windpowerlib Documentation

oemof developer group

Sep 10, 2019

Contents

1 Getting started **3**
- 1.1 Introduction . . . 3
- 1.2 Documentation . . . 3
- 1.3 Installation . . . 3
- 1.4 Examples and basic usage . . . 4
- 1.5 Wind turbine data . . . 4
- 1.6 Contributing . . . 4
- 1.7 Citing the windpowerlib . . . 5
- 1.8 License . . . 5

2 Examples **7**
- 2.1 ModelChain example . . . 7
- 2.2 TurbineClusterModelChain example . . . 14

3 Model description **19**
- 3.1 Wind power plants . . . 19
- 3.2 Height correction and conversion of weather data . . . 19
- 3.3 Power output calculations . . . 19
- 3.4 Wake losses . . . 20
- 3.5 Smoothing of power curves . . . 20
- 3.6 The modelchains . . . 20

4 What's New **21**
- 4.1 v0.2.0 (September 9, 2019) . . . 21
- 4.2 v0.1.3 (September 2, 2019) . . . 22
- 4.3 v0.1.2 (June 24, 2019) . . . 23
- 4.4 v0.1.1 (January 31, 2019) . . . 24
- 4.5 v0.1.0 (January 17, 2019) . . . 24
- 4.6 v0.0.6 (July 07, 2017) . . . 25
- 4.7 v0.0.5 (April 10, 2017) . . . 26

5 API **27**
- 5.1 Classes . . . 27
- 5.2 Temperature . . . 38
- 5.3 Density . . . 39
- 5.4 Wind speed . . . 40
- 5.5 Wind turbine data . . . 43

5.6 Data Container . . . 45
5.7 Wind farm calculations . . . 46
5.8 Wind turbine cluster calculations . . . 48
5.9 Power output . . . 50
5.10 Alteration of power curves . . . 53
5.11 Wake losses . . . 55
5.12 ModelChain . . . 56
5.13 TurbineClusterModelChain . . . 59
5.14 Tools . . . 62
5.15 ModelChain example . . . 65
5.16 TurbineClusterModelChain example . . . 67

6 Indices and tables **69**

Index **71**

Contents:

CHAPTER 1

Getting started

1.1 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 photovoltaic) 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.

1.2 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.

1.3 Installation

If you have a working Python 3 environment, use pypi to install the latest windpowerlib version:

```
pip install windpowerlib
```

The windpowerlib is designed for Python 3 and tested on Python >= 3.5. We highly recommend to use virtual environments. Please see the installation page of the oemof documentation for complete instructions on how to install python and a virtual environment on your operating system.

1.3.1 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 pip:

```
pip install matplotlib
```

1.4 Examples and basic usage

The basic usage of the windpowerlib is shown in the ModelChain example that is available as jupyter notebook and python script:

- `ModelChain example (Python script)`
- `ModelChain example (Jupyter notebook)`

To run the example you need the example weather and turbine data used:

- `Example weather data file`
- `Example power curve data file`
- `Example power coefficient curve data file`
- `Example nominal power data file`

Furthermore, you have to install the windpowerlib and to run the notebook you also need to install *notebook* using pip3. To launch jupyter notebook type `jupyter notebook` in the 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.

Further functionalities, like the modelling of wind farms and wind turbine clusters, are shown in the TurbineClusterModelChain example. As the ModelChain example it is available as jupyter notebook and as python script. The weather and turbine data in this example is the same as in the example above.

- `TurbineClusterModelChain example (Python script)`
- `TurbineClusterModelChain example (Jupyter notebook)`

You can also look at the examples in the *Examples* section.

1.5 Wind turbine data

The windpowerlib provides wind turbine data (power curves, hub heights, etc.) for a large set of wind turbines. See *Initialize wind turbine* in *Examples* on how to use this data in your simulations.

The dataset is hosted and maintained on the OpenEnergy database (oedb). To update your local files with the latest version of the oedb turbine library you can execute the following in your python console:

```
from windpowerlib.wind_turbine import load_turbine_data_from_oedb
load_turbine_data_from_oedb()
```

We would like to encourage anyone to contribute to the turbine library by adding turbine data or reporting errors in the data. See here for more information on how to contribute.

