windpowerlib Documentation

Release

oemof developer group

Jul 07, 2017

Contents

Contents:

CHAPTER 1

Getting started

Introduction

The windpowerlib is a library that provides a set of functions and classes to calculate the power output of wind turbines. It was originally part of the feedinlib (windpower and pv) but was taken out to build up a community concentrating on wind power models.

For a quick start see the *Examples and basic usage* section.

Documentation

Full documentation can be found at readthedocs.

Use the project site of readthedocs to choose the version of the documentation. Go to the download page to download different versions and formats (pdf, html, epub) of the documentation.

Installation

If you have a working Python3 environment, use pypi to install the latest windpowerlib version.

```
pip3 install windpowerlib
```

So far, the windpowerlib is mainly tested on python 3.4 but seems to work down to 2.7. Please see the installation page of the oemof documentation for complete instructions on how to install python on your operating system.

Optional Packages

To see the plots of the windpowerlib example in the *Examples and basic usage* section you should install the matplotlib package. Matplotlib can be installed using pip3 though some Linux users reported that it is easier and more stable to use the pre-built packages of your Linux distribution.

Examples and basic usage

The basic usage of the windpowerlib is shown *here*. The presented example is available as jupyter notebook and python script. You can download them along with example weather data:

- `Python script`
- `Jupyter notebook`
- `Example data file`

To run the examples you first have to install the windpowerlib. To run the notebook you also need to install notebook using pip3. To launch jupyter notebook type `jupyter notebook` in terminal. This will open a browser window. Navigate to the directory containing the notebook to open it. See the jupyter notebook quick start guide for more information on how to install and how to run jupyter notebooks.

Contributing

Clone/Fork: https://github.com/wind-python/windpowerlib

If you are interested in wind models and want to help improve the existing model do not hesitate to contact us. As the windpowerlib started with contributors from the oemof developer group we use the same developer rules.

License

CHAPTER 2

Basic usage

This example shows you the basic usage of the windpowerlib. There are mainly three steps. First you have to import your weather data, then you need to specify your wind turbine, and in the last step call the windpowerlib functions to calculate the feedin timeseries.

Before you start you have to import the packages needed for these steps.

Import necessary packages and modules

```
In [1]: __copyright__ = "Copyright oemof developer group"
        __license__ = "GPLv3"

        import os
        import pandas as pd

        from windpowerlib.modelchain import ModelChain
        from windpowerlib.wind_turbine import WindTurbine
        from windpowerlib import wind_turbine as wt
```

You can use the logging package to get logging messages from the windpowerlib. Change the logging level if you want more or less messages.

```
In [2]: import logging
        logging.getLogger().setLevel(logging.DEBUG)
```

Import weather data

In order to use the windpowerlib you need to at least provide wind speed data for the time frame you want to analyse. The function below imports example weather data from the weather.csv file provided along with the windpowerlib. The data include wind speed at two different heights in m/s, air temperature in two different heights in K, surface roughness length in m and air pressure in Pa.

To find out which weather data in which units need to be provided to use the ModelChain or other functions of the windpowerlib see the individual function documentation.

```
In [3]: def get_weather_data(filename='weather.csv', **kwargs):
            r"""
            Imports weather data from a file.

            The data include wind speed at two different heights in m/s, air
            temperature in two different heights in K, surface roughness length in m
            and air pressure in Pa. The file is located in the example folder of the
            windpowerlib. The height in m for which the data applies is specified in
            the second row.

            Parameters
            ----------
            filename : string
                Filename of the weather data file. Default: 'weather.csv'.

            Other Parameters
            ----------------
            datapath : string, optional
                Path where the weather data file is stored.
                Default: 'windpowerlib/example'.

            Returns
            -------
            weather_df : pandas.DataFrame
                DataFrame with time series for wind speed `wind_speed` in m/s,
                temperature `temperature` in K, roughness length `roughness_length`
                in m, and pressure `pressure` in Pa.
                The columns of the DataFrame are a MultiIndex where the first level
                contains the variable name (e.g. wind_speed) and the second level
                contains the height at which it applies (e.g. 10, if it was
                measured at a height of 10 m).

            """

            if 'datapath' not in kwargs:
                kwargs['datapath'] = os.path.join(os.path.split(
                    os.path.dirname(__file__))[0], 'example')
            file = os.path.join(kwargs['datapath'], filename)
            # read csv file
            weather_df = pd.read_csv(file, index_col=0, header=[0, 1])
            # change type of index to datetime and set time zone
            weather_df.index = pd.to_datetime(weather_df.index).tz_localize(
                'UTC').tz_convert('Europe/Berlin')
            # change type of height from str to int by resetting columns
            weather_df.columns = [weather_df.axes[1].levels[0][
                                      weather_df.axes[1].labels[0]],
                                  weather_df.axes[1].levels[1][
                                      weather_df.axes[1].labels[1]].astype(int)]
            return weather_df


        # Read weather data from csv
        weather = get_weather_data(filename='weather.csv', datapath='')
        print(weather[['wind_speed', 'temperature', 'pressure']][0:3])
```

```
variable_name             wind_speed          temperature         pressure
height                            10       80          2       10       0
```

```
2010-01-01 00:00:00+01:00    5.32697  7.80697      267.60  267.57  98405.7
2010-01-01 01:00:00+01:00    5.46199  7.86199      267.60  267.55  98382.7
2010-01-01 02:00:00+01:00    5.67899  8.59899      267.61  267.54  98362.9
```

Initialise wind turbine

To initialise a specific turbine you need a dictionary that contains the basic parameters. A turbine is defined by its nominal power, hub height, rotor diameter, and power or power coefficient curve.

There are two ways to initialise a WindTurbine object in the windpowerlib. You can either specify your own turbine, as done below for 'myTurbine', or fetch power and/or power coefficient curve data from data files provided by the windpowerlib, as done for the 'enerconE126'.

