, 941
Edge3, 507
edit_traits() (nipype.interfaces.matlab.MatlabInputSpec method), 1608
editable_traits() (nipype.interfaces.matlab.MatlabInputSpec method), 1608
EditTransform, 775
EditWMwithAseg, 822
EM, 1241
EMRegister, 851
EMSegmentCommandLine, 1473
EMSegmentTransformToNewFormat, 1484
EngineBase (class in nipype.pipeline.engine.base), 1612
EPIDeWarp, 936
EpiReg, 943
Erode, 1147
ErodeImage, 958, 1335
ErrorMap, 373
ESLR, 1378
EstimateContrast, 1263, 1487
EstimateFOD, 1190
EstimateResponseForSH, 1161
EstimateResponseSH, 736
EulerNumber, 869
Eval, 509
evaluate_connect_function() (in module nipype.pipeline.engine.utils), 1621
expand_iterables() (in module nipype.pipeline.engine.utils), 1621
ExpertAutomatedRegistration, 1442
export() (nipype.pipeline.engine.workflows.Workflow method), 1625
export_graph() (in module nipype.pipeline.engine.utils), 1621
Extract, 1082
ExtractMainComponent, 870
extractNrrdVectorIndex, 1291
ExtractROI, 1036
ExtractSkeleton, 1423

F
FactorialDesign, 1490
FAST, 1008
csvs_to_hdf5, 1397
FEAT, 978
FEATModel, 979
FEATRegister, 980
FeatureExtractor, 952
fiberprocess, 1321
fiberstats, 1320
fibertrack, 1323
FiducialRegistration, 1466
FieldMap, 1506
filename() (nipype.sphinxext.plot_workflow.ImageFile method), 1628
filenames() (nipype.sphinxext.plot_workflow.ImageFile method), 1628
FillLesions, 1246
FILMGLS, 980
FilterRegressor, 1037
FilterTracks, 1167
Fim, 439
FindCenterOfBrain, 1392
FindShPeaks, 1163
FindTheBiggest, 919
FIRST, 1010
FitAsl, 1211
FitDwi, 1217
FitGLM, 1264
FitMSParams, 823
FitQt1, 1221
FitTensor, 1192
FixTopology, 871
FLAMEO, 982
FlippedDifference, 1336
FLIRT, 1011
FNIRT, 1015
format_dot() (in module nipype.pipeline.engine.utils), 1621
format_node() (in module nipype.pipeline.engine.utils), 1621
Fourier, 440
FramewiseDisplacement, 360
FreeSurferSource, 1573
FUGUE, 1019
fullname (nipype.pipeline.engine.base.EngineBase attribute), 1613
fullname (nipype.pipeline.engine.nodes.JoinNode attribute), 1615
fullname (nipype.pipeline.engine.nodes.MapNode attribute), 1617
fullname (nipype.pipeline.engine.nodes.Node attribute), 1620
fullname (nipype.pipeline.engine.workflows.Workflow attribute), 1625
Function, 1536
FuseSegmentations, 777
FuzzyOverlap, 374, 383
FWHMx, 510

G
GaussianBlurImageFilter, 1420
GCOR, 513
Generate5tt, 1202
generate_expanded_graph() (in module nipype.pipeline.engine.utils), 1621
GenerateBrainClippedImage, 1336
GenerateCsfClippedFromClassifiedImage, 1328
GenerateDirections, 1164
GenerateEdgeMapImage, 1275
GenerateLabelMapFromProbabilityMap, 1394
GeneratePurePlugMask, 1276
GenerateSummedGradientImage, 1337
GenerateTestImage, 1338
GenerateWhiteMatterMask, 1148
Gennlxfm, 1085
GenWarpFields, 547
get() (nipype.interfaces.matlab.MatlabInputSpec method), 1608
get_all_files() (in module nipype.pipeline.engine.utils), 1621
get_hashval() (nipype.interfaces.matlab.MatlabInputSpec method), 1608
get_levels() (in module nipype.pipeline.engine.utils), 1621
get_matlab_command() (in module nipype.interfaces.matlab), 1612
get_node() (nipype.pipeline.engine.workflows.Workflow method), 1625
get_output() (nipype.pipeline.engine.nodes.JoinNode method), 1615
get_output() (nipype.pipeline.engine.nodes.MapNode method), 1617
get_output() (nipype.pipeline.engine.nodes.Node method), 1620
get_print_name() (in module nipype.pipeline.engine.utils), 1621
get_subnodes() (nipype.pipeline.engine.nodes.MapNode method), 1617
get_traitsfree() (nipype.interfaces.matlab.MatlabInputSpec method), 1608
get_wf_formats() (in module nipype.sphinxext.plot_workflow), 1628

