pyeemd Documentation

Release 1.1

Perttu Luukko

September 19, 2016

Contents

1 Contents: **1**
- 1.1 Installing pyeemd . . . 1
- 1.2 Tutorial . . . 1
- 1.3 API documentation . . . 3

2 Indices and tables **7**

Bibliography **9**

Python Module Index **11**

Python Module Index **13**

CHAPTER 1

Contents:

1.1 Installing pyeemd

The *pyeemd* module comes with a regular Python distutils installation script, so installing it should be quite straightforward. The only catch is that you need to first compile `libeemd.so`, since *pyeemd* is only a wrapper for that library. Please see the `README` file distributed with *libeemd* on more details on how to compile *libeemd*, but if you are unpatient and already have the necessary dependencies installed (GCC, GSL), you can just run `make` in the top-level directory of *libeemd* and you are done.

To install *pyeemd* please run:

```
python2 setup.py install
```

In the top-level directory of *pyeemd* (the one with `setup.py`).

If you want to specify an alternative installation prefix, you can do it as follows:

```
python2 setup.py install --prefix=$HOME/usr
```

1.2 Tutorial

After installing *pyeemd* as described in *Installing pyeemd* you are all set to using it with:

```
import pyeemd
```

The three main decomposition routines implemented in *pyeemd* – EMD, EEMD and CEEMDAN – are available as *emd()*, *eemd()* and *ceemdan()*, respectively. All these methods use similar conventions so interchanging one for another is easy.

Input data to these routines can be any kind of Python sequence that `numpy` can convert to an 1D array of floating point values. The output data will be a 2D `numpy` array, where each row of the array represents a single *intrinsic mode function* (IMF).

As an example, the *examples* subfolder of *pyeemd* contains a file `ecg.csv`, which contains ECG (electrocardiogram) data from the MIT-BIH Normal Sinus Rhythm Database. The data is in CSV (comma separated value) format, which can be read into Python in many ways, one of which is using `numpy.loadtxt()` using the appropriate delimiter:

```
from numpy import loadtxt

ecg = loadtxt("ecg.csv", delimiter=',')
```

Now we have the data in a `numpy` array *ecg*. We can quickly plot what the data looks like using `matplotlib.pyplot`.

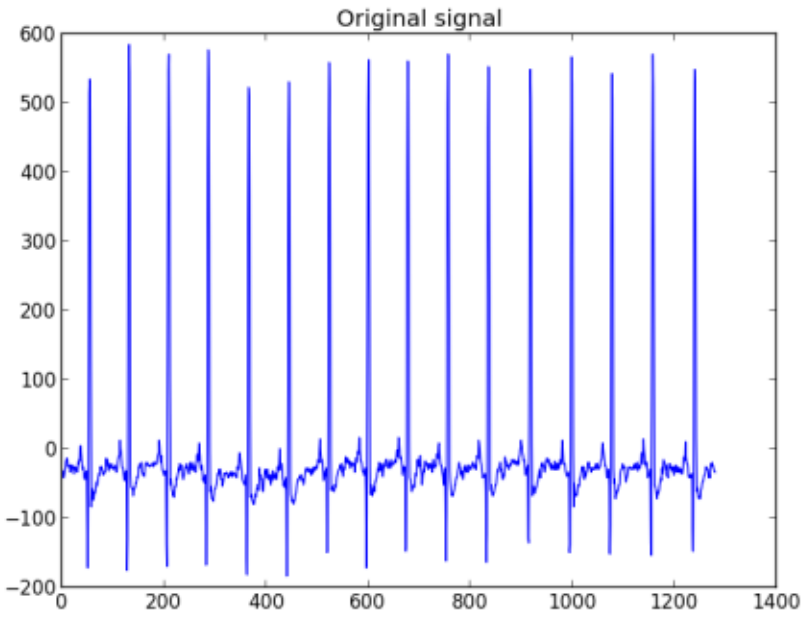

Fig. 1.1: Original ECG signal as plotted by `matplotlib.pyplot`.

```
from matplotlib.pyplot import plot, show, title

title("Original signal")
plot(ecg)
show()
```

The data stored in *ecg* can be decomposed with CEEMDAN using the routine *`ceemdan()`*. The only thing we need to decide is what to use as the stopping criterion for the sifting iterations. In this example we use a S-number of 4. The other choice would be to set a maximum number of siftings (as used in the reference EEMD code) by using the parameter *num_siftings*:

```
from pyeemd import ceemdan

imfs = ceemdan(ecg, S_number=4)
```

Now the rows of the 2D array *imfs* are the IMFs the original signal decomposes to when applying CEEMDAN. We can plot these IMFs using `matplotlib.pyplot` as before, but *pyeemd* also comes with an utility function *`plot_imfs()`* for easily plotting the IMFs (using `matplotlib.pyplot`) in separate figures.

```
from pyeemd.utils import plot_imfs

plot_imfs(imfs, plot_splines=False)
show()
```

The `plot_splines=False` argument prevents the plotting of the envelope curves of the IMFs, which would otherwise be shown.