You can execute the following to get a list of all wind turbines for which power or power coefficient curves are provided.

```
In [4]: # get power curves
        # get names of wind turbines for which power curves are provided (default)
        # set print_out=True to see the list of all available wind turbines
        wt.get_turbine_types(print_out=False)

        # get power coefficient curves
        # write names of wind turbines for which power coefficient curves are provided
        # to 'turbines' DataFrame
        turbines = wt.get_turbine_types(filename='power_coefficient_curves.csv', print_out=False)
        # find all Enercons in 'turbines' DataFrame
        print(turbines[turbines["turbine_id"].str.contains("ENERCON")])
```

```
            turbine_id    p_nom
52   ENERCON E 70 2300  2300000
64  ENERCON E 101 3000  3000000
65  ENERCON E 126 7500  7500000
66  ENERCON E 115 2500  2500000
67    ENERCON E 48 800   800000
68   ENERCON E 82 2000  2000000
69    ENERCON E 53 800   800000
70   ENERCON E 58 1000  1000000
71   ENERCON E 70 2000  2000000
72   ENERCON E 82 2300  2300000
73   ENERCON E 82 3000  3000000
74   ENERCON E 92 2300  2300000
75  ENERCON E 112 4500  4500000
```

```
In [5]: # specification of own wind turbine (Note: power coefficient values and
        # nominal power have to be in Watt)
        myTurbine = {
            'turbine_name': 'myTurbine',
            'nominal_power': 3e6,  # in W
            'hub_height': 105,  # in m
            'rotor_diameter': 90,  # in m
            'power_curve': pd.DataFrame(
                    data={'values': [p * 1000 for p in [
                                0.0, 26.0, 180.0, 1500.0, 3000.0, 3000.0]],  # in W
                          'wind_speed': [0.0, 3.0, 5.0, 10.0, 15.0, 25.0]})  # in m/s
            }
        # initialise WindTurbine object
        my_turbine = WindTurbine(**myTurbine)
```

```
In [6]: # specification of wind turbine where power curve is provided
        # if you want to use the power coefficient curve add
        # {'fetch_curve': 'power_coefficient_curve'} to the dictionary
        enerconE126 = {
            'turbine_name': 'ENERCON E 126 7500',  # turbine name as in register
            'hub_height': 135,  # in m
            'rotor_diameter': 127  # in m
            }
        # initialise WindTurbine object
        e126 = WindTurbine(**enerconE126)
```

Use the ModelChain to calculate turbine power output

The ModelChain is a class that provides all necessary steps to calculate the power output of a wind turbine. If you use the 'run_model' method first the wind speed and density (if necessary) at hub height are calculated and then used to calculate the power output. You can either use the default methods for the calculation steps, as done for 'my_turbine', or choose different methods, as done for the 'e126'.

```
In [7]: # power output calculation for my_turbine
        # initialise ModelChain with default parameters and use run_model
        # method to calculate power output
        mc_my_turbine = ModelChain(my_turbine).run_model(weather)
        # write power output timeseries to WindTurbine object
        my_turbine.power_output = mc_my_turbine.power_output
```

```
DEBUG:root:Calculating wind speed using logarithmic wind profile.
DEBUG:root:Calculating power output using power curve.
```

```
In [8]: # power output calculation for e126
        # own specifications for ModelChain setup
        modelchain_data = {
            'wind_speed_model': 'logarithmic',      # 'logarithmic' (default),
                                                    # 'hellman' or
                                                    # 'interpolation_extrapolation'
            'density_model': 'ideal_gas',           # 'barometric' (default), 'ideal_gas'
                                                    #  or 'interpolation_extrapolation'
            'temperature_model': 'linear_gradient', # 'linear_gradient' (def.) or
                                                    # 'interpolation_extrapolation'
            'power_output_model': 'power_curve',    # 'power_curve' (default) or
                                                    # 'power_coefficient_curve'
            'density_correction': True,             # False (default) or True
            'obstacle_height': 0,                   # default: 0
            'hellman_exp': None}                    # None (default) or None

        # initialise ModelChain with own specifications and use run_model method to
        # calculate power output
        mc_e126 = ModelChain(e126, **modelchain_data).run_model(
            weather)
        # write power output timeseries to WindTurbine object
        e126.power_output = mc_e126.power_output
```

```
DEBUG:root:Calculating wind speed using logarithmic wind profile.
DEBUG:root:Calculating temperature using temperature gradient.
DEBUG:root:Calculating density using ideal gas equation.
DEBUG:root:Calculating power output using power curve.
```

Plot results

If you have matplotlib installed you can visualise the calculated power output and used power (coefficient) curves.

```
In [9]: # try to import matplotlib
        try:
            from matplotlib import pyplot as plt
            # matplotlib inline needed in notebook to plot inline
            %matplotlib inline
        except ImportError:
            plt = None
```

```
In [10]: # plot turbine power output
         if plt:
             e126.power_output.plot(legend=True, label='Enercon E126')
             my_turbine.power_output.plot(legend=True, label='myTurbine')
             plt.show()
```

```
In [11]: # plot power (coefficient) curves
         if plt:
             if e126.power_coefficient_curve is not None:
                 e126.power_coefficient_curve.plot(
                     x='wind_speed', y='values', style='*',
                     title='Enercon E126 power coefficient curve')
                 plt.show()
             if e126.power_curve is not None:
                 e126.power_curve.plot(x='wind_speed', y='values', style='*',
                                       title='Enercon E126 power curve')
                 plt.show()
             if my_turbine.power_coefficient_curve is not None:
                 my_turbine.power_coefficient_curve.plot(
                     x='wind_speed', y='values', style='*',
                     title='myTurbine power coefficient curve')
                 plt.show()
             if my_turbine.power_curve is not None:
                 my_turbine.power_curve.plot(x='wind_speed', y='values', style='*',
                                             title='myTurbine power curve')
                 plt.show()
```

```
In [12]:
```

CHAPTER 3

What's New

These are new features and improvements of note in each release

Releases

v0.0.6 (July 07, 2017)

Documentation

- added basic usage section and jupyter notebook

Testing

- added tests for different data types and wind_turbine module
- added example to tests
- added and fixed doctests

Other changes

- various API changes due to having better comprehensible function and variable names
- renamed Modelchain to ModelChain
- moved functions for temperature calculation to temperature module

- new function for interpolation and extrapolation

Contributors

- Sabine Haas
- Birgit Schachler

v0.0.5 (April 10, 2017)

New features

- complete restructuring of the windpowerlib
- power curves for numerous wind turbines are provided
- new function for calculation of air density at hub height (rho_ideal_gas)
- new function for calculation of wind speed at hub height (v_wind_hellman)
- density correction for calculation of power output
- modelchain for convenient usage

Documentation

- added references

Testing

- tests for all modules were added

Contributors

- Sabine Haas
- Birgit Schachler

CHAPTER 4

API

Classes

wind_turbine.WindTurbine(turbine_name, ...)	Defines a standard set of wind turbine attributes.
modelchain.ModelChain(wind_turbine[, ...])	Model to determine the output of a wind turbine

windpowerlib.wind_turbine.WindTurbine

class windpowerlib.wind_turbine.**WindTurbine**(*turbine_name*, *hub_height*, *rotor_diameter=None*, *power_coefficient_curve=None*, *power_curve=None*, *nominal_power=None*, *fetch_curve='power_curve'*)

Defines a standard set of wind turbine attributes.

