PyTimeVar

Release 1.1.0

Mingxuan Song, Bernhard van der Sluis, Yicong Lin

Jul 21, 2025

CONTENTS:

The PyTimeVar package offers state-of-the-art estimation and statistical inference methods for time series regression models with flexible trends and/or time-varying coefficients.

The Overview section provides details on the installation and usage of the package.

CHAPTER

ONE

README FILE

CHAPTER

TWO

PYTIMEVAR: A PYTHON PACKAGE FOR TRENDING TIME-VARYING TIME SERIES MODELS

Authors: Mingxuan Song (m3.song@student.vu.nl, Vrije Universiteit Amsterdam), Bernhard van der Sluis (vandersluis@ese.eur.nl, Erasmus Universiteit Rotterdam), and Yicong Lin (yc.lin@vu.nl, Vrije Universiteit Amsterdam & Tinbergen Institute)

Discussion paper titled "PyTimeVar: A Python Package for Trending Time-Varying Time Series Models" is available at https://tinbergen.nl/discussion-paper/6365/24-060-iii-pytimevar-a-python-package-for-trending-time-varying-time-series-models.

2.1 Purpose of the package

The PyTimeVar package offers state-of-the-art estimation and statistical inference methods for time series regression models with flexible trends and/or time-varying coefficients. The package implements nonparametric estimation along with multiple recently proposed bootstrap-assisted inference methods. Pointwise confidence intervals and simultaneous bands of parameter curves via bootstrap can be easily obtained using user-friendly commands. The package also includes four commonly used methods for modeling trends and time-varying relationships: boosted Hodrick-Prescot filter, power-law trend models, state-space models, and score-driven models. This allows users to compare different approaches within a unified environment.

The package is built upon several papers and books. We list the key references below.

2.1.1 Local linear kernel estimation and bootstrap inference

Friedrich and Lin (2024) (doi: https://doi.org/10.1016/j.jeconom.2022.09.004); Lin et al. (2025) (doi: https://doi.org/10.1080/10618600.2024.2403705); Friedrich et al. (2020) (doi: https://doi.org/10.1016/j.jeconom.2019.05.006); Smeekes and Urbain (2014) (doi: https://doi.org/10.26481/umagsb.2014008) Zhou and Wu (2010) (doi: https://doi.org/10.1111/j.1467-9868.2010.00743.x); Buhlmann (1998) (doi: https://doi.org/10.1214/aos/1030563978);

2.1.2 Boosted HP filter

Mei et al. (2024) (doi: https://doi.org/10.1002/jae.3086); Biswas et al. (2024) (doi: https://doi.org/10.1080/07474938.2024.2380704); Phillips and Shi (2021) (doi: https://doi.org/10.1111/iere.12495);

2.1.3 Power-law trend models

Lin and Reuvers (2025) (doi: https://doi.org/10.1111/jtsa.12805); Robinson (2012) (doi: https://doi.org/10.3150/10-BEJ349);

2.1.4 State-space models

Durbin and Koopman (2012) (doi: https://doi.org/10.1093/acprof:oso/9780199641178.001.0001);

2.1.5 Score-driven models

Creal et al. (2013) (doi: https://doi.org/10.1002/jae.1279); Harvey (2013) (doi: https://doi.org/10.1017/CBO9781139540933); Harvey and Luati (2014) (doi: https://doi.org/10.1080/01621459.2014.887011) Blasques et al. (2016) (doi: https://doi.org/10.1016/j.ijforecast.2015.11.018);

2.2 Features

- Nonparametric estimation of time-varying time series models, along with various bootstrap-assisted methods for inference, including local blockwise wild bootstrap, wild bootstrap, sieve bootstrap, sieve wild bootstrap, autoregressive wild bootstrap.
- Alternative estimation methods for modeling trend and time-varying relationships, including boosted HP filter, power-law trend models, state-space, and score-driven models. The package includes inference methods for power-law trend models, state-space models, and score-driven models.
- Unified framework for comparison of methods.
- Multiple datasets for illustration.

