Submitted to *Annals of Actuarial Science*

Supplement to: Package **AdvEMDpy**: Algorithmic Variations of Empirical Mode Decomposition in Python

Cole van Jaarsveldt[1], Matthew Ames[2], Gareth W. Peters[3], and Mike Chantler[4]

[1]School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh, United Kingdom of Great Britain and Northern Ireland (UK), `cv25@hw.ac.uk` [2]ResilientML, Melbourne, Australia, `matt.ames@resilientml.com` [3]Department of Statistics & Applied Probability, University of California, Santa Barbara, California, United States of America (USA), `garethpeters@ucsb.edu` [4]School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh, UK, `m.j.chantler@hw.ac.uk`

Abstract

This work serves as a formal supplement to 'Package **AdvEMDpy**: Algorithmic Variations of Empirical Mode Decomposition in Python' with additional synthetic and real-world examples. All of Section 7, from the 'Package **AdvEMDpy**: Algorithmic Variations of Empirical Mode Decomposition in Python', will be repeated here verbatim with other viable algorithmic variations and further details on the extensions to the algorithm already alluded to in Section 7. This is done to make the original work more concise, whilst simultaneously providing those interested in further reading, examples, and algorithmic variations with further compelling literature. **AdvEMDpy** will be shown to be more accurate than its Python competitors in resolving the underlying driving function of the Duffing Equation, before it is used to isolate different frequency structures present in Carbon ETF data. An annual fluctuation will be extracted and possibly causally linked to the seasonal trend of the Carbon Dioxide concentration in the atmosphere. These examples are by no means exhaustive and merely serve as demonstrations of **AdvEMDpy**'s usage and superiority.

Keywords: Empirical Mode Decomposition (EMD), Statistical EMD (SEMD), Enhanced EMD (EEMD), Ensemble EMD, Hilbert transform, time series analysis, filtering, graduation, Winsorization, downsampling, splines, knot optimisation, Python, R, MATLAB

Software Availability

The software accompanying this paper is available on GitHub at:

https://github.com/Cole-vJ/AdvEMDpy.

Instructions on how to install, extensive worked examples, as well as the package versions required for complete reproducibility can all be found in the repository.

10 Core Details of AdvEMDpy Package

In this section, the details of each of the components of **AdvEMDpy** package are outlined both in how to interact with the package through specific functionalities and what expectations one should have on outputs created and features that can be customised.

10.1 Base Implementation of AdvEMDpy Package

Before discussing the nuances of the algorithmic variations, one would benefit from some choice remarks concerning non-essential, but helpful outputs of the **AdvEMDpy** package. These play no direct part in the implementation of the algorithm but assist in iterative understanding and assessment of the EMD decompositions performance. The base implementation of the algorithm may be seen below. More detailed descriptions may be found in the associated scripts as well as a base implementation in the associated `README.md` file.

```
# instantiate EMD class with time (optional
# keyword argument) and time series
emd = EMD(time=time, time_series=time_series)

# base implementation with optional helpful inputs shown
# knots and knot_time are optional keyword arguments
imfs, hts, ifs = emd.empirical_mode_decomposition(knots=knots,
                                                  knot_time=knot_time,
                                                  debug=False,
                                                  dft='envelopes',
                                                  verbose=True,
                                                  stopping_criterion='sd',
                                                  output_coefficients=False,
                                                  output_knots=False)[:3]
```

The details of each field in the input to the function are detailed in the following subsections. If time is not specified as a keyword argument, then the following code is implemented where time is simply an index set of the same length as the original time series.

```
self.time = np.arange(len(self.time_series))
```

If the knot sequence or the associated knot time are also unspecified keyword arguments, then the following code is implemented that sets knot time equal to time and spaces the knots 10 times further apart than the time points. This is problematic if the default knot sequence is insufficient to capture the highest frequency structures which is demonstrated in `new_user.ipynb`.

```
knots = np.linspace(0, self.time[-1], int(len(self.time) / 10 + 1))
knot_time = self.time
```

The only required argument is the time series. This is detailed further in `new_user.ipynb` which is intended for new users to get comfortable with the package and the available options.

10.1.1 Debug flag in AdvEMDpy Package

If true, each iteration of the local mean estimation through the chosen detrended mean threshold technique will be plotted for analysis. The debugging output displayed depends on the choice of `dft`. If `dft='envelopes'` (base implementation), the extrema, the extrema envelopes, and the calculated local mean will be plotted. If `dft='inflection_points'` the extrema, inflection points and the associated spline fitted through the inflection points will be displayed. If `dft='binomial_average'` the extrema, the points used for the binomial average, the binomial average of these points, and the associated spline fitted will be displayed. Finally, if `dft='enhanced'`, the extrema, the optimal 'extrema', the associated splines, and the resulting local mean are displayed.

10.1.2 Verbose flag in AdvEMDpy Package

If true, each iteration of the algorithm will output text describing if the stopping criterion chosen or the mean threshold is met. The intermediate numbering of the potential IMFs is output for consistency and the iteration counter is printed if the maximum number of iterations is met. Different text is displayed for each of the optional stopping criteria. If, for example, `stopping_criterion='sd'` then the text displayed for the fifth candidate for the second IMF, should the IMF candidate not meet the stopping criterion, will be `IMF_25 Standard deviation STOPPING CRITERION NOT MET with sd = 1000.0` or it will be `IMF_25 Standard deviation STOPPING CRITERION MET with sd = 0.05 < sd threshold = 0.1` with the stopping criterion threshold being another possible input such as `stopping_criterion_threshold=0.1`, if the stopping criterion condition is met. For `stopping_criterion_threshold`, a value of $0.2 - 0.3$ is recommended in the paper that originally proposed the method, Huang et al. (1998). This value should be time series dependent as the value is very much dependent on the level of noise in the time series as well as the number of time points available, but more on this (and other stopping criteria) in Section 10.5.

10.1.3 *Output Coefficients flag in AdvEMDpy Package*

If `output_coefficients=True`, cubic B-spline coefficients corresponding to each IMF output (including initial smoothed time series and trend) are output. One must remember the six additional coefficients corresponding to the three additional knots on either edge of the time series as demonstrated in Figure 1. These additional bases result in implicit smoothing through a non-natural spline which does not tend to zero at the edges in both position and the higher-order derivatives. It would be unnecessarily restrictive to impose a natural spline on the sifting algorithm and the resulting IMFs.

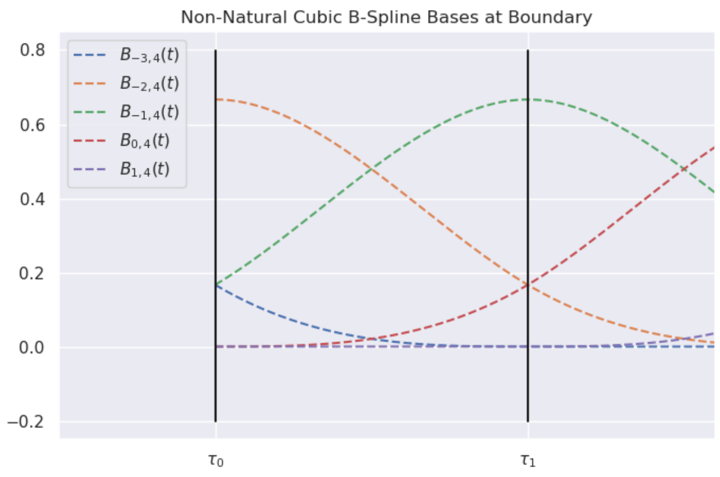

Figure 1: Figure demonstrating incomplete bases at the boundary of the time series to create non-natural cubic B-splines that can accommodate non-zero edge values, derivatives, and curvatures.

10.1.4 *Output Knots flag in AdvEMDpy Package*

If `output_knots=True`, knot points are output. If knot points are not optimised, that is if `optimise_knots=0`, then original knot points are repeated at each iteration of the sifting procedure when extracting recursively each IMF. In other words, all IMF's will have the same knot sequence. If the user provides a uniform knot sequence and chooses not to optimise the knot locations, then a uniform knot sequence will be used throughout as demonstrated in Figure 2.

Alternatively, one can optimize the knots on an original spline (`optimise_knots=1`) representation of the input time series signal using optimal knot placement methodology as outlined in Section 10.8 where two options are available using simple bisection and serial bisection, but more about this will follow in the dedicated section. In this way the knot points will still be universal for all IMF's however, they will be optimized for the given signal input and will not dynamically adjust as the residuals become increasingly less complex. For this reason, this can be referred to as the statically optimised knots and an example demonstrating this can be seen in Figure 3 where the optimised non-uniform knots can be seen to be used throughout the algorithm despite the decreasing complexity.

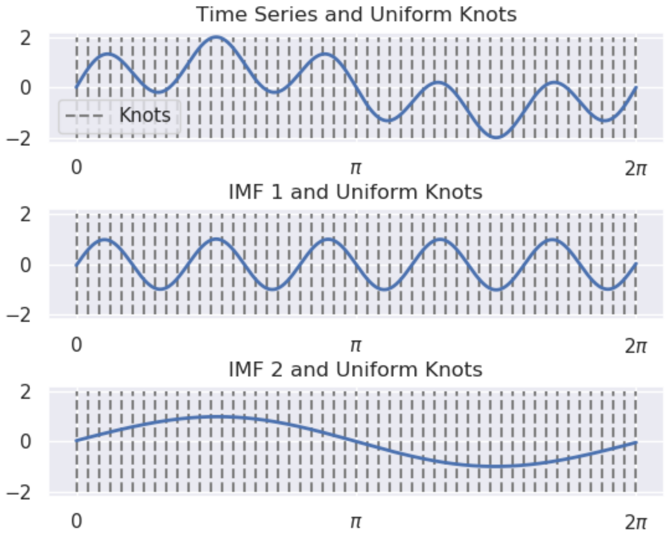

Figure 2: Figure demonstrating predefined uniform knot placement and the resulting IMFs.

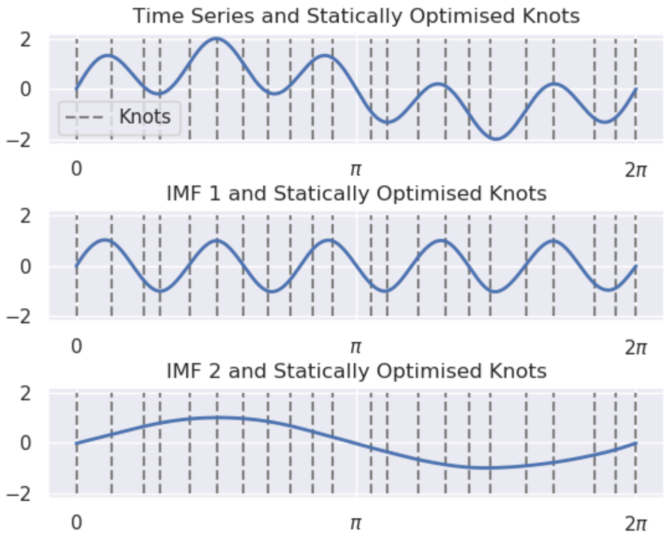

Figure 3: Figure demonstrating statically optimised knot placement, which is optimised once at outset and used throughout the sifting, and the resulting IMFs.

In addition, one could also optimize the knot points per iteration (`optimise_knots=2`) of the EMD sifting procedure once extracting an IMF, the next iteration of EMD on the residual signal could first have the knot point optimisation performed when fitting the spline to the residual to proceed with the next rounds of sifting to extract the next IMF. This results in different numbers of knot points per IMF and different placements, in general one would expect to require far fewer knots for later IMFs extracted which have lower frequency content compared to those first few IMF bases extracted. An example demonstrating this can be seen in Figure 4 where there are far fewer non-uniform knots for the second IMF than was required for the first IMF.

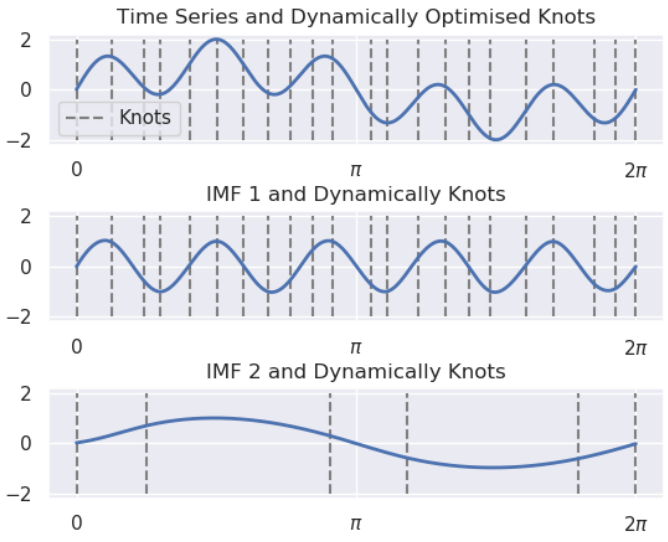

Figure 4: Figure demonstrating dynamically optimised knot placement, which is optimised at the beginning of each internal sifting routine, and the resulting IMFs.

Performing the third option may increase the speed of the algorithm if a large number of siftings need to take place and a large number of IMFs need to be extracted, despite the added time required to find the optimised knots at each stage. Progressively fewer knots will be required throughout the algorithm with the added advantage of having a more parsimonious representation with fewer parameters required for higher-order IMFs which have guaranteed reducing oscillation in their representation.

10.1.5 Recommendations for Base Implementation of AdvEMDpy Package

For the initial application of the algorithm to the desired time series, it is recommended that the user display the results of the selected stopping criterion (`verbose=True`) and run to completion. The user can then inspect the outputs and should anything be irregular or if the user wants to observe each iterative step of the algorithm one can use `debug=True` which will cause the method to plot each iteration of the algorithm.

```
# recommended initial base implementation
imfs, hts, ifs = emd.empirical_mode_decomposition(knots=knots,
                                                  knot_time=knot_time,
                                                  debug=True,
                                                  verbose=True)[:3]
```

10.2 Preprocessing flag in AdvEMDpy Package

The raw time series signals that can be passed to the EMD package may contain a wide variety of structures. This could include different non-linear structures, non-stationarity features in trend, volatility etc. as well as corruption by observation noise that may be present in the collection of the data. Therefore, in the **AdvEMDpy** package there is the option to pre-process the time series data input to reduce the effect of these features if they may be adverse to the user's analysis. This is particularly the case in the context of signals observed in various noise environments.

Therefore, before the sifting algorithm is implemented, there are several choices available for the pre-processing of the raw time series. This was a necessary inclusion in the algorithm owing to the wide variety of time series and their potentially vastly different statistical characteristics that would need to be accommodated in the EMD basis decomposition. These preprocessing techniques may be broadly grouped into filtering and smoothing. The filtering methods developed out of a necessity to mitigate the corruption of the IMFs that would otherwise arise as a result of the permeation of error that would occur due to the presence in the input time series signal of noise corruptions such as heavy-tailed noise, mixed noise (Gaussian noise with different variances), and Poisson noise. Smoothing developed out of the broader field of trend extraction amongst cyclical components where within EMD defining an individual cyclical component amongst others and within a non-stationary setting becomes increasingly challenging. The base implementation is as follows:

```
imfs, hts, ifs = \
    emd.empirical_mode_decomposition(knots=knots,
                                     knot_time=knot_time,
                                     initial_smoothing=True,
                                     preprocess='none',
                                     preprocess_window_length=51,
                                     preprocess_quantile=0.9,
                                     preprocess_penalty=1,
                                     preprocess_order=13,
                                     preprocess_norm_1=2,
                                     preprocess_norm_2=1,
                                     downsample_window='hamming',
                                     downsample_decimation_factor=20,
                                     downsample_window_factor=20)[:3].
```

The following subsections will outline the different functionalities these flags provide in the pre-processing options that users can select in the **AdvEMDpy** package.

10.2.1 *Filtering*

Filtering uses a localising window to filter extrema out of the time series before the sifting procedure is performed. The preprocessing window length must be an odd integer as the window centres on a particular time point and calculates the new preprocessed time series accordingly. As an example, if `preprocess='median_filter'` and `preprocess_window_length=51`, the central time series point is replaced with the median of the 51 time series points. All filtering techniques are displayed in Figure 5 for quick reference with quantile filters used by Winsorization filtering included assisting with understanding.

Mean Filter - Each point in the time series is replaced with the mean to mitigate the effects of outliers (`preprocess='mean_filter'`) over the specified window width centred on the specific point. This is extremely susceptible to outliers and is therefore far less robust than the other filters, but is included for classical usage and completeness-sake.

Median Filter - Each point in the time series is replaced with the median of the points over the specified window width (`preprocess='median_filter'`). This technique is far less susceptible to outliers with it forming the most basic technique in a toolbox of robust statistics.

Winsorization - Both this method and the one below are based on the ground-breaking work in Hastings Jr. et al. (1947). A local quantile window is created for every point such that extreme values are restricted by making them equal to the boundary values (`preprocess='winsorize'`). Should the point be above the maximum quantile it is replaced with the maximum quantile value at that point and if it is below the minimum quantile it is replaced with the minimum quantile value at that point. This is a slightly more complex robust statistical technique in that it will leave the majority of the time series untouched and simply restrict the extreme values. The level of the boundary is controlled with `preprocess_quantile=0.9` - this would result in the maximum boundary being such that 95% of the time series within the window is below the maximum boundary and 5% of the time series within the window is below minimum boundary - i.e. 90% of the time series is within the boundaries.

Winsorization Interpolation - As above, a local quantile window is created for every point, except the extreme values are treated in a slightly different manner than being made equal to the boundary values (`preprocess='winsorize_interpolate'`). All the points above or below the quantile windows are removed and they are linearly interpolated - this further removes extremes from the time series.

