PVAnalytics

pvlib

Feb 18, 2022

CONTENTS:

PVAnalytics is a python library that supports analytics for PV systems. It provides functions for quality control, filtering, and feature labeling and other tools supporting the analysis of PV system-level data.

The source code for PVAnalytics is hosted on github.

CHAPTER

ONE

LIBRARY OVERVIEW

The functions provided by PVAnalytics are organized in submodules based on their anticipated use. The list below provides a general overview; however, not all modules have functions at this time, see the API reference for current library status.

- `quality` contains submodules for different kinds of data quality checks.
  - `quality.irradiance` contains quality checks for irradiance measurements.
  - `quality.weather` contains quality checks for weather data (e.g. tests for physically plausible values of temperature, wind speed, humidity).
  - `quality.outliers` contains functions for identifying outliers.
  - `quality.gaps` contains functions for identifying gaps in the data (i.e. missing values, stuck values, and interpolation).
  - `quality.time` quality checks related to time (e.g. timestamp spacing, time shifts).
  - `quality.util` general purpose quality functions (e.g. simple range checks).
- `filtering` as the name implies, contains functions for data filtering.
- `features` contains submodules with different methods for identifying and labeling salient features.
  - `features.clipping` functions for labeling inverter clipping.
  - `features.clearsky` functions for identifying periods of clear sky conditions.
  - `features.daytime` functions for identifying periods of day and night.
  - `features.orientation` functions for identifying orientation-related features in the data (e.g. days where the data looks like there is a functioning tracker). These functions are distinct from the functions in the `system` module in that we are identifying features of data rather than properties of the system that produced the data.
  - `features.shading` functions for identifying shadows.
- `system` identification of PV system characteristics from data (e.g. nameplate power, tilt, azimuth)
- `translate` contains functions for translating data to other conditions (e.g. IV curve translators, temperature adjustment, irradiance adjustment)
- `metrics` contains functions for computing PV system-level metrics (e.g. performance ratio)
- `fitting` contains submodules for different types of models that can be fit to data (e.g. temperature models)
- `dataclasses` contains classes for normalizing data (e.g. an `IVCurve` class)

CHAPTER

TWO

DEPENDENCIES

This project follows the guidelines laid out in NEP-29. It supports:

- All minor versions of Python released 42 months prior to the project, and at minimum the two latest minor versions.
- All minor versions of numpy released in the 24 months prior to the project, and at minimum the last three minor versions
- The latest release of PVLib.

PVAnalytics depends on the following packages:

```
numpy>=1.15.0
pandas>=0.24.0,!=1.1.*
pvlib>=0.8.0
scipy>=1.2.0
statsmodels>=0.9.0
scikit-image>=0.16.0
```

CHAPTER

THREE

CONTENTS

3.1 API Reference

3.1.1 Quality

Irradiance

The `check_*_limits_qcrad` functions use the QCRad algorithm[1] to identify irradiance measurements that are beyond physical limits.

quality.irradiance.check_ghi_limits_qcrad(...)	Test for physical limits on GHI using the QCRad criteria.
quality.irradiance.check_dhi_limits_qcrad(...)	Test for physical limits on DHI using the QCRad criteria.
quality.irradiance.check_dni_limits_qcrad(...)	Test for physical limits on DNI using the QCRad criteria.

pvanalytics.quality.irradiance.check_ghi_limits_qcrad

pvanalytics.quality.irradiance.**check_ghi_limits_qcrad**(*ghi*, *solar_zenith*, *dni_extra*, *limits=None*)

Test for physical limits on GHI using the QCRad criteria.

Test is applied to each GHI value. A GHI value passes if value > lower bound and value < upper bound. Lower bounds are constant for all tests. Upper bounds are calculated as

$$ub = min + mult * dni\_extra * cos(solar\_zenith)^{exp}$$

Parameters

- **ghi** (*Series*) – Global horizontal irradiance in W/m^2
- **solar_zenith** (*Series*) – Solar zenith angle in degrees
- **dni_extra** (*Series*) – Extraterrestrial normal irradiance in W/m^2
- **limits** (*dict, default QCRAD_LIMITS*) – Must have keys 'ghi_ub' and 'ghi_lb'. For 'ghi_ub' value is a dict with keys {'mult', 'exp', 'min'} and float values. For 'ghi_lb' value is a float.

Returns True where value passes limits test.

[1] C. N. Long and Y. Shi, An Automated Quality Assessment and Control Algorithm for Surface Radiation Measurements, The Open Atmospheric Science Journal 2, pp. 23-37, 2008.

Return type Series

Notes

pvanalytics.quality.irradiance.check_dhi_limits_qcrad

pvanalytics.quality.irradiance.**check_dhi_limits_qcrad**(*dhi*, *solar_zenith*, *dni_extra*, *limits=None*)

Test for physical limits on DHI using the QCRad criteria.

Test is applied to each DHI value. A DHI value passes if value > lower bound and value < upper bound. Lower bounds are constant for all tests. Upper bounds are calculated as

$$ub = min + mult * dni\_extra * cos(solar\_zenith)^{exp}$$

Parameters

- **dhi** (*Series*) – Diffuse horizontal irradiance in W/m^2
- **solar_zenith** (*Series*) – Solar zenith angle in degrees
- **dni_extra** (*Series*) – Extraterrestrial normal irradiance in W/m^2
- **limits** (*dict, default QCRAD_LIMITS*) – Must have keys 'dhi_ub' and 'dhi_lb'. For 'dhi_ub' value is a dict with keys {'mult', 'exp', 'min'} and float values. For 'dhi_lb' value is a float.

Returns True where value passes limit test.

Return type Series

Notes

pvanalytics.quality.irradiance.check_dni_limits_qcrad

pvanalytics.quality.irradiance.**check_dni_limits_qcrad**(*dni*, *solar_zenith*, *dni_extra*, *limits=None*)

Test for physical limits on DNI using the QCRad criteria.

Test is applied to each DNI value. A DNI value passes if value > lower bound and value < upper bound. Lower bounds are constant for all tests. Upper bounds are calculated as

$$ub = min + mult * dni\_extra * cos(solar\_zenith)^{exp}$$

Parameters

- **dni** (*Series*) – Direct normal irradiance in W/m^2
- **solar_zenith** (*Series*) – Solar zenith angle in degrees
- **dni_extra** (*Series*) – Extraterrestrial normal irradiance in W/m^2

- **limits** (*dict, default QCRAD_LIMITS*) – Must have keys 'dni_ub' and 'dni_lb'. For 'dni_ub' value is a dict with keys {'mult', 'exp', 'min'} and float values. For 'dni_lb' value is a float.

Returns True where value passes limit test.

Return type Series

Notes

Copyright (c) 2019 SolarArbiter. See the file LICENSES/SOLARFORECASTARBITER_LICENSE at the top level directory of this distribution and at https://github.com/pvlib/pvanalytics/blob/master/LICENSES/SOLARFORECASTARBITER_LICENSE for more information.

All three checks can be combined into a single function call.

quality.irradiance.check_irradiance_limits_qcrad(...)	Test for physical limits on GHI, DHI or DNI using the QCRad criteria.

pvanalytics.quality.irradiance.check_irradiance_limits_qcrad

pvanalytics.quality.irradiance.**check_irradiance_limits_qcrad**(*solar_zenith*, *dni_extra*, *ghi=None*, *dhi=None*, *dni=None*, *limits=None*)

Test for physical limits on GHI, DHI or DNI using the QCRad criteria.

Criteria from[1] are used to determine physically plausible lower and upper bounds. Each value is tested and a value passes if value > lower bound and value < upper bound. Lower bounds are constant for all tests. Upper bounds are calculated as

$$ub = min + mult * dni\_extra * cos(solar\_zenith)^{exp}$$

Note: If any of *ghi*, *dhi*, or *dni* are None, the corresponding element of the returned tuple will also be None.

Parameters

- **solar_zenith** (*Series*) – Solar zenith angle in degrees
- **dni_extra** (*Series*) – Extraterrestrial normal irradiance in W/m^2
- **ghi** (*Series or None, default None*) – Global horizontal irradiance in W/m^2
- **dhi** (*Series or None, default None*) – Diffuse horizontal irradiance in W/m^2
- **dni** (*Series or None, default None*) – Direct normal irradiance in W/m^2
- **limits** (*dict, default QCRAD_LIMITS*) – for keys 'ghi_ub', 'dhi_ub', 'dni_ub', value is a dict with keys {'mult', 'exp', 'min'} and float values. For keys 'ghi_lb', 'dhi_lb', 'dni_lb', value is a float.

Returns

- **ghi_limit_flag** (*Series*) – True for each value that is physically possible. None if *ghi* is None.
- **dhi_limit_flag** (*Series*) – True for each value that is physically possible. None if *dni* is None.

[1] C. N. Long and Y. Shi, An Automated Quality Assessment and Control Algorithm for Surface Radiation Measurements, The Open Atmospheric Science Journal 2, pp. 23-37, 2008.

