PyEMD Documentation

Release 0.2.9

Dawid Laszuk

Apr 10, 2020

Table of Content

Writing documentation is hard. If more clarifications are needed, or you think others might benefit from extra explanation, don't hesitate to contact me through contact page.

Intro

1.1 General

PyEMD is a Python implementation of Empirical Mode Decomposition (EMD) and its variations. One of the most popular expansion is Ensemble Empirical Mode Decomposition (EEMD), which utilises an ensemble of noise-assisted executions.

As a result of *EMD* one will obtain a set of components that possess oscillatory features. In case of plain *EMD* algorithm, these are called Intrinsic Mode Functions (IMFs) as they are expected to have a single mode. In contrary, *EEMD* will unlikely produce pure oscillations as the effects of injected noise can propagate throughout the decomposition.

1.2 Installation

1.2.1 Recommended

In order to get the newest version it is recommended to download source code from git repository. **Don't worry, installation is simple.** Simply download this directory either directly from GitHub, or using command line:

```
$ git clone https://github.com/laszukdawid/PyEMD
```

Then go into the downloaded project and run from command line:

```
$ python setup.py install
```

1.2.2 PyPi (simplest)

Packaged obtained from PyPi is/will be slightly behind this project, so some features might not be the same. However, it seems to be the easiest/nicest way of installing any Python packages, so why not this one?

```
$ pip install EMD-signal
```

Usage

2.1 Typical Usage

Majority, if not all, methods follow the same usage pattern:

- Import method
- Initiate method
- Apply method on data

On vanilla EMD this is as

```python
from PyEMD import EMD
emd = EMD()
imfs = emd(s)
```

2.2 Parameters

The decomposition can be changed by adjusting parameters related to either sifting or stopping conditions.

2.2.1 Sifting

The sifting depends on the used method so these parameters ought to be looked within the methods. However, the typical parameters relate to spline method or the number of mirroring points.

2.2.2 Stopping conditions

All methods have the same two conditions, *FIXE* and *FIXE_H*, for stopping which relate to the number of sifting iterations. Setting parameter *FIXE* to any positive value will fix the number of iterations for each IMF to be exactly *FIXE*.

Example:

```
emd = EMD()
emd.FIXE = 10
imfs = emd(s)
```

Parameter *FIXE_H* relates to the number of iterations when the proto-IMF signal fulfils IMF conditions, i.e. number of extrema and zero-crossings differ at most by one and the mean is close to zero. This means that there will be at least *FIXE_H* iteration per IMF.

Example:

```
emd = EMD()
emd.FIXE_H = 5
imfs = emd(s)
```

When both *FIXE* and *FIXE_H* are 0 then other conditions are checked. These can be checking for convergence between consecutive iterations or whether the amplitude of output is below acceptable range.

CHAPTER 3

Speedup tricks

EMD is inherently slow with little chances on improving its performance. This is mainly due to it being a serial method. That's both on within IMF stage, i.e. iterative sifting, or between IMFs, i.e. the next IMF depends on the previous. On top of that, the common configuration of the EMD uses the natural cubic spline to span envelopes, which in turn additionally decreases performance since it depends on all extrema in the signal.

Since the EMD is the basis for other methods like EEMD and CEEMDAN these will also suffer from the same problem. What's more, these two methods perform the EMD many (hundreds) times which significantly increases any imperfections. It is expected that when it'll take more than a minute to perform an EEMD/CEEMDAN with default settings on a 10k+ samples long signal with a "medium complexity". There are, however, a couple of tweaks one can do to do make the computation finish sooner.

Sections below describe a tweaks one can do to improve performance of the EMD. In short, these changes are:

- *Change data type* (downscale)
- *Change spline method* to piecewise
- *Decrease number of trials*
- *Limit numer of output IMFs*

3.1 Change data type

Many programming frameworks by default casts numerical values to the largest data type it has. In case of Python's Numpy that's going to be numpy.float64. It's unlikely that one needs such resolution when using EMD*[0]. A suggestion is to downcast your data, e.g. to float16. The PyEMD should handle the same data type without upcasting but it can be additionally enforce a specific data type. To enable data type enforcement one needs to pass the DTYPE, i.e.