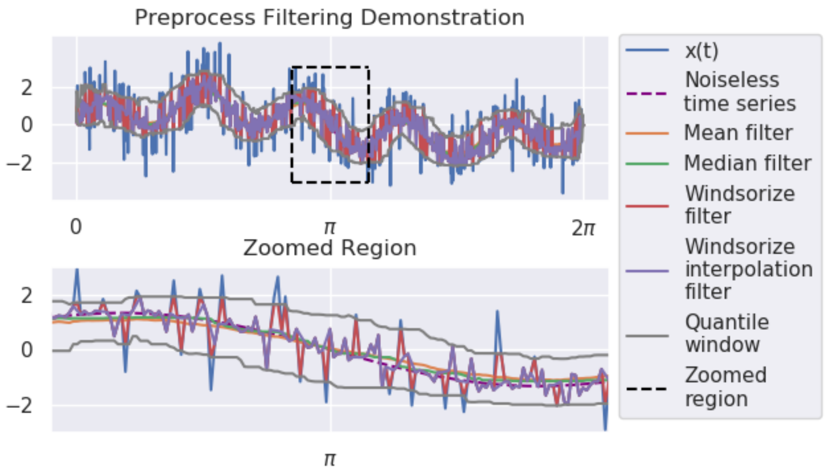

Figure 5: Figure demonstrating family of filtering methods available to the user with quantile envelope included for clarifying Winsorization calculation.

10.2.2 *Smoothing*

Rather than explicitly limiting the effects of extrema by filtering the original time series, one can fit a smoothing spline directly to the time series that implicitly limits the effect of extrema, whilst simultaneously removing less extreme noise present in the time series. The smoothing preprocessing options are displayed in Figure 6 where both an undecimated and decimated downsampled time series is shown.

Smoothing - If an initial smoothing (`initial_smoothing=True`) is not done, then the first IMF extracted will not be explainable in terms of cubic B-spline coefficients. It will mostly consist of random noise in the system with little-to-no physical significance as corroborated by numerous papers. This smoothing is implemented separately from other preprocessing techniques to allow smoothing to be performed on a time series once some preprocessing has been done. A natural example promoting this necessity would be a time series with mixed-noise.

Generalised Hodrick-Prescott - This is a generalised form of Hodrick-Prescott filtering (`preprocess='HP'`), introduced in Hodrick and Prescott (1997), but in this context it is classified under smoothing:

$$min_z\left(\sum_{t=0}^{T}(y_t - z_t)^{n_1} + \lambda\sum_{t=0}^{T-d}(\nabla^d z_t)^{n_2}\right), \tag{1}$$

where in the original Hodrick-Prescott filter assumes values $n_1 = 2$ (`preprocess_norm_1=2`), $n_2 = 2$ (`preprocess_norm_2=1`), the penalty term $\lambda = 1$ (`preprocess_penalty=1`) was not originally present, and $d = 2$ (`preprocess_order=2`) is the order of differencing with ∇ being the differencing function such that $\nabla z_t = z_{t+1} - z_t$, $\nabla^2 z_t = \nabla(z_{t+1} - z_t) = z_{t+2} - 2z_{t+1} + z_t$, and so forth.

Henderson-Whittaker Graduation - The well-known classical Whittaker-Henderson graduation method (`preprocess='HW'`), introduced in Henderson (1916), Whittaker (1922), and Henderson (1924), uses appropriately chosen window widths and weights to graduate the time series to approximate a cubic spline. This may also be generalised to a higher or lower order of smoothing with appropriately chosen constraints (such as in Equation (2) and Equation (3) below for a cubic spline). In the interest of being able to generalise, the classical Henderson-Whitaker weighting parameters are derived using the following objective function with the order, d, being such that $D = d + 6$:

$$min_{\omega}\left(\sum_{t=0}^{D-4}(\nabla^3\omega_t)^2\right), \tag{2}$$

subject to the constraints:

$$\begin{gathered}\sum_{t=0}^{D-1}\omega_t = 1, \ \sum_{t=0}^{D-1}\left(t - \frac{D-1}{2}\right)^2\omega_t = 0, \ \sum_{t=0}^{D-1}\left(t - \frac{D-1}{2}\right)\omega_t = 0, \\ \omega_0 = 0, \ \omega_1 = 0, \ \omega_2 = 0, \ \omega_{D-3} = 0, \ \omega_{D-2} = 0, \ \text{and } \omega_{D-1} = 0,\end{gathered} \tag{3}$$

with the final cubic spline inducing weighting parameters then being the appropriately chosen subset $\{\omega_3, \ldots, \omega_{D-4}\}$. The derivation of the asymmetric weights at the edges of the times series have been researched extensively by Musgrave (1964b), Musgrave (1964a), Dagum and Bianconcini (2006), Bianconcini (2006), and Dagum and Bianconcini (2008). This problem is related to the present issue of dealing with the edges discussed in Section 10.3. This problem goes back much further as evidenced by De Forest (1877).

Downsampling - Certain time series may benefit from varying degrees of downsampling (`preprocess='downsample'` or `preprocess='downsample_decimate'`). Downsampling and the associated process of decimation are thoroughly reviewed in Crochiere and Rabiner (1983). This process involves two separate, but related transforms. The first transform uses a normalised sinc filter augmented with an appropriate windowing or tapering function - the most common of which are Hamming and Hann windows - to downsample or truncate the frequency content of the time series. A sinc function is an idealised filter that removes all frequency content above a certain frequency, but since this sinc function is tapered using a Hamming window or a Hann window, it is merely a close approximation of this idealised filter. Once the frequency content has been truncated, the time series should undergo a decimation step where an appropriate subset of the time series can be sampled (`preprocess='downsample_decimate'`) to reflect the lower frequency content of the downsampled time series. In the smoothing context, to keep the lengths of the time series consistent with the original unpreprocessed time series, it is advised to perform only the downsampling (`preprocess='downsample'`).

Having incorporated downsampling as a preprocessing technique into EMD, Compressive Sampling, introduced in Candès et al. (2006), Candès (2006), and Candès and Wakin (2008), should also be integrated into EMD to better isolate IMFs from sparsely sampled time series. This technique relies on the time series being sparse in some domain, such as the frequency domain. A simple sinusoid is exceptionally sparse in the frequency domain, where it is represented as two Dirac functions at both the appropriate negative and positive frequencies. EMD would most certainly benefit from the integration of Compressive Sampling. The sparsity problem becomes more complicated when the IMFs are amplitude and frequency modulated. This technique is called Compressive Sampling Empirical Mode Decomposition (CSEMD) where amplitude and frequency modulated waves are translated into a sparse domain to fit sparsely sampled time series from which IMFs can be extracted using EMD. This technique is experimental and is included with the package in `emd_experimental.py` for completeness sake.

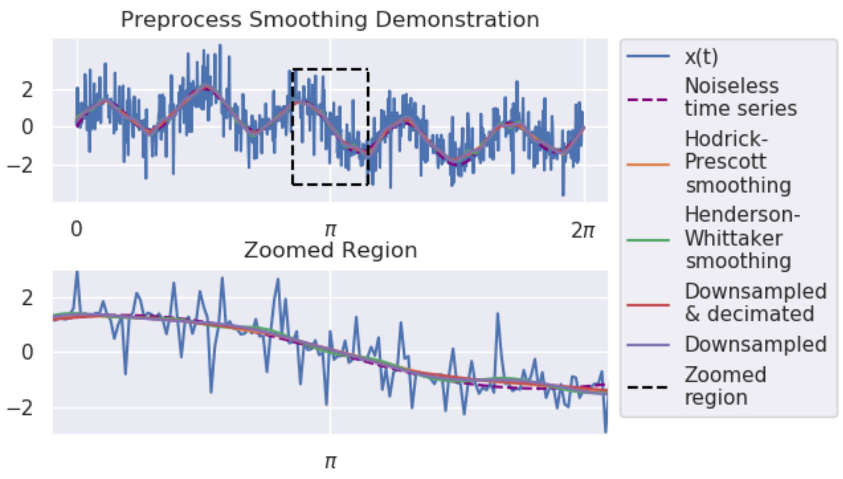

Figure 6: Figure demonstrating family of smoothing methods available to the user with both decimated and undecimated downsampling being displayed.

10.2.3 Recommendations for Preprocessing flag in AdvEMDpy Package

The preprocessing required depends upon the level of noise present in the time series and is very time series and process dependent. There is no preprocessing to suit all time series, but the most robust version of preprocessing would be median filtering the noise out of the time series (`preprocess='median_filter'` with an appropriate window length. The window length is dependent upon the highest frequency present in the time series and the sampling rate. Following the median filter, the user should then perform an initial smoothing (`initial_smoothing=True`) to remove the discontinuities introduced by the median filter.

```
imfs, hts, ifs = \
    emd.empirical_mode_decomposition(knots=knots,
                                     knot_time=knot_time,
                                     initial_smoothing=True,
                                     preprocess='median_filter',
                                     preprocess_window_length=51)[:3]
```

10.3 Edge Effects in AdvEMDpy Package

The most prevalent challenge in EMD is the propagation of errors throughout the IMF and the permeation of the errors throughout the sifting process. Owing to the iterative nature of the algorithm, incorrectly estimated edge leads to errors being ubiquitous throughout all the IMFs. There are several techniques used to estimate the extrema beyond the edges of the signal - this **AdvEMDpy** package attempts to give the reader as many reasonable and researched options as possible. The base implementation of the edge effects code is as follows:

```
imfs, hts, ifs = emd.empirical_mode_decomposition(knots=knots,
                                                   knot_time=knot_time,
                                                   edge_effect='symmetric',
                                                   sym_alpha=0.1,
                                                   nn_m=200,
                                                   nn_k=100,
                                                   nn_method='grad_descent',
                                                   nn_learning_rate=0.01,
                                                   nn_iter=100)[:3].
```

10.3.1 *Symmetric Methods*

Without loss of generality, the procedure will be explained for the right edge maxima of the signal. Figure 7 demonstrates examples of the three possible symmetric edge effects. In no particular order, the technique demonstrated in Rilling et al. (2003) and Wu and Qu (2008) is referred to in this paper as the Symmetric Discard technique owing to the discarding of the end of time series in the new extrema approximation.

Symmetric Discard - In Figure 7 below, an implementation of the Symmetric Discard technique (`edge_effect='symmetric_discard'`) maxima is shown with a purple dot. The end of the time series between the last blue dot and the orange dot is disregarded when approximating the next extreme. Taking the last maxima value, $X(t_M^{max})$, the associated time point, t_M^{max}, the last minima value, $X(t_m^{min})$, and the associated time point, t_m^{min}, the next extreme is calculated as:

$$X(t_{M+1}^{max}) = X(t_M^{max}), \tag{4}$$

with the associated time point being calculated as:

$$t_{M+1}^{max} = t_m^{min} + (t_m^{min} - t_M^{max}). \tag{5}$$

Symmetric Anchor - In Figure 7 below, an implementation of the Symmetric Anchor technique (`edge_effect='symmetric_anchor'`) maxima is shown with an orange dot. In Zhao and Huang (2001) and Zeng and He (2004) the technique creates an extreme at the endpoint - this is why this technique is referred to in this paper as the Symmetric Anchor technique. In this paper, this technique has been generalised to conditionally create an extreme depending on the difference in vertical displacement between the last two extrema and the difference in vertical displacement between the last extrema and the end of the signal - this can be referred to as the Conditional Symmetric Anchor technique. The Conditional Symmetric Anchor is calculated as follows - if $\beta L \geq (1-\alpha)L$ where β is the ratio of the vertical height between the last extrema and the end of the time series to the vertical height between the last two extrema, L is the vertical height between the last two extrema, and α (`sym_alpha=0.1`) is the significance level input, then:

$$X(t_{M+1}^{max}) = X(t_N), \tag{6}$$

with the associated time point being calculated as:

$$t_{M+1}^{max} = t_N. \tag{7}$$

The above is the method followed (without conditions) in Zhao and Huang (2001) and Zeng and He (2004)). If, however, $\beta L < (1-\alpha)L$, then:

$$X(t_{M+1}^{max}) = X(t_M^{max}), \tag{8}$$

with the associated time point being calculated as:

$$t_{M+1}^{max} = t_N + (t_N - t_M^{max}). \tag{9}$$

The Conditional Symmetric Anchor technique collapses to the Symmetric Anchor technique when $\alpha = 1$. The other extreme where $\alpha = -\infty$ leads to the following method.

Symmetric - The Symmetric technique (`edge_effect='symmetric'`) does not anchor the extrema envelope to the ends of the signal under any condition and is equivalent to the Conditional Symmetric Anchor technique where $\alpha = -\infty$. The values are calculated as follows:

$$X(t_{M+1}^{max}) = X(t_M^{max}), \tag{10}$$

with the associated time point being calculated as:

$$t_{M+1}^{max} = t_N + (t_N - t_M^{max}). \tag{11}$$

This point is denoted with a gray dot in Figure 7.

Anti-Symmetric - The Anti-Symmetric (`edge_effect='anti-symmetric'`) approach reflects the uni-variate signal about both axes - the approximated maximum will be the reflected minimum. It is formally calculated as:

$$X(t_{M+1}^{max}) = X(t_N) + (X(t_N) - X(t_m^{min})), \tag{12}$$

with the associated time point being calculated as:

$$t_{M+1}^{max} = t_N + (t_N - t_m^{min}). \tag{13}$$

This point is denoted with a green dot in Figure 7. This technique is a more practical variation of that proposed in Zeng and He (2004), where the points are reflected about the axis rather than about the endpoints.

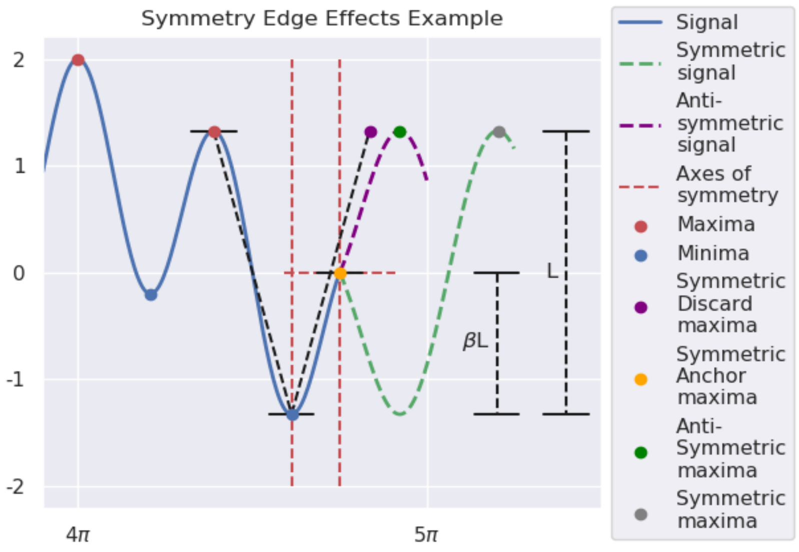

Figure 7: Example time series demonstrating four (five if Conditional Symmetric Anchor is included) different symmetric edge effect techniques with axes of symmetry included.

10.3.2 *Slope-Based Methods*

The slope-based methods offer a different approach to the edge effect problem. These methods use the calculated slopes and difference in temporal location between the extrema nearest to the edges to approximate the subsequent slopes and temporal locations and therefore the location of the subsequent points.

Slope-Based - This technique (`edge_effect='slope_based_method'`), which relies on the four nearest extrema, is introduced in Dätig and Schlurmann (2004). The slopes (s_1 and s_2) can be calculated as follows:

$$s_1 = \frac{X(t_M^{max}) - X(t_{m-1}^{min})}{t_M^{max} - t_{m-1}^{min}}, \tag{14}$$

and

$$s_2 = \frac{X(t_M^{max}) - X(t_m^{min})}{t_M^{max} - t_m^{min}}. \tag{15}$$

With $\Delta t_M^{max} = t_M^{max} - t_{M-1}^{max}$, the slope-based maximum ($X(t_{M+1}^{max})$) is calculated as follows:

$$t_{M+1}^{max} = t_M^{max} + \Delta t_M^{max}, \tag{16}$$

and

$$X(t_{M+1}^{max}) = X(t_m^{min}) + s_1 \times (t_{M+1}^{max} - t_m^{min}). \tag{17}$$

With $\Delta t_m^{min} = t_m^{min} - t_{m-1}^{min}$, the slope-based minimum ($X(t_{m+1}^{min})$) is calculated as follows:

$$t_{m+1}^{min} = t_m^{min} + \Delta t_m^{min}, \tag{18}$$

and

$$X(t_{m+1}^{min}) = X(t_{M+1}^{max}) + s_2 \times (t_{m+1}^{min} - t_{M+1}^{max}). \tag{19}$$

Improved Slope-Based - This (`edge_effect='improved_slope_based_method'`) is introduced in Wu and Qu (2008) as an improvement upon the slope-based method that takes into account the end of the signal and is therefore a conditional edge effect much like the Symmetric Anchor edge effect. With s_1, s_2, Δt_M^{max}, Δt_m^{min}, and $X(t_{M+1}^{max})$ defined as above the Improved Slope-Based maximum is calculated as follows: If $X(t_N) < X(t_{M+1}^{max})$, then proceed as in Slope-Based method, otherwise, set:

$$t_{M+1}^{max} = t_N, \tag{20}$$

and

$$X(t_{M+1}^{max}) = X(t_N). \tag{21}$$

The Improved Slope-Based minimum is then calculated as:

$$t_{m+1}^{min} = t_m^{min} + \Delta t_m^{min}, \tag{22}$$

and

$$X(t_{m+1}^{min}) = X(t_{M+1}^{max}) + s_2 \times (t_{m+1}^{min} - t_{M+1}^{max}). \tag{23}$$

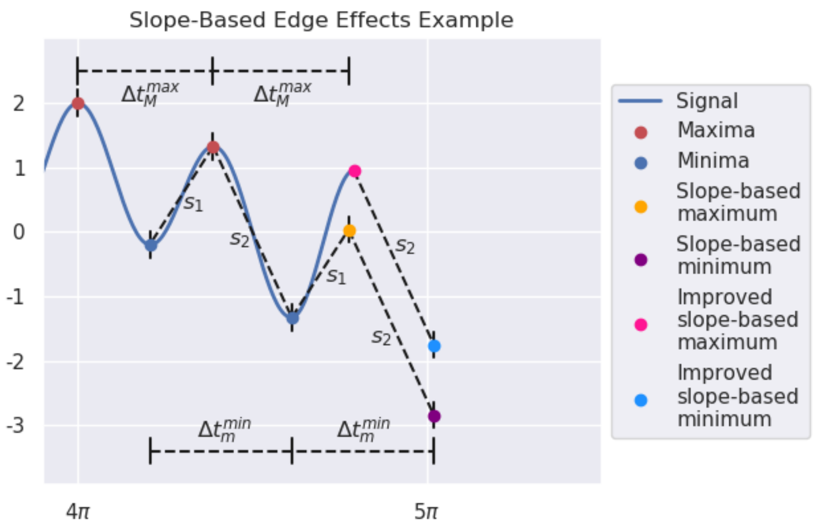

Figure 8: Example time series demonstrating two different slope-based edge effect techniques.

