Task 19 Ice Loss

Release 2.2

IEA Wind Task 19

Nov 07, 2019

CONTENTS:

CHAPTER

ONE

DISCLAIMER

The Software is provided “AS IS” and VTT makes no representations or warranties of any kind with respect to the Software or proprietary rights, whether express or implied, including, but not limited to merchantability, fitness for a particular purpose and non-infringement of third party rights such as copyrights, trade secrets or any patent. VTT shall have no liability whatsoever for the use of the Software or any output obtained through the use of the Software by you.

The software requires using following third-party products. The licenses of these products can be found from links below.

Python: https://docs.python.org/3/license.html

NumPy: http://www.numpy.org/license.html

SciPy: https://www.scipy.org/scipylib/license.html

Matplotlib: http://matplotlib.org/users/license.html

CHAPTER

TWO

LICENSE

CHAPTER

THREE

TASK 19 ICE LOSS METHOD

3.1 Description

This document describes a method to assess production losses due to icing based on standard SCADA data available from modern wind turbines. An open source Python code will be publically available based on the method presented in this guideline document. This method is formulated by IEA Task 19 , an international expert group with an aim to increase and disseminate knowledge and information about cold climate wind energy related issues. Task 19 aims to contribute in lowering the costs and reducing the risks of wind energy deployment in cold climates.

3.2 Motivation

Currently production losses due to wind turbine rotor icing are calculated with different methods all resulting to different results. Task 19 has three main reasons on why a standardized production loss calculation method is needed:

1. There is a large need to compare different sites with each other with a systematic analysis method
2. To validate the IEA Ice Classification
3. Evaluate effectiveness of various blade heating systems

Current production loss methods usually use a constant -15% or -25% clean power curve drop as an indication of icing. Similarly standard deviation (or multiples of it) has been widely used to define iced turbine production losses. Both of these methods result to different results and are not necessarily representing the actual ice build-up and removal process to wind turbine blades reliably enough.

With the method described here, anyone with access to SCADA data from wind turbines can assess and calculate turbine specific production losses due to icing. The method uses existing standards and is developed in order to minimize the uncertainties related to production loss estimations from SCADA data. The method does not require icing measurements as input.

3.3 Method

Task 19 proposes a method that is robust, easily adaptable, filters outliers automatically and does not assume a normal distribution of the SCADA data for individual turbines and wind farms. The proposed method uses percentiles of the reference, non-iced power curve in combination with temperature measurements. Ice build-up on turbine blades gradually deteriorates the power output (or results to overproduction to to iced anemometer) so for increased accuracy the method uses three consecutive 10-minute data points for defining start-stop timestamps for icing events. In other words, the turbine rotor is used as an ice detector. Iced turbine power losses are defined by comparing the performance to the calculated power curve using heated anemometers from nacelle and the measured reference, expected power curve. Production losses are separated into 2 categories: operation and standstill losses due to icing.

On a general level, the method can be divided into 3 main steps:

1. Calculate reference, non-iced power curve
2. Calculate start-stop timestamps for different icing event classes
3. Calculate production losses due to icing

Below is a minimal list of signals used to calculate the icing losses

Signal	Description	Unit	Value
ws	Hub height wind speed	m/s	10-minute mean
temp	**Ambient temperature** (hub height)	°C	10-minute mean
pwr mean	Turbine output power	kW	10-minute mean
mode	Turbine operational mode		10-minute mean

3.4 Step 1: Calculate reference, non-iced power curve

This is the first and very important step in defining the production losses due to icing as one always needs to compare iced rotor performance to reference, non-iced operational values. All iced turbine production losses (operational or stand-still related) will be compared to what the turbine could produce during (and after) icing events.

Air density is to be corrected to hub height according to ISO 2533 by scaling the wind speed and air pressure by taking site elevation above sea level. As air pressure measurements are typically missing from turbine SCADA, a static pressure according to site elevation above sea level is calculated[1]. Site air density and air pressure are used to calibrate the nacelle wind speed as follows:

$$w_{site} = w_{std} \times \left(\frac{\rho_{std}}{\rho_{site}}\right)^{\frac{1}{3}} = w_{std} \times \left(\frac{\frac{P_{std}}{T_{std}}}{\frac{P_{site}}{T_{site}}}\right)^{\frac{1}{3}}$$

$$w_{site} = w_{std} \times \left(\frac{T_{std}}{T_{site}}(1 - 2.25577 \times 10^{-5} \times h)^{5.25588}\right)^{\frac{1}{3}}$$

where w_{site} is calibrated nacelle wind speed, w_{std} measured nacelle wind speed, T_{site} is nacelle temperature[2] and T_{std} is standard temperature of 15°C (288.15 K) resulting to air density of 1.225 kg/m3 at sea level P_{std} = 101325 Pa ambient air pressure, h is site elevation in meters above sea level[3].

The IEC 61400-12-1 "Power performance measurements" is applicable for very detailed power production calculations using a standard met mast wind measurements as input. However, the method for production loss calculation using SCADA data only results to using nacelle top wind measurements which are disturbed by the rotating rotor. The nacelle anemometer is thus less accurate and simplified binning of the reference data is proposed. As a first step, reference turbine data needs to be temperature and air pressure corrected and filtered according to production mode[4] as follows:

- air density and static air pressure correction with nacelle temperature and site elevation

[1] Alternatively, detailed weather model air pressure values can used. Of course if air pressure is measured, that is the preferred alternative

[2] Warning: Some nacelle temperature sensors have shown a constant bias of +2…3 °C due to radiation heat of nacelle. Investigating this bias is recommended (compare to met mast, weather models etc)

[3] Engineering ToolBox, (2003). Altitude above Sea Level and Air Pressure. [online] Available at: https://www.engineeringtoolbox.com/air-altitude-pressure-d_462.html

[4] Alternatively, if controller mode is not available or known, use following filter criterias: IF P_{min} > 0.005 P_{rated} AND P_{mean} > 0.05 P_{rated} THEN Power production mode = normal

- power production operating states only AND temperature[5] > +3°C

For power curve calculation the data is binned according to wind speed and optionally according to wind direction as well. Usefulness of wind direction-based binning depends on the site geography and is not always necessary. A separate power curve is then calculated for each wind direction bin.

The power curve calculation results in several for each wind speed bin:

1. Median output power in the bin
2. Standard deviation of power in the bin
3. 10th percentile of power in the bin
4. 90th percentile of power in the bin
5. Power curve uncertainty in the bin defined as [standard deviation] / [power]
6. Sample count in the bin

Sample count can be used to determine the appropriate binning resolution, it is recommended to have at least 6 hours of data in abin to get a representative result.

