enlopy Documentation

Release 0.1.dev9

Kavvadias Konstantinos

Apr 20, 2021

Contents

Version 0.1.dev9

Date Apr 20, 2021

CHAPTER 1

Contents

1.1 Reference/API

1.1.1 Generate module

Methods that generate or adjusted energy related timeseries based on given assumptions/input

enlopy.generate.**disag_upsample**(*Load*, *disag_profile*, *to_offset='h'*)

Upsample given timeseries, disaggregating based on given load profiles. e.g. From daily to hourly. The load of each day is distributed according to the disaggregation profile. The sum of each day remains the same.

Parameters

- **Load** (*pd.Series*) – Load profile to disaggregate
- **disag_profile** (*pd.Series, np.ndarray*) – disaggregation profile to be used on each timestep of the load. Has to be compatible with selected offset.
- **to_offset** (*str*) – Resolution of upsampling. has to be a valid pandas offset alias. (check here for all available offsets)

Returns the upsampled timeseries

Return type pd.Series

enlopy.generate.**gen_daily_stoch_el**(*total_energy=1.0*)

Generate stochastic dummy daily load based on hardcoded values. These values are the result of statistical analysis of electric loads profiles of more than 100 households. Mean and standard deviations per timestep were extracted from the normalized series. These are fed to *gen_gauss_markov()* method. :param total_energy: Sum of produced timeseries (daily load)

Returns random realization of timeseries

Return type nd.array

enlopy.generate.**gen_load_from_daily_monthly**(*ML*, *DWL*, *DNWL*, *weight=0.5*, *year=2015*)

Generate annual timeseries using monthly demand and daily profiles. Working days and weekends are built from different profiles having different weighting factors.

Parameters

- **ML** – monthly load (size = 12)
- **DWL** – daily load (working day) (size = 24). Have to be normalized (sum=1)
- **DNWL** – daily load (non working day) (size = 24) Have to be normalized (sum=1)
- **weight** – weighting factor between working and non working day (0 - 1)

Returns Generated timeseries

Return type pd.Series

enlopy.generate.**gen_load_sinus**(*daily_1*, *daily_2*, *monthly_1*, *monthly_2*, *annually_1*, *annually_2*)

Generate sinusoidal load with daily, weekly and yearly seasonality. Each term is estimated based on the following expression: $f(x; A1, A2, w) = A1\cos(2\pi/w \cdot x) + A2\sin(2\pi/w \cdot x)$

Parameters

- **daily_1** (*float*) – cosine coefficient for daily component (period 24)
- **daily_2** (*float*) – sinus coefficient for daily component (period 24)
- **monthly_1** (*float*) – cosine coefficient for monthly component (period 168)
- **monthly_2** (*float*) – sinus coefficient for monthly component (period 168)
- **annually_1** (*float*) – cosine coefficient for annual component (period 8760)
- **annually_2** (*float*) – sinus coefficient for annual component (period 8760)

Returns Generated timeseries

Return type pd.Series

enlopy.generate.**gen_load_from_LDC**(*LDC*, *Y=None*, *N=8760*)

Generate loads based on a Inverse CDF, such as a Load Duration Curve (LDC) Inverse transform sampling: Compute the value x such that F(x) = u. Take x to be the random number drawn from the distribution described by F.

Note: Due to the sampling process this function produces load profiles with unrealistic temporal sequence, which means that they cannot be treated as timeseries. It is recommended that *gen_load_from_PSD()* is used afterwards.

Parameters

- **LDC** (*np.ndarray*) – Load duration curve (2 x N) vector of the x, y coordinates of an LDC function (results of (get_LDC). x coordinates have to be normalized (max: 1 => 8760hrs)
- **Y** (*nd.array*) – a vector of random numbers. To be used for correlated loads. If None is supplied a random vector (8760) will be created.
- **N** (*int*) – Length of produced timeseries (if Y is not provided)

Returns vector with the same size as Y that respects the statistical distribution of the LDC

Return type np.ndarray

enlopy.generate.**gen_load_from_PSD**(*Sxx*, *x*, *dt=1*)

Algorithm for generating samples of a random process conforming to spectral density Sxx(w) and probability density function p(x).

