nighres Documentation

Release 1.1.0b1

Julia Huntenburg

Apr 13, 2019

Getting started

Nighres is a Python package for processing of high-resolution neuroimaging data. It developed out of CBS High-Res Brain Processing Tools and aims to make those tools easier to install, use and extend. Nighres now includes new functions from the IMCN imaging toolkit.

CHAPTER 1

Installing Nighres

Please see *Troubleshooting* if you run into errors during installation.

1.1 Requirements

To build Nighres you need:

- Python 3.5 or higher
- Java JDK 1.7 or higher
- JCC 3.0 or higher

The following Python packages are automatically installed with Nighres

- numpy
- nibabel
- psutils

For further dependencies of specific interfaces see *Optional dependencies*.

1.2 From PyPI

You can download the latest stable release of Nighres from PyPI.

Because parts of the package have to be built locally it is currently not possible to use `pip install` directly from PyPI. Instead, please download and unpack the tarball to *Build Nighres*. (Or use the *Docker image*)

1.3 From Github

You can also get the latest version from Github

```
git clone https://github.com/nighres/nighres
```

Or download and unpack the zip file from Github under **Clone and download -> Download ZIP**

1.4 Build Nighres

1. Make sure you have Java JDK and JCC installed and set up. You will likely need to point the JCC_JDK variable to you Java JDK installation, e.g on a Debian/Ubuntu amd64 system:

```
sudo apt-get install openjdk-8-jdk
export JCC_JDK=/usr/lib/jvm/java-8-openjdk-amd64
python3 -m pip install jcc
```

2. Navigate to the Nighres directory you downloaded and unpacked, and run the build script:

```
./build.sh
```

3. Install the Python package:

```
python3 -m pip install .
```

1.5 Testing the installation

You can often catch installation problems by simply import Nighres in Python. Make sure to navigate out of the directory from which you installed to make sure Nighres has actually been installed correctly and can be accessed from any location

```
python3 -c "import nighres"
```

If that works, you can try running one of the examples. You can find them inside the unpacked Nighres directory, in the subdirectory *examples*. Alternatively, you can also download the *examples* from the online documentation.

1.6 Docker

To quickly try out nighres in a preset, batteries-included environment, you can use the included Dockerfile, which includes Ubuntu 14 Trusty, openJDK-8, nighres, and Jupyter Notebook. The only thing you need to install is Docker, a lightweight container platform that runs on Linux, Windows and Mac OS X.

To build the Docker image, do the following:

```
git clone https://github.com/nighres/nighres
cd nighres
docker build . -t nighres
```

To run the Docker container:

```
docker run --rm -p 8888:8888 nighres
```

Now go with your browser to https://localhost:8888 to start a notebook. You should be able to import nighres by entering:

```
import nighres
```

into the first cell of your notebook.

Usually you also want to have access to some data when you run nighres. You can grant the Docker container access to a data folder on your host OS by using the *-v* tag when you start the container:

```
docker run --rm -v /home/me/my_data:/data -p 8888:8888 nighres
```

Now, in your notebook you will be able to access your data on the path */data*

1.7 Optional dependencies

Working with surface mesh files

- pandas

Using the registration tools

- nipype
- ANTs

Plotting in the examples

- Nilearn and its dependencies, if Nilearn is not installed, plotting in the examples will be skipped and you can view the results in any other nifti viewer

Using the docker image

- Docker

Building the documentation

- sphinx
- sphinx-gallery
- matplotlib
- sphinx-rtd-theme (pip install sphinx-rtd-theme)
- pillow (pip install pillow)
- mock

1.8 Troubleshooting

If you experience errors not listed here, please help us by reporting them through neurostars.org using the tag **nighres**, or on github. Or if you solve them yourself help others by contributing your solution here (see *Developers guide*)

1.8.1 Missing Java libraries

If you get errors regarding missing java libraries (such as ljvm/libjvm or ljava/libjava), although you install Java JDK, it means that JCC does not find the libraries. It can help to search for the "missing" library and make a symbolic link to it like this:

```
sudo find / -type f -name libjvm.so
>> /usr/lib/jvm/java-11-openjdk-amd64/lib/server/libjvm.so
sudo ln -s /usr/lib/jvm/java-11-openjdk-amd64/lib/server/libjvm.so /usr/lib/libjvm.so
```

1.8.2 Missing Python packages

If you get errors about Python packages not being installed, it might be that you are trying to run a function that requires *Optional dependencies*. If packages are reported missing that you think you have installed, make sure that they are installed under the same python installation as nighres. They should be listed when you run:

```
python3 -m pip list
```

If they aren't, install them using:

```
python3 -m pip install <package_name>
```

If there is still confusion, make sure nighres is installed in the same directory that your python3 -m pip command points to. These two commands should give the same base directory:

```
python3 -m pip
python3 -c 'import nighres; print(nighres.__file__)'
```

CHAPTER 2

Nighres usage examples

Note: Click *here* to download the full example code

2.1 Tissue classification from MP2RAGE data

This example shows how to obtain a tissue classification from MP2RAGE data by performing the following steps:

1. Downloading open MP2RAGE dataset using *nighres.data.download_7T_TRT()*
2. Remove the skull and create a brain mask using *nighres.brain.mp2rage_skullstripping()*
3. Atlas-guided tissue classification using MGDM *nighres.brain.mgdm_segmentation()*[1]

2.1.1 Import and download

First we import `nighres` and the `os` module to set the output directory Make sure to run this file in a directory you have write access to, or change the `out_dir` variable below.

```
import nighres
import os

in_dir = os.path.join(os.getcwd(), 'nighres_examples/data_sets')
out_dir = os.path.join(os.getcwd(), 'nighres_examples/tissue_classification')
```

We also try to import Nilearn plotting functions. If Nilearn is not installed, plotting will be skipped.

[1] Bogovic, Prince and Bazin (2013). A multiple object geometric deformable model for image segmentation. DOI: 10.1016/j.cviu.2012.10.006.A

```
skip_plots = False
try:
    from nilearn import plotting
except ImportError:
    skip_plots = True
    print('Nilearn could not be imported, plotting will be skipped')
```

Now we download an example MP2RAGE dataset. It is the structural scan of the first subject, first session of the 7T Test-Retest dataset published by Gorgolewski et al (2015)[2].

```
dataset = nighres.data.download_7T_TRT(in_dir, subject_id='sub001_sess1')
```

2.1.2 Skull stripping

First we perform skull stripping. Only the second inversion image is required to calculate the brain mask. But if we input the T1map and T1w image as well, they will be masked for us. We also save the outputs in the `out_dir` specified above and use a subject ID as the base file_name.

```
skullstripping_results = nighres.brain.mp2rage_skullstripping(
                                            second_inversion=dataset['inv2'],
                                            t1_weighted=dataset['t1w'],
                                            t1_map=dataset['t1map'],
                                            save_data=True,
                                            file_name='sub001_sess1',
                                            output_dir=out_dir)
```

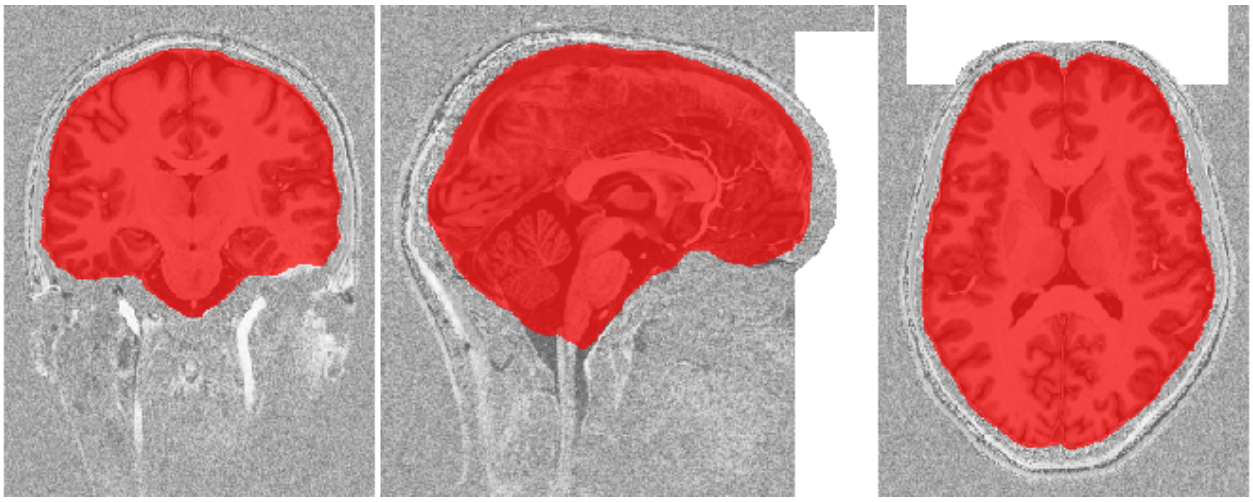
Tip: in Nighres functions that have several outputs return a dictionary storing the different outputs. You can find the keys in the docstring by typing `nighres.brain.mp2rage_skullstripping?` or list them with `skullstripping_results.keys()`

To check if the skull stripping worked well we plot the brain mask on top of the original image. You can also open the images stored in `out_dir` in your favourite interactive viewer and scroll through the volume.

Like Nilearn, we use Nibabel SpatialImage objects to pass data internally. Therefore, we can directly plot the outputs using Nilearn plotting functions .

```
if not skip_plots:
    plotting.plot_roi(skullstripping_results['brain_mask'], dataset['t1w'],
                      annotate=False, black_bg=False, draw_cross=False,
                      cmap='autumn')
```

[2] Gorgolewski et al (2015). A high resolution 7-Tesla resting-state fMRI test-retest dataset with cognitive and physiological measures. DOI: 10.1038/sdata.2014.54



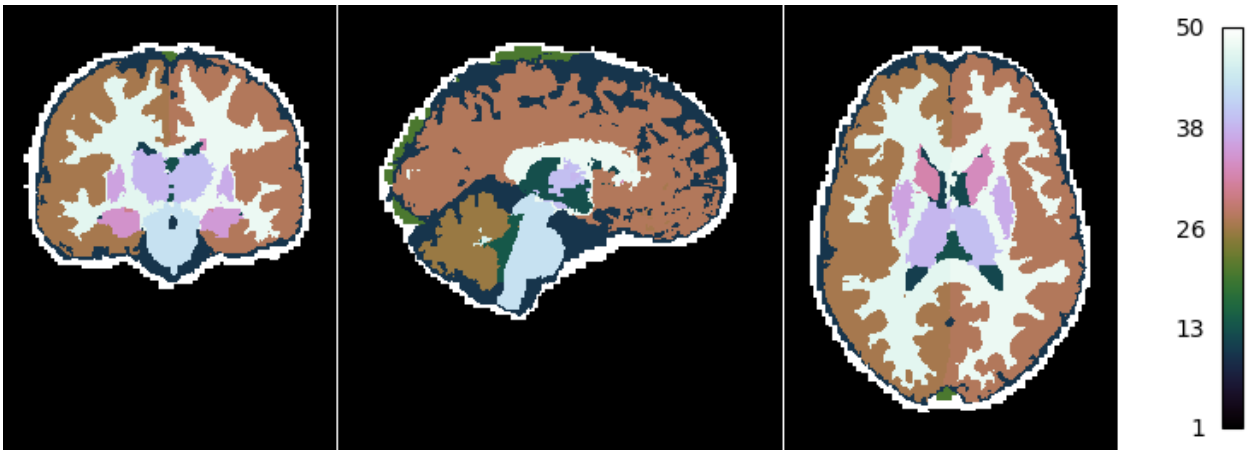
2.1.3 MGDM classification

Next, we use the masked data as input for tissue classification with the MGDM algorithm. MGDM works with a single contrast, but can be improved with additional contrasts. In this case we use the T1-weigthed image as well as the quantitative T1map.

```
mgdm_results = nighres.brain.mgdm_segmentation(
                        contrast_image1=skullstripping_results['t1w_masked'],
                        contrast_type1="Mp2rage7T",
                        contrast_image2=skullstripping_results['t1map_masked'],
                        contrast_type2="T1map7T",
                        save_data=True, file_name="sub001_sess1",
                        output_dir=out_dir)
```

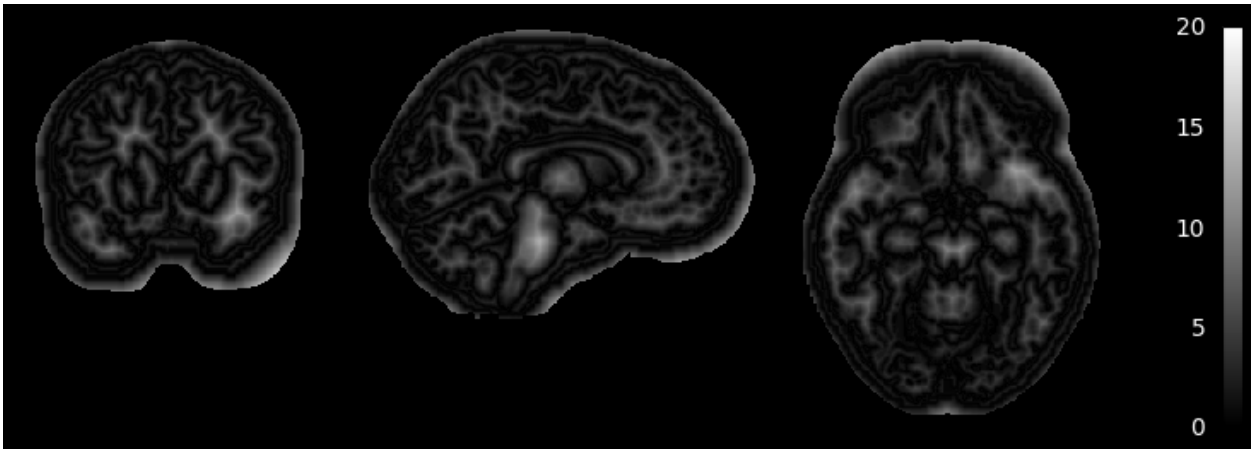
Now we look at the topology-constrained segmentation MGDM created

```
if not skip_plots:
    plotting.plot_img(mgdm_results['segmentation'],
                      vmin=1, vmax=50, cmap='cubehelix',  colorbar=True,
                      annotate=False,  draw_cross=False)
```



MGDM also creates an image which represents for each voxel the distance to its nearest border. It is useful to assess where partial volume effects may occur

```
if not skip_plots:
    plotting.plot_anat(mgdm_results['distance'], vmin=0, vmax=20,
                       annotate=False, draw_cross=False, colorbar=True)
```



If the example is not run in a jupyter notebook, render the plots:

```
if not skip_plots:
    plotting.show()
```

2.1.4 References

Total running time of the script: (0 minutes 0.000 seconds)

Note: Click *here* to download the full example code

2.2 Cortical depth estimation from MGDM segmentation

This example shows how to obtain a cortical laminar depth representation from an MGDM segmentation result with the following steps:

1. Get a segmentation result from the *tissue_classification* example
2. Extract the cortex of the left hemisphere with *nighres.brain.extract_brain_region()*
3. Cortical reconstruction with CRUISE *nighres.cortex.cruise_cortex_extraction()*[1]
4. Anatomical depth estimation trough *nighres.laminar.volumetric_layering()*[2]

Important note: this example assumes you have run the tissue classification example first (example_tissue_classification.py) Import and point to previous example ———————————— First we import `nighres` and the `os` module to set the output directory Make sure to run this file in a directory you have write access to, or change the `out_dir` variable below.