---

[0] I, the PyEMD's author, will go even a bit further. If one needs such large resolution then the EMD is not suitable for them. The EMD is not robust. Hundreds of iterations make any small difference to be emphasised and potentially leading to a significant change in final decomposition. This is the reason for creating EEMD and CEEMDAN which add small perturbation in a hope that the ensemble provides a robust solution.

```
from PyEMD import EMD

emd = EMD(DTYPE=np.float16)
```

3.2 Change spline method

EMD was presented with the natural cubic spline method to span envelops and that's the default option in the PyEMD. It's great for signals with not many extrema but its not suggested for longer/more complex signals. The suggestion is to change the spline method to some piecewise splines like 'Akima' or 'piecewise cubic'.

Example:

```
from PyEMD import EEMD

eemd = EEMD(spline_kind='akima')
```

3.3 Decrease number of trials

This relates more to EEMD and CEEMDAN since they perform an EMD a multiple times with slightly modified signal. It's difficult to choose a correct number of iterations. This definitely relates to the signal in question. The more iterations the more certain that the solution is convergent but there is likely a point beyond which more evaluations change little. On the other side, the quicker we can get output the quicker we can use it.

In the PyEMD, the number of iterations is referred to by *trials* and it's an explicit parameter to EEMD and CEEMDAN. The default value was selected arbitrarily and it's most likely wrong. An example on updating it:

```
from PyEMD import CEEMDAN

ceemdan = CEEMAN(trials=20)
```

3.4 Limit numer of output IMFs

Each method, by default, will perform decomposition until all components are returned. However, many use cases only require the first component. One can limit the number of returned components by setting up an implicit variable *max_imf* to the desired value.

Example:

```
from PyEMD import EEMD

eemd = EEMD(max_imfs=2)
```

Example

Some examples can be found in PyEMD/example directory.

4.1 EMD

4.1.1 Quick start

In most cases default settings are enough. Simply import `EMD` and pass your signal to *emd* method.

```python
from PyEMD import EMD

s = np.random.random(100)
emd = EMD()
IMFs = emd.emd(s)
```

Something more

Here is a complete script on how to create and plot results.

```python
from PyEMD import EMD
import numpy  as np
import pylab as plt

# Define signal
t = np.linspace(0, 1, 200)
s = np.cos(11*2*np.pi*t*t) + 6*t*t

# Execute EMD on signal
IMF = EMD().emd(s,t)
N = IMF.shape[0]+1
```
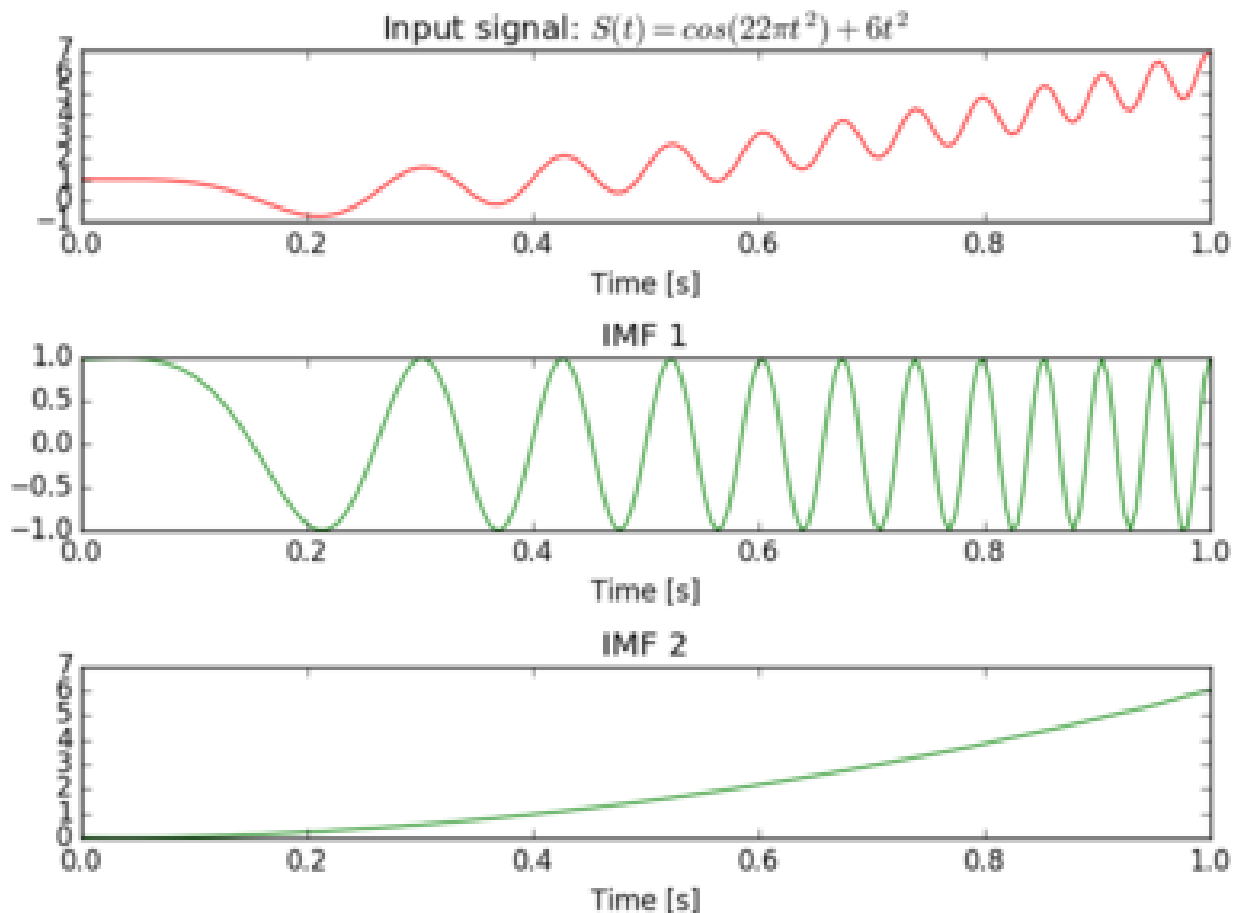
```
# Plot results
plt.subplot(N,1,1)
plt.plot(t, s, 'r')
plt.title("Input signal: $S(t)=cos(22\pi t^2) + 6t^2$")
plt.xlabel("Time [s]")

for n, imf in enumerate(IMF):
    plt.subplot(N,1,n+2)
    plt.plot(t, imf, 'g')
    plt.title("IMF "+str(n+1))
    plt.xlabel("Time [s]")

plt.tight_layout()
plt.savefig('simple_example')
plt.show()
```

