<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Installation</td>
<td>3</td>
</tr>
<tr>
<td>1.1 Using conda</td>
<td>3</td>
</tr>
<tr>
<td>1.2 Using pip</td>
<td>3</td>
</tr>
<tr>
<td>2 Tutorial</td>
<td>5</td>
</tr>
<tr>
<td>2.1 Calculation</td>
<td>5</td>
</tr>
<tr>
<td>2.2 Analyze</td>
<td>6</td>
</tr>
<tr>
<td>2.3 Plotting</td>
<td>6</td>
</tr>
<tr>
<td>2.4 Writing</td>
<td>7</td>
</tr>
<tr>
<td>3 Basic examples</td>
<td>9</td>
</tr>
<tr>
<td>4 DocStrings</td>
<td>11</td>
</tr>
<tr>
<td>4.1 Tra</td>
<td>11</td>
</tr>
<tr>
<td>4.2 LagrantoRun</td>
<td>14</td>
</tr>
<tr>
<td>5 Indices and tables</td>
<td>17</td>
</tr>
</tbody>
</table>
Contents:
Installation

1.1 Using conda

conda install -c conda-forge lagranto

1.2 Using pip

Ideally install it in a virtual environment. See for example virtualenvwrapper.

pip install Lagranto

Then you can install the dependencies for testing or the documentations as follow:

pip install Lagranto[testing]

pip install Lagranto[docs]

To build the documentation do the following:

cd /path/to/lagranto/location
cd docs
make html

To test the package do the following:

cd /path/to/lagranto/location
pytest
The goal of this section is to show how to calculate trajectories, analyze them, and plot them using the Basic examples.

- Calculation
- Analyze
- Plotting
- Writing

## 2.1 Calculation

The Basic examples provide a class, `lagranto.LagrantoRun`, to wrap the Lagranto programs in python. It allows to calculate trajectories in parallel. You can take a look at the docstring to get familiar with the class.

Let’s say that we want to calculate Warm Conveyor Belt trajectories for a 5 day period in June 2013. Using Era-Interim we can start trajectories every 6 hour and we will calculate them for 48 hours forward in time. Since `lagranto.LagrantoRun` needs a list of (startdate, enddate), we can build the dates as follows:

```python
from datetime import datetime, timedelta
from dypy.small_tools import interval
startdate = datetime(2013, 6, 1, 0)
dateend = startdate + timedelta(days=5)
dates = [(d, d + timedelta(hours=48)) for d in interval(startdate, enddate, timedelta(hours=6))]
```

If the Era-interim data are in the `erainterim` folder and if the output files should be written in the `output` folder, then the `lagranto.LagrantoRun` can be initialized as follows:

```python
from lagranto import LagrantoRun
lrun = LagrantoRun(dates, workingdir='erainterim', outputdir='output', version='ecmwf')
```

We want to start the trajectories every 20km in the box [5E, 40E, 30N, 60N], so let’s create a starting file:

```python
specifier = "'box.eqd(5,20,40,50,20)@profile(850,500,10)@hPa'"
out_create_startf = lrun.create_startf(startdate, specifier, tolist=True)
```
The `tolist` argument is needed if we want to use the same starting file for all starting time of the trajectories. We can now calculate the trajectories, but first starting for a single date to test our setup:

```python
out_caltra = lrun.caltra(*dates[1])
```

We can also test tracing Q along a trajectories:

```python
out_trace = lrun.trace(dates[1][0], field='Q 1.1')
```

We can now calculate and trace the trajectories in parallel, but for this we will use a tracevars file:

```python
tracevars = """Q 1. 0 P
U 1. 0 P
"""""""out = lrun.run_parallel(trace_kw={'tracevars_content': tracevars}, type='both')
```

The `tracevars_content` keyword argument will be passed to trace to create a tracevars file with Q and U. The `type` keyword argument determine what is run in parallel, currently both, trace, and caltra are available.

### 2.2 Analyze

Now that we have calculated trajectories let’s read them and analyze them. By default the name of the files are formatted as `lsl_{%Y%m%d%H}.4`. So if we want to read the trajectories started at 00 UTC 01 June 2013 we can do as follow:

```python
from lagranto import Tra
filename_template = 'output/lsl_{:%Y%m%d%H}.4'
filename = filename_template.format(date=dates[-1][0])
trajs = Tra()
trajs.load_netcdf(filename)
print(trajs)
```

We can now test if the trajectories fulfill the standard criteria for WCB, an ascent greater than 500 hPa in 48 hours. To make it clear, the goal of this example is not to replace the fortran routines of the LAGRANTO package but to illustrate the possibilities that python provides to analyze trajectories using a simple example.

```python
wcb_index = np.where((trajs['p'][:, :1] - trajs['p']) > 500)
wcb_trajs = Tra()
wcb_trajs.set_array(trajs[wcb_index[0], :])
print(wcb_trajs)
```

### 2.3 Plotting

Now that we have WCB trajectories, let’s plot them on a map. We will use cartopy for this.

```python
import cartopy.crs as ccrs
import cartopy.feature as cfeature
from lagranto.plotting import plot_trajs
import matplotlib.pyplot as plt

fig = plt.figure()
ax = plt.axes(projection=crs)
land_50m = cfeature.NaturalEarthFeature('cultural', 'admin_0_countries', '50m', edgecolor='gray', facecolor='none', linewidth=0)
```

```python
```
2.4 Writing

The WCB trajectories can also be written to disk as follow:

```python
wcb_trajs.write_netcdf('output/wcb_trajs.nc')
```
In a first step, let’s simply read the trajectories:

```python
from lagranto import Tra
filename = 'lsl_20110123_10'
trajs = Tra()
trajs.load_ascii(filename)
```

or to read a netcdf file:

```python
filename = 'lsl_20110123_10.4'
trajs.load_netcdf(filename)
```

The proprieties of the trajectories can be shown as follow::

```python
print(trajs)  
24 trajectories with 41 time steps. 
Available fields: time/lon/lat/p/Q/RH/TH/BLH 
total duration: -14400.0 minutes
print(trajs.variables())
['time', 'lon', 'lat', 'p', 'Q', 'RH', 'TH', 'BLH']
print(trajs['Q'].shape)
(24, 41)
```
4.1 Tra

class lagranto.Tra(filename="", usedatetime=True, array=None, **kwargs)
Class to work with LAGRANTO output.

Read trajectories from a LAGRANTO file and return a structured numpy array.

Parameters

- **filename** (string) – File containing lagranto trajectories
- **usedatetime** (bool) – Read times as datetime objects, default True
- **array** (structured array) – If defined creates a new Tra object filled with the array

Returns structured array (Tra)

Return type trajs(ntra,ntime) with variables as tuple.

Examples

```python
>>> filename = 'mylabelfile.nc'
>>> trajs = Tra()
>>> trajs.load_netcdf(filename)
>>> trajs['lon'][0, :]
# return the longitudes for the first trajectory.

>>> trajs = Tra(filename)
>>> selected_trajs = Tra(array=trajs[[10, 20, 30], :])
```

Author [Nicolas Piaget, ETH Zurich, 2014] Sebastiaan Crezee, ETH Zurich, 2014

append (trajs)
append trajectories

Parameters **trajs** (single Tra or list of Tra) – Trajectories to concatenate with the current one
Examples

```python
>>> files = ['lsl_20001010_00.4', 'lsl_2001010_01.4']
>>> filename = files[0]
>>> trajs = Tra(filename)
>>> newtrajs = [Tra(f) for f in files]
>>> trajs.append(newtrajs)
```

`concatenate(trajs, time=False, inplace=False)`

To concatenate trajectories together.

Concatenate trajectories together and return a new object. The trajectories should contain the same variables. If `time=False`, the number of timestep in each trajs should be the same if `time=True`, the number of trajectories in each `Tra` should be the same.

**Parameters**

- `trajs` ([`Tra` or list of `Tra`]) – Trajectories to concatenate with the current one.
- `time` (bool, default False) – if True concatenate along the time dimension.
- `inplace` (bool, default False) – if True append the trajs to current `Tra` object and return None.

**Returns**

Return a new `Tra` (trajectories) object.

**Return type** `Tra`

Examples

Create a new `Tra` object which include `trajs` and all `newtrajs`

```python
>>> files = ['lsl_20001010_00.4', 'lsl_2001010_01.4']
>>> filename = files[0]
>>> trajs = Tra(filename)
>>> newtrajs = [Tra(f) for f in files]
>>> alltrajs = trajs.concatenate(newtrajs)
```

Append `newtrajs` to `trajs`

```python
>>> trajs = Tra(filename)
>>> newtrajs = [Tra(f) for f in files]
>>> trajs.concatenate(newtrajs, inplace=True)
```

duration

Time duration in minutes.

`get_array()`

Return the trajectories array as numpy object.

`initial`

Give the initial time of the trajectories.

`load_ascii(filename, usedatetime=True, msv=-999.999, gz=False)`

Load trajectories from an ascii file.

**Parameters:**

- `usedatetime`: bool, default True – If true return the dates as datetime object.
- `msv`: float, default -999.999 – Change `<msv>` value into np.nan.
- `gz`: bool, default False – If true read from gzip file.
load_netcdf(filename, usedatetime=True, msv=-999, unit='hours', **kwargs)
Load trajectories from a netcdf

Parameters

- **filename** (string) – path to a netcdf file containing trajectories
- **usedatetime** (bool, default True) – If True then return time as datetime object
- **msv** (float, default -999) – Define the missing value
- **unit** (string, default hours) – Define the units of the times (hours, seconds or hhmm)
- **exclude** (list of string, default empty) – Define a list of variables to exclude from reading
- **date** (datetime or list) – Can be used to select particular dates, for example to read in a single timestep
- **indices** (list or tuple) – Can be used to select particular trajectories

ntime
Return the number of time steps

ntra
Return the number of trajectories

set_array(array)
To change the trajectories array.

startdate
Return the starting date of the trajectories

variables
Return the names of the variables

write(filename, fileformat='netcdf')
Method to write the trajectories to a file

write_ascii(filename, gz=False, digit=3, mode='w')
Write the trajectories in an ASCII format

Parameters:

- **filename** [string] filename where the trajectories are written
- **mode** [string, default w] define the mode for opening the file. By default in write mode (‘w’), append (‘a’) is another option

Parameters

- **filename** (string) – filename where the trajectories are written
- **mode** (string, default w) – define the mode for opening the file. By default in write mode (‘w’), append (‘a’) is another option.
- **gz** (boolean, default False) – If true write the file as a gzip file
- **digit** (int, default 3) – Number of digit after the comma to use for lon, lat; Only 3 or 2 digits allowed

write_netcdf(filename, exclude=None, unit='hours')
Write the trajectories in a netCDF file

Parameters

- **trajs** (Tra) – A Tra instance
- **filename** (string) – The name of the output file
• **exclude** *(list, optional)* – A list of variables to exclude

• **unit** *(string, optional)* – The unit of the dates, can be hours, seconds or hhmm

## 4.2 LagrantoRun

**class lagra.**

```python
class lagra.LagrantoRun(dates, workingdir='.', outputdir='.', startf='startf.4',
                        lslname='lsl_{:%Y%m%d%H}.4', tracevars='', field='',
                        version='cosmo', linkfiles=None, nprocesses=10, sdate=None,
                        fresh=False, cmd_header=None, no_tmp_dir=False)
```

`Perform Lagranto calculation.`

**Parameters**

• **dates** *(list)* – list of (startdate, enddate) tuple

• **workingdir** *(string, optional)* – path to the model output directory, default to current

• **outputdir** *(string, optional)* – path to the trajectory output directory, default to current

• **startf** *(string, optional)* – name of the startf to use (or to create), default to startf.4 The format parameter date can be used, for ex: startf_{date:%Y%m%d%H}.4

• **lslname** *(string, optional)* – name of the lsl file, define its type, default to lsl_{:%Y%m%d%H}.4

• **tracevars** *(string, optional)* – name of a tracevars file as used by trace, default to none

• **field** *(string, optional)* – name of a single field to trace, default to none

• **version** *(string, optional)* – name of the model version to use, currently only cosmo (default)

• **linkfiles** *(function, optional)* – function used to overwrite link_files in run. Should be used if COSMO output is not standard netcdf

• **nprocesses** *(int, optional)* – Number of processes used when running in parallel, default to 10

• **sdate** *(datetime object,)* – Starting date of the simulation; useful if files are named in forecast mode

• **fresh** *(bool, optional)* – Fresh start. Remove output directory first.

• **cmd_header** *(string, optional)* – Change the header using for running the command; see run_cmd help for more details.

• **no_tmp_dir** *(bool, optional)* – If set to True, do not create temporary folder. Useful only for special version of lagra, and if the system has no tmp filesystem.

**caltra** *(startdate, enddate, filename='', outfile='', **kwargs)*

`Compute trajectories for the given startdate`

**clear()**

`Remove the output directory`

**create_startf** *(date, specifier, filename='', tolist=False, **kwargs)*

`Create a file with the starting position of the trajectories.`

**Parameters**

• **date** *(datetime)* – date for which the startf date is produced
**specifier (string)** – detailed description of starting positions; (see LAGRANTO documentations for more details)

**filename (string)** – filename where starting positions are saved; the default is defined by `LagrantoRun`. Can make use of the format parameter `date`, for example: `startf_{date:%Y%m%d%H}.4`

**tolist (bool)** – If the starting position should be saved as a list of points; Useful if the same file is used for different starting times

**kwargs (dict)** – key, value options, possible values are listed below;
- `cmd_header`: header to add to the shell command; (see `run_cmd` for more details)
- `options`: list of options to pass to the startf function; (see the LAGRANTO documentation for more details)

**density (inpfile=None, outfile=None, **kwargs)**
Return the density of trajectories

**run (caltra_kw=None, trace_kw=None, startf_kw=None, **kwargs)**
Run caltra, trace, and/or `create_startf`. Run `single_run` for each dates tuple. See `single_run` description for more details.