The code will also try to interpolate over empty bins or bins that have too few samples in them. In these cases a linear interpolation between two closest bins is used.

3.5 Step 2: Calculate start-stop timestamps for different icing event classes

Once the reference has been established, next the exact time periods when ice is present on the turbine rotor are needed. As only SCADA data is used as input to define icing events, special care needs to take place in order to minimize false icing event alarms. False iced rotor alarms are minimized by assuming that ice is affecting the rotor for 30 minutes or more consecutively at below 0°C temperatures. The required output power reduction (or over production) uses a certain percentile of the reference data. This enables a robust yet simple threshold.

In total, there are three different icing event classes detected from the SCADA data:

1: Decreased production ,icing event class a), shortly IEa 2: Standstill icing event class b), IEb 3: Iced up heated anemometer ws or overproduction icing event class c), IEc)

In addition to these, if blade heating system is available, the moments when blade heating is on can be categorized separately and if ice detector is available, icing events detected by the ice detector can be categorized separately.

3.5.1 Icing event class a)

The start of a typical reduced power output icing event class a) [IEa] for an operational turbine is defined as follows:

> If temp is below 0°C AND power is below 10th percentile of the respective reference (non-iced) wind bin for 30 minutes or more, THEN icing event class a) starts

An icing event class a) ends as follows:

> If power is above 10th percentile of the respective reference wind bin for 30-min or more, THEN icing event class a) ends

In the output files icing event class a is referenced as *Production losses due to icing*

[5] This temperature limit needs to be set high enough to assume that turbine is not influenced by icing at these temperatures

3.5.2 Icing event class b)

Icing can cause the turbine to shut-down and cause the turbine to standstill for a number of reasons. Standstill due to icing caused by icing event class b) [IEb] begins as follows:

> If temp is below 0°C AND power is below 10th percentile of the respective reference wind bin for 10-min resulting to a shutdown (power < 0.5 % of rated power of the turbine for at least 20-min, THEN standstill due to icing starts

Icing event b) ends as follows:

> If power is above 10th percentile of the respective reference wind bin for 30-min or more, THEN icing event class b) ends

3.5.3 Manual analysis of shut-downs in wintertime

Sometimes the turbine controller shuts down the turbine due to safety reasons during iced turbine operation even before power P10-P90 thresholds are exceeded. Different turbine types react very differently to icing of the rotor during operation. Some turbine types are very sensitive to rotor icing and thus shut-down very quickly after icing influences the rotor. Other turbine models are extremely robust and are able to operate with iced blades for long periods even under severe icing conditions. Manual analysis of standstill losses is recommended because standstill losses are typically larger than operational losses and analysing operational losses only underestimates production losses due to icing.

Typical shut -down controller error messages report excess tower side-to-side vibrations or that the nacelle wind speed does not correspond to output power. This type of behaviour can be considered to be caused by icing and is to be manually added when summing up all production losses.

It is possible to define certain SCADA status codes to represent a stopped turbine and calculate the losses caused by these stops separately from all other production losses. This can be useful in some cases to understand the distribution of production losses into different categories.

3.5.4 Icing event class c)

The heated anemometer ws may sometimes be influenced by ice resulting to overproduction. The start of an overproduction (iced up anemometer) icing event class c) [IEc] for an operational turbine is as follows:

> If temp is below 0°C AND power is above 90th percentile of the respective reference wind bin for 30-min or more, THEN icing event class c) starts

Icing event class c) ends as follows:

> If power is below 90th percentile of the respective reference wind bin for 30-min or more, THEN icing event class c) ends

For IEa and IEb, the production losses can be defined. However, if the measured output power is above expected wind speed (ie overproducing) in IEc, there is reason to expect the anemometer is influenced by ice and for this case, the production losses cannot be defined unless accurate wind speed are available from another source. If the number of hours with IEc is large, the estimated total production losses can be considered as minimum losses because all icing influences cannot be assessed.

3.5.5 Step 3: Calculate production losses due to icing

Once the icing events have been identified the difference in power between the reference and actual measured output power will be calulated for each time step during the icing events. In addition to this a production losses in kWh and

as a percentage of total ar calculated for ice event classes IEa and IEb. For overproduction (class IEc) only the total duration is documented.

The output of the method and the formatting of the results are described in the usage section of the documentation

CHAPTER

FOUR

USING THE TASK 19 POWER LOSS COUNTER

The project contains sample code to calculate icing induced data losses for a single wind turbine according to the specification by IEA Wind Task 19.

4.1 Installation

Code is written in Python 3 and it uses several libraries included in the Scipy stack.

Needed libraries are:

```
numpy, scipy, matplotlib, (sphinx)
```

Information on installing the scipy stack can be found from the Scipy website

Sphinx is needed to build the documentation. It's not mandatory, the release package includes a compiled documentation.

Note that you will need python 3 versions of the libraries.

Easiest way to get everything is to use a prepackaged installer such as Anaconda

Anaconda installer contains all the required Python libraries and more and is provided as one easy to use installer. It is free and open source, available for Windows/Mac OS X/Linux and is self contained i.e. can be installed side by side with an existing python installation.

4.2 Using the code

The code can be used to calculate losses and it will output several kinds of statistics about your data for later analysis.

The production loss calculator is configured by setting up a config file. (see section The .ini file). Then calling the script `t19_counter.py` by giving the ini file as a command line parameter as

```
python t19_counter.py site.ini
```

where `site.ini` contains the case definition relevant for your site.

4.3 Input data

The code is meant to calculate losses from one time series at a time. One time series in this case means one wind turbine. For multiple turbines you need to define a separate .ini file for each wind turbine and calculate losses separately. After this you have to use other tools to combine individual turbines to each other.

4.3.1 Example input data

Below is an example of what input data could look like. Notice the Status and fault code columns on the right. In this case there needs to be additional filtering to replace the status and fault codes with numerical values.