Parameters

- **turbine_name** (*string*) – Name of the wind turbine type. Use get_turbine_types() to see a list of all wind turbines for which power (coefficient) curve data is provided.
- **hub_height** (*float*) – Hub height of the wind turbine in m.
- **rotor_diameter** (*None or float*) – Diameter of the rotor in m.
- **power_coefficient_curve** (*None, pandas.DataFrame or dictionary*) – Power coefficient curve of the wind turbine. DataFrame/dictionary must have 'wind_speed' and 'values' columns/keys with wind speeds in m/s and the corresponding power coefficients. Default: None.
- **power_curve** (*None, pandas.DataFrame or dictionary*) – Power curve of the wind turbine. DataFrame/dictionary must have 'wind_speed' and 'values' columns/keys with wind speeds in m/s and the corresponding power curve value in W. Default: None.
- **nominal_power** (*None or float*) – The nominal output of the wind turbine in W.

- **fetch_curve** (*string*) – Parameter to specify whether the power or power coefficient curve should be retrieved from the provided turbine data. Valid options are 'power_curve' and 'power_coefficient_curve'. Default: 'power_curve'.

turbine_name
string – Name of the wind turbine type. Use get_turbine_types() to see a list of all wind turbines for which power (coefficient) curve data is provided.

hub_height
float – Hub height of the wind turbine in m.

rotor_diameter
None or float – Diameter of the rotor in m.

power_coefficient_curve
None, pandas.DataFrame or dictionary – Power coefficient curve of the wind turbine. DataFrame/dictionary must have 'wind_speed' and 'values' columns/keys with wind speeds in m/s and the corresponding power coefficients. Default: None.

power_curve
None, pandas.DataFrame or dictionary – Power curve of the wind turbine. DataFrame/dictionary must have 'wind_speed' and 'values' columns/keys with wind speeds in m/s and the corresponding power curve value in W. Default: None.

nominal_power
None or float – The nominal output of the wind turbine in W.

fetch_curve
string – Parameter to specify whether the power or power coefficient curve should be retrieved from the provided turbine data. Valid options are 'power_curve' and 'power_coefficient_curve'. Default: 'power_curve'.

power_output
pandas.Series – The calculated power output of the wind turbine.

Examples

```
>>> from windpowerlib import wind_turbine
>>> enerconE126 = {
...     'hub_height': 135,
...     'rotor_diameter': 127,
...     'turbine_name': 'ENERCON E 126 7500'}
>>> e126 = wind_turbine.WindTurbine(**enerconE126)
>>> print(e126.nominal_power)
7500000
```

__init__(*turbine_name*, *hub_height*, *rotor_diameter=None*, *power_coefficient_curve=None*, *power_curve=None*, *nominal_power=None*, *fetch_curve='power_curve'*)

Methods

__init__(turbine_name, hub_height[, ...])	
fetch_turbine_data()	Fetches data of the requested wind turbine.

windpowerlib.modelchain.ModelChain

class windpowerlib.modelchain.**ModelChain**(*wind_turbine*, *wind_speed_model='logarithmic'*, *temperature_model='linear_gradient'*, *density_model='barometric'*, *power_output_model='power_curve'*, *density_correction=False*, *obstacle_height=0*, *hellman_exp=None*)

Model to determine the output of a wind turbine

Parameters

- **wind_turbine** (WindTurbine) – A *WindTurbine* object representing the wind turbine.
- **wind_speed_model** (*string*) – Parameter to define which model to use to calculate the wind speed at hub height. Valid options are 'logarithmic', 'hellman' and 'interpolation_extrapolation'. Default: 'logarithmic'.
- **temperature_model** (*string*) – Parameter to define which model to use to calculate the temperature of air at hub height. Valid options are 'linear_gradient' and 'interpolation_extrapolation'. Default: 'linear_gradient'.
- **density_model** (*string*) – Parameter to define which model to use to calculate the density of air at hub height. Valid options are 'barometric', 'ideal_gas' and 'interpolation_extrapolation'. Default: 'barometric'.
- **power_output_model** (*string*) – Parameter to define which model to use to calculate the turbine power output. Valid options are 'power_curve' and 'power_coefficient_curve'. Default: 'power_curve'.
- **density_correction** (*boolean*) – If the parameter is True the density corrected power curve is used for the calculation of the turbine power output. Default: False.
- **obstacle_height** (*float*) – Height of obstacles in the surrounding area of the wind turbine in m. Set *obstacle_height* to zero for wide spread obstacles. Default: 0.
- **hellman_exp** (*float*) – The Hellman exponent, which combines the increase in wind speed due to stability of atmospheric conditions and surface roughness into one constant. Default: None.

wind_turbine
WindTurbine – A *WindTurbine* object representing the wind turbine.

wind_speed_model
string – Parameter to define which model to use to calculate the wind speed at hub height. Valid options are 'logarithmic', 'hellman' and 'interpolation_extrapolation'. Default: 'logarithmic'.

temperature_model
string – Parameter to define which model to use to calculate the temperature of air at hub height. Valid options are 'linear_gradient' and 'interpolation_extrapolation'. Default: 'linear_gradient'.

density_model
string – Parameter to define which model to use to calculate the density of air at hub height. Valid options are 'barometric', 'ideal_gas' and 'interpolation_extrapolation'. Default: 'barometric'.

power_output_model
string – Parameter to define which model to use to calculate the turbine power output. Valid options are 'power_curve' and 'power_coefficient_curve'. Default: 'power_curve'.

density_correction
boolean – If the parameter is True the density corrected power curve is used for the calculation of the turbine power output. Default: False.

hellman_exp
float – The Hellman exponent, which combines the increase in wind speed due to stability of atmospheric conditions and surface roughness into one constant. Default: None.

obstacle_height
float – Height of obstacles in the surrounding area of the wind turbine in m. Set *obstacle_height* to zero for wide spread obstacles. Default: 0.

power_output
pandas.Series – Electrical power output of the wind turbine in W.