- **dni_limit_flag** (*Series*) – True for each value that is physically possible. None if *dni* is None.

Notes

Copyright (c) 2019 SolarArbiter. See the file LICENSES/SOLARFORECASTARBITER_LICENSE at the top level directory of this distribution and at https://github.com/pvlib/pvanalytics/blob/master/LICENSES/SOLARFORECASTARBITER_LICENSE for more information.

References

Irradiance measurements can also be checked for consistency.

quality.irradiance.check_irradiance_consistency_qcrad(...)	Check consistency of GHI, DHI and DNI using QCRad criteria.

pvanalytics.quality.irradiance.check_irradiance_consistency_qcrad

pvanalytics.quality.irradiance.**check_irradiance_consistency_qcrad**(*ghi*, *solar_zenith*, *dhi*, *dni*, *param=None*)

Check consistency of GHI, DHI and DNI using QCRad criteria.

Uses criteria given in[1] to validate the ratio of irradiance components.

Warning: Not valid for night time. While you can pass data from night time to this function, be aware that the truth values returned for that data will not be valid.

Parameters

- **ghi** (*Series*) – Global horizontal irradiance in W/m^2
- **solar_zenith** (*Series*) – Solar zenith angle in degrees
- **dhi** (*Series*) – Diffuse horizontal irradiance in W/m^2
- **dni** (*Series*) – Direct normal irradiance in W/m^2
- **param** (*dict*) – keys are 'ghi_ratio' and 'dhi_ratio'. For each key, value is a dict with keys 'high_zenith' and 'low_zenith'; for each of these keys, value is a dict with keys 'zenith_bounds', 'ghi_bounds', and 'ratio_bounds' and value is an ordered pair [lower, upper] of float.

Returns

- **consistent_components** (*Series*) – True where *ghi*, *dhi* and *dni* components are consistent.
- **diffuse_ratio_limit** (*Series*) – True where diffuse to GHI ratio passes limit test.

[1] C. N. Long and Y. Shi, An Automated Quality Assessment and Control Algorithm for Surface Radiation Measurements, The Open Atmospheric Science Journal 2, pp. 23-37, 2008.

Notes

References

GHI and POA irradiance can be validated against clearsky values to eliminate data that is unrealistically high.

quality.irradiance.clearsky_limits(measured, ...)	Identify irradiance values which do not exceed clearsky values.

pvanalytics.quality.irradiance.clearsky_limits

pvanalytics.quality.irradiance.**clearsky_limits**(*measured*, *clearsky*, *csi_max=1.1*)

Identify irradiance values which do not exceed clearsky values.

Uses `pvlib.irradiance.clearsky_index()` to compute the clearsky index as the ratio of *measured* to *clearsky*. Compares the clearsky index to *csi_max* to identify values in *measured* that are less than or equal to *csi_max*.

Parameters

- **measured** (*Series*) – Measured irradiance in W/m^2.
- **clearsky** (*Series*) – Expected clearsky irradiance in W/m^2.
- **csi_max** (*float, default 1.1*) – Maximum ratio of *measured* to *clearsky* (clearsky index).

Returns True for each value where the clearsky index is less than or equal to *csi_max*.

Return type Series

Notes

You may want to identify entire days that have unrealistically high or low insolation. The following function examines daily insolation, validating that it is within a reasonable range of the expected clearsky insolation for the same day.

quality.irradiance.daily_insolation_limits(...)	Check that daily insolation lies between minimum and maximum values.

pvanalytics.quality.irradiance.daily_insolation_limits

pvanalytics.quality.irradiance.**daily_insolation_limits**(*irrad*, *clearsky*, *daily_min=0.4*, *daily_max=1.25*)

Check that daily insolation lies between minimum and maximum values.

Irradiance measurements and clear-sky irradiance on each day are integrated with the trapezoid rule to calculate daily insolation.

Parameters

- **irrad** (*Series*) – Irradiance measurements (GHI or POA).
- **clearsky** (*Series*) – Clearsky irradiance.
- **daily_min** (*float, default 0.4*) – Minimum ratio of daily insolation to daily clearsky insolation.
- **daily_max** (*float, default 1.25*) – Maximum ratio of daily insolation to daily clearsky insolation.

Returns True for values on days where the ratio of daily insolation to daily clearsky insolation is between *daily_min* and *daily_max*.

Return type Series

Notes

The default limits (*daily_max* and *daily_min*) have been set for GHI and POA irradiance for systems with *fixed* azimuth and tilt. If you pass POA irradiance for a tracking system it is recommended that you increase *daily_max* to 1.35.

The default values for *daily_min* and *daily_max* were taken from the PVFleets QA Analysis project.

Gaps

Identify gaps in the data.

quality.gaps.interpolation_diff(x[, window, ...])	Identify sequences which appear to be linear.

pvanalytics.quality.gaps.interpolation_diff

pvanalytics.quality.gaps.**interpolation_diff**(*x*, *window=6*, *rtol=1e-05*, *atol=1e-08*, *mark='tail'*)

Identify sequences which appear to be linear.

Sequences are linear if the first difference appears to be constant. For a window of length N, the last value (index N-1) is flagged if all values in the window appear to be a line segment.

Parameters *rtol* and *atol* have the same meaning as in `numpy.allclose()`.

Parameters

- **x** (*Series*) – data to be processed
- **window** (*int, default 6*) – number of sequential values that, if the first difference is constant, are classified as a linear sequence

- **rtol** (*float, default 1e-5*) – tolerance relative to max(abs(x.diff()) for detecting a change
- **atol** (*float, default 1e-8*) – absolute tolerance for detecting a change in first difference
- **mark** (*str, default 'tail'*) – How much of the window to mark `True` when a sequence of interpolated values is detected. Can be one of 'tail', 'end', or 'all'.
  - If 'tail' (the default) then every point in the window *except* the first point is marked `True`.
  - If 'end' then the first *window - 1* values in an interpolated sequence are marked `False` and all subsequent values in the sequence are marked `True`.
  - If 'all' then every point in the window *including* the first point is marked `True`.

Returns True for each value that is part of a linear sequence

Return type Series

Raises `ValueError` – If *window* < *3* or *mark* is not one of 'tail', 'end', or 'all'.

Notes

Data sometimes contains sequences of values that are "stale" or "stuck." These are contiguous spans of data where the value does not change within the precision given. The functions below can be used to detect stale values.

Note: If the data has been altered in some way (i.e. temperature that has been rounded to an integer value) before being passed to these functions you may see unexpectedly large amounts of stale data.

quality.gaps.stale_values_diff(x[, window, ...])	Identify stale values in the data.
quality.gaps.stale_values_round(x[, window, ...])	Identify stale values by rounding.

pvanalytics.quality.gaps.stale_values_diff

pvanalytics.quality.gaps.**stale_values_diff**(*x*, *window=6*, *rtol=1e-05*, *atol=1e-08*, *mark='tail'*)

Identify stale values in the data.

For a window of length N, the last value (index N-1) is considered stale if all values in the window are close to the first value (index 0).

Parameters *rtol* and *atol* have the same meaning as in `numpy.allclose()`.

Parameters

- **x** (*Series*) – data to be processed
- **window** (*int, default 6*) – number of consecutive values which, if unchanged, indicates stale data
- **rtol** (*float, default 1e-5*) – relative tolerance for detecting a change in data values
- **atol** (*float, default 1e-8*) – absolute tolerance for detecting a change in data values

- **mark** (*str, default 'tail'*) – How much of the window to mark `True` when a sequence of stale values is detected. Can one be of 'tail', 'end', or 'all'.
  - If 'tail' (the default) then every point in the window *except* the first point is marked `True`.
  - If 'end' then the first *window - 1* values in a stale sequence sequence are marked `False` and all subsequent values in the sequence are marked `True`.
  - If 'all' then every point in the window *including* the first point is marked `True`.

Returns True for each value that is part of a stale sequence of data

Return type Series

Raises `ValueError` – If *window* < *2* or *mark* is not one of 'tail', 'end', or 'all'.

Notes

pvanalytics.quality.gaps.stale_values_round

pvanalytics.quality.gaps.**stale_values_round**(*x*, *window=6*, *decimals=3*, *mark='tail'*)

Identify stale values by rounding.

A value is considered stale if it is part of a sequence of length *window* of values that are identical when rounded to *decimals* decimal places.

Parameters

- **x** (*Series*) – Data to be processed.
- **window** (*int, default 6*) – Number of consecutive identical values for a data point to be considered stale.
- **decimals** (*int, default 3*) – Number of decimal places to round to.
- **mark** (*str, default 'tail'*) – How much of the window to mark `True` when a sequence of stale values is detected. Can be one of 'tail', 'end', or 'all'.
  - If 'tail' (the default) then every point in the window *except* the first point is marked `True`.
  - If 'end' then the first *window - 1* values in a stale sequence sequence are marked `False` and all subsequent values in the sequence are marked `True`.
  - If 'all' then every point in the window *including* the first point is marked `True`.

Returns True for each value that is part of a stale sequence of data.