Time stamp (0)	Temperature (1)	RPM (2)	Wind Speed (3)	Power(4)	Direction(5)	Status(6)	Fault code(7)
2013-02-01 17:20	-3.14	10.2	7.5988	1277.235	133	Run	OK
2013-02-01 17:30	-3.80	10.8	7.6623	1235.741	132	Run	OK
2013-02-01 17:40	-3.23	10.9	7.5914	1297.725	134	Run	OK
2013-02-01 17:50	-3.57	10.5	7.9407	1227.176	130	Run	OK
2013-02-01 18:00	-3.79	10.9	7.8154	1256.481	132	Run	OK
2013-02-01 18:10	-3.73	10.9	7.6261	1274.133	132	Run	OK
2013-02-01 18:20	-3.63	10.8	7.3955	1249.529	136	Run	OK
2013-02-01 18:30	-3.87	10.9	7.691	1232.532	137	Run	OK
2013-02-01 18:40	-3.29	10.3	7.7816	1270.953	135	Run	OK
2013-02-01 18:50	-3.52	10.6	7.9739	1299.535	135	Run	OK
2013-02-01 19:00	-3.15	10.9	7.3878	1221.514	131	Run	OK
2013-02-01 19:10	-3.34	10.9	7.8072	1256.669	131	Run	OK
2013-02-01 19:20	-3.15	10.8	7.7349	1284.479	134	Run	OK
2013-02-01 19:30	-3.08	10.3	7.8621	1288.962	135	Run	OK
2013-02-01 19:40	-3.13	10.7	7.4672	1230.259	133	Run	OK
2013-02-01 19:50	-3.48	10.9	7.509	1279.426	138	Run	OK
2013-02-01 20:00	-3.34	10.5	7.9378	1239.045	139	Run	OK
2013-02-01 20:10	-3.02	10.3	7.1774	1273.976	132	Run	OK
2013-02-01 20:20	-3.50	10.5	7.3004	1254.343	136	Run	OK
2013-02-01 20:30	-3.47	10.7	7.7331	1278.701	131	Run	OK
2013-02-01 20:40	-3.53	10.7	7.6289	1269.522	134	Run	OK
2013-02-01 20:50	-3.04	10.1	7.0893	1296.482	135	Run	OK
2013-02-01 21:00	-3.31	10.7	7.8652	1278.775	133	Run	OK
2013-02-01 21:10	-3.44	10.7	7.6277	1232.615	134	Run	OK
2013-02-01 21:20	-3.46	10.5	7.9821	1219.4	135	Run	OK
2013-02-01 21:30	-3.80	10.7	7.5614	1280.438	132	Run	OK
2013-02-01 21:40	-3.26	10.7	7.2718	1253.659	136	Run	OK
2013-02-01 21:50	-3.75	10.5	7.6549	0	137	Fault	Fault code A
2013-02-01 22:00	-3.15	10.8	7.6856	0	133	Fault	Fault code A
2013-02-01 22:10	-3.89	10.7	7.8238	0	135	Fault	Fault code A
2013-02-01 22:20	-3.80	10.4	7.1408	0	133	Fault	Fault code A
2013-02-01 22:30	-3.86	10.4	7.1721	0	133	Fault	Fault code A
2013-02-01 22:40	-3.04	10.1	7.6194	0	136	Fault	Fault code A

4.4 Outputs

There are multiple different outputs available.

4.4.1 Summary file

Summary file that contains some statistics about the data. A useful tool to get an overview of the data and some statistics

Contains the following information.

Value Field name	Purpose
Dataset name	Data set name as defined in the config file
Production losses due to icing	Production losses during operation, that are classified to be icing related, in kWh
Relative production losses due to icing	Previous line's losses as % of reference
Losses due to icing related stops	Losses due to stops during operation that are classified to be icing related
Relative losses due to icing related stops	Previous line's losses as % of reference
Icing during production	Icing time in hours during production. Same definition of icing as on row 2
Icing during production (% of total data)	Previous line's value as % of the entire dataset
Turbine stopped during production	Amount of time turbine is stopped due to icing. Same definition of stop as "icing related stops" above
Turbine stopped production (% of total data)	Previous line's value as % of the entire dataset
Over production hours	Amount of time in hours the production is above P90 curve and temperature is below the alarm limit
Over production hours (% of total)	Previous line's value as % of the entire dataset
IPS on hours	Number of hours blade heating is on. (Will only appear in summary if the site in question has IPS)
IPS on hours (% of total)	Previous line's value as % of the entire dataset
Losses during IPS operation	Sum of production losses during the times IPS is operating. The loss here is difference between reference and actual value, IPS self consumption is not taken into account. (Will only appear in summary if the site in question has IPS).
Relative losses during IPS operation	Previous line's losses as % of reference
IPS self consumption	If there is an IPS power consumption value in the source data, IPS self consumption in kWh, will show up here
IPS self consumption (% of total)	Previous line's losses as % of reference
SCADA forced stops	Number of hours the turbine is stopped due to some reason as indicated by the SCADA status code
Time Based Availability (TBA)	Percentage of the time the turbine is operating normally
Loss during SCADA stops	Production loss during the times turbine is not operating in kWh
Relative losses during SCADA stops (% of total)	Previous line's losses as % of reference
Power curve uncertainty	Average of power curve uncertainty (calculated only for bins between 4 m/s and 15 m/s)
Production upper limit (std.dev)	Upper limit for the production using power curve uncertainty above

Continued on next page

Table 2 – continued from previous page

Value Field name	Purpose
Production lower limit (std.dev)	Lower limit for the production using power curve uncertainty above
Production P90	Production estimate using the P90 power curve
Production P10	Production estimate using the P10 power curve
Theoretical mean production	Production assuming the reference power curve, using the wind speed measurement in file, not taking turbine state into account
Observed power production	Total production calculated from the output power column
Total Losses	Observed power - Theoretical mean power
Energy Based Availability (EBA)	Observed Power / Theoretical mean power as %
Data start time	First time stamp used for analysis
Data stop time	Last time stamp used for analysis
Total amount of data	difference between start and stop time in hours
Reference data start time	First time stamp in data
Reference data stop time	Last time stamp in data
Total amount of data in reference dataset	difference between start and stop time in reference data hours
Data availability	% of data available between first and last timestamp
Sample count in original data	Sample count in the dataset that is read in at first stage
Sample count in after filtering	Sample count after all filtering steps
Data loss due to filtering	Amount of data lost during filtering
Sample count in reference data	Sample count in reference data, used to build the reference power curve
Reference dataset as % of original data	reference dataset size as % of original

4.4.2 data time series

Prints a time series data as a .csv file that can be used for further analysis. Data is formatted as columns

timestamp, alarm, wind speed, reference power, temperature, power, limit

Here **alarm** indicates possible icing events. Alarm codes in this data are

0. no alarm
1. icing during production. Reduced power output
2. Turbine stopped due to icing
3. Overproduction. The turbine output is above the power curve.

reference power is power calculated from the power curve. Limit is the P10 limit used to identify reduced power output. Timestamp, wind speed and output power are drawn from the source data.

4.4.3 Power curve

Produces one file, that contains individual power curves for each wind direction bin.