2.3 Getting started

The PyTimeVar can implemented as a PyPI package. To download the package in your Python environment, use the following command:

```
pip install PyTimeVar
```

2.4 Support

The documentation of the package can be found at the GitHub repository https://github.com/bpvand/PyTimeVar, and ReadTheDocs https://pytimevar.readthedocs.io/en/latest/.

For any questions or feedback regarding the PyTimeVar package, please feel free to contact the authors via email: m3.song@student.vu.nl; vandersluis@ese.eur.nl; yc.lin@vu.nl.

CHAPTER

THREE

PYTIMEVAR

3.1 PyTimeVar package

3.1.1 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`(*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*) – If True, a progress bar is displayed.

Returns

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

`plot`(*tau=None*)

Plots the true data against estimated trend

Parameters
tau (*list, optional*) – The list looks the following: tau = [start,end]. The function will plot all data and estimates between start and end.

Raises
ValueError – No valid tau is provided.

`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'. Maximum end year is 2017.
- **`regions`** (*`list, optional`*) – Regions to be selected from data. Available options are:

  AUSTRALIA, AUSTRIA, BELGIUM, CANADA, DENMARK, FINLAND, FRANCE, GERMANY, ITALY, JAPAN, NETHERLANDS, NEW ZEALAND, NORWAY, PORTUGAL, SPAIN, SWEDEN, SWITZERLAND, UNITED KINGDOM, UNITED STATES, CHILE, SRI LANKA, URUGUAY, BRAZIL, GREECE, PERU, VENEZUELA, COLOMBIA, ECUADOR, INDIA, MEXICO

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.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 date to filter the data. Format 'YYYY-MM-DD'. Minimum start date is 2015-01-05.
- **`end_date`** (*`str, optional`*) – The end date to filter the data. Format 'YYYY-MM-DD'. Maximum end date is 2022-04-29.
- **`data_replication`** (*`bool, optional`*) – If True, the data is returned to replicate the output in Section 4.2 of the reference paper Song et al. (2024).

Returns

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

Return type

pandas.DataFrame

Warning

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.temperature package

Submodules

PyTimeVar.datasets.temperature.data module

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

Load the temperature dataset and optionally filter by by date range and/or regions. This dataset contains the average yearly temperature change in degrees Celsius for different regions of the world from 1961 to 2023.

Parameters

- **`start_date`** (*`str, optional`*) – The start year to filter the data. Format 'YYYY'. Minimum start year is 1961.
- **`end_date`** (*`str, optional`*) – The end year to filter the data. Format 'YYYY'. Maximum end year is 2023.
- **`regions`** (*`list, optional`*) –

  List of regions to filter the dataset by. Available options are:

  World, Africa, Asia, Europe, North America, Oceania, South America

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_date is earlier than the minimum year in the data or the end_date 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`**`(`*start_date=None*, *end_date=None*, *type='Open'*`)`

Load the USD index dataset and optionally filter by date range. This dataset contains the USD index data from 1961 to 2023.

Parameters

- **`start_date`** (*`str, optional`*) – The start date to filter the data. Format 'YYYY-MM-DD'. Minimum start date is 2015-01-20.
- **`end_date`** (*`str, optional`*) – The end date to filter the data. Format 'YYYY-MM-DD'. Maximum end date is 2024-09-06.
- **`type`** (*`str, optional`*) – The type of data to load. Available options are: ['Open', 'High', 'Low', 'Close']

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.inflation package

Submodules

PyTimeVar.datasets.inflation.data module

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

Load the inflation dataset, construct inflation rate dataset and optionally filter by date range. This dataset contains the inflation rate data from 1947 to 2024.

Parameters

- **start_date** (*str, optional*) – The start year-month to filter the data. Format 'YYYY-MM'. Minimum start year-month is 1947-02.
- **end_date** (*str, optional*) – The end year-month to filter the data. Format 'YYYY'. Maximum end year is 2024-08.