The Figure below was produced with input:

$$S(t) = cos(22\pi t^2) + 6t^2$$

4.2 EEMD

Simplest case of using Esnembld EMD (EEMD) is by importing *EEMD* and passing your signal to *eemd* method.

```python
from PyEMD import EEMD
import numpy as np
import pylab as plt

# Define signal
t = np.linspace(0, 1, 200)

sin = lambda x,p: np.sin(2*np.pi*x*t+p)
S = 3*sin(18,0.2)*(t-0.2)**2
S += 5*sin(11,2.7)
S += 3*sin(14,1.6)
S += 1*np.sin(4*2*np.pi*(t-0.8)**2)
S += t**2.1 -t

# Assign EEMD to `eemd` variable
eemd = EEMD()

# Say we want detect extrema using parabolic method
emd = eemd.EMD
emd.extrema_detection="parabol"

# Execute EEMD on S
eIMFs = eemd.eemd(S, t)
nIMFs = eIMFs.shape[0]

# Plot results
plt.figure(figsize=(12,9))
plt.subplot(nIMFs+1, 1, 1)
plt.plot(t, S, 'r')

for n in range(nIMFs):
    plt.subplot(nIMFs+1, 1, n+2)
    plt.plot(t, eIMFs[n], 'g')
    plt.ylabel("eIMF %i" % (n+1))
    plt.locator_params(axis='y', nbins=5)

plt.xlabel("Time [s]")
plt.tight_layout()
plt.savefig('eemd_example', dpi=120)
plt.show()
```
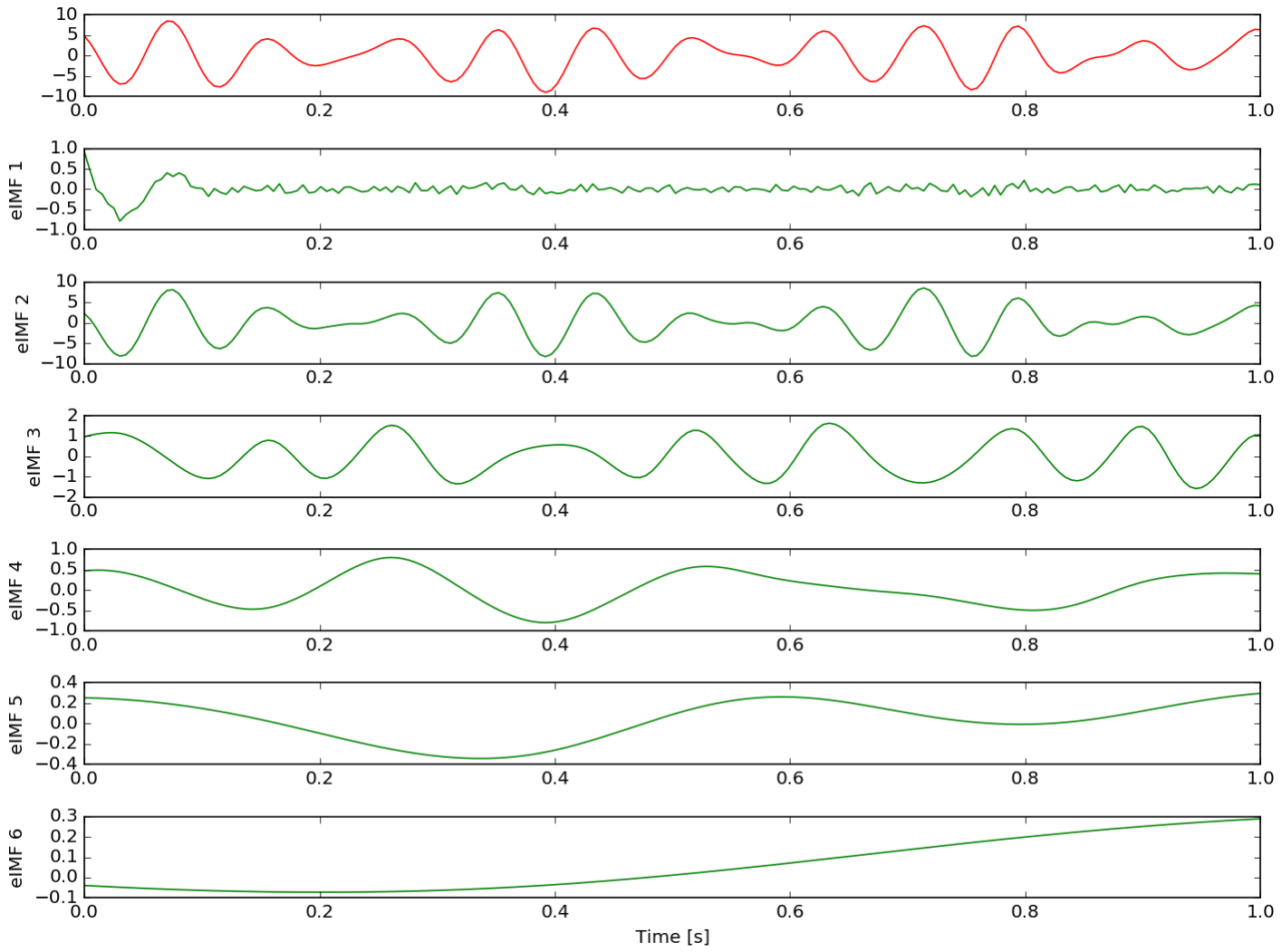
EMD

Empirical Mode Decomposition (EMD) is an iterative procedure which decomposes signal into a set of oscillatory components, called *Intrisic Mode Functions* (IMFs).

class `PyEMD`.**EMD**(*spline_kind='cubic'*, *nbsym=2*, ***config*)
Empirical Mode Decomposition

Method of decomposing signal into Intrinsic Mode Functions (IMFs) based on algorithm presented in Huang et al. [R7d832100fd35-Huang1998].

Algorithm was validated with Rilling et al. [R7d832100fd35-Rilling2003] Matlab's version from 3.2007.

Threshold which control the goodness of the decomposition:

- *std_thr* — Test for the proto-IMF how variance changes between siftings.

- *svar_thr* – Test for the proto-IMF how energy changes between siftings.

- *total_power_thr* — Test for the whole decomp how much of energy is solved.

- *range_thr* — Test for the whole decomp whether the difference is tiny.

Parameters

spline_kind [string, (default: 'cubic')] Defines type of spline, which connects extrema. Possible: cubic, akima, slinear.

nbsym [int, (default: 2)] Number of extrema used in boundary mirroring.

extrema_detection [string, (default: 'simple')] How extrema are defined.

- *simple* - Extremum is above/below neighbours.

- *parabol* - Extremum is a peak of a parabola.