The power curve is output as a table in a text file where different wind speed bins are in each row of the table and different columns indicate different wind direction bins. The row and column headers contain the center points of all bins.

The file contains the following variables binned for wind speed and direction:

- Mean power in the bin
- P10 value of the bin
- P90 value of the bin
- Bin power standard deviation
- Power curve uncertainty in the bin
- Power curve upper and lower limits (mean power +- uncertainty)
- Sample count in the bin

4.4.4 plot

Creates two interactive plots that can be used to look at the data. One contains full time series of the data with icing events marked on the timeline. Other contains the power curve and a scatter plot of the full time series with icing events marked on the data.

4.4.5 icing event list

It is possible to output a collected summary of icing events. This is output into two separate files. One that contains a list of all cases where the power output was reduced according to the set conditions and a another one listing all the icing induced stops. both files are text .csv files that containing the fields:

starttime	stoptime	loss_sum	event_length

Here `loss_sum` is the total losses during the event in kilowatt hours and `event_length` is the total length of said individual event in hours.

4.4.6 filtered time series

Produces the raw time series that is used after initial filtering to perform all calculations. Can be used for further analysis to get a common starting point.

4.5 The .ini file

All configuration is done in the .ini file.

Options are denoted in the file as:

```
name of option = value
```

File is divided into sections, section headers are enclosed in square brackets [].

Capitalization of sections and options is important, they need to be spelled the same way as in the example file.

Not all options are needed. Some variables have a preset default value that does not need to be set. A minimal .inifile is included with the release

4.6 Config file sections

The file is divided into Five logical sections that set certain parameters that will change from site to site and between runs.

Contents of each section are listed below and the purpose of all options is explained briefly.

4.6.1 Section: Source File

id

Identifier for the data set. This can be for example the name of the site or a combinations of site name and turbine identifier. **id** is used for example in naming the output files. **id** needs to be unique, if output files with the same identifier exist in the result directory the script will overwrite them. **id** is a mandatory value.

filename

the source data filename and path. The source data needs to be in a `.csv` file. Or any other kind of text file.

delimiter

field delimiter in the source file. If data is tab-delimited write `TAB` here. Default value is `,`.

quotechar

Character used to indicate text fields in the source file. If no special quote character is used write `none`. `none` is also the default.

datetime format

Formatting of timestamps. Uses same notation as Python `datetime` class function. See documentation at python.org here

Example: timestamp `2019-09-13 16:09:10` corresponds to format string `%Y-%m-%d %H:%M:%S`

Defaults to ISO 8601 format `%Y-%m-%d %H:%M:%S`

datetime extra char

number of extra character at the end of the timestamp. Sometimes there are some characters add to timestamps e.g. to indicate timezone. The numbers of these need to be defined even if zero. Default value is 0.

fault columns

data file columns that contain the turbine status or fault code. **Zero based** i.e. leftmost column in source file is column 0. If information about the turbine state is contained in multiple places add all of these columns here separated by commas e.g.:

```
fault columns = 8,9,10
```

In the *Example input data* you would put 6 and 7 here. Because both of those columns can then be used to filter the data based on status information.

This is mandatory value

replace fault codes

filtering option needed in case the source file contains status/fault codes that are not numbers. Non-numeric data in the data set cause issues for the analysis code, so the fault codes need to filtered first. In case the fault/status codes in the source data are text, set:

```
replace fault codes = True
```

if the replacement is not needed set this to `False`. In the example earlier *Example input data*. This filtering is needed. in some cases the output fault codes are already numeric, so in those cases it can be false.

Defaults to `False`

4.6.2 Section: Output

This section defines the output produced by the power loss counter script.

The script allows the user to set what kind of outputs are needed. All data is output into text files in a results directory. All output files are named us the *id* identifier.

If a certain output is needed set the value of the corresponding key to `True`

For example producing the alarm time series is relatively slow. Setting unneeded parts to `False` can make calculations faster.

By default all outputs are set to `True` and the results are written to the local directory of the script.

result directory

directory where the results will be written to

summary

Prints a summary statistics file containing overall information about the original data.

data time series

sets time series saving on or off. **NOTE** constructing the time series can take a long time depending on the size of the data set. When doing preliminary analysis, unless absolutely required, it is recommended to keep this set as False

power curve

Prints a file that contains the power curve calculated from the data.

plot

sets plotting on or off. Script makes a power curve plot with icing events highlighted. The plots are saved in to the results directory as `.png`

icing event list

set the icing event list saving on or off

filtered raw data

switch the raw data saving on or off

Alarm time series

Print a time series file of the icing alarms. The file will be a .csv file with the following columns:

Times-tamp	Alarm signal value	Wind Speed	Reference Power	Tempera-ture	Power	Power limit (P10)

Here `Alarm signal value` indicates the icing status. Values of the alarm signal are listed in the table below

Alarm signal value	Interpretation
0	No alarm
1	Icing alarm, reduced production
2	Icing alarm, stop during operation
3	Overproduction

4.6.3 Section: Data Structure

This section defines the format of the source data. Note that the leftmost column in your source data is column 0.

All of these are always required.

timestamp index

index of the timestamps in the original data.

wind speed index

index of wind speed

wind direction index

index of wind direction

temperature index

index of temperature measurements. Temperature needs to be in degrees Celsius.

power index

Index of output power measurement in source data. (Preferably in kilowatts, the units are assumed in some places when formatting output files.)

Note: if source data uses relative values of output power the ice detection methods in the scripts do still work. The overall values for lost production might not make sense, but the timing of the icing events can still be calculated.

rated power

rated power of the turbine.

state index

indexes of state values or status codes used in data filtering. These can be found in multiple columns, just put everything here separated by commas i.e.:

```
state index = 8,9,10
```

normal state

The value of the state variable in so called normal state, used for filtering the data. This can be text or a number just use the same format as in the source data. Also you can specify multiple values here, just write them all on one line separated by commas.

set these in same order as the state index above. If you want to include multiple valid values for one state variable add the appropriate index into state index once for each required value.

Note: if the actual code contains a comma, the code will interpret that as two separate values and will crash.

site elevation

site elevation in meters above sea level, used for correcting the wind measurements.

status index

Index of the status signal. Used for collecting statistics of known stops

status code stop value

Value of the status code that indicates that the turbine has stopped.

4.6.4 Section: Icing

If the turbines on the site have ice detection or some kind of ice prevention system (anti- or de-icing) the code can take this into account and produce statistics of the Ice prevention system operation.

This section is not mandatory, if there is no ice detector or no blade heating available. If `Icing` as a section is included, then all of these need to be defined as well.