This concludes our simple tutorial. For more in-depth information about the methods available in *pyeemd* please head to the *API documentation*. You can also look at example code at the `examples` subdirectory of *pyeemd*.

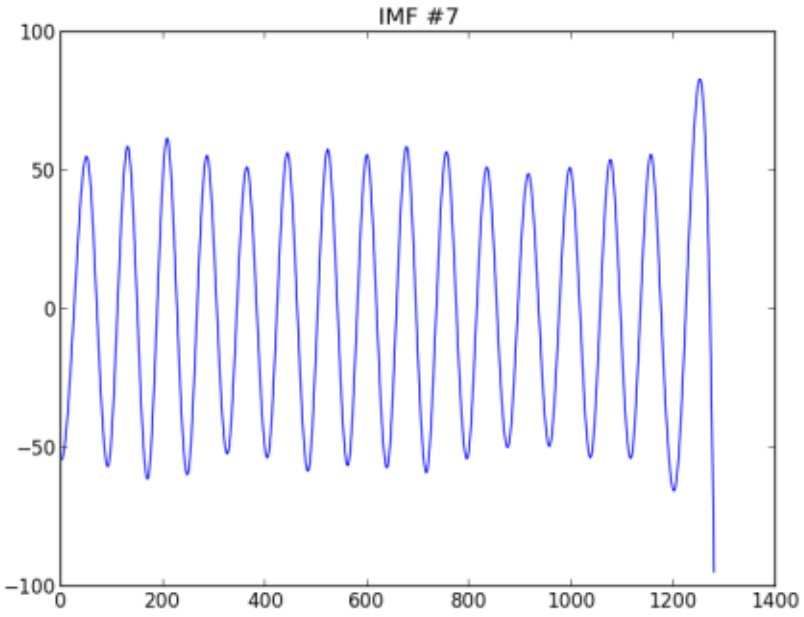

Fig. 1.2: IMF 7 extracted from ECG data with `ceemdan()` and plotted with `plot_imfs()`.

How you choose to use or process the IMFs obtained by the decomposition routines is beyond the scope of this document – and beyond the scope of *pyeemd* – but you might be interested in the Hilbert transform routine offered by `scipy.fftpack.hilbert()`.

1.3 API documentation

1.3.1 Main decomposition routines

pyeemd.**eemd**(*inp*, *ensemble_size=250*, *noise_strength=0.2*, *S_number=0*, *num_siftings=0*, *rng_seed=0*)

Decompose input data array *inp* to Intrinsic Mode Functions (IMFs) with the Ensemble Empirical Mode Decomposition algorithm *[R1]*.

The size of the ensemble and the relative magnitude of the added noise are given by parameters *ensemble_size* and *noise_strength*, respectively. The stopping criterion for the decomposition is given by either a S-number or an absolute number of siftings. In the case that both are positive numbers, the sifting ends when either of the conditions is fulfilled.

Parameters **inp** : array_like, shape (N,)

The input signal to decompose. Has to be a one-dimensional array-like object.

ensemble_size : int, optional

Number of copies of the input signal to use as the ensemble.

noise_strength : float, optional

Standard deviation of the Gaussian random numbers used as additional noise. **This value is relative** to the standard deviation of the input signal.

S_number : int, optional

Use the S-number stopping criterion *[R2]* for the EMD procedure with the given values of *S*. That is, iterate until the number of extrema and zero crossings in the signal differ at most by one, and stay the same for *S* consecutive iterations. Typical values are in the range *3 .. 8*. If *S_number* is zero, this stopping criterion is ignored.

num_siftings : int, optional

Use a maximum number of siftings as a stopping criterion. If *num_siftings* is zero, this stopping criterion is ignored.

rng_seed : int, optional

A seed for the random number generator. A value of zero denotes an implementation-defined default value.

Returns **imfs** : ndarray, shape (M, N)

A *MxN* array with *M = emd_num_imfs(N)*. The rows of the array are the IMFs of the input signal, with the last row being the final residual.

See also:

emd The regular Empirical Mode Decomposition routine.

emd_num_imfs The number of IMFs returned for a given input length.

Notes

At least one of *S_number* and *num_siftings* must be positive. If both are positive, the iteration stops when either of the criteria is fulfilled.

References

[R1], *[R2]*

pyeemd.**emd**(*inp*, *S_number=0*, *num_siftings=0*)

A convenience function for performing EMD (not EEMD). This simply calls function `eemd()` with `ensemble_size=1` and `noise_strength=0`.

pyeemd.**ceemdan**(*inp*, *ensemble_size=250*, *noise_strength=0.2*, *S_number=0*, *num_siftings=0*, *rng_seed=0*)

Decompose input data array *inp* to Intrinsic Mode Functions (IMFs) with the Complete Ensemble Empirical Mode Decomposition with Adaptive Noise (CEEMDAN) algorithm *[R3]*, a variant of EEMD. For description of the input parameters and output, please see documentation of `eemd()`.

See also:

eemd The regular Ensemble Empirical Mode Decomposition routine.

emd_num_imfs The number of IMFs returned for a given input length.

References

[R3]

1.3.2 Auxiliary routines

pyeemd.**emd_num_imfs**(*N*)
Return number of IMFs that will be extracted from input data of length *N*, including the final residual.

pyeemd.**emd_find_extrema**(*x*)
Find the local minima and maxima from input data *x*. This includes the artificial extrema added to the ends of the data as specified in the original EEMD article *[R4]*.

Parameters **x** : array_like, shape (N,)

The input data. Has to be a one-dimensional array_like object.

Returns **all_extrema_good** : bool

Specifies whether the extrema fulfill the requirements of an IMF, i.e. , the local minima are negative and the local maxima are positive.

maxx : ndarray

The x-coordinates of the local maxima.

maxy : ndarray

The y-coordinates of the local maxima.

minx : ndarray

The x-coordinates of the local minima.

miny : ndarray

The y-coordinates of the local minima.

References

[R4]

pyeemd.**emd_evaluate_spline**(*x*, *y*)
Evaluates a cubic spline with the given (x, y) points as knots.

Parameters **x** : array_like, shape (N,)

The x coordinates of the knots. The array must be sorted, start from 0 and end at an integer.

y : array_like, shape (N,)

The y coordinates of the knots.

Returns **spline_y** : ndarray

The cubic spline curve defined by the knots and the "not-a-knot" end conditions, evaluated at integer points from 0 to `max(x)`.

See also:

emd_find_extrema A method of finding the extrema for spline fitting.

Notes

As you can see from the definition, this method is tuned to work only in the case needed by EMD. This method is made available mainly for visualization and unit testing purposes. Better general purpose spline methods exist already in `scipy.interpolate`.

1.3.3 Utility code: pyeemd.utils

Some utility functions for visualizing IMFs produced by the (E)EMD methods.

`utils.`**`plot_imfs`**(*imfs*, *new_figs=True*, *plot_splines=True*)

Plot utility method for plotting IMFs and their envelope splines with `pylab`.

Parameters **imfs** : ndarray

The IMFs as returned by *`pyeemd.emd()`*, *`pyeemd.eemd()`*, or *`pyeemd.ceemdan()`*.

new_figs : bool, optional

Whether to plot the IMFs in separate figures.

plot_splines : bool, optional

Whether to plot the envelope spline curves as well.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

[R1] Z. Wu and N. Huang, “Ensemble Empirical Mode Decomposition: A Noise-Assisted Data Analysis Method”, Advances in Adaptive Data Analysis, Vol. 1 (2009) 1–41

[R2] N. E. Huang, Z. Shen and S. R. Long, “A new view of nonlinear water waves: The Hilbert spectrum”, Annual Review of Fluid Mechanics, Vol. 31 (1999) 417–457

[R3] M. Torres et al, “A Complete Ensemble Empirical Mode Decomposition with Adaptive Noise” IEEE Int. Conf. on Acoust., Speech and Signal Proc. ICASSP-11, (2011) 4144–4147

[R4] Z. Wu and N. Huang, “Ensemble Empirical Mode Decomposition: A Noise-Assisted Data Analysis Method”, Advances in Adaptive Data Analysis, Vol. 1 (2009) 1–41

u

utils, 6

u

utils, 6

C

ceemdan() (in module pyeemd), 4

E

eemd() (in module pyeemd), 3
emd() (in module pyeemd), 4
emd_evaluate_spline() (in module pyeemd), 5
emd_find_extrema() (in module pyeemd), 5
emd_num_imfs() (in module pyeemd), 5

P

plot_imfs() (in module utils), 6

U

utils (module), 6