Returns

- *pandas.DataFrame* – DataFrame containing the filtered data with columns 'Date' and CPI-AUCSI.
- *Warnings*
- *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

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*, *mX*, *method='none'*, *vgamma0=None*, *bounds=None*, *options=None*, *niter=None*, *if_hetero=False*)

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. Choose between 'gaussian' or 'student'.
- **vgamma0** (*np.ndarray*) – Initial parameter vector.
- **bounds** (*list*) – List to define parameter space.
- **options** (*dict*) – Stopping criteria for optimization.

- **niter** (*int*) – The number of optimization iterations, for scipy.optimize.basinhopping()
- **if_hetero** (*bool*) – If True, a heteroskedastic specification is assumed. Filter additionally returns the estimated path of time-varying variance.

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

bounds

List to define parameter space.

Type
list

options

Stopping criteria for optimization.

Type
dict

niter

The number of optimization iterations, for scipy.optimize.basinhopping()

Type
int

`if_hetero`

If True, a heteroskedastic specification is assumed. Filter additionally returns the estimated path of time-varying variance.

Type
bool

`success`

If True, optimization was successful.

Type
bool

`betas`

The estimated coefficients.

Type
np.ndarray

`params`

The estimated GAS parameters.

Type
np.ndarray

`sigma2_t`

The estimated path of time-varying variance. None for a homoskedastic specification.

Type
np.ndarray

`inv_hessian`

The inverse Hessian after optimization.

Type
np.ndarray

Raises
`ValueError` – No valid number of initial parameters is provided.

Parameters

- **vY** (*ndarray*)
- **mX** (*ndarray*)
- **method** (*str*)
- **vgamma0** (*ndarray*)
- **bounds** (*list*)
- **options** (*dict*)
- **niter** (*int*)
- **if_hetero** (*bool*)

`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.
- **sigma2_hat** (*np.ndarray*) – The path of estimated time-varying variance. Only returned if_hetero = True.

`plot`(*tau=None*, *confidence_intervals=False*, *alpha=0.05*, *iM=1000*)

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

Parameters

- **tau** (*list, optional*) – The list looks the following: tau = [start,end]. The function will plot all data and estimates between start and end.
- **confidence_intervals** (*bool, optional*) – If True, simulation-based confidence intervals will be plotted around the estimates.
- **alpha** (*float*) – Significance level for confidence intervals.
- **iM** (*int*) – The number of simulations for simulation-based confidence intervals.

Raises

`ValueError` – No valid tau is provided.

Module contents

PyTimeVar.kalman package

Submodules

PyTimeVar.kalman.kalman module

`class PyTimeVar.kalman.kalman.Kalman`(*vY=None*, *T=None*, *R=None*, *Q=None*, *sigma_u=None*, *b_1=None*, *P_1=None*, *mX=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 variances of the state space model. Each entry corresponds to the observation noise variance at a time point.

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

`pred`

The predicted coefficients.

Type
np.ndarray

`smooth`

The smoothed coefficients.

`P_filt`

The filtered state variances.

Type
np.ndarray

`P`

The predicted state variances.

Type
np.ndarray

`V`

The smoothed state variances.

Type
np.ndarray

`fit`(*option='filter'*)

Computes the Kalman filtered states, one-step ahead predicted states or smoothed states for the data.

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

Raises
`ValueError` – No valid option is provided.

Returns
Estimated trend. If option='all', a list of trends is returned:

[filter, predictor, smoother]

Return type
np.ndarray

`plot`(*individual=False*, *tau=None*, *confidence_intervals=False*, *alpha=0.05*)

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

Parameters

- **`individual`** (*`bool, optional`*) – If True, the filtered states, the predictions, and smoothed states are shown in separate figures.
- **`tau`** (*`list, optional`*) – The list looks the following: tau = [start,end]. The function will plot all data and estimates between start and end.
- **`confidence_intervals`** (*`bool, optional`*) – If True, the confidence intervals will be plotted around the filtered states, the predictions, and smoothed states. The confidence intervals are plotted for individual plots only, not for joint plots.
- **`alpha`** (*`float`*) – Significance level for confidence intervals.