Note: This is done by an iterative process which 'shuffles' the timeseries till convergence of both power spectrum and marginal distribution is reached. Also known as "Iterated Amplitude Adjusted Fourier Transform (IAAFT). Adopted from *J.M. Nichols, C.C. Olson, J.V. Michalowicz, F. Bucholtz, (2010), "A simple algorithm for generating spectrally colored, non-Gaussian signals" Probabilistic Engineering Mechanics, Vol 25, 315-322* and *Schreiber, T. and Schmitz, A. (1996) "Improved Surrogate Data for Nonlinearity Tests", Physical Review Letters, Vol 77, 635-638.*

Parameters

- **Sxx** – Spectral density (two sided)
- **x** – Sequence of observations created by the desirable PDF. You can use *gen_load_from_LDC()* for that.
- **dt** – Desired temporal sampling interval. [Dt = 2pi / (N * Dw)]

Returns The spectrally corrected timeseries

Return type pd.Series

enlopy.generate.**gen_gauss_markov**(*mu*, *st*, *r*)

Generate timeseries based on means, stadnard deviation and autocorrelation per timestep

---

Note: Based on *A.M. Breipohl, F.N. Lee, D. Zhai, R. Adapa, A Gauss-Markov load model for the application in risk evaluation and production simulation, Transactions on Power Systems, 7 (4) (1992), pp. 1493-1499*

Parameters

- **mu** – array of means. Can be either 1d or 2d
- **st** – array of standard deviations. Can be either 1d or 2d. Can be either scalar (same for entire timeseries or array with the same length as the timeseries
- **r** – Autoregressive coefficient AR(1). Has to be between [-1,1]. Can be either scalar (same for entire timeseries or array with the same length as the timeseries

Returns a realization of the timeseries

Return type pd.Series, pd.DataFrame

enlopy.generate.**remove_outliers**(*Load*, ***kwargs*)

Removes outliers identified by `detect_outliers()` and replaces them by interpolated value.

Parameters

- **Load** – input timeseries
- ****kwargs** – Exposes keyword arguments of `detect_outliers()`

Returns Timeseries cleaned from outliers

enlopy.generate.**gen_demand_response**(*Load*, *percent_peak_hrs_month=0.03*, *percent_shifted=0.05*, *shave=False*)

Simulate a demand response mechanism that makes the load profile less peaky. The load profile is analyzed per selected period (currently month) and the peak hours have their load shifted to low load hours or shaved. When

not shaved the total load is the same as that one from the initial timeseries, otherwise it is smaller due to the shaved peaks. The peak load is reduced by a predefined percentage.

Parameters

- **Load** (*pd.Series*) – Load
- **percent_peak_hrs_month** (*float*) – fraction of hours to be shifted
- **percent_shifted** (*float*) – fraction of energy to be shifted if the day is tagged for shifting/shaving
- **shave** (*bool*) – If False peak load will be transfered to low load hours, otherwise it will be shaved.

Returns New load profile with reduced peaks. The peak can be shifted to low load hours or shaved

Return type pd.Series

enlopy.generate.**add_noise**(*Load*, *mode*, *st*, *r=0.9*, *Lmin=0*)

Add noise with given characteristics.

Parameters

- **Load** (*pd.Series,pd.DataFrame*) – 1d or 2d timeseries
- **mode** (*int*) – 1 Normal Distribution, 2: Uniform Distribution, 3: Gauss Markov (autoregressive gaussian)
- **st** (*float*) – Noise parameter. Scaling of random values
- **r** (*float*) – Applies only for mode 3. Autoregressive coefficient AR(1). Has to be between [-1,1]
- **Lmin** (*float*) – minimum load values. This is used to trunc values below zero if they are generated with a lot of noise

Returns Load with noise

Return type pd.Series

enlopy.generate.**gen_corr_arrays**(*Na*, *length*, *M*, *to_uniform=True*)

Generating correlated normal variates. Assume one wants to create a vector of random variates Z which is distributed according to Z~N(μ,Σ) where μ is the vector of means, and Σ is the variance-covariance matrix. http://comisef.wikidot.com/tutorial:correlateduniformvariates

Parameters

- **Na** (*int*) – number of vectors e.g (3)
- **length** (*int*) – generated vector size (e.g 8760)
- **M** (*np.ndarray*) – correlation matrix. Should be of size Na x Na
- **to_uniform** (*bool*) – True if the correlation matrix needs to be adjusted for uniforms

Returns Realization of randomly generated correlated variables. Size : (Na, length) e.g. (3, 8760)

Return type np.ndarray

enlopy.generate.**gen_analytical_LDC**(*U*, *duration=8760*, *bins=1000*)

Generates the Load Duration Curve based on empirical parameters. The following equation is used. $f(x; P, CF, BF) =$
$fracP - xP - BFcdotP^{fracCF-1BF-CF}$

Parameters **U** (*tuple*) – parameter vector [Peak load, capacity factor%, base load%, hours] or dict

Returns a 2D array [x, y] ready for plotting (e.g. plt(*gen_analytical_LDC(U)))

Return type np.ndarray

1.1.2 Analysis module

enlopy.analysis.**reshape_timeseries**(*Load*, *x='dayofyear'*, *y=None*, *aggfunc='sum'*)

Returns a reshaped pandas DataFrame that shows the aggregated load for selected timeslices. e.g. time of day vs day of year

Parameters

- **Load** (*pd.Series, np.ndarray*) – timeseries
- **x** (*str*) – x axis aggregator. Has to be an accessor of pd.DatetimeIndex (year, dayoftime, week etc.)
- **y** (*str*) – similar to above for y axis

Returns reshaped pandas dataframe according to x,y

enlopy.analysis.**get_LDC**(*Load*, *x_norm=True*, *y_norm=False*)

Generates the Load Duration Curve based on a given load. For 2-dimensional dataframes the x-axis sorting is done based on sum of all series. Sorting on the y-axis is done based on the coefficient of variance.