Examples

```
>>> from windpowerlib import modelchain
>>> from windpowerlib import wind_turbine
>>> enerconE126 = {
...     'hub_height': 135,
...     'rotor_diameter': 127,
...     'turbine_name': 'ENERCON E 126 7500'}
>>> e126 = wind_turbine.WindTurbine(**enerconE126)
>>> modelchain_data = {'density_model': 'ideal_gas'}
>>> e126_mc = modelchain.ModelChain(e126, **modelchain_data)
>>> print(e126_mc.density_model)
ideal_gas
```

__init__(*wind_turbine*, *wind_speed_model='logarithmic'*, *temperature_model='linear_gradient'*, *density_model='barometric'*, *power_output_model='power_curve'*, *density_correction=False*, *obstacle_height=0*, *hellman_exp=None*)

Methods

__init__(wind_turbine[, wind_speed_model, ...])	
density_hub(weather_df)	Calculates the density of air at hub height.
run_model(weather_df)	Runs the model.
temperature_hub(weather_df)	Calculates the temperature of air at hub height.
turbine_power_output(wind_speed_hub, density_hub)	Calculates the power output of the wind turbine.
wind_speed_hub(weather_df)	Calculates the wind speed at hub height.

Temperature

Function for calculating air temperature at hub height.

temperature.linear_gradient(temperature, ...)	Calculates the temperature at hub height using a linear gradient.

windpowerlib.temperature.linear_gradient

windpowerlib.temperature.**linear_gradient**(*temperature*, *temperature_height*, *hub_height*)

Calculates the temperature at hub height using a linear gradient.

A linear temperature gradient of -6.5 K/km is assumed. This function is carried out when the parameter *temperature_model* of an instance of the `ModelChain` class is 'temperature_gradient'.

Parameters

- **temperature** (*pandas.Series or numpy.array*) – Air temperature in K.
- **temperature_height** (*float*) – Height in m for which the parameter *temperature* applies.
- **hub_height** (*float*) – Hub height of wind turbine in m.

Returns Temperature at hub height in K.

Return type pandas.Series or numpy.array

Notes

The following equation is used[1]:

$$T_{hub} = T_{air} - 0.0065 \cdot (h_{hub} - h_{T,data})$$

with: T: temperature [K], h: height [m]

$h_{T,data}$ is the height in which the temperature T_{air} is measured and T_{hub} is the temperature at hub height h_{hub} of the wind turbine.

Assumptions:

•Temperature gradient of -6.5 K/km (-0.0065 K/m)

References

Density

Functions for calculating air density at hub height.

density.barometric(pressure, ...)	Calculates the density of air at hub height using the barometric height equation.
density.ideal_gas(pressure, pressure_height, ...)	Calculates the density of air at hub height using the ideal gas equation.

windpowerlib.density.barometric

windpowerlib.density.**barometric**(*pressure*, *pressure_height*, *hub_height*, *temperature_hub_height*)

Calculates the density of air at hub height using the barometric height equation.

[1] ICAO-Standardatmosphäre (ISA). http://www.dwd.de/DE/service/lexikon/begriffe/S/Standardatmosphaere_pdf.pdf?__blob=publicationFile&v=3

This function is carried out when the parameter *density_model* of an instance of the `ModelChain` class is 'barometric'.

Parameters

- **pressure** (*pandas.Series or numpy.array*) – Air pressure in Pa.
- **pressure_height** (*float*) – Height in m for which the parameter *pressure* applies.
- **hub_height** (*float*) – Hub height of wind turbine in m.
- **temperature_hub_height** (*pandas.Series or numpy.array*) – Air temperature at hub height in K.

Returns Density of air at hub height in kg/m^3. Returns a pandas.Series if one of the input parameters is a pandas.Series.

Return type pandas.Series or numpy.array

Notes

The following equation is used[1,2] :

$$\rho_{hub} = \left(p/100 - (h_{hub} - h_{p,data}) \cdot \frac{1}{8}\right) \cdot \frac{\rho_0 T_0 \cdot 100}{p_0 T_{hub}}$$

with: T: temperature [K], h: height [m], ρ: density [kg/m^3], p: pressure [Pa]

$h_{p,data}$ is the height of the measurement or model data for pressure, p_0 the ambient air pressure, ρ_0 the ambient density of air, T_0 the ambient temperature and T_{hub} the temperature at hub height h_{hub}.

Assumptions:

•Pressure gradient of -1/8 hPa/m

References

windpowerlib.density.ideal_gas

windpowerlib.density.**ideal_gas**(*pressure*, *pressure_height*, *hub_height*, *temperature_hub_height*)

Calculates the density of air at hub height using the ideal gas equation.

This function is carried out when the parameter *density_model* of an instance of the `ModelChain` class is 'ideal_gas'.

Parameters

- **pressure** (*pandas.Series or numpy.array*) – Air pressure in Pa.
- **pressure_height** (*float*) – Height in m for which the parameter *pressure* applies.
- **hub_height** (*float*) – Hub height of wind turbine in m.
- **temperature_hub_height** (*pandas.Series or numpy.array*) – Air temperature at hub height in K.

Returns Density of air at hub height in kg/m^3. Returns a pandas.Series if one of the input parameters is a pandas.Series.

[1] Hau, E.: "Windkraftanlagen - Grundlagen, Technik, Einsatz, Wirtschaftlichkeit". 4. Auflage, Springer-Verlag, 2008, p. 560

[2] Deutscher Wetterdienst: http://www.dwd.de/DE/service/lexikon/begriffe/D/Druckgradient_pdf.pdf?__blob=publicationFile&v=4

Return type pandas.Series or numpy.array

Notes

The following equations are used[1],[2]:

$$\rho_{hub} = p_{hub}/(R_s T_{hub})$$

and[3]:

$$p_{hub} = \left(p/100 - (h_{hub} - h_{p,data}) \cdot \frac{1}{8}\right) \cdot 100$$

with: T: temperature [K], ρ: density [kg/m^3], p: pressure [Pa]

$h_{p,data}$ is the height of the measurement or model data for pressure, R_s is the specific gas constant of dry air (287.058 J/(kg*K)) and p_{hub} is the pressure at hub height h_{hub}.

References

Wind speed

Functions for calculating wind speed at hub height.

`wind_speed.logarithmic_profile`(wind_speed, ...)	Calculates the wind speed at hub height using a logarithmic wind profile.
`wind_speed.hellman`(wind_speed, ...[, ...])	Calculates the wind speed at hub height using the hellman equation.

windpowerlib.wind_speed.logarithmic_profile

windpowerlib.wind_speed.**logarithmic_profile**(*wind_speed*, *wind_speed_height*, *hub_height*, *roughness_length*, *obstacle_height=0.0*)

Calculates the wind speed at hub height using a logarithmic wind profile.

The logarithmic height equation is used. There is the possibility of including the height of the surrounding obstacles in the calculation. This function is carried out when the parameter *wind_speed_model* of an instance of the `ModelChain` class is 'logarithmic'.

Parameters

- **wind_speed** (*pandas.Series or numpy.array*) – Wind speed time series.
- **wind_speed_height** (*float*) – Height for which the parameter *wind_speed* applies.
- **hub_height** (*float*) – Hub height of wind turbine.

[1] Ahrendts J., Kabelac S.: "Das Ingenieurwissen - Technische Thermodynamik". 34. Auflage, Springer-Verlag, 2014, p. 23

[2] Biank, M.: "Methodology, Implementation and Validation of a Variable Scale Simulation Model for Windpower based on the Georeferenced Installation Register of Germany". Master's Thesis at RLI, 2014, p. 57

[3] Deutscher Wetterdienst: http://www.dwd.de/DE/service/lexikon/begriffe/D/Druckgradient_pdf.pdf?__blob=publicationFile&v=4

- **roughness_length** (*pandas.Series or numpy.array or float*) – Roughness length.
- **obstacle_height** (*float*) – Height of obstacles in the surrounding area of the wind turbine. Set *obstacle_height* to zero for wide spread obstacles. Default: 0.

Returns Wind speed at hub height. Data type depends on type of *wind_speed*.

Return type pandas.Series or numpy.array

Notes

The following equation is used[1]:

$$v_{wind,hub} = v_{wind,data} \cdot \frac{\ln\left(\frac{h_{hub}-d}{z_0}\right)}{\ln\left(\frac{h_{data}-d}{z_0}\right)}$$

with: v: wind speed, h: height, z_0: roughness length, d: boundary layer offset (estimated by d = 0.7 * *obstacle_height*)

For d = 0 it results in the following equation[2],[3]:

$$v_{wind,hub} = v_{wind,data} \cdot \frac{\ln\left(\frac{h_{hub}}{z_0}\right)}{\ln\left(\frac{h_{data}}{z_0}\right)}$$

h_{data} is the height at which the wind speed $v_{wind,data}$ is measured and $v_{wind,hub}$ is the wind speed at hub height h_{hub} of the wind turbine.

Parameters *wind_speed_height*, *roughness_length*, *hub_height* and *obstacle_height* have to be of the same unit.

References

windpowerlib.wind_speed.hellman

windpowerlib.wind_speed.**hellman**(*wind_speed*, *wind_speed_height*, *hub_height*, *roughness_length=None*, *hellman_exponent=None*)

Calculates the wind speed at hub height using the hellman equation.

It is assumed that the wind profile follows a power law. This function is carried out when the parameter *wind_speed_model* of an instance of the *ModelChain* class is 'hellman'.

Parameters

- **wind_speed** (*pandas.Series or numpy.array*) – Wind speed time series.
- **wind_speed_height** (*float*) – Height for which the parameter *wind_speed* applies.
- **hub_height** (*float*) – Hub height of wind turbine.
- **roughness_length** (*pandas.Series or numpy.array or float*) – Roughness length. If given and *hellman_exponent* is None: *hellman_exponent* = 1 / ln(hub_height/roughness_length), otherwise *hellman_exponent* = 1/7. Default: None.

[1] Quaschning V.: "Regenerative Energiesysteme". München, Hanser Verlag, 2011, p. 278
[2] Gasch, R., Twele, J.: "Windkraftanlagen". 6. Auflage, Wiesbaden, Vieweg + Teubner, 2010, p. 129
[3] Hau, E.: "Windkraftanlagen - Grundlagen, Technik, Einsatz, Wirtschaftlichkeit". 4. Auflage, Springer-Verlag, 2008, p. 515

- **hellman_exponent** (*None or float*) – The Hellman exponent, which combines the increase in wind speed due to stability of atmospheric conditions and surface roughness into one constant. If None and roughness length is given *hellman_exponent* = 1 / ln(hub_height/roughness_length), otherwise *hellman_exponent* = 1/7. Default: None.

Returns Wind speed at hub height. Data type depends on type of *wind_speed*.

Return type pandas.Series or numpy.array

Notes

The following equation is used[1],[2],[3]:

$$v_{wind,hub} = v_{wind,data} \cdot \left(\frac{h_{hub}}{h_{data}} \right)^{\alpha}$$

with: v: wind speed, h: height, α: Hellman exponent

h_{data} is the height in which the wind speed $v_{wind,data}$ is measured and $v_{wind,hub}$ is the wind speed at hub height h_{hub} of the wind turbine.

For the Hellman exponent α many studies use a value of 1/7 for onshore and a value of 1/9 for offshore. The Hellman exponent can also be calulated by the following equation[2],[3]:

$$\alpha = \frac{1}{ln\left(\frac{h_{hub}}{z_0}\right)}$$

with: z_0: roughness length

Parameters *wind_speed_height*, *roughness_length*, *hub_height* and *obstacle_height* have to be of the same unit.

References

Wind turbine data

Functions and methods to obtain the nominal power as well as power curve or power coefficient curve needed by the *WindTurbine* class.

wind_turbine.WindTurbine.fetch_turbine_data()	Fetches data of the requested wind turbine.
wind_turbine.get_turbine_types([print_out])	Get the names of all possible wind turbine types for which the power coefficient curve or power curve is provided in the data files in the directory windpowerlib/data.
wind_turbine.read_turbine_data(**kwargs)	Fetches power (coefficient) curves from a file.

[1] Sharp, E.: "Spatiotemporal disaggregation of GB scenarios depicting increased wind capacity and electrified heat demand in dwellings". UCL, Energy Institute, 2015, p. 83

[2] Hau, E.: "Windkraftanlagen - Grundlagen, Technik, Einsatz, Wirtschaftlichkeit". 4. Auflage, Springer-Verlag, 2008, p. 517

[3] Quaschning V.: "Regenerative Energiesysteme". München, Hanser Verlag, 2011, p. 279

windpowerlib.wind_turbine.WindTurbine.fetch_turbine_data

WindTurbine.**fetch_turbine_data**()

Fetches data of the requested wind turbine.