Return type Series

Raises `ValueError` – If *mark* is not one of 'tail', 'end', or 'all'.

Notes

Based on code from the pvfleets_qa_analysis project. Copyright (c) 2020 Alliance for Sustainable Energy, LLC.

The following functions identify days with incomplete data.

quality.gaps.completeness_score(series[, ...])	Calculate a data completeness score for each day.
quality.gaps.complete(series[, ...])	Select data points that are part of days with complete data.

pvanalytics.quality.gaps.completeness_score

pvanalytics.quality.gaps.**completeness_score**(*series*, *freq=None*, *keep_index=True*)

Calculate a data completeness score for each day.

The completeness score for a given day is the fraction of time in the day for which there is data (a value other than NaN). The time duration attributed to each value is equal to the timestamp spacing of *series*, or *freq* if it is specified. For example, a 24-hour time series with 30 minute timestamp spacing and 24 non-NaN values would have data for a total of 12 hours and therefore a completeness score of 0.5.

Parameters

- **series** (*Series*) – A DatetimeIndexed series.
- **freq** (*str, default None*) – Interval between samples in the series as a pandas frequency string. If None, the frequency is inferred using `pandas.infer_freq()`.
- **keep_index** (*boolean, default True*) – Whether or not the returned series has the same index as *series*. If False the returned series will be indexed by day.

Returns A series of floats giving the completeness score for each day (fraction of the day for which *series* has data).

Return type Series

Raises `ValueError` – If *freq* is longer than the frequency inferred from *series*.

pvanalytics.quality.gaps.complete

pvanalytics.quality.gaps.**complete**(*series*, *minimum_completeness=0.333*, *freq=None*)

Select data points that are part of days with complete data.

A day has complete data if its completeness score is greater than or equal to *minimum_completeness*. The completeness score is calculated by *completeness_score()*.

Parameters

- **series** (*Series*) – The data to be checked for completeness.
- **minimum_completeness** (*float, default 0.333*) – Fraction of the day that must have data.
- **freq** (*str, default None*) – The expected frequency of the data in *series*. If none then the frequency is inferred from the data.

Returns A series of booleans with True for each value that is part of a day with completeness greater than *minimum_completeness*.

Return type Series

Raises `ValueError` – See *completeness_score()*.

See also:

completeness_score

Many data sets may have leading and trailing periods of days with sporadic or no data. The following functions can be used to remove those periods.

quality.gaps.start_stop_dates(series[, days])	Get the start and end of data excluding leading and trailing gaps.
quality.gaps.trim(series[, days])	Mask the beginning and end of the data if not all True.
quality.gaps.trim_incomplete(series[, ...])	Trim the series based on the completeness score.

pvanalytics.quality.gaps.start_stop_dates

pvanalytics.quality.gaps.**start_stop_dates**(*series*, *days=10*)

Get the start and end of data excluding leading and trailing gaps.

Parameters

- **series** (*Series*) – A DatetimeIndexed series of booleans.
- **days** (*int, default 10*) – The minimum number of consecutive days where every value in *series* is True for data to start or stop.

Returns

- **start** (*Datetime or None*) – The first valid day. If there are no sufficiently long periods of valid days then None is returned.
- **stop** (*Datetime or None*) – The last valid day. None if start is None.

pvanalytics.quality.gaps.trim

pvanalytics.quality.gaps.**trim**(*series*, *days=10*)

Mask the beginning and end of the data if not all True.

Parameters

- **series** (*Series*) – A DatetimeIndexed series of booleans
- **days** (*int, default 10*) – Minimum number of consecutive days that are all True for 'good' data to start.

Returns A series of booleans with True for all data points between the first and last block of *days* consecutive days that are all True in *series*. If *series* does not contain such a block of consecutive True values, then the returned series will be entirely False.

Return type Series

See also:

start_stop_dates

pvanalytics.quality.gaps.trim_incomplete

pvanalytics.quality.gaps.**trim_incomplete**(*series*, *minimum_completeness=0.333333*, *days=10*, *freq=None*)

Trim the series based on the completeness score.

Combines *completeness_score()* and *trim()*.

Parameters

- **series** (*Series*) – A DatetimeIndexed series.
- **minimum_completeness** (*float, default 0.333333*) – The minimum completeness score for each day.
- **days** (*int, default 10*) – The number of consecutive days with completeness greater than *minumum_completeness* for the 'good' data to start or end. See *start_stop_dates()* for more information.
- **freq** (*str, default None*) – The expected frequency of the series. See *completeness_score()* fore more information.

Returns A series of booleans with the same index as *series* with False up to the first complete day, True between the first and the last complete days, and False following the last complete day.

Return type Series

See also:

trim, *completeness_score*

Outliers

Functions for detecting outliers.

quality.outliers.tukey(data[, k])	Identify outliers based on the interquartile range.
quality.outliers.zscore(data[, zmax, nan_policy])	Identify outliers using the z-score.
quality.outliers.hampel(data[, window, ...])	Identify outliers by the Hampel identifier.

pvanalytics.quality.outliers.tukey

pvanalytics.quality.outliers.**tukey**(*data*, *k=1.5*)

Identify outliers based on the interquartile range.

A value x is considered an outlier if it does *not* satisfy the following condition

$$Q_1 - k(Q_3 - Q_1) \leq x \leq Q_3 + k(Q_3 - Q_1)$$

where Q_1 is the value of the first quartile and Q_3 is the value of the third quartile.

Parameters

- **data** (*Series*) – The data in which to find outliers.
- **k** (*float, default 1.5*) – Multiplier of the interquartile range. A larger value will be more permissive of values that are far from the median.

Returns A series of booleans with True for each value that is an outlier.

Return type Series

pvanalytics.quality.outliers.zscore

pvanalytics.quality.outliers.**zscore**(*data*, *zmax=1.5*, *nan_policy='raise'*)

Identify outliers using the z-score.

Points with z-score greater than *zmax* are considered as outliers.

Parameters

- **data** (*Series*) – A series of numeric values in which to find outliers.
- **zmax** (*float*) – Upper limit of the absolute values of the z-score.
- **nan_policy** (*{'raise', 'omit'}, default 'raise'*) – Define how to handle NaNs in the input series. If 'raise', a ValueError is raised when *data* contains NaNs. If 'omit', NaNs are ignored and False is returned at indices that contained NaN in *data*.

Returns A series of booleans with True for each value that is an outlier.

Return type Series

pvanalytics.quality.outliers.hampel

pvanalytics.quality.outliers.**hampel**(*data*, *window=5*, *max_deviation=3.0*, *scale=None*)

Identify outliers by the Hampel identifier.

The Hampel identifier is computed according to[1].

Parameters

- **data** (*Series*) – The data in which to find outliers.
- **window** (*int or offset, default 5*) – The size of the rolling window used to compute the Hampel identifier.
- **max_deviation** (*float, default 3.0*) – Any value with a Hampel identifier > *max_deviation* standard deviations from the median is considered an outlier.
- **scale** (*float, optional*) – Scale factor used to estimate the standard deviation as $MAD/scale$. If *scale=None* (default), then the scale factor is taken to be `scipy.stats.norm.ppf(3/4.)` (approx. 0.6745), and $MAD/scale$ approximates the standard deviation of Gaussian distributed data.

Returns True for each value that is an outlier according to its Hampel identifier.

Return type Series

[1] Pearson, R.K., Neuvo, Y., Astola, J. et al. Generalized Hampel Filters. EURASIP J. Adv. Signal Process. 2016, 87 (2016). https://doi.org/10.1186/s13634-016-0383-6

References

Time

Quality control related to time. This includes things like time-stamp spacing, time-shifts, and time zone validation.

quality.time.spacing(times, freq)	Check that the spacing between *times* conforms to *freq*.

pvanalytics.quality.time.spacing

pvanalytics.quality.time.**spacing**(*times*, *freq*)

Check that the spacing between *times* conforms to *freq*.

Parameters

- **times** (*DatetimeIndex*) –
- **freq** (*string or Timedelta*) – Expected frequency of *times*.

Returns True when the difference between one time and the time before it conforms to *freq*.

Return type Series

Notes

Copyright (c) 2019 SolarArbiter. See the file LICENSES/SOLARFORECASTARBITER_LICENSE at the top level directory of this distribution and at https://github.com/pvlib/pvanalytics/blob/master/LICENSES/SOLARFORECASTARBITER_LICENSE for more information.

Timestamp shifts, such as daylight savings, can be identified with the following functions.

quality.time.shifts_ruptures(event_times, ...)	Identify time shifts using the ruptures library.
quality.time.has_dst(events, tz[, window, ...])	Return True if *events* appears to have daylight savings shifts at the dates on which *tz* transitions to or from daylight savings time.

pvanalytics.quality.time.shifts_ruptures

pvanalytics.quality.time.**shifts_ruptures**(*event_times*, *reference_times*, *period_min=2*, *shift_min=15*, *round_up_from=None*, *prediction_penalty=13*)

Identify time shifts using the ruptures library.