Parameters

- **Load** (*pd.Series*) – timeseries
- **x_norm** (*bool*) – Normalize x axis (0,1)
- **y_norm** (*bool*) – Normalize y axis (0,1)

Returns tuple (x, y) ready for plotting (e.g. plt(*LDC_load(load)))

Return type np.ndarray

enlopy.analysis.**get_load_archetypes**(*Load*, *k=2*, *x='hour'*, *y='dayofyear'*, *plot_diagnostics=False*)

Extract typical load profiles using k-means and vector quantization. the time scale of archetypes depend on the selected dimensions (x,y). For the default values daily archetypes will be extracted.

Parameters

- **Load** (*pd.Series*) – timeseries
- **k** (*int*) – number of archetypes to identify and extract
- **x** (*str*) – This will define how the timeseries will be grouped by. Has to be an accessor of pd.DatetimeIndex
- **y** (*str*) – similar to above for y axis.
- **plot_diagnostics** (*bool*) – If true a figure is plotted showing an overview of the results

Returns dimensions (k, len(x))

Return type np.ndarray

enlopy.analysis.**get_load_stats**(*Load*, *per='a'*)

Find load profile characteristics. Among other it estimates: peak, load factor, base load factor, operating hours,

Parameters

- **Load** – timeseries of load to be examined. A timeseries index is needed.

- **per** – reporting periods. Annual by default. Based on pandas time offsets

Returns Parameter dictionary

Return type dict

enlopy.analysis.**detect_outliers**(*Load*, *threshold=None*, *window=5*, *plot_diagnostics=False*)

Detect and optionally remove outliers based on median rolling window filtering. Inspired by https://ocefpaf.github.io/python4oceanographers/blog/2015/03/16/outlier_detection/

Parameters

- **Load** – input timeseries
- **threshold** – if None then 3 sigma is selected as threshold
- **window** – how many values to check
- **plot_diagnostics** – Plot diagnostics to check whether the outliers were removed accurately

Returns index position of detected outliers

1.1.3 Plotting module

enlopy.plot.**plot_heatmap**(*Load*, *x='dayofyear'*, *y='hour'*, *aggfunc='sum'*, *bins=8*, *figsize=(16, 6)*, *edgecolors='none'*, *cmap='Oranges'*, *colorbar=True*, *ax=None*, ***pltargs*)

Returns a 2D heatmap of the reshaped timeseries based on x, y

Parameters

- **Load** – 1D pandas with timed index
- **x** – Parameter for *enlopy.analysis.reshape_timeseries()*
- **y** – Parameter for *enlopy.analysis.reshape_timeseries()*
- **bins** – Number of bins for colormap
- **edgecolors** – colour of edges around individual squares. 'none' or 'w' is recommended.
- **cmap** – colormap name (from colorbrewer, matplotlib etc.)
- ****pltargs** – Exposes matplotlib.plot arguments

Returns 2d heatmap

enlopy.plot.**plot_3d**(*Load*, *x='dayofyear'*, *y='hour'*, *aggfunc='sum'*, *bins=15*, *cmap='Oranges'*, *colorbar=True*, ***pltargs*)

Returns a 3D plot of the reshaped timeseries based on x, y

Parameters

- **Load** – 1D pandas with timed index
- **x** – Parameter for *enlopy.analysis.reshape_timeseries()*
- **y** – Parameter for *enlopy.analysis.reshape_timeseries()*
- **bins** – Number of bins for colormap
- **cmap** – colormap name (from colorbrewer, matplotlib etc.)
- ****pltargs** – Exposes `matplotlib.pyplot.surface()` arguments

Returns 3d plot

enlopy.plot.**plot_percentiles**(*Load, x='hour', zz='week', perc_list=[[5, 95], [25, 75], 50], ax=None, color='blue', **kwargs*)

Plot predefined percentiles per timestep

Parameters

- **Load** – 1D pandas with timed index
- **x** (*str*) – x axis aggregator. See *enlopy.analysis.reshape_timeseries()*
- **zz** (*str*) – similar to above for y axis
- **perc_list** (*list*) – List of percentiles to plot. If it is an integer then it will be plotted as a line. If it is list it has to contain two items and it will be plotted using fill_between()
- ****kwargs** – exposes arguments of `matplotlib.pyplot.fill_between()`