10.3.3 *Characteristic Wave Methods*

This family of methods uses the extrema at the edge of the time series to estimate amplitudes and periods to approximate the next extrema using a sinusoidal wave. The original Huang Characteristic Wave approach is discussed in Huang et al. (1998) and Wu and Qu (2008) - the Modified Huang Characteristic Wave method takes the trend of the edge implicitly into account when approximating the next extrema. The Coughlin Characteristic Wave approach is discussed in Coughlin and Tung (2004) and Wu and Qu (2008) - this method uses only the last two extrema and approximates the next extrema using a simple sinusoidal extension.

Modified Huang Characteristic Wave - The descriptions of the original technique in Huang et al. (1998) and Wu and Qu (2008) are very nebulous and are open to interpretation, but by reviewing Huang et al. (1998), Coughlin and Tung (2004), and Wu and Qu (2008) a reasonable method (`edge_effect='characteristic_wave_mod_Huang'`) can be inferred and modified with some confidence. By calculating $p_2 = 2 \times (t_{M-1}^{max} - t_{m-1}^{min})$, $p_1 = 2 \times (t_M^{max} - t_m^{min})$, $a_2 = \frac{(X(t_{M-1}^{max}) - X(t_{m-1}^{min}))}{2}$, and $a_1 = \frac{(X(t_M^{max}) - X(t_m^{min}))}{2}$, as seen in Figure 9, the next maximum may be estimated as follows:

$$t_{M+1}^{max} = t_m^{min} + \frac{p_1}{p_2} \times (t_M^{max} - t_{m-1}^{min}), \tag{24}$$

and

$$X(t_{M+1}^{max}) = X(t_m^{min}) + \frac{a_1}{a_2} \times (X(t_M^{max}) - X(t_{m-1}^{min})), \tag{25}$$

with the next minimum being estimated as:

$$t_{m+1}^{min} = t_{M+1}^{max} + \frac{p_1}{p_2} \times (t_m^{min} - t_M^{max}), \tag{26}$$

and

$$X(t_{m+1}^{min}) = X(t_{M+1}^{max}) + \frac{a_1}{a_2} \times (X(t_m^{min}) - X(t_M^{max})). \tag{27}$$

Coughlin Characteristic Wave - The Coughlin Characteristic Wave is a less intensive technique (`edge_effect='characteristic_wave_Coughlin'`) and only uses t_M^{max}, t_m^{min}, $X(t_M^{max})$, and $X(t_m^{min})$ - the last two extrema. This method disregards any trend in the structure of the edges, but is simpler to calculate. With a_1 and p_1 as before, the next maximum is calculated as follows:

$$t_{M+1}^{max} = t_m^{min} + \frac{p_1}{2}, \tag{28}$$

and

$$X(t_{M+1}^{max}) = X(t_M^{max}), \tag{29}$$

with the next minimum being estimated as:

$$t_{m+1}^{min} = t_{M+1}^{max} + \frac{p_1}{2}, \tag{30}$$

and

$$X(t_{m+1}^{min}) = X(t_m^{min}). \tag{31}$$

This is equivalent to a simple sinusoidal wave with amplitude a_1 and period p_1. The maximum derived in this manner is equivalent to the maximum derived using the Symmetric Discard technique, but this is where the equivalency ends as the minimums are significantly different.

Average Characteristic Wave - The Average Characteristic Wave plot is not included in Figure 9 as there is no amplitude defined for either the maxima characteristic wave or the minima characteristic wave, but the calculation of the maximum and minimum rely on the characteristic periods and extrema (and hence indirectly on the amplitude) and should be included in this class of method. Unlike the other characteristic wave methods (`edge_effect='average'`), the Average Characteristic Wave method implicitly uses two separate characteristic waves for the maximum and minimum calculation respectively - this can be seen in the calculations below as the maximum calculation does not refer to the minima and vice versa. This method is used in Chiew et al. (2005). The average maximum is calculated as follows:

$$t_{M+1}^{max} = t_M^{max} + (t_M^{max} - t_{M-1}^{max}), \tag{32}$$

and

$$X(t_{M+1}^{max}) = \frac{(X(t_M^{max}) + X(t_{M-1}^{max}))}{2}, \tag{33}$$

with the average minimum being calculated as follows:

$$t_{m+1}^{min} = t_m^{min} + (t_m^{min} - t_{m-1}^{min}), \tag{34}$$

and

$$X(t_{m+1}^{min}) = \frac{(X(t_m^{min}) + X(t_{m-1}^{min}))}{2}. \tag{35}$$

10.3.4 *Explicit Methods*

All of the above edge effect techniques may be further sub-classified under implicit edge effect methods as the time series itself is not explicitly extrapolated to find the next extrema from the extrapolated time series. Instead, only the nearest two or four extrema are used to approximate where the next two will occur. Another approach is to explicitly extrapolate the time series and extract the resulting extrema. Only one method in this family will be explored, but there are many more.

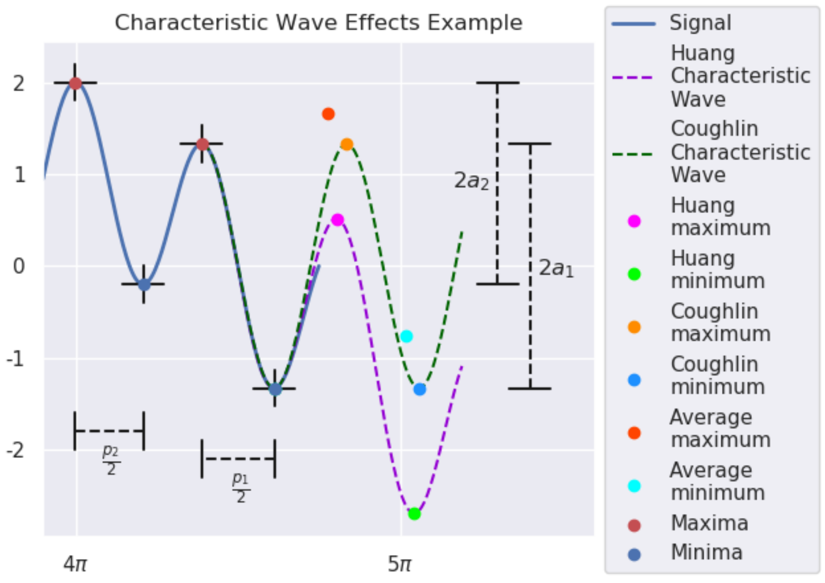

Figure 9: Example time series demonstrating three different characteristic wave edge effect techniques with average characteristic waves excluded.

Single Neuron Neural Network - The method (`edge_effect='neural_network'`) demonstrated here follows the work done in Deng et al. (2001). The objective function is:

$$min_{\bar{w}}(\bar{w}P - \bar{y}), \tag{36}$$

with

$$\bar{w} = \begin{bmatrix} w_1 \; w_2 \; \ldots \; w_k \end{bmatrix}, \tag{37}$$

$$P = \begin{bmatrix} y_{-(m+k)} & y_{-(m+k-1)} & \cdots & y_{-(k+1)} \\ y_{-(m+k-1)} & y_{-(m+k-2)} & \cdots & y_{-(k)} \\ \vdots & \vdots & \ddots & \vdots \\ y_{-(m+1)} & y_{-m} & \cdots & y_{-2} \end{bmatrix}, \tag{38}$$

and

$$\bar{y} = \begin{bmatrix} y_{-m} \; y_{-(m-1)} \; \cdots \; y_{-1} \end{bmatrix}. \tag{39}$$

In the absence of an activation function or the use of the identity function as an activation function a single neuron neural network simply reduces to linear regression as can be seen in Equation (36). The naming convention is kept for consistency with the method proposed in Deng et al. (2001). In this problem k (`nn_k=100`) is the number of previous points (number of parameters) needed to estimate the next point in the time series and m (`nn_m=200`) is the number of target outputs used to estimate the parameters. Two different techniques may be used to estimate $\bar{w}$ - either gradient descent (`nn_method='grad_descent'`) or steepest descent (`nn_method='steep_descent'`). The optimisation problem is arranged in the same manner for both techniques. The weights are initialised ($\bar{w}_0$), a learning rate is stipulated (l) (`nn_learning_rate=0.01`), and a fixed number of iterations is set (`nn_iter=100`). The algorithm proceeds as follows:

```
w = w_0

for iter in iterations:

    y_0 = np.matmul(w, P)
    e = (y - y_0)
    grad = - e * P
    av_grad = np.mean(grad, axis=1)
    if method == ‘grad_descent’:
        adj = - l * av_grad
    elif method == ‘steep_descent’:
        max_grad = av_grad * (np.abs(av_grad) == max(np.abs(av_grad)))
        adj = - l * max_grad
    w += adj.
```

The gradient descent optimisation method requires a lower learning rate than the steepest descent optimisation method as all the weights are optimised simultaneously in gradient descent in each iteration, whereas only a single weight is optimised in each iteration during steepest descent. Depending on the complexity of the time series, the weighting vector obtained using steepest descent may be very sparse and this could be desirable in certain situations.

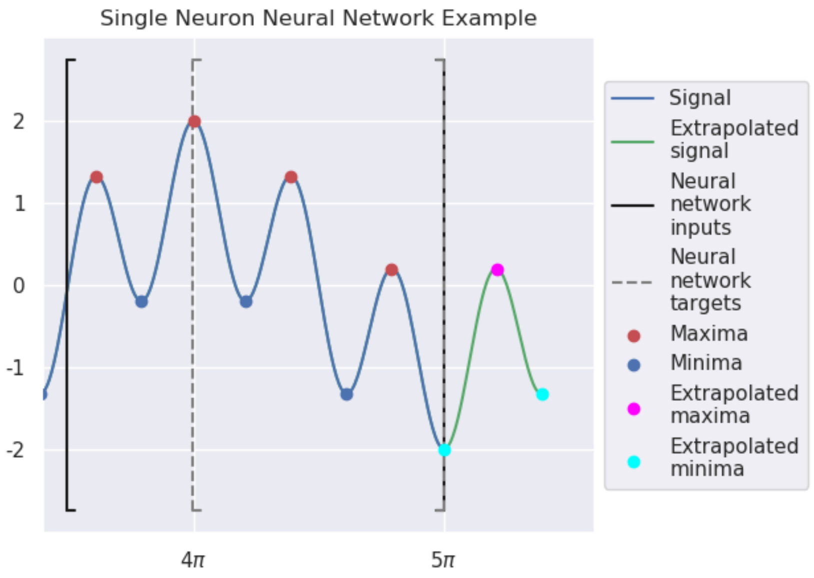

Figure 10: Example time series demonstrating single neuron neural network edge effect with $m = 200$ and $k = 100$ with forecasted time series in green.

10.3.5 Recommendations for Edge Effects in AdvEMDpy Package

The edge effect is the most diverse method in this paper with many options available. No method suits every time series. The most robust methods would be the Conditional Symmetric Anchor method or the Coughlin Characteristic Wave. Both of these methods only use the last two extrema to forecast the next extrema, but the Conditional Symmetric Discard method explicitly takes into account the end of the time series. The Conditional Symmetric Anchor method is therefore recommended. The Single Neuron Neural Network has not been as extensively studied and robustified as the other method and it is advised to be used with caution.

```
imfs, hts, ifs = \
    emd.empirical_mode_decomposition(knots=knots,
                                     knot_time=knot_time,
                                     edge_effect='symmetric_anchor',
                                     sym_alpha=0.1)[:3]
```

10.4 Detrended Fluctuation Analysis in AdvEMDpy Package

At EMD's core, the algorithm iterates by progressively removing the local mean from the IMF candidate. Owing to the numerous ways of defining and estimating a local mean, local mean estimation should be referred to as detrended fluctuation analysis. Detrended fluctuation analysis was originally introduced in Peng et al. (1994) and intended for a different purpose where data was partitioned into equal sets and detrended using discontinuous linear approximations of the local trends. The average of the variances of the detrended data in these partitioned sets was calculated before the subsets were increased in size and the process was repeated. This was done to determine the presence of long-memory processes or rather long-term correlation structures. For our purposes, each set of data (data points between two consecutive knots) is detrended using continuous cubic B-splines to approximate the local trend or local mean. The trends are extracted using local windows defined using the knot sequences. The time series can therefore be estimated as a series of bases and thought of as a sequence of locally defined segments.

In Figure 11 detrended fluctuation analysis is demonstrated using the following time series:

$$f(t) = \cos(2t) + \cos(4t) + \cos(8t) + \epsilon(t), \tag{40}$$

where $\epsilon(t) \in \mathcal{N}(0,1)$ for $t \in \{t_0, \ldots, t_{1000}\}$. One can observe in Figure 11 that as the distance between the uniformly placed knots is increased, the fitted splines cannot detect the higher-frequency structures.

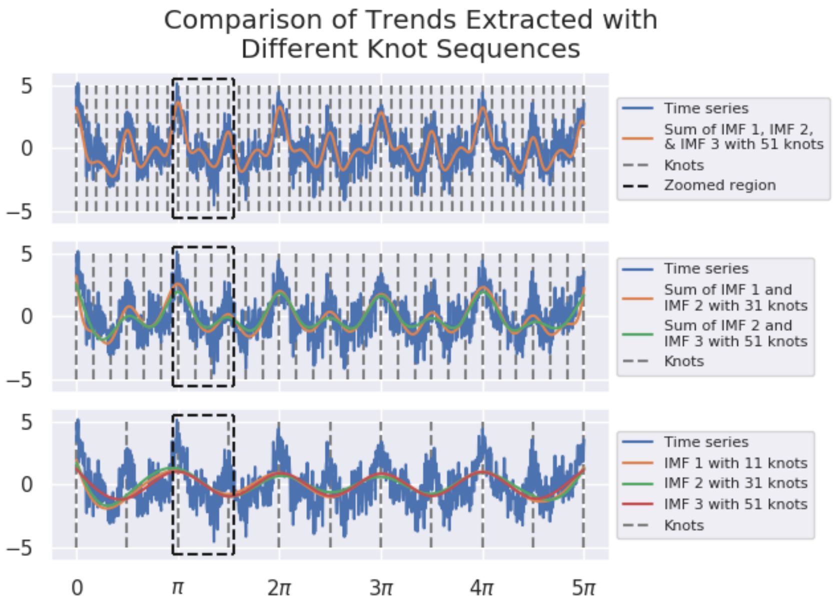

Figure 11: Example time series demonstrating detrended fluctuation analysis of time series with different knot sequences resulting in different trend estimation.

In Figure 11 one can observe the relationship between detrended fluctuation analysis and the iterative IMF extraction method known as EMD. In the top image of Figure 11, one can observe the trend as the sum of all the IMFs extracted with a sufficient knot sequence. In the middle image, the trend is approximated with a knot sequence that is insufficient to capture the highest frequency structure. One can also see how closely this trend resembles the sum of IMF 2 and IMF 3 for EMD when the knot sequence is sufficient. Finally, in the bottom image, one can see how the trend is estimated with a wholly insufficient knot sequence compared with the lowest order IMF of each previous sifting procedure.

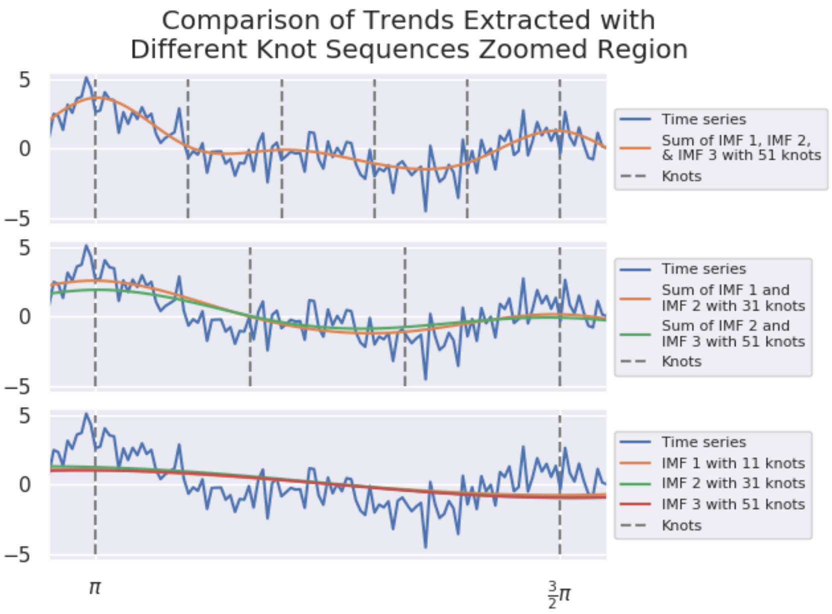

Figure 12: Example time series demonstrating detrended fluctuation analysis of time series with different knot sequences resulting in different trend estimation.

In Figure 12 the different frequency structures are more easily visible. It can be shown that:

$$\lim_{n\rightarrow\infty}\sqrt{\frac{1}{n-1}\sum_{k=1}^{n}\cos^2\left(\frac{2\pi fk}{n}\right)}=\frac{1}{\sqrt{2}}\approx 0.707, \tag{41}$$

with $f\in\mathbb{N}$ and $f<<n$. With this in mind, and noting the structure of Equation (40), one can understand the following results:

$$\begin{aligned} SD\left(f(t)-\sum_{i=1}^{3}IMF_i^{51}(t)\right) &= 1.005\approx 1\\ SD\left(f(t)-\sum_{i=1}^{2}IMF_i^{31}(t)\right) &= 1.558\\ SD\left(f(t)-IMF_1^{11}(t)\right) &= 2.072, \end{aligned} \tag{42}$$

with IMF_i^j being the i^{th} IMF with knot sequence j. The first equality in Equation (42) most closely calculates the true underlying noise present in the system as a result of the random fluctuations caused by the Standard Normally distributed Gaussian Noise. In the other equalities, the standard deviation calculations are confounded by undetected underlying high-frequency structures with their individual "standard deviations" being approximated by Equation (41). The parallels between detrended fluctuation analysis and EMD should be noted by the user. By noting Figure 11, Figure 12, and Figure 13 one can observe that as a result of the defined knot sequence that fluctuation is measured relative to some frequency band. Each IMF exists in some frequency range and the fluctuations calculated in Equation (42) measure the local fluctuation relative to some implicit frequency boundary as a result of the knot sequence defined.

