PyTimeVar

Release 0.0.4

Mingxuan Song, Bernhard van der Sluis, Yicong Lin

Sep 15, 2024

CONTENTS:

Add your content using `reStructuredText` syntax. See the reStructuredText documentation for details.

CHAPTER

ONE

PYTIMEVAR

1.1 PyTimeVar package

1.1.1 Subpackages

PyTimeVar.bhpfilter package

Submodules

PyTimeVar.bhpfilter.bHP module

class `PyTimeVar.bhpfilter.bHP.`**`BoostedHP`**(*vY*, *dLambda=1600*, *iMaxIter=100*)

Bases: `object`

Class for performing the boosted HP filter

Parameters

- **vY** (*np.ndarray*) – The dependent variable (response) array.
- **dLambda** (*float*) – The smoothing parameter.
- **iMaxIter** (*int*) – The maximum number of iterations for the boosting algorithm.

vY

The input time series data.

Type
array-like

dLambda

The smoothing parameter.

Type
float

iMaxIter

The maximum number of iterations for the boosting algorithm.

Type
int

results

A tuple containing the results of the Boosted HP filter.

Type
tuple

`dAlpha`

The significance level for the stopping criterion ‘adf’.

Type
float

`stop`

Stopping criterion (‘adf’, ‘bic’, ‘aic’, ‘hq’).

Type
string

`results`

Contains the trends per iteration, the current residuals, the information criteria values, the number of iterations, and the estimated trend.

Type
tuple

`fit()`

Fits the Boosted HP filter to the data.

`summary()`

Prints a summary of the results.

`plot()`

Plot true data against estimated trend.

`fit`(*boost=True*, *stop='adf'*, *dAlpha=0.05*, *verbose=False*)

Fits the Boosted HP filter to the data.

Parameters

- **boost** (*bool*) – if True, boosting is used.
- **stop** (*str*) – Stopping criterion (‘adf’, ‘bic’, ‘aic’, ‘hq’).
- **dAlpha** (*float*) – The significance level for the stopping criterion ‘adf’.
- **verbose** (*bool*) – Whether to display a progress bar.

Returns

- **vbHP** (*np.ndarray*) – The estimated trend.
- **vCurrentRes** (*np.ndarray*) – The residuals.

`plot()`

Plots the true data against estimated trend

`summary()`

Prints a summary of the results.

Module contents

PyTimeVar.datasets package

Subpackages

PyTimeVar.datasets.co2 package

Submodules

PyTimeVar.datasets.co2.data module

`PyTimeVar.datasets.co2.data.`**`load`**(*start_date=None*, *end_date=None*, *regions=None*)

Load the CO2 emissions dataset and optionally filter by date range and/or countries. This dataset contains the emissions data from 1900 to 2017, for a range of countries.

Parameters

- **start_date** (*str, optional*) – The start_year to filter the data. Format 'YYYY'. Minimum start year is 1900.
- **end_date** (*str, optional*) – The end_year to filter the data. Format 'YYYY'.
- **regions** (*list, optional*) – Regions to be selected from data.

Returns

DataFrame containing the filtered data with columns 'Date' and regions.

Return type

pandas.DataFrame

> **Warning**
>
> Prints warnings if any provided regions are not found in the dataset. Prints warnings if the start_year is earlier than the minimum year in the data or the end_year is later than the maximum year in the data.

Module contents

PyTimeVar.datasets.gold package

Submodules

PyTimeVar.datasets.gold.data module

`PyTimeVar.datasets.gold.data.`**`load`**(*currencies=None*, *start_date=None*, *end_date=None*)

Load the gold dataset and optionally filter by specific currencies and date range.

Parameters

- **currencies** (*list, optional*) – List of currency codes to filter the dataset by. Available options are: ['USD', 'EUR', 'JPY', 'GBP', 'CAD', 'CHF', 'INR', 'CNY', 'TRY', 'SAR',

  'IDR', 'AED', 'THB', 'VND', 'EGP', 'KRW', 'RUB', 'ZAR', 'AUD']

- **start_date** (*str, optional*) – The start date to filter the data. Format 'YYYY-MM-DD'.
- **end_date** (*str, optional*) – The end date to filter the data. Format 'YYYY-MM-DD'.

