ewstools Documentation

Thomas M Bury

Aug 04, 2023

CONTENTS

1 ewstools package **3**
- 1.1 ewstools.core submodule 3
- 1.2 ewstools.helpers submodule 10
- 1.3 ewstools.models submodule 13

2 Installation **17**

3 Demos **19**

4 Support **21**

5 License **23**

6 Indices and tables **25**

Python Module Index **27**

Index **29**

Early warning signals (EWS) for bifurcations in time series data.

CHAPTER

ONE

EWSTOOLS PACKAGE

1.1 ewstools.core submodule

The 'core' submodule contains high-level functions designed for the user.

class ewstools.core.**MultiTimeSeries**(*data*, *transition=None*)

Bases: object

Multivariate time series data on which to compute early warning signals.

Parameters

- **data** (*pandas.DataFrame*) – Contains each time series as a column. Index represents time values and is carried over.
- **transition** (*float*) – Time value at which transition occurs, if any. If defined, early warning signals are only computed up to this point in the time series

compute_corr(*rolling_window=0.25*)

Compute the (Pearson) correlation matrix over a rolling window. If residuals have not been computed, computation will be performed over state variable.

Put correlation matrices into self.corr

Parameters

rolling_window (*float*) – Length of rolling window used to compute variance. Can be specified as an absolute value or as a proportion of the length of the data being analysed. Default is 0.25.

Return type

None.

compute_covar(*rolling_window=0.25*, *leading_eval=False*)

Compute the covariance matrix over a rolling window. If residuals have not been computed, computation will be performed over state variable.

Put covariance matrices into self.covar Put leading eigenvalue into self.ews

Parameters

- **rolling_window** (*float*) – Length of rolling window used to compute variance. Can be specified as an absolute value or as a proportion of the length of the data being analysed. Default is 0.25.
- **leading_eval** (*bool*) – Whether to compute the leading eigenvalue of the covariance matrix. This has been suggested as an early warning signal (Carpenter et al. 2008, Ecol. Letters)

Return type
None.

detrend(*method='Gaussian'*, *bandwidth=0.2*, *span=0.2*)

Detrend the time series using a chosen method. Add column to the dataframe for 'smoothing' and 'residuals' for each variable

Parameters

- **method** (*str, optional*) – Method of detrending to use. Select from ['Gaussian', 'Lowess'] The default is 'Gaussian'.
- **bandwidth** (*float, optional*) – Bandwidth of Gaussian kernel. Provide as a proportion of data length or as a number of data points. As in the R function ksmooth (used by the earlywarnings package in R), we define the bandwidth such that the kernel has its quartiles at +/- 0.25*bandwidth. The default is 0.2.
- **span** (*float, optional*) – Span of time-series data used for Lowess filtering. Provide as a proportion of data length or as a number of data points. The default is 0.2.

Return type
None.

class ewstools.core.**TimeSeries**(*data*, *transition=None*)

Bases: object

Univariate time series data on which to compute early warning signals.

Parameters

- **data** (*list, numpy.ndarray or pandas.Series*) – 1D list of data that makes up time series. If given as a pandas.Series, then the index, which should represent time values, is carried over.
- **transition** (*float*) – Time value at which transition occurs, if any. If defined, early warning signals are only computed up to this point in the time series.

apply_classifier(*classifier*, *tmin*, *tmax*, *name='c1'*, *verbose=1*)

Apply a deep learning classifier to the residual time series from tmin to tmax. If time series has not been detrended, apply to the raw data. Predictions from the classifier are saved into the attribute dl_preds.

Parameters

- **classifier** (*keras.engine.sequential.Sequential*) – Tensorflow classifier
- **tmin** (*float*) – Earliest time in time series segment (inclusive)
- **tmax** (*float*) – Latest time in time series segment (not inclusive)
- **name** (*str, optional*) – Name assigned to the classifier. The default is 'c1'.
- **verbose** (*int, optional*) – Verbosity of update messages from TensorFlow. 0 = silent, 1 = progress bar, 2 = single line. The default is 1.

Return type
None.

apply_classifier_inc(*classifier*, *inc=10*, *name='c1'*, *verbose=1*)

Apply a deep learning classifier to incrementally increasing time series lengths. First prediction is made on time series segment from data point at index 0 to data point at index inc. Second prediction is made on time series segment from 0 to 2*inc. Third prediction is made on time series segment from 0 to 3*inc. Etc.