Raises
`ValueError` – No valid tau is provided.

`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*, *mX*, *h=0*, *bw_selection=None*, *kernel='epanechnikov'*, *LB_bw=None*, *UB_bw=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'.
- **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

tau_bw_selection

Points at which the cross-validation 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
int

`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

`h_gcv`
The optimal bandwidth value according to Generalized CV method.

Type
string

`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

Raises
`ValueError` – No valid bandwidth selection procedure is provided.

Parameters

- **`vY`** (*ndarray*)
- **`mX`** (*ndarray*)
- **`h`** (*float*)
- **`bw_selection`** (*str*)
- **`kernel`** (*str*)

- `LB_bw` (*float*)
- `UB_bw` (*float*)

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. (2025).
Bootstrap inference for linear time-varying coefficient models in locally stationary time series. Journal of Computational and Graphical Statistics, 34(2), 654-667.

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

Compute and plot confidence bands.

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. Default is 1299, if not provided by the user.
- `plots` (*bool*) – If True, plots are shown of the estimated coefficients and corresponding confidence bands.
- `bootstrap_type` (*str*)

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.

`fit()`

Fits the linear model by local linear regression to the data.

Returns
self.betahat – The estimated coefficients.

Return type
np.ndarray

`plot_betas`(*tau=None*)

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

Parameters

tau (*list, optional*) – The list looks the following: tau = [start,end]. The function will plot all data and estimates between start and end.

Raises

ValueError – No valid tau is provided.

`plot_predicted`(*tau=None*)

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

Parameters

tau (*list, optional*) – The list looks the following: tau = [start,end]. The function will plot all data and estimates between start and end.

Raises

ValueError – No valid tau is provided.

`plot_residuals`(*tau=None*)

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

Parameters

tau (*list, optional*) – The list looks the following: tau = [start,end]. The function will plot all data and estimates between start and end.

Raises

ValueError – No valid tau is provided.

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*, *n_powers=None*, *vgamma0=None*, *bounds=None*, *options=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.
- **bounds** (*tuple*) – Tuple to define parameter space.
- **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`

Tuple to define parameter space.

Type
tuple

`cons`

Dictionary that defines the 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

`C_LB_trend`

The lower bounds of the pointwise confidence intervals for the trend.

Type
np.ndarray

`C_UB_trend ; np.ndarray`

The upper bound of the pointwise confidence intervals for the trend.

`alpha`

The significance level for the confidence intervals.

Type
float

Raises
`ValueError` – No valid bounds are provided.

Parameters

- `vY` (*ndarray*)
- `n_powers` (*float*)
- `vgamma0` (*ndarray*)
- `bounds` (*tuple*)
- `options` (*dict*)

`confidence_intervals`(*bootstraptype*, *alpha=None*, *gamma=None*, *ic=None*, *B=1299*, *block_constant=2*, *verbose=True*)

Construct confidence intervals 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'
- `B` (*int*) – The number of bootstrap samples. Deafult is 1299, if not provided by the user.
- `block_constant` (*float*) – The constant to determine the window length for blocks bootstraps. Default is 2.
- `verbose` (*bool*) – If True, a progress bar is displayed.

Raises
`ValueError` – No valid bootstrap type is provided.

Returns
The pointwise lower and upper confidence intervals for the coefficients, powers, and trend, respectively.

Return type
list of tuples

`fit()`

Fits the Power-Law model to the data.

Returns

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

`plot`(*tau=None*, *confidence_intervals=True*)

Plots the original series and the trend component.

Parameters

- **tau** (*list, optional*) – The list looks the following: tau = [start,end]. The function will plot all data and estimates between start and end.
- **confidence_intervals** (*bool*) – If True, the estimated confidence intervals are displayed as well. If the confidence intervals are not computed yet, no confidence intervals are displayed.

Raises

`ValueError` – No valid tau is provided.

`summary()`

Print the mathematical equation for the fitted model

Module contents

3.1.2 Module contents

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX