PHETS Documentation

Release 1.0

Dec 07, 2017

CONTENTS

This package offers high-level tools for exploration and visualization of delay coordinate embedding and persistent homology. It is used to investigate the utilization of these tools together as a signal processing technique.

PHETS encompasses four submodules:

- *signals*
- phomology
- embed
- prfstats

signals holds the TimeSeries and Trajectory classes, which can be initialized from arrays or text files. Calling the embed method of a TimeSeries returns a Trajectory; calling the project method of Trajectory returns a TimeSeries. TimeSeries and Trajectory both inherit from BaseTrajectory, where all cropping, windowing, and normalization is handled.

phomology holds the Filtration class, which is initialized from a Trajectory and a dict of filtration parameters. Filtration movies, persistence diagrams, and persistence rank functions are created by calling the respective methods of the Filtration class.

embed holds the embed function, as well as functions for generating movies. The movies functions take one or more TimeSeries and return one or more Trajectory objects (created in the process of building the movies).

prfstats holds functions for statistical analysis of PRFs. Generally, they take one or two Trajectory objects, create PRFs from the windows of the the Trajectory objects, do some analysis, and then save plots from the results.

CHAPTER

ONE

TROUBLESHOOTING

1.1 matplotlib backend error

Comment out the first line of `PHETS/matplotlibrc`

1.2 compiling `find_landmarks.c` on OSX

PHETS requires the OpenMP C library `omp.h`. From what I can tell, OpenMP is not included in clang (the default C compiler on macOS), and may only be installed /configured for recent versions, and not with great ease. For these reasons, we've never tried to run PHETS on clang, and cannot guarantee it will work correctly.

On the other hand, OpenMP works with gcc out of the box, and you may already have a version of gcc installed. If so, determine the version and edit `find_landmarks_c_compile_str` in `config.py` to match. (NOTE: on macOS, `gcc` is a symlink for clang. This is avoided by including the version number, eg `gcc-5`.)

If you do not have gcc installed, you can do `brew install gcc` and then, as above, tweak `config.py`. You can also tell brew to install a particular version if you would like (anything 5+ should work).

A quick way to test if things are working is to run `python refresh.py` in the PHETS directory. This script will remove a number of temporary files and attempt to compile `find_landmarks.c`

If the compiler is still giving errors (don't mind warnings), try `brew upgrade gcc` or `brew reinstall gcc --without-multilib`

See here and here for more information.

CHAPTER

TWO

REGRESSION TESTS

pytest is used for testing. To run the test suite, type `pytest --tb=short` from the top-level directory. (Running pytest within a subdirectory will only execute the tests for that submodule.)

Each submodule contains a `unit_test` directory. The tests themselves are defined in `unit_tests/test__<submodule>.py`. These are not exactly unit tests – rather, each one calls or initializes a user-facing feature and compares the result to a saved reference. The input data is found in `unit_tests/data` and the references in `<unit_tests/ref>`.

In the case of the `prfstats``module, in order to keep test execution time lwo, the input data is pre-computed sets of Filtration objects to keep test execution time low. A small, correct change to PHETS can break Python's ability to load these objects from file, breaking the tests. I this case, run the routines in ``prfstats/unit_tests/prepare__data.py`, and the tests should work correctly. Routines in `unit_tests/prepare__refs.py` should be run *only when you wish to change the behavior of existing functionality*. They should also be run and individually (that is, don't change the refs for features that you aren't intentionally modifying).

CHAPTER

THREE

THIS DOCUMENTATION

This documentation is built with Sphinx. The autodoc extension is used to generate the library reference from doc-strings in the Python code. The text and layout for all other sections (eg this paragraph) is defined in `docs/source/index.rst`.

To build this documentation, TeX must be installed, along with the following:

- texlive-latex-recommended
- texlive-fonts-recommended
- texlive-latex-extra
- latexmk

I used `sudo apt-get install <package>` for each.

Now, simply .. code-block:

```
cd docs
make latexpdf
```

The updated documentation is saved to `docs/latex/PHETS.pdf`.

CHAPTER

FOUR

REFERENCE

4.1 signals

class signals.**BaseTrajectory**(*data*, *crop=(None*, *None)*, *num_windows=None*, *window_length=None*, *vol_norm=(False*, *False*, *False)*, *time_units='samples'*, *name=None*, *fname=None*)

Parameters

- **data** (*str or array*) – The filename to load, or array. If a filename, sets `fname`.
- **crop** (*array, optional*) – Range of signal to work with. Observes `time_units`. Either or both bounds may be None. format: (start, stop). default: (None, None)
- **num_windows** (*int, optional*) – Slice signal into `windows` evenly spaced windows. default: None
- **window_length** (*int or float, optional*) – Observes `time_units` if None, `window_length == len(data) / num_windows` default: None
- **vol_norm** (*arr, optional*) – Normalize amplitude by (full, crop, window). default: (False, False, False)
- **time_units** (*str, optional*) – `'samples'` or `'seconds'` Observes `config.SAMPLE_RATE` default: `'samples'`
- **name** (*string, optional*) – Sets `name`, a label used for titles for plots. If None and `fname` is not None, `name` is derived from `fname`. default: None
- **fname** (*string, optional*) – If `data` is not a filename (i.e. is an array), sets `fname`. default: None

crop(*crop_cmd*)

Set `data` to the region of `data_full` specified by `crop_cmd` and `time_units`.

Parameters **crop_cmd** (*array*) – observes `time_units` format: (start, stop)

slice(*num_windows*, *window_length=None*)

Sets `windows`, an array of evenly spaced windows from `data`.

Parameters

- **num_windows** (*int*) –
- **window_length** (*int or float, optional*) – observes 'time_units'' if None, `window_length == len(data) / num_windows` default: None

class signals.**TimeSeries**(*data*, ***kwargs*)

Bases: `signals.signals.BaseTrajectory`

See *BaseTrajectory* for parameter descriptions

embed(*tau*, *m*)
Embed `data_full`, re-apply crop and slicing.

Parameters

- **tau** (*int or float*) – observes `time_units`
- **m** (*int*) –

Returns

Return type *Trajectory*

plot(*filename*)
Plot full time series with crop and windows demarcated, save to `filename`.

Parameters **filename** (*str*) –

plot_crop(*filename*)
Plot time series (crop only), save to `filename`. :param filename: :type filename: str

class `signals.`**Trajectory**(*data*, ***kwargs*)
Bases: `signals.signals.BaseTrajectory`

See *BaseTrajectory* for parameter descriptions

filtrations(*filt_params*, *quiet=True*, *status_str=None*)
Compute filtration for each window of trajectory.

Parameters

- **filt_params** (*dict*) – see Filtration
- **quiet** (*bool*) – terminal output noise

Returns array of Filtration objects

Return type array

project(*axis=0*)
Project `self.data_full` to time series, re-apply crop and slicing.

Parameters **axis** (*int*) –

Returns

Return type *TimeSeries*

4.2 PH

4.3 DCE

4.4 PRFstats

PYTHON MODULE INDEX

S

INDEX