Parameters

- **classifier** (*keras.engine.sequential.Sequential*) – TensorFlow classifier.
- **inc** (*int, optional*) – Increment to tmax (the end time of each time series segment) after each classification. The default is 10.
- **name** (*str, optional*) – Name assigned to the classifier. The default is 'c1'.
- **verbose** (*int, optional*) – Verbosity of update messages from TensorFlow. 0 = silent, 1 = progress bar, 2 = single line. The default is 1.

Return type
None.

clear_dl_preds()

Clear the attribute *dl_preds*

Return type
None.

compute_auto(*rolling_window=0.25*, *lag=1*)

Compute autocorrelation over a rolling window. Add to dataframe.

Parameters

- **rolling_window** (*float*) – Length of rolling window used to compute autocorrelation. Can be specified as an absolute value or as a proportion of the length of the data being analysed. Default is 0.25.
- **lag** (*int*) – Lag of autocorrelation

Return type
None.

compute_cv(*rolling_window=0.25*)

Compute coefficient of variation over a rolling window. This is the standard deviation of the residuals divided by the mean of the state variable. If residuals have not been computed, computation will be performed over state variable.

Put into 'ews' dataframe

Parameters
rolling_window (*float*) – Length of rolling window used to compute variance. Can be specified as an absolute value or as a proportion of the length of the data being analysed. Default is 0.25.

Return type
None.

compute_ktau(*tmin='earliest'*, *tmax='latest'*)

Compute kendall tau values of CSD-based EWS. Output is placed in the attribute *ktau*, which is a Python dictionary contianing Kendall tau values for each CSD-based EWS.

Parameters

- **tmin** (*float or 'earliest'*) – Start time for kendall tau computation. If 'earliest', then time is taken as earliest time point in EWS time series.
- **tmax** (*float or 'latest'*) – End time for kendall tau computation. If 'latest', then time is taken as latest time point in EWS time series, which could be the end of the state time series, or a defined transtion point.

compute_kurt(*rolling_window=0.25*)

Compute kurtosis over a rolling window. If residuals have not been computed, computation will be performed over state variable.

Add to dataframe.

Parameters
rolling_window (*float*) – Length of rolling window used to compute variance. Can be specified as an absolute value or as a proportion of the length of the data being analysed. Default is 0.25.

Return type
None.

compute_skew(*rolling_window=0.25*)

Compute skew over a rolling window. If residuals have not been computed, computation will be performed over state variable.

Add to dataframe.

Parameters
rolling_window (*float*) – Length of rolling window used to compute variance. Can be specified as an absolute value or as a proportion of the length of the data being analysed. Default is 0.25.

Return type
None.

compute_smax()

Compute Smax (the maximum power in the power spectrum). This can only be applied after applying compute_spectrum(). Stores Smax values in TimeSeries.ews_spec

Return type
None.

compute_spec_type(*sweep=False*)

Fit the analytical forms of the Fold, Hopf and Null power spectrum to the empirical power spectrum. Get Akaike Information Criterion (AIC) weights to determine best fit. Store AIC weights in TimeSeries.ews_spec Store fitted power spectra in TimeSeries.pspec_fits

Parameters
sweep (*bool*) – If 'True', sweep over a range of intialisation parameters when optimising to compute AIC scores, at the expense of longer computation. If 'False', intialisation parameter is taken as the 'best guess'. The default is False.

Return type
None.

compute_spectrum(*rolling_window=0.25*, *ham_length=40*, *ham_offset=0.5*, *pspec_roll_offset=20*, *w_cutoff=1*)

Compute the power spectrum over a rolling window. Stores the power spectra as a DataFrame in TimeSeries.pspec

Parameters

- **rolling_window** (*float*) – Length of rolling window used to compute variance. Can be specified as an absolute value or as a proportion of the length of the data being analysed. Default is 0.25.
- **ham_length** (*int*) – Length of the Hamming window used to compute the power spectrum. The default is 40.

- **ham_offset** (*float*) – Hamming offset as a proportion of the Hamming window size. The default is 0.5.
- **pspec_roll_offset** (*int*) – Rolling window offset used when computing power spectra. Power spectrum computation is relatively expensive so this is rarely taken as 1 (as is the case for the other EWS). The default is 20.
- **w_cutoff** (*floag*) – Cutoff frequency used in power spectrum. Given as a proportion of the maximum permissable frequency in the empirical power spectrum.