1.6 Contributing

We are warmly welcoming all who want to contribute to the windpowerlib. If you are interested in wind models and want to help improving the existing model do not hesitate to contact us via github or email (windpowerlib@rl-institut.de).

Clone: https://github.com/wind-python/windpowerlib and install the cloned repository using pip:

```
pip install -e /path/to/the/repository
```

As the windpowerlib started with contributors from the oemof developer group we use the same developer rules.

How to create a pull request:

- Fork the windpowerlib repository to your own github account.
- Change, add or remove code.
- Commit your changes.
- Create a pull request and describe what you will do and why.
- Wait for approval.

Generally the following steps are required when changing, adding or removing code:

- Add new tests if you have written new functions/classes.
- Add/change the documentation (new feature, API changes . . .).
- Add a whatsnew entry and your name to Contributors.
- Check if all tests still work by simply executing pytest in your windpowerlib directory:

```
pytest
```

1.7 Citing the windpowerlib

We use the zenodo project to get a DOI for each version. Search zenodo for the right citation of your windpowerlib version.

1.8 License

Copyright (C) 2017 oemof developer group

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see http://www.gnu.org/licenses/.

CHAPTER 2

Examples

2.1 ModelChain example

This example shows you the basic usage of the windpowerlib by using the `ModelChain` class. 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 feed-in time series.

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

2.1.1 Import necessary packages and modules

[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.

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

2.1.2 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 analyze. The function below imports example weather data from the weather.csv file provided along with the windpowerlib.

The data includes 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.

[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 : str
        Filename of the weather data file. Default: 'weather.csv'.

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

    Returns
    -------
    weather_df : :pandas:`pandas.DataFrame<frame>`
        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],
        date_parser=lambda idx: pd.to_datetime(idx, utc=True))

    # change type of index to datetime and set time zone
    weather_df.index = pd.to_datetime(weather_df.index).tz_convert(
        'Europe/Berlin')

    # change type of height from str to int by resetting columns
    l0 = [_[0] for _ in weather_df.columns]
    l1 = [int(_[1]) for _ in weather_df.columns]
    weather_df.columns = [l0, l1]
```

(continues on next page)

(continued from previous page)

```
    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
```

2.1.3 Initialize wind turbine

To initialize 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 three ways to initialize a WindTurbine object in the windpowerlib. You can either use turbine data from the OpenEnergy Database (oedb) turbine library that is provided along with the windpowerlib, as done for the 'enercon_e126', or specify your own turbine by directly providing a power (coefficient) curve, as done below for 'my_turbine', or provide your own turbine data in csv files, as done for 'dummy_turbine'.

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

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

# find all Enercons
print(df[df["manufacturer"].str.contains("Enercon")])
```

```
INFO:root:Data base connection successful.
```

```
   manufacturer turbine_type  has_power_curve  has_cp_curve
1       Enercon   E-101/3050             True          True
2       Enercon   E-101/3500             True          True
3       Enercon   E-115/3000             True          True
4       Enercon   E-115/3200             True          True
5       Enercon   E-126/4200             True          True
6       Enercon   E-141/4200             True          True
7       Enercon     E-53/800             True          True
8       Enercon    E-70/2000             True          True
9       Enercon    E-70/2300             True          True
10      Enercon    E-82/2000             True          True
11      Enercon    E-82/2300             True          True
12      Enercon    E-82/2350             True          True
13      Enercon    E-82/3000             True          True
14      Enercon    E-92/2350             True          True
15      Enercon   E/126/7500             True         False
16      Enercon      E48/800             True          True
```

[32]:
```
# find all Enercon 101 turbines
print(df[df["turbine_type"].str.contains("E-101")])
```

```
  manufacturer turbine_type  has_power_curve  has_cp_curve
1      Enercon    E-101/3050             True          True
2      Enercon    E-101/3500             True          True
```

[6]:
```
# specification of wind turbine where power curve is provided in the
# oedb turbine library

enercon_e126 = {
        'turbine_type': 'E-126/4200',  # turbine type as in oedb turbine library
        'hub_height': 135  # in m
    }
# initialize WindTurbine object
e126 = WindTurbine(**enercon_e126)
```

```
/home/birgit/virtualenvs/windpowerlib/git_repos/windpowerlib/windpowerlib/
↪wind_turbine.py:322: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/
↪indexing.html#indexing-view-versus-copy
  wpp_df.dropna(axis=1, inplace=True)
```