Method fetches nominal power as well as power coefficient curve or power curve from a data file provided along with the windpowerlib. You can also use this function to import your own power (coefficient) curves. Therefore the wind speeds in m/s have to be in the first row and the corresponding power coefficient curve values or power curve values in W in a row where the first column contains the turbine name (See directory windpowerlib/data as reference).

Returns

Return type self

Examples

```
>>> from windpowerlib import wind_turbine
>>> enerconE126 = {
...     'hub_height': 135,
...     'rotor_diameter': 127,
...     'turbine_name': 'ENERCON E 126 7500',
...     'fetch_curve': 'power_coefficient_curve'}
>>> e126 = wind_turbine.WindTurbine(**enerconE126)
>>> print(e126.power_coefficient_curve['values'][5])
0.423
>>> print(e126.nominal_power)
7500000
```

windpowerlib.wind_turbine.get_turbine_types

windpowerlib.wind_turbine.**get_turbine_types**(*print_out=True*, ***kwargs*)

Get the names of all possible wind turbine types for which the power coefficient curve or power curve is provided in the data files in the directory windpowerlib/data.

Parameters **print_out** (*boolean*) – Directly prints the list of types if set to True. Default: True.

Examples

```
>>> from windpowerlib import wind_turbine
>>> turbines = wind_turbine.get_turbine_types(print_out=False)
>>> print(turbines[turbines["turbine_id"].str.contains("ENERCON")].iloc[0])
turbine_id    ENERCON E 101 3000
p_nom                    3000000
Name: 25, dtype: object
```

windpowerlib.wind_turbine.read_turbine_data

windpowerlib.wind_turbine.**read_turbine_data**(***kwargs*)

Fetches power (coefficient) curves from a file.

The data files are provided along with the windpowerlib and are located in the directory windpowerlib/data.

Other Parameters

- **datapath** (*string, optional*) – Path where the data file is stored. Default: './data'
- **filename** (*string, optional*) – Name of data file. Provided data files are 'power_curves.csv' and 'power_coefficient_curves.csv'. Default: 'power_curves.csv'.

Returns Power coefficient curve values (dimensionless) or power curve values in kW with corresponding wind speeds in m/s of all available wind turbines with turbine name in column 'turbine_id', turbine nominal power in column 'p_nom'.

Return type pandas.DataFrame

Power output

Functions for calculating power output of a wind turbine.

power_output.power_coefficient_curve(...[, ...])	Calculates the turbine power output using a power coefficient curve.
power_output.power_curve(wind_speed, ...[, ...])	Calculates the turbine power output using a power curve.
power_output.power_curve_density_correction(...)	Calculates the turbine power output using a density corrected power curve.

windpowerlib.power_output.power_coefficient_curve

windpowerlib.power_output.**power_coefficient_curve**(*wind_speed*, *power_coefficient_curve_wind_speeds*, *power_coefficient_curve_values*, *rotor_diameter*, *density*, *density_correction=False*)

Calculates the turbine power output using a power coefficient curve.

This function is carried out when the parameter *power_output_model* of an instance of the *ModelChain* class is 'power_coefficient_curve'. If the parameter *density_correction* is True the density corrected power curve (See *power_curve_density_correction()*) is used.

Parameters

- **wind_speed** (*pandas.Series or numpy.array*) – Wind speed at hub height in m/s.
- **power_coefficient_curve_wind_speeds** (*pandas.Series or numpy.array*) – Wind speeds in m/s for which the power coefficients are provided in *power_coefficient_curve_values*.
- **power_coefficient_curve_values** (*pandas.Series or numpy.array*) – Power coefficients corresponding to wind speeds in *power_coefficient_curve_wind_speeds*.
- **rotor_diameter** (*float*) – Rotor diameter in m.
- **density** (*pandas.Series or numpy.array*) – Density of air at hub height in kg/m^3.
- **density_correction** (*boolean*) – If the parameter is True the density corrected power curve is used for the calculation of the turbine power output. Default: False.

Returns Electrical power output of the wind turbine in W. Data type depends on type of *wind_speed*.

Return type pandas.Series or numpy.array

Notes

The following equation is used if the parameter *density_corr* is False[1],[2]:

$$P = \frac{1}{8} \cdot \rho_{hub} \cdot d_{rotor}^2 \cdot \pi \cdot v_{wind}^3 \cdot cp\left(v_{wind}\right)$$

with: P: power [W], ρ: density [kg/m^3], d: diameter [m], v: wind speed [m/s], cp: power coefficient

It is assumed that the power output for wind speeds above the maximum and below the minimum wind speed given in the power coefficient curve is zero.

References

windpowerlib.power_output.power_curve

windpowerlib.power_output.**power_curve**(*wind_speed*, *power_curve_wind_speeds*, *power_curve_values*, *density=None*, *density_correction=False*)

Calculates the turbine power output using a power curve.

This function is carried out when the parameter *power_output_model* of an instance of the `ModelChain` class is 'power_curve'. If the parameter *density_correction* is True the density corrected power curve (See `power_curve_density_correction()`) is used.

Parameters

- **wind_speed** (*pandas.Series or numpy.array*) – Wind speed at hub height in m/s.
- **power_curve_wind_speeds** (*pandas.Series or numpy.array*) – Wind speeds in m/s for which the power curve values are provided in *power_curve_values*.
- **power_curve_values** (*pandas.Series or numpy.array*) – Power curve values corresponding to wind speeds in *power_curve_wind_speeds*.
- **density** (*pandas.Series or numpy.array*) – Density of air at hub height in kg/m^3. This parameter is needed if *density_correction* is True. Default: None.
- **density_correction** (*boolean*) – If the parameter is True the density corrected power curve is used for the calculation of the turbine power output. In this case *density* cannot be None. Default: False.

Returns Electrical power output of the wind turbine in W. Data type depends on type of *wind_speed*.

Return type pandas.Series or numpy.array

Notes

It is assumed that the power output for wind speeds above the maximum and below the minimum wind speed given in the power curve is zero.

[1] Gasch, R., Twele, J.: "Windkraftanlagen". 6. Auflage, Wiesbaden, Vieweg + Teubner, 2010, pages 35ff, 208
[2] Hau, E.: "Windkraftanlagen - Grundlagen, Technik, Einsatz, Wirtschaftlichkeit". 4. Auflage, Springer-Verlag, 2008, p. 542

windpowerlib.power_output.power_curve_density_correction

windpowerlib.power_output.**power_curve_density_correction**(*wind_speed*, *power_curve_wind_speeds*, *power_curve_values*, *density*)

Calculates the turbine power output using a density corrected power curve.