Compares the event time in the expected time zone (*reference_times*) with the actual event time in *event_times*.

The Pelt changepoint detection method is applied to the difference between *event_times* and *reference_times*. For each period between change points the mode of the difference is rounded to a multiple of *shift_min* and returned as the time-shift for all days in that period.

Parameters

- **event_times** (*Series*) – Time of an event in minutes since midnight. Should be a time series of integers with a single value per day. Typically the time mid-way between sunrise and sunset.
- **reference_times** (*Series*) – Time of event in minutes since midnight for each day in the

expected timezone. For example, passing solar transit time in a fixed offset time zone can be used to detect daylight savings shifts when it is unknown whether or not *event_times* is in a fixed offset time zone.

- **period_min** (*int, default 2*) – Minimum number of days between shifts. Must be less than or equal to the number of days in *event_times*. [days]

  Increasing this parameter will make the result less sensitive to transient shifts. For example if your intent is to find and correct daylight savings time shifts passing *period_min=60* can give good results while excluding shorter periods that appear shifted.

- **shift_min** (*int, default 15*) – Minimum shift amount in minutes. All shifts are rounded to a multiple of *shift_min*. [minutes]

- **round_up_from** (*int, optional*) – The number of minutes greater than a multiple of *shift_min* for a shift to be rounded up. If a shift is less than *round_up_from* then it will be rounded towards 0. If not specified then the shift will be rounded up from *shift_min // 2*. Using a larger value will effectively make the shift detection more conservative as small variations will tend to be rounded to zero. [minutes]

- **prediction_penalty** (*int, default 13*) – Penalty used in assessing change points. See `ruptures.detection.Pelt.predict()` for more information.

Returns

- **shifted** (*Series*) – Boolean series indicating whether there appears to be a time shift on that day.

- **shift_amount** (*Series*) – Time shift in minutes for each day in *event_times*. These times can be used to shift the data into the same time zone as *reference_times*.

Raises `ValueError` – If the number of days in *event_times* is less than *period_min*.

Notes

Timestamped data from monitored PV systems may not always be localized to a consistent timezone. In some cases, data is timestamped with local time that may or may not be adjusted for daylight savings time transitions. This function helps detect issues of this sort, by detecting points where the time of some daily event (e.g. solar noon) changes significantly with respect to a reference time for the event. If the data's timestamps have not been adjusted for daylight savings transitions, the time of day at solar noon will change by roughly 60 minutes in the days before and after the transition.

To use this changepoint detection method to determine if your data's timestamps involve daylight savings transitions, first reduce your PV system data (irradiance or power) to a daily time series, with each point being the observed midday time in minutes. For example, if sunrise and sunset are inferred from the PV system data, the midday time can be inferred as the average of each day's sunrise and sunset time of day. To establish the expected midday time, calculate solar transit time in time of day.

Derived from the PVFleets QA project.

pvanalytics.quality.time.has_dst

pvanalytics.quality.time.**has_dst**(*events*, *tz*, *window=7*, *min_difference=45*, *missing='raise'*)

Return True if *events* appears to have daylight savings shifts at the dates on which *tz* transitions to or from daylight savings time.

The mean event time in minutes since midnight is calculated over the *window* days before and after the date of each daylight savings transition in *tz*. For each date, the two mean event times (before and after) are compared, and if the difference is greater than *min_difference* then a shift has occurred on that date.

Parameters

- **events** (*Series*) – Series with one timestamp for each day. The timestamp should correspond to an event that occurs at roughly the same time on each day. For example, you may pass sunrise, sunset, or solar transit time. *events* need not be localized.
- **tz** (*str*) – Name of a timezone that observes daylight savings and has the same or similar UTC offset as the expected time zone for *events*.
- **window** (*int, default 7*) – Number of days before and after the shift date to consider. When passing rounded timestamps in *events* it may be necessary to use a smaller window. [days]
- **min_difference** (*int, default 45*) – Minimum difference between the mean event time before the shift date and the mean event time after the event time. If the difference is greater than *min_difference* a shift has occurred on that date. [minutes]
- **missing** (*{'raise', 'warn'}, default 'raise'*) – Whether to raise an exception or issue a warning when there is no data at a transition date. Can be 'raise' or 'warn'. If 'warn' and there is no data adjacent to a transition date, False is returned for that date.

Returns Boolean Series with the same index as *events* True for dates that appear to have daylight savings transitions.

Return type Series

Raises **ValueError** – If there is no data in the *window* days before or after a shift date in *events*.

Utilities

The `quality.util` module contains general-purpose/utility functions for building your own quality checks.

quality.util.check_limits(val[, ...])	Check whether a value falls withing the given limits.
quality.util.daily_min(series, minimum[, ...])	Return True for data on days when the day's minimum exceeds *minimum*.

pvanalytics.quality.util.check_limits

pvanalytics.quality.util.**check_limits**(*val*, *lower_bound=None*, *upper_bound=None*, *inclusive_lower=False*, *inclusive_upper=False*)

Check whether a value falls withing the given limits.

At least one of *lower_bound* or *upper_bound* must be provided.

Parameters

- **val** (*array_like*) – Values to test.

- **lower_bound** (*float, default None*) – Lower limit.
- **upper_bound** (*float, default None*) – Upper limit.
- **inclusive_lower** (*bool, default False*) – Whether the lower bound is inclusive (*val* >= *lower_bound*).
- **inclusive_upper** (*bool, default False*) – Whether the upper bound is inclusive (*val* <= *upper_bound*).

Returns True for every value in *val* that is between *lower_bound* and *upper_bound*.

Return type array_like

Raises **ValueError** – if *lower_bound* nor *upper_bound* is provided.

Notes

pvanalytics.quality.util.daily_min

pvanalytics.quality.util.**daily_min**(*series*, *minimum*, *inclusive=False*)

Return True for data on days when the day's minimum exceeds *minimum*.

Parameters

- **series** (*Series*) – A Datetimeindexed series of floats.
- **minimum** (*float*) – The smallest acceptable value for the daily minimum.
- **inclusive** (*boolean, default False*) – Use greater than or equal to when comparing daily minimums from *series* to *minimum*.

Returns True for values on days where the minimum value recorded on that day is greater than (or equal to) *minimum*.

Return type Series

Notes

This function is derived from code in the pvfleets_qa_analysis project under the terms of the 3-clause BSD license. Copyright (c) 2020 Alliance for Sustainable Energy, LLC.

Weather

Quality checks for weather data.

quality.weather.relative_humidity_limits(...)	Identify relative humidity values that are within limits.
quality.weather.temperature_limits(...[, limits])	Identify temperature values that are within limits.
quality.weather.wind_limits(wind_speed[, limits])	Identify wind speed values that are within limits.

pvanalytics.quality.weather.relative_humidity_limits

pvanalytics.quality.weather.**relative_humidity_limits**(*relative_humidity*, *limits=(0, 100)*)
Identify relative humidity values that are within limits.

Parameters

- **relative_humidity** (*Series*) – Relative humidity in %.
- **limits** (*tuple, default (0, 100)*) – (lower bound, upper bound) for relative humidity.

Returns True if *relative_humidity* >= lower bound and *relative_humidity* <= upper_bound.

Return type Series

Notes

Copyright (c) 2019 SolarArbiter. See the file LICENSES/SOLARFORECASTARBITER_LICENSE at the top level directory of this distribution and at https://github.com/pvlib/pvanalytics/blob/master/LICENSES/SOLARFORECASTARBITER_LICENSE for more information.

pvanalytics.quality.weather.temperature_limits

pvanalytics.quality.weather.**temperature_limits**(*air_temperature*, *limits=(- 35.0, 50.0)*)
Identify temperature values that are within limits.

Parameters

- **air_temperature** (*Series*) – Air temperature [C].
- **limits** (*tuple, default (-35, 50)*) – (lower bound, upper bound) for temperature.

Returns True if *air_temperature* > lower bound and *air_temperature* < upper bound.

Return type Series

Notes

Copyright (c) 2019 SolarArbiter. See the file LICENSES/SOLARFORECASTARBITER_LICENSE at the top level directory of this distribution and at https://github.com/pvlib/pvanalytics/blob/master/LICENSES/SOLARFORECASTARBITER_LICENSE for more information.

pvanalytics.quality.weather.wind_limits

pvanalytics.quality.weather.**wind_limits**(*wind_speed*, *limits=(0.0, 50.0)*)
Identify wind speed values that are within limits.

Parameters

- **wind_speed** (*Series*) – Wind speed in m/s
- **limits** (*tuple, default (0, 50)*) – (lower bound, upper bound) for wind speed.

Returns True if *wind_speed* >= lower bound and *wind_speed* < upper bound.

Return type Series

Notes

In addition to validating temperature by comparing with limits, module temperature should be positively correlated with irradiance. Poor correlation could indicate that the sensor has become detached from the module, for example. Unlike other functions in the `quality` module which return Boolean masks over the input series, this function returns a single Boolean value indicating whether the entire series has passed (`True`) or failed (`False`) the quality check.

quality.weather.module_temperature_check(...)	Test whether the module temperature is correlated with irradiance.

pvanalytics.quality.weather.module_temperature_check

pvanalytics.quality.weather.**module_temperature_check**(*module_temperature*, *irradiance*, *correlation_min=0.5*)

Test whether the module temperature is correlated with irradiance.