GLM, 985
GLMfit, 784
GradientAnisotropicDiffusion, 1421
GradientAnisotropicDiffusionImageFilter, 1339
GraphError (class in nipype.sphinxext.plot_workflow), 1628
GrayscaleFillHoleImageFilter, 1426
GrayscaleGrindPeakImageFilter, 1427
GrayscaleModelMaker, 1477
GroupAndStack, 1558
gtractAnisotropyMap, 1291
gtractAverageBValues, 1292
gtractClipAnisotropy, 1293
gtractConcatDwi, 1297
gtractCopyImageOrientation, 1298
gtractCoRegAnatomy, 1294
gtractCoregValues, 1299
gtractCostFastMarching, 1301
gtractCreateGuideFiber, 1302
gtractFastMarchingTracking, 1303
gtractFiberTracking, 1305
gtractImageConformity, 1308
gtractInvertBSplineTransform, 1309
gtractInvertDisplacementField, 1310
gtractInvertRigidTransform, 1310
gtractResampleAnisotropy, 1311
gtractResampleB0, 1312
gtractResampleCodeImage, 1313
gtractResampleDWIInPlace, 1314
gtractResampleFibers, 1316
gtractTensor, 1317

Index

H

HammerAttributeCreator, 1339
HARDIMat, 722

Help

help() (nipype.interfaces.matlab.MatlabCommand class method), 1603
help() (nipype.pipeline.engine.nodes.JoinNode method), 1615
help() (nipype.pipeline.engine.nodes.MapNode method), 1617
help() (nipype.pipeline.engine.nodes.Node method), 1620
Hemisplit, 627
Hist, 441
HistogramMatching, 1424
HistogramMatchingFilter, 1277

I

ICA_AROMA, 911
ICC, 367
IdentityInterface, 1531
Image2Voxel, 648
ImageFile (class in nipype.sphinxext.plot_workflow), 1628
ImageLabelCombine, 1425
ImageMaths, 1038
ImageMeants, 1039
ImageRegionPlotter, 1394
ImageStats, 698, 1040
input_spec (nipype.interfaces.matlab.MatlabCommand attribute), 1603
inputs (nipype.pipeline.engine.base.EngineBase attribute), 1613
inputs (nipype.pipeline.engine.nodes.JoinNode attribute), 1615
inputs (nipype.pipeline.engine.nodes.MapNode attribute), 1617
inputs (nipype.pipeline.engine.nodes.Node attribute), 1620
inputs (nipype.pipeline.engine.workflows.Workflow attribute), 1625
insertMidACPCpoint, 1398
IntensityDifferenceMetric, 1451
interface (nipype.pipeline.engine.nodes.JoinNode attribute), 1615
interface (nipype.pipeline.engine.nodes.MapNode attribute), 1617
interface (nipype.pipeline.engine.nodes.Node attribute), 1620
InvWarp, 1041
IOBase, 1576
is_cached() (nipype.pipeline.engine.nodes.JoinNode method), 1615
is_cached() (nipype.pipeline.engine.nodes.MapNode method), 1617
is_cached() (nipype.pipeline.engine.nodes.Node method), 1620
IsotropicSmooth, 959
items() (nipype.interfaces.matlab.MatlabInputSpec method), 1609
itername (nipype.pipeline.engine.base.EngineBase attribute), 1613
itername (nipype.pipeline.engine.nodes.JoinNode attribute), 1615
itername (nipype.pipeline.engine.nodes.MapNode attribute), 1617
itername (nipype.pipeline.engine.nodes.Node attribute), 1620
itername (nipype.pipeline.engine.workflows.Workflow attribute), 1625

J
Jacobian, 872
JistBrainMgdmDuraEstimation, 1113
JistBrainMgdmSkullStripping, 1115
JistBrainMgdmSurfaceMeshInflation, 1117
JistIntesityMgdmMasking, 1120
JistLaminarProfileCalculator, 1121
JistLaminarProfileGeometry, 1122
JistLaminarProfileSampling, 1123
JistLaminarROIAligning, 1124
JistLaminarVolumetricLayering, 1125
joinfield (nipype.pipeline.engine.nodes.JoinNode attribute), 1615
JoinNode (class in nipype.pipeline.engine.nodes), 1614
joinsource (nipype.pipeline.engine.nodes.JoinNode attribute), 1615
JointFusion, 585
JointHistogram, 1395
JSONFileGrabber, 1576
JSONFileSink, 1577

K
KellyKapowski, 587

L
L2Model, 988
Label2Annot, 789
Label2Label, 790
Label2Vol, 791
LabelConfig, 1181
LabelConvert, 1182
LabelFusion, 1244
LabelGeometry, 597
LabelMapSmoothing, 1478
landmarksConstellationAligner, 1399
landmarksConstellationWeights, 1399
LaplacianThickness, 589
Level1Design, 988, 1491
LFCD, 442
LibraryBaseInterface, 604
LinearRegistration, 1444
LinRecon, 691
list_node_names() (nipype.pipeline.engine.workflows.Workflow method), 1625
load() (nipype.pipeline.engine.base.EngineBase method), 1613
load() (nipype.pipeline.engine.nodes.JoinNode method), 1615
load() (nipype.pipeline.engine.nodes.MapNode method), 1617
load() (nipype.pipeline.engine.nodes.Node method), 1620
load() (nipype.pipeline.engine.workflows.Workflow method), 1625
load_inputs_from_json() (nipype.interfaces.matlab.MatlabCommand method), 1603
load_resultfile() (in module nipype.pipeline.engine.utils), 1621
LocalBistat, 514
Localstat, 516
LookupMeta, 1559
LTAConvert, 873

M
MakeAverageSubject, 885
MakeDyadicVectors, 920
MakeSurfaces, 886
MapNode (class in nipype.pipeline.engine.nodes), 1616
mark_wf_labels() (in module nipype.sphinxext.plot_workflow), 1628
Maskave, 443
MaskScalarVolume, 1416
MaskTool, 519
Math, 1086
MathsCommand, 960, 1251
Matlab2CSV, 384
MatlabCommand (class in nipype.interfaces.matlab), 1602
MatlabInputSpec (class in nipype.interfaces.matlab), 1604
maxcurvature, 1319
MaxImage, 961
MaxnImage, 962
MCFLIRT, 1022
MeanImage, 963
Means, 444
MeasureImageSimilarity, 553
MedianFilter3D, 1153
MedianImage, 964
MedianImageFilter, 1422
MedicAlgorithmImageCalculator, 1127
MedicAlgorithmLesionToads, 1128
MedicAlgorithmMipavReorient, 1131
MedicAlgorithmN3, 1132
MedicAlgorithmSPECTRE2010, 1134
MedicAlgorithmThresholdToBinaryMask, 1138
MELODIC, 989
mem_gb (nipype.pipeline.engine.nodes.JoinNode attribute), 1615
mem_gb (nipype.pipeline.engine.nodes.MapNode attribute), 1617
mem_gb (nipype.pipeline.engine.nodes.Node attribute), 1620
Memory (class in nipype.caching.memory), 1600
Merge, 520, 1043, 1252, 1532
merge_bundles () (in module nipype.pipeline.engine.utils), 1622
merge_dict() (in module nipype.pipeline.engine.utils), 1622
MergeCNetworks, 711
MergeCSVFiles, 385
MergeModels, 1479
MergeNIfiti, 1559
MergeROIs, 385
MESD, 692
Mesh2PVE, 1206
MeshFix, 1585
MeshWarpMaths, 370
MetricResample, 1541
MinImage, 965
MNIBiasCorrection, 824
ModelFit, 665
ModelMaker, 1480
ModelToLabelMap, 1482
modify_paths() (in module nipype.pipeline.engine.utils), 1622
ModifyAffine, 386
MotionOutliers, 1044
MpiCommandLine, 604
MRtoMNII305, 852
MRConvert, 1149, 1204
MRDeGibbs, 1186
MRICoreg, 853
MRIFill, 875
MRIMarchingCubes, 876
MRIPretrans, 876
MRISCALabel, 832
MRISCalc, 879
MRISCombine, 880
MRISConvert, 881
MRISExpand, 883
MRISInflate, 884
MRISPpreproc, 793
MRISPpreprocReconAll, 795
MRITessellate, 878
MRMath, 1205
MRMultiple, 1150
MRTransform, 1151
MRTrax2TrackVis, 1145
MRTrax3Base, 1179
MRTraxViewer, 1152
MS_LDA, 797
MultiImageMaths, 966
MultipleRegressionDesign, 993
MultipleRegressionDesign, 1492
MultiplyImages, 598
MultiplyScalarVolumes, 1416
MultiResolutionAffineRegistration, 1446
MySQLSink, 1577
N
N4BiasFieldCorrection, 591
N4ITKBiasFieldCorrection, 1428
n_procs (nipype.pipeline.engine.nodes.JoinNode attribute), 1615
n_procs (nipype.pipeline.engine.nodes.MapNode attribute), 1617
n_procs (nipype.pipeline.engine.nodes.Node attribute), 1620
name (nipype.pipeline.engine.base.EngineBase attribute), 1620
name (nipype.pipeline.engine.nodes.JoinNode attribute), 1615
name (nipype.pipeline.engine.nodes.MapNode attribute), 1617
name (nipype.pipeline.engine.nodes.Node attribute), 1620
name (nipype.pipeline.engine.workflows.Workflow attribute), 1625
needed_outputs {nipype.pipeline.engine.nodes.JoinNode attribute}, 1615

needed_outputs {nipype.pipeline.engine.nodes.MapNode attribute}, 1617

needed_outputs {nipype.pipeline.engine.nodes.Node attribute}, 1620

NeighborhoodMean, 1340
NeighborhoodMedian, 1341
NetworkBasedStatistic, 712
NetworkXMetrics, 714
NewSegment, 1507
NIftiDT2Camino, 649
NiftiGeneratorBase, 1560
NiftyFitCommand, 1214
NiftyRegCommand, 1225
NiftySegCommand, 1241
NilearnBaseInterface, 1589
NipyBaseInterface, 1263
nipype.algorithms.confounds (module), 363
nipype.algorithms.icc (module), 367
nipype.algorithms.misc (module), 390
nipype.algorithms.modalgen (module), 398
nipype.caching.memory (module), 1599
nipype.conftest (module), 1601
nipype.interfaces.base.specs (module), 607
nipype.interfaces.base.support (module), 607
nipype.interfaces.base.traits_extension (module), 607
nipype.interfaces.brainsuite.brainsuite (module), 637
nipype.interfaces.cmtk.cmtk (module), 709
nipype.interfaces.cmtk.nbs (module), 713
nipype.interfaces.cmtk.nx (module), 715
nipype.interfaces.cmtk.parcellation (module), 717
nipype.interfaces.dcmstack (module), 1560
nipype.interfaces.dipy.base (module), 732
nipype.interfaces.dipy.preprocess (module), 734
nipype.interfaces.freesurfer.utils (module), 910
nipype.interfaces.fsl.model (module), 998
nipype.interfaces.io (module), 1584
nipype.interfaces.matlab (module), 1602
nipype.interfaces.minc.base (module), 1065
nipype.interfaces.minc.testdata (module), 1112
nipype.interfaces.mrtrix.convert (module), 1145
nipype.interfaces.mrtrix.tensors (module), 1165
nipype.interfaces.niftyreg.base (module), 1225
nipype.workflows.smri.freesurfer.autorecon2
   (module), 102
nipype.workflows.smri.freesurfer.bem
   (module), 102
nipype.workflows.smri.freesurfer.recon
   (module), 104
nipype.workflows.smri.freesurfer.utils
   (module), 106
nipype.workflows.smri.niftyreg.groupwise
   (module), 112
NitimeBaseInterface, 1270
NlpFit, 1091
Node (class in nipype.pipeline.engine.nodes)
   (in module nipype.pipeline.engine.nodes), 1618
num_subnodes ()
   (nipype.pipeline.engine.nodes.MapNode
   method), 1618
NwarpAdjust, 522
NwarpApply, 524
NwarpCat, 525
ODFRecon, 723
ODFTracker, 725
on_trait_change ()
   (nipype.interfaces.matlab.MatlabInputSpec
   method), 1609
on_trait_event ()
   (nipype.interfaces.matlab.MatlabInputSpec
   method), 1609
OneDToolPy, 527
OneSampleTTest, 798
OneSampleTTestDesign, 1494
OrientScalarVolume, 1402
OtsuThresholdImageFilter, 1436
OtsuThresholdSegmentation, 1450
out_of_date ()
   (in module nipype.sphinxext.plot_workflow), 1628
OutlierCount, 446
output_dir ()
   (nipype.pipeline.engine.nodes.JoinNode
   method), 1615
output_dir ()
   (nipype.pipeline.engine.nodes.MapNode
   method), 1618
output_dir ()
   (nipype.pipeline.engine.nodes.Node
   method), 1620
output_spec
   (nipype.interfaces.matlab.MatlabCommand
   attribute), 1603
outputs
   (nipype.pipeline.engine.nodes.JoinNode
   attribute), 1615
outputs
   (nipype.pipeline.engine.nodes.MapNode
   attribute), 1618
outputs
   (nipype.pipeline.engine.nodes.Node
   attribute), 1620
outputs
   (nipype.pipeline.engine.workflows.Workflow
   attribute), 1625
Paint, 856
PairedTTestDesign, 1495
Parcellate, 716
ParcellationStats, 888
ParseDICOMDir, 834
PatchMatch, 1257
PercentileImage, 967
PETPVC, 1591
PETStandardUptakeValueComputation, 1452
Pialmesh, 628
PickAtlas, 388
PicoPDFs, 667
Pik, 1094