[35]:
```
# specification of own wind turbine (Note: power curve values and
# nominal power have to be in Watt)
my_turbine = {
    'nominal_power': 3e6,  # in W
    'hub_height': 105,  # in m
    'power_curve': pd.DataFrame(
            data={'value': [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
    }
# initialize WindTurbine object
my_turbine = WindTurbine(**my_turbine)
```

[8]:
```
# specification of wind turbine where power coefficient curve and nominal
# power is provided in an own csv file
csv_path = 'data'
dummy_turbine = {
    'turbine_type': 'DUMMY 1',  # turbine type as in file
    'hub_height': 100,  # in m
    'rotor_diameter': 70,  # in m
    'path': csv_path
}
# initialize WindTurbine object
dummy_turbine = WindTurbine(**dummy_turbine)
```

2.1.4 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. When calling 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'. Of course, you can also use the default methods while only changing one or two of them, as done for 'dummy_turbine'.

[9]:
```
# power output calculation for my_turbine

# initialize 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 time series 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.
```

[10]:
```
# 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

# initialize 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 time series 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.
```

[11]:
```
# power output calculation for example_turbine
# own specification for 'power_output_model'
mc_example_turbine = ModelChain(
    dummy_turbine,
    power_output_model='power_coefficient_curve').run_model(weather)
dummy_turbine.power_output = mc_example_turbine.power_output
```

```
DEBUG:root:Calculating wind speed using logarithmic wind profile.
DEBUG:root:Calculating temperature using temperature gradient.
DEBUG:root:Calculating density using barometric height equation.
DEBUG:root:Calculating power output using power coefficient curve.
```

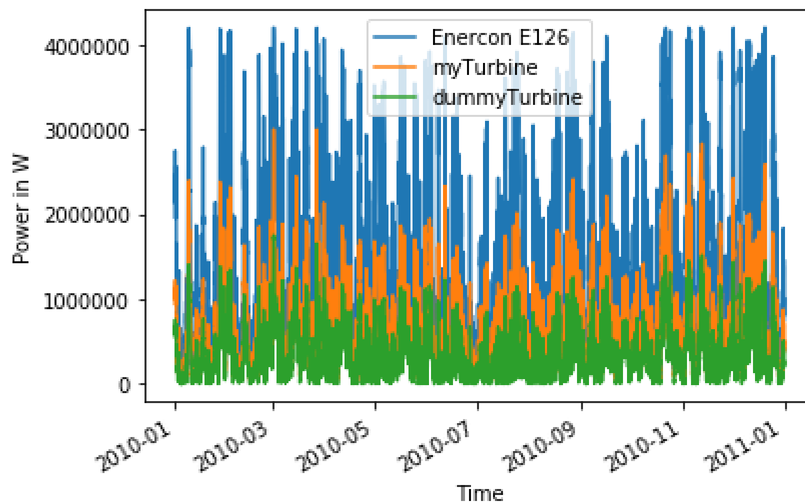
2.1.5 Plot results

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

[12]:
```
# try to import matplotlib
logging.getLogger().setLevel(logging.WARNING)
try:
    from matplotlib import pyplot as plt
    # matplotlib inline needed in notebook to plot inline
    %matplotlib inline
except ImportError:
    plt = None
```

[13]:
```
# plot turbine power output
if plt:
    e126.power_output.plot(legend=True, label='Enercon E126')
    my_turbine.power_output.plot(legend=True, label='myTurbine')
    dummy_turbine.power_output.plot(legend=True, label='dummyTurbine')
    plt.xlabel('Time')
    plt.ylabel('Power in W')
    plt.show()
```

```
DEBUG:matplotlib.axes._base:update_title_pos
DEBUG:matplotlib.font_manager:findfont: Matching␣
↪:family=sans-serif:style=normal:variant=normal:weight=normal:stretch=normal:size=10.
↪0 to DejaVu Sans ('/home/sabine/virtualenvs/windpowerlib/lib/python3.6/
↪site-packages/matplotlib/mpl-data/fonts/ttf/DejaVuSans.ttf') with score of 0.050000.
DEBUG:matplotlib.axes._base:update_title_pos
DEBUG:matplotlib.axes._base:update_title_pos
DEBUG:matplotlib.axes._base:update_title_pos
```