Returns
DataFrame containing the filtered data.

Return type
pandas.DataFrame

> **Warning**
>
> Prints warnings if any provided currencies are not found in the dataset. Prints warnings if the start_date is earlier than the minimum date in the data or the end_date is later than the maximum date in the data.

Module contents

PyTimeVar.datasets.herding package

Submodules

PyTimeVar.datasets.herding.data module

`PyTimeVar.datasets.herding.data.load`(*start_date=None*, *end_date=None*, *data_replication=False*)

Load the Herding dataset and optionally filter by date range. This dataset contains the herding data from Jan 5 2015 to Apr 29 2022.

Parameters

- **start_date** (*str, optional*) – The start_year to filter the data. Format 'YYYY-MM-DD'. Minimum start year is 1900.
- **end_date** (*str, optional*) – The end_year to filter the data. Format 'YYYY'.

Returns
DataFrame containing the filtered data with columns 'Date' and 'Herding'.

Return type
pandas.DataFrame

> **Warning**
>
> Prints warnings if any provided regions are not found in the dataset. Prints warnings if the start_year is earlier than the minimum year in the data or the end_year is later than the maximum year in the data.

Module contents

PyTimeVar.datasets.temperature package

Submodules

PyTimeVar.datasets.temperature.data module

`PyTimeVar.datasets.temperature.data.load`(*regions=None*, *start_date=None*, *end_date=None*)

Load the temperature dataset and optionally filter by specific regions and date range. This dataset contains the average yearly temperature change in degrees Celsius for different regions of the world from 1961 to 2023. :param regions: List of regions to filter the dataset by. Available options are:

['World', 'Africa', 'Asia', 'Europe', 'North America', 'Oceania', 'South America']

Parameters

- **start_date** (*str, optional*) – The start date to filter the data. Format 'YYYY-MM-DD'.
- **end_date** (*str, optional*) – The end date to filter the data. Format 'YYYY-MM-DD'.

Returns

DataFrame containing the filtered data.

Return type

pandas.DataFrame

> **Warning**
>
> Prints warnings if any provided regions are not found in the dataset. Prints warnings if the start_year is earlier than the minimum year in the data or the end_year is later than the maximum year in the data.

Module contents

PyTimeVar.datasets.usd package

Submodules

PyTimeVar.datasets.usd.data module

`PyTimeVar.datasets.usd.data.load`(*type='Open'*, *start_date=None*, *end_date=None*)

Load the USD index dataset and optionally filter by date range.

Parameters

- **type** (*str, optional*) – The type of data to load. Available options are: ['Open', 'High', 'Low', 'Close']
- **start_date** (*str, optional*) – The start date to filter the data. Format 'YYYY-MM-DD'.
- **end_date** (*str, optional*) – The end date to filter the data. Format 'YYYY-MM-DD'.

Returns
DataFrame containing the filtered data.

Return type
pandas.DataFrame

> ⚠ **Warning**
>
> Prints warnings if any provided currencies are not found in the dataset. Prints warnings if the start_date is earlier than the minimum date in the data or the end_date is later than the maximum date in the data.

Module contents

Submodules

PyTimeVar.datasets.utils module

`PyTimeVar.datasets.utils.`**`load_csv`**`(`*base_file*, *csv_name*, *sep=','*, *convert_float=False*`)`

Standard simple csv loader

Module contents

Dataset module for PyTimeVar package.

PyTimeVar.gas package

Submodules

PyTimeVar.gas.GAS module

class `PyTimeVar.gas.GAS.`**`GAS`**`(`*vY: ndarray*, *mX: ndarray*, *method: str = 'none'*, *vgamma0: ndarray | None = None*, *bounds: list | None = None*, *options: dict | None = None*, *maxiter: int = 5*`)`

Bases: `object`

Class for performing score-driven (GAS) filtering.