Ice detection

Set this to `True` if there is an ice detection signal in the data. Leave the value to `False` if not. Used for collecting production statistics. This only cares about the presence of an explicit ice detection signal, sometimes a heated site might not have a visible ice detection signal in the data.

icing alarm code

Code in the data that corresponds to icing alarm.

icing alarm index

Zero-based index of the icing alarm code

heating

Set to `True` if site has blade heating.

ips status code

The code in the data that indicates that blade heating is on.

ips status index

Zero-based index of the ips status code

ips status type

Sets the type of the ips status code. Set to 1 if the ips status code value defined in `ips status code` indicates that ips is on and the blade heating is active. If this is set to 2 the code interprets all other values except the value in `ips status code` as blade heating being on.

ips power consumption index

If ips power measurement exists in the data, use this to give the index of the power consumption signal (zero-based). If there is no power consumption signal in the data, set this value to -1.

4.6.5 Section: Binning

Sets the binning options for the power curve calculations.

This is not required.

minimum wind speed

minimum wind speed, all values below this will be sorted in the firs bin. Usually set to 0. Defaults to 0, if not set.

maximum wind speed

Maximum wind speed for the power curve, all values above this will end up in the last bin. Default value 20.

wind speed bin size

Wind speed bin size in meters per second. Default value 1.

wind direction bin size

Wind direction bin size in degrees.

NOTE: If you do not want to use wind direction based binning set the bin size to 360 degrees.

Default is set 360 i.e. no direction-based binning is used by default.

4.6.6 Section: Filtering

Data is filtered prior to analysis. The options for the filter are set in this section.

power drop limit

Lower limit for the power curve, defaults to *10* meaning using the P10 value to indicate the lower limit value used for ice detection.

overproduction limit

upper limit for normal operation. Used to mark overproduction in the data, defaults to *90* corresponding to top 90 percentile.

icing time filter

Number of continuous samples required to be under the lower limit in order to indicate an icing event has started.

Note: this is number of samples, so for ten-minute data use 3 for 30 minutes and so on. Default value is 3.

stop filter type

Sets the source of what is counted as an icing induced turbine stop when calculating icing events. Stop filter here refers to an extra filtering step that can be used to remove turbine stops from the data if there is status code information that indicates that the turbine was stopped for reasons other than icing. Can have three different values:

0. Power level based filter (default). No extra filtering.
1. Status code stop. If the value of `stop filter type` is *1* filter out the bits where the status code in column set by `status index` is set to value defined by `status code stop value`
2. Status code normal operational state. If the value of `stop filter type` is *2*, keep only the parts of data where `status index` is set to value defined by `status code stop value`

In case *2* `status code stop value` refers to turbine normal state.

stop time filter

Time filter used in stop detection. This is also the number of consecutive samples. Default value 6.

statefilter type

sets the filtering rule used to filter the data according to the state variable set earlier. State filter has four options

1. inclusive: Default value, keep only the part of the data where the state variable matches the defined normal state
2. exclusive: remove all data where state variable matches the defined normal state
3. greater than: keep only lines of data where state filter value greater than or equal to the value set
4. less than: keep only values where ste filter value is less than or equal to the value set

The name `normal state` for the filtering variable can be misleading due to option 2 here. In the *Example input data* you could filter based on column 6 using option 1 setting the normal value to `OK`.

power level filter

Filter limit to remove stoppages from data. A power multiplier, defaults to 0.01. Power level filtering is used in order to remove times when turbine is stopped from the data. Useful if for example no turbine state information is known. This is applied to data

reference temperature

Initial reference data set is created by filtering out all measurements where temperature is below this limit. Defaults to 3 degrees Celsius.

temperature filter

Temperature limit for ice detection. If production is below the limit set in `power drop limit` **and** temperature is below the value set here, events are classified as icing. Default value is 1.

icing time

Minimum time needed to trigger an icing event. If production is below the designated level for at least the **number of samples** defined here and temperature is below the limit set with `temperature filter`, an icing alarm is triggered.

stop time filter

When calculating stops from production, the production needs to be below the value defined in `stop limit multiplier` for at least the **number of samples** defined here in order to declare the samples as an icing induced stop. Default value is 3.

stop limit multiplier

Multiplier to define the lower limit for power. If output power is below this times nominal power the turbines is determined to have stopped. Defaults to 0.005

min bin size

Minimum sample count in a single bin when creating power curves. Defaults to 36.

distance filter

set this to `True` to add an additional filtering step to power curve calculation. This can improve results in most cases, on by default. Can be removed by setting `distance filter = False`

start time

If you want to calculate icing events and their losses to a period other than the whole data set, you can specify a different start time for your analysis. This uses same formatting that is specified in Section: Source file under *datetime format*.

If you want to use the data set from the beginning write `NONE` here in all caps. Set to `NONE` by default.

stop time

If you want to calculate icing events and their losses to a period other than the whole data set, you can specify a different stop time for your analysis. This uses same formatting that is specified in Section: Source file under *datetime format*.

If you want to use the data set till the end write `NONE` here in all caps. Set to `NONE` by default.

4.6.7 Mandatory values

The following values need to be set for every dataset.

- Section: Source file:
  - id
  - filename

  - fault columns
- Section: Data Structure:
  - timestamp index
  - wind speed index
  - wind direction index
  - temperature index
  - power index
  - rated power
  - state index
  - normal state
  - site elevation
  - status index
  - status code stop value

4.6.8 Default values

Set defaults are listed below:

- Section 'Source file':
  - delimiter: ','
  - quotechar: 'NONE'
  - datetime format: '%Y-%m-%d %H:%M:%S'
  - datetime extra char: '0'
  - replace fault codes': 'False'
- Section 'Output':
  - result directory: '.'
  - summary: 'True',
  - plot: 'True',
  - alarm time series: 'True',
  - filtered raw data: 'True',
  - icing events: 'True'
  - power curve: 'True'
- Section 'Binning':
  - minimum wind speed: '0',
  - maximum wind speed: '20',
  - wind speed bin size: '1',
  - wind direction bin size: '360'
- Section: 'Filtering':

- power drop limit: '10',
- overproduction limit: '90',
- power level filter: '0.01',
- temperature filter: '1',
- reference temperature: '3',
- icing time: '3',
- stop filter type: '0',
- stop limit multiplier: '0.005',
- stop time filter: '6',
- statefilter type: '1',
- min bin size: '36',
- distance filter: 'True',
- start time: 'None',
- stop time: 'None'

4.7 Wind park analysis

The script by itself only operates on one time series (one turbine) at a time. If you are dealing with a data set that contains more than one turbine, using this scrip requires that you write a separate .ini file for each turbine. After this it is possible to write a small script or a batch file that runs the script for each turbine separately. One such example is included in the release .zip as `multifile_t19_example.py`

This script also combines the summary files into one for easier comparison between the turbines.