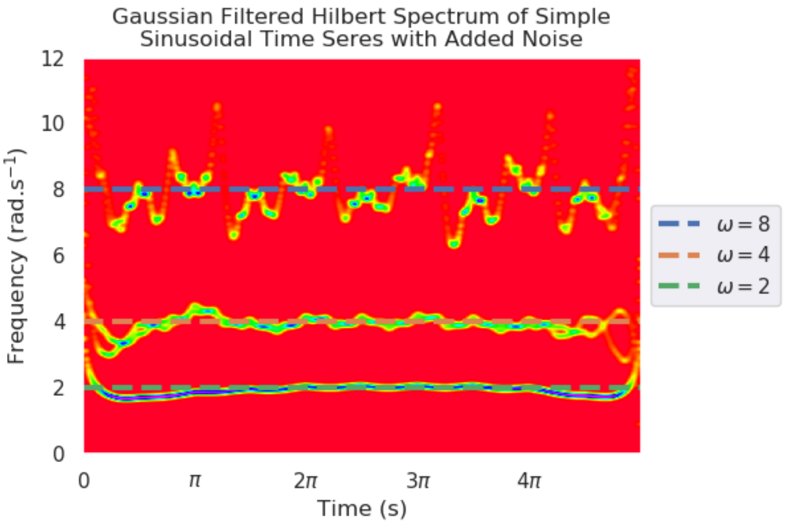

Figure 13: Hilbert spectrum of example time series demonstrating the frequencies of the three IMFs present when sufficient knots are used.

In Figure 13 the frequencies of the three constituent structures (apart from the added standard normal Gaussian noise) are visible. The highest frequency structure is perturbed by the noise as expected. EMD can be viewed as a generalisation of detrended fluctuation analysis whereby the trend is decomposed into separate frequency structures in descending order of IF using a defined knot sequence (in general).

The local mean can be estimated using the framework in Section 3.3 without any explicit or implicit smoothing. Explicit smoothing that deals simultaneously with smoothing and the edge effects is referred to in the literature as Statistical EMD (SEMD). The base implementation of the detrended fluctuation analysis is as follows:

```
imfs, hts, ifs = emd.empirical_mode_decomposition(knots=knots,
                                                  knot_time=knot_time,
                                                  smooth=True,
                                                  smoothing_penalty=0.1,
                                                  dft='envelopes',
                                                  order=15,
                                                  increment=10)[:3].
```

10.4.1 *Statistical EMD (P-Splines)*

This method (`smooth=True`) is introduced in Kim et al. (2012). SEMD, in the cubic B-spline envelope setting, can be implemented by introducing a smoothing parameter into the objective function, Equation (14) of 'Package AdvEMDpy: Algorithmic Variations of Empirical Mode Decomposition in Python'. The specific introduction of a penalty into B-splines is discussed in Eilers and Marx (1996) and is referred to as P-splines. The matricification of the P-spline objective function can be seen below. Second-order smoothing is done on the coefficients, but this can be generalised to higher-order smoothing. The second-order smoothing of the coefficients is incorporated using the $\boldsymbol{D}$ matrix seen below:

$$\boldsymbol{D} = \begin{bmatrix} 1 & -2 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & -2 & 1 & 0 & \cdots & 0 \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ 0 & \cdots & 0 & 1 & -2 & 1 & 0 \\ 0 & \cdots & 0 & 0 & 1 & -2 & 1 \end{bmatrix}, \tag{43}$$

and with $\boldsymbol{P}$ defined as below,

$$\boldsymbol{P} = \boldsymbol{D}^T\boldsymbol{D}, \tag{44}$$

Equation Equation (14) of 'Package AdvEMDpy: Algorithmic Variations of Empirical Mode Decomposition in Python', with $\boldsymbol{s}$, $\boldsymbol{B}$, and $\boldsymbol{c}$ defined as before and with discrete penalty term λ, becomes:

$$DPMSE(\boldsymbol{c}|\boldsymbol{s}) = (\boldsymbol{s} - \boldsymbol{B}\boldsymbol{c})^T(\boldsymbol{s} - \boldsymbol{B}\boldsymbol{c}) + \lambda\boldsymbol{c}^T\boldsymbol{P}\boldsymbol{c}. \tag{45}$$

The magnitude of the penalty term, λ (`smoothing_penalty=0.1`), determines the amount of smoothing. SEMD is implemented separately to the other detrended fluctuation techniques as smoothing can be done irrespective of the detrended fluctuation technique used.

It is highly recommended that `smooth=True` and the smoothing penalty is non-zero when `dft='envelopes'` as the extrema are not guaranteed to satisfy the Schoenberg–Whitney Conditions (SWC) for an arbitrary knot sequence. This will allow envelopes to be fitted even when there are no extrema between successive knot points, otherwise nonsensical envelopes may result. For an arbitrary set of extrema $\mathbf{y} = \{y_1, y_2, \ldots, y_j\}$, a cubic B-spline knot sequence $\boldsymbol{\tau} = \{\tau_1, \tau_2, \ldots, \tau_{j+4}\}$, the SWC can be stated as:

$$\tau_i \leq y_i, \leq \tau_{i+1} \; \forall \; i \in \{1, 2, \ldots, j-1\}. \tag{46}$$

In Figure 14 the following time series is plotted to demonstrate the necessity for either the SWC to be satisfied or for the envelopes to be smoothed:

$$g(t) = \cos(t) + \cos(5t), \tag{47}$$

with $t \in [0, 5\pi]$. With $\lambda = 0$ in Equation (45) and with the SWC not being satisfied by either the maxima or the minima, the envelopes are stretched towards zero. This is, unfortunately, unavoidable without either the SWC being satisfied or some form of smoothing.

10.4.2 *Enhanced EMD*

In Kopsinis and McLaughlin (2007a) Enhanced EMD (`dft='enhanced'`) is introduced. In Kopsinis and McLaughlin (2007b) and Kopsinis and McLaughlin (2008) a genetic search algorithm is used to optimise the interpolation point allocation to optimise the local mean estimation when compared against the known local mean but in Kopsinis and McLaughlin (2007a), the EMD is performed on the derivative of the time series to isolate the highest frequency component. Using the highest frequency component of the derivative, the optimal interpolation point allocation can be estimated for the estimation of the local mean.

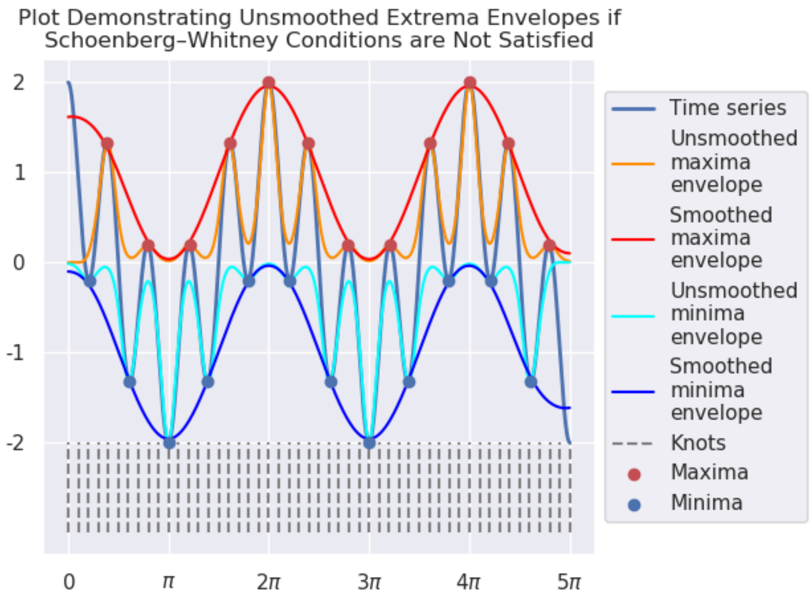

Figure 14: Example time series demonstrating unsmoothed extrema envelopes being fitted when SWC are not satisfied resulting in nonsensical envelopes.

The only difference to the above is the preprocessing (in a sense) of the time series to estimate the optimal extrema points. The derivative of the time series can be calculated using the first forward difference:

$$\boldsymbol{s'} = \begin{bmatrix} s'(t_0) = \frac{s(t_1)-s(t_0)}{t_1-t_0} \\ s'(t_1) = \frac{s(t_2)-s(t_1)}{t_2-t_1} \\ \vdots \\ s'(t_{N-1}) = \frac{s(t_N)-s(t_{N-1})}{t_N-t_{N-1}} \end{bmatrix}. \tag{48}$$

Given the B-spline framework developed in Section 3.3 and using a result stated in Chen et al. (2006) and proved in de Boor (1978) the derivative of a B-spline basis (and therefore a spline fitted using B-splines) can be calculated as:

$$B'_{j,k,\tau}(t) = \frac{k-1}{\tau_{j+k-1}-\tau_j}B_{j,k-1,\tau}(t) - \frac{k-1}{\tau_{j+k}-\tau_{j+1}}B_{j+1,k-1,\tau}(t). \tag{49}$$

Therefore, with

$$\boldsymbol{B'} = \begin{bmatrix} B'_{0,4}(t_0) & \cdots & B'_{(M-4),4}(t_0) \\ B'_{0,4}(t_1) & \cdots & B'_{(M-4),4}(t_1) \\ \vdots & \ddots & \vdots \\ B'_{0,4}(t_N) & \cdots & B'_{(M-4),4}(t_N) \end{bmatrix}, \tag{50}$$

and $\boldsymbol{c}$ optimised in either Equation (14) of 'Package AdvEMDpy: Algorithmic Variations of Empirical Mode Decomposition in Python' or Equation (45), the derivative of of a B-spline curve can be calculated as:

$$\boldsymbol{s'} = \boldsymbol{B'}\boldsymbol{c}. \tag{51}$$

Once the derivative of the spline curve has been obtained (by whichever method), the EMD method is then performed on the derivative until the first IMF (highest frequency structure) is isolated. The derivative is therefore dichotomised such that:

$$s'(t) = s'_h(t) + s'_l(t). \tag{52}$$

Using the same technique as above such that:

$$\begin{aligned} B''_{j,k,\tau}(t) &= \frac{k-1}{\tau_{j+k-1}-\tau_j} B'_{j,k-1,\tau}(t) - \frac{k-1}{\tau_{j+k}-\tau_{j+1}} B'_{j+1,k-1,\tau}(t) \\ &= \frac{(k-1)(k-2)}{(\tau_{j+k-1}-\tau_j)(\tau_{j+k-2}-\tau_j)} B_{j,k-2,\tau}(t) \\ &- \frac{(k-1)(k-2)}{(\tau_{j+k-1}-\tau_j)(\tau_{j+k-1}-\tau_{j+1})} B_{j+1,k-2,\tau}(t) \\ &- \frac{(k-1)(k-2)}{(\tau_{j+k}-\tau_{j+1})(\tau_{j+k-1}-\tau_{j+1})} B_{j+1,k-2,\tau}(t) \\ &+ \frac{(k-1)(k-2)}{(\tau_{j+k}-\tau_{j+1})(\tau_{j+k}-\tau_{j+2})} B_{j+2,k-2,\tau}(t) \end{aligned} , \tag{53}$$

the derivative of $s'_h(t)$, $s''_h(t)$, can be calculated as:

$$\boldsymbol{s''_h} = \boldsymbol{B''c_h}, \tag{54}$$

with

$$\boldsymbol{B''} = \begin{bmatrix} B''_{0,4}(t_0) & \cdots & B''_{(M-4),4}(t_0) \\ B''_{0,4}(t_1) & \cdots & B''_{(M-4),4}(t_1) \\ \vdots & \ddots & \vdots \\ B''_{0,4}(t_N) & \cdots & B''_{(M-4),4}(t_N) \end{bmatrix}, \tag{55}$$

and $\boldsymbol{c_h}$ being the coefficients corresponding to $s'_h(t)$. The optimised maxima for the extraction of the first IMF can then be calculated as the points in $s(t)$ such that:

$$s'_h(t) = 0 \text{ and } s''_h(t) < 0, \tag{56}$$

with the optimised minima being calculated as the points in $s(t)$ such that:

$$s'_h(t) = 0 \text{ and } s''_h(t) > 0. \tag{57}$$

The EMD algorithm proceeds as before with the extrema not changing until an IMF is extracted. The new optimised extrema are then calculated and the algorithm continues.

10.4.3 *Inflection Point Interpolation*

The detrended fluctuation analysis of a time series to estimate the local mean by interpolating through the interpolation points (`dft='inflection_points'`) was first proposed in Kopsinis and McLaughlin (2007b). The inflection points are calculated using:

$$\boldsymbol{s''} = \boldsymbol{B''c}, \tag{58}$$

with the inflection points being such that:

$$s''(t) = 0. \tag{59}$$

10.4.4 Binomial Average Interpolation

All the above methods are well suited to smoothed data. If the above methods were to be directly applied to noisy data there would be a proliferation of extrema and any potentially meaningful information in the first IMF would be obscured by the noise. This is a double-edged sword as over-smoothing would also result in the loss of meaningful high-frequency information.

In Chen et al. (2006), estimating the local mean structure is done by taking a binomial average (`dft='binomial_average'`) of the surrounding points. This may be classified as a form of preprocessing as smoothing is then required to create a local mean to be extracted, otherwise high-frequency content may be present in lower-order IMFs.

The binomial average is calculated as:

$$\mu(\tau_j) = \frac{1}{2^{k-1}} \sum_{h=j-\frac{(k-1)}{2}}^{j+\frac{(k-1)}{2}} \binom{k-1}{h-\left(j-\frac{(k-1)}{2}\right)} s(\tau_h), \tag{60}$$

with $\mu(\tau_j)$ being the binomial average of the time series at time point τ_j, k (`order=15`) being the order of the binomial averaging such that $k = 2n+1$ with $n \in \mathbb{N}$, and $s(\tau_h)$ being the value of time series at time point τ_j with j being every 10^{th} (`increment=10`) point.

All of the discussed detrended fluctuation techniques can be seen in Figure 15 - when the time series is well-behaved or smooth all the methods converge, but nuances still exist.

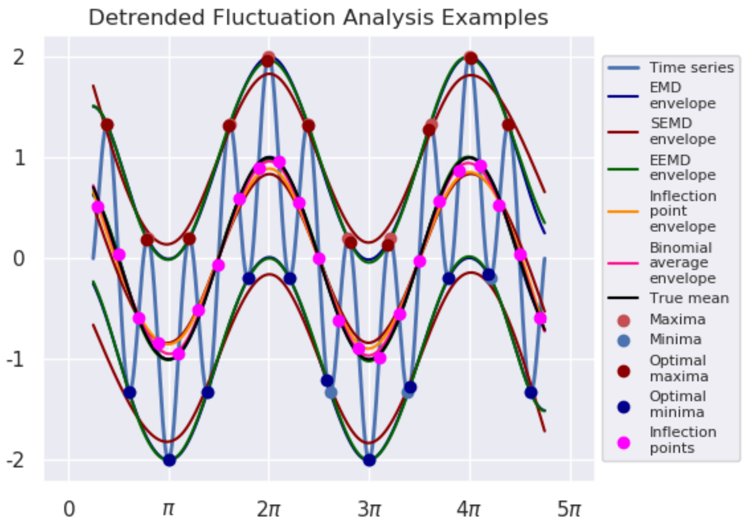

Figure 15: Example time series demonstrating five different local mean estimation techniques through detrended fluctuation analysis.

10.4.5 Recommendations for Detrended Fluctuation Analysis in AdvEMDpy Package

The most studied and utilised local mean estimation technique is the standard envelope technique (`dft='envelopes'`) originally put forward in Huang et al. (1998), Huang et al. (1999), and Huang (1999) and, as already mentioned, should be performed with smoothing such that `smooth=True` and `smoothing_penalty=0.1`. This technique is recommended and is intended to be applied after an initial preprocessing or smoothing, otherwise, the first few IMFs may be nonsensical as the noise in the time series will result in the proliferation of extrema and confound our IMFs.

```
imfs, hts, ifs = \
    emd.empirical_mode_decomposition(knots=knots, knot_time=knot_time,
                                     initial_smoothing=True, smooth=True,
                                     smoothing_penalty=0.1,
                                     dft=‘envelopes’)[:3]
```

10.5 Stopping Criteria in AdvEMDpy Package

To prevent over-sifting resulting in physically meaningless IMFs and little discernible information about the process under observation, several stopping criteria are provided to prevent over-sifting. The validity of the various stopping criteria warrants further study as some are more related to algorithmic steps than others. The base implementation of the stopping criteria follows:

```
imfs, hts, ifs = \
    emd.empirical_mode_decomposition(knots=knots, knot_time=knot_time,
                                     stop_crit=‘S_stoppage’,
                                     stop_crit_threshold=10,
                                     mft_theta_1=0.05, mft_theta_2=0.5,
                                     mft_alpha=0.05, mean_threshold=10,
                                     max_internal_iter=30, max_imfs=10)[:3].
```

Condition 1 and Condition 2 are restated here for ease of reference:

Condition 1 $\text{abs}\left(\left|\left\{\frac{d\gamma_k(t)}{dt}=0:t\in(0,T)\right\}\right|-\left|\left\{\gamma_k(t)=0:t\in(0,T)\right\}\right|\right)\leq 1,$

Condition 2 $\tilde{\gamma}_k^{\mu}(t)=\left(\frac{\tilde{\gamma}_k^M(t)+\tilde{\gamma}_k^m(t)}{2}\right)=0\;\forall\,t\in[0,T]$ with,

$\gamma_k(t)=\tilde{\gamma}_k^M(t)$ if $\frac{d\gamma_k(t)}{dt}=0$ and $\frac{d^2\gamma_k(t)}{dt^2}<0$,
$\gamma_k(t)\leq\tilde{\gamma}_k^M(t)\;\forall\,t\in[0,T]$,
$\gamma_k(t)=\tilde{\gamma}_k^m(t)$ if $\frac{d\gamma_k(t)}{dt}=0$ and $\frac{d^2\gamma_k(t)}{dt^2}>0$, and
$\gamma_k(t)\geq\tilde{\gamma}_k^m(t)\;\forall\,t\in[0,T]$.