Parameters

- **vY** (*np.ndarray*) – The dependent variable (response) array.
- **mX** (*np.ndarray*) – The independent variable (predictor) matrix.
- **method** (*string*) – Method to estimate GAS model.
- **vgamma0** (*np.ndarray*) – Initial parameter vector.
- **bounds** (*list*) – List to define parameter space.
- **options** (*dict*) – Stopping criteria for optimization.
- **maxiter** (*int*) – Maximum number of repitions of the optimization algorithm. If not provided, default is set to five repitions with different initial parameters.

`vY`

The dependent variable (response) array.

Type
np.ndarray

`mX`

The independent variable (predictor) matrix.

Type
np.ndarray

`n`

The length of vY.

Type
int

`n_est`

The number of coefficients.

Type
int

`method`

Method to estimate GAS model.

Type
string

`vgamma0`

The initial parameter vector.

Type
np.ndarray

`maxiter`

Maximum number of repitions of the optimization algorithm.

Type
int

`bounds`

List to define parameter space.

Type
list

`options`

Stopping criteria for optimization.

Type
dict

`success`

If True, optimization was successful.

Type
bool

`betas`
The estimated coefficients.

Type
np.ndarray

`params`
The estimated GAS parameters.

Type
np.ndarray

`fit()`
Fit score-driven model, according to the specified method ('gaussian' or 'student')

`plot()`
Plot estimated coefficients against true data

`construct_likelihood`(*vbeta0*, *vpara*)
Calculated the log-likelihood value, according to the specified self.method.

Parameters

- **vbeta0** (*np.ndarray*) – Initial coefficients at time zero.
- **vpara** (*np.ndarray*) – Array of GAS parameters.

Returns
lhVal – Log-likelihood value.

Return type
float

`fit()`
Fit score-driven model, according to the specified method ('gaussian' or 'student')

Returns

- **mBetaHat** (*np.ndarray*) – The estimated coefficients.
- **vparaHat** (*np.ndarray*) – The estimated GAS parameters.

`plot()`
Plot the beta coefficients over a normalized x-axis from 0 to 1.

Module contents

PyTimeVar.kalman package

Submodules

PyTimeVar.kalman.kalman module

`class PyTimeVar.kalman.kalman.Kalman`(*vY: ndarray | None = None*, *T: ndarray | None = None*, *R: ndarray | None = None*, *Q: ndarray | None = None*, *sigma_u: float | None = None*, *b_1: ndarray | None = None*, *P_1: ndarray | None = None*, *mX: ndarray | None = None*)

Bases: `object`

Class for performing Kalman filtering and smoothing.

Parameters

- **vY** (*np.ndarray*) – The dependent variable (response) array.
- **T** (*np.ndarray, optional*) – The transition matrix of the state space model.
- **R** (*np.ndarray, optional*) – The transition correlation matrix of the state space model.
- **Q** (*np.ndarray, optional*) – The transition covariance matrix of the state space model.
- **sigma_u** (*np.ndarray, optional*) – The observation noise variance of the state space model.
- **b_1** (*np.ndarray, optional*) – The initial mean of the state space model.
- **P_1** (*np.ndarray, optional*) – The initial covariance matrix of the state space model.
- **mX** (*np.ndarray, optional*) – The regressors to use in the model. If provided, the model will be a linear regression model.

vY

The dependent variable (response) array.

Type
np.ndarray

n

The length of vY.

Type
int

isReg

If True, regressors are provided by the user.

Type
bool

T

The transition matrix of the state space model.

Type
np.ndarray

Z

Auxiliary (1,1)-vector of a scalar 1. This is used in case there are no regressors.

Type
np.ndarray

R

The transition correlation matrix of the state space model.

Type
np.ndarray

`Q`

The transition covariance matrix of the state space model.

Type
np.ndarray

`H`

The observation noise variance of the state space model.