References

[R7d832100fd35-Huang1998], [R7d832100fd35-Rilling2003]

Examples

```
>>> import numpy as np
>>> T = np.linspace(0, 1, 100)
>>> S = np.sin(2*2*np.pi*T)
>>> emd = EMD()
>>> emd.extrema_detection = "parabol"
>>> IMFs = emd.emd(S)
>>> IMFs.shape
(1, 100)
```

__init__ (*self*, *spline_kind='cubic'*, *nbsym=2*, ***config*)
    Initiate *EMD* instance.

    Configuration, such as threshold values can be passed as config.

```
>>> config = {"std_thr": 0.01, "range_thr": 0.05}
>>> emd = EMD(**config)
```

check_imf (*self*, *imf_new*, *imf_old*, *eMax*, *eMin*)
    Huang criteria for **IMF** (similar to Cauchy convergence test). Signal is an IMF if consecutive siftings do not affect signal in a significant manner.

emd (*self*, *S*, *T=None*, *max_imf=-1*)
    Performs Empirical Mode Decomposition on signal S. The decomposition is limited to *max_imf* imfs. Returns IMF functions in numpy array format.

    Parameters

    S [numpy array,] Input signal.

    T [numpy array, (default: None)] Position or time array. If None passed or if self.extrema_detection == "simple", then numpy arange is created.

    max_imf [int, (default: -1)] IMF number to which decomposition should be performed. Negative value means *all*.

    Returns

    IMF [numpy array] Set of IMFs produced from input signal.

end_condition (*self*, *S*, *IMF*)
    Tests for end condition of whole EMD. The procedure will stop if:

    • Absolute amplitude (max - min) is below *range_thr* threshold, or

    • Metric L1 (mean absolute difference) is below *total_power_thr* threshold.

    Parameters

    S [numpy array] Original signal on which EMD was performed.

    IMF [numpy 2D array] Set of IMFs where each row is IMF. Their order is not important.

    Returns

    end [bool] Whether sifting is finished.

extract_max_min_spline (*self*, *T*, *S*)
    Extracts top and bottom envelopes based on the signal, which are constructed based on maxima and minima, respectively.

    Parameters

> > **T** [numpy array] Position or time array.
>
> > **S** [numpy array] Input data S(T).
>
> > **Returns**
>
> > > **max_spline** [numpy array] Spline spanned on S maxima.
> >
> > > **min_spline** [numpy array] Spline spanned on S minima.

find_extrema(*self*, *T*, *S*)

> Returns extrema (minima and maxima) for given signal S. Detection and definition of the extrema depends on `extrema_detection` variable, set on initiation of EMD.
>
> > **Parameters**
> >
> > > **T** [numpy array] Position or time array.
> > >
> > > **S** [numpy array] Input data S(T).
> >
> > **Returns**
> >
> > > **local_max_pos** [numpy array] Position of local maxima.
> > >
> > > **local_max_val** [numpy array] Values of local maxima.
> > >
> > > **local_min_pos** [numpy array] Position of local minima.
> > >
> > > **local_min_val** [numpy array] Values of local minima.

get_imfs_and_residue(*self*)

> Provides access to separated imfs and residue from recently analysed signal. :return: (imfs, residue)

prepare_points(*self*, *T*, *S*, *max_pos*, *max_val*, *min_pos*, *min_val*)

> Performs extrapolation on edges by adding extra extrema, also known as mirroring signal. The number of added points depends on *nbsym* variable.
>
> > **Parameters**
> >
> > > **S** [numpy array] Input signal.
> > >
> > > **T** [numpy array] Position or time array.
> > >
> > > **max_pos** [iterable] Sorted time positions of maxima.
> > >
> > > **max_vali** [iterable] Signal values at max_pos positions.
> > >
> > > **min_pos** [iterable] Sorted time positions of minima.
> > >
> > > **min_val** [iterable] Signal values at min_pos positions.
> >
> > **Returns**
> >
> > > **min_extrema** [numpy array (2 rows)] Position (1st row) and values (2nd row) of minima.
> > >
> > > **min_extrema** [numpy array (2 rows)] Position (1st row) and values (2nd row) of maxima.

spline_points(*self*, *T*, *extrema*)

> Constructs spline over given points.
>
> > **Parameters**
> >
> > > **T** [numpy array] Position or time array.
> > >
> > > **extrema** [numpy array] Position (1st row) and values (2nd row) of points.
> >
> > **Returns**
> >
> > > **T** [numpy array] Position array (same as input).

spline [numpy array] Spline array over given positions T.