Parameters

- **module_temperature** (*Series*) – Time series of module temperature.
- **irradiance** (*Series*) – Time series of irradiance with the same index as *module_temperature*. This should be of relatively high quality (outliers and other problems removed).
- **correlation_min** (*float, default 0.5*) – Minimum correlation between *module_temperature* and *irradiance* for the module temperature sensor to 'pass'

Returns True if the correlation between *module_temperature* and *irradiance* exceeds *correlation_min*.

Return type bool

References

3.1.2 Features

Functions for detecting features in the data.

Clipping

Functions for identifying inverter clipping

features.clipping.levels(ac_power[, window, ...])	Label clipping in AC power data based on levels in the data.
features.clipping.threshold(ac_power[, ...])	Detect clipping based on a maximum power threshold.
features.clipping.geometric(ac_power[, ...])	Identify clipping based on a the shape of the *ac_power* curve on each day.

pvanalytics.features.clipping.levels

pvanalytics.features.clipping.**levels**(*ac_power*, *window=4*, *fraction_in_window=0.75*, *rtol=0.005*, *levels=2*)

Label clipping in AC power data based on levels in the data.

Parameters

- **ac_power** (*Series*) – Time series of AC power measurements.
- **window** (*int, default 4*) – Number of data points in a window used to detect clipping.
- **fraction_in_window** (*float, default 0.75*) – Fraction of points which indicate clipping if AC power at each point is close to the plateau level.
- **rtol** (*float, default 5e-3*) – A point is close to a clipped level M if abs(ac_power - M) < rtol * max(ac_power)
- **levels** (*int, default 2*) – Number of clipped power levels to consider.

Returns True when clipping is indicated.

Return type Series

Notes

pvanalytics.features.clipping.threshold

pvanalytics.features.clipping.**threshold**(*ac_power*, *slope_max=0.0035*, *power_min=0.75*, *power_quantile=0.995*, *freq=None*)

Detect clipping based on a maximum power threshold.

This is a two-step process. First a clipping threshold is identified, then any values in *ac_power* greater than or equal to that threshold are flagged.

The clipping threshold is determined by computing a 'daily power curve' which is the *power_quantile* quantile of all values in *ac_power* at each minute of the day. This gives a rough estimate of the maximum power produced at each minute of the day.

The daily power curve is normalized by its maximum and the minutes of the day are identified where the normalized curve's slope is less than *slope_max*. If there is a continuous period of time spanning at least one hour where the slope is less than *slope_max* and the value of the normalized daily power curve is greater than *power_min* times the median of the normalized daily power curve then the data has clipping in it. If no sufficiently long period with both a low slope and high power exists then there is no clipping in the data. The average of the daily power curve (not normalized) during the longest period that satisfies the criteria above is the clipping threshold.

Parameters

- **ac_power** (*Series*) – DatetimeIndexed series of AC power data.
- **slope_max** (*float, default 0.0035*) – Maximum absolute value of slope of AC power quantile for clipping to be indicated. The default value has been derived empirically to prevent false positives for tracking PV systems.

- **power_min** (*float, default 0.75*) – The power during periods with slope less than *slope_max* must be greater than *power_min* times the median normalized daytime power.
- **power_quantile** (*float, default 0.995*) – Quantile used to calculate the daily power curve.
- **freq** (*string, default None*) – A pandas string offset giving the frequency of data in *ac_power*. If None then the frequency is inferred from the series index.

Returns True when *ac_power* is greater than or equal to the clipping threshold.

Return type Series

Notes

This function is based on the pvfleets_qa_analysis project.

pvanalytics.features.clipping.geometric

pvanalytics.features.clipping.**geometric**(*ac_power*, *window=None*, *slope_max=0.2*, *freq=None*, *tracking=False*)

Identify clipping based on a the shape of the *ac_power* curve on each day.

Each day is checked for periods where the slope of *ac_power* is small. The power values in these periods are used to calculate a minimum and a maximum clipped power level for that day. Any power values that are within this range are flagged as clipped. The methodology for computing the thresholds varies depending on the frequency of *ac_power*. For high frequency data (less than 10 minute timestamp spacing) the minimum clipped power is the mean of the low-slope period(s) on that day minus 2 times the standard deviation in the same period(s). For lower frequency data the absolute minimum and maximum of the low slope period(s) on each day are used.

If the frequency of *ac_power* is less than ten minutes, then *ac_power* is down-sampled to 15 minutes and the mean value in each 15-minute period is used to reduce noise inherent in high frequency data.

Parameters

- **ac_power** (*Series*) – AC power data.
- **window** (*int, optional*) – Size of the rolling window used to identify low-slope periods. If not specified and *tracking* is False then *window=3* is used. If not specified and *tracking* is True then *window=5* is used.
- **slope_max** (*float, default 0.2*) – Maximum difference in maximum and minimum power for a window to be flagged as clipped. Units are percent of average power in the interval.
- **freq** (*str, optional*) – Frequency of *ac_power*. If not specified then pandas.infer_freq() is used.
- **tracking** (*bool, default False*) – If True then a larger default *window* is used. If *window* is specified then *tracking* has no effect.

Returns Boolean Series with True for values that appear to be clipped.

Return type Series

Raises ValueError – If the index of *ac_power* is not sorted.

Notes

Based on code from the PVFleets QA project.

Clearsky

features.clearsky.reno(ghi, ghi_clearsky)	Identify times when GHI is consistent with clearsky conditions.

pvanalytics.features.clearsky.reno

pvanalytics.features.clearsky.**reno**(*ghi*, *ghi_clearsky*)

Identify times when GHI is consistent with clearsky conditions.

Uses the function `pvlib.clearsky.detect_clearsky()`.

Note: Must be given GHI data with regular (constant) time intervals of 15 minutes or less.

Parameters

- **ghi** (*Series*) – Global horizontal irradiance in W/m^2. Must have an index with time intervals of at most 15 minutes.
- **ghi_clearsky** (*Series*) – Global horizontal irradiance in W/m^2 under clearsky conditions.

Returns True when clear sky conditions are indicated.

Return type Series

Raises `ValueError` – if the time intervals are greater than 15 minutes.

Notes

Clear-sky conditions are inferred when each of six criteria are met; see `pvlib.clearsky.detect_clearsky()` for references and details. Threshold values for each criterion were originally developed for ten minute windows containing one-minute data[1]. As indicated in[2], the algorithm also works for longer windows and data at different intervals if threshold criteria are roughly scaled to the window length. Here, the threshold values are based on [1] with the scaling indicated in [2].

[1] Reno, M.J. and C.W. Hansen, "Identification of periods of clear sky irradiance in time series of GHI measurements" Renewable Energy, v90, p. 520-531, 2016.

[2] B. H. Ellis, M. Deceglie and A. Jain, "Automatic Detection of Clear-Sky Periods From Irradiance Data," in IEEE Journal of Photovoltaics, vol. 9, no. 4, pp. 998-1005, July 2019. doi: 10.1109/JPHOTOV.2019.2914444

References

Examples using pvanalytics.features.clearsky.reno

- *Clear-Sky Detection*

Orientation

System orientation refers to mounting type (fixed or tracker) and the azimuth and tilt of the mounting. A system's orientation can be determined by examining power or POA irradiance on days that are relatively sunny.

This module provides functions that operate on power or POA irradiance to identify system orientation on a daily basis. These functions can tell you whether a day's profile matches that of a fixed system or system with a single-axis tracker.

Care should be taken when interpreting function output since other factors such as malfunctioning trackers can interfere with identification.

features.orientation.fixed_nrel(...[, ...])	Flag days that match the profile of a fixed PV system on a sunny day.
features.orientation.tracking_nrel(...[, ...])	Flag days that match the profile of a single-axis tracking PV system on a sunny day.

pvanalytics.features.orientation.fixed_nrel

pvanalytics.features.orientation.**fixed_nrel**(*power_or_irradiance*, *daytime*, *r2_min=0.94*, *min_hours=5*, *peak_min=None*)

Flag days that match the profile of a fixed PV system on a sunny day.

This algorithm relies on the observation that the power profile of a fixed tilt PV system often resembles a quadratic polynomial on a sunny day, with a single peak when the sun is near the system azimuth.

A day is marked True when the r^2 for a quadratic fit to the power data is greater than *r2_min*.

Parameters

- **power_or_irradiance** (*Series*) – Timezone localized series of power or irradiance measurements.
- **daytime** (*Series*) – Boolean series with True for times that are during the day. For best results this mask should exclude early morning and evening as well as night. Data at these times may have problems with shadows that interfere with curve fitting.
- **r2_min** (*float, default 0.94*) – Minimum r^2 of a quadratic fit for a day to be marked True.
- **min_hours** (*float, default 5.0*) – Minimum number of hours with data to attempt a fit on a day.
- **peak_min** (*float, default None*) – The maximum *power_or_irradiance* value for a day must be greater than *peak_min* for a fit to be attempted. If the maximum for a day is less than *peak_min* then the day will be marked False.