Return type
None.

compute_std(*rolling_window=0.25*)

Compute standard deviation over a rolling window. If residuals have not been computed, computation will be performed over state variable.

Put into 'ews' dataframe

Parameters
rolling_window (*float*) – Length of rolling window used to compute variance. Can be specified as an absolute value or as a proportion of the length of the data being analysed. Default is 0.25.

Return type
None.

compute_var(*rolling_window=0.25*)

Compute variance over a rolling window. If residuals have not been computed, computation will be performed over state variable.

Put into 'ews' dataframe

Parameters
rolling_window (*float*) – Length of rolling window used to compute variance. Can be specified as an absolute value or as a proportion of the length of the data being analysed. Default is 0.25.

Return type
None.

detrend(*method='Gaussian'*, *bandwidth=0.2*, *span=0.2*)

Detrend the time series using a chosen method. Add column to the dataframe for 'smoothing' and 'residuals'

Parameters

- **method** (*str, optional*) – Method of detrending to use. Select from ['Gaussian', 'Lowess'] The default is 'Gaussian'.
- **bandwidth** (*float, optional*) – Bandwidth of Gaussian kernel. Provide as a proportion of data length or as a number of data points. As in the R function ksmooth (used by the earlywarnings package in R), we define the bandwidth such that the kernel has its quartiles at +/- 0.25*bandwidth. The default is 0.2.
- **span** (*float, optional*) – Span of time-series data used for Lowess filtering. Provide as a proportion of data length or as a number of data points. The default is 0.2.

Return type
None.

make_plotly(*kendall_tau=True*, *ens_avg=False*)

Make an interactive Plotly figure to view all EWS computed

Parameters

- **kendall_tau** (*bool, optional*) – Set as true to show Kendall tau values (if they have been computed) Default is True.
- **ens_avg** (*bool, optional*) – Plot the ensenble average of DL predictions. Default is False.

Return type
Plotly figure

ewstools.core.**block_bootstrap**(*series*, *n_samples*, *bs_type='Stationary'*, *block_size=10*)

Computes block-bootstrap samples of series.

Parameters

- **series** (*pd.Series*) – Time-series data in the form of a Pandas Series indexed by time
- **n_samples** (*int*) – Number of bootstrapped samples to output.
- **bs_type** (*{'Stationary', 'Circular'}*) – Type of block-bootstrapping to perform.
- **block_size** (*int*) – Size of resampling blocks. Should be big enough to capture important frequencies in the series.

Returns
DataFrame containing the block-bootstrapped samples of series. Indexed by sample number, then time.

Return type
pd.DataFrame

ewstools.core.**eval_recon_rolling**(*df_in*, *roll_window=0.4*, *roll_offset=1*, *smooth='Lowess'*, *span=0.1*, *band_width=0.2*, *upto='Full'*)

Compute reconstructed eigenvalues from residuals of multi-variate non-stationary time series

Parameters

- **df_in** (*pd.DataFrame*) – Time series data with variables in different columns, indexed by time
- **roll_window** (*float*) – Rolling window size as a proportion of the length of the time series data.
- **roll_offset** (*int*) – Offset of rolling window upon each EWS computation - make larger to save on computational time
- **smooth** (*{'Gaussian', 'Lowess', 'None'}*) – Type of detrending.
- **band_width** (*float*) – Bandwidth of Gaussian kernel. Taken as a proportion of time-series length if in (0,1), otherwise taken as absolute.
- **span** (*float*) – Span of time-series data used for Lowess filtering. Taken as a proportion of time-series length if in (0,1), otherwise taken as absolute.
- **upto** (*int or 'Full'*) – Time up to which EWS are computed. Enter 'Full' to use the entire time-series. Otherwise enter a time value.

Returns
DataFrame indexed by time, with columns of the Jacobian, eigenvalues and eigenvectors at each point in time.

Return type
pd.DataFrame

ewstools.core.**mean_ci**(*data*, *alpha=0.95*)

Compute confidence intervals (to alpha%) of the mean of data. This is performed using bootstrapping.

Parameters

- **data** (*pd.Series*) – Data provided as a Pandas Series
- **alpha** (*float*) – Confidence percentage.