10.5.1 Modified Mean Threshold

In order to be classified as an IMF, Condition 1 and Condition 2 need to both be satisfied. In practice, Condition 1 is easily achievable and takes relatively few iterations, but Condition 2 is computationally very expensive and the large number of iterations required for the local mean of an IMF candidate to approach zero would, unfortunately, remove a large portion of meaningful content from the time series. This necessitates the statement of a modified version of Condition 2 as stated in Section 6 and repeated here for ease of reference.

Modified Condition 2 $\sum_t\left|\tilde{\gamma}_k^{\mu}(t)\right|\leq\epsilon,$
for some chosen threshold value ϵ (`mean_threshold=10`).

10.5.2 Fixed Iteration

A simple stopping criterion puts a hard limit on the number of iterations allowed before the internal iteration loop stops and the IMF is extracted (`max_internal_iter=10`). The fixed iteration stopping criterion and the modified mean threshold criterion are implemented in addition to other stopping criteria with them functioning as a hard limit on the number of siftings and a modification of the classical IMF condition, respectively.

10.5.3 S Stoppage

Much like the fixed iteration count stopping criterion, the S Stoppage, as in Huang and Wu (2008) (`stop_crit=‘S_stoppage’`), is not mathematically rigorous but relies on a count variable. Unlike the Fixed Iteration count stopping criterion, the S Stoppage criterion has a conditional count variable. The stopping criterion is reached when the difference between the number of zero crossings and the number of extrema remains constant over S iterations with the number of iterations as another input (`stop_crit_threshold=10`).

10.5.4 Cauchy-Type Convergence

This stopping criterion (`stop_crit='sd'`) was first proposed in Huang et al. (1998). Its name results from its similarity to the Cauchy convergence of a series in functional analysis. The standard deviation of IMF candidate j at iteration k is calculated as:

$$SD_{(j,k)} = \sum_{t=t_0}^{T_N} \left[\frac{\left|(h_{(j,k-1)}(t) - h_{(j,k)}(t))\right|^2}{h^2_{(j,k-1)}(t)} \right] < \epsilon, \tag{61}$$

for some threshold value ϵ (`stop_crit_threshold=10`), with $h_{(j,k)}(t)$ being IMF candidate j on iteration k. In Huang et al. (1998) a threshold value of $0.2 - 0.3$ is suggested with other works citing a value of 0.1. These values are subjective and are very dependent on the length of the time series under observation - a better description would be threshold per time point.

10.5.5 Cauchy-Type Convergence 11a

The first variation (`stop_crit='sd_11a'`) of the above stopping criterion proposed in Huang and Wu (2008) as Equation (11a) is calculated as:

$$SD^{11a}_{(j,k)} = \frac{\sum_{t=t_0}^{T_N} \left|(h_{(j,k-1)}(t) - h_{(j,k)}(t))\right|^2}{\sum_{t=t_0}^{T_N} h^2_{(j,k-1)}(t)} < \epsilon. \tag{62}$$

This stopping criterion measures the relative energy difference between IMFs, rather than the Cauchy-type convergence.

10.5.6 Cauchy-Type Convergence 11b

The second variation (`stop_crit='sd_11b'`) of the above Cauchy-type convergence stopp-ping criterion is also proposed in Huang and Wu (2008) as Equation (11b) and is calculated as:

$$SD^{11b}_{(j,k)} = \frac{\sum_{t=t_0}^{T_N} \mu^2_{(j,k)}(t)}{\sum_{t=t_0}^{T_N} h^2_{(j,k)}(t)} < \epsilon. \tag{63}$$

Equation (63) can be seen as a combination of the Modified Mean Threshold and the Cauchy-Type Convergence. It has standardised the allowed mean discrepancy.

10.5.7 Mean Fluctuation

This stopping criterion (`stop_crit='mft'`) is proposed in Rilling et al. (2003) and modified in Tabrizi et al. (2014). It takes into account locally large fluctuations that would otherwise cause over-sifting throughout the majority of the signal. With $\mu(t)$ being the calculated local mean and $a(t)$ being the mode amplitude calculated as:

$$a(t) = \frac{|M(t) - m(t)|}{2}, \tag{64}$$

with $M(t)$ being the maximum envelope and $m(t)$ being the minimum envelope the evaluation function can be calculated. The evaluation function is calculated as follows:

$$\sigma(t) = \left|\frac{\mu(t)}{a(t)}\right|. \tag{65}$$

Unlike the other stopping criteria, this stopping criterion requires three parameters, namely θ_1 (`mft_theta_1=0.05`), θ_2 (`mft_theta_2=0.5`), and α (`mft_alpha=0.05`). The stopping criterion is met (and the internal loop is terminated) when $\sigma(t) < \theta_1$ for fraction $(1-\alpha)$ of the entire range of the time series, and $\sigma(t) < \theta_2$ for the remainder of the time series. In Rilling et al. (2003) typical values for the parameters are proposed as $\theta_1 = 0.05$, $\theta_2 = 10\theta_1$, and $\alpha = 0.05$.

10.5.8 Energy Difference Tracking

This method (`stop_crit='edt'`) was proposed in Junsheng et al. (2006). It takes advantage of the orthogonality of the IMFs and as such the cumulative energy of the IMFs should be equal to the energy of the original time series. The energy of an individual IMF candidate can be calculated as:

$$E_{h_{(j,k)}} = E\left[h_{(j,k)}(t)\right] = \sum_{t=t_0}^{T_{N-1}} \left|h_{(j,k)}(t)\right|^2 \Delta t. \tag{66}$$

Should this candidate be selected as an IMF, the energy of the potential remainder of the time series after the removal of the IMF is calculated as:

$$E_{r_j} = E\left[r_j(t)\right] = \sum_{t=t_0}^{T_{N-1}} \left|r_j(t)\right|^2 \Delta t. \tag{67}$$

The energy of the residual time series before the removal of the IMF is calculated as:

$$E_{r_{j-1}} = E\left[r_{j-1}(t)\right] = \sum_{t=t_0}^{T_{N-1}} \left|r_{j-1}(t)\right|^2 \Delta t. \tag{68}$$

The internal iteration stops when the energy difference falls below a certain threshold value ϵ:

$$E_{diff} = \left|E_{r_{j-1}} - \left(E_{h_{(j,k)}} + E_{r_j}\right)\right| < \epsilon. \tag{69}$$

10.5.9 Recommendations for Stopping Criteria in AdvEMDpy Package

There are several recommendations for this particular aspect of the algorithm. The most widely used stopping criterion is the Cauchy-Type Convergence, but this is often unstable and does not converge steadily such as an exponential decaying variable. One should use S Stoppage (`stop_crit='S_stoppage'`) with a threshold of 10 (`stop_crit_threshold=10`). In addition to this criterion, one should impose a value on the Modified Mean Threshold (`mean_threshold=10`), the maximum number of internal iterations (`max_internal_iter=30`), and the maximum allowed number of IMFs (`max_imfs=10`). With all these conditions the stopping criteria aspect of the algorithm should be sufficiently managed.

```
imfs, hts, ifs = emd.empirical_mode_decomposition(knots=knots,
                                                   knot_time=knot_time,
                                                   stop_crit='S_stoppage',
                                                   stop_crit_threshold=10,
                                                   mean_threshold=10,
                                                   max_internal_iter=30,
                                                   max_imfs=10)[:3]
```

10.6 Spline Methods in AdvEMDpy Package

All the above techniques are listed with the cubic B-spline implementation of EMD in mind. Other spline techniques are effective when using EMD. The other splines used have a different basis, but because they are both cubic bases, they may be easily mapped onto one another. The base implementation of cubic B-spline EMD is:

```
imfs, hts, ifs = \
    emd.empirical_mode_decomposition(knots=knots,
                                     knot_time=knot_time,
                                     spline_method='b_spline')[:3].
```

EMD using cubic Hermite spline interpolation (CHSI) and Akima spline interpolation (ASI) is performed in Egambaram et al. (2016) to demonstrate their effectiveness in removing eye-blink artifacts from electroencephalograms. The basis for both CHSI and ASI is:

$$\begin{aligned} p(t) = h_{00}(t^*)y(\tau_k) + h_{10}(t^*)(\tau_{k+1} - \tau_k)m(\tau_k) \\ + h_{01}(t^*)y(\tau_{k+1}) + h_{11}(t^*)(\tau_{k+1} - \tau_k)m(\tau_{k+1}), \end{aligned} \tag{70}$$

with,

$$\begin{aligned} t^* &= \frac{t - \tau_k}{\tau_{k+1} - \tau_k}, \\ h_{00}(t^*) &= 2{t^*}^3 - 3{t^*}^2 + 1, \\ h_{10}(t^*) &= {t^*}^3 - 2{t^*}^2 + t^*, \\ h_{01}(t^*) &= -2{t^*}^3 + 3{t^*}^2, \text{ and} \\ h_{11}(t^*) &= {t^*}^3 - {t^*}^2. \end{aligned} \tag{71}$$

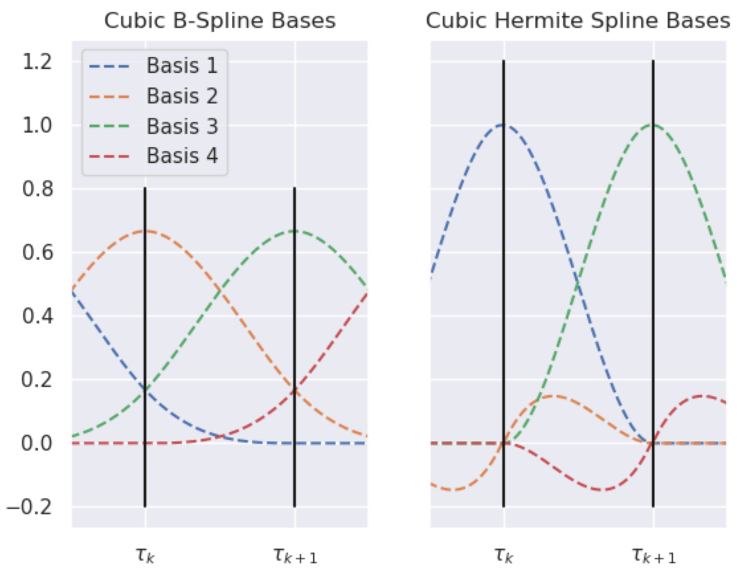

Figure 16: Comparison of cubic B-spline bases versus cubic Hermite spline bases between two arbitrary knots of uniform knot sequence.

Despite both CHSI and ASI using the same bases, the interpolation techniques differ when calculating the derivative values $m(\tau_k)$ and $m(\tau_{k+1})$. The positional values for both techniques are calculated as $y(\tau_k) = s(\tau_k)$ and $y(\tau_{k+1}) = s(\tau_{k+1})$. One can note the differences between the cubic Hermite bases and the cubic B-spline bases in Figure 16.

10.6.1 Cubic Hermite Spline Interpolation

There are a number of methods used to calculate the tangential parameters for CHSI curve-fitting (`spline_method='chsi'`). With $t_i = \tau_k$ and $t_j = \tau_{k+1}$ being the time points corresponding to the knot points, a common method, with low computational expense, for calculating the tangential values are:

$$m_{CHSI}(\tau_k) = \frac{1}{2}\left(\frac{s(t_{i+1}) - s(t_i)}{t_{i+1} - t_i} + \frac{s(t_i) - s(t_{i-1})}{t_i - t_{i-1}}\right), \tag{72}$$

and

$$m_{CHSI}(\tau_{k+1}) = \frac{1}{2}\left(\frac{s(t_{j+1}) - s(t_j)}{t_{j+1} - t_j} + \frac{s(t_j) - s(t_{j-1})}{t_j - t_{j-1}}\right). \tag{73}$$

10.6.2 *Akima Spline Interpolation*

ASI (`spline_method='asi'`) should be viewed as a specific case of CHSI as the bases are the same with the only difference being a defined method for calculating the tangential values. With t_i and t_j being defined as above and with:

$$v(t_i) = \frac{s(t_{i+1}) - s(t_i)}{t_{i+1} - t_i}, \tag{74}$$

the tangential values for ASI are defined as:

$$m_{ASI}(\tau_k) = \frac{|v(t_{i+1}) - v(t_i)| \times v(t_{i-1}) + |v(t_{i-1}) - v(t_{i-2})| \times v(t_i)}{|v(t_{i+1}) - v(t_i)| + |v(t_{i-1}) - v(t_{i-2})|}, \tag{75}$$

and

$$m_{ASI}(\tau_{k+1}) = \frac{|v(t_{j+1}) - v(t_j)| \times v(t_{j-1}) + |v(t_{j-1}) - v(t_{j-2})| \times v(t_j)}{|v(t_{j+1}) - v(t_j)| + |v(t_{j-1}) - v(t_{j-2})|}. \tag{76}$$

10.6.3 *Recommendations for Spline Methods in AdvEMDpy Package*

Cubic Hermite splines and Akima splines lack second-order continuity and as such are not as smooth as cubic B-splines. This may also lead to overfitting and the higher frequency IMFs being unwanted smoothed noise. In addition to using cubic B-splines interpolation method (`spline_method='b_spline'`), the user should also smooth the cubic B-splines (`smooth=True`) for reasons already stated in Section 10.4.

```
imfs, hts, ifs = emd.empirical_mode_decomposition(knots=knots,
                                                   knot_time=knot_time,
                                                   spline_method='b_spline',
                                                   smooth=True,
                                                   smoothing_penalty=0.1)[:3]
```

10.7 *Discrete-Time Hilbert Transforms in AdvEMDpy Package*

Despite B-splines having closed form solutions to the HT, discrete solutions to the HT may be needed for a number of reasons. Here are two widely known and used methods that are included. The base implementation of the DTHT is as follows:

```
imfs, hts, ifs = emd.empirical_mode_decomposition(knots=knots,
                                                   knot_time=knot_time,
                                                   dtht=False,
                                                   dtht_method='fft')[:3].
```

10.7.1 Basic DTHT

This method (`dtht_method='kak'`) is stated and proved in Kak (1970) and is often referred to in the literature as the Basis DTHT - it is quick and accurate. The Basic DTHT is calculated as follows:

$$\bar{\mathcal{H}}(X)(t[k]) \approx \begin{cases} \sum_{n \text{ is odd}} \dfrac{2}{\pi(k-n)} X(t[n]), & k \text{ is even} \\ \sum_{n \text{ is even}} \dfrac{2}{\pi(k-n)} X(t[n]), & k \text{ is odd} \end{cases}. \tag{77}$$

10.7.2 FFT DTHT

This method (`dtht_method='fft'`), which follows Python Core Team (2019), relies on the relationship that exists between the Fourier transform and the HT. The fast-Fourier transform is performed on the time series resulting in the below:

$$\bar{\mathcal{F}}(X)(k) = \sum_{n=0}^{N-1} e^{-2\pi i(\frac{kn}{N})} X(n). \tag{78}$$

The inverse fast-Fourier transform is then performed on twice the positive frequency components, zero times the negative frequency components, and one times the zero frequency component. There is a slight difference in implementation when the time series is of odd or even length. When the length of the time series, N, is even:

$$\begin{aligned} \bar{\mathcal{H}}(X)(t) \approx & \frac{1}{N}\sum_{n=0}^{N-1} X(n) \\ & + \frac{2}{N}\sum_{k=1}^{\frac{N}{2}-1} e^{2\pi i(\frac{kn}{N})}\bar{\mathcal{F}}(X)(k) \\ & + \frac{1}{N}(-1)^n \sum_{n=0}^{N-1} (-1)^n X(n), \end{aligned} \tag{79}$$

and when the length, N, is odd:

$$\begin{aligned} \bar{\mathcal{H}}(X)(t) \approx & \frac{1}{N}\sum_{n=0}^{N-1} X(n) \\ & + \frac{2}{N}\sum_{k=1}^{\frac{N-1}{2}} e^{2\pi i(\frac{kn}{N})}\bar{\mathcal{F}}(X)(k). \end{aligned} \tag{80}$$

10.7.3 Recommendations for Discrete-Time Hilbert Transforms in AdvEMDpy Package

A DTHT is not output by default, but should the user want a DTHT as well as the closed-form cubic B-spline HT for comparison, the FFT DTHT is recommended. The FFT is significantly faster than the Basic DTHT owing to the relationship that exists between the Fourier transform and the HT. Despite this significant increase in computational speed, it is, however, slightly less accurate than the Basic DTHT - especially at the edges of the time series.

```
imfs, hts, ifs, _, _, discrete_time_hts, discrete_time_ifs = \
    emd.empirical_mode_decomposition(knots=knots,
                                     knot_time=knot_time,
                                     dtht=True,
                                     dtht_method='fft')
```

10.8 Knot Point Optimisations in AdvEMDpy Package

Two methods are available for the optimisation of knot point allocation. Both methods are in the bisection family of knot optimisation techniques. The first technique simply bisects the domain iteratively until some error bound is met. The next method is a variation on this that extends the domain or diminishes the domain based on some error bound. The base implementation of the knot point optimisation is as follows:

```
imfs, hts, ifs = emd.empirical_mode_decomposition(knots=knots,
                                                   knot_time=knot_time,
                                                   optimise_knots=0,
                                                   knot_method='ser_bisect',
                                                   knot_error=10,
                                                   knot_lamda=1,
                                                   knot_epsilon=0.5)[:3].
```

10.8.1 Bisection

The bisection method (`knot_method='bisection'`) is the most basic of knot point optimisation techniques in that, given an error term (`knot_error=10`), the distance between two knots is iteratively halved until the sum of absolute errors between the spline and the time series over this interval is below the error term. In Figure 17, given knot point τ_k, the next knot, τ_{k+1}^0, is assumed to be the end of the time series (to start the iterative process). If the error between the spline and the time series is too large the distance between the knots is halved.

The spline is fitted again with the next knot point, τ_{k+1}^1, and if the error is still too large, the distance is halved again. The spline is then fitted between τ_k and τ_{k+1}^2. If the calculated error is below the maximum allowed error, then the iterative knot optimisation stops, τ_{k+1}^2 is accepted as an optimised knot point (τ_{k+1}). The knot optimisation process is repeated for $\tau_{k+1} = \tau_{k+1}^2$ and $\tau_{k+2}^0 = \tau_{k+1}^0$.

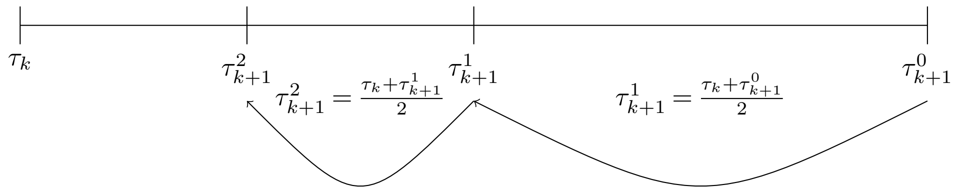
τ_k
τ_{k+1}^2
$\tau_{k+1}^2 = \frac{\tau_k + \tau_{k+1}^1}{2}$
τ_{k+1}^1
$\tau_{k+1}^1 = \frac{\tau_k + \tau_{k+1}^0}{2}$
τ_{k+1}^0

Figure 17: Diagram demonstrating bisection knot point optimisation method where distance between last accepted knot and next potential knot is continuously halved until stopping criteria satisfied.

10.8.2 Serial Bisection

The serial bisection knot optimisation method (`knot_method='ser_bisect'`) is introduced in Dung and Tjahjowidodo (2017). As above, the domain is bisected until the calculated error is below the maximum allowed error. Once the error is satisfied, however, the domain between potential knot points is extended again by half the distance between the knot points. If the calculated error is now more than the maximum error, the distance is decreased by half the distance it was increased by previously, if the calculated error is still less than the maximum allowed error the distance is increased by half the distance it was previously increased and so forth until the distance by which the distance is increased or decreased is less than ϵ (`knot_epsilon=0.5`).

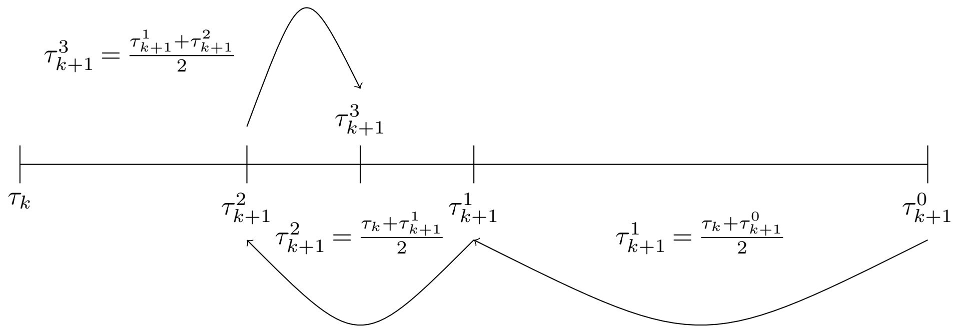
$\tau^3_{k+1} = \frac{\tau^1_{k+1}+\tau^2_{k+1}}{2}$
τ^3_{k+1}
τ_k
τ^2_{k+1}
$\tau^2_{k+1} = \frac{\tau_k+\tau^1_{k+1}}{2}$
τ^1_{k+1}
$\tau^1_{k+1} = \frac{\tau_k+\tau^0_{k+1}}{2}$
τ^0_{k+1}

Figure 18: Diagram demonstrating serial bisection knot point optimisation method where distance between last accepted knot and next potential knot can also be increased.

Since a cubic B-spline is used to optimise the knot allocation, a λ value (`knot_lamda=1`) is used to smooth the splines that are fitted to prevent abrupt changes in the coefficients of the B-splines to accommodate the maximum allowed error. Any spline-fitting technique may be used to optimise the knot point allocation, but B-splines are used in this package.

10.8.3 Recommendations for Knot Point Optimisations in AdvEMDpy Package

There are several options and combinations of options available to the user. Depending on the length and complexity of the time series, it is advised that the algorithm is first run with a uniform set of knots (either user-defined or defaults) with `optimise_knots=0`. Once the algorithm has run without error, the user may opt for the knots to be optimised once at the outset (`optimise_knots=1`). The allowed error (`knot_error=10`) and the allowed minimum knot placement distance (`knot_epsilon=0.5`) are both very time series specific and should be adjusted with care. The Bisection method (`knot_method='bisection'`) is significantly faster than the Serial Bisection method (`knot_method='ser_bisect'`), but does result in slightly more knot points. One should first run the Bisection method with one optimisation at the outset. If one wants knots that dynamically adjust to each potential IMF, then one can use `optimise_knots=2`.

```
imfs, hts, ifs, _, optimised_knots, _, _ = \
    emd.empirical_mode_decomposition(knots=knots,
                                     knot_time=knot_time,
                                     optimise_knots=1,
                                     knot_method='bisection',
                                     knot_error=10,
                                     knot_lamda=1,
                                     knot_epsilon=0.5)
```

10.9 Ensemble Empirical Mode Decomposition in AdvEMDpy Package

This is a noise-assisted data analysis (NADA) technique introduced in Wu and Huang (2009) that utilises white noise. This technique relies on the ability of the algorithm to discern consistent structures in the presence of different sets of randomised white noise. The base implementation of the EEMD is implemented as follows:

```
imfs, hts, ifs = emd.empirical_mode_decomposition(knots=knots,
                                                  knot_time=knot_time,
                                                  ensemble=False,
                                                  ensemble_sd=0.5,
                                                  ensemble_iter=10)[:3].
```

The original time series is median filtered to approximate the original level of noise in the system. The median is removed from the time series before the standard deviation is calculated, σ_s. The noise added to the system has a mean of zero and a standard deviation of $\sigma_s \times \sigma_e$, where σ_e (`ensemble_sd=0.5`) is the chosen level of standard deviation in the ensemble system as a fraction of the original standard deviation in the system. The IMFs are calculated for this modified time series and stored before a new set of noise is added to the original time series and the sifting process is repeated - the procedure is repeated a predetermined number of iterations (`ensemble_iter=10`).

Further applications of this technique will benefit from random additions of other colours of noise such as violet noise, blue noise, pink noise, and red noise (also known as Brownian noise), which all have non-constant power spectral densities, compared with white noise that has a constant power spectral density. This technique, with the random addition of other colours of noise, is called Full-Spectrum Ensemble Empirical Mode Decomposition (FSEEMD). This technique is experimental and is also included with the package in the script entitled `emd_experimental.py` for completeness sake.

This method performs a similar task, utilising a different methodology, to ICA-EMD, introduced in van Jaarsveldt et al. (2021), which is finding the most consistent structures amongst noisy data. EEMD proceeds by introducing noise to the time series and isolating IMFs from the noisy time series. This task is repeated several times and IMFs are simply averaged. A more sophisticated sorting technique such as the Minimum Description Length Principle (introduced in Fayyad and Irani (1993)) should be used to ensure structures of similar frequency are grouped as in van Jaarsveldt et al. (2021). ICA-EMD proceeds by applying Independent Component Analysis (ICA) directly to all the noisy IMFs to isolate the most consistent structure, before EMD is performed on the ICA component to isolate IMFs.

10.9.1 Recommendations for Ensemble Empirical Mode Decomposition in AdvEMDpy Package

This technique shows promise, but it should be implemented with care. The implementation of the EEMD method removes some additional features such as the output of coefficients, knots, as well as the DTHT and IF from the corresponding DTHT. One should apply this technique after applying EMD for comparison.

```
imfs, hts, ifs = emd.empirical_mode_decomposition(knots=knots,
                                                  knot_time=knot_time,
                                                  ensemble=True,
                                                  ensemble_sd=0.5,
                                                  ensemble_iter=10)
```

S Illustrative Examples of EMD Method

In these supplementary examples we provide two additional case studies. The intentions are firstly to demonstrate the application of the toolbox in a real data example and secondly to compare the accuracy of the methods implement in the proposed toolbox versus other existing EMD packages, comparing **AdvEMDpy** against **PyEMD 0.2.10** and **emd 0.3.3**.

In Section S.1, a synthetic example is created using a specific case of the Duffing equation as provided in Huang et al. (1998) to demonstrate using these scripts how to arrive at the desired output. The accuracy of the three methods is quantitatively compared by calculating the absolute deviations of the second IMFs extracted using each method with the underlying driving function.

In Section S.2, a well-known real-world data set is used to demonstrate how EMD can be applied to real-world data. The knot sequence would need to be appropriately constructed. Should the data (unlike the Carbon Dioxide data) be noisy, some preprocessing (Package **AdvEMDpy**: Algorithmic Variations of Empirical Mode Decomposition in Python, Section 7.2) may be appropriate or one would benefit by not performing initial smoothing (`initial_smoothing=False`) as all the noise would be captured in a nonsensical first IMF. As there is no true underlying IMF structure as in Section S.1, the three methods are compared by measuring how accurately the IFs of each method replicate the underlying annual structure.

S.1 Supplementary Example One

The motion of a mass attached to a frictionless spring in a vacuum that is perturbed slightly is governed by:

$$\frac{d^2x(t)}{dt^2} + \omega x(t) = 0, \tag{81}$$

with ω^2 (squared by convention) being the constant natural frequency of the system and 0 being the driving force of the system - it oscillates about zero. A specific Duffing equation is governed by:

$$\frac{d^2x(t)}{dt^2} + \left(1 - x^2(t)\right)x(t) = \frac{1}{10}\cos\left(\frac{1}{25}2\pi t\right), \tag{82}$$

with the natural frequency of the system being a function of the displacement and the driving force of the system no longer being zero. This creates a more complex system than a simple sinusoid (solution to Equation (81)) - the displacement and velocity can be viewed in Figure 19. The simple script below can be used to produce Figure 19. For further details on how to reproduce this figure, and all others in this section, one can see `aas_figures_replication_script.py` and `Worked_examples.ipynb`.

```
def duffing_equation(XY, t):
    return [XY[1], XY[0] - XY[0] ** 3 + 0.1 * np.cos(0.08 * np.pi * t)]

t = np.linspace(0, 150, 1501)
XY0 = [1, 1]
solution = odeint(duffing_equation, XY0, t)
x = solution[:, 0]
dxdt = solution[:, 1]
```

In a frictionless environment, the total energy of the system is completely determined by the initial conditions of $x(0) = 1$ and $\frac{dx(0)}{dt} = 1$. The Hamiltonian frequency of the system (in the absence of a driving function) can be shown to be $f = 0.124Hz$. This, as well as the frequency of the driving function, $f = 0.04Hz$, can be seen in Figure 21, Figure 22, and Figure 23. One can extract the IMFs from the structure using the code below.

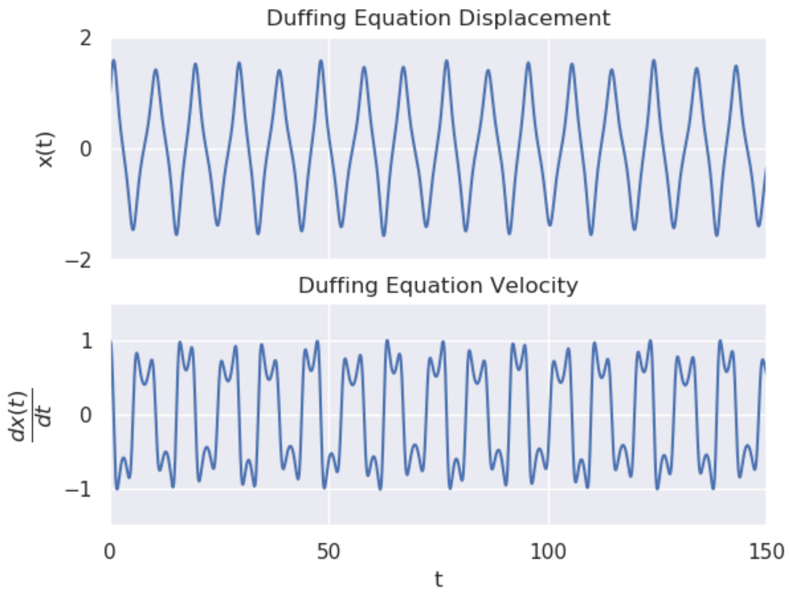

Figure 19: Figure demonstrating Duffing equation (Equation (82)) displacement and velocity for $t \in [0, 150]$.

```
emd_duffing = EMD(time=t, time_series=x)
emd_duff, emd_ht_duff, emd_if_duff = \
    emd_duffing.empirical_mode_decomposition()[:3]
```

In Figure 20 one can see how accurately the second IMF structure extracted captures the forcing function in the Duffing equation. For comparison, IMF 1 and IMF 2 extracted using **PyEMD 0.2.10** and **emd 0.3.3** are also plotted. The various methods are also quantitatively compared by measuring the sum of absolute differences between the second IMFs of each method against the true underlying driving function, the errors are tabulated below:

$$\begin{aligned}\sum_{i=0}^{1500}\left|IMF_2^{emd}(t_i) - 0.1\cos(0.08\pi(t_i))\right| &= 21.434\\ \sum_{i=0}^{1500}\left|IMF_2^{PyEMD}(t_i) - 0.1\cos(0.08\pi(t_i))\right| &= 20.774\\ \sum_{i=0}^{1500}\left|IMF_2^{AdvEMDpy}(t_i) - 0.1\cos(0.08\pi(t_i))\right| &= 15.978,\end{aligned} \tag{83}$$

with $IMF_j^k(t_i)$ being IMF j isolated using method k at time point t_i. From Equation (83) it can be seen that **AdvEMDpy** most closely captures the underlying driving function.

The Hilbert spectrum in Figure 23 captures the complex frequency structures present more accurately than **PyEMD 0.2.10** or **emd 0.3.3**. The Hilbert spectrum can be calculated using the following code.

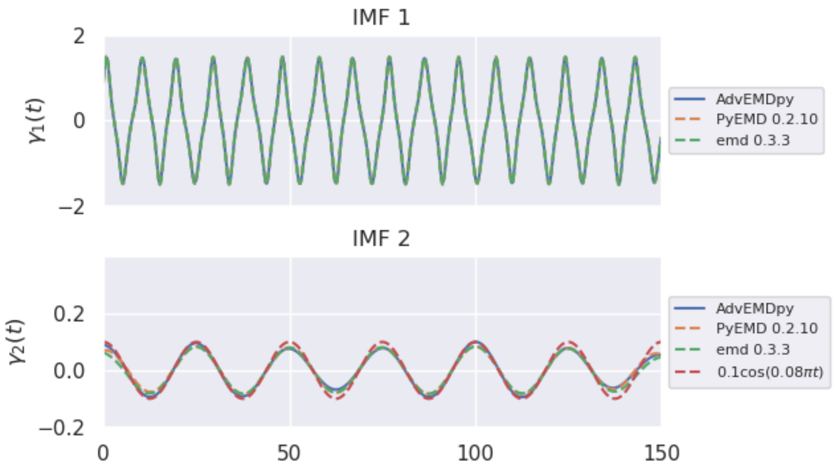

Figure 20: Figure demonstrating the first three IMFs extracted from Duffing equation displacement along with driving force equation displayed alongside IMF 2.

```
x_hs, y, z = hilbert_spectrum(t, emd_duff, emd_ht_duff, emd_if_duff,
                              max_frequency=1.3, plot=False)
y = y / (2 * np.pi)  # convert from angular frequency
ax.pcolormesh(x_hs, y, np.abs(z), cmap='gist_rainbow',
              vmin=0, vmax=np.abs(z).max())
```

For completeness, the Hilbert spectrum of the other two methods can be obtained by using the following code. One can note that in addition to the increased accuracy of **AdvEMDpy** in resolving the underlying driving function, the resolution of the driving function in the Hilbert spectrum of Figure 23 is not nearly as obscured by the IF of the first IMF as in Figure 21 and Figure 22.

```
pyemd = pyemd0215()
py_emd = pyemd(x)
IP, IF, IA = emd040.spectra.frequency_transform(py_emd.T, 10, 'hilbert')
freq_edges, freq_bins = emd040.spectra.define_hist_bins(0, 0.2, 100)
hht = emd040.spectra.hilberthuang(IF, IA, freq_edges)
hht = gaussian_filter(hht, sigma=1)
plt.pcolormesh(t, freq_bins, hht, cmap='gist_rainbow',
              vmin=0, vmax=np.max(np.max(np.abs(hht))))

emd_sift = emd040.sift.sift(x)
IP, IF, IA = emd040.spectra.frequency_transform(emd_sift, 10, 'hilbert')
freq_edges, freq_bins = emd040.spectra.define_hist_bins(0, 0.2, 100)
hht = emd040.spectra.hilberthuang(IF, IA, freq_edges)
hht = gaussian_filter(hht, sigma=1)
plt.pcolormesh(t, freq_bins, hht, cmap='gist_rainbow',
              vmin=0, vmax=np.max(np.max(np.abs(hht))))
```

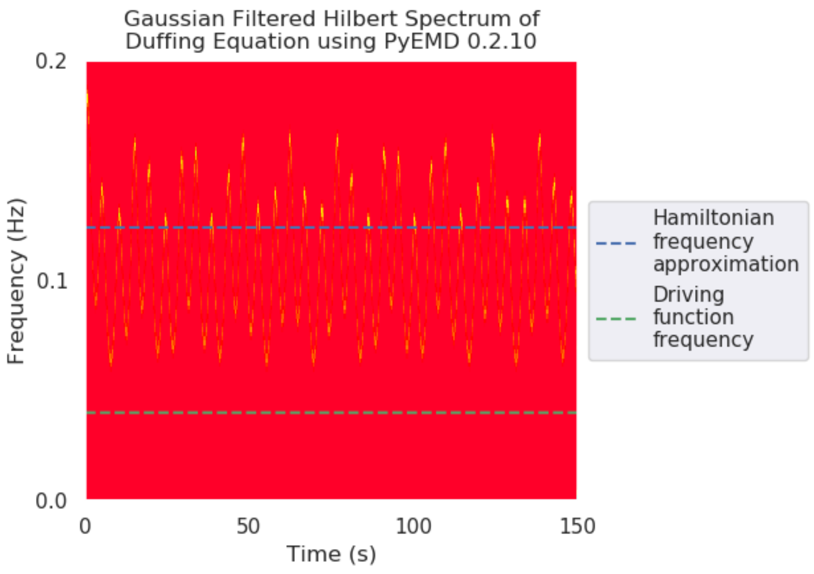

Figure 21: Hilbert spectrum of Duffing equation IMFs using **PyEMD 0.2.10**.

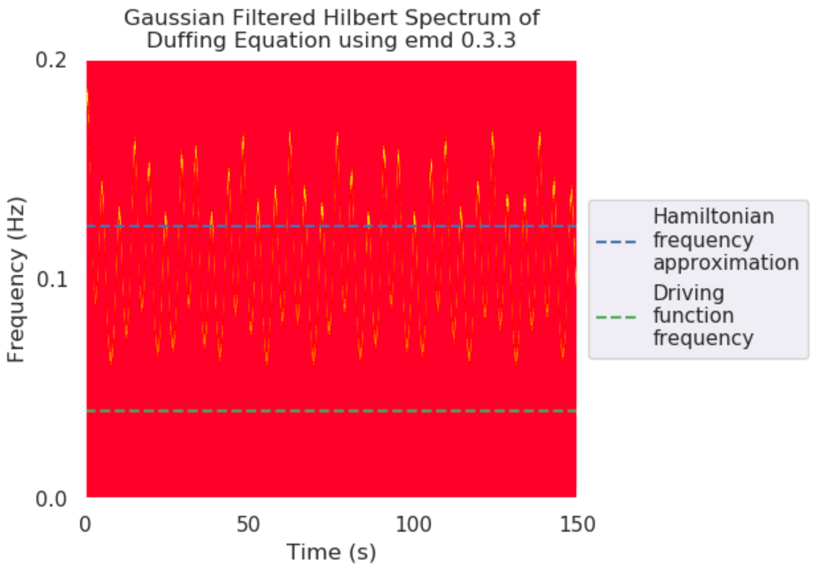

Figure 22: Hilbert spectrum of Duffing equation IMFs using **emd 0.3.3**.

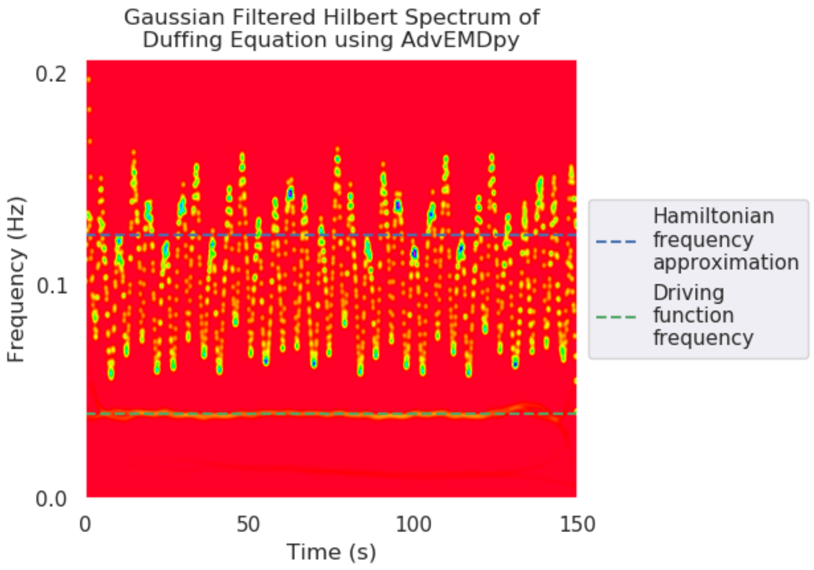

Figure 23: Hilbert spectrum of Duffing equation IMFs using **AdvEMDpy** with frequency of driving force equation visible despite being of lower intensity.

S.2 *Supplementary Example Two*

One would benefit from an example applied to real-world data. The following well-known data set is from Tans and Keeling (2020) to demonstrate the effectiveness of **AdvEMDpy** over **PyEMD 0.2.10** and **emd 0.3.3**. One can see the transient nature of the data in Figure 24 which hinders analysis using base-level applications of **PyEMD 0.2.10** and **emd 0.3.3**. One can replicate Figure 24 using the below code.

```
CO2_data = pd.read_csv('../Data/co2_mm_mlo.csv', header=51)
plt.plot(CO2_data['month'], CO2_data['decimal date'])
```

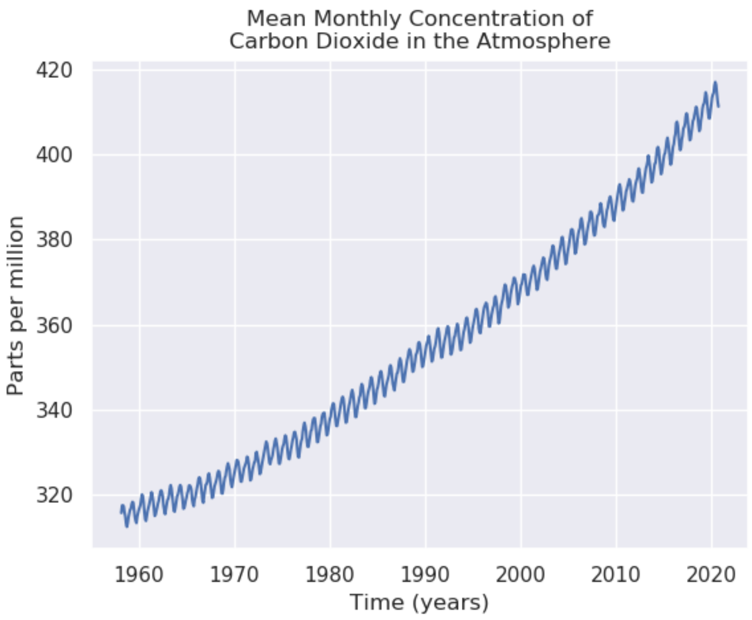

Figure 24: Monthly mean atmospheric concentration of Carbon Dioxide in parts per million from March 1958 until September 2020.

One can apply EMD to the time series with the following code:

```
co2_knots = np.linspace(co2_time[0], co2_time[-1], 200)
emd = EMD(time=co2_time, time_series=co2_time_series)
imfs, hts, ifs = \
    emd.empirical_mode_decomposition(knots=co2_knots,
                                     knot_time=co2_time)[:3].
```

and plotting the resulting outputs, one will arrive at Figure 25. The number of knots is increased above the base-level application to accurately capture the annual cycle. The original time series is plotted as well as the smoothed version of the time series. A single IMF has been extracted as well as a residual after the original time series was smoothed.

To plot the Hilbert spectrum, as seen in Figure 28, one can implement the following code:

```
x_hs, y, z = hilbert_spectrum(time, imfs, hts, ifs,
                              max_frequency=10, which_imfs=[1],
                              plot=False)
```

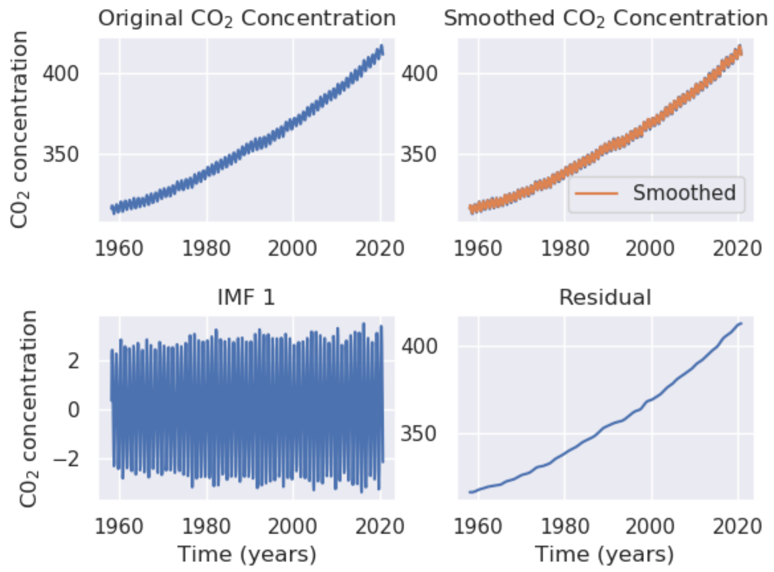

Figure 25: EMD of monthly mean Carbon Dioxide concentration in parts per million from March 1958 until September 2020.

```
y = y / (2 * np.pi)
fig, ax = plt.subplots()
ax.pcolormesh(x_hs, y, np.abs(z), cmap='gist_rainbow',
              vmin=0, vmax=np.abs(z).max()).
```

The accuracy of **AdvEMDpy** versus **PyEMD 0.2.10** and **emd 0.3.3** in the resolution of the annual structure can be seen in Equation (84).

$$\begin{aligned}\sum_{i=0}^{750}\left|IF_1^{PyEMD}(t_i)-1\right| &= 181.076\\ \sum_{i=0}^{750}\left|IF_1^{emd}(t_i)-1\right| &= 174.047\\ \sum_{i=0}^{750}\left|IF_1^{AdvEMDpy}(t_i)-1\right| &= 37.616,\end{aligned}\tag{84}$$

with $IF_j^k(t_i)$ being the IF of IMF j using method k at time point t_i. The annual cycle is most accurately resolved in Figure 28 compared against Figure 26 and Figure 27. To reproduce Figure 26 one can implement the code below.

```
pyemd = pyemd0215()
py_emd = pyemd(signal)
IP, IF, IA = emd040.spectra.frequency_transform(py_emd[:2, :].T,
                                                 12, 'hilbert')
freq_edges, freq_bins = emd040.spectra.define_hist_bins(0, 2, 100)
```

```
hht = emd040.spectra.hilberthuang(IF, IA, freq_edges)
hht = gaussian_filter(hht, sigma=1)
fig, ax = plt.subplots()
plt.pcolormesh(time, freq_bins, hht, cmap='gist_rainbow',
               vmin=0, vmax=np.max(np.max(np.abs(hht))))
```

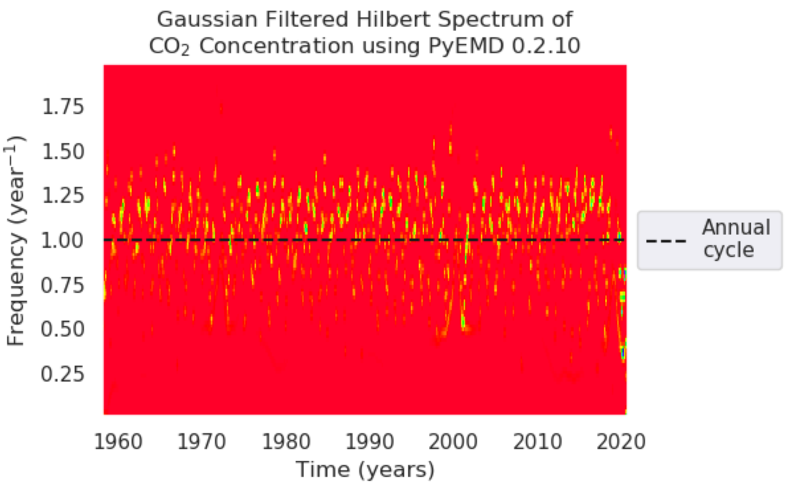

Figure 26: HT of monthly mean Carbon Dioxide concentration in parts per million from March 1958 until September 2020 using **PyEMD 0.2.10**.

To reproduce Figure 27 one can implement the code below. By comparing Figure 26 and Figure 27 against Figure 28 and by comparing the equalities in Equation (84), one can note the increased resolution power of **AdvEMDpy** over **PyEMD 0.2.10** and **emd 0.3.3**.

```
emd_sift = emd040.sift.sift(signal)
IP, IF, IA = emd040.spectra.frequency_transform(emd_sift[:, :1],
                                                   12, 'hilbert')
freq_edges, freq_bins = emd040.spectra.define_hist_bins(0, 2, 100)
hht = emd040.spectra.hilberthuang(IF, IA, freq_edges)
hht = gaussian_filter(hht, sigma=1)
fig, ax = plt.subplots()
plt.pcolormesh(time, freq_bins, hht, cmap='gist_rainbow',
               vmin=0, vmax=np.max(np.max(np.abs(hht))))
```

In van Jaarsveldt et al. (2021), the resolution of EMD is improved even further by augmenting EMD with X11 and forming the new technique known as EMD-X11. X11 is a simpler trend filtering technique that functions as a post-processing technique in this setting that improves the resolution power of EMD. This will be included in a later version of the software along with many other novel contributions to the field.

In order to minimize carbon emissions in keeping with intergovernmental agreements, nations issue a certain number of allowances which allows the company the right to emit a certain amount of carbon dioxide into the atmosphere in the past business year. One such agreement is the EU's emissions agreement where one European Union Allowance (EUA) allows a company to emit one tonne of carbon dioxide in the previous business year. These EUAs are issued in lots of one thousand EUA which forms a very liquid market. These lots are traded as ECF1 and ECF2 on the EU Emissions Trading Strategy (ETS). The price in Euros of each of these instruments from 22 April 2005 until 17 June 2022 can be seen in Figure 29.

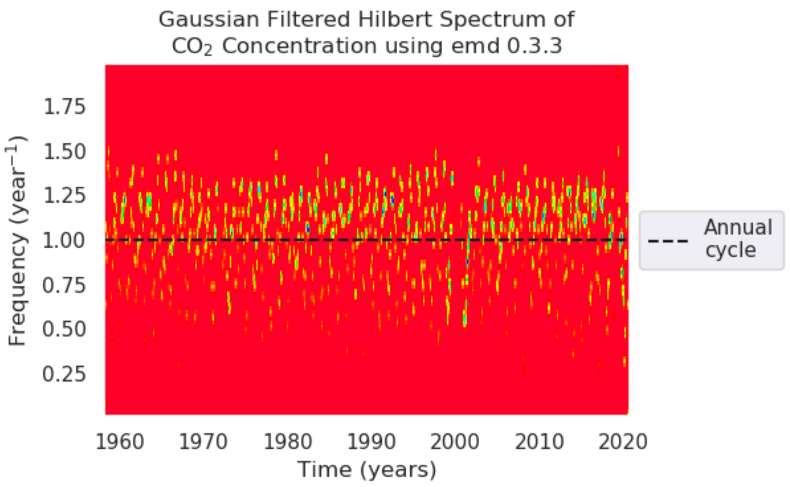

Figure 27: HT of monthly mean Carbon Dioxide concentration in parts per million from March 1958 until September 2020 using **emd 0.3.3**.

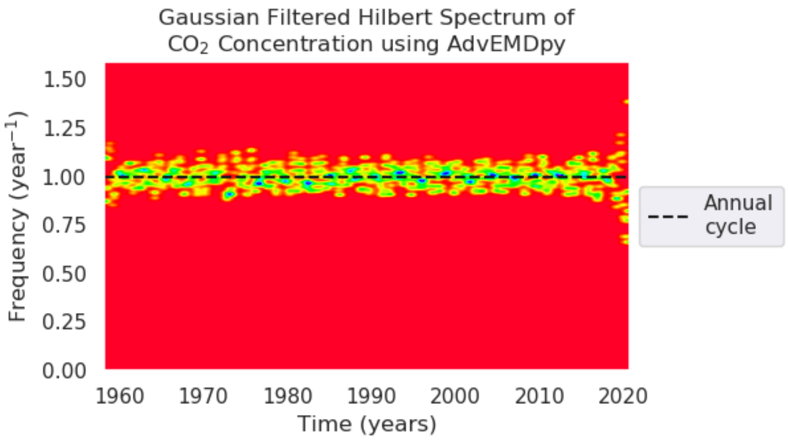

Figure 28: HT of monthly mean Carbon Dioxide concentration in parts per million from March 1958 until September 2020 using **AdvEMDpy**.

In Figure 30 and Figure 31 the Hilbert transform of the first IMF from ECF1 and ECF2 are presented, respectively. In Figure 30, the results are somewhat disturbed from the presence of the discontinuity in ECF1 which is mostly removed in the first IMF. In Figure 31, the twelve-to-thirteen day cycle is clearly visible.

In Figure 31, the IFs of IMF 2 and IMF 3 from both ECF1 and ECF2 are shown - lower frequency structures are iteratively removed. In Figure 33, the annual structure, first seen in 28, is shown relative to the fifth IMF of both. There is a clear causal link between the two processes, and EMD has assisted in establishing an annual cycle is present in the ECF1 and ECF2 data which can be used in Carbon emission instrument modelling.

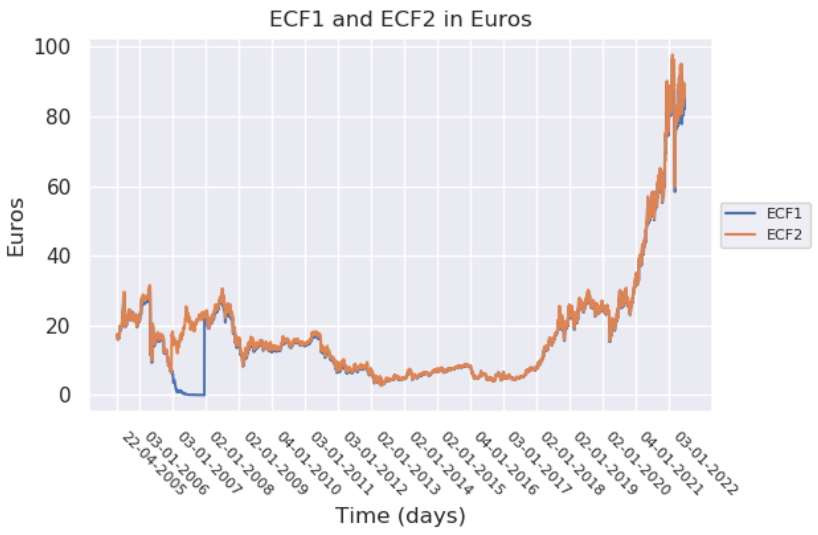

Figure 29: ECF1 and ECF2 in Euros from 22 April 2005 until 17 June 2022.