Returns True for values on days where *power_or_irradiance* matches the expected parabolic profile for a fixed PV system on a sunny day.

Return type Series

Notes

This algorithm is based on the PVFleets QA Analysis project. Copyright (c) 2020 Alliance for Sustainable Energy, LLC.

pvanalytics.features.orientation.tracking_nrel

pvanalytics.features.orientation.**tracking_nrel**(*power_or_irradiance*, *daytime*, *r2_min=0.915*, *r2_fixed_max=0.96*, *min_hours=5*, *peak_min=None*, *quadratic_mask=None*)

Flag days that match the profile of a single-axis tracking PV system on a sunny day.

This algorithm relies on the observation that the power profile of a single-axis tracking PV system tends to resemble a quartic polynomial on a sunny day, I.e., two peaks are observed, one before and one after the sun crosses the tracker azimuth. By contrast, the power profile for a fixed tilt PV system often resembles a quadratic polynomial on a sunny day, with a single peak when the sun is near the system azimuth.

The algorithm fits both a quartic and a quadratic polynomial to each day's data. A day is marked True if the quartic fit has a sufficiently high r^2 and the quadratic fit has a sufficiently low r^2. Specifically, a day is marked True when three conditions are met:

1. a restricted quartic[1] must fit the data with r^2 greater than *r2_min*
2. the r^2 for the restricted quartic fit must be greater than the r^2 for a quadratic fit
3. the r^2 for a quadratic fit must be less than *r2_fixed_max*

Values on days where any one of these conditions is not met are marked False.

Parameters

- **power_or_irradiance** (*Series*) – Timezone localized series of power or irradiance measurements.
- **daytime** (*Series*) – Boolean series with True for times that are during the day. For best results this mask should exclude early morning and late afternoon as well as night. Data at these times may have problems with shadows that interfere with curve fitting.
- **r2_min** (*float, default 0.915*) – Minimum r^2 of a quartic fit for a day to be marked True.
- **r2_fixed_max** (*float, default 0.96*) – If the r^2 of a quadratic fit exceeds *r2_fixed_max*, then tracking/fixed cannot be distinguished and the day is marked False.
- **min_hours** (*float, default 5.0*) – Minimum number of hours with data to attempt a fit on a day.
- **peak_min** (*float, default None*) – The maximum *power_or_irradiance* value for a day must be greater than *peak_min* for a fit to be attempted. If the maximum for a day is less than *peak_min* then the day will be marked False.
- **quadratic_mask** (*Series, default None*) – If None then *daytime* is used. This Series is used to remove morning and afternoon times from the data before applying a quadratic fit. The mask should typically exclude more data than *daytime* in order to eliminate long tails in the morning or afternoon that can appear if a tracker is stuck in a West or East orientation.

Returns Boolean series with True for every value on a day that has a tracking profile (see criteria above).

[1] The specific quartic used for this fit is centered within 70 minutes of 12:00, the y-value at the center must be within 15% of the median for the day, and it must open downwards.

Return type Series

Notes

This algorithm is based on the PVFleets QA Analysis project. Copyright (c) 2020 Alliance for Sustainable Energy, LLC.

Daytime

Functions that return a Boolean mask indicating day and night.

features.daytime.power_or_irradiance(series)	Return True for values that are during the day.

pvanalytics.features.daytime.power_or_irradiance

pvanalytics.features.daytime.**power_or_irradiance**(*series*, *outliers=None*, *low_value_threshold=0.003*, *low_median_threshold=0.0015*, *low_diff_threshold=0.0005*, *median_days=7*, *clipping=None*, *freq=None*, *correction_window=31*, *hours_min=5*, *day_length_difference_max=30*, *day_length_window=14*)

Return True for values that are during the day.

After removing outliers and normalizing the data, a time is classified as night when two of the following three criteria are satisfied:

- near-zero value
- near-zero first-order derivative
- near-zero rolling median at the same time over the surrounding week (see *median_days*)

Mid-day times where power goes near zero or stops changing may be incorrectly classified as night. To correct these errors, night or day periods with duration that is too long or too short are identified, and times in these periods are re-classified to have the majority value at the same time on preceding and following days (as set by *correction_window*).

Finally any values that are True in *clipping* are marked as day.

Parameters

- **series** (*Series*) – Time series of power or irradiance.
- **outliers** (*Series, optional*) – Boolean time series with True for values in *series* that are outliers.
- **low_value_threshold** (*float, default 0.003*) – Maximum normalized power or irradiance value for a time to be considered night.
- **low_median_threshold** (*float, default 0.0015*) – Maximum rolling median of power or irradiance for a time to be considered night.
- **low_diff_threshold** (*float, default 0.0005*) – Maximum derivative of normalized power or irradiance for a time to be considered night.
- **median_days** (*int, default 7*) – Number of days to use to calculate the rolling median at each minute. [days]

- **clipping** (*Series, optional*) – True when clipping indicated. Any values where clipping is indicated are automatically considered 'daytime'.
- **freq** (*str, optional*) – A pandas freqstr specifying the expected timestamp spacing for the series. If None, the frequency will be inferred from the index.
- **correction_window** (*int, default 31*) – Number of adjacent days to examine when correcting day/night classification errors. [days]
- **hours_min** (*float, default 5*) – Minimum number of hours in a contiguous period of day or night. A day/night period shorter than *hours_min* is flagged for error correction. [hours]
- **day_length_difference_max** (*float, default 30*) – Days with length that is *day_length_difference_max* minutes less than the median length of surrounding days are flagged for corrections.
- **day_length_window** (*int, default 14*) – The length of the rolling window used for calculating the median length of the day when correcting errors in the morning or afternoon. [days]

Returns Boolean time series with True for times that are during the day.

Return type Series

Notes

`NA` values are treated like zeros.

Derived from the PVFleets QA Analysis project.

Shading

Functions for labeling shadows.

features.shading.fixed(ghi, daytime, clearsky)	Detects shadows from fixed structures such as wires and poles.

pvanalytics.features.shading.fixed

pvanalytics.features.shading.**fixed**(*ghi*, *daytime*, *clearsky*, *interval=None*, *min_gradient=2*)

Detects shadows from fixed structures such as wires and poles.

Uses morphological image processing methods to identify shadows from fixed local objects in GHI data. GHI data are assumed to be reasonably complete with relatively few missing values and at a fixed time interval nominally of 1 minute over the course of several months. Detection focuses on shadows with relatively short duration. The algorithm forms a 2D image of the GHI data by arranging time of day along the x-axis and day of year along the y-axis. Rapid change in GHI in the x-direction is used to identify edges of shadows; continuity in the y-direction is used to separate local object shading from cloud shadows.

Parameters

- **ghi** (*Series*) – Time series of GHI measurements. Data must be in local time at 1-minute frequency and should cover at least 60 days.
- **daytime** (*Series*) – Boolean series with True for times when the sun is up.

- **clearsky** (*Series*) – Clearsky GHI with same index as *ghi*.
- **interval** (*int, optional*) – Interval between data points in minutes. If not specified the interval is inferred from the frequency of the index of *ghi*.
- **min_gradient** (*float, default 2*) – Threshold value for the morphological gradient[3].

Returns

- *Series* – Boolean series with true for times that are impacted by shadows.
- *ndarray* – A boolean image (black and white) showing the shadows that were detected.

References

3.1.3 System

This module contains functions and classes relating to PV system parameters such as nameplate power, tilt, azimuth, or whether the system is equipped with tracker.

Tracking

system.Tracker(value)	Enum describing the orientation of a PV System.
system.is_tracking_envelope(series, daytime, ...)	Infer whether the system is equipped with a tracker.

pvanalytics.system.Tracker

class pvanalytics.system.**Tracker**(*value*)

Enum describing the orientation of a PV System.

Attributes

FIXED	A system with a fixed azimuth and tilt.
TRACKING	A system equipped with a tracker.
UNKNOWN	A system where the tracking cannot be determined.

pvanalytics.system.is_tracking_envelope

pvanalytics.system.**is_tracking_envelope**(*series*, *daytime*, *clipping*, *clip_max=0.1*, *envelope_quantile=0.995*, *envelope_min_fraction=0.05*, *fit_median=True*, *median_min_fraction=0.025*, *median_r2_min=0.9*, *fit_params=None*, *seasonal_split='north-america'*)

Infer whether the system is equipped with a tracker.

Data is grouped by season (optional) and within each season by the minute of the day. A maximum power or irradiance envelope (the *envelope_quantile* value at each minute) is calculated. Quadratic and quartic curves are fit to this daily envelope and the r^2 of the curve fits are used determine whether the system is tracking or fixed.

[3] https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.morphological_gradient.html

If the quadratic fit is a sufficiently good in both seasons, then `Tracker.FIXED` is returned.