[1] Han et al (2004) CRUISE: Cortical Reconstruction Using Implicit Surface Evolution, NeuroImage, vol. 23, pp. 997–1012.
[2] Waehnert et al (2014) Anatomically motivated modeling of cortical laminae. DOI: 10.1016/j.neuroimage.2013.03.078

```
import nighres
import os

in_dir = os.path.join(os.getcwd(), 'nighres_examples/tissue_classification')
out_dir = os.path.join(os.getcwd(),
                       'nighres_examples/cortical_depth_estimation')
```

We also try to import Nilearn plotting functions. If Nilearn is not installed, plotting will be skipped.

```
skip_plots = False
try:
    from nilearn import plotting
except ImportError:
    skip_plots = True
    print('Nilearn could not be imported, plotting will be skipped')
```

Now we pull the MGDM results from previous example

```
segmentation = os.path.join(in_dir, 'sub001_sess1_mgdm-seg.nii.gz')
boundary_dist = os.path.join(in_dir, 'sub001_sess1_mgdm-dist.nii.gz')
max_labels = os.path.join(in_dir, 'sub001_sess1_mgdm-lbls.nii.gz')
max_probas = os.path.join(in_dir, 'sub001_sess1_mgdm-mems.nii.gz')

if not (os.path.isfile(segmentation) and os.path.isfile(boundary_dist)
        and os.path.isfile(max_labels) and os.path.isfile(max_probas)) :
    print('This example builds upon the example_tissue_segmentation.py one')
    print('Please run it first')
    exit()
```

2.2.1 Region Extraction

Here we pull from the MGDM output the needed regions for cortical reconstruction: the GM cortex ('region'), the underlying WM (with filled subcortex and ventricles, 'inside') and the surrounding CSF (with masked regions, 'background')

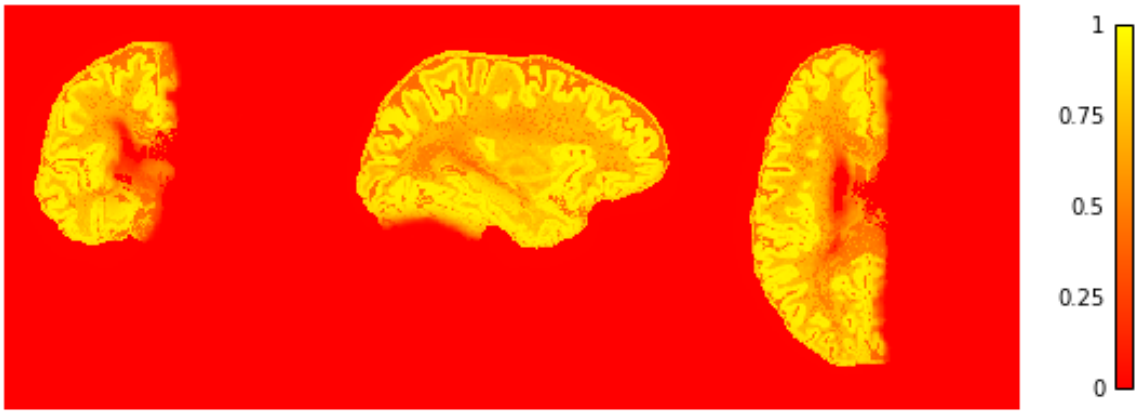
```
cortex = nighres.brain.extract_brain_region(segmentation=segmentation,
                                            levelset_boundary=boundary_dist,
                                            maximum_membership=max_probas,
                                            maximum_label=max_labels,
                                            extracted_region='left_cerebrum',
                                            save_data=True,
                                            file_name='sub001_sess1_left_cerebrum',
                                            output_dir=out_dir)
```

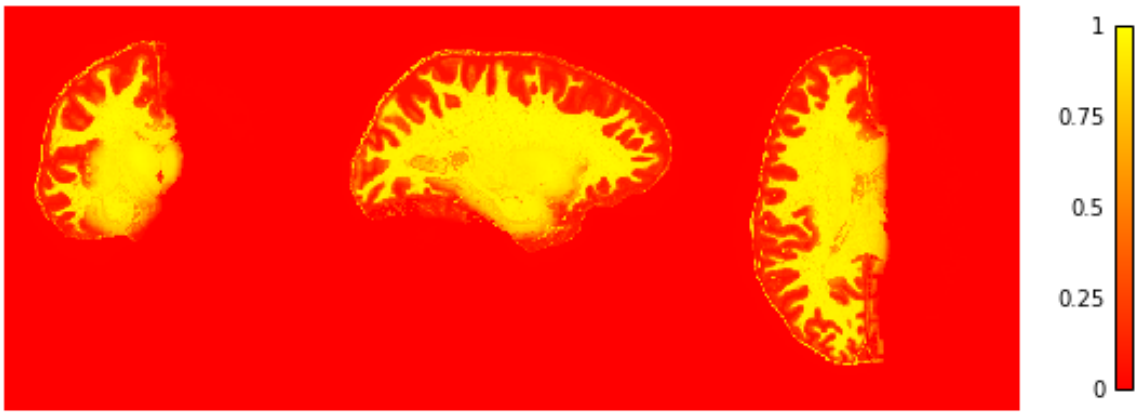
Tip: in Nighres functions that have several outputs return a dictionary storing the different outputs. You can find the keys in the docstring by typing `nighres.brain.mp2rage_extract_brain_region?` or list them with `cortex.keys()`

To check if the extraction worked well we plot the GM and WM probabilities. You can also open the images stored in `out_dir` in your favourite interactive viewer and scroll through the volume.

Like Nilearn, we use Nibabel SpatialImage objects to pass data internally. Therefore, we can directly plot the outputs using Nilearn plotting functions .

```
if not skip_plots:
    plotting.plot_img(cortex['region_proba'],
                      vmin=0, vmax=1, cmap='autumn',  colorbar=True,
                      annotate=False,  draw_cross=False)
    plotting.plot_img(cortex['inside_proba'],
                      vmin=0, vmax=1, cmap='autumn',  colorbar=True,
                      annotate=False,  draw_cross=False)
```





2.2.2 CRUISE cortical reconstruction

Next, we use the extracted data as input for cortex reconstruction with the CRUISE algorithm. CRUISE works with the membership functions as a guide and the WM inside mask as a (topologically spherical) starting point to grow a refined GM/WM boundary and CSF/GM boundary

```
cruise = nighres.cortex.cruise_cortex_extraction(
                        init_image=cortex['inside_mask'],
                        wm_image=cortex['inside_proba'],
                        gm_image=cortex['region_proba'],
```

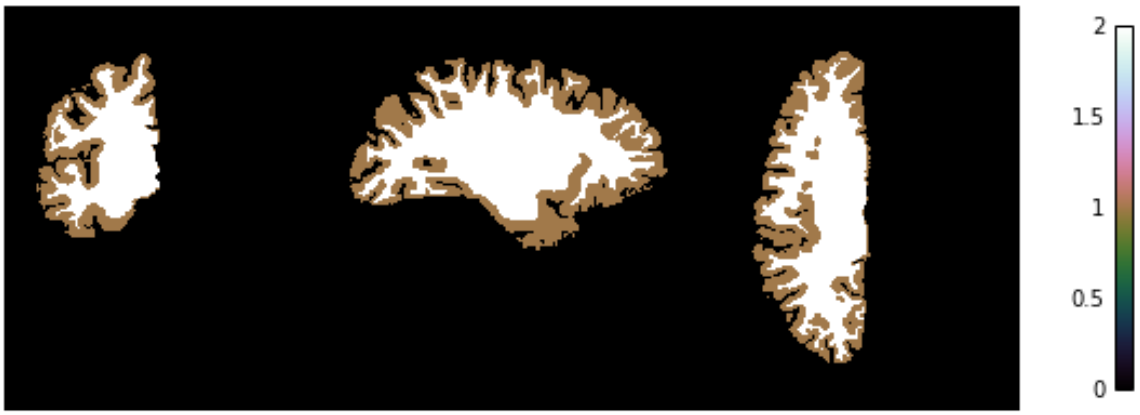
(continues on next page)

(continued from previous page)

```
                        csf_image=cortex['background_proba'],
                        normalize_probabilities=True,
                        save_data=True,
                        file_name="sub001_sess1_left_cerebrum",
                        output_dir=out_dir)
```

Now we look at the topology-constrained segmentation CRUISE created

```
if not skip_plots:
    plotting.plot_img(cruise['cortex'],
                      vmin=0, vmax=2, cmap='cubehelix',  colorbar=True,
                      annotate=False,  draw_cross=False)
```



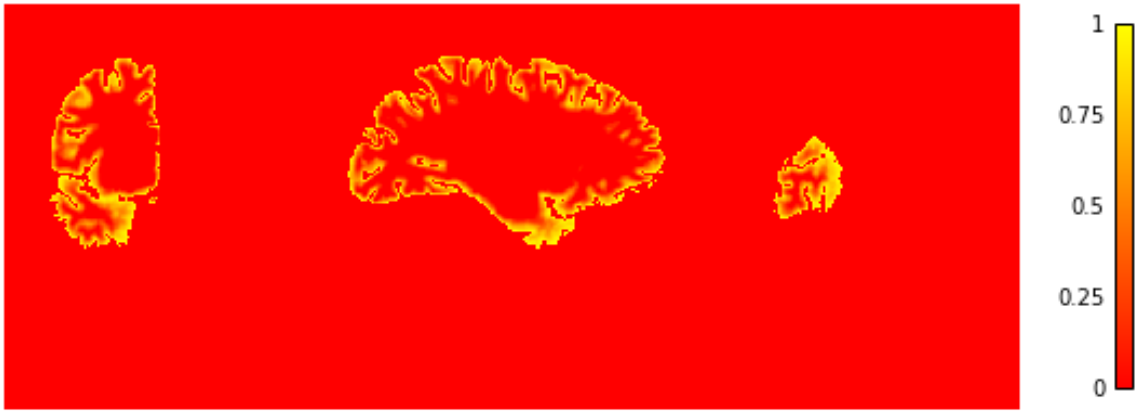
2.2.3 Volumetric layering

Finally, we use the GM/WM boundary (GWB) and CSF/GM boundary (CGB) from CRUISE to compute cortical depth with a volume-preserving technique

```
depth = nighres.laminar.volumetric_layering(
                        inner_levelset=cruise['gwb'],
                        outer_levelset=cruise['cgb'],
                        n_layers=4,
                        save_data=True,
                        file_name="sub001_sess1_left_cerebrum",
                        output_dir=out_dir)
```

Now we look at the laminar depth estimates

```
if not skip_plots:
    plotting.plot_img(depth['depth'],
                      vmin=0, vmax=1, cmap='autumn',  colorbar=True,
                      annotate=False,  draw_cross=False)
```



If the example is not run in a jupyter notebook, render the plots:

```
if not skip_plots:
    plotting.show()
```

2.2.4 References

Total running time of the script: (0 minutes 0.000 seconds)

Note: Click *here* to download the full example code

2.3 Brain co-registration from MP2RAGE data

This example shows how to co-register MP2RAGE data by performing the following steps:

1. **Downloading two open MP2RAGE datasets using** *nighres.data.download_7T_TRT()*
2. **Remove the skull and create a brain mask using** *nighres.brain.mp2rage_skullstripping()*
3. **Co-register non-linearly the brains using** *nighres.registration.embedded_antsreg()*[1]
4. **Deform additional contrasts using** `nighres.registration.apply_deformation()`

2.3.1 Import and download

First we import `nighres` and the `os` module to set the output directory Make sure to run this file in a directory you have write access to, or change the `out_dir` variable below.

[1] Avants et al (2008). Symmetric diffeomorphic image registration with cross-correlation: evaluating automated labeling of elderly and neurodegenerative brain. DOI: 10.1016/j.media.2007.06.004

```
import nighres
import os

in_dir = os.path.join(os.getcwd(), 'nighres_examples/data_sets')
out_dir = os.path.join(os.getcwd(), 'nighres_examples/brain_registration')
```

We also try to import Nilearn plotting functions. If Nilearn is not installed, plotting will be skipped.

```
skip_plots = False
try:
    from nilearn import plotting
except ImportError:
    skip_plots = True
    print('Nilearn could not be imported, plotting will be skipped')
```

Now we download an example MP2RAGE dataset. It is the structural scan of the first subject, first session of the 7T Test-Retest dataset published by Gorgolewski et al (2015)[2].

```
dataset1 = nighres.data.download_7T_TRT(in_dir, subject_id='sub001_sess1')
dataset2 = nighres.data.download_7T_TRT(in_dir, subject_id='sub002_sess1')
```

2.3.2 Skull stripping

First we perform skull stripping. Only the second inversion image is required to calculate the brain mask. But if we input the T1map and T1w image as well, they will be masked for us. We also save the outputs in the `out_dir` specified above and use a subject ID as the base file_name.

```
skullstripping_results1 = nighres.brain.mp2rage_skullstripping(
                                        second_inversion=dataset1['inv2'],
                                        t1_weighted=dataset1['t1w'],
                                        t1_map=dataset1['t1map'],
                                        save_data=True,
                                        file_name='sub001_sess1',
                                        output_dir=out_dir, overwrite=False)

skullstripping_results2 = nighres.brain.mp2rage_skullstripping(
                                        second_inversion=dataset2['inv2'],
                                        t1_weighted=dataset2['t1w'],
                                        t1_map=dataset2['t1map'],
                                        save_data=True,
                                        file_name='sub002_sess1',
                                        output_dir=out_dir, overwrite=False)
```

Tip: in Nighres functions that have several outputs return a dictionary storing the different outputs. You can find the keys in the docstring by typing `nighres.brain.mp2rage_skullstripping?` or list them with `skullstripping_results.keys()`

To check if the skull stripping worked well we plot the brain mask on top of the original image. You can also open the images stored in `out_dir` in your favourite interactive viewer and scroll through the volume.

Like Nilearn, we use Nibabel SpatialImage objects to pass data internally. Therefore, we can directly plot the outputs using Nilearn plotting functions .