Returns Plot

enlopy.plot.**plot_rug**(*df_series, on_off=False, cmap='Greys', fig_title='', fig_width=14, normalized=False*)

Create multiaxis rug plot from pandas Dataframe

Parameters

- **df_series** (*pd.DataFrame*) – 2D pandas with timed index
- **on_off** (*bool*) – if True all points that are above 0 will be plotted as one color. If False all values will be colored based on their value.
- **cmap** (*str*) – colormap name (from colorbrewer, matplotlib etc.)
- **fig_title** (*str*) – Figure title
- **normalized** (*bool*) – if True, all series colormaps will be normalized based on the maximum value of the dataframe

Returns plot

enlopy.plot.**plot_boxplot**(*Load, by='day', **pltargs*)

Return boxplot plot for each day of the week

Parameters

- **Load** (*pd.Series*) – 1D pandas Series with timed index
- **by** (*str*) – group results by 'day' or 'hour'
- ****pltargs** (*dict*) – Exposes `matplotlib.pyplot.plot()` arguments

Returns plot

enlopy.plot.**plot_LDC**(*Load, stacked=True, x_norm=True, y_norm=False, cmap='Spectral', color='black', legend=False, zoom_peak=False, ax=None, **kwargs*)

Plot Load duration curve

Parameters

- **Load** (*pd.Series*) – 1D pandas Series with timed index
- **x_norm** (*bool*) – Normalize x axis (0,1)
- **y_norm** (*bool*) – Normalize y axis (0,1)
- **color** (*str*) – color of line. For Series only (1D)
- **cmap** (*str*) – Colormap of area. For Dataframes only (2D)
- **legend** (*bool*) – Show legend. For Dataframes only (2D)

- **zoom_peak** (*bool*) – Show zoomed plot of peak
- **kwargs** (*dict*) – exposes arguments of pd.DataFrame.plot.area

Returns Load duration curve plot

1.1.4 Utilities module

enlopy.utils.**make_timeseries**(*x=None*, *year=None*, *length=None*, *startdate=None*, *freq=None*)

Convert numpy array to a pandas series with a timed index. Convenience wrapper around a datetime-indexed pd.DataFrame.

Parameters

- **x** – (nd.array) raw data to wrap into a pd.Series
- **startdate** – pd.datetime
- **year** – year of timeseries
- **freq** – offset keyword (e.g. 15min, H)
- **length** – length of timeseries

Returns pd.Series or pd.Dataframe with datetimeindex

enlopy.utils.**clean_convert**(*x*, *force_timed_index=True*, *always_df=False*, ***kwargs*)

Converts a list, a numpy array, or a dataframe to pandas series or dataframe, depending on the compatibility and the requirements. Designed for maximum compatibility.

Parameters

- **x** (*list, np.ndarray*) – Vector or matrix of numbers. it can be pd.DataFrame, pd.Series, np.ndarray or list
- **force_timed_index** (*bool*) – if True it will return a timeseries index
- **year** (*int*) – Year that will be used for the index
- **always_df** (*bool*) – always return a dataframe even if the data is one dimensional
- ****kwargs** – Exposes arguments of *make_timeseries()*

Returns Timeseries

Return type pd.Series

enlopy is an open source python library with methods to generate, process, analyze, and plot energy related timeseries.

While it can be used for any kind of data it has a strong focus on those that are related with energy i.e. electricity/heat demand or generation, prices etc. The methods included here are carefully selected to fit in that context and they had been, gathered, generalized and encapsulated during the last years while working on different research studies.

The aim is to provide a higher level API than the one that is already available in commonly used scientific packages (pandas, numpy, scipy). This facilitates the analysis and processing of energy load timeseries that can be used for modelling and statistical analysis. In some cases it is just a convenience wrapper of common packages just as pandas and in other cases it implements methods or statistical models found in literature.

It consists of four modules that include among others the following:

- Analysis: Overview of descriptive statistics, reshape, load duration curve, extract daily archetypes (clustering)
- Plot: 2d heatmap, 3d plot, boxplot, rugplot

- `Generate`: generate from daily and monthly profiles, generate from sinusoidal function, sample from given load duration curve, or from given PSD, add noise gaussian and autoregressive noise, generate correlated load profiles, fit to analytical load duration curve
- `Statistics`: Feature extraction from timeseries for a quick overview of the characteristics of any load curve. Useful when coupled with machine learning packages.

This library is not focusing on regression and prediction (e.g. ARIMA, state-space etc.), since there are numerous relevant libraries around.

The documentation is under development. Please check the source code, the *Reference/API* or the example jupyter notebook in the github repository for feature details.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

e

Index