Returns
Dicitonary of mean, lower and upper bound of data

Return type
dict

ewstools.core.**roll_bootstrap**(*raw_series*, *span=0.1*, *roll_window=0.25*, *roll_offset=1*, *upto='Full'*, *n_samples=20*, *bs_type='Stationary'*, *block_size=10*)

Smooths raw_series and computes residuals over a rolling window. Bootstraps each segment and outputs samples.

Parameters

- **raw_series** (*pd.Series*) – Time-series data in the form of a Pandas Seires indexed by time.
- **span** (*float*) – Proportion of data used for Loess filtering.
- **roll_windopw** (*float*) – Size of the rolling window (as a proportion of the length of the data).
- **roll_offset** (*int*) – Number of data points to shift the rolling window upon each iteration (reduce to increase computation time).
- **upto** (*int/'Full'*) – If 'Full', use entire time-series, otherwise input time up to which EWS are to be evaluated.
- **n_samples** (*int*) – Number of bootstrapped samples to output.
- **bs_type** (*{'Stationary', 'Circular'}*) – Type of block-bootstrapping to perform.
- **block_size** (*int*) – Size of resampling blocks. Should be big enough to capture important frequencies in the series.

Returns
DataFrame containing the block-bootstrapped samples at each time in raw_series. Indexed by time in raw_series, then, sample number, then time within the rolling window.

Return type
pd.DataFrame

1.2 ewstools.helpers submodule

The 'helpers' submodule contains low-level functions that are useful in the implementation of the high-level functions.

ewstools.helpers.**aic_weights**(*aic_scores*)

Computes AIC weights, given AIC scores.

Parameters
aic_scores (*np.array*) – An array of AIC scores

Returns
Array of the corresponding AIC weights

Return type
np.array

ewstools.helpers.**compute_autocov**(*df_in*)

Computes the autocovariance (lag-1) matrix of n time series provided in df_in. Using the definition phi_ij = < X_i(t+1) X_j(t) > for each element of the autocovariance matrix phi.

Parameters
df_in (*DataFrame with n columns indexed by time*) –

Returns
autocovariance matrix

Return type
np.array

ewstools.helpers.**eval_recon**(*df_in*)

Constructs estimate of Jacobian matrix from stationary time-series data and outputs the eigenvalues, eigenvectors and jacobian.

Parameters
df_in (*DataFrame with two columns indexed by time*) –

Returns
Consists of 'Eigenvalues': np.array of eigenvalues. 'Eigenvectors': np.array of eigenvectors. 'Jacobian': pd.DataFrame of Jacobian entries.

Return type
dict

ewstools.helpers.**fit_flip**(*pspec*, *init*)

Fit the Flip power spectrum model to pspec and compute AIC score. Uses the package LMFIT for optimisation.

Parameters

- **pspec** (*pd.Series*) – Power spectrum data as a Series indexed by frequency.
- **init** (*list of floats*) – Initial parameter guesses of the form [sigma_init, r_init].

Returns
Form [aic, result] where aic is the AIC score for the model fit, and result is a handle that contains further information on the fit.

Return type
list

ewstools.helpers.**fit_fold**(*pspec*, *init*)

Fit the Fold power spectrum model to pspec and compute AIC score. Uses the package LMFIT for optimisation.

Parameters

- **pspec** (*pd.Series*) – Power spectrum data as a Series indexed by frequency.
- **init** (*list of floats*) – Initial parameter guesses of the form [sigma_init, lambda_init].

Returns
Form [aic, result] where aic is the AIC score for the model fit, and result is a handle that contains further information on the fit.

Return type
list

ewstools.helpers.**fit_hopf**(*pspec*, *init*)

Fit the Hopf power spectrum model to pspec and compute AIC score. Uses the package LMFIT for optimisation.

Parameters

- **pspec** (*pd.Series*) – Power spectrum data as a Series indexed by frequency
- **init** (*list of floats*) – Initial parameter guesses of the form [sigma_init, mu_init, w0_init]

Returns
Form [aic, result] where aic is the AIC score for the model fit, and result is a handle that contains further information on the fit.

Return type
list

ewstools.helpers.**fit_null**(*pspec*, *init*)

Fit the Null power spectrum model to pspec and compute AIC score. Uses the package LMFIT for optimisation.