[2] Gorgolewski et al (2015). A high resolution 7-Tesla resting-state fMRI test-retest dataset with cognitive and physiological measures. DOI: 10.1038/sdata.2014.54

```
if not skip_plots:
    plotting.plot_roi(skullstripping_results1['brain_mask'], dataset1['t1map'],
                      annotate=False, black_bg=False, draw_cross=False,
                      cmap='autumn')
    plotting.plot_roi(skullstripping_results2['brain_mask'], dataset2['t1w'],
                      annotate=False, black_bg=False, draw_cross=False,
                      cmap='autumn')
```

SyN co-registration Next, we use the masked data as input for co-registration. The T1 maps are used here as they are supposed to be more similar

syn_results = nighres.registration.embedded_syn(source_image=skullstripping_results1['t1map_masked'], target_image=skullstripping_results2['t1map_masked'], coarse_iterations=0, medium_iterations=0, fine_iterations=0, run_rigid_first=True, cost_function='MutualInformation', interpolation='NearestNeighbor', save_data=True, file_name="sub001_sess1", output_dir=out_dir, overwrite=True)

```
syn_results = nighres.registration.embedded_antsreg(
                        source_image=skullstripping_results1['t1map_masked'],
                        target_image=skullstripping_results2['t1map_masked'],
                        run_rigid=True, run_syn=True,
                        rigid_iterations=1000, coarse_iterations=40,
                        medium_iterations=0, fine_iterations=0,
                        cost_function='MutualInformation',
                        interpolation='NearestNeighbor',
                        save_data=True, file_name="sub001_sess1",
                        output_dir=out_dir, overwrite=True)
```

Now we look at the coregistered image that SyN created

```
if not skip_plots:
    plotting.plot_img(syn_results['transformed_source'],
                      annotate=False,  draw_cross=False)
```

2.3.3 Apply deformations to images

Finally, we use the computed deformation to transform other associated images

```
deformed = nighres.registration.apply_coordinate_mappings(
                        image=dataset1['t1map'],
                        mapping1=syn_results['mapping'],
                        save_data=True, file_name="sub001_sess1_t1map",
                        output_dir=out_dir, overwrite=False)

inverse = nighres.registration.apply_coordinate_mappings(
                        image=dataset2['t1w'],
                        mapping1=syn_results['inverse'],
                        save_data=True, file_name="sub002_sess1_t1w",
                        output_dir=out_dir, overwrite=False)
```

Now we look at the coregistered images from applying the deformation

```
if not skip_plots:
    plotting.plot_img(deformed['result'],
                      annotate=False,  draw_cross=False)
    plotting.plot_img(inverse['result'],
                      annotate=False,  draw_cross=False)
```

If the example is not run in a jupyter notebook, render the plots:

```
if not skip_plots:
    plotting.show()
```

2.3.4 References

Total running time of the script: (0 minutes 0.000 seconds)

Note: Click *here* to download the full example code

2.4 Multiatlas Segmentation

This example shows how to perform multi-atlas segmentation based on MP2RAGE data by performing the following steps:

1. **Downloading three open MP2RAGE datasets using** *nighres.data.download_7T_TRT()*
2. **Remove the skull and create a brain mask using** *nighres.brain.mp2rage_skullstripping()*
3. **Atlas-guided tissue classification using MGDMfor first two subjects to** be used as an atlas using *nighres.brain.mgdm_segmentation()*[1]
3. **Co-register non-linearly the atlas brains the the third subject using** `nighres.registration.embedded_syn()`[2]
4. **Deform segmentation labels using** `nighres.registration.apply_deformation()`
5. **Turn individual labels into levelset surfaces using** *nighres.surface.probability_to_levelset()*
6. **Build a final shape average using**

   func *nighres.shape.levelset_fusion*

Important note: this example is both computationally expensive (recomputing everything from basic inputs) and practically pointless (a direct MGDM segmentation or a multi-atlas approach with manually defined labels and more subjects would both be meaningful). This example is only meant as illustration. Import and download ———————- First we import `nighres` and the `os` module to set the output directory Make sure to run this file in a directory you have write access to, or change the `out_dir` variable below.

```
import nighres
import os
import nibabel as nb

in_dir = os.path.join(os.getcwd(), 'nighres_examples/data_sets')
out_dir = os.path.join(os.getcwd(), 'nighres_examples/multiatlas_segmentation')
```

We also try to import Nilearn plotting functions. If Nilearn is not installed, plotting will be skipped.

[1] Bogovic, Prince and Bazin (2013). A multiple object geometric deformable model for image segmentation. DOI: 10.1016/j.cviu.2012.10.006.A

[2] Avants et al (2008). Symmetric diffeomorphic image registration with cross-correlation: evaluating automated labeling of elderly and neurodegenerative brain. DOI: 10.1016/j.media.2007.06.004

```
skip_plots = False
try:
    from nilearn import plotting
except ImportError:
    skip_plots = True
    print('Nilearn could not be imported, plotting will be skipped')
```

Now we download an example MP2RAGE dataset. It is the structural scan of the first subject, first session of the 7T Test-Retest dataset published by Gorgolewski et al (2015)[3].

```
dataset1 = nighres.data.download_7T_TRT(in_dir, subject_id='sub001_sess1')
dataset2 = nighres.data.download_7T_TRT(in_dir, subject_id='sub002_sess1')
dataset3 = nighres.data.download_7T_TRT(in_dir, subject_id='sub003_sess1')
```

2.4.1 Skull stripping

First we perform skull stripping. Only the second inversion image is required to calculate the brain mask. But if we input the T1map and T1w image as well, they will be masked for us. We also save the outputs in the `out_dir` specified above and use a subject ID as the base file_name.

```
skullstripping_results1 = nighres.brain.mp2rage_skullstripping(
                                        second_inversion=dataset1['inv2'],
                                        t1_weighted=dataset1['t1w'],
                                        t1_map=dataset1['t1map'],
                                        save_data=True,
                                        file_name='sub001_sess1',
                                        output_dir=out_dir)

skullstripping_results2 = nighres.brain.mp2rage_skullstripping(
                                        second_inversion=dataset2['inv2'],
                                        t1_weighted=dataset2['t1w'],
                                        t1_map=dataset2['t1map'],
                                        save_data=True,
                                        file_name='sub002_sess1',
                                        output_dir=out_dir)

skullstripping_results3 = nighres.brain.mp2rage_skullstripping(
                                        second_inversion=dataset3['inv2'],
                                        t1_weighted=dataset3['t1w'],
                                        t1_map=dataset3['t1map'],
                                        save_data=True,
                                        file_name='sub003_sess1',
                                        output_dir=out_dir)
```

Tip: in Nighres functions that have several outputs return a dictionary storing the different outputs. You can find the keys in the docstring by typing `nighres.brain.mp2rage_skullstripping?` or list them with `skullstripping_results.keys()`

To check if the skull stripping worked well we plot the brain mask on top of the original image. You can also open the images stored in `out_dir` in your favourite interactive viewer and scroll through the volume.

[3] Gorgolewski et al (2015). A high resolution 7-Tesla resting-state fMRI test-retest dataset with cognitive and physiological measures. DOI: 10.1038/sdata.2014.54

Like Nilearn, we use Nibabel SpatialImage objects to pass data internally. Therefore, we can directly plot the outputs using Nilearn plotting functions .

```
if not skip_plots:
    plotting.plot_roi(skullstripping_results1['brain_mask'], dataset1['t1map'],
                      annotate=False, black_bg=False, draw_cross=False,
                      cmap='autumn')
    plotting.plot_roi(skullstripping_results2['brain_mask'], dataset2['t1w'],
                      annotate=False, black_bg=False, draw_cross=False,
                      cmap='autumn')
    plotting.plot_roi(skullstripping_results3['brain_mask'], dataset3['t1w'],
                      annotate=False, black_bg=False, draw_cross=False,
                      cmap='autumn')
```

2.4.2 MGDM classification

Next, we use MGDM to estimate anatomical labels from subjects 1 and 2

```
mgdm_results1 = nighres.brain.mgdm_segmentation(
                        contrast_image1=skullstripping_results1['t1w_masked'],
                        contrast_type1="Mp2rage7T",
                        contrast_image2=skullstripping_results1['t1map_masked'],
                        contrast_type2="T1map7T",
                        save_data=True, file_name="sub001_sess1",
                        output_dir=out_dir)

mgdm_results2 = nighres.brain.mgdm_segmentation(
                        contrast_image1=skullstripping_results2['t1w_masked'],
                        contrast_type1="Mp2rage7T",
                        contrast_image2=skullstripping_results2['t1map_masked'],
                        contrast_type2="T1map7T",
                        save_data=True, file_name="sub002_sess1",
                        output_dir=out_dir)
```

Now we look at the topology-constrained segmentation MGDM created

```
if not skip_plots:
    plotting.plot_img(mgdm_results1['segmentation'],
                      vmin=1, vmax=50, cmap='cubehelix',  colorbar=True,
                      annotate=False,  draw_cross=False)
    plotting.plot_img(mgdm_results2['segmentation'],
                      vmin=1, vmax=50, cmap='cubehelix',  colorbar=True,
                      annotate=False,  draw_cross=False)
```

2.4.3 SyN co-registration

Next, we use the masked data as input for co-registration. The T1 maps are used here as they are supposed to be more similar

```
syn_results1 = nighres.registration.embedded_antsreg(
                        source_image=skullstripping_results1['t1map_masked'],
                        target_image=skullstripping_results3['t1map_masked'],
                        run_rigid=True, run_affine=True, run_syn=True,
                        coarse_iterations=40,
```

(continues on next page)

(continued from previous page)

```
                          medium_iterations=0, fine_iterations=0,
                          cost_function='MutualInformation',
                          interpolation='NearestNeighbor',
                          save_data=True, file_name="sub001_sess1",
                          output_dir=out_dir)

syn_results2 = nighres.registration.embedded_antsreg(
                          source_image=skullstripping_results2['t1map_masked'],
                          target_image=skullstripping_results3['t1map_masked'],
                          run_rigid=True, run_affine=True, run_syn=True,
                          coarse_iterations=40,
                          medium_iterations=0, fine_iterations=0,
                          cost_function='MutualInformation',
                          interpolation='NearestNeighbor',
                          save_data=True, file_name="sub002_sess1",
                          output_dir=out_dir)
```

Now we look at the coregistered image that SyN created

```
if not skip_plots:
    plotting.plot_img(syn_results1['transformed_source'],
                      annotate=False,  draw_cross=False)
    plotting.plot_img(syn_results2['transformed_source'],
                      annotate=False,  draw_cross=False)
```

2.4.4 Apply deformations to segmentations

We use the computed deformation to transform MGDM segmentations

```
deformed1 = nighres.registration.apply_coordinate_mappings(
                          image=mgdm_results1['segmentation'],
                          mapping1=syn_results1['mapping'],
                          save_data=True, file_name="sub001_sess1_seg",
                          output_dir=out_dir)

deformed2 = nighres.registration.apply_coordinate_mappings(
                          image=mgdm_results2['segmentation'],
                          mapping1=syn_results2['mapping'],
                          save_data=True, file_name="sub002_sess1_seg",
                          output_dir=out_dir)
```

Now we look at the segmentations deformed by SyN

```
if not skip_plots:
    plotting.plot_img(deformed1['result'],
                      annotate=False,  draw_cross=False)
    plotting.plot_img(deformed2['result'],
                      annotate=False,  draw_cross=False)
```

2.4.5 Transform a selected label into levelset representation

We use the deformed MGDM segmentations

```
# label 32 = left caudate
img1 = nighres.io.load_volume(deformed1['result'])
struct1 = nb.Nifti1Image((img1.get_data()==32).astype(float),
                         img1.affine, img1.header)

img2 = nighres.io.load_volume(deformed2['result'])
struct2 = nb.Nifti1Image((img2.get_data()==32).astype(float),
                         img2.affine, img2.header)

levelset1 = nighres.surface.probability_to_levelset(
                      probability_image=struct1,
                      save_data=True, file_name="sub001_sess1_struct",
                      output_dir=out_dir)

levelset2 = nighres.surface.probability_to_levelset(
                      probability_image=struct2,
                      save_data=True, file_name="sub002_sess1_struct",
                      output_dir=out_dir)

final_seg = nighres.shape.levelset_fusion(levelset_images=[levelset1['result'],
                      levelset2['result']],
                      correct_topology=True,
                      save_data=True, file_name="sub003_sess1_struct_seg",
                      output_dir=out_dir, overwrite=True)
```

Now we look at the final segmentation from shape fusion

```
if not skip_plots:
    img = nighres.io.load_volume(levelset1['result'])
    mask = nb.Nifti1Image((img.get_data()<0).astype(bool),
                          img.affine, img.header)
    plotting.plot_roi(mask, dataset3['t1map'],
                      annotate=False, black_bg=False, draw_cross=False,
                      cmap='autumn')

    img = nighres.io.load_volume(levelset2['result'])
    mask = nb.Nifti1Image((img.get_data()<0).astype(bool),
                          img.affine, img.header)
    plotting.plot_roi(mask, dataset3['t1map'],
                      annotate=False, black_bg=False, draw_cross=False,
                      cmap='autumn')

    img = nighres.io.load_volume(final_seg['result'])
    mask = nb.Nifti1Image((img.get_data()<0).astype(bool),
                          img.affine, img.header)
    plotting.plot_roi(mask, dataset3['t1map'],
                      annotate=False, black_bg=False, draw_cross=False,
                      cmap='autumn')
```

If the example is not run in a jupyter notebook, render the plots:

```
if not skip_plots:
    plotting.show()
```

2.4.6 References

Total running time of the script: (0 minutes 0.000 seconds)

CHAPTER 3

Brain

3.1 mp2rage_skullstripping

nighres.brain.**mp2rage_skullstripping**(*second_inversion*, *t1_weighted=None*, *t1_map=None*, *skip_zero_values=True*, *topology_lut_dir=None*, *save_data=False*, *overwrite=False*, *output_dir=None*, *file_name=None*)

MP2RAGE skull stripping

Estimates a brain mask from MRI data acquired with the MP2RAGE sequence. At least a T1-weighted or a T1 map image is required

Parameters

- **second_inversion** (*niimg*) – Second inversion image derived from MP2RAGE sequence
- **t1_weighted** (*niimg*) – T1-weighted image derived from MP2RAGE sequence (also referred to as "uniform" image) At least one of t1_weighted and t1_map is required
- **t1_map** (*niimg*) – Quantitative T1 map image derived from MP2RAGE sequence At least one of t1_weighted and t1_map is required
- **skip_zero_values** (*bool*) – Ignores voxels with zero value (default is True)
- **topology_lut_dir** (*str, optional*) – Path to directory in which topology files are stored (default is stored in TOPOLOGY_LUT_DIR)
- **save_data** (*bool*) – Save output data to file (default is False)
- **overwrite** (*bool*) – Overwrite existing results (default is False)
- **output_dir** (*str, optional*) – Path to desired output directory, will be created if it doesn't exist
- **file_name** (*str, optional*) – Desired base name for output files with file extension (suffixes will be added)

Returns

Dictionary collecting outputs under the following keys (suffix of output files in brackets)

- brain_mask (niimg): Binary brain mask (_strip-mask)
- inv2_masked (niimg): Masked second inversion imamge (_strip-inv2)
- t1w_masked (niimg): Masked T1-weighted image (_strip-t1w)
- t1map_masked (niimg): Masked T1 map (_strip-t1map)

Return type dict

Notes

Original Java module by Pierre-Louis Bazin. Details on the algorithm can be found in[1] and a presentation of the MP2RAGE sequence in[2]

References

3.1.1 Examples using `nighres.brain.mp2rage_skullstripping`

- *Tissue classification from MP2RAGE data*
- *Brain co-registration from MP2RAGE data*
- *Multiatlas Segmentation*

3.2 mp2rage_dura_estimation

`nighres.brain.`**`mp2rage_dura_estimation`**(*second_inversion*, *skullstrip_mask*, *background_distance=5.0*, *output_type='dura_region'*, *save_data=False*, *overwrite=False*, *output_dir=None*, *file_name=None*)

MP2RAGE dura estimation

Filters a MP2RAGE brain image to obtain a probability map of dura matter.