Type
np.ndarray

`a_1`

The initial mean of the state space model.

Type
np.ndarray, optional

`P_1`

The initial covariance matrix of the state space model.

Type
np.ndarray

`mX`

The regressors to use in the model. If provided, the model will be a linear regression model.

Type
np.ndarray

`Z_reg`

The regressors matrix in correct format to use in filtering.

Type
np.ndarray

`p_dim`

The number of coefficients.

Type
int

`m_dim`

The number of response variables. This is always 1.

Type
int

`filt`

The filtered coefficients.

Type
np.ndarray

`smooth`

The smoothed coefficients.

`fit()`

Fit state-space model by Kalman filter or smoother, according to the specified option ('filter' or 'smoother') `summary()`

Print a summary of the Kalman filter/smoother specifications, including the values of H, Q, R and T

`plot()`

Plot filtered/smoothed estimates against true data

`compute_likelihood_LL(vTheta)`

Computes the negative log-likelihood value for a given parameter vector.

Parameters
`vTheta` (*np.ndarray*) – The parameter vector.

Returns
The negative log-likelihood value.

Return type
float

`filter()`

Performs the filtering step of the Kalman filter.

Returns
The filtered state means at each time step.

Return type
np.ndarray

`fit(option)`

Fits the Kalman filter or smoother to the data.

Parameters
`option` (*string*) – Denotes the fitted trend: filter or smoother.

Raises
`ValueError` – No valid option is provided.

Returns
Estimated trend.

Return type
np.ndarray

`plot()`

Plot the beta coefficients over a normalized x-axis from 0 to 1 or over a date range.

`smoother()`

Performs the smoothing step of the Kalman filter.

Returns
The smoothed state means at each time step.

Return type
np.ndarray

`summary()`

Prints a summary of the state-space model specification.

Module contents

PyTimeVar.locallinear package

Submodules

PyTimeVar.locallinear.LLR module

class PyTimeVar.locallinear.LLR.**LocalLinear**(*vY: ndarray*, *mX: ndarray*, *h: float = 0*, *bw_selection: str | None = None*, *tau: ndarray | None = None*, *kernel: str = 'epanechnikov'*, *LB_bw: float | None = None*, *UB_bw: float | None = None*)

Bases: `object`

Class for performing local linear regression (LLR).

Local linear regression is a non-parametric regression method that fits linear models to localized subsets of the data to form a regression function. The code is based on the code provided by Lin et al. [1].

Parameters

- **vY** (*np.ndarray*) – The dependent variable (response) array.
- **mX** (*np.ndarray*) – The independent variable (predictor) matrix.
- **h** (*float*) – The bandwidth parameter controlling the size of the local neighborhood. If not provided, it is estimated as the average of all methods in the package.
- **bw_selection** (*string*) – The name of the bandwidth selection method to be used. Choice between 'aic', 'gcv', 'lmcv-l' with l=0,2,4,6,etc., or 'all'. If not provided, it is set to 'all'.
- **tau** (*np.ndarray*) – The array of points at which predictions are made. If not provided, it is set to a linear space between 0 and 1 with the same length as *vY*.
- **kernel** (*string*) – The name of the kernel function used for estimation. If not provided, it is set to 'epanechnikov' for the Epanechnikov kernel.
- **LB_bw** (*float*) – The lower bound for the bandwidth selection. If not provided, it is set to 0.06.
- **UB_bw** (*float*) – The upper bound for the bandwidth selection. If not provided, it is set to 0.2 for LMCV-l and to 0.7 for AIC and GCV.

vY

The response variable array.

Type
np.ndarray

mX

The predictor matrix.

Type
np.ndarray

n

The length of vY.

Type
int

`times`
: Linearly spaced time points for the local regression.

Type
np.ndarray

`tau`
: Points at which the regression is evaluated.

Type
np.ndarray

`n_est`
: The number of coefficients.

Type
int

`kernel`
: The name of the kernel function.

Type
string

`bw_selection`
: The name of the bandwidth selection.