Parameters

- **pspec** (*pd.Series*) – Power spectrum data as a Series indexed by frequency
- **init** (*list of floats*) – Initial parameter guesses of the form [sigma_init]

Returns
Form [aic, result] where aic is the AIC score for the model fit, and result is a handle that contains further information on the fit.

Return type
list

ewstools.helpers.**psd_flip**(*w*, *sigma*, *r*)

Analytical approximation for the power spectrum prior to a Flip bifurcation

ewstools.helpers.**psd_fold**(*w*, *sigma*, *lam*)

Analytical approximation for the power spectrum prior to a Fold bifurcation

ewstools.helpers.**psd_hopf**(*w*, *sigma*, *mu*, *w0*)

Analytical approximation for the power spectrum prior to a Hopf bifurcation

ewstools.helpers.**psd_null**(*w*, *sigma*)

Power spectrum of white noise (flat).

ewstools.helpers.**pspec_metrics**(*pspec*, *ews=['smax', 'cf', 'aic']*, *aic=['Fold', 'Hopf', 'Null']*, *sweep=False*)

Compute the metrics associated with pspec that can be used as EWS.

Parameters

- **pspec** (*pd.Series*) – Power spectrum as a Series indexed by frequency

- **ews** (*list of {'smax', 'cf', 'aic'}*) – EWS to be computed. Options include peak in the power spectrum ('smax'), coherence factor ('cf'), AIC weights ('aic').
- **aic** (*AIC weights to compute*) –
- **sweep** (*bool*) – If 'True', sweep over a range of intialisation parameters when optimising to compute AIC scores, at the expense of longer computation. If 'False', intialisation parameter is taken as the 'best guess'.

Returns
A dictionary of spectral EWS obtained from pspec

Return type
dict

ewstools.helpers.**pspec_welch**(*yVals*, *dt*, *ham_length=40*, *ham_offset=0.5*, *w_cutoff=1*, *scaling='spectrum'*)

Computes the power spectrum of a time-series using Welch's method.

The time-series is assumed to be stationary and to have equally spaced measurements in time. The power spectrum is computed using Welch's method, which computes the power spectrum over a rolling window of subsets of the time-series and then takes the average.

Parameters

- **yVals** (*array of floats*) – Array of time-series values.
- **dt** (*float*) – Seperation between data points.
- **ham_length** (*int*) – Length of Hamming window (number of data points).
- **ham_offset** (*float*) – Hamming offset as a proportion of the Hamming window size.
- **w_cutoff** (*float*) – Cutoff frequency used in power spectrum. Given as a proportion of the maximum permissable frequency in the empirical power spectrum.
- **scaling** (*{'spectrum', 'density'}*) – Whether to compute the power spectrum ('spectrum') or the power spectral density ('density'). The power spectral density is the power spectrum normalised (such that the area underneath equals one).

Returns
Power values indexed by frequency

Return type
pd.Series

ewstools.helpers.**sflip_init**(*smax*, *stot*)

Compute the 'best guess' initialisation values for sigma and r when fitting sflip to the empirical power spectrum.

Parameters

- **smax** (*float*) – Maximum power in the power spectrum.
- **stot** (*float*) – Total power in the power spectrum.

Returns
List containing the initialisation parameters [sigma, r]

Return type
list of floats

ewstools.helpers.**sfold_init**(*smax*, *stot*)

Compute the 'best guess' initialisation values for sigma and lamda when fitting sfold to the empirical power spectrum.

Parameters

- **smax** (*float*) – Maximum power in the power spectrum.
- **stot** (*float*) – Total power in the power spectrum.

Returns
List containing the initialisation parameters [sigma, lambda]

Return type
list of floats

ewstools.helpers.**shopf_init**(*smax*, *stot*, *wdom*)

Compute the 'best guess' initialisation values for sigma, mu and w0, when fitting sHopf to the empirical power spectrum.

Parameters

- **smax** (*float*) – Maximum power in the power spectrum.
- **stot** (*float*) – Total power in the power spectrum.
- **wdom** (*float*) – Frequency that has the highest power.

Returns
List containing the initialisation parameters [sigma, mu, w0]

Return type
list of floats

ewstools.helpers.**snull_init**(*stot*)

Compute the 'best guess' initialisation values for sigma when fitting snull to the empirical power spectrum.

Parameters
stot (*float*) – Total power in the power spectrum.

Returns
List containing the initialisation parameters [sigma].