This function is carried out when the parameter *density_correction* of an instance of the `ModelChain` class is True.

Parameters

- **wind_speed** (*pandas.Series or numpy.array*) – Wind speed at hub height in m/s.
- **power_curve_wind_speeds** (*pandas.Series or numpy.array*) – Wind speeds in m/s for which the power curve values are provided in *power_curve_values*.
- **power_curve_values** (*pandas.Series or numpy.array*) – Power curve values corresponding to wind speeds in *power_curve_wind_speeds*.
- **density** (*pandas.Series or numpy.array*) – Density of air at hub height in kg/m^3.

Returns Electrical power output of the wind turbine in W. Data type depends on type of *wind_speed*.

Return type pandas.Series or numpy.array

Notes

The following equation is used for the wind speed at site[1],[2],[3]:

$$v_{site} = v_{std} \cdot \left(\frac{\rho_0}{\rho_{site}} \right)^{p(v)}$$

with:

$$p = \begin{cases} \frac{1}{3} & v_{std} \leq 7.5 \text{ m/s} \\ \frac{1}{15} \cdot v_{std} - \frac{1}{6} & 7.5 \text{ m/s} < v_{std} < 12.5 \text{ m/s} \\ \frac{2}{3} & \geq 12.5 \text{ m/s} \end{cases},$$

v: wind speed [m/s], ρ: density [kg/m^3]

v_{std} is the standard wind speed in the power curve (v_{std}, P_{std}), v_{site} is the density corrected wind speed for the power curve (v_{site}, P_{std}), ρ_0 is the ambient density (1.225 kg/m^3) and ρ_{site} the density at site conditions (and hub height).

It is assumed that the power output for wind speeds above the maximum and below the minimum wind speed given in the power curve is zero.

[1] Svenningsen, L.: "Power Curve Air Density Correction And Other Power Curve Options in WindPRO". 1st edition, Aalborg, EMD International A/S , 2010, p. 4

[2] Svenningsen, L.: "Proposal of an Improved Power Curve Correction". EMD International A/S , 2010

[3] Biank, M.: "Methodology, Implementation and Validation of a Variable Scale Simulation Model for Windpower based on the Georeferenced Installation Register of Germany". Master's Thesis at Reiner Lemoine Institute, 2014, p. 13

References

ModelChain

Creating a ModelChain object.

modelchain.ModelChain(wind_turbine[, ...])	Model to determine the output of a wind turbine

Running the ModelChain.

modelchain.ModelChain.run_model(weather_df)	Runs the model.

windpowerlib.modelchain.ModelChain.run_model

ModelChain.**run_model**(*weather_df*)

Runs the model.

Parameters **weather_df** (*pandas.DataFrame*) – DataFrame with time series for wind speed *wind_speed* in m/s, and roughness length *roughness_length* in m, as well as optionally temperature *temperature* in K, pressure *pressure* in Pa and density *density* in kg/m^3 depending on *power_output_model* and *density_model chosen*. The columns of the DataFrame are a MultiIndex where the first level contains the variable name (e.g. wind_speed) and the second level contains the height at which it applies (e.g. 10, if it was measured at a height of 10 m). See below for an example on how to create the weather_df DataFrame.

Returns

Return type self

Examples

```
>>> import numpy as np
>>> import pandas as pd
>>> weather_df = pd.DataFrame(np.random.rand(2,6),
...                           index=pd.date_range('1/1/2012',
...                                               periods=2,
...                                               freq='H'),
...                           columns=[np.array(['wind_speed',
...                                              'wind_speed',
...                                              'temperature',
...                                              'temperature',
...                                              'pressure',
...                                              'roughness_length']),
...                                    np.array([10, 80, 10, 80,
...                                              10, 0])])
>>> weather_df.columns.get_level_values(0)[0]
'wind_speed'
```

Methods of the ModelChain object.

modelchain.ModelChain.temperature_hub(weather_df)	Calculates the temperature of air at hub height.
modelchain.ModelChain.density_hub(weather_df)	Calculates the density of air at hub height.
modelchain.ModelChain.wind_speed_hub(weather_df)	Calculates the wind speed at hub height.
modelchain.ModelChain.turbine_power_output(...)	Calculates the power output of the wind turbine.

windpowerlib.modelchain.ModelChain.temperature_hub

ModelChain.**temperature_hub**(*weather_df*)

Calculates the temperature of air at hub height.

The temperature is calculated using the method specified by the parameter *temperature_model*.

Parameters **weather_df** (*pandas.DataFrame*) – DataFrame with time series for temperature *temperature* in K. The columns of the DataFrame are a MultiIndex where the first level contains the variable name (e.g. temperature) and the second level contains the height at which it applies (e.g. 10, if it was measured at a height of 10 m). See documentation of *ModelChain.run_model()* for an example on how to create the weather_df DataFrame.

Returns **temperature_hub** – Temperature of air in K at hub height.

Return type pandas.Series or numpy.array

Notes

If *weather_df* contains temperatures at different heights the given temperature(s) closest to the hub height are used.

windpowerlib.modelchain.ModelChain.density_hub

ModelChain.**density_hub**(*weather_df*)

Calculates the density of air at hub height.

The density is calculated using the method specified by the parameter *density_model*. Previous to the calculation of the density the temperature at hub height is calculated using the method specified by the parameter *temperature_model*.

Parameters **weather_df** (*pandas.DataFrame*) – DataFrame with time series for temperature *temperature* in K, pressure *pressure* in Pa and/or density *density* in kg/m^3, depending on the *density_model* used. The columns of the DataFrame are a MultiIndex where the first level contains the variable name (e.g. temperature) and the second level contains the height at which it applies (e.g. 10, if it was measured at a height of 10 m). See documentation of *ModelChain.run_model()* for an example on how to create the weather_df DataFrame.

Returns **density_hub** – Density of air in kg/m^3 at hub height.

Return type pandas.Series or numpy.array

Notes

If *weather_df* contains data at different heights the data closest to the hub height are used. If *interpolation_extrapolation* is used to calculate the density at hub height, the *weather_df* must contain at least two time series for density.

windpowerlib.modelchain.ModelChain.wind_speed_hub

ModelChain.**wind_speed_hub**(*weather_df*)

Calculates the wind speed at hub height.

The method specified by the parameter *wind_speed_model* is used.