Type
string

`lmcv_type`
: If LMCV is used for bandwidth selection, this attribute denotes the l in leave-2*l+1-out.

Type
float

`dict_bw`
: The dictionary that contains the optimnal bandwidth values for each individual method.

Type
dict

`h`
: The bandwidth used for local linear regression.

Type
float

`betahat`
: The estimated coefficients.

Type
np.ndarray

`predicted_y`
: The fitted values for the response variable.

Type
np.ndarray

`residuals`

The residuals resulting from the local linear regression.

Type

np.ndarray

`fit()`

Fit the model by local linear estimation and return estimated coefficients.

`summary()`

Print a summary of local linear regression results, including bandwidth, number of observations, and beta coefficients.

`plot_betas()`

Plot estimated coefficients curve over a normalized x-axis from 0 to 1.

`plot_predicted()`

Plot true data against fitted values.

`plot_residuals()`

Plot residuals.

`confidence_bands()`

Construct and plot bootstrap confidence intervals/bands for each coefficient curve.

Notes

The local linear regression is computed at each point specified in *tau*. The bandwidth *h* controls the degree of smoothing.

References

[1] Lin Y, Song M, van der Sluis B (2024),

Bootstrap inference for linear time-varying coefficient models in locally stationary time series, Journal of Computational and Graphical Statistics, Forthcoming.

`ABS_value`(*qtau*, *diff*, *tau*, *alpha*, *B*)

Calculate the absolute value for quantiles.

Parameters

- **qtau** (*np.ndarray*) – Array of quantiles.
- **diff** (*np.ndarray*) – Difference array.
- **tau** (*np.ndarray*) – Array of tau values.
- **alpha** (*float*) – The significance level.
- **B** (*int*) – The number of bootstrap samples.

Returns

Absolute value for quantiles.

Return type

float

AICmodx(*h*)

Computes the AIC value for a given mean squared error and trace.

Parameters
: **h** (*float*) – The bandwidth parameter.

Returns
: AIC value.

Return type
: float

AR(*zhat*, *T*, *ic=None*)

Estimate the AR model and compute residuals.

Parameters

- **zhat** (*np.ndarray*) – Array of predicted values.
- **T** (*int*) – Number of observations.
- **ic** (*string*) – Information criterion to select number of lags. Default criterion is AIC

Returns

- **epsilonhat** (*np.ndarray*) – Array of residuals.
- **max_lag** (*int*) – Maximum lag order.
- **armodel** (*AutoReg*) – Fitted autoregressive model.

AW_BT(*zhat: ndarray*, *mX: ndarray*, *betatilde: ndarray*, *T: int*, *h: float*, *gamma: float*)

Autoregressive Wild Bootstrap Compute a bootstrap sample using the autoregressive wild bootstrap.

Parameters

- **zhat** (*np.ndarray*) – Array of residuals.
- **mX** (*np.ndarray*) – Array of exogenous variables.
- **betatilde** (*np.ndarray*) – Array of estimated coefficients.
- **T** (*int*) – Total number of observations.
- **h** (*float*) – The bandwidth parameter.
- **gamma** (*float*) – The AR(1) coefficient and the standard deviation of the wild component in the AWB.

Returns
: **vYstar** – Transformed response variable.

Return type
: np.ndarray

GCVmodx(*h*)

Computes the GCV value for a bandwidth value.

Parameters
: **h** (*float*) – The bandwidth parameter.

Returns
: GCV value.

Return type
: float

`K_ZW_Q`(*vTi_t*, *h*)

Kernel function for Multiplier Bootstrap quantile.

Parameters

- **vTi_t** (*np.ndarray*) – The array of time indices.
- **h** (*float*) – The bandwidth parameter.

Returns

Array of kernel values.

Return type

np.ndarray

`LBW_BT`(*zhat: ndarray*, *mX: ndarray*, *betatilde: ndarray*, *T: int*, *h: float*, *gamma: float*)

Local Blockwise Wild Bootstrap Performs the local blockwise wild bootstrap algorithm to generate a bootstrap sample.

Parameters

- **zhat** (*np.ndarray*) – The input array of shape (T, 1) containing the original data.
- **mX** (*np.ndarray*) – The input array of shape (T, k) containing the covariates.
- **betatilde** (*np.ndarray*) – The input array of shape (k,) or (k, 1) containing the estimated coefficients.
- **T** (*int*) – The number of observations.
- **h** (*float*) – The bandwidth parameter.
- **gamma** (*float*) – The AR(1) coefficient and the standard deviation of the wild component in the AWB.

Returns

vYstar – The bootstrap sample generated by the LBW_BT algorithm.

Return type

np.ndarray

`MC_ZW`(*alpha*, *h*, *vY*, *mX*, *T*, *B*)

Perform the Multiplier Bootstrap bootstrap algorithm. Zhou, Z., & Wu, W. B. (2010). Simultaneous inference of linear models with time varying coefficients. Journal of the Royal Statistical Society. Series B (Statistical Methodology), 72(4), 513–531. http://www.jstor.org/stable/40802223

Parameters

- **alpha** – The significance level for the bootstrap.
- **h** (*float*) – Bandwidth parameter.
- **vY** (*np.ndarray*) – Array of dependent variable values.
- **mX** (*np.ndarray*) – Array of covariates.
- **T** (*int*) – Number of observations.
- **B** (*int*) – Number of bootstrap iterations.

Returns

Lower and upper confidence bands and beta coefficients.

Return type

tuple

`SW_BT`(*epsilontilde: ndarray*, *max_lag: int*, *armodel*, *mX: ndarray*, *betatilde: ndarray*, *T: int*)

Sieve Wild Bootstrap Calculate the transformed response variable using the local linear regression model.

Parameters

- **epsilontilde** (*np.ndarray*) – Array of residuals.
- **max_lag** (*int*) – Maximum lag order.
- **armodel** (*AutoReg*) – Fitted autoregressive model.
- **mX** (*np.ndarray*) – Array of exogenous variables.
- **betatilde** (*np.ndarray*) – Array of estimated coefficients.
- **T** (*int*) – Total number of observations.

Returns

vYstar – Transformed response variable.

Return type

np.ndarray

`S_BT`(*epsilontilde: ndarray*, *max_lag: int*, *armodel*, *mX: ndarray*, *betatilde: ndarray*, *T: int*)

Sieve Bootstrap Calculate the transformed response variable using the local linear regression model.

Parameters

- **epsilontilde** (*np.ndarray*) – Array of residuals.
- **max_lag** (*int*) – Maximum lag order for the autoregressive model.
- **armodel** (*AutoReg*) – Fitted autoregressive model.
- **mX** (*np.ndarray*) – Array of exogenous variables.
- **betatilde** (*np.ndarray*) – Array of estimated coefficients.
- **T** (*int*) – Total number of observations.

Returns

vYstar – Transformed response variable.

Return type

np.ndarray

`W_BT`(*zhat: ndarray*, *mX: ndarray*, *betatilde: ndarray*, *T: int*, *h: float*, *gamma: float*)

Wild Bootstrap Calculate the transformed response variable using the local linear regression model.

Parameters

- **zhat** (*np.ndarray*) – Array of residuals.
- **mX** (*np.ndarray*) – Array of exogenous variables.
- **betatilde** (*np.ndarray*) – Array of estimated coefficients.
- **T** (*int*) – Total number of observations.
- **h** (*float*) – The bandwidth parameter.
- **gamma** (*float*) – The AR(1) coefficient and the standard deviation of the wild component in the AWB.

Returns

vYstar – Transformed response variable.

Return type
np.ndarray

`bandwidth_selection`(*LB_bw*, *UB_bw*)

Calculate the optimal bandwidth for the local linear regression model using LMCV, AIC and GCV.

Parameters

- **LB_bw** (*float*) – The lower bound for the bandwidth selection.
- **UB_bw** (*float*) – The upper bound for the bandwidth selection

Returns
The optimal bandwidths for each individual method.

Return type
dict

`confidence_bands`(*bootstrap_type: str = 'LBWB'*, *alpha: float | None = None*, *gamma: float | None = None*, *ic: str | None = None*, *Gsubs=None*, *Chtilde: float = 2*, *B: float = 1299*, *plots: bool = False*)

Parameters

- **bootstraptype** (*str*) – Type of bootstrap to use ('SB', 'WB', 'SWB', 'MB', 'LBWB, 'AWB').
- **alpha** (*float*) – Significance level for quantiles.
- **gamma** (*float*) – Parameter value for Autoregressive Wild Bootstrap.
- **ic** (*str*) – Type of information criterion to use for Sieve and Sieve Wild Bootstrap. Possible values are: 'aic', 'hqic', 'bic'
- **Gsubs** (*list of tuples*) – List of sub-ranges for G. Each sub-range is a tuple (start_index, end_index). Default is None, which uses the full range (0, T).
- **Chtilde** (*float*) – Multiplication constant to determine size of oversmoothing bandwidth htilde. Default is 2, if none or negative is specified.
- **B** (*int*) – The number of bootstrap samples. Deafult is 1299, if not provided by the user.
- **plots** (*bool*) – If True, plots are shown of the estimated coefficients and corresponding confidence bands.

Returns

- **S_LB** (*np.ndarray*) – The lower simultaneous confidence bands.
- **S_UB** (*np.ndarray*) – The upper simultaneous confidence bands.
- **P_LB** (*np.ndarray*) – The lower pointwise confidence intervals.
- **P_UB** (*np.ndarray*) – The upper pointwise confidence intervals.

`construct_confidence_bands`(*bootstraptype: str*, *alpha: float | None = None*, *gamma: float | None = None*, *ic: str | None = None*, *Gsubs: list | None = None*, *Chtilde: float | None = None*, *B: float = 1299*)

Construct confidence bands using bootstrap methods.

Parameters

- **bootstraptype** (*str*) – Type of bootstrap to use ('SB', 'WB', 'SWB', 'MB', 'LBWB, 'AWB').
- **alpha** (*float*) – Significance level for quantiles.

- **gamma** (*float*) – Parameter value for Autoregressive Wild Bootstrap.
- **ic** (*str*) – Type of information criterion to use for Sieve and Sieve Wild Bootstrap. Possible values are: 'aic', 'hqic', 'bic'
- **Gsubs** (*list of tuples*) – List of sub-ranges for G. Each sub-range is a tuple (start_index, end_index). Default is None, which uses the full range (0, T).
- **Chtilde** (*float*) – Multiplication constant to determine size of oversmoothing bandwidth htilde. Default is 2, if none or negative is specified.
- **B** (*int*) – The number of bootstrap samples. Deafult is 1299, if not provided by the user.

Returns
Each tuple contains simultaneous and pointwise lower and upper bands for each sub-range, and beta coefficients for each sub-range.

Return type
list of tuples

`fit()`

Fits the local linear regression model to the data.

Returns
self.betahat – The estimated coefficients.

Return type
np.ndarray

`getDelta`(*mX*, *vEps*, *iM*)

Calculate delta values for Multiplier Bootstrap samples.

Parameters

- **mX** (*np.ndarray*) – Array of covariates.
- **vEps** (*np.ndarray*) – Array of residuals.
- **iM** (*int*) – Number of blocks.

Returns
Array of delta values.

Return type
np.ndarray

`getLambda`(*mmDelta*, *iN*, *iM*, *taut*, *dT*, *dTau*, *dGamma*)

Calculate lambda values for Multiplier Bootstrap samples.

Parameters

- **mmDelta** (*np.ndarray*) – Array of delta values.
- **iN** (*int*) – Number of observations.
- **iM** (*int*) – Number of blocks.
- **taut** (*np.ndarray*) – Array of tau values.
- **dT** (*float*) – Time parameter.
- **dTau** (*float*) – Tau parameter.
- **dGamma** (*float*) – Gamma parameter.

Returns
Array of lambda values.

Return type
np.ndarray

`getM`(*mX*, *T*, *dT*, *h*)

Calculate M values for Multiplier Bootstrap samples.

Parameters

- **mX** (*np.ndarray*) – Array of covariates.
- **T** (*int*) – Number of observations.
- **dT** (*float*) – Time parameter.
- **h** (*float*) – Bandwidth parameter.

Returns
Array of M values.

Return type
np.ndarray

`getQ`(*B*, *T*, *taut*, *h*, *alpha*)

Calculate quantile for Multiplier Bootstrap samples.

Parameters

- **B** (*int*) – The number of bootstrap samples.
- **T** (*int*) – the number of observations.
- **taut** (*np.ndarray*) – The points at which the regression is evaluated.
- **h** (*float*) – Bandwidth parameter.
- **alpha** (*float*) – Alpha level for quantiles.

Returns
dQ – Quantile value.

Return type
float

`getSigma`(*mM*, *mLam*)

Calculate sigma values for Multiplier Bootstrap samples.

Parameters

- **mM** (*np.ndarray*) – Array of M values.
- **mLam** (*np.ndarray*) – Array of lambda values.

Returns
Array of sigma values.

Return type
np.ndarray

`min_alphap`(*diff*, *tau*, *alpha*, *B*)

Calculate the minimum alpha value for confidence bands.

Parameters

- **diff** (*np.ndarray*) – Difference array.
- **tau** (*np.ndarray*) – Array of tau values.
- **alpha** (*float*) – The significance level.
- **B** (*int*) – The number of bootstrap samples.

Returns
Minimum alpha value.

Return type
float

`plot_betas()`

Plot the beta coefficients over a normalized x-axis from 0 to 1.

`plot_predicted()`

Plot the actual values of Y against the predicted values of Y over a normalized x-axis from 0 to 1.

`plot_residuals()`

Plot the residuals over a normalized x-axis from 0 to 1.

Returns
self.residuals – Array of residuals.

Return type
np.ndarray

`summary()`

Print a summary of the regression results.

Module contents

PyTimeVar.powerlaw package

Submodules

PyTimeVar.powerlaw.pwr module

`class PyTimeVar.powerlaw.pwr.PowerLaw`(*vY: ndarray*, *n_powers: float | None = None*, *vgamma0: ndarray | None = None*, *options: dict | None = None*)

Bases: `object`

Class for implementing the Power-Law method.

Parameters

- **vY** (*np.ndarray*) – The dependent variable (response) array.
- **n_powers** (*int*) – The number of powers.
- **vgamma0** (*np.ndarray*) – The initial parameter vector.
- **options** (*dict*) – Stopping criteria for optimization.

`vY`

The dependent variable (response) array.

Type
np.ndarray

`n`

The length of vY.

Type
int

`p`

The number of powers. Default is set to 2.

Type
int

`vgamma0`

The initial parameter vector.

Type
np.ndarray

`bounds`

List to define parameter space.

Type
list

`cons`

Dictionary to define constraints.

Type
dict

`trendHat`

The estimated trend.

Type
np.ndarray

`gammaHat`

The estimated power parameters.

Type
np.ndarray

`coeffHat`

The estimated coefficients.

Type
np.ndarray

`fit()`

Fit by power-law model and return trend estimates and parameter estimates .

`summary()`

Print fitted prediction equation.

`plot()`

Plot true data against estimated trend

`fit()`

Returns

- **self.trendHat** (*np.ndarray*) – The estimated trend.
- **self.gammaHat** (*np.ndarray*) – The estimated power parameters.

`plot()`

Plots the original series and the trend component.

`summary()`

Print the mathematical equation for the fitted model

Module contents

1.1.2 Module contents

PYTHON MODULE INDEX

p

INDEX

I

K

L

M

N

O

P

Q

R

S

T

V

W

Z