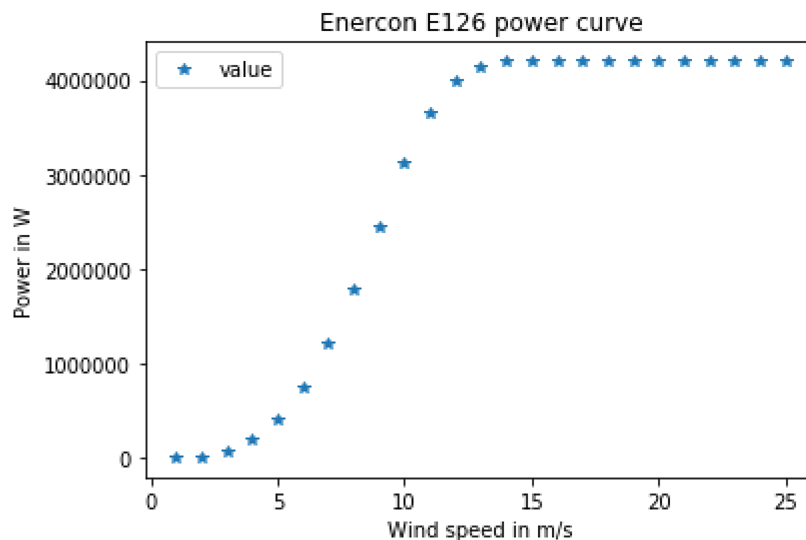
[14]:
```
# plot power (coefficient) curves
if plt:
    if e126.power_coefficient_curve is not None:
        e126.power_coefficient_curve.plot(
            x='wind_speed', y='value', style='*',
            title='Enercon E126 power coefficient curve')
        plt.xlabel('Wind speed in m/s')
        plt.ylabel('Power in W')
        plt.show()
    if e126.power_curve is not None:
        e126.power_curve.plot(x='wind_speed', y='value', style='*',
                              title='Enercon E126 power curve')
```

(continues on next page)

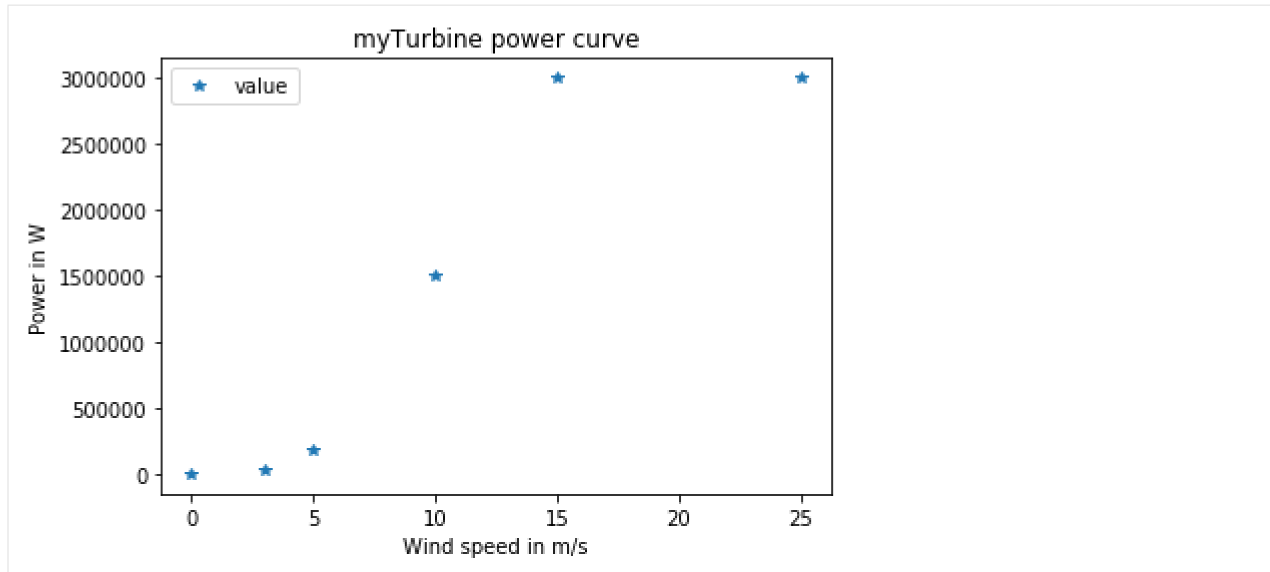
(continued from previous page)

```
        plt.xlabel('Wind speed in m/s')
        plt.ylabel('Power in W')
        plt.show()
    if my_turbine.power_coefficient_curve is not None:
        my_turbine.power_coefficient_curve.plot(
            x='wind_speed', y='value', style='*',
            title='myTurbine power coefficient curve')
        plt.xlabel('Wind speed in m/s')
        plt.ylabel('Power in W')
        plt.show()
    if my_turbine.power_curve is not None:
        my_turbine.power_curve.plot(x='wind_speed', y='value', style='*',
                                    title='myTurbine power curve')
        plt.xlabel('Wind speed in m/s')
        plt.ylabel('Power in W')
        plt.show()
```

```
DEBUG:matplotlib.axes._base:update_title_pos
DEBUG:matplotlib.font_manager:findfont: Matching␣
↪:family=sans-serif:style=normal:variant=normal:weight=normal:stretch=normal:size=12.
↪0 to DejaVu Sans ('/home/sabine/virtualenvs/windpowerlib/lib/python3.6/
↪site-packages/matplotlib/mpl-data/fonts/ttf/DejaVuSans.ttf') with score of 0.050000.
DEBUG:matplotlib.axes._base:update_title_pos
DEBUG:matplotlib.axes._base:update_title_pos
DEBUG:matplotlib.axes._base:update_title_pos
```




```
DEBUG:matplotlib.axes._base:update_title_pos
DEBUG:matplotlib.axes._base:update_title_pos
DEBUG:matplotlib.axes._base:update_title_pos
DEBUG:matplotlib.axes._base:update_title_pos
```