Parameters

- **`second_inversion`** (*`niimg`*) – Second inversion image derived from MP2RAGE sequence
- **`skullstrip_mask`** (*`niimg`*) – Skullstripping mask defining the approximate region including the brain
- **`background_distance`** (*`float`*) – Maximum distance within the mask for dura (default is 5.0 mm)
- **`output_type`** (*{`'dura_region'`, `'boundary'`, `'dura_prior'`, `'bg_prior'`,}*) – 'intens_prior'} Type of output result (default is 'dura_region')
- **`save_data`** (*`bool`*) – Save output data to file (default is False)

[1] Bazin et al. (2014). A computational framework for ultra-high resolution cortical segmentation at 7 Tesla. DOI: 10.1016/j.neuroimage.2013.03.077

[2] Marques et al. (2010). MP2RAGE, a self bias-field corrected sequence for improved segmentation and T1-mapping at high field. DOI: 10.1016/j.neuroimage.2009.10.002

- **overwrite** (*bool*) – Overwrite existing results (default is False)
- **output_dir** (*str, optional*) – Path to desired output directory, will be created if it doesn't exist
- **file_name** (*str, optional*) – Desired base name for output files with file extension (suffixes will be added)

Returns

Dictionary collecting outputs under the following keys (suffix of output files in brackets)

- result (niimg): Dura probability image (_dura-proba)

Return type dict

Notes

Original Java module by Pierre-Louis Bazin. Details on the algorithm can be found in[1] and a presentation of the MP2RAGE sequence in[2]

References

3.3 mgdm_segmentation

nighres.brain.**mgdm_segmentation**(*contrast_image1*, *contrast_type1*, *contrast_image2=None*, *contrast_type2=None*, *contrast_image3=None*, *contrast_type3=None*, *contrast_image4=None*, *contrast_type4=None*, *n_steps=5*, *max_iterations=800*, *topology='wcs'*, *atlas_file=None*, *topology_lut_dir=None*, *adjust_intensity_priors=False*, *compute_posterior=False*, *diffuse_probabilities=False*, *save_data=False*, *overwrite=False*, *output_dir=None*, *file_name=None*)

MGDM segmentation

Estimates brain structures from an atlas for MRI data using a Multiple Object Geometric Deformable Model (MGDM)

Parameters

- **contrast_image1** (*niimg*) – First input image to perform segmentation on
- **contrast_type1** (*str*) – Contrast type of first input image, must be listed as a prior in used atlas(specified in atlas_file). Possible inputs by default are DWIFA3T, DWIMD3T, T1map9T, Mp2rage9T, T1map7T, Mp2rage7T, PV, Filters, T1pv, Mprage3T, T1map3T, Mp2rage3T, HCPT1w, HCPT2w, NormMPRAGE.
- **contrast_image2** (*niimg, optional*) – Additional input image to inform segmentation, must be in the same space as constrast_image1, requires contrast_type2
- **contrast_type2** (*str, optional*) – Contrast type of second input image, must be listed as a prior in used atlas (specified in atlas_file). Possible inputs by default are the same as with parameter contrast_type1 (see above).

[1] Bazin et al. (2014). A computational framework for ultra-high resolution cortical segmentation at 7 Tesla. DOI: 10.1016/j.neuroimage.2013.03.077

[2] Marques et al. (2010). MP2RAGE, a self bias-field corrected sequence for improved segmentation and T1-mapping at high field. DOI: 10.1016/j.neuroimage.2009.10.002

- **contrast_image3** (*niimg, optional*) – Additional input image to inform segmentation, must be in the same space as constrast_image1, requires contrast_type3
- **contrast_type3** (*str, optional*) – Contrast type of third input image, must be listed as a prior in used atlas (specified in atlas_file). Possible inputs by default are the same as with parameter contrast_type1 (see above).
- **contrast_image4** (*niimg, optional*) – Additional input image to inform segmentation, must be in the same space as constrast_image1, requires contrast_type4
- **contrast_type4** (*str, optional*) – Contrast type of fourth input image, must be listed as a prior in used atlas (specified in atlas_file). Possible inputs by default are the same as with parameter contrast_type1 (see above).
- **n_steps** (*int, optional*) – Number of steps for MGDM (default is 5, set to 0 for quick testing of registration of priors, which does not perform true segmentation)
- **max_iterations** (*int, optional*) – Maximum number of iterations per step for MGDM (default is 800, set to 1 for quick testing of registration of priors, which does not perform true segmentation)
- **topology** (*{'wcs', 'no'}, optional*) – Topology setting, choose 'wcs' (well-composed surfaces) for strongest topology constraint, 'no' for no topology constraint (default is 'wcs')
- **atlas_file** (*str, optional*) – Path to plain text atlas file (default is stored in DEFAULT_ATLAS) or atlas name to be searched in ATLAS_DIR
- **topology_lut_dir** (*str, optional*) – Path to directory in which topology files are stored (default is stored in TOPOLOGY_LUT_DIR)
- **adjust_intensity_priors** (*bool*) – Adjust intensity priors based on dataset (default is False)
- **compute_posterior** (*bool*) – Compute posterior probabilities for segmented structures (default is False)
- **diffuse_probabilities** (*bool*) – Regularize probability distribution with a non-linear diffusion scheme (default is False)
- **save_data** (*bool*) – Save output data to file (default is False)
- **overwrite** (*bool*) – Overwrite existing results (default is False)
- **output_dir** (*str, optional*) – Path to desired output directory, will be created if it doesn't exist
- **file_name** (*str, optional*) – Desired base name for output files with file extension (suffixes will be added)

Returns

Dictionary collecting outputs under the following keys (suffix of output files in brackets)

- segmentation (niimg): Hard brain segmentation with topological constraints (if chosen) (_mgdm_seg)
- labels (niimg): Maximum tissue probability labels (_mgdm_lbls)
- memberships (niimg): Maximum tissue probability values, 4D image where the first dimension shows each voxel's highest probability to belong to a specific tissue, the second dimension shows the second highest probability to belong to another tissue etc. (_mgdm_mems)
- distance (niimg): Minimum distance to a segmentation boundary (_mgdm_dist)

Return type dict

Notes

Original Java module by Pierre-Louis Bazin. Algorithm details can be found in[1] and[2]

References

3.3.1 Examples using `nighres.brain.mgdm_segmentation`

- *Tissue classification from MP2RAGE data*
- *Multiatlas Segmentation*

3.4 extract_brain_region

nighres.brain.**extract_brain_region**(*segmentation*, *levelset_boundary*, *maximum_membership*, *maximum_label*, *extracted_region*, *atlas_file=None*, *normalize_probabilities=False*, *estimate_tissue_densities=False*, *partial_volume_distance=1.0*, *save_data=False*, *overwrite=False*, *output_dir=None*, *file_name=None*)

Extract Brain Region

Extracts masks, probability maps and levelset surfaces for specific brain regions and regions from a Multiple Object Geometric Deformable Model (MGDM) segmentation result.

Parameters

- **segmentation** (*niimg*) – Segmentation result from MGDM.
- **levelset_boundary** (*niimg*) – Levelset boundary from MGDM.
- **maximum_membership** (*niimg*) – 4D image of the maximum membership values from MGDM.
- **maximum_label** (*niimg*) – 4D imageof the maximum labels from MGDM.
- **atlas_file** (*str, optional*) – Path to plain text atlas file (default is stored in DEFAULT_ATLAS). or atlas name to be searched in ATLAS_DIR
- **extracted_region** (*{'left_cerebrum', 'right_cerebrum', 'cerebrum',}*) – ‘cerebellum’, ‘cerebellum_brainstem’, ‘subcortex’, ‘tissues(anat)’, ‘tissues(func)’, ‘brain_mask’} Region to be extracted from the MGDM segmentation.
- **normalize_probabilities** (*bool*) – Whether to normalize the output probabilities to sum to 1 (default is False).
- **estimate_tissue_densities** (*bool*) – Wheter to recompute partial volume densities from the probabilites (slow, default is False).
- **partial_volume_distance** (*float*) – Distance in mm to use for tissues densities, if recomputed (default is 1mm).

[1] Bazin et al. (2014). A computational framework for ultra-high resolution cortical segmentation at 7 Tesla. doi: 10.1016/j.neuroimage.2013.03.077

[2] Bogovic et al. (2013). A multiple object geometric deformable model for image segmentation. doi:10.1016/j.cviu.2012.10.006.A

- **save_data** (*bool*) – Save output data to file (default is False)
- **overwrite** (*bool*) – Overwrite existing results (default is False)
- **output_dir** (*str, optional*) – Path to desired output directory, will be created if it doesn't exist
- **file_name** (*str, optional*) – Desired base name for output files with file extension (suffixes will be added)

Returns

Dictionary collecting outputs under the following keys (suffix of output files in brackets, # stands for shorthand names of the different extracted regions, respectively: rcr, lcr, cr, cb, cbs, sub, an, fn)

- region_mask (niimg): Hard segmentation mask of the (GM) region of interest (_xmask-#gm)
- inside_mask (niimg): Hard segmentation mask of the (WM) inside of the region of interest (_xmask-#wm)
- background_mask (niimg): Hard segmentation mask of the (CSF) region background (_xmask-#bg)
- region_proba (niimg): Probability map of the (GM) region of interest (_xproba-#gm)
- inside_proba (niimg): Probability map of the (WM) inside of the region of interest (_xproba-#wm)
- background_proba (niimg): Probability map of the (CSF) region background (_xproba-#bg)
- region_lvl (niimg): Levelset surface of the (GM) region of interest (_xlvl-#gm)
- inside_lvl (niimg): Levelset surface of the (WM) inside of the region of interest (_xlvl-#wm)
- background_lvl (niimg): Levelset surface of the (CSF) region background (_xlvl-#bg)

Return type dict

Notes

Original Java module by Pierre-Louis Bazin.

3.4.1 Examples using `nighres.brain.extract_brain_region`

- *Cortical depth estimation from MGDM segmentation*

3.5 filter_stacking

nighres.brain.**filter_stacking**(*dura_img=None*, *pvcsf_img=None*, *arteries_img=None*, *save_data=False*, *overwrite=False*, *output_dir=None*, *file_name=None*)

Filter stacking

A small utility to combine multiple priors derived from filtering of the CSF partial voluming, arteries, and/or remaining dura mater. The filter priors (in [0,1]) are arranged in a specific way (in increments of 2) that is expected by the Filters contrast type in MGDM

Parameters

- **dura_img** (*niimg, optional*) – Prior for the location of remaining dura mater after skull stripping. At least one prior image is required
- **pvcsf_img** (*niimg, optional*) – Prior for the location of CSF partial voluming (mostly in sulcal regions). At least one prior image is required
- **arteries_img** (*niimg, optional*) – Prior for the location of arteries, visible e.g. in MP2RAGE images. At least one prior image is required
- **save_data** (*bool*) – Save output data to file (default is False)
- **overwrite** (*bool*) – Overwrite existing results (default is False)
- **output_dir** (*str, optional*) – Path to desired output directory, will be created if it doesn't exist
- **file_name** (*str, optional*) – Desired base name for output files with file extension (suffixes will be added)

Returns

Dictionary collecting outputs under the following keys (suffix in brackets)

- result (niimg): Combined image, where only the strongest priors are kept (_bfs-img)

Return type dict

Notes

Original Java module by Pierre-Louis Bazin.

CHAPTER 4

Cortex

4.1 cruise_cortex_extraction

nighres.cortex.**cruise_cortex_extraction**(*init_image*, *wm_image*, *gm_image*, *csf_image*, *vd_image=None*, *data_weight=0.4*, *regularization_weight=0.1*, *max_iterations=500*, *normalize_probabilities=False*, *correct_wm_pv=True*, *wm_dropoff_dist=1.0*, *topology='wcs'*, *topology_lut_dir=None*, *save_data=False*, *overwrite=False*, *output_dir=None*, *file_name=None*)

CRUISE cortex extraction

Segments the cortex from a whole brain segmented data set with the CRUISE method (includes customized partial voluming corrections and the Anatomically-Consistent Enhancement (ACE) of sulcal fundi).

Note that the main input images are generated by the nighres module *nighres.brain.extract_brain_region()*.

Parameters

- **init_image** (*niimg*) – Initial white matter (WM) segmentation mask (binary mask>0 inside WM)
- **wm_image** (*niimg*) – Filled WM probability map (values in [0,1], including subcortical GM and ventricles)
- **gm_image** (*niimg*) – Cortical gray matter (GM) probability map (values in [0,1], highest inside the cortex)
- **csf_image** (*niimg*) – Sulcal cerebro-spinal fluid (CSf) and background probability map (values in [0,1], highest in CSf and masked regions)
- **vd_image** (*niimg, optional*) – Additional probability map of vessels and dura mater to be excluded
- **data_weight** (*float*) – Weighting of probability-based balloon forces in CRUISE (default 0.4, sum of {data_weight,regularization_weight} should be below or equal to 1)

- **regularization_weight** (*float*) – Weighting of curvature regularization forces in CRUISE (default 0.1, sum of {data_weight,regularization_weight} should be below or equal to 1)
- **max_iterations** (*int*) – Maximum number of iterations in CRUISE (default is 500)
- **normalize_probabilities** (*bool*) – Whether to normalize the wm, gm, and csf probabilities (default is False)
- **correct_wm_pv** (*bool*) – Whether to correct for WM partial voluming in gyral crowns (default is True)
- **wm_dropoff_dist** (*float*) – Distance parameter to lower WM probabilities away from current segmentation (default is 1.0 voxel)
- **topology** (*{'wcs', 'no'}*) – Topology setting, choose 'wcs' (well-composed surfaces) for strongest topology constraint, 'no' for no topology constraint (default is 'wcs')
- **topology_lut_dir** (*str*) – Path to directory in which topology files are stored (default is stored in TOPOLOGY_LUT_DIR)
- **save_data** (*bool*) – Save output data to file (default is False)
- **overwrite** (*bool*) – Overwrite existing results (default is False)
- **output_dir** (*str, optional*) – Path to desired output directory, will be created if it doesn't exist
- **file_name** (*str, optional*) – Desired base name for output files with file extension (suffixes will be added)

Returns

Dictionary collecting outputs under the following keys (suffix of output files in brackets)

- cortex (niimg): Hard segmentation of the cortex with labels background=0, gm=1, and wm=2 (_cruise_cortex)
- gwb (niimg): Gray-White matter Boundary (GWB) level set function (_cruise_gwb)
- cgb (niimg): CSF-Gray matter Boundary (CGB) level set function (_cruise_cgb)
- avg (niimg): Central level set function, obtained as geometric average of GWB and CGB (*not* the middle depth of the cortex, use volumetric_layering if you want accurate depth measures) (_cruise-avg)
- thickness (niimg): Simple cortical thickness estimate: distance to the GWB and CGB surfaces, in mm (_cruise-thick)
- pwm (niimg): Optimized WM probability, including partial volume and distant values correction (_cruise-pwm)
- pgm (niimg): Optimized GM probability, including CSF sulcal ridges correction (_cruise_pgm)
- pcsf (niimg): Optimized CSF probability, including sulcal ridges and vessel/dura correction (_cruise-pwm)

Return type dict

Notes

Original algorithm by Xiao Han. Java module by Pierre-Louis Bazin. Algorithm details can be found in[1]

References

4.1.1 Examples using `nighres.cortex.cruise_cortex_extraction`

- *Cortical depth estimation from MGDM segmentation*

[1] X. Han, D.L. Pham, D. Tosun, M.E. Rettmann, C. Xu, and J. L. Prince, CRUISE: Cortical Reconstruction Using Implicit Surface Evolution, NeuroImage, vol. 23, pp. 997–1012, 2004

CHAPTER 5

Data

5.1 download_data

nighres.data.**download_7T_TRT**(*data_dir*, *overwrite=False*, *subject_id='sub001_sess1'*)

Downloads the MP2RAGE data from the 7T Test-Retest dataset published by Gorgolewski et al (2015)[1]

Parameters

- **data_dir** (*str*) – Writeable directory in which downloaded files should be stored. A subdirectory called '7T_TRT' will be created in this location.
- **overwrite** (*bool*) – Overwrite existing files in the same exact path (default is False)
- **subject_id** (*'sub001_sess1', 'sub002_sess1', 'sub003_sess1'}*) – Which dataset to download (default is 'sub001_sess1')

Returns

Dictionary with keys pointing to the location of the downloaded files

- inv2 : path to second inversion image
- t1w : path to T1-weighted (uniform) image
- t1map : path to quantitative T1 image

Return type dict

Notes

The full dataset is available at http://openscience.cbs.mpg.de/7t_trt/

[1] Gorgolewski et al (2015). A high resolution 7-Tesla resting-state fMRI test-retest dataset with cognitive and physiological measures. DOI: 10.1038/sdata.2014.

References

5.1.1 Examples using `nighres.data.download_7T_TRT`

- *Tissue classification from MP2RAGE data*
- *Brain co-registration from MP2RAGE data*
- *Multiatlas Segmentation*

CHAPTER 6

Filtering

6.1 total_variation_filtering

nighres.filtering.**total_variation_filtering**(*image*, *mask=None*, *lambda_scale=0.05*, *tau_step=0.125*, *max_dist=0.0001*, *max_iter=500*, *save_data=False*, *overwrite=False*, *output_dir=None*, *file_name=None*)

Total Variation Filtering

Total variation filtering.

Parameters

- **image** (*niimg*) – Input image to filter
- **mask** (*niimg, optional*) – Data mask for processing
- **lambda_scale** (*float, optional*) – Relative intensity scale for total variation smoothing (default is 0.5)
- **tau_step** (*float, optional*) – Internal step parameter (default is 0.125)
- **max_dist** (*float, optional*) – Maximum distance for convergence (default is 1e-4)
- **max_iter** (*int, optional*) – Maximum number of iterations (default is 500)
- **save_data** (*bool*) – Save output data to file (default is False)
- **overwrite** (*bool*) – Overwrite existing results (default is False)
- **output_dir** (*str, optional*) – Path to desired output directory, will be created if it doesn't exist
- **file_name** (*str, optional*) – Desired base name for output files with file extension (suffixes will be added)

Returns

Dictionary collecting outputs under the following keys (suffix of output files in brackets)

- filtered (niimg): The filtered image (_tv-img)
- residual (niimg): The image residuals (_tv-res)

Return type dict

Notes

Original Java module by Pierre-Louis Bazin. Algorithm adapted from[1]

References

6.2 filter_ridge_structures

nighres.filtering.**filter_ridge_structures**(*input_image*, *structure_intensity='bright'*, *output_type='probability'*, *use_strict_min_max_filter=True*, *save_data=False*, *overwrite=False*, *output_dir=None*, *file_name=None*)

Filter Ridge Structures

Uses an image filter to make a probabilistic image of ridge structures.

Parameters

- **input_image** (*niimg*) – Image containing structure-of-interest
- **structure_intensity** (*{'bright', 'dark', 'both}*) – Image intensity of structure-of-interest'
- **output_type** (*{'probability','intensity'}*) – Whether the image should be normalized to reflect probabilities
- **use_strict_min_max_filter** (*bool, optional*) – Choose between the more specific recursive ridge filter or a more sensitive bidirectional filter (default is True)
- **save_data** (*bool, optional*) – Save output data to file (default is False)
- **overwrite** (*bool*) – Overwrite existing results (default is False)
- **output_dir** (*str, optional*) – Path to desired output directory, will be created if it doesn't exist
- **file_name** (*str, optional*) – Desired base name for output files with file extension (suffixes will be added)

Returns

Dictionary collecting outputs under the following keys (suffix of output files in brackets)

- ridge_structure_image: Image that reflects the presensence of ridges in the image (_rdg-img)

Return type dict

[1] Chambolle (2004). An Algorithm for Total Variation Minimization and Applications. doi:10.1023/B:JMIV.0000011325.36760.1e

Notes

Original Java module by Pierre-Louis Bazin.

6.3 recursive_ridge_diffusion

nighres.filtering.**recursive_ridge_diffusion**(*input_image*, *ridge_intensities*, *ridge_filter*, *surface_levelset=None*, *orientation='undefined'*, *loc_prior=None*, *min_scale=0*, *max_scale=3*, *diffusion_factor=1.0*, *similarity_scale=0.1*, *max_iter=100*, *max_diff=0.001*, *save_data=False*, *overwrite=False*, *output_dir=None*, *file_name=None*)

Recursive Ridge Diffusion

Extracts planar of tubular structures across multiple scales, with an optional directional bias.

Parameters

- **input_image** (*niimg*) – Input image
- **ridge_intensities** (*{'bright','dark','both'}*) – Which intensities to consider for the filtering
- **ridge_filter** (*{'2D','1D','0D'}*) – Whether to filter for 2D ridges, 1D vessels, or 0D holes
- **surface_levelset** (*niimg, optional*) – Level set surface to restrict the orientation of the detected features
- **orientation** (*{'undefined','parallel','orthogonal'}*) – The orientation of features to keep with regard to the surface or its normal
- **loc_prior** (*niimg, optional*) – Location prior image to restrict the search for features
- **min_scale** (*int*) – Minimum scale (in voxels) to look for features (default is 0)
- **max_scale** (*int*) – Maximum scale (in voxels) to look for features (default is 3)
- **diffusion_factor** (*float*) – Scaling factor for the diffusion weighting in [0,1] (default is 1.0)
- **similarity_scale** (*float*) – Scaling of the similarity function as a factor of intensity range
- **max_iter** (*int*) – Maximum number of diffusion iterations
- **max_diff** (*int*) – Maximum difference to stop the diffusion
- **save_data** (*bool*) – Save output data to file (default is False)
- **overwrite** (*bool*) – Overwrite existing results (default is False)
- **output_dir** (*str, optional*) – Path to desired output directory, will be created if it doesn't exist
- **file_name** (*str, optional*) – Desired base name for output files with file extension (suffixes will be added)

Returns

Dictionary collecting outputs under the following keys (suffix of output files in brackets)

- filter (niimg): raw filter response (_rrd-filter)
- propagation (niimg): propagated probabilistic response after diffusion (_rrd-propag)
- scale (niimg): scale of the detection filter (_rrd-scale)
- ridge_dir (niimg): estimated local ridge direction (_rrd-dir)
- ridge_pv (niimg): ridge partial volume map, taking size into account (_rrd-pv)
- ridge_size (niimg): estimated size of each detected component (rrd-size)

Return type dict

Notes

Original Java module by Pierre-Louis Bazin. Extension of the recursive ridge filter in[1].

References

[1] Bazin et al (2016), Vessel segmentation from quantitative susceptibility maps for local oxygenation venography, Proc ISBI.

CHAPTER 7

Intensity

7.1 mp2rage_t1_mapping

nighres.intensity.**mp2rage_t1_mapping**(*first_inversion*, *second_inversion*, *inversion_times*, *flip_angles*, *inversion_TR*, *excitation_TR*, *N_excitations*, *efficiency=0.96*, *correct_B1=False*, *B1_map=None*, *scale_phase=True*, *save_data=False*, *overwrite=False*, *output_dir=None*, *file_name=None*)

MP2RAGE T1 mapping

Estimate T1/R1 by a look-up table method adapted from[1]

Parameters

- **first_inversion** (*[niimg]*) – List of {magnitude, phase} images for the first inversion
- **second_inversion** (*[niimg]*) – List of {magnitude, phase} images for the second inversion
- **inversion_times** (*[float]*) – List of {first, second} inversion times, in seconds
- **flip_angles** (*[float]*) – List of {first, second} flip angles, in degrees
- **inversion_TR** (*float*) – Inversion repetition time, in seconds
- **excitation_TR** (*[float]*) – List of {first,second} repetition times,in seconds
- **N_excitations** (*int*) – Number of excitations
- **efficiency** (*float*) – Inversion efficiency (default is 0.96)
- **correct_B1** (*bool*) – Whether to correct for B1 inhomogeneities (default is False)
- **B1_map** (*niimg*) – Computed B1 map

[1] Marques, Kober, Krueger, van der Zwaag, Van de Moortele, Gruetter (2010) MP2RAGE, a self bias-field corrected sequence for improved segmentation and T1-mapping at high field. doi: 10.1016/j.neuroimage.2009.10.002.

- **scale_phase** (*bool*) – Whether to rescale the phase image in [0,2PI] or to assume it is already in radians
- **save_data** (*bool*) – Save output data to file (default is False)
- **overwrite** (*bool*) – Overwrite existing results (default is False)
- **output_dir** (*str, optional*) – Path to desired output directory, will be created if it doesn't exist
- **file_name** (*str, optional*) – Desired base name for output files with file extension (suffixes will be added)

Returns

Dictionary collecting outputs under the following keys (suffix of output files in brackets)

- t1 (niimg): Map of estimated T1 times (_qt1map-t1)
- r1 (niimg): Map of estimated R1 relaxation rate (_qt1map-r1)
- uni (niimg): Estimated PD weighted image at TE=0 (_qt1map-uni)

Return type dict

Notes

Original Java module by Pierre-Louis Bazin.

References

7.2 flash_t2s_fitting

nighres.intensity.**flash_t2s_fitting**(*image_list*, *te_list*, *save_data=False*, *overwrite=False*, *output_dir=None*, *file_name=None*)

FLASH T2* fitting

Estimate T2*/R2* by linear least squares fitting in log space.

Parameters

- **image_list** (*[niimg]*) – List of input images to fit the T2* curve
- **te_list** (*[float]*) – List of input echo times (TE)
- **save_data** (*bool, optional*) – Save output data to file (default is False)
- **overwrite** (*bool, optional*) – Overwrite existing results (default is False)
- **output_dir** (*str, optional*) – Path to desired output directory, will be created if it doesn't exist
- **file_name** (*str, optional*) – Desired base name for output files with file extension (suffixes will be added)

Returns

Dictionary collecting outputs under the following keys (suffix of output files in brackets)

- t2s (niimg): Map of estimated T2* times (_qt2fit-t2s)
- r2s (niimg): Map of estimated R2* relaxation rate (_qt2fit-r2s)

- s0 (niimg): Estimated PD weighted image at TE=0 (_qt2fit-s0)
- residuals (niimg): Estimated residuals between input and estimated echoes (_qt2fit-err)

Return type dict

Notes

Original Java module by Pierre-Louis Bazin.

7.3 background_estimation

nighres.intensity.**background_estimation**(*image*, *distribution='exponential'*, *ratio=0.001*, *skip_zero=True*, *iterate=True*, *dilate=0*, *threshold=0.5*, *save_data=False*, *overwrite=False*, *output_dir=None*, *file_name=None*)

Background Estimation

Estimates image background by robustlyfitting various distribution models

Parameters

- **image** (*niimg*) – Input image
- **distribution** (*{'exponential','half-normal'}*) – Distribution model to use for the background noise
- **ratio** (*float, optional*) – Robustness ratio for estimating image intensities
- **skip_zero** (*bool, optional*) – Whether to consider or skip zero values
- **iterate** (*bool, optional*) – Whether to run an iterative estimation (preferred, but sometimes unstable)
- **dilate** (*int, optional*) – Number of voxels to dilate or erode in the final mask
- **save_data** (*bool*) – Save output data to file (default is False)
- **overwrite** (*bool*) – Overwrite existing results (default is False)
- **output_dir** (*str, optional*) – Path to desired output directory, will be created if it doesn't exist
- **file_name** (*str, optional*) – Desired base name for output files with file extension (suffixes will be added)

Returns

Dictionary collecting outputs under the following keys (suffix of output files in brackets)

- masked (niimg): The background-masked input image
- proba (niimg): The probability map of the foreground
- mask (niimg): The mask of the foreground

Return type dict

Notes

Original Java module by Pierre-Louis Bazin.

7.4 intensity_propagation

nighres.intensity.**intensity_propagation**(*image*, *mask=None*, *combine='mean'*, *distance_mm=5.0*, *target='zero'*, *scaling=1.0*, *save_data=False*, *overwrite=False*, *output_dir=None*, *file_name=None*)

Intensity Propogation

Propagates the values inside the mask (or non-zero) into the neighboring voxels

Parameters

- **image** (*niimg*) – Input image
- **mask** (*niimg, optional*) – Data mask to specify acceptable seeding regions
- **combine** (*{'min','mean','max'}, optional*) – Propagate using the mean (default), max or min data from neighboring voxels
- **distance_mm** (*float, optional*) – Distance for the propagation (note: this algorithm will be slow for large distances)
- **target** (*{'zero','mask','lower','higher'}, optional*) – Propagate into zero (default), masked out, lower or higher neighboring voxels
- **scaling** (*float, optional*) – Multiply the propagated values by a factor <=1 (default is 1)
- **save_data** (*bool*) – Save output data to file (default is False)
- **overwrite** (*bool*) – Overwrite existing results (default is False)
- **output_dir** (*str, optional*) – Path to desired output directory, will be created if it doesn't exist
- **file_name** (*str, optional*) – Desired base name for output files with file extension (suffixes will be added)

Returns

Dictionary collecting outputs under the following keys (suffix of output files in brackets)

- result (niimg): The propagated intensity image

Return type dict

Notes

Original Java module by Pierre-Louis Bazin.

7.5 lcpca_denoising

nighres.intensity.**lcpca_denoising**(*image_list*, *phase_list=None*, *ngb_size=4*, *stdev_cutoff=1.05*, *min_dimension=0*, *max_dimension=-1*, *unwrap=True*, *save_data=False*, *overwrite=False*, *output_dir=None*, *file_names=None*)

LCPCA denoising

Denoise multi-contrast data with a local complex-valued PCA-based method

Parameters

- **image_list** (*[niimg]*) – List of input images to denoise
- **phase_list** (*[niimg], optional*) – List of input phase to denoise (order must match that of image_list)
- **ngb_size** (*int, optional*) – Size of the local PCA neighborhood, to be increased with number of inputs (default is 4)
- **stdev_cutoff** (*float, optional*) – Factor of local noise level to remove PCA components. Higher values remove more components (default is 1.05)
- **min_dimension** (*int, optional*) – Minimum number of kept PCA components (default is 0)
- **max_dimension** (*int, optional*) – Maximum number of kept PCA components (default is -1 for all components)
- **unwrap** (*bool, optional*) – Whether to unwrap the phase data of keep it as is, assuming radians (default is True)
- **save_data** (*bool*) – Save output data to file (default is False)
- **overwrite** (*bool*) – Overwrite existing results (default is False)
- **output_dir** (*str, optional*) – Path to desired output directory, will be created if it doesn't exist
- **file_names** (*[str], optional*) – Desired base names for output files with file extension (suffixes will be added)

Returns

Dictionary collecting outputs under the following keys (suffix of output files in brackets)

- denoised ([niimg]): The list of denoised input images (_lcpca_den)
- dimensions (niimg): Map of the estimated local dimensions (_lcpca_dim)
- residuals (niimg): Estimated residuals between input and denoised images (_lcpca_err)

Return type dict

Notes

Original Java module by Pierre-Louis Bazin. Algorithm adapted from[1] with a different approach to set the adaptive noise threshold and additional processing to handle the phase data.

References

7.6 phase_unwrapping

nighres.intensity.**phase_unwrapping**(*image*, *mask=None*, *nquadrants=3*, *tv_flattening=False*, *tv_scale=0.5*, *save_data=False*, *overwrite=False*, *output_dir=None*, *file_name=None*)

Fast marching phase unwrapping

Fast marching method for unwrapping phase images, based on _[1]

[1] Manjon, Coupe, Concha, Buades, Collins, Robles (2013). Diffusion Weighted Image Denoising Using Overcomplete Local PCA doi:10.1371/journal.pone.0073021

Parameters

- **image** (*niimg*) – Input phase image to unwrap
- **mask** (*niimg, optional*) – Data mask to specify acceptable seeding regions
- **nquadrants** (*int, optional*) – Number of image quadrants to use (default is 3)
- **tv_flattening** (*bool, optional*) – Whether or not to run a post-processing step to remove background phase variations with a total variation filter (default is False)
- **tv_scale** (*float, optional*) – Relative intensity scale for the TV filter (default is 0.5)
- **save_data** (*bool*) – Save output data to file (default is False)
- **overwrite** (*bool*) – Overwrite existing results (default is False)
- **output_dir** (*str, optional*) – Path to desired output directory, will be created if it doesn't exist
- **file_name** (*str, optional*) – Desired base name for output files with file extension (suffixes will be added)

Returns

Dictionary collecting outputs under the following keys (suffix of output files in brackets)

- result (niimg): The unwrapped image rescaled in radians

Return type dict

Notes

Original Java module by Pierre-Louis Bazin. Algorithm adapted from[1] with additional seeding in multiple image quadrants to reduce the effects of possible phase singularities

References

[1] Abdul-Rahman, Gdeisat, Burton and Lalor. Fast three-dimensional phase-unwrapping algorithm based on sorting by reliability following a non-continuous path. doi: 10.1117/12.611415

CHAPTER 8

Input/Output

8.1 io_volume

nighres.io.**load_volume**(*volume*)
Load volumetric data into a Nibabel SpatialImage

Parameters **volume** (*niimg*) – Volumetric data to be loaded, can be a path to a file that nibabel can load, or a Nibabel SpatialImage

Returns **image**

Return type Nibabel SpatialImage

Notes

Originally created as part of Laminar Python [1]_ .

References

8.1.1 Examples using `nighres.io.load_volume`

- *Multiatlas Segmentation*

nighres.io.**save_volume**(*filename*, *volume*, *dtype='float32'*, *overwrite_file=True*)
Save volumetric data that is a Nibabel SpatialImage to a file

Parameters

- **filename** (*str*) – Full path and filename under which volume should be saved. The extension determines the file format (must be supported by Nibabel)
- **volume** (*Nibabel SpatialImage*) – Volumetric data to be saved
- **dtype** (*str, optional*) – Datatype in which volumetric data should be stored (default is float32)

- **overwrite_file** (*bool, optional*) – Overwrite existing files (default is True)

Notes

Originally created as part of Laminar Python [1]_ .

References

8.2 io_mesh

nighres.io.**load_mesh_geometry**(*surf_mesh*)

Load a mesh geometry into a dictionary with entries "points" and "faces"

Parameters **surf_mesh** – Mesh geometry to be loaded, can be a path to a file (currently supported formats are freesurfer geometry formats, gii and ASCII-coded vtk, ply or obj) or a dictionary with the keys "points" and "faces"

Returns Dictionary with a numpy array with key "points" for a Numpy array of the x-y-z coordinates of the mesh vertices and key "faces" for a Numpy array of the the indices (into points) of the mesh faces

Return type dict

Notes

Originally created as part of Laminar Python [1]_

References

nighres.io.**save_mesh_geometry**(*filename*, *surf_dict*)

Saves surface mesh geometry to file

Parameters

- **filename** (*str*) – Full path and filename under which surfaces data should be saved. The extension determines the file format. Currently supported are freesurfer geometry formats, gii and ASCII-coded vtk, obj, ply'
- **surf_dict** (*dict*) – Surface mesh geometry to be saved. Dictionary with a numpy array with key "points" for a Numpy array of the x-y-z coordinates of the mesh vertices and key "faces2 for a Numpy array of the the indices (into points) of the mesh faces

Notes

Originally created as part of Laminar Python [1]_

References

nighres.io.**load_mesh_data**(*surf_data*, *gii_darray=None*)

Loads mesh data into a Numpy array

Parameters

- **surf_data** – Mesh data to be loaded, can be a Numpy array or a path to a file. Currently supported formats are freesurfer data formats (mgz, curv, sulc, thickness, annot, label), nii, gii, ASCII-coded vtk and txt
- **gii_darray** (*int, optional*) – Index of gii data array to load (default is to load all)

Returns Numpy array containing the data

Return type np.ndarray

Notes

Originally created as part of Laminar Python [1]_

References

nighres.io.**save_mesh_data**(*filename*, *surf_data*)

Saves surface data that is a Numpy array to file

Parameters

- **filename** (*str*) – Full path and filename under which surfaces data should be saved. The extension determines the file format. Currently supported are freesurfer formats curv, thickness, sulc and ASCII-coded txt'
- **surf_data** (*np.ndarray*) – Surface data to be saved

Notes

Originally created as part of Laminar Python [1]_

References

CHAPTER 9

Laminar

9.1 volumetric_layering

nighres.laminar.**volumetric_layering**(*inner_levelset*, *outer_levelset*, *n_layers=4*, *topology_lut_dir=None*, *save_data=False*, *overwrite=False*, *output_dir=None*, *file_name=None*)

Equivolumetric layering of the cortical sheet.

Parameters

- **inner_levelset** (*niimg*) – Levelset representation of the inner surface, typically GM/WM surface
- **outer_levelset** (*niimg*) – Levelset representation of the outer surface, typically GM/CSF surface
- **n_layers** (*int, optional*) – Number of layers to be created (default is 10)
- **topology_lut_dir** (*str, optional*) – Path to directory in which topology files are stored (default is stored in TOPOLOGY_LUT_DIR)
- **save_data** (*bool*) – Save output data to file (default is False)
- **overwrite** (*bool*) – Overwrite existing results (default is False)
- **output_dir** (*str, optional*) – Path to desired output directory, will be created if it doesn't exist
- **file_name** (*str, optional*) – Desired base name for output files with file extension (suffixes will be added)

Returns

Dictionary collecting outputs under the following keys (suffix of output files in brackets)

- depth (niimg): Continuous depth from 0 (inner surface) to 1 (outer surface) (_layering-depth)

- layers (niimg): Discrete layers from 1 (bordering inner surface) to n_layers (bordering outer surface) (_layering-layers)
- boundaries (niimg): Levelset representations of boundaries between all layers in 4D (_layering-boundaries)

Return type dict

Notes

Original Java module by Miriam Waehnert, Pierre-Louis Bazin and Juliane Dinse. Algorithm details can be found in[1]

References

9.1.1 Examples using `nighres.laminar.volumetric_layering`

- *Cortical depth estimation from MGDM segmentation*

9.2 profile_sampling

nighres.laminar.**profile_sampling**(*profile_surface_image*, *intensity_image*, *save_data=False*, *overwrite=False*, *output_dir=None*, *file_name=None*)

Sampling data on multiple intracortical layers

Parameters

- **profile_surface_image** (*niimg*) – 4D image containing levelset representations of different intracortical surfaces on which data should be sampled
- **intensity_image** (*niimg*) – Image from which data should be sampled
- **save_data** (*bool*) – Save output data to file (default is False)
- **overwrite** (*bool*) – Overwrite existing results (default is False)
- **output_dir** (*str, optional*) – Path to desired output directory, will be created if it doesn't exist
- **file_name** (*str, optional*) – Desired base name for output files with file extension (suffixes will be added)

Returns

Dictionary collecting outputs under the following keys (suffix of output files in brackets)

- result (niimg): 4D profile image , where the 4th dimension represents the profile for each voxel (_lps-data)

Return type dict

Notes

Original Java module by Pierre-Louis Bazin and Juliane Dinse

[1] Waehnert et al (2014) Anatomically motivated modeling of cortical laminae. DOI: 10.1016/j.neuroimage.2013.03.078

9.3 laminar_iterative_smoothing

nighres.laminar.**laminar_iterative_smoothing**(*profile_surface_image*, *intensity_image*, *fwhm_mm*, *roi_mask_image=None*, *save_data=False*, *overwrite=False*, *output_dir=None*, *file_name=None*)

Smoothing data on multiple intracortical layers

Parameters

- **data_image** (*niimg*) – Image from which data should be sampled
- **profile_surface_image** (*niimg*) – 4D image containing levelset representations of different intracortical surfaces on which data should be sampled
- **fwhm_mm** (*float*) – Full width half maximum distance to use in smoothing (in mm)
- **roi_mask_image** (*niimg, optional*) – Mask image defining a region of interest to restrict the smoothing
- **save_data** (*bool*) – Save output data to file (default is False)
- **overwrite** (*bool*) – Overwrite existing results (default is False)
- **output_dir** (*str, optional*) – Path to desired output directory, will be created if it doesn't exist
- **file_name** (*str, optional*) – Desired base name for output files with file extension (suffixes will be added)

Returns

Dictionary collecting outputs under the following keys (suffix of output files in brackets)

- result (niimg): smoothed intensity image (_lis-smooth)

Return type dict

Notes

Original Java module by Pierre-Louis Bazin

Important: this method assumes isotropic voxels

CHAPTER 10

Microscopy

10.1 mgdm_cells

nighres.microscopy.**mgdm_cells**(*contrast_image1*, *contrast_type1*, *contrast_image2=None*, *contrast_type2=None*, *contrast_image3=None*, *contrast_type3=None*, *stack_dimension='2D'*, *force_weight=0.6*, *curvature_weight=0.3*, *cell_threshold=0.1*, *max_iterations=200*, *min_change=0.0001*, *topology='wcs'*, *topology_lut_dir=None*, *save_data=False*, *overwrite=False*, *output_dir=None*, *file_name=None*)

MGDM cell segmentation

Estimates cell structures using a Multiple Object Geometric Deformable Model (MGDM)

Parameters

- **contrast_image1** (*niimg*) – First input image to perform segmentation on
- **contrast_type1** (*{"centroid-proba", "local-maxima", "foreground-proba",}*) – "image-intensities"} Contrast type of first input image
- **contrast_image2** (*niimg, optional*) – Additional input image to inform segmentation, must be in the same space as constrast_image1, requires contrast_type2
- **contrast_type2** (*str, {"centroid-proba", "local-maxima", "foreground-proba",}*) – "image-intensities"}, optional Contrast type of second input image
- **contrast_image3** (*niimg, optional*) – Additional input image to inform segmentation, must be in the same space as constrast_image1, requires contrast_type3
- **contrast_type3** (*{"centroid-proba", "local-maxima", "foreground-proba",}*) – "image-intensities"}, optional Contrast type of third input image
- **stack_dimension** (*{'2D','3D'}, optional*) – Dimension of the data for processing, either a stack of independent 2D slices or a fully 3D stack

- **max_iterations** (*int, optional*) – Maximum number of iterations per step for MGDM (default is 800, set to 1 for quick testing of registration of priors, which does not perform true segmentation)
- **min_change** (*float, optional*) – Minimum amount of change in the segmentation for MGDM to stop (default is 0.001)
- **force_weight** (*float, optional*) – Forces to drive MGDM to cell boundaries (default is 0.5)
- **curvature_weight** (*float, optional*) – Curvature regularization forces (default is 0.1)
- **cell_threshold** (*float, optional*) – Ratio of lower intensities from the local maximum to be included in a given cell (default is 0.1)
- **topology** (*{'wcs', 'no'}, optional*) – Topology setting, choose 'wcs' (well-composed surfaces) for strongest topology constraint, 'no' for no topology constraint (default is 'wcs')
- **topology_lut_dir** (*str, optional*) – Path to directory in which topology files are stored (default is stored in TOPOLOGY_LUT_DIR)
- **save_data** (*bool, optional*) – Save output data to file (default is False)
- **overwrite** (*bool, optional*) – Overwrite existing results (default is False)
- **output_dir** (*str, optional*) – Path to desired output directory, will be created if it doesn't exist
- **file_name** (*str, optional*) – Desired base name for output files with file extension (suffixes will be added)

Returns

Dictionary collecting outputs under the following keys (suffix of output files in brackets)

- segmentation (niimg): Hard brain segmentation with topological constraints (if chosen) (_mgdmc-seg)
- distance (niimg): Minimum distance to a segmentation boundary (_mgdmc-dist)

Return type dict

Notes

Original Java module by Pierre-Louis Bazin. Algorithm details can be found in[1] and[2]

References

[1] Bogovic, Prince and Bazin (2013). A multiple object geometric deformable model for image segmentation. doi:10.1016/j.cviu.2012.10.006.A

[2] Fan, Bazin and Prince (2008). A multi-compartment segmentation framework with homeomorphic level sets. DOI: 10.1109/CVPR.2008.4587475

CHAPTER 11

Registration

11.1 embedded_antsreg

nighres.registration.**embedded_antsreg**(*source_image*, *target_image*, *run_rigid=False*, *rigid_iterations=1000*, *run_affine=False*, *affine_iterations=1000*, *run_syn=True*, *coarse_iterations=40*, *medium_iterations=50*, *fine_iterations=40*, *cost_function='MutualInformation'*, *interpolation='NearestNeighbor'*, *regularization='Medium'*, *convergence=1e-06*, *ignore_affine=False*, *ignore_header=False*, *save_data=False*, *overwrite=False*, *output_dir=None*, *file_name=None*)

Embedded ANTS Registration

Runs the rigid and/or Symmetric Normalization (SyN) algorithm of ANTs and formats the output deformations into voxel coordinate mappings as used in CBSTools registration and transformation routines.

Parameters

- **source_image** (*niimg*) – Image to register
- **target_image** (*niimg*) – Reference image to match
- **run_rigid** (*bool*) – Whether or not to run a rigid registration first (default is False)
- **rigid_iterations** (*float*) – Number of iterations in the rigid step (default is 1000)
- **run_affine** (*bool*) – Whether or not to run a affine registration first (default is False)
- **affine_iterations** (*float*) – Number of iterations in the affine step (default is 1000)
- **run_syn** (*bool*) – Whether or not to run a SyN registration (default is True)
- **coarse_iterations** (*float*) – Number of iterations at the coarse level (default is 40)
- **medium_iterations** (*float*) – Number of iterations at the medium level (default is 50)

- **fine_iterations** (*float*) – Number of iterations at the fine level (default is 40)
- **cost_function** (*{'CrossCorrelation', 'MutualInformation'}*) – Cost function for the registration (default is 'MutualInformation')
- **interpolation** (*{'NearestNeighbor', 'Linear'}*) – Interpolation for the registration result (default is 'NearestNeighbor')
- **regularization** (*{'Low', 'Medium', 'High'}*) – Regularization preset for the SyN deformation (default is 'Medium')
- **convergence** (*float*) – Threshold for convergence, can make the algorithm very slow (default is convergence)
- **ignore_affine** (*bool*) – Ignore the affine matrix information extracted from the image header (default is False)
- **ignore_header** (*bool*) – Ignore the orientation information and affine matrix information extracted from the image header (default is False)
- **save_data** (*bool*) – Save output data to file (default is False)
- **overwrite** (*bool*) – Overwrite existing results (default is False)
- **output_dir** (*str, optional*) – Path to desired output directory, will be created if it doesn't exist
- **file_name** (*str, optional*) – Desired base name for output files with file extension (suffixes will be added)

Returns

Dictionary collecting outputs under the following keys (suffix of output files in brackets)

- transformed_source (niimg): Deformed source image (_ants-def)
- mapping (niimg): Coordinate mapping from source to target (_ants-map)
- inverse (niimg): Inverse coordinate mapping from target to source (_ants-invmap)

Return type dict

Notes

Port of the CBSTools Java module by Pierre-Louis Bazin. The main algorithm is part of the ANTs software by Brian Avants and colleagues[1]. The interfacing with ANTs is performed through Nipype[2]. Parameters have been set to values commonly found in neuroimaging scripts online, but not necessarily optimal.

References

11.1.1 Examples using `nighres.registration.embedded_antsreg`

- *Brain co-registration from MP2RAGE data*
- *Multiatlas Segmentation*

[1] Avants et al (2008), Symmetric diffeomorphic image registration with cross-correlation: evaluating automated labeling of elderly and neurodegenerative brain, Med Image Anal. 12(1):26-41

[2] Gorgolewski et al (2011) Nipype: a flexible, lightweight and extensible neuroimaging data processing framework in python. Front Neuroinform 5. doi:10.3389/fninf.2011.00013

11.2 embedded_antsreg_2d

nighres.registration.**embedded_antsreg_2d**(*source_image*, *target_image*, *run_rigid=False*, *rigid_iterations=1000*, *run_affine=False*, *affine_iterations=1000*, *run_syn=True*, *coarse_iterations=40*, *medium_iterations=50*, *fine_iterations=40*, *cost_function='MutualInformation'*, *interpolation='NearestNeighbor'*, *convergence=1e-06*, *ignore_affine=False*, *ignore_header=False*, *save_data=False*, *overwrite=False*, *output_dir=None*, *file_name=None*)

Embedded ANTS Registration 2D

Runs the rigid and/or Symmetric Normalization (SyN) algorithm of ANTs and formats the output deformations into voxel coordinate mappings as used in CBSTools registration and transformation routines.

Parameters

- **source_image** (*niimg*) – Image to register
- **target_image** (*niimg*) – Reference image to match
- **run_rigid** (*bool*) – Whether or not to run a rigid registration first (default is False)
- **rigid_iterations** (*float*) – Number of iterations in the rigid step (default is 1000)
- **run_affine** (*bool*) – Whether or not to run a affine registration first (default is False)
- **affine_iterations** (*float*) – Number of iterations in the affine step (default is 1000)
- **run_syn** (*bool*) – Whether or not to run a SyN registration (default is True)
- **coarse_iterations** (*float*) – Number of iterations at the coarse level (default is 40)
- **medium_iterations** (*float*) – Number of iterations at the medium level (default is 50)
- **fine_iterations** (*float*) – Number of iterations at the fine level (default is 40)
- **cost_function** (*{'CrossCorrelation', 'MutualInformation'}*) – Cost function for the registration (default is 'MutualInformation')
- **interpolation** (*{'NearestNeighbor', 'Linear'}*) – Interpolation for the registration result (default is 'NearestNeighbor')
- **convergence** (*flaot*) – Threshold for convergence, can make the algorithm very slow (default is convergence)
- **ignore_affine** (*bool*) – Ignore the affine matrix information extracted from the image header (default is False)
- **ignore_header** (*bool*) – Ignore the orientation information and affine matrix information extracted from the image header (default is False)
- **save_data** (*bool*) – Save output data to file (default is False)
- **overwrite** (*bool*) – Overwrite existing results (default is False)
- **output_dir** (*str, optional*) – Path to desired output directory, will be created if it doesn't exist
- **file_name** (*str, optional*) – Desired base name for output files with file extension (suffixes will be added)

Returns

Dictionary collecting outputs under the following keys (suffix of output files in brackets)

- transformed_source (niimg): Deformed source image (_ants-def)
- mapping (niimg): Coordinate mapping from source to target (_ants-map)
- inverse (niimg): Inverse coordinate mapping from target to source (_ants-invmap)

Return type dict

Notes

Port of the CBSTools Java module by Pierre-Louis Bazin. The main algorithm is part of the ANTs software by Brian Avants and colleagues[1]. The interfacing with ANTs is performed through Nipype[2]. Parameters have been set to values commonly found in neuroimaging scripts online, but not necessarily optimal.

References

11.3 generate_coordinate_mapping

nighres.registration.**generate_coordinate_mapping**(*reference_image*, *source_image=None*, *transform_matrix=None*, *invert_matrix=False*, *save_data=False*, *overwrite=False*, *output_dir=None*, *file_name=None*)

Generate coordiante mapping

Generate a coordinate mapping for the image(s) and linear transformation as used in CBSTools registration and transformation routines.

Parameters

- **reference_image** (*niimg*) – Image to generate a coordinate mapping from, listing its X,Y,Z coordinates
- **source_image** (*niimg, optional*) – In case the mapping is from a source and target in different coordinate spaces, this image represents the source space
- **transform_matrix** (*niimg, optional*) – Whether to use a MIPAV formatted transformation matrix to define the mapping
- **invert_matrix** (*bool*) – Whether or not to invert the transformation, if given
- **save_data** (*bool*) – Save output data to file (default is False)
- **overwrite** (*bool*) – Overwrite existing results (default is False)
- **output_dir** (*str, optional*) – Path to desired output directory, will be created if it doesn't exist
- **file_name** (*str, optional*) – Desired base name for output files with file extension (suffixes will be added)

[1] Avants et al (2008), Symmetric diffeomorphic image registration with cross-correlation: evaluating automated labeling of elderly and neurodegenerative brain, Med Image Anal. 12(1):26-41

[2] Gorgolewski et al (2011) Nipype: a flexible, lightweight and extensible neuroimaging data processing framework in python. Front Neuroinform 5. doi:10.3389/fninf.2011.00013

Returns

Dictionary collecting outputs under the following keys (suffix of output files in brackets)

- result (niimg): Coordinate mapping of the reference image (_coord-map)

Return type dict

Notes

Port of the CBSTools Java module by Pierre-Louis Bazin. Currently the transformation amtrix follows the MIPAV conventions.

11.4 apply_coordinate_mappings

nighres.registration.**apply_coordinate_mappings**(*image*, *mapping1*, *mapping2=None*, *mapping3=None*, *mapping4=None*, *interpolation='nearest'*, *padding='closest'*, *save_data=False*, *overwrite=False*, *output_dir=None*, *file_name=None*)

Apply a coordinate mapping (or a succession of coordinate mappings) to a 3D or 4D image.

Parameters

- **image** (*niimg*) – Image to deform
- **mapping1** (*niimg*) – First coordinate mapping to apply
- **mapping2** (*niimg, optional*) – Second coordinate mapping to apply
- **mapping3** (*niimg, optional*) – Third coordinate mapping to apply
- **mapping4** (*niimg, optional*) – Fourth coordinate mapping to apply
- **interpolation** (*{'nearest', 'linear'}*) – Interpolation method (default is 'nearest')
- **padding** (*{'closest', 'zero', 'max'}*) – Image padding method (default is 'closest')
- **save_data** (*bool*) – Save output data to file (default is False)
- **overwrite** (*bool*) – Overwrite existing results (default is False)
- **output_dir** (*str, optional*) – Path to desired output directory, will be created if it doesn't exist
- **file_name** (*str, optional*) – Desired base name for output files with file extension (suffixes will be added)

Returns

Dictionary collecting outputs under the following keys (suffix of output files in brackets)

- result (niimg): Result image (_def-img)

Return type dict

Notes

Original Java module by Pierre-Louis Bazin

11.4.1 Examples using `nighres.registration.apply_coordinate_mappings`

- *Brain co-registration from MP2RAGE data*
- *Multiatlas Segmentation*

11.5 simple_align

nighres.registration.**simple_align**(*source_image*, *target_image*, *copy_header=False*, *align_center=False*, *rescale=False*, *data_type='intensity'*, *ignore_affine=False*, *ignore_header=False*, *save_data=False*, *overwrite=False*, *output_dir=None*, *file_name=None*)

Simple alignment routines

Simple routines to align image headers (image data is unchanged)

Parameters

- **source_image** (*niimg*) – Image to align
- **target_image** (*niimg*) – Reference image to match
- **copy_header** (*bool*) – To copy the target header to the source (default is False)
- **align_center** (*bool*) – To align the source center of mass to the target (default is False)
- **rescale** (*bool*) – To rescale the source to the volume of the target (default is False)
- **data_type** (*{'intensity','nonzero','boundingbox'}*) – The type of datato consider for alignment (default is 'intensity')
- **ignore_affine** (*bool*) – Ignore the affine matrix information extracted from the image header (default is False)
- **ignore_header** (*bool*) – Ignore the orientation information and affine matrix information extracted from the image header (default is False)
- **save_data** (*bool*) – Save output data to file (default is False)
- **overwrite** (*bool*) – Overwrite existing results (default is False)
- **output_dir** (*str, optional*) – Path to desired output directory, will be created if it doesn't exist
- **file_name** (*str, optional*) – Desired base name for output files with file extension (suffixes will be added)

Returns

Dictionary collecting outputs under the following keys (suffix of output files in brackets)

- result (niimg): Aligne source image (_al-img)

Return type dict

Notes

This uses the ANTs/ITK conventions with regard to Nifti headers, for better or for worse. Note that Nibabel conventions, of course, are different.

CHAPTER 12

Shape

12.1 topology_correction

nighres.shape.**topology_correction**(*image*, *shape_type*, *connectivity='wcs'*, *propagation='object->background'*, *minimum_distance=1e-05*, *topology_lut_dir=None*, *save_data=False*, *overwrite=False*, *output_dir=None*, *file_name=None*)

Topology correction

Corrects the topology of a binary image, a probability map or a levelset surface to spherical with a fast marching technique[1].

Parameters

- **image** (*niimg*) – Image representing the shape of interest
- **shape_type** (*{'binary_object','probability_map', 'signed_distance_function'}*) – Which type of image is used as input
- **connectivity** (*{'wcs','6/18','6/26','18/6','26/6'}*) – What connectivity type to use (default is wcs: well-composed surfaces)
- **propagation** (*{'object->background','background->object'}*) – Which direction to use to enforce topology changes (default is 'object->background')
- **minimum_distance** (*float, optional*) – Minimum distance to impose between successive voxels (default is 1e-5)
- **topology_lut_dir** (*str, optional*) – Path to directory in which topology files are stored (default is stored in TOPOLOGY_LUT_DIR)
- **save_data** (*bool, optional*) – Save output data to file (default is False)
- **overwrite** (*bool, optional*) – Overwrite existing results (default is False)

[1] Bazin and Pham (2007). Topology correction of segmented medical images using a fast marching algorithm doi:10.1016/j.cmpb.2007.08.006

- **output_dir** (*str, optional*) – Path to desired output directory, will be created if it doesn't exist
- **file_name** (*str, optional*) – Desired base name for output files with file extension (suffixes will be added)

Returns

Dictionary collecting outputs under the following keys (suffix of output files in brackets)

- corrected (niimg): Corrected image (output file suffix _tpc-img)
- object (niimg): Corrected binary object (output file suffix _tpc-obj)

Return type dict

Notes

Original Java module by Pierre-Louis Bazin

References

12.2 levelset_fusion

nighres.shape.**levelset_fusion**(*levelset_images*, *correct_topology=True*, *topology_lut_dir=None*, *save_data=False*, *overwrite=False*, *output_dir=None*, *file_name=None*)

Levelset fusion

Creates an average levelset surface representations from a collection of levelset surfaces, with same avearage volume and (optionally) spherical topology

Parameters

- **levelset_images** (*niimg*) – List of levelset images to combine.
- **correct_topology** (*bool, optional*) – Corrects the average shape to ensure correct topology (default is True)
- **topology_lut_dir** (*str, optional*) – Path to directory in which topology files are stored (default is stored in TOPOLOGY_LUT_DIR)
- **save_data** (*bool, optional*) – Save output data to file (default is False)
- **overwrite** (*bool, optional*) – Overwrite existing results (default is False)
- **output_dir** (*str, optional*) – Path to desired output directory, will be created if it doesn't exist
- **file_name** (*str, optional*) – Desired base name for output files with file extension (suffixes will be added)

Returns

Dictionary collecting outputs under the following keys (suffix of output files in brackets)

- result (niimg): Levelset representation of combined surface (_lsf-avg)

Return type dict

Notes

Original Java module by Pierre-Louis Bazin

12.2.1 Examples using `nighres.shape.levelset_fusion`

- *Multiatlas Segmentation*

CHAPTER 13

Statistics

13.1 segmentation_statistics

nighres.statistics.**segmentation_statistics**(*segmentation*, *intensity=None*, *template=None*, *statistics=None*, *output_csv=None*, *atlas=None*, *skip_first=True*, *ignore_zero=True*, *save_data=False*, *overwrite=False*, *output_dir=None*, *file_name=None*)

Segmentation Statistics

Compute various statistics of image segmentations

Parameters

- **segmentation** (*niimg*) – Input segmentation image
- **intensity** (*niimg, optional*) – Input intensity image for intensity-based statistics
- **template** (*niimg, optional*) – Input template segmentation for comparisons
- **statistics** (*{"Voxels", "Volume", "Mean_intensity", "Std_intensity", "10_intensity", "25_intensity", "50_intensity", "75_intensity", "90_intensity", "Volumes", "Dice_overlap", "Jaccard_overlap", "Volume_difference", "False_positives", "False_negatives", "Dilated_Dice_overlap", "Dilated_false_positive", "Dilated_false_negative", "Dilated_false_negative_volume", "Dilated_false_positive_volume", "Detected_clusters", "False_detections", "Cluster_numbers", "Mean_cluster_sizes", "Cluster_maps", "Average_surface_distance", "Average_surface_difference", "Average_squared_surface_distance", "Hausdorff_distance"}*) – Statistics to compute
- **output_csv** (*str, optional*) – File name of the statistics file to generate or expand

- **atlas** (*str, optional*) – File name of an atlas file defining the segmentation labels
- **skip_first** (*bool, optional*) – Whether to skip the first segmentation label (usually representing the background, default is True)
- **ignore_zero** (*bool, optional*) – Whether to ignore zero intensity values in the intensity image (default is True)
- **save_data** (*bool, optional*) – Save output data to file (default is False)
- **overwrite** (*bool, optional*) – Overwrite existing results (default is False)
- **output_dir** (*str, optional*) – Path to desired output directory, will be created if it doesn't exist
- **file_name** (*str, optional*) – Desired base name for output files with file extension (suffixes will be added)

Returns

Dictionary collecting outputs under the following keys (suffix of output files in brackets)

- csv (str): The csv statistics file
- map (niimg): Map of the estimated statistic, if relevant (stat-map)

Return type dict

Notes

Original Java module by Pierre-Louis Bazin.

CHAPTER 14

Surface

14.1 probability_to_levelset

nighres.surface.**probability_to_levelset**(*probability_image*, *save_data=False*, *overwrite=False*, *output_dir=None*, *file_name=None*)

Levelset from probability map

Creates a levelset surface representations from a probabilistic map or a mask. The levelset indicates each voxel's distance to the closest boundary. It takes negative values inside and positive values outside of the object.

Parameters

- **probability_image** (*niimg*) – Probability image to be turned into levelset. Values should be in [0, 1], either a binary mask or defining the boundary at 0.5.
- **save_data** (*bool, optional*) – Save output data to file (default is False)
- **overwrite** (*bool, optional*) – Overwrite existing results (default is False)
- **output_dir** (*str, optional*) – Path to desired output directory, will be created if it doesn't exist
- **file_name** (*str, optional*) – Desired base name for output files with file extension (suffixes will be added)

Returns

Dictionary collecting outputs under the following keys (suffix of output files in brackets)

- result (niimg): Levelset representation of surface (_p2l-surf)

Return type dict

Notes

Original Java module by Pierre-Louis Bazin

14.1.1 Examples using `nighres.surface.probability_to_levelset`

- *Multiatlas Segmentation*

14.2 levelset_to_probability

nighres.surface.**levelset_to_probability**(*levelset_image*, *distance_mm=5*, *save_data=False*, *overwrite=False*, *output_dir=None*, *file_name=None*)

Levelset to probability

Creates a probability map from the distance to a levelset surface representation.

Parameters

- **levelset_image** (*niimg*) – Levelset image to be turned into probabilities
- **distance_mm** (*float, optional*) – Distance used for the range of probabilities in]0,1[
- **save_data** (*bool, optional*) – Save output data to file (default is False)
- **overwrite** (*bool, optional*) – Overwrite existing results (default is False)
- **output_dir** (*str, optional*) – Path to desired output directory, will be created if it doesn't exist
- **file_name** (*str, optional*) – Desired base name for output files with file extension (suffixes will be added)

Returns

Dictionary collecting outputs under the following keys (suffix of output files in brackets)

- result (niimg): Probability map (output file suffix _l2p-proba)

Return type dict

Notes

Ported from original Java module by Pierre-Louis Bazin

14.3 levelset_to_mesh

nighres.surface.**levelset_to_mesh**(*levelset_image*, *connectivity='18/6'*, *level=0.0*, *inclusive=True*, *save_data=False*, *overwrite=False*, *output_dir=None*, *file_name=None*)

Levelset to mesh

Creates a triangulated mesh from the distance to a levelset surface representation using a connectivity-consistent marching cube algorithm.

Parameters

- **levelset_image** (*niimg*) – Levelset image to be turned into probabilities
- **connectivity** (*{"6/18","6/26","18/6","26/6"}, optional*) – Choice of digital connectivity to build the mesh (default is 18/6)

- **level** (*float, optional*) – Value of the levelset function to use as isosurface (default is 0)
- **inclusive** (*bool, optional*) – Whether voxels at the exact 'level' value are inside the isosurface (default is True)
- **save_data** (*bool, optional*) – Save output data to file (default is False)
- **overwrite** (*bool, optional*) – Overwrite existing results (default is False)
- **output_dir** (*str, optional*) – Path to desired output directory, will be created if it doesn't exist
- **file_name** (*str, optional*) – Desired base name for output files with file extension (suffixes will be added)

Returns

Dictionary collecting outputs under the following keys (suffix of output files in brackets)

- result (mesh): Surface mesh dictionary of "points" and "faces" (_l2m-mesh)

Return type dict

Notes

Ported from original Java module by Pierre-Louis Bazin. Original algorithm from[1] and adapted from[2].

References

14.4 surface_mesh_mapping

nighres.surface.**surface_mesh_mapping**(*intensity_image*, *surface_mesh*, *inflated_mesh=None*, *mapping_method='closest_point'*, *save_data=False*, *overwrite=False*, *output_dir=None*, *file_name=None*)

Surface mesh mapping

Maps volumetric data onto a surface mesh. A second mesh with the same graph topology (e.g. an inflated cortical surface) can also be mapped with the same data.

Parameters

- **intensity_image** (*niimg*) – Intensity image to map onto the surface mesh
- **surface_mesh** (*mesh*) – Mesh model of the surface
- **inflated_mesh** (*mesh, optional*) – Mesh model of the inflated surface
- **mapping_method** (*{"closest_point","linear_interp", "highest_value"}, optional*) – Choice of mapping method
- **save_data** (*bool, optional*) – Save output data to file (default is False)
- **overwrite** (*bool, optional*) – Overwrite existing results (default is False)
- **output_dir** (*str, optional*) – Path to desired output directory, will be created if it doesn't exist

[1] Han et al (2003). A Topology Preserving Level Set Method for Geometric Deformable Models doi:
[2] Lucas et al (2010). The Java Image Science Toolkit (JIST) for Rapid Prototyping and Publishing of Neuroimaging Software doi:

- **file_name** (*str, optional*) – Desired base name for output files with file extension (suffixes will be added)

Returns

Dictionary collecting outputs under the following keys (suffix of output files in brackets)

- original (mesh): Surface mesh dictionary of "points" and "faces" (_map-orig)
- inflated (mesh): Surface mesh dictionary of "points" and "faces" (_map-inf)

Return type dict

Notes

Ported from original Java module by Pierre-Louis Bazin

CHAPTER 15

Data handling and formats

Nighres represents data internally using NibabelSpatialImage objects which are refered to as `niimg`.

Much of the input and output functionality has been adopted or inspired from Nilearn's conventions for data handling

Todo: Explanation why this is useful and little example how it works, also mention dictionary outputs

CHAPTER 16

Saving outputs

Each Nighres processing interface allows you to save the outputs by setting the `save_data` parameter to `True`. If this parameter is not specified, it defaults to `False` and the data is returned as a data object (see *Data handling and formats*) but not saved to disk.

If `save_data` is set to True, Nighres applies the following logic:

Output directory

1. If `output_dir` is specified, the data is saved there. In case `output_dir` doesn't exist, it is created
2. If `output_dir` is not specified, Nighres tries to use the location of an input file as the location for saving. This only works if the input is a file name and not a data object
3. Otherwise, the data is saved in the current working directory

File names

1. If `file_name` is specified, this name is used as a base to create the output names. A suffix is added to each output (you can see in the docstrings which suffix refers to which output). The extension of `file_name` specifies the format in which the output will be saved. If `file_name` has no extension, Nighres defaults to *nii.gz*
2. If `file_name` is not specified, Nighres tries to use the name of an input file as a base name for saving. This only works if the input is indeed a file name and not a data object

CHAPTER 17

Levelsets

Todo: What is a levelset and why is Nighres using it, example of creating levelset using *nighres.surface.probability_to_levelset()*

CHAPTER 18

Overview

You want to contribute to Nighres? That's fantastic! A few words to get started:

You don't have to be an expert!

Contributing to open source software is a great way of learning by doing, and others in the Nighres community will be happy to help you out if you have questions or get stuck.

What to do

Every contribution helps – whether it's fixing a typo in the documentation or adding an entire new feature. Often it's a good idea to start by fixing a smaller issue to get familiar with the process. If you don't know where to start, you can look at the issue tracker.

Git/Github

To contribute you will need to know some basics about git and Github, which are not covered in this guide. There are lots of tutorials online (e.g. https://www.atlassian.com/git/tutorials, especially "Getting started" and "Collaborating")

Questions?

For specifics about code you want to contribute *Making a Pull Request* is a great way to start a discussion and get input. Otherwise, you can post a question on neurostars.org. Please use the tag **nighres** for your question.

Ready to go?

The next pages will take you through the steps you need from cloning the Nighres repository, to making a pull request.

Note: Help us improve this developer's guide! If you find anything unclear or not covered, open an issue on github or add what you were missing (see *Adapting the docs*)

CHAPTER 19

Setting up

1. Fork and clone the Nighres github repository

You can find a good description here. Make sure that you set up your local clone to track both your fork (typically as *origin*) as well as the original Nighres repo (typically as *upstream*).

2. Make a new branch to work on

```
git checkout -b <branch_name>
```

This is important. When you make a pull request later, we will likely ask you to rebase because the Nighres master might have moved on since you forked it. You need a clean master branch for that.

Pick a descriptive branch name. For example, when you fix a bug in the function load_volume a good name is `fix/load_volume`. When you write a new interface called laminar_connectivity (that would be cool!) a good name is `enh/laminar_connectivity`.

Make one branch for each new feature or fix. That way independent changes can be handled in different *pull requests* later.

3. Install in editable mode (optional)

```
pip install -e <path_to_nighres_directory>
```

This way, when you import Nighres functions in Python, they will always come from the current version of the code in your Nighres directory. This is convenient for testing your code while developing.

(Alternatively, you can stay inside your Nighres directory and import functions directly from there without installing at all)

4. Let the coding begin!

If you want to work on an existing function, you can most likely just make your changes, check if the *Examples* still run and the move on to *Adapting the docs* and *Making a Pull Request*.

If you want to add new functions be sure to check out our intructions on *Wrapping an existing CBS Tools class* or *Adding a new Python function*

Important: Please adhere to PEP8 style conventions. This is easiest by using a Python linter which is available in most editors.

CHAPTER 20

Wrapping an existing CBS Tools class

You want to wrap an existing CBS Tools module into Nighres? Great!

20.1 How does it work?

The CBS Tools modules have been first created for JIST. Most of them are however not directly dependent on the JIST and MIPAV libraries, which is why we can turn them into a JCC-compatible, lightweight library.

JCC requires that all data is passed as numbers, strings, and 1D arrays, so this is what the CBS Tools modules need to manipulate as input and output, and also why we created a second python layer for structured data handling. Once a CBS Tools module has been formatted to the adequate i/o structure, JCC wraps it and you can call it directly from python.

20.2 Standard procedures

Each JCC-wrapped CBS Tools module has three important functions: .setSomeParameter(), .getSomeResult(), and .execute(). You need to set the inputs with meaningful values, execute the module, and retrieve the results. Note that default parameters are always assumed when possible, but a good wrapper should expose all parameters.

20.3 General wrapper content

1 Start the Java Virtual Machine (JVM)

```
cbstools.initVM(initialheap='6000m', maxheap='6000m')
```

Note that the initial and maximum amount of allocated memory may be adjusted or set up as a global parameter.

2 Create an instance of the module

```
mymodule = cbstools.SomeCoolModule()
```

3 Set all the parameters and data arrays

```
my_module.setThisImportantParameter(some_value)            my_module.
setInputImage(cbstools.JArray('float')((my_image_data.flatten('F')).
astype(float))
```

4 Run the module

```
my_module.execute()
```

5 Retrieve the outputs

```
my_result_data = np.reshape(np.array(my_module.getCoolResultImage(),
dtype=np.float32), dimensions, 'F')
```

Note that because you are passing simple 1D arrays, you need to keep a record of image dimensions, resolutions, headers, etc.

20.4 Convenience python layer

We strongly advise (at least for inclusion into Nighres) to handle the inputs and outputs of python structures as we do, i.e. loading images from files or passing them as Nifti1Image, and allowing to save all outputs automatically.

We also adhere to a somewhat strict formatting of the outputs with two suffixes, first for the module name, and second for the specific output, e.g. ‘’_mgdm_seg” for the segmentation obtained from the mgdm_brain_segmentation module.

20.5 What if my favorite CBS Tools module is not ready to wrap?

Because preparing modules for wrapping requires some reformatting, only a few modules so far are ready. You can accelerate the process in two ways: 1) send a request to the CBS Tools developers or 2) write a reformatted module for CBS Tools yourself and make a pull request. All the currently wrapped CBS Tools modules follow the same formatting procedure, so transforming a JIST module definition into a core module (which then can be called either from JIST or from Nighres) should be reasonably easy in most cases.

CHAPTER 21

Adding a new Python function

Great, you want to contribute a Python function or module to Nighres!

21.1 Before you begin

Please take a look at the code that is under development or discussion. If you see something related get in touch with the developer(s) directly to combine your efforts.

Take a moment to familiarise yourself with the documentation and functions, it is likely that some helper functions that you will want to use (e.g., I/O) have already been coded and tested.

21.2 Let's get into it

1. Follow the steps described in *Setting up*
2. Decide where to put your code

   If the functionality that you are adding falls within the scope of a pre-existing submodule, edit or create files within the submodule. If it is entirely new (e.g., *statistics*), create a new submodule with its own dedicated subdirectory.
3. Coding If you are creating a submodule from scratch, an easy way to start is to copy the *__init.py__* and initial import statements from an existing module.

   Please code PEP8 compliant, best to use a Python linter in your editor.

   Please keep within our *documentation guidelines*.

   Leave plenty of comments in your code so that others can understand and possibly adapt your code later.

   Use the standard I/O interfaces wherever possible but also fee free to add additional I/O functionality as necessary.

   Test you code internally. We aim to add unittests in the future, feel free to make a start on that.

4. *Write an example* showcasing your new function
5. To get your code into the Nighres master follow *Adapting the docs* and *Making a Pull Request*
6. Pat yourself on the back, you have now joined the Nighres developer community!

A word on dependencies

One of Python's strengths are the many different packages with specific functionalities. We try to keep the dependencies for Nighres as slim as possible, so please keep that in mind while you are coding. For example, if you need to perform a correlation and would normally use scipy.stats.pearsonr maybe you can get by with numpy.coeff (numpy is already a dependency, scipy isn't). If additional packages are required, they should be pip installable and potentially relevant to other functionality that may be coded in the future.

CHAPTER 22

Writing examples

CHAPTER 23

Adapting the docs

Nighres uses Sphinx which makes it easy to automatically pull the docstrings of functions into the online documentation. This means once a function has a good docstring, the work is essentially done. It also means that writing great docstrings is important.

23.1 If you changed an existing function

Make sure to check if your changes also require changes in the docstring. Adapting an existing docstring is usually straightforward. If you are curious how your changes will look in the online docs, you can try out *Building the docs locally*.

23.2 If you added a new function

Make sure to write a comprehensive docstring. We use NumPy/SciPy docstring conventions. But the easiest is to just copy another function's docstring and adapt. Then you need to add a few things to the `nighres/doc` directory:

1. Go to the subdirectory which refers to the submodule you added a function to, e.g. `nighres/doc/laminar/`.
2. Create a new file named after your new function, e.g. *laminar_connectivity.rst*
3. Add the name of this file in the *index.rst* file of the submodule
4. Copy the content of another function's rst file inside of your new file
5. Adapt the title and the two mentions of the function name in your new file to match your new function
6. Submit the changes to the docs along with your PR

Again, you can check how the changes you made will look in the online documentation by *Building the docs locally*.

23.3 If you added a new submodule

This is going to be a rare case. But if indeed added a new submodule in your PR, say its called *nighres.fancypants*:

1. In the doc directory, add a new subdirectory for your new module, e.g. `nighres/doc/fancypants`
2. In that subdirectory make a file called *index.rst*. Best to just copy the index.rst from another submodule. Then adapt the title in the file to the name of your new submodule, and remove all function names in the toctree.
3. Follow steps 3 to 5 from *If you added a new function* to add all functions of your new submodule
4. Add your submodule in the *index.rst* file in the `nighres/doc` main directory under the "Modules and Functions" toctree. In our case we would add the line *fancypants/index*
5. Submit the changes to the docs along with your PR

23.4 More than docstrings

You can also make changes to parts of the documentation that are not function docstrings. For example to help improving this guide!

The easiest way is to browse to the page in the online documentation that you want to change and click on **Edit on Github** to find the location of the file you have to change. For example, if you want to make a change to this page, the link would send you to https://github.com/nighres/nighres/blob/master/doc/developers/docs.rst, which tells you that in your local copy of the repo you should edit the file *nighres/doc/developers/docs.rst*.

Sphinx uses reStructuredText (reST) syntax. For formatting take a look at this Sphinx Cheatsheet.

You can also add whole new parts to the online documentation. This might require learning a bit more about Sphinx. You can also open an issue and get help.

When you are done, you can check your changes by *Building the docs locally*.

23.5 Building the docs locally

If you make changes to the documentation, they will only appear online at http://nighres.readthedocs.io/en/latest/ once your PR has been merged and a new release was made. You can, however, build the docs locally for a preview:

1. Make sure to have all *additional dependencies* installed that you need to build the documentation
2. Navigate into your local copy of `nighres/doc`
3. Type `make clean` and then `make html`
4. Open an Internet browser and point it to the local html pages that were just built for you, e.g. file:///home/nighres/doc/_build/html/index.html

You can make changes and repeat this process until you are satisfied.

CHAPTER 24

Making a Pull Request

If you are working on changes to the Nighres code, let others know by opening a Pull Request (PR) .

You don't have to wait until your code is perfect, opening a PR early is a great way of letting other members of the community know that you are working on this. That avoids redundant work and gets you input at an early stage.

To open a PR:

1. Go to your fork of Nighres in your Github account
2. In the **Branch** drop-down menu, switch to the branch that you have worked on
3. Click on **New pull request**. You will see the changes that will be included in your PR.
4. Enter a descriptive title and small comment why these changes are necessary. If you are addressing a specific issue, refer to the issue number.
5. Click **Create pull request** and wait for responses (by default you will be informed about activity on your PR per e-mail)

After your PR is reviewed, you might be asked to make some changes. This is normal, almost no PR is merged without revisions. Pushing changes to your own branch automatically updates the code in the PR.

When all comments are addressed we will likely ask you to rebase, and then merge your code into the Nighres repository.

Reference

Huntenburg, Steele & Bazin (2018). Nighres: processing tools for high-resolution neuroimaging. GigaScience, 7(7). https://doi.org/10.1093/gigascience/giy082

Make sure to also cite the references indicated for the particular functions you use!

Credit

Nighres is a community-developed project made possible by its contributors. The project was born and continues to evolve at brainhack. We thank the Google Summer of Code 2017 and INCF as a mentoring organization, for supporting the initial development phase of Nighres. See also the development blog.

Index

T

V