Return type
list of floats

1.3 ewstools.models submodule

The 'models' submodule contains functions to run stochastic simulations of various mechanistic models. These can be used as data sources to test early warning signals.

ewstools.models.**simulate_may**(*tmax=500*, *dt=0.01*, *tburn=100*, *r=1*, *k=1*, *s=0.1*, *h=[0.15, 0.27]*, *sigma=0.01*, *x0=0.8*)

Run a numerical simulation of May's harvesting model with additive white noise.

Allows for linearly increasing/decreasing harvesting rate (h)

The model at the default parameter settings has a fold bifurcation at h=0.260

Parameters

- **tmax** (*int, optional*) – Total simulation time. The default is 500.
- **dt** (*float, optional*) – Time increment for each iteration of the Euler Maruyama scheme. The default is 0.01.
- **tburn** (*int, optional*) – Total burn in time to remove transients. The default is 100.

- **r** (*float, optional*) – Intrinsic growth rate The default is 1.
- **k** (*float, optional*) – Carrying capacity The default is 1.
- **s** (*float, optional*) – Half-saturation constant of harvesting function The default is 0.1.
- **h** (*float or list, optional*) – Harvesting rate. Can be provided as a list containing the start and end value if linear change is desired. The default is [0.15,0.27].
- **sigma** (*float, optional*) – Noise amplitude. The default is 0.01.
- **x0** (*float, optional*) – Initial condition. The default is 0.8.

Returns
Trajectories of state variable (x) indexed by time.

Return type
pd.DataFrame

ewstools.models.**simulate_ricker**(*tmax=500*, *tburn=100*, *r=0.75*, *k=10*, *h=0.75*, *F=[0, 2.7]*, *sigma=0.02*, *x0=0.8*)

Run a numerical simulation of the Ricker model with a Holling Type II harvesting term and additive white noise. Allows for linearly increasing/decreasing harvesting rate. Default parameter configuration takes model through a Fold bifurcation.

Model configuration is as in Bury et al. (2020) Roy. Soc. Interface https://royalsocietypublishing.org/doi/full/10.1098/rsif.2020.0482

Parameters

- **tmax** (*int, optional*) – Number of time steps. The default is 500.
- **tburn** (*int, optional*) – Number of time steps to use as a burn in period to remove transients. The default is 100.
- **r** (*float or list, optional*) – Intrinsic growth rate. Can be provided as a list containing the start and end value if linear change is desired. The default is 0.75.
- **k** (*float, optional*) – Population carrying capacity. The default is 10.
- **h** (*float, optional*) – Half-saturation constant of the harvesting expression. The default is 0.75.
- **F** (*float or list, optional*) – Maximum harvesting rate. Can be provided as a list containing the start and end value if linear change is desired. The default is 0.
- **sigma** (*float, optional*) – Noise amplitude. The default is 0.02.
- **x0** (*float, optional*) – Initial condition. The default is 0.8.

Returns
Trajectory indexed by time.

Return type
pd.Series

ewstools.models.**simulate_rosen_mac**(*tmax=500*, *dt=0.01*, *tburn=100*, *r=4*, *k=1.7*, *h=0.15*, *e=0.5*, *m=2*, *a=[12, 16]*, *sigma_x=0.01*, *sigma_y=0.01*, *x0=1*, *y0=0.4*)

Run a numerical simulation of the Rosenzweig-MacArthur model with additive white noise.

Allows for linearly increasing/decreasing attack rate (a).

The model at the default parameter settings has a transcritical bifurcation at a=5.60 and a Hopf bifurcation at a=15.69.

Model configuration is as in Geller et al. (2016), Theoretical Ecology https://link.springer.com/article/10.1007/s12080-016-0303-2.

Parameters

- **tmax** (*int, optional*) – Total simulation time. The default is 500.
- **dt** (*float, optional*) – Time increment for each iteration of the Euler Maruyama scheme. The default is 0.01.
- **tburn** (*int, optional*) – Total burn in time to remove transients. The default is 100.
- **r** (*float, optional*) – Intrinsic growth rate of the resource (x) The default is 4.
- **k** (*float, optional*) – Carrying capacity The default is 1.7.
- **h** (*float, optional*) – Handling time The default is 0.15.
- **e** (*float, optional*) – Conversion factor The default is 0.5.
- **m** (*float, optional*) – Per captia consumer mortality rate. The default is 2.
- **a** (*float or list, optional*) – Attack rate of the consumer (y). Can be provided as a list containing the start and end value if linear change is desired. The default is [12,16].
- **sigma_x** (*float, optional*) – Noise amplitude for the resource (x). The default is 0.01.
- **sigma_y** (*float, optional*) – Noise amplitude for the consumer (y). The default is 0.01.
- **x0** (*float, optional*) – Initial condition for the resource. The default is 1.
- **y0** (*float, optional*) – Initial condition for the consumer. The default is 0.4.