[ ]:

```

```

2.2 TurbineClusterModelChain example

This example shows you how to calculate the power output of wind farms and wind turbine clusters using the windpowerlib. A cluster can be useful if you want to calculate the feed-in of a region for which you want to use one single weather data point.

Functions that are used in the ModelChain example, like the initialization of wind turbines, are imported and used without further explanation.

2.2.1 Imports and initialization of wind turbines

The import of weather data and the initialization of wind turbines is done as in the `modelchain_example`. Be aware that currently for wind farm and wind cluster calculations wind turbines need to have a power curve as some calculations do not work with the power coefficient curve.

[1]:

```
__copyright__ = "Copyright oemof developer group"
__license__ = "GPLv3"

import pandas as pd

import modelchain_example as mc_e
from windpowerlib import TurbineClusterModelChain, WindTurbineCluster, WindFarm

import logging
logging.getLogger().setLevel(logging.DEBUG)
```

[2]:

```
# Get weather data
weather = mc_e.get_weather_data('weather.csv')
print(weather[['wind_speed', 'temperature', 'pressure']][0:3])

# Initialize wind turbines
```

(continues on next page)

(continued from previous page)

```
my_turbine, e126, dummy_turbine = mc_e.initialize_wind_turbines()
print()
print('nominal power of my_turbine: {}'.format(my_turbine.nominal_power))
```

```
                          wind_speed          temperature          pressure
                                  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

nominal power of my_turbine: 3000000.0
```

```
/home/sabine/virtualenvs/windpowerlib/windpowerlib/windpowerlib/wind_turbine.py:324:␣
↪SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/
↪indexing.html#indexing-view-versus-copy
  wpp_df.dropna(axis=1, inplace=True)
```

2.2.2 Initialize wind farm

To initialize a specific wind farm you need to provide a wind turbine fleet specifying the wind turbines and their number or total installed capacity (in Watt) in the farm. Optionally, you can specify a wind farm efficiency and a name as an identifier.

```
[3]: # specification of wind farm data where turbine fleet is provided in a
# pandas.DataFrame
# for each turbine type you can either specify the number of turbines of
# that type in the wind farm (float values are possible as well) or the
# total installed capacity of that turbine type in W
wind_turbine_fleet = pd.DataFrame(
        {'wind_turbine': [my_turbine, e126],  # as windpowerlib.WindTurbine
         'number_of_turbines': [6, None],
         'total_capacity': [None, 12.6e6]}
    )
# initialize WindFarm object
example_farm = WindFarm(name='example_farm',
                        wind_turbine_fleet=wind_turbine_fleet)
```