Parameters **weather_df** (*pandas.DataFrame*) – DataFrame with time series for wind speed *wind_speed* in m/s and roughness length *roughness_length* in m. The columns of the DataFrame are a MultiIndex where the first level contains the variable name (e.g. wind_speed) and the second level contains the height at which it applies (e.g. 10, if it was measured at a height of 10 m). See documentation of *ModelChain.run_model()* for an example on how to create the weather_df DataFrame.

Returns **wind_speed_hub** – Wind speed in m/s at hub height.

Return type pandas.Series or numpy.array

Notes

If *weather_df* contains wind speeds at different heights the given wind speed(s) closest to the hub height are used.

windpowerlib.modelchain.ModelChain.turbine_power_output

ModelChain.**turbine_power_output**(*wind_speed_hub*, *density_hub*)

Calculates the power output of the wind turbine.

The method specified by the parameter *power_output_model* is used.

Parameters

- **wind_speed_hub** (*pandas.Series or numpy.array*) – Wind speed at hub height in m/s.
- **density_hub** (*pandas.Series or numpy.array*) – Density of air at hub height in kg/m^3.

Returns Electrical power output of the wind turbine in W.

Return type pandas.Series

Tools

Additional functions used in the windpowerlib.

Continued on next page

Table 4.12 – continued from previous page

tools.linear_interpolation_extrapolation(df, ...)	Inter- or extrapolates between the values of a data frame.

windpowerlib.tools.linear_interpolation_extrapolation

windpowerlib.tools.**linear_interpolation_extrapolation**(*df*, *target_height*)

Inter- or extrapolates between the values of a data frame.

This function can be used for the inter-/extrapolation of a parameter (e.g wind speed) available at two or more different heights, to approximate the value at hub height. The function is carried out when the parameter *wind_speed_model*, *density_model* or *temperature_model* of an instance of the *ModelChain* class is 'interpolation_extrapolation'.

Parameters

- **df** (*pandas.DataFrame*) – DataFrame with time series for parameter that is to be interpolated or extrapolated. The columns of the DataFrame are the different heights for which the parameter is available. If more than two heights are given, the two closest heights are used. See example below on how the DataFrame should look like and how the function can be used.
- **target_height** (*float*) – Height for which the parameter is approximated (e.g. hub height).

Returns Result of the inter-/extrapolation (e.g. wind speed at hub height).

Return type pandas.Series

Notes

For the inter- and extrapolation the following equation is used:

$$f(x) = \frac{(f(x_2) - f(x_1))}{(x_2 - x_1)} \cdot (x - x_1) + f(x_1)$$

Examples

```
>>> import numpy as np
>>> import pandas as pd
>>> wind_speed_10m = np.array([[3], [4]])
>>> wind_speed_80m = np.array([[6], [6]])
>>> weather_df = pd.DataFrame(np.hstack((wind_speed_10m,
...                                      wind_speed_80m)),
...                           index=pd.date_range('1/1/2012',
...                                               periods=2,
...                                               freq='H'),
...                           columns=[np.array(['wind_speed',
...                                              'wind_speed']),
...                                    np.array([10, 80])])
>>> linear_interpolation_extrapolation(
...     weather_df['wind_speed'], 100)[0]
6.8571428571428577
```

Example

The basic example consists of the following functions.

basic_example.get_weather_data([filename])	Imports weather data from a file.
basic_example.initialise_wind_turbines()	Initialises two *WindTurbine* objects.
basic_example.calculate_power_output(...)	Calculates power output of wind turbines using the *ModelChain*.
basic_example.plot_or_print(my_turbine, e126)	Plots or prints power output and power (coefficient) curves.
basic_example.run_basic_example()	Run the basic example.

example.basic_example.get_weather_data

example.basic_example.**get_weather_data**(*filename='weather.csv'*, ***kwargs*)

Imports weather data from a file.

The data include wind speed at two different heights in m/s, air temperature in two different heights in K, surface roughness length in m and air pressure in Pa. The file is located in the example folder of the windpowerlib. The height in m for which the data applies is specified in the second row.

Parameters **filename** (*string*) – Filename of the weather data file. Default: 'weather.csv'.

Other Parameters **datapath** (*string, optional*) – Path where the weather data file is stored. Default: 'windpowerlib/example'.

Returns **weather_df** – DataFrame with time series for wind speed *wind_speed* in m/s, temperature *temperature* in K, roughness length *roughness_length* in m, and pressure *pressure* in Pa. The columns of the DataFrame are a MultiIndex where the first level contains the variable name as string (e.g. 'wind_speed') and the second level contains the height as integer at which it applies (e.g. 10, if it was measured at a height of 10 m).

Return type pandas.DataFrame

example.basic_example.initialise_wind_turbines

example.basic_example.**initialise_wind_turbines**()

Initialises two *WindTurbine* objects.

Function shows two ways to initialise a WindTurbine object. You can either specify your own turbine, as done below for 'myTurbine', or fetch power and/or power coefficient curve data from data files provided by the windpowerlib, as done for the 'enerconE126'. Execute `windpowerlib.wind_turbine.get_turbine_types()` or `windpowerlib.wind_turbine.get_turbine_types( filename='power_coefficient_curves.csv')` to get a list of all wind turbines for which power and power coefficient curves respectively are provided.

Returns

Return type Tuple (*WindTurbine*, *WindTurbine*)

example.basic_example.calculate_power_output

example.basic_example.**calculate_power_output**(*weather*, *my_turbine*, *e126*)

Calculates power output of wind turbines using the *ModelChain*.

The *ModelChain* is a class that provides all necessary steps to calculate the power output of a wind turbine. You can either use the default methods for the calculation steps, as done for 'my_turbine', or choose different methods, as done for the 'e126'.

Parameters

- **weather** (*pd.DataFrame*) – Contains weather data time series.
- **my_turbine** (WindTurbine) – WindTurbine object with self provided power curve.
- **e126** (WindTurbine) – WindTurbine object with power curve from data file provided by the windpowerlib.

example.basic_example.plot_or_print

example.basic_example.**plot_or_print**(*my_turbine*, *e126*)

Plots or prints power output and power (coefficient) curves.

Parameters

- **my_turbine** (WindTurbine) – WindTurbine object with self provided power curve.
- **e126** (WindTurbine) – WindTurbine object with power curve from data file provided by the windpowerlib.

example.basic_example.run_basic_example

example.basic_example.**run_basic_example**()

Run the basic example.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Index

Symbols

B

C

D

F

G

H

I

L

M

N

O

P

R

T

W