Returns
Trajectories of resource (x) and consumer (y) indexed by time.

Return type
pd.DataFrame

CHAPTER

TWO

INSTALLATION

Install ewstools with pip

```
pip install ewstools
```

CHAPTER

THREE

DEMOS

For tutorials on using *ewstools*: https://github.com/ThomasMBury/ewstools/tree/main/tutorials

CHAPTER

FOUR

SUPPORT

Issues and suggestions may be posted on Github via the issue tracker: https://github.com/ThomasMBury/ewstools/issues

CHAPTER

FIVE

LICENSE

The project is licensed under the MIT license.

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

e

ewstools.core, 3
ewstools.helpers, 10
ewstools.models, 13

INDEX

A

aic_weights() (*in module ewstools.helpers*), 10
apply_classifier() (*ewstools.core.TimeSeries method*), 4
apply_classifier_inc() (*ewstools.core.TimeSeries method*), 4

B

block_bootstrap() (*in module ewstools.core*), 8

C

clear_dl_preds() (*ewstools.core.TimeSeries method*), 5
compute_auto() (*ewstools.core.TimeSeries method*), 5
compute_autocov() (*in module ewstools.helpers*), 10
compute_corr() (*ewstools.core.MultiTimeSeries method*), 3
compute_covar() (*ewstools.core.MultiTimeSeries method*), 3
compute_cv() (*ewstools.core.TimeSeries method*), 5
compute_ktau() (*ewstools.core.TimeSeries method*), 5
compute_kurt() (*ewstools.core.TimeSeries method*), 5
compute_skew() (*ewstools.core.TimeSeries method*), 6
compute_smax() (*ewstools.core.TimeSeries method*), 6
compute_spec_type() (*ewstools.core.TimeSeries method*), 6
compute_spectrum() (*ewstools.core.TimeSeries method*), 6
compute_std() (*ewstools.core.TimeSeries method*), 7
compute_var() (*ewstools.core.TimeSeries method*), 7

D

detrend() (*ewstools.core.MultiTimeSeries method*), 4
detrend() (*ewstools.core.TimeSeries method*), 7

E

eval_recon() (*in module ewstools.helpers*), 10
eval_recon_rolling() (*in module ewstools.core*), 8
ewstools.core
    module, 3
ewstools.helpers
    module, 10
ewstools.models
    module, 13

F

fit_flip() (*in module ewstools.helpers*), 10
fit_fold() (*in module ewstools.helpers*), 10
fit_hopf() (*in module ewstools.helpers*), 11
fit_null() (*in module ewstools.helpers*), 11

M

make_plotly() (*ewstools.core.TimeSeries method*), 7
mean_ci() (*in module ewstools.core*), 9
module
    ewstools.core, 3
    ewstools.helpers, 10
    ewstools.models, 13
MultiTimeSeries (*class in ewstools.core*), 3

P

psd_flip() (*in module ewstools.helpers*), 11
psd_fold() (*in module ewstools.helpers*), 11
psd_hopf() (*in module ewstools.helpers*), 11
psd_null() (*in module ewstools.helpers*), 11
pspec_metrics() (*in module ewstools.helpers*), 11
pspec_welch() (*in module ewstools.helpers*), 12

R

roll_bootstrap() (*in module ewstools.core*), 9

S

sflip_init() (*in module ewstools.helpers*), 12
sfold_init() (*in module ewstools.helpers*), 12
shopf_init() (*in module ewstools.helpers*), 13
simulate_may() (*in module ewstools.models*), 13
simulate_ricker() (*in module ewstools.models*), 14
simulate_rosen_mac() (*in module ewstools.models*), 14
snull_init() (*in module ewstools.helpers*), 13

T

TimeSeries (*class in ewstools.core*), 4