Following, a wind farm with a constant efficiency is defined. A wind farm efficiency can also be dependent on the wind speed in which case it needs to be provided as a dataframe with 'wind_speed' and 'efficiency' columns containing wind speeds in m/s and the corresponding dimensionless wind farm efficiency.

```
[4]: # specification of wind farm data (2) containing a wind farm efficiency
# wind turbine fleet is provided using the to_group function
example_farm_2_data = {
    'name': 'example_farm_2',
    'wind_turbine_fleet': [my_turbine.to_group(6),
                           e126.to_group(total_capacity=12.6e6)],
    'efficiency': 0.9}

# initialize WindFarm object
example_farm_2 = WindFarm(**example_farm_2_data)
```

(continues on next page)

(continued from previous page)

```
print('nominal power of first turbine type of example_farm_2: {}'.format(
    example_farm_2.wind_turbine_fleet.loc[0, 'wind_turbine'].nominal_power))
```

```
nominal power of first turbine type of example_farm_2: 3000000.0
```

2.2.3 Initialize wind turbine cluster

Like for a wind farm for the initialization of a wind turbine cluster you can use a dictionary that contains the basic parameters. A wind turbine cluster is defined by its wind farms.

```
[5]: # specification of cluster data
example_cluster_data = {
    'name': 'example_cluster',
    'wind_farms': [example_farm, example_farm_2]}

# initialize WindTurbineCluster object
example_cluster = WindTurbineCluster(**example_cluster_data)
```

2.2.4 Use the TurbineClusterModelChain to calculate power output

The TurbineClusterModelChain is a class that provides all necessary steps to calculate the power output of a wind farm or wind turbine cluster.

Like the ModelChain (see *basic example*) you can use the TurbineClusterModelChain with default parameters as shown in this example for the wind farm or specify custom parameters as done here for the cluster. If you use the 'run_model' method first the aggregated power curve and the mean hub height of the wind farm/cluster is calculated, then inherited functions of the ModelChain are used to calculate the wind speed and density (if necessary) at hub height. After that, depending on the parameters, wake losses are applied and at last the power output is calculated.

```
[6]: # power output calculation for example_farm
# initialize TurbineClusterModelChain with default parameters and use
# run_model method to calculate power output
mc_example_farm = TurbineClusterModelChain(example_farm).run_model(weather)
# write power output time series to WindFarm object
example_farm.power_output = mc_example_farm.power_output
```

```
DEBUG:root:Wake losses considered by dena_mean wind efficiency curve.
DEBUG:root:Aggregated power curve not smoothed.
DEBUG:root:Calculating wind speed using logarithmic wind profile.
DEBUG:root:Calculating power output using power curve.
```

```
[7]: # set efficiency of example_farm to apply wake losses
example_farm.efficiency = 0.9

# power output calculation for turbine_cluster
# own specifications for TurbineClusterModelChain setup
modelchain_data = {
        'wake_losses_model': 'wind_farm_efficiency',  #
                                    # 'dena_mean' (default), None,
                                    # 'wind_farm_efficiency' or name
                                    #  of another wind efficiency curve
                #  see :py:func:`~.wake_losses.get_wind_efficiency_curve`
```

(continues on next page)

(continued from previous page)

```
        'smoothing': True,  # False (default) or True
        'block_width': 0.5,  # default: 0.5
        'standard_deviation_method': 'Staffell_Pfenninger',  #
                                    # 'turbulence_intensity' (default)
                                    # or 'Staffell_Pfenninger'
        'smoothing_order': 'wind_farm_power_curves',  #
                                # 'wind_farm_power_curves' (default) or
                                # 'turbine_power_curves'
        '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

# initialize TurbineClusterModelChain with own specifications and use
# run_model method to calculate power output
mc_example_cluster = TurbineClusterModelChain(
        example_cluster, **modelchain_data).run_model(weather)
# write power output time series to WindTurbineCluster object
example_cluster.power_output = mc_example_cluster.power_output
```