CHAPTER

FIVE

CLASS DOCUMENTATION FOR THE TASK 19 POWER LOSS COUNTER

class t19_ice_loss.**AEPcounter**

set of functions to calculate AEP losses from structured data

air_density_correction(*data*)

Calculate air density correction for wind speed according to specifications in the IEA document returns a new array with corrected wind speed in place of the measured one

Corrected wind speed for the site can be calculated as:

ws_site = ws_std*((temp_site*P_std)/(temp_std*(101325*(1-2.2557e-5*h)^5.25588)))^(1/3)

ws_site is the corrected wind speed for the site
ws_std is the measured nacelle wind speed
temp_site is the site temperature
P_std is the standard air pressure at sea level (101325 Pa)
temp_std is the standard temperature of 15 C (288.15 K)
h is site height in meters

Parameters **data** –

Returns corrected data

bin_measurement(*data*, *bin_centers*, *comp_column*, *bin_index_column*, *direction=False*)

puts a measurement into an appropriate bin

Parameters

- **data** – contains a single line of measurements
- **bin_centers** – a numpy.ndarray containing centerpoints of the bin division
- **comp_column** – column index of the variable used for binning e.g. wind speed index when searching for the appropriate wind speed bin
- **bin_index_column** – column index where bin indexes are stored

Returns parameter data with the bin index appended

bin_size_filter(*pc*, *size_limit*)

bin size based filtering for the power curve removes all data from the bin if not enough values available

Parameters

- **pc** – power curve matrix
- **size_limit** – number of measurements required for the bin to be used

Returns a boolean matrix that can be used to mark too small bin as empty

calculate_production(*data*, *index*, *delta=datetime.timedelta(seconds=600)*)
calculates total production between from previous time stamp to current, assuming the difference is constant skips all occurrences where the difference between two adjacent timestamps is not constant

NOTE: assumes timestamp is at index 0

Parameters

- **data** – input data, containing the measured output
- **index** – index of the production measurement
- **delta** – difference between two timestamps defaults to ten minutes

Returns structure containing [end timestep, production]

calculate_production_stats(*data*, *pc*, *ice_alarms*, *ice_stops*, *status_stops*, *ips_on*, *ice_detection*)
Calculates month-by-month statistics from the data.

Parameters

- **data** – input data used to asses production
- **pc** – power curve used to calculate theoretical production

Returns

combine_timeseries(*pow_alms1*, *stops*, *pow_alms2*)
combine all different types of alarms into one big timeseries

Timeseries file combines the alarm files into one timeseries that classifies the ice cases according ot the naming convention in the documentation

1 = power loss 2 = stop 3 = overproduction

Parameters

- **pow_alms1** –
- **stops** –
- **pow_alms2** –

Returns

count_availability(*data*)
Calculate availability number for data

availability tells in % the amount of possible data available in the dataset checks if there is data available for each timestamp between first and last timestamp if there is, returns availability of 100 %

does not check data integrity, only if there is some kind of data available. Available data could be garbage.

Parameters **data** – data array of the time series

Returns availability number

count_power_curves(*data*)
Calculates a set of power curves from the input data bins the data according to wind speed and direction the binning and the column indexes of the data are defined in the class variables

Power curves contain the power curve and the P10 limit for said curve separately for each wind direction defined in the wind direction binning.

output is three dimensional array containing:

- median wind speed
- median wind direction
- P10 value
- bin size (number of measurements in this particular bin)

for each speed and direction defined in self.wind_bins and self_direction bins

missing data (empty bins) are marked as nan missing values can then be interpolated over so that there are no empty bins.

Also possible to do other kinds of filtering to improve the end result and to reduce the impact of outliers in the data

Method automatically filters the data according to elsewhere defined state variable filter (is this a good idea???)

Only part of data where temperature is more than 3 degrees is used to make the power curves

Parameters

- **data** – input data time series. can be unfiltered
- **temperature_filter_level** – temperature in degrees, all data with temperatures above this level are used to build the reference dataset
- **lower_limit** – percentile used as limit for power reduction (default 10)
- **upper_limit** – percentile used as limit for overproduction (default 90)

Return pc a numpy.ndarray that contains the power curves, warning limits and bin sizes for each bin, sorted by wind speed and direction

define_removable_indexes(*data*, *timings*)

calculate the start and stop indexes in data to remove all data defined in the array timings

Parameters

- **data** – original dataset
- **timings** – array containg the incident starts and stops

Returns indexes that can be used to filter the original data, a list of ranges

diff_filter(*data*, *diff_limit=0.001*)

appplies a filter to data that discards all values that differ less than diff_limit from the previous one :param data: :param diff_limit: :return:

distance_filter(*pc*, *target_value*)

calculate distances between different power curves, if value is dramatically different replace with mean of all others goes through all curves bin by bin can be useful to automatically weed out outliers in the data

Parameters

- **pc** – prefilterd power curves
- **target_value** – index of the value used for filtering e.g. power

Returns fltered power curve matrix

distance_to_neighbours(*pc*, *speed_bin*, *direction_bin*, *variable_index*)
helper function used in power curve post-processing: calculates the mean distance of point at index in different curves stored in data Operates only on one bin at a time. requires power curves divided according to wind speed and direction

Parameters

- **pc** – original dataset containing the power curves as defined by count_power_curves function
- **speed_bin** – index of active wind speed bin
- **direction_bin** – index of active wind direction bin
- **variable_index** – processed variable. usually power

Returns mean distance to all other power curves in the particular bin

expand_array(*arr*, *n*)
add n columns to the right-hand side of a numpy ndarray used for bin indices when binning data

Parameters

- **arr** – original array
- **n** – number column to be added

Returns appended array

fetch_bin_contents_2d(*data*, *bin_index1*, *bin_number1*, *bin_index2*, *bin_number2*)
returns the contents of a particular bin in case that data is binned according to two different variables for example wind speed and direction

Parameters

- **data** – binned data wind bin indexes
- **bin_index1** – column where the first bin index is found
- **bin_number1** – requested bin on the first bin index
- **bin_index2** – column where the second bin index is found
- **bin_number2** – requested bin on the second bin index

Returns slice of the data containing the contents of the wanted bin

find_icing_related_stops(*data*, *power_curve*)
Finds timestamps from the data, when the turbine has stopped for whatever reason

uses filtering requirements defined in the specification document: pwr_mean< 0.005*P_rated

Parameters

- **data** – timeseries data of output
- **power_curve** – power curve array used

Returns filtered data with stops flagged

get_fallback_value(*section*, *config_var*)
Helper function used when reading .ini files. Sets non-mandatory values to common fallback settings :param section, where config_var is defined :param config_var: :return: fallback value

increase_reference_dataset(*data*, *stop_timings*, *alarm_timings*, *over_timings*)
re-increase the size of reference dataset to include all the non-iced datapoints.