If, in both seasons, the quartic fit is sufficiently good and the quadratic fit is sufficiently bad, then `Tracker.TRACKING` is returned.

If neither fit is sufficiently good, or the results from each season disagree, then `Tracker.UNKNOWN` is returned.

Optionally, an additional fit is made to the median of the data at each minute to confirm the determination of tracking or fixed. If performed, this result must be consistent with the fit to the upper envelope. If not, `Tracker.UNKNOWN` is returned.

Parameters

- **series** (*Series*) – Timezone localized Series of power or irradiance data.
- **daytime** (*Series*) – Boolean Series with True for times that are during the day.
- **clipping** (*Series*) – Boolean Series identifying where power or irradiance is being clipped.
- **clip_max** (*float, default 0.1*) – If the fraction of data flagged as clipped is greater than *clip_max* then it cannot be determined whether the system is tracking or fixed and `Tracker.UNKNOWN` is returned.
- **envelope_quantile** (*float, default 0.995*) – Quantile used to determine the upper power or irradiance envelope.
- **envelope_min_fraction** (*float, default 0.05*) – After calculating the power or irradiance envelope, data less than *envelope_min_fraction* times the maximum of the envelope is removed. This excludes data from morning and evening that may interfere with curve fitting.
- **fit_median** (*boolean, default True*) – Perform a secondary fit with the median power or irradiance to validate that the profile is consistent through the entire data set.
- **median_min_fraction** (*float, default 0.025*) – After calculating the median power or irradiance at each minute, data less than *median_min_fraction* times the maximum is removed. This excludes data from morning and evening that may interfere with curve fitting.
- **median_r2_min** (*float, default 0.9*) – Minimum r^2 for a curve fit to the median power or irradiance at each minute of the day (Applies only if *fit_median* is True).
- **fit_params** (*dict or None, default None*) – Minimum r-squared for curve fits according to the fraction of data with clipping. This should be a dictionary with tuple keys and dictionary values. The key must be a 2-tuple of `(clipping_min, clipping_max)` where the values specify the minimum and maximum fraction of data with clipping for which the associated fit parameters are applicable. The values of the dictionary are themselves dictionaries with keys `'fixed'` and `'tracking'`, which give the minimum r^2 for the curve fits, and `'fixed_max'` which gives the maximum r^2 for a quadratic fit if the system appears to have a tracker.

  If None `PVFLEETS_FIT_PARAMS` is used.
- **seasonal_split** (*dict or str or None, default 'north-america'*) – A dictionary with two keys, 'winter' and 'summer' with a list of integers specifying the winter months and summer months respectively. Seasonal grouping can be disabled by passing *seasonal_split=None*. Either season can be ignored by passing a dict that omits the key or sets its value to None. The default value, 'north-america' uses `{'winter': [11, 12, 1, 2], 'summer': [5, 6, 7, 8]}` which works well for PV systems located in North America.

Returns The tracking determined by curve fitting (FIXED, TRACKING, or UNKNOWN).

Return type *Tracker*

Notes

Derived from the PVFleets QA Analysis project.

See also:

pvanalytics.features.orientation.tracking_nrel, *pvanalytics.features.orientation.fixed_nrel*

Orientation

The following function can be used to infer system orientation from power or plane of array irradiance measurements.

system.infer_orientation_daily_peak(...)	Determine system azimuth and tilt from power or POA using solar azimuth at the daily peak.
system.infer_orientation_fit_pvwatts(...[, ...])	Get the tilt and azimuth that give PVWatts output that most closely fits the data in *power_ac*.

pvanalytics.system.infer_orientation_daily_peak

pvanalytics.system.**infer_orientation_daily_peak**(*power_or_poa*, *sunny*, *tilts*, *azimuths*, *solar_azimuth*, *solar_zenith*, *ghi*, *dhi*, *dni*)

Determine system azimuth and tilt from power or POA using solar azimuth at the daily peak.

The time of the daily peak is estimated by fitting a quadratic to to the data for each day in *power_or_poa* and finding the vertex of the fit. A brute force search is performed on clearsky POA irradiance for all pairs of candidate azimuths and tilts (*azimuths* and *tilts*) to find the pair that results in the closest azimuth to the azimuths calculated at the peak times from the curve fitting step. Closest is determined by minimizing the sum of squared difference between the solar azimuth at the peak time in *power_or_poa* and the solar azimuth at maximum clearsky POA irradiance.

The accuracy of the tilt and azimuth returned by this function will vary with the time-resolution of the clearsky and solar position data. For the best accuracy pass *solar_azimuth*, *solar_zenith*, and the clearsky data (*ghi*, *dhi*, and *dni*) with one-minute timestamp spacing. If *solar_azimuth* has timestamp spacing less than one minute it will be resampled and interpolated to estimate azimuth at each minute of the day. Regardless of the timestamp spacing these parameters must cover the same days as *power_or_poa*.

Parameters

- **power_or_poa** (*Series*) – Timezone localized series of power or POA irradiance measurements.
- **sunny** (*Series*) – Boolean series with True for values during clearsky conditions.
- **tilts** (*array-like*) – Candidate tilts in degrees.
- **azimuths** (*array-like*) – Candidate azimuths in degrees.
- **solar_azimuth** (*Series*) – Time series of solar azimuth.
- **solar_zenith** (*Series*) – Time series of solar zenith.
- **ghi** (*Series*) – Clear sky GHI.
- **dhi** (*Series*) – Clear sky DHI.
- **dni** (*Series*) – Clear sky DNI.

Returns

- **azimuth** (*float*)
- **tilt** (*float*)

Notes

Based on PVFleets QA project.

pvanalytics.system.infer_orientation_fit_pvwatts

pvanalytics.system.**infer_orientation_fit_pvwatts**(*power_ac*, *ghi*, *dhi*, *dni*, *solar_zenith*, *solar_azimuth*, *temperature=25*, *wind_speed=0*, *temperature_coefficient=- 0.004*, *temperature_model_parameters=None*)

Get the tilt and azimuth that give PVWatts output that most closely fits the data in *power_ac*.

Input data *power_ac*, *ghi*, *dhi*, *dni* should reflect clear-sky conditions.

Uses non-linear least squares to optimize over four free variables to find the values that result in the best fit between power modeled using PVWatts and *power_ac*. The four free variables are

- surface tilt
- surface azimuth
- the DC capacity of the system
- the DC input limit of the inverter.

Of these four parameters, only tilt and azimuth are returned. While, DC capacity and the DC input limit are calculated, their values may not be accurate. While its value is not returned, because the DC input limit is used as a free variable for the optimization process, this function can operate on *power_ac* data that includes inverter clipping.

All parameters passed as a Series must have the same index and must not contain any undefined values (i.e. NaNs). If any input contains NaNs a ValueError is raised.

Parameters

- **power_ac** (*Series*) – AC power from the system in clear sky conditions.
- **ghi** (*Series*) – Clear sky GHI with same index as *power_ac*. [W/m^2]
- **dhi** (*Series*) – Clear sky DHI with same index as *power_ac*. [W/m^2]
- **dni** (*Series*) – Clear sky DNI with same index as *power_ac*. [W/m^2]
- **solar_zenith** (*Series*) – Solar zenith. [degrees]
- **solar_azimuth** (*Series*) – Solar azimuth. [degrees]
- **temperature** (*float or Series, default 25*) – Air temperature at which to model the hypothetical system. If a float then a constant temperature is used. If a Series, must have the same index as *power_ac*. [C]
- **wind_speed** (*float or Series, default 0*) – Wind speed. If a float then a constant wind speed is used. If a Series, must have the same index as *power_ac*. [m/s]
- **temperature_coefficient** (*float, default -0.004*) – Temperature coefficient of DC power. [1/C]

- **temperature_model_parameters** (*dict, optional*) – Parameters fot the cell temperature model. If not specified `pvlib.temperature.TEMPERATURE_MODEL_PARAMETERS['sapm'][ 'open_rack_glass_glass']` is used. See `pvlib.temperature.sapm_cell()` for more information.

Returns

- **surface_tilt** (*float*) – Tilt of the array. [degrees]
- **surface_azimuth** (*float*) – Azimuth of the array. [degrees]
- **r_squared** (*float*) – r^2 value for the fit at the returned orientation.

Raises `ValueError` – If any input passed as a Series contains undefined values (i.e. NaNs).

3.1.4 Metrics

Performance Ratio

The following functions can be used to calculate system performance metrics.

metrics.performance_ratio_nrel(poa_global, ...)	Calculate NREL Performance Ratio.

pvanalytics.metrics.performance_ratio_nrel

pvanalytics.metrics.**performance_ratio_nrel**(*poa_global*, *temp_air*, *wind_speed*, *pac*, *pdc0*, *a=- 3.56*, *b=- 0.075*, *deltaT=3*, *gamma_pdc=- 0.00433*)

Calculate NREL Performance Ratio.

See equation [5] in Weather-Corrected Performance Ratio[1] for details on the weighted method for Tref.

Parameters

- **poa_global** (*numeric*) – Total incident irradiance [W/m^2].
- **temp_air** (*numeric*) – Ambient dry bulb temperature [C].
- **wind_speed** (*numeric*) – Wind speed at a height of 10 meters [m/s].
- **pac** (*float*) – AC power [kW].
- **pdc0** (*float*) – Power of the modules at 1000 W/m2 and cell reference temperature [kW].
- **a** (*float*) – Parameter a in SAPM model [unitless].
- **b** (*float*) – Parameter b in in SAPM model [s/m].
- **deltaT** (*float*) – Parameter ΔT in SAPM model [C].
- **gamma_pdc** (*float*) – The temperature coefficient in units of 1/C. Typically -0.002 to -0.005 per degree C [1/C].

Returns **performance_ratio** – Performance Ratio of data.

Return type float

[1] Dierauf et al. "Weather-Corrected Performance Ratio". NREL, 2013. https://www.nrel.gov/docs/fy13osti/57991.pdf

References

Variability

Functions to calculate variability statistics.

metrics.variability_index(measured, clearsky)	Calculate the variability index.

pvanalytics.metrics.variability_index

pvanalytics.metrics.**variability_index**(*measured*, *clearsky*, *freq=None*)

Calculate the variability index.

Parameters

- **measured** (*Series*) – Time series of measured GHI. [W/m2]
- **clearsky** (*Series*) – Time series of the expected clearsky GHI. [W/m2]
- **freq** (*pandas datetime offset, optional*) – Aggregation period (e.g. 'D' for daily). If not specified, the variability index for the entire time series will be returned.

Returns **vi** – The calculated variability index

Return type Series or float

References

3.2 Example Gallery

This gallery shows examples of pvanalytics functionality. Community contributions are welcome!

3.2.1 Clear-Sky Detection

Identifying periods of clear-sky conditions using measured irradiance.

Identifying and filtering for clear-sky conditions is a useful way to reduce noise when analyzing measured data. This example shows how to use *pvanalytics.features.clearsky.reno()* to identify clear-sky conditions using measured GHI data. For this example we'll use GHI measurements from NREL in Golden CO.

```
import pvanalytics
from pvanalytics.features.clearsky import reno
import pvlib
import matplotlib.pyplot as plt
import pandas as pd
import pathlib
```

First, read in the GHI measurements. For this example we'll use an example file included in pvanalytics covering a single day, but the same process applies to data of any length.

```
pvanalytics_dir = pathlib.Path(pvanalytics.__file__).parent
ghi_file = pvanalytics_dir / 'data' / 'midc_bms_ghi_20220120.csv'
data = pd.read_csv(ghi_file, index_col=0, parse_dates=True)
```

(continues on next page)

(continued from previous page)

```
# or you can fetch the data straight from the source using pvlib:
# date = pd.to_datetime('2022-01-20')
# data = pvlib.iotools.read_midc_raw_data_from_nrel('BMS', date, date)

measured_ghi = data['Global CMP22 (vent/cor) [W/m^2]']
```

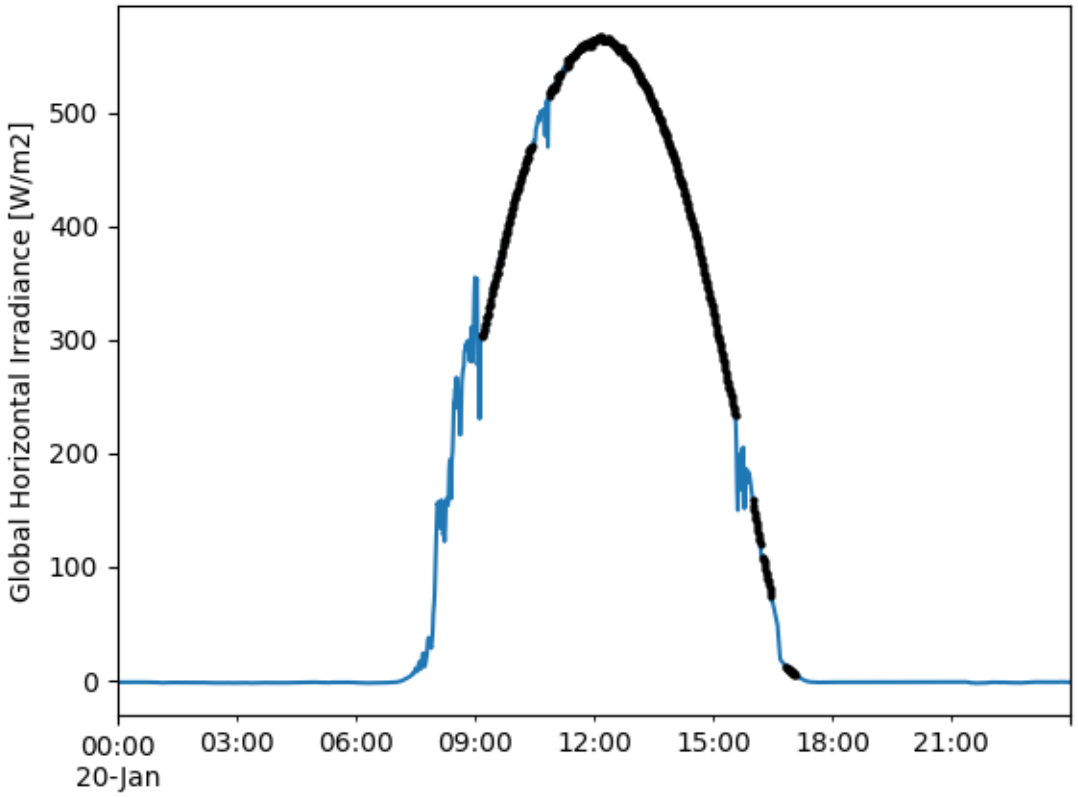
Now model clear-sky irradiance for the location and times of the measured data:

```
location = pvlib.location.Location(39.742, -105.18)
clearsky = location.get_clearsky(data.index)
clearsky_ghi = clearsky['ghi']
```

Finally, use *pvanalytics.features.clearsky.reno()* to identify measurements during clear-sky conditions:

```
is_clearsky = reno(measured_ghi, clearsky_ghi)

# clear-sky times indicated in black
measured_ghi.plot()
measured_ghi[is_clearsky].plot(ls='', marker='o', ms=2, c='k')
plt.ylabel('Global Horizontal Irradiance [W/m2]')
plt.show()
```



Total running time of the script: (0 minutes 0.302 seconds)

3.3 Release Notes

These are the bug-fixes, new features, and improvements for each release.

3.3.1 0.1.1 (February 18, 2022)

Enhancements

- Quantification of irradiance variability with *pvanalytics.metrics.variability_index()*. (GH60, GH106)
- Internal refactor of *pvanalytics.metrics.performance_ratio_nrel()* to support other performance ratio formulas. (GH109)
- Detect shadows from fixed objects in GHI data using *pvanalytics.features.shading.fixed()*. (GH24, GH101)

Bug Fixes

- Added `nan_policy` parameter to zscore calculation in *pvanalytics.quality.outliers.zscore()*. (GH102, GH108)
- Prohibit pandas versions in the 1.1.x series to avoid an issue in `.groupby().rolling()`. Newer versions starting in 1.2.0 and older versions going back to 0.24.0 are still allowed. (GH82, GH118)
- Fixed an issue with *pvanalytics.features.clearsky.reno()* in recent pandas versions (GH125, GH128)
- Improved convergence in *pvanalytics.features.orientation.fixed_nrel()* (GH119, GH120)

Requirements

- Drop support for python 3.6, which reached end of life Dec 2021 (GH129)

Documentation

- Started an example gallery and added an example for *pvanalytics.features.clearsky.reno()* (GH125, GH127)

Contributors

- Kevin Anderson (@kanderso-nrel)
- Cliff Hansen (@cwhanse)
- Will Vining (@wfvining)
- Kirsten Perry (@kperrynrel)
- Michael Hopwood (@MichaelHopwood)
- Carlos Silva (@camsilva)
- Ben Taylor (@bt-)

3.3.2 0.1.0 (November 20, 2020)

This is the first release of PVAnalytics. As such, the list of "changes" below is not specific. Future releases will describe specific changes here along with references to the relevant github issue and pull requests.

API Changes

Enhancements

- Quality control functions for irradiance, weather and time series data. See `pvanalytics.quality` for content.
- Feature labeling functions for clipping, clearsky, daytime, and orientation. See `pvanalytics.features` for content.
- System parameter inference for tilt, azimuth, and whether the system is tracking or fixed. See `pvanalytics.system` for content.
- NREL performance ratio metric (*pvanalytics.metrics.performance_ratio_nrel()*).

Bug Fixes

Contributors

- Will Vining (@wfvining)
- Cliff Hansen (@cwhanse)
- Saurabh Aneja (@spaneja)

Special thanks to Matt Muller and Kirsten Perry of NREL for their assistance in adapting components from the PVFleets QA project to PVAnalytics.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- search

INDEX

V

W

Z