**Parameters**
- `caltra_kw (dictionary)` – Arguments to pass to the `caltra` function
- `trace_kw (dictionary)` – Arguments to pass to the `trace` function
- `startf_kw (dictionary)` – Arguments to pass to the `create_startf` function
- `kwargs` – Arguments to pass to the `single_run` function

**Returns** A list of `single_run` result

**Return type** list

**run_parallel (caltra_kw=None, trace_kw=None, startf_kw=None, nprocesses=None, **kwargs)**
Run caltra, trace, and/or `create_startf` in parallel.
Run `single_run` for each dates tuple using `multiprocessing.Pool`. See `single_run` description for more details.

**Warning:** If you use `run_parallel` with `create_startf` and if the starting file is not a list, be sure that every starting file has a different name, for example doing as follow:

```python
>>> LagrantoRun(startf='startf_{date:%Y%m%d_%H.4}', ...)```

**Parameters**
- `caltra_kw (dictionary)` – Arguments to pass to the `caltra` function
- `trace_kw (dictionary)` – Arguments to pass to the `trace` function
- `startf_kw (dictionary)` – Arguments to pass to the `create_startf` function
- `nprocesses (int,)` – Number of processors to use
- `kwargs` – Arguments to pass to the `single_run` function

**Returns** A list of `single_run` results

**Return type** list

**select (inpfile=None, outfile=None, **kwargs)**
Perform selection of trajectories
**single_run**(*sd, ed, caltra_kw=None, trace_kw=None, startf_kw=None, type='both', debug=False, **kwargs*)

Run caltra, trace and/or create_startf for a given starting date

Compute a trajectory set using caltra, trace and/or create_startf in a temporary directory. Need a working installation of LAGRANTO.

If caltra, trace or create_startf return a error, single_run return a LagrantoException error with the path of the temporary directory. In this case, the temporary directory is not deleted.

**Parameters**

- **sd** (*datetime*) – Starting date of the trajectory
- **ed** (*datetime*) – Ending date of the trajectory
- **caltra_kw** (*dictionary*) – Arguments to pass to the caltra function
- **trace_kw** (*dictionary*) – Arguments to pass to the trace function
- **startf_kw** (*dictionary*) – Arguments to pass to the create_startf function
- **type** (*string (default both)*) – Define the type a run to perform: both: caltra + trace caltra: caltra trace: trace all: create_startf + caltra + trace create: create_startf + caltra
- **debug** (*Boolean*) – If True raise LagrantoException instead of return it as string
- **kwargs** (*dictionary*) – Arguments to pass to the link_files function

**Returns** output of caltra, trace and/or create_startf

**Return type**  *str*

**trace**(*date, filename="", outfile="", tracevars="", tracevars_content="", field="", **kwargs*)

Trace variable along a trajectory.

Trace meteorological fields along trajectories

**Parameters**

- **date** (*datetime object*) – starting date of the trajectories to trace
- **filename** (*string, optional*) – If not specified use `LagrantoRun.lslname`
- **outfile** (*string, optional*) – If not specified same as filename
- **tracevars** (*string, optional*) – Name of the file with the field to trace; If not specified use `LagrantoRun.tracevars`
- **tracevars_content** (*string, optional*) – Content of the tracevars file; """n QV 1000. P 1n PV 1. T 1n ""
- **field** (*string,*) – Specify a field to trace as follow: QV 1.
- **kwargs** –
CHAPTER 5

Indices and tables

• genindex
• modindex
• search
Index

A
append() (lagranto.Tra method), 11

C
caltra() (lagranto.LagrantoRun method), 14
clear() (lagranto.LagrantoRun method), 14
concatenate() (lagranto.Tra method), 12
create_startf() (lagranto.LagrantoRun method), 14

density() (lagranto.LagrantoRun method), 15
duration (lagranto.Tra attribute), 12

G
get_array() (lagranto.Tra method), 12

I
initial (lagranto.Tra attribute), 12

L
LagrantoRun (class in lagranto), 14
load_ascii() (lagranto.Tra method), 12
load_netcdf() (lagranto.Tra method), 12

N
ntime (lagranto.Tra attribute), 13
ntra (lagranto.Tra attribute), 13

R
run() (lagranto.LagrantoRun method), 15
run_parallel() (lagranto.LagrantoRun method), 15

S
select() (lagranto.LagrantoRun method), 15
set_array() (lagranto.Tra method), 13
single_run() (lagranto.LagrantoRun method), 15
startdate (lagranto.Tra attribute), 13

T
Tra (class in lagranto), 11
trace() (lagranto.LagrantoRun method), 16

V
variables (lagranto.Tra attribute), 13

W
write() (lagranto.Tra method), 13
write_ascii() (lagranto.Tra method), 13
write_netcdf() (lagranto.Tra method), 13