Parameters

- **data** – original dataset
- **stop_timings** – stops calculated from the data
- **alarm_timings** – alarm incidents calculated from the data
- **over_timigns** – overproduction incidents from the data

Returns new reference dataset

interpolate_over_nans(*pc*)
helper function to interpolate over possible nan values in power curves caused by empty bins during binning input is a single power curve i.e. 2d matrix of wind speed versus power

Parameters **pc** – power curve matrix as returned by self.count_power_curves

Returns interpolated power curves

mean_power_curve(*pc*)
calculates mean (direction independent power curve)

Parameters **pc** –

Returns mean power curve, averagesd accross direction bins

nan_helper(*y*)
Helper to handle indices and logical indices of NaNs.

Input:
- y, 1d numpy array with possible NaNs

Output:
- nans, logical indices of NaNs
- index, a function, with signature indices= index(logical_indices),
  to convert logical indices of NaNs to 'equivalent' indices

Example:

```
# linear interpolation of NaNs
nans, x= nan_helper(y)
y[nans]= np.interp(x(nans), x(~nans), y[~nans])
```

Parameters **y** – input array

Returns converted array

one_year_month_sums(*data*, *wanted_year*, *index*)
Helper function, calculates the monthly sums of any timeseries data for a given year :param data: :param wanted_year: :param index: :return:

power_alarms(*data*, *power_curves*, *time_filter=True*, *over=False*)
flag timestamps that match wanted power alarm criteria.

For each measurement in data, search the proper value from the power curves If the power at any moment is below the previously calculated P10 value AND temperature is below a threshold, flag the timestamp.

after all the data is processed do an additional time-based filtering step where all cases where there are not enough consecutive alarms are discarded. default idea is to demand that the power should remain below the P10 value for at least half an hour before the incident is considered a confirmed icing event

Another considered ice class is cases where iced anemometer results in apparent overproduction. These cases are seen in the data as appearing above the P90 line. They are flagged in a similar way and same time filtering applies here as well

The specification lists these as ice case A (production loss) and ice case C (overproduction) These are marked in the output as 1 for case A and 3 for case C in the alarm variable

TODO: assumes ten minute data, should probably be a parameter

Parameters

- **data** – input data
- **power_curves** – calculated power curves, binned based on wind speed and direction
- **time_filter** – if True, an additional time filter is applied to the data
- **time_filter_length** – number of consecutive values below the alarm limit required to trigger the icing alarm
- **over** – if True, flags the timestamps where the power is above P90 instead

Returns an array of the format [timestamp, alarm, wind speed, reference power, temperature, power, limit]

power_curve_uncertainty_average(*pc*, *low_limit=4*, *high_limit=15*)

Calculate a mean value for power curve uncertainty for the summary file use only wind speeds between low_limit and high_limit and return just one value

Parameters

- **pc** – power curve structure
- **low_limit** – lowest applicable wind speed
- **high_limit** – highest applicable wind speed

Returns one value to represent power curve uncertainty

power_level_filter(*data*)

remove all data points where power is below the wanted limit level

limit is defined as percentage of rated power

Parameters

- **data** – unfiltered input data
- **limit_level** – filtering level as fraction of rated

Returns data with the unwanted timestamps removed

power_loss_during_alarm(*data*, *ips_alarm=False*)

Collect the start and stop times of icing alarms and calculate the total power/production loss during the icing event

counts the production loss by calculating the approximate area between the estimated production curve and the actual production curve as calculated by power_alarms

Parameters

- **data** – data produced by the power_alarms function
- **ips_alarm** – set to True if alarm was caused by IPS system

Returns a structure containing the starts and stops and losses formatted as [starttime stoptime powerloss]

prettyprint_power_curves(*pc*, *index=2*, *print_result=False*)
print matrix representations of different variables inside the power curve data structure returns a human readable matrix of wanted quantity

Parameters

- **pc** – power curve array
- **index** – index in the power curve to be printed:

  2 : mean power 3 : P90 4 : P10 5 : bin standard deviation 6 : bin uncertainty (std_dev / mean) 7 : bin size

- **print_result** – if True, prints the resulting array to stdout, defaults to False

Returns sanitized matrix

put_data_into_bins(*data*, *bins*, *comp_column*, *direction=False*)
Bins into externally defined set of bins. Adds a column to the data matrix containing a bin index. After this contents of any bin can be found using features of a numpy.ndarray

Example:

we need contents of bin number 7. This is returned by data[data[:,-1]==7,:]

Parameters

- **data** – original data that needs to be separated in to bins. A numpy.ndarray
- **bins** – center points of the bins
- **comp_column** – column in the data containing the value used for binning (wind speed index when binning by wind speed)

Returns expanded array, contains the original data and one additional column that contains a bin index for each line

set_binning_options_from_file(*filename*)
set bin division based on a config file

set_data_options_from_file(*filename*)
read in configuration settings from a config file

set_filtering_options_from_file(*filename*)
set filtering options based on a config file

set_ips_options_from_file(*filename*)
set config options for a heated site, first check if "Icing" section even exists, then set the options

state_filter_data(*data*)
remove all data where the state variable is something else than normal_state correct state variable values depend on turbine type if exclude set to true, remove all lines where state==normal_state exclude is set in a class variable

if state_filter_type == 3 filter removes data below a certain threshold threshold is set in normal_state state filter type 3 is used in case there is no explicit turbine state and the filtering has to be done using output power or such

Parameters **data** – data to be filtered

Returns filtered data with the filterd liens removed

status_code_stops(*data*, *power_curves*, *filter_type='stop'*)
Flag the moments in data where the turbine status code indicates icing The statuscode is defined in self.stopcodes

Parameters **data** – input data to be processed

Return [timestamp, alarm, wind speed, reference power, temperature, power, limit]

temperature_filter_data(*data*)
remove all data points with temperature below set threshold

Parameters **data** – input data

Returns data set containing only the data in previously specified range

theoretical_output_power(*data*, *power_curves*)
calculates the theoretical, expected output power based on power curve and measured wind speed