EEMD

6.1 Info

Ensemble empirical mode decomposition (EEMD) creates an ensemble of worker each of which performs an EMD on a copy of the input signal with added noise. When all workers finish their work a mean over all workers is considered as the true result.

6.2 Class

class `PyEMD`.**EEMD** (*trials=100*, *noise_width=0.05*, *ext_EMD=None*, *parallel=True*, ***config*)
Ensemble Empirical Mode Decomposition

Ensemble empirical mode decomposition (EEMD) [Re8a0cfda063a-Wu2009] is noise-assisted technique, which is meant to be more robust than simple Empirical Mode Decomposition (EMD). The robustness is checked by performing many decompositions on signals slightly perturbed from their initial position. In the grand average over all IMF results the noise will cancel each other out and the result is pure decomposition.

Parameters

trials [int (default: 100)] Number of trials or EMD performance with added noise.

noise_width [float (default: 0.05)] Standard deviation of Gaussian noise ($\hat{\sigma}$). It's relative to absolute amplitude of the signal, i.e. $\hat{\sigma} = \sigma \cdot |\max(S) - \min(S)|$, where σ is noise_width.

ext_EMD [EMD (default: None)] One can pass EMD object defined outside, which will be used to compute IMF decompositions in each trial. If none is passed then EMD with default options is used.

References

[Re8a0cfda063a-Wu2009]

eemd (*self*, *S*, *T=None*, *max_imf=-1*)
Performs EEMD on provided signal.

For a large number of iterations defined by *trials* attr the method performs *emd()* on a signal with added white noise.

> **Parameters**
>
> > **S** [numpy array,] Input signal on which EEMD is performed.
> >
> > **T** [numpy array, (default: None)] If none passed samples are numerated.
> >
> > **max_imf** [int, (default: -1)] Defines up to how many IMFs each decomposition should be performed. By default (negative value) it decomposes all IMFs.
>
> **Returns**
>
> > **eIMF** [numpy array] Set of ensemble IMFs produced from input signal. In general, these do not have to be, and most likely will not be, same as IMFs produced using EMD.

emd (*self*, *S*, *T*, *max_imf=-1*)
Vanilla EMD method.

Provides emd evaluation from provided EMD class. For reference please see *PyEMD.EMD*.

generate_noise (*self*, *scale*, *size*)
Generate noise with specified parameters. Currently supported distributions are:

- *normal* with std equal scale.

- *uniform* with range [-scale/2, scale/2].

> **Parameters**
>
> > **scale** [float] Width for the distribution.
> >
> > **size** [int] Number of generated samples.
>
> **Returns**
>
> > **noise** [numpy array] Noise sampled from selected distribution.

get_imfs_and_residue (*self*)
Provides access to separated imfs and residue from recently analysed signal. :return: (imfs, residue)

noise_seed (*self*, *seed*)
Set seed for noise generation.

BEMD

7.1 Warning

Important This is an experimental module. Please use it with care as no guarantee can be given for obtaining reasonable results, or that they will be computed index the most computation optimal way.

7.2 Info

BEMD performed on bidimensional data such as images. This procedure uses morphological operators to detect regional maxima which are then used to span surface envelope with a radial basis function.

7.3 Class

EMD2D

8.1 Warning

Important This is an experimental module. Please use it with care as no guarantee can be given for obtaining reasonable results, or that they will be computed index the most computation optimal way.

8.2 Info

EMD performed on images. This version uses for envelopes 2D splines, which are span on extrema defined through maximum filter.

8.3 Class

class `PyEMD.EMD2d.`**`EMD2D`**(***config*)

Empirical Mode Decomposition on images.

Important This is an experimental module. Experiments performed using this module didn't provide acceptable results, either in actual output nor in computation performance. The author is not an expert in image processing so it's very likely that the code could have been improved. Take your best shot.

Method decomposes images into 2D representations of loose Intrinsic Mode Functions (IMFs).

The current version of the algorithm detects local extrema, separately minima and maxima, and then connects them to create envelopess. These are then used to create a mean trend and subtracted from the input.

Threshold values that control goodness of the decomposition:

- *mse_thr* — proto-IMF check whether small mean square error.

- *mean_thr* — proto-IMF chekc whether small mean value.

check_proto_imf(*self*, *proto_imf*, *proto_imf_prev*, *mean_env*)
> Check whether passed (proto) IMF is actual IMF. Current condition is solely based on checking whether the mean is below threshold.
>
> > **Parameters**
> >
> > > **proto_imf** [numpy 2D array] Current iteration of proto IMF.
> > >
> > > **proto_imf_prev** [numpy 2D array] Previous iteration of proto IMF.
> > >
> > > **mean_env** [numpy 2D array] Local mean computed from top and bottom envelopes.
> >
> > **Returns**
> >
> > > **boolean** Whether current proto IMF is actual IMF.

emd(*self*, *image*, *max_imf=-1*)
> Performs EMD on input image with specified parameters.
>
> > **Parameters**
> >
> > > **image** [numpy 2D array,] Image which will be decomposed.
> > >
> > > **max_imf** [int, (default: -1)] IMF number to which decomposition should be performed. Negative value means *all*.
> >
> > **Returns**
> >
> > > **IMFs** [numpy 3D array] Set of IMFs in form of numpy array where the first dimension relates to IMF's ordinary number.

classmethod end_condition(*cls*, *image*, *IMFs*)
> Determins whether decomposition should be stopped.
>
> > **Parameters**
> >
> > > **image** [numpy 2D array] Input image which is decomposed.
> > >
> > > **IMFs** [numpy 3D array] Array for which first dimensions relates to respective IMF, i.e. (numIMFs, imageX, imageY).

extract_max_min_spline(*self*, *image*)
> Calculates top and bottom envelopes for image.
>
> > **Parameters**
> >
> > > **image** [numpy 2D array]
> >
> > **Returns**
> >
> > > **min_env** [numpy 2D array] Bottom envelope in form of an image.
> > >
> > > **max_env** [numpy 2D array] Top envelope in form of an image.

classmethod find_extrema(*cls*, *image*)
> Finds extrema, both mininma and maxima, based on local maximum filter. Returns extrema in form of two rows, where the first and second are positions of x and y, respectively.
>
> > **Parameters**
> >
> > > **image** [numpy 2D array] Monochromatic image or any 2D array.
> >
> > **Returns**
> >
> > > **min_peaks** [numpy array] Minima positions.
> > >
> > > **max_peaks** [numpy array] Maxima positions.

classmethod prepare_image(*cls*, *image*)

Prepares image for edge extrapolation. Method bloats image by mirroring it along all axes. This turns extrapolation on edges into interpolation within bigger image.

> **Parameters**
>
> > **image** [numpy 2D array] Image for which interpolation is required,
>
> **Returns**
>
> > **image** [numpy 2D array] Big image based on the input. Grid 3x3 where the center block is input and neighbouring panels are respective mirror images.

classmethod spline_points(*cls*, *X*, *Y*, *Z*, *xi*, *yi*)

Interpolates for given set of points

CEEMDAN

9.1 Info

Complete ensembe EMD with adaptive noise (CEEMDAN) performs an EEMD with the difference that the information about the noise is shared among all workers.

9.2 Class

class `PyEMD`.**CEEMDAN** (*trials=100*, *epsilon=0.005*, *ext_EMD=None*, *parallel=True*, ***config*)
   "Complete Ensemble Empirical Mode Decomposition with Adaptive Noise"

   "Complete ensemble empirical mode decomposition with adaptive noise" (CEEMDAN) [Rd0298acbfa1b-Torres2011] is noise-assisted EMD technique. Word "complete" presumably refers to decomposing completly everything, even added perturbation (noise).

   Provided implementation contains proposed "improvmenets" from paper [Rd0298acbfa1b-Colominas2014].

   Any parameters can be updated directly on the instance or passed through a *configuration* dictionary.

   Goodness of the decomposition can be configured by modifying threshold values. Two are *range_thr* and *total_power_thr* which relate to the value range (max - min) and check for total power below, respectively.

   Parameters

   trials [int (default: 100)] Number of trials or EMD performance with added noise.

   epsilon [float (default: 0.005)] Scale for added noise (ϵ) which multiply std σ: $\beta = \epsilon \cdot \sigma$

   ext_EMD [EMD (default: None)] One can pass EMD object defined outside, which will be used to compute IMF decompositions in each trial. If none is passed then EMD with default options is used.