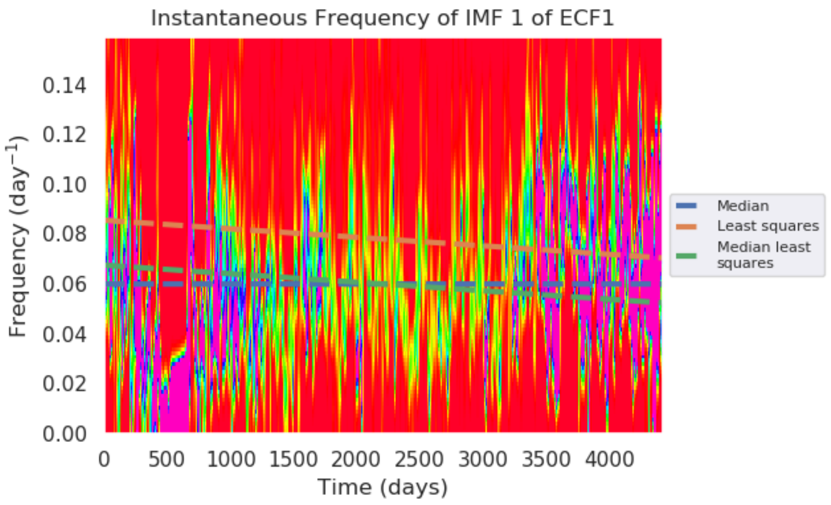

Figure 30: IF of IMF 1 for ECF1 in Euros from 22 April 2005 until 17 June 2022 with median and best fits.

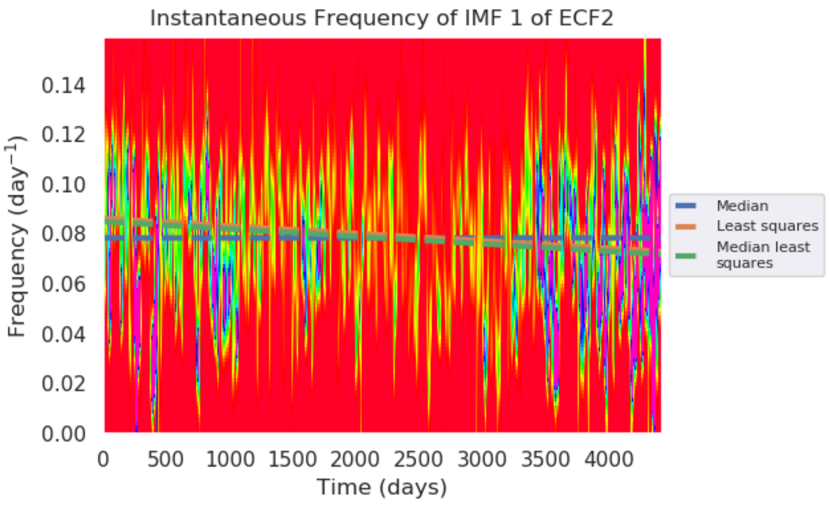

Figure 31: IF of IMF 1 for ECF2 in Euros from 22 April 2005 until 17 June 2022 with median and best fits.

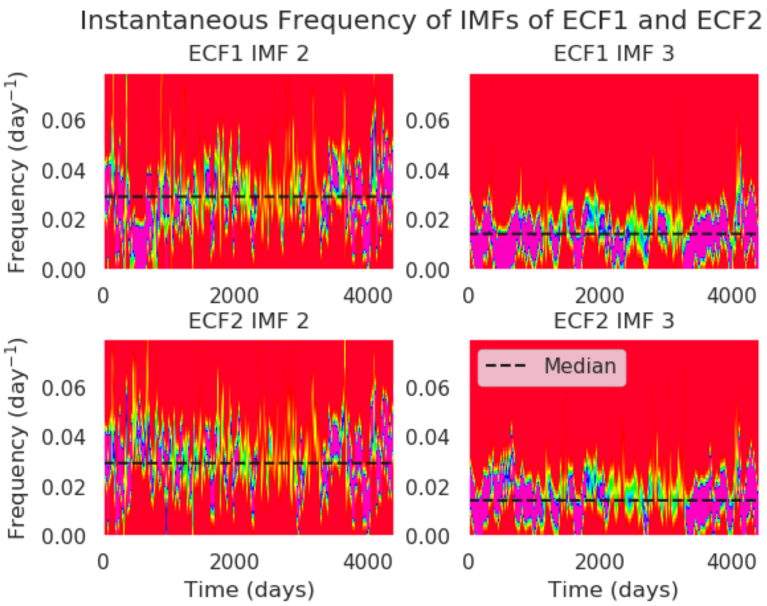

Figure 32: IF of IMF 2 and IMF 3 for ECF1 and ECF2 in Euros from 22 April 2005 until 17 June 2022 with medians.

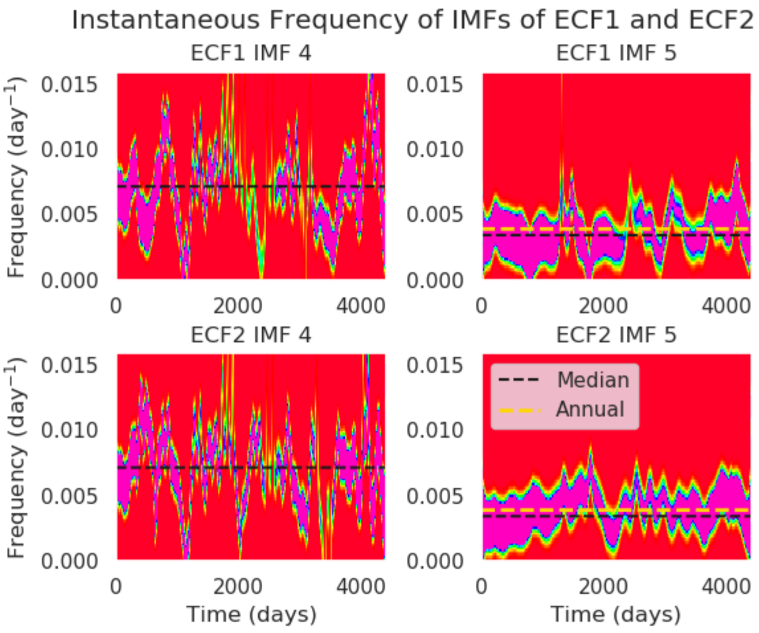

Figure 33: IF of IMF 4 and IMF 5 for ECF1 and ECF2 in Euros from 22 April 2005 until 17 June 2022 with medians and annual structure.

References

S. Bianconcini. 2006. *Trend-Cycle Estimation in Reproducing Kernel Hilbert Spaces*. Ph. D. Dissertation. Department of Statistics, University of Bologna, Bologna.

E. Candès. 2006. Compressive Sampling. In *Proceedings of the International Congress of Mathematicians*, M. Sanz-Solé, J. Soria, J. Varona, and J. Verdera (Eds.), Vol. 3. European Mathematical Society, European Mathematical Society Publishing, Madrid, 1433–1452. https://doi.org/10.4171/022-3/69

E. Candès, J. Romberg, and T. Tao. 2006. Robust Uncertainty Principles: Exact Signal Reconstruction from Highly Incomplete Frequency Information. *IEEE Transactions on Information Theory* 52, 2 (2006), 489–509. https://doi.org/10.1109/TIT.2005.862083

E. Candès and M. Wakin. 2008. An Introduction to Compressive Sampling. *IEEE Signal Processing Magazine* 25, 2 (2008), 21–30. https://doi.org/10.1109/MSP.2007.914731

Q. Chen, N. Huang, S. Riemenschneider, and Y. Xu. 2006. A B-spline Approach for Empirical Mode Decompositions. *Advances in Computational Mathematics* 24, 1-4 (2006), 171–195. https://doi.org/10.1007/s10444-004-7614-3

F. Chiew, M. Peel, G. Amirthanathan, and G. Pegram. 2005. Identification of oscillations in historical global streamflow data using empirical mode decomposition. In *Regional Hydrological Impacts of Climatic Change - Hydroclimatic Variabiltiy* (Brazil), S. Franks, T. Wagener, E. Bøgh, H. Gupta, L. Bastidas, C. Nobre, and C. Galvão (Eds.), Vol. 296. International Association of Hydrological Sciences, 53–62. https://www.cabdirect.org/cabdirect/abstract/20053083220

K. Coughlin and K. Tung. 2004. 11-Year solar cycle in the stratosphere extracted by the empirical mode decomposition method. *Advances in Space Research* 34, 2 (2004), 323–329. https://doi.org/10.1016/j.asr.2003.02.045

R. Crochiere and L. Rabiner. 1983. Multirate Digital Signal Processing. In *Prentice-Hall Signal Processing Series: Advanced Monographs*, A. Oppenheim (Ed.). Prentice-Hall, Inc., Englewood Cliffs, New Jersey, SA. https://books.google.co.uk/books?id=X_NSAAAAMAAJ

E. Dagum and S. Bianconcini. 2006. Local Polynomial Trend-Cycle Predictors in Reproducing Kernel Hilbert Spaces for Current Economic Analysis. In *Anales de Economia Aplicada*. 1–16.

E. Dagum and S. Bianconcini. 2008. The Henderson Smoother in Reproducing Kernel Hilbert Space. *Journal of Business & Economic Statistics* 26, 4 (2008), 536–545. https://doi.org/10.1198/073500107000000322

M. Dätig and T. Schlurmann. 2004. Performance and limitations of the Hilbert-Huang transformation (HHT) with an application to irregular water waves. *Ocean Engineering* 31, 14-15 (2004), 1783–1834. https://doi.org/10.1016/j.oceaneng.2004.03.007

C. de Boor. 1978. *A Practical Guide to Splines*. Applied Mathematical Sciences, Vol. 27. Springer-Verlag, New York, USA. https://doi.org/10.2307/2006241

E. De Forest. 1877. On Adjustment Formulas. *The Analyst* 4, 3 (1877), 79–86. https://doi.org/10.2307/2636257

Y. Deng, W. Wang, C. Qian, Z. Wang, and D. Dai. 2001. Boundary-processing-technique in EMD method and Hilbert transform. *Chinese Science Bulletin* 46, 1 (2001), 954–960. https://doi.org/10.1007/BF02900475

V. Dung and T. Tjahjowidodo. 2017. A direct method to solve optimal knots of B-spline curves: An application for non-uniform B-spline curves fitting. *PLoS ONE* 12, 3 (2017), e0173857. https://doi.org/10.1371/journal.pone.0173857

A. Egambaram, N. Badruddin, V. Asirvadam, and T. Begum. 2016. Comparison of Envelope Interpolation Techniques in Empirical Mode Decomposition (EMD) for Eyeblink Artifact Removal From EEG. In *IEEE EMBS Conference on Biomedical Engineering and Sciences (IECBES)*. IEEE, 590–595. https://doi.org/10.1109/IECBES.2016.7843518

P. Eilers and B. Marx. 1996. Flexible Smoothing with B-splines and Penalties. *Statist. Sci.* 11, 2 (1996), 89–121. https://doi.org/10.1214/ss/1038425655

U. Fayyad and K. Irani. 1993. Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93)*, Vol. 2. 1022–1027. http://hdl.handle.net/2014/35171

C. Hastings Jr., F. Mosteller, J. Tukey, and C. Winsor. 1947. Low Moments for Small Samples: A Comparative Study of Order Statistics. *The Annals of Mathematical Statistics* 18, 3 (1947), 413–426. https://doi.org/10.1214/aoms/1177730388

R. Henderson. 1916. Note on Graduation by Adjusted Average. *Transactions of the Actuarial Society of America* 17 (1916), 43–48.

R. Henderson. 1924. A New Method of Graduation. *Transactions of the Actuarial Society of America* 25 (1924), 29–40.

R. Hodrick and E. Prescott. 1997. Postwar US Business Cycles: An Empirical Investigation. *Journal of Money, Credit, and Banking* 29, 1 (1997), 1–16. https://doi.org/10.2307/2953682

N. Huang. 1999. Computer Implemented Empirical Mode Decomposition Method, Apparatus and Article of Manufacture. Patent. US Patent 5,983,162.

N. Huang, Z. Shen, and S. Long. 1999. A New View of Nonlinear Water Waves: The Hilbert Spectrum. *Annual Review of Fluid Mechanics* 31, 1 (1999), 417–457. https://doi.org/10.1146/annurev.fluid.31.1.417

N. Huang, Z. Shen, S. Long, M. Wu, H. Shih, Q. Zheng, N. Yen, C. Tung, and H. Liu. 1998. The empirical mode decomposition and the Hilbert spectrum for nonlinear and non-stationary time series analysis. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 454, 1971 (1998), 903–995. https://doi.org/10.1098/rspa.1998.0193

N. Huang and Z. Wu. 2008. A review on Hilbert-Huang transform: Method and its applications to geophysical studies. *Reviews of Geophysics* 46, 2 (2008), (RG2006) 1–23. https://doi.org/10.1029/2007RG000228

C. Junsheng, Y. Dejie, and Y. Yu. 2006. Research on the Intrinsic Mode Function (IMF) Criterion in EMD Method. *Mechanical Systems and Signal Processing* 20, 4 (2006), 817–824. https://doi.org/10.1016/j.ymssp.2005.09.011

S. Kak. 1970. The Discrete Hilbert Transform. *Proc. IEEE* 58, 4 (1970), 585–586. https://doi.org/10.1109/PROC.1970.7696

D. Kim, K. Kim, and H. Oh. 2012. Extending the scope of empirical mode decomposition by smoothing. *EURASIP Journal on Advances in Signal Processing* 168, 2012 (2012), 1–17. https://doi.org/10.1186/1687-6180-2012-168

Y. Kopsinis and S. McLaughlin. 2007a. Enhanced Empirical Mode Decomposition using a Novel Sifting-Based Interpolation Points Detection. In *Proceedings of the IEEE/SP 14th Workshop on Statistical Signal Processing (SSP'07)*. IEEE, Madison, Wisconsin, USA, 725–729. https://doi.org/10.1109/SSP.2007.4301354

Y. Kopsinis and S. McLaughlin. 2007b. Investigation of the Empirical Mode Decomposition Based on Genetic Algorithm Optimization Schemes. In *Proceedings of the 32nd IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'07)*, Vol. 3. IEEE, Honolulu, Hawaii, USA, 1397–1400. https://doi.org/10.1109/ICASSP.2007.367107

Y. Kopsinis and S. McLaughlin. 2008. Investigation and Performance Enhancement of the Empirical Mode Decomposition Method Based on a Heuristic Search Optimization Approach. *IEEE Transactions on Signal Processing* 56, 1 (2008), 1–13. https://doi.org/10.1109/TSP.2007.901155

J. Musgrave. 1964a. Alternative Sets of Weights for Proposed X-11 Seasonal Factor Curve Moving Averages. Working paper. U.S. Bureau of the Census, U.S. Department of Commerce, Washington D.C., USA.

J. Musgrave. 1964b. A Set of End Weights to End All End Weights. Working paper. U.S. Bureau of the Census, U.S. Department of Commerce, Washington D.C., USA.

C. Peng, S. Buldyrev, S. Havlin, M. Simons, H. Stanley, and A. Goldberger. 1994. Mosaic organization of DNA nucleotides. *Physical Review E: Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics* 49, 2 (1994), 1685–1689. https://doi.org/10.1103/PhysRevE.49.1685

G. Rilling, P. Flandrin, and P. Goncalves. 2003. On Empirical Mode Decomposition and its Algorithms. In *IEEE-EURASIP Workshop on Nonlinear Signal and Image Processing*, Vol. 3. NSIP-03, Grado (I), 8–11. https://hal.inria.fr/inria-00570628

A. Tabrizi, L. Garibaldi, A. Fasana, and S. Marchesiello. 2014. Influence of Stopping Criterion for Sifting Process of Empirical Mode Decomposition (EMD) on Roller Bearing Fault Diagnosis. In *Advances in Condition Monitoring of Machinery in Non-Stationary Operations*, G. Dalpiaz, R. Rubini, G. D'Elia, M. Cocconcelli, F. Chaari, R. Zimroz, W. Bartelmus, and M. Haddar (Eds.). Springer-Verlag, Berlin, Heidelberg, 389–398. https://doi.org/10.1007/978-3-642-39348-8_33

Python Core Team. 2019. *Python: A Dynamic, Open Source Programming Language*. Python Software Foundation, Amsterdam, Netherlands. https://www.python.org/ Python Version 3.7.4.

Tans, P. (NOAA-GML) (www.esrl.noaa.gov/gmd/ccgg/trends/) and Keeling, R. (Scripps Institution of Oceanography) (scrippsco2.ucsd.edu/). 2020. Trends in Atmospheric Carbon Dioxide. https://www.esrl.noaa.gov/gmd/ccgg/trends/data.html. Online; accessed 19 October 2020.

C. van Jaarsveldt, G. Peters, M. Ames, and M. Chantler. 2021. Tutorial on Empirical Mode Decomposition: Basis Decomposition and Frequency Adaptive Graduation in Non-Stationary Time Series. *Available at SSRN 3913330* (2021). https://doi.org/10.2139/ssrn.3913330

E. Whittaker. 1922. On a New Method of Graduation. *Proceedings of the Edinburgh Mathematical Society* 41 (1922), 63–75. https://doi.org/10.1017/S0013091500077853

F. Wu and L. Qu. 2008. An improved method for restraining the end effect in empirical mode decomposition and its applications to the fault diagnosis of large rotating machinery. *Journal of Sound and Vibration* 314, 3-5 (2008), 586–602. https://doi.org/10.1016/j.jsv.2008.01.020

Z. Wu and N. Huang. 2009. Ensemble Empirical Mode Decomposition: a noise-assisted data analysis method. *Advances in Adaptive Data Analysis* 1, 1 (2009), 1–41. https://doi.org/10.1142/S1793536909000047

K. Zeng and M. He. 2004. A Simple Boundary Process Technique for Empirical Mode Decomposition. In *IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, Vol. 6. IEEE, 4258–4261. https://doi.org/10.1109/IGARSS.2004.1370076

J. Zhao and D. Huang. 2001. Mirror Extending and Circular Spline Function for Empirical Mode Decomposition Method. *Journal of Zhejiang University A: Science* 2, 3 (2001), 247–252. https://doi.org/10.1007/BF02839453

Cite this article: C. van Jaarsveldt, M. Ames, G. Peters, and M. Chantler, Supplement to: Package **AdvEMDpy**: Algorithmic Variations of Empirical Mode Decomposition in Python. *Annals of Actuarial Science*, 1–47. 10.1017/xxxxx