Parameters

- **data** –
- **power_curve** –

Returns rerference power, in structure [timestamp, interpolated reference power, actual measured output power]

time_filter_data(*data*)

Parameters

- **data** – the data to be filtered
- **start** – start time as datetime.datetime
- **stop** – stop time as datetime.datetime

Returns filtered dataset

timefilter_ice_alarms(*data*, *window*)
clean outliers from ice alarms, demand, that there is at least window number of consecutive alarms begin the icing event from the first switch from 0->1 end at the switch from 1->0

Parameters

- **data** – time series of alarms created by the power_alarms function
- **window** – length of hte filtering window

Return data reformatted data, with individual events removed

wind_dir_mean(*a*)
calculates the mean of wind direction measurements contained in array data

Parameters **a** – array of wind direction measurements in degrees

Returns mean of the array

wind_speed_filter(*data*, *limit_level*)
remove all data points with wind speed below a preset level

Parameters

- **data** – original data
- **limit_level** – filtering level

Returns filtered data

class t19_ice_loss.**CSVimporter**(*inputfilename=''*)
sets up an importer that reads in a set of data from a predefined .csv file

create_faultfile()
Creates a filename for the fault file based on the name of the input file

Returns filename of the fault dictionary

create_new_faultcodes(*column_num*, *write_to_file=False*, *outfilename=''*)
Generate replacement faultcodes from the data in case faultcodes are in some kind of alphanumeric format i.e. not numbers

column num is the column that contains all the fault codes.

Parameters

- **column_num** – column index that contains the fault codes
- **write_to_file** – if True, resutls written to disk into file specified by _outfilename_
- **outfilename** – name of the output file

Returns fault_dict a python dictionary containig all discovered fault codes (dictionary keys) and numbers to replace them with (dictionary values)

is_float(*char_string*)
checks whether or not a character string can be converted into a float

Parameters **char_string** –

Returns status of conversion

process_fault_codes()
create a data structure that can be used to replace textual fault codes in the data that is read in for processing

read_data()
read pre-specified .csv formatted datafile, return a numpy nparray of data in format:

Specifications of the original file are defined as class variables.

sets values of self.data and self.headers according to the contents of the file

[timestamp, value, . . .]

read_fault_codes(*infilename*)
read fault codes from a previously generated .json file

Parameters **infilename** – name of the input file

Returns fault_dict a python dictionary containig all discovered fault codes (dictionary keys) and numbers to replace them with (dictionary values)

read_file_options_from_file(*config_filename*)
set file options from a config file see the documentation for full listing of options

Parameters **config_filename** – name and full path of the config file

update_fault_codes(*column_num*, *infilename*, *write_to_file=False*, *outfilename=''*)
updates an already saved list of fault codes from the inputfile

Parameters

- **column_num** – column index of the fault variable
- **infilename** – name of the input file
- **file** (*write_to*) – toggles writing the updated fault dictionary to a file

- **outfilename** – nae of the output file

Returns the updated fault dictionary

write_fault_dict(*outfilename*, *fault_dict*)
writes a fault dictionary on disk as a .json file

Parameters

- **outfilename** – name of the output filename
- **fault_dict** – fault code dictionary

class t19_ice_loss.**Result_file_writer**
sets up a writer to deal with results of the counter

generate_standard_plots(*data*, *pc*, *aepc*, *red_power*, *overprod*, *stops*, *data_sizes*, *alarm_timings*, *over_timings*, *stop_timings*, *ips_on_flags*, *write=False*)
create two predefined plots from the time series data

Parameters

- **data** – input data
- **pc** – power curve structure
- **aepc** – active AEP Counter object
- **red_power** – timeseries of reduced power
- **overprod** – timeseries of overproduction
- **stops** – timeseries of stops
- **data_sizes** – lengths of differently filtered datasets
- **alarm_timings** – statistics of reduced power incidents
- **over_timings** – statistics for over production
- **timigns** (*stop*) – statistics of icing induced stop events
- **ips_on_flags** – IPS stops, only valid for heated systems, will be None if not heated
- **write** – if True, write to disk, otherwise run matplotlib.pyplot.show()

insert_fault_codes(*data*, *aepc*, *reader*)
re-insert the textual fault codes into the data time series table

Parameters

- **data** – input data
- **aepc** – aep counter object used
- **reader** – active CSVReader object

Returns the filtered data as numpy.ndarray

read_powercurve_from_file(*filename*)
read powercurve from file produced by the program

Some information is lost during the save process right now. real bin centers are not saved and neither are sample counts

Parameters **filename** – filename where the power curve sits

Returns power curve structure formatted in same way as earlier

summary_statistics(*aepc*, *data*, *pc*, *alarm_timings*, *stop_timings*, *over_timings*, *status_timings*, *ice_timings*, *ips_timings*, *data_sizes*)

Calculate summary statistics for the dataset. contains: availability data loss due to filtering size of the reference dataset hour counts for different ice classes production losses due to different causes Theoretical maximum production observed production losses due to all reasons Written to a file

Parameters

- **aepc** – the active aeoc object
- **data** – data used to calculate statistics
- **pc** – power curve structure
- **alarm_timings** – reduced power incidents
- **stop_timings** – icing induced stops
- **over_timings** – overproduction incidents
- **data_sizes** – sizes after each filtering step

Returns status of the write operation, full filename ,possible error

write_alarm_file(*result_filepath*, *array*)

writes the alarm timeseries in a file in working directory

Parameters

- **result_filepath** – full filename
- **array** – data array

Returns status of write operation, possible error message

write_alarm_timings(*result_filepath*, *array*)

writes a data timeseries in a file in working directory

Parameters

- **result_filepath** – path of result file
- **array** – data array

Returns status of writing, error

write_monthly_stats(*data*, *pc*, *aepc*, *ice_events*, *ice_stops*, *status_stops*, *ips_on_flags*, *ice_detected*)

write production loss statistics to file

Parameters

- **data** – input data
- **pc** – calculated power curve
- **aepc** – aep counter used to calculate the stats

Returns status of the write operation, filename, error

write_power_curve(*aepc*, *pc*, *pc_id=''*)

Write power curve, P10 and P90 into a file

Parameters

- **aepc** – AEPCounter used to calculate the power curve

- **pc** – the actual power curve

Returns status of the write operation, filename of the power curve file, possible error

write_time_series_file(*result_filepath*, *array*, *headers*, *aepc*, *pc*)
writes a data timeseries in a file in working directory

Parameters

- **result_filepath** –
- **array** –
- **headers** –

Returns status of the writing, possible error

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

t

INDEX

R

S

T

U

W