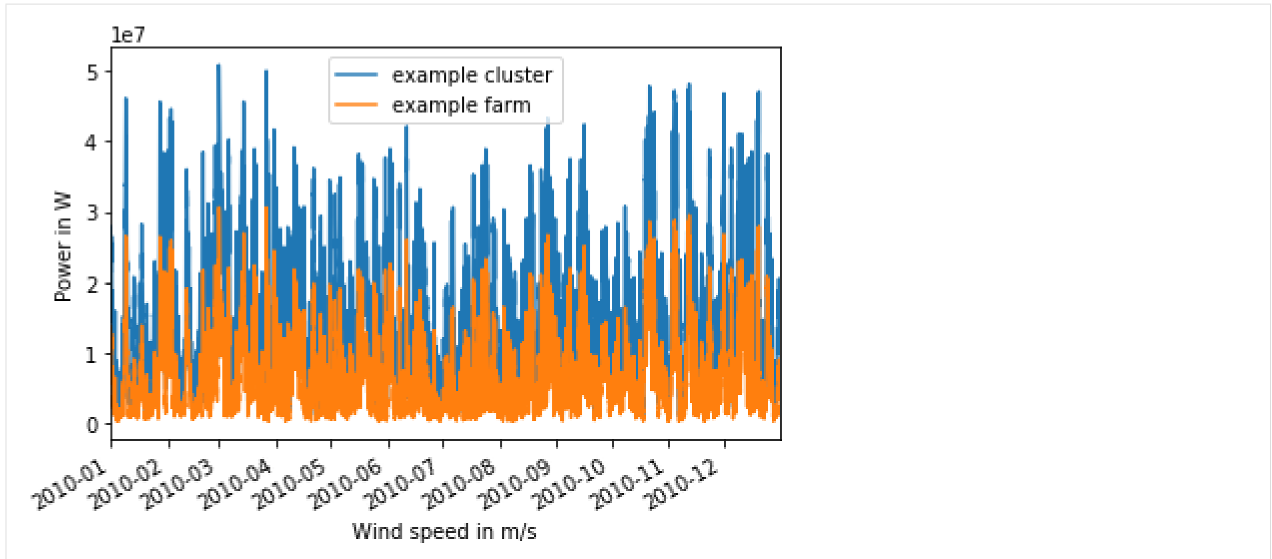
```
DEBUG:root:Wake losses considered with wind_farm_efficiency.
DEBUG:root:Aggregated power curve smoothed by method: Staffell_Pfenninger
DEBUG:root:Calculating wind speed using logarithmic wind profile.
DEBUG:root:Calculating temperature using temperature gradient.
DEBUG:root:Calculating density using ideal gas equation.
DEBUG:root:Calculating power output using power curve.
```

2.2.5 Plot results

If you have matplotlib installed you can visualize the calculated power output.

[8]:
```
# try to import matplotlib
logging.getLogger().setLevel(logging.WARNING)
try:
    from matplotlib import pyplot as plt
    # matplotlib inline needed in notebook to plot inline
    %matplotlib inline
except ImportError:
    plt = None
```

[9]:
```
# plot turbine power output
if plt:
    example_cluster.power_output.plot(legend=True, label='example cluster')
    example_farm.power_output.plot(legend=True, label='example farm')
    plt.xlabel('Wind speed in m/s')
    plt.ylabel('Power in W')
    plt.show()
```


1e7
5
4
3
2
1
0
Power in W
example cluster
example farm
2010-01
2010-02
2010-03
2010-04
2010-05
2010-06
2010-07
2010-08
2010-09
2010-10
2010-11
2010-12
Wind speed in m/s

```
[ ]:
```

CHAPTER 3

Model description

3.1 Wind power plants

The windpowerlib provides three classes for modelling wind power as wind turbines (`WindTurbine`), wind farms (`WindFarm`) and wind turbine clusters (`WindTurbineCluster`).

Descriptions can also be found in the sections *Wind turbine data*, *Wind farm calculations* and *Wind turbine cluster calculations*.

3.2 Height correction and conversion of weather data

Weather data is usually available for a restricted amount of heights above ground. However, for wind feed-in time series calculations weather data is needed at hub height of the examined wind turbines. Thus, the windpowerlib provides functions for the height correction of weather data.

Functions for the height correction of wind speed to the hub height of a wind turbine are described in the *Wind speed* module. Respectively a function for the height correction of temperature data is provided in the *Temperature* module. Functions for density calculations can be found in the *Density* module.