PipeFunc (class in nipype.caching.memory), 1601
PlotMotionParams, 1047
PlotTimeSeries, 1048
PointsWarp, 773
PowerSpectrum, 1050
PowerSpectrumDE, 1024
PrepareFieldmap, 946
print_traits ()
   (nipype.interfaces.matlab.MatlabInputSpec
   method), 1610
ProbabilisticSphericallyDeconvolutedStreamlineTracking,
   1169
ProbeVolumeWithModel, 1483
ProbTrackX, 920
ProbTrackX2, 924
ProcStreamlines, 650
ProjThresh, 928
Pvc, 630
pytest_unconfigure ()
   (in module nipype.conftest), 1601
QBallMX, 694
QualityIndex, 447
Quickshear, 1595
Qwarp, 448
QwarpPlusMinus, 458
raise_exception()
    (nipype.interfaces.matlab.MatlabCommand
        method), 1603
Randomise, 994
RandomVol, 1139
read_log()  (in module nipype.caching.memory), 1601
Realign, 1513
RealignUnwarp, 1514
ReconAll, 835
references_  (nipype.interfaces.matlab.MatlabCommand
        attribute), 1603
Refit, 530
RegAladin, 1225
RegAverage, 1231
RegF3D, 1228
Register, 857
RegisterAVIToTalairach, 858
Registration, 554, 774
RegistrationSynQuick, 565
RegJacobian, 1233
RegMeasure, 1234
RegResample, 1235
RegTools, 1236
RegTransform, 1238
ReHo, 528
RelabelHypointensities, 890
Rmlfit, 412
remove_coding()  (in module nipype.sphinxext.plot_workflow), 1628
remove_nodes()  (nipype.pipeline.engine.workflows.Workflow
        method), 1625
remove_trait()  (nipype.pipeline.engine.MapNode
        method), 1610
remove_trait_listener()  (nipype.interfaces.matlab.MatlabInputSpec
        method), 1610
RemoveIntersection, 891
RemoveNeck, 892
Rename, 1533
render_figures()  (in module nipype.sphinxext.plot_workflow), 1629
Reorient, 1565
Reorient2Std, 1050
ReportCapableInterface, 1141
Resample, 532, 734, 841, 1096
ResampleDTIVolume, 1410
ResampleScalarVectorDWIVolume, 1430
ResampleScalarVolume, 1437
Rescale, 1566
reset_traits()  (nipype.interfaces.matlab.MatlabInputSpec
        method), 1610
Reshape, 1101
Reslice, 1528
ResliceToReference, 1529
resource_monitor  (nipype.interfaces.matlab.MatlabCommand
        attribute), 1603
ResponseSD, 1188
RESTORE, 737
result  (nipype.pipeline.engine.nodes.JoinNode
        attribute), 1615
result  (nipype.pipeline.engine.nodes.MapNode
        attribute), 1618
result  (nipype.pipeline.engine.nodes.Node
        attribute), 1620
Retroicor, 468
Rigid, 753
RigidRegistration, 1447
RigidTask, 754
rm_all_but()  (in module nipype.caching.memory),
    1601
RobustFOV, 1051
RobustRegister, 842
RobustStatisticsSegmenter, 1476
RobustTemplate, 778
ROIGen, 708
ROIStats, 466
run()  (in module nipype.sphinxext.plot_workflow), 1629
run()  (nipype.interfaces.matlab.MatlabCommand
        method), 1603
run()  (nipype.pipeline.engine.nodes.JoinNode
        method), 1616
run()  (nipype.pipeline.engine.nodes.MapNode
        method), 1618
run()  (nipype.pipeline.engine.nodes.Node
        method), 1620
run()  (nipype.pipeline.engine.workflows.Workflow
        method), 1625
run_code()  (in module nipype.sphinxext.plot_workflow), 1629
S
S3DataGrabber, 1578
SampleToSurface, 892
save()  (nipype.pipeline.engine.base.EngineBase
        method), 1613
save()  (nipype.pipeline.engine.nodes.JoinNode
        method), 1616
save()  (nipype.pipeline.engine.nodes.MapNode
        method), 1618
save()  (nipype.pipeline.engine.nodes.Node
        method), 1620
save()  (nipype.pipeline.engine.workflows.Workflow
        method), 1625
save_hashfile()  (in module nipype.pipeline.engine.utils), 1622