References

[Rd0298acbfa1b-Torres2011], [Rd0298acbfa1b-Colominas2014]

__init__ (*self*, *trials=100*, *epsilon=0.005*, *ext_EMD=None*, *parallel=True*, ***config*)
Configuration can be passed through config dictionary. For example, updating threshold would be through:

```
>>> config = {"range_thr": 0.001, "total_power_thr": 0.01}
>>> emd = EMD(**config)
```

emd (*self*, *S*, *T*, *max_imf=-1*)
Vanilla EMD method.

Provides emd evaluation from provided EMD class. For reference please see *PyEMD.EMD*.

end_condition (*self*, *S*, *cIMFs*, *max_imf*)
Test for end condition of CEEMDAN.

Procedure stops if:

- number of components reach provided *max_imf*, or

- last component is close to being pure noise (range or power), or

- set of provided components reconstructs sufficiently input.

> **Parameters**
>
>> **S** [numpy array] Original signal on which CEEMDAN was performed.
>>
>> **cIMFs** [numpy 2D array] Set of cIMFs where each row is cIMF.
>
> **Returns**
>
>> **end** [bool] Whether to stop CEEMDAN.

generate_noise (*self*, *scale*, *size*)
Generate noise with specified parameters. Currently supported distributions are:

- *normal* with std equal scale.

- *uniform* with range [-scale/2, scale/2].

> **Parameters**
>
>> **scale** [float] Width for the distribution.
>>
>> **size** [int] Number of generated samples.
>
> **Returns**
>
>> **noise** [numpy array] Noise sampled from selected distribution.

get_imfs_and_residue (*self*)
Provides access to separated imfs and residue from recently analysed signal. :return: (imfs, residue)

noise_seed (*self*, *seed*)
Set seed for noise generation.

Visualisation

A simple visualisation helper.

class `PyEMD`.**Visualisation**(*emd_instance=None*)
 Simple visualisation helper.

 This class is for quick and simple result visualisation.

 plot_imfs(*self*, *imfs=None*, *residue=None*, *t=None*, *include_residue=True*)
  Plots and shows all IMFs.

  All parameters are optional since the *emd* object could have been passed when instantiating this object.

  The residual is an optional and can be excluded by setting *include_residue=False*.

 plot_instant_freq(*self*, *t*, *imfs=None*, *order=False*, *alpha=None*)
  Plots and shows instantaneous frequencies for all provided imfs.

  The necessary parameter is *t* which is the time array used to compute the EMD. One should pass *imfs* if no *emd* instances is passed when creating the Visualisation object.

  Parameters

   order [bool (default: False)] Represents whether the finite difference scheme is low-order (1st order forward scheme) or high-order (6th order compact scheme). The default value is False (low-order)

   alpha [float (default: None)] Filter intensity. Default value is None, which is equivalent to *alpha* = 0.5, meaning that no filter is applied. The *alpha* values must be in between -0.5 (fully active) and 0.5 (no filter).

Contact

Feel free to send email with any questions, concerns or for whatever reason you feel like.

Email \laszuk\dawid\@\gmail.com (remove slashes \).

Homepage http://www.laszukdawid.com

GitHub You can also visit PyEMD GitHub project page for this project.

CHAPTER 12

Indices and tables

- genindex
- search

Bibliography

[R7d832100fd35-Huang1998] N. E. Huang et al., "The empirical mode decomposition and the Hilbert spectrum for non-linear and non stationary time series analysis", Proc. Royal Soc. London A, Vol. 454, pp. 903-995, 1998

[R7d832100fd35-Rilling2003] G. Rilling, P. Flandrin and P. Goncalves, "On Empirical Mode Decomposition and its algorithms", IEEE-EURASIP Workshop on Nonlinear Signal and Image Processing NSIP-03, Grado (I), June 2003

[Re8a0cfda063a-Wu2009] Z. Wu and N. E. Huang, "Ensemble empirical mode decomposition: A noise-assisted data analysis method", Advances in Adaptive Data Analysis, Vol. 1, No. 1 (2009) 1-41.

[Rd0298acbfa1b-Torres2011] M.E. Torres, M.A. Colominas, G. Schlotthauer, P. Flandrin A complete ensemble empirical mode decomposition with adaptive noise. Acoustics, Speech and Signal Processing (ICASSP), 2011, pp. 4144–4147

[Rd0298acbfa1b-Colominas2014] M.A. Colominas, G. Schlotthauer, M.E. Torres, Improved complete ensemble EMD: A suitable tool for biomedical signal processing, In Biomed. Sig. Proc. and Control, V. 14, 2014, pp. 19–29

Index