If weather data is available for at least two different heights the respective figure at hub height can be determined by using linear or logarithmic inter-/extrapolation functions of the *Tools* module.

3.3 Power output calculations

Wind feed-in time series can be calculated via power curves and power coefficient curves in the windpowerlib. Functions for power output calculations are described in the *Power output* module.

3.4 Wake losses

The windpowerlib provides two options for the consideration of wake losses in wind farms: reduction of wind speeds and wind farm efficiency (reduction of power in power curves).

For the first option wind efficiency curves are provided which determine the average reduction of wind speeds within a wind farm induced by wake losses depending on the wind speed. These curves were taken from the dena-Netzstudie II and the dissertation of Kaspar Knorr (for references see `get_wind_efficiency_curve()`). The following graph shows all provided wind efficiency curves. The mean wind efficiency curves were calculated in the dena-Netzstudie II and by Kaspar Knorr by averaging wind efficiency curves of 12 wind farms distributed over Germany (dena) or respectively of over 2000 wind farms in Germany (Knorr). Curves with the appendix 'extreme' are wind efficiency curves of single wind farms that are extremely deviating from the respective mean wind efficiency curve.

The second option to consider wake losses is to apply them to power curves by reducing the power values by a constant or on a wind speed depending wind farm efficiency (see `wake_losses_to_power_curve()`). Applying the wind farm efficiency (curve) to power curves instead of feed-in time series has the advantage that the power curves can further be aggregated to obtain turbine cluster power curves (see `WindTurbineCluster`).

3.5 Smoothing of power curves

To account for the spatial distribution of wind speeds within an area the windpowerlib provides a function for power curve smoothing and uses the approach of Nørgaard and Holttinen (for references see `smooth_power_curve()`).

3.6 The modelchains

The modelchains are implemented to ensure an easy start into the Windpowerlib. They work like models that combine all functions provided in the library. Via parameteres desired functions of the windpowerlib can be selected. For parameters not being specified default parameters are used. The *ModelChain* is a model to determine the output of a wind turbine while the *TurbineClusterModelChain* is a model to determine the output of a wind farm or wind turbine cluster. The usage of both modelchains is shown in the *Examples* section.

CHAPTER 4

What's New

These are new features and improvements of note in each release

Releases

- *v0.2.0 (September 9, 2019)*
- *v0.1.3 (September 2, 2019)*
- *v0.1.2 (June 24, 2019)*
- *v0.1.1 (January 31, 2019)*
- *v0.1.0 (January 17, 2019)*
- *v0.0.6 (July 07, 2017)*
- *v0.0.5 (April 10, 2017)*

4.1 v0.2.0 (September 9, 2019)

4.1.1 API changes

- The `WindTurbine` API has been revised. Main changes are that the parameters *fetch_curve* and *data_source* have been removed. These parameters were formerly used to specify whether the power or power coefficient curve should be retrieved and the source to retrieve them from. Now per default the power curve and/or power coefficient curve are tried to be retrieved from the oedb turbine library that is provided along with the windpowerlib and holds turbine data for a large set of wind turbines. Further important changes are the renaming of the parameter *name* to *turbine_type* and the removal of the *coordinates*. See the `WindTurbine` docstring and *Initialize wind turbine* in the *Examples* section for more information. (PR 62)
- The `WindFarm` API has been revised. The *wind_turbine_fleet* parameter can now be provided as a pandas DataFrame (PR 63) or as a list using the `to_group()` method (PR 68. Furthermore, the option to specify the

wind turbine fleet using the total installed capacity of each turbine type has been added. See the `WindFarm` docstring and *Initialize wind farm* in the *Examples* section for more information.

- get_installed_power() methods in `WindFarm` and `WindTurbineCluster` were removed. Installed power is instead now directly calculated inside the nominal_power getter.
- Removed unnecessary *wake_losses_model* parameter in `wake_losses_to_power_curve()`. Whether a constant wind farm efficiency or a wind farm efficiency curve is used is decided by the type of the wind farm efficiency.
- Combined options 'constant_efficiency' and 'power_efficiency_curve' of *wake_losses_model* parameter in `TurbineClusterModelChain()` to 'wind_farm_efficiency'. Therefore, default value of *wake_losses_model* in `assign_power_curve()` and `assign_power_curve()` changed to 'wind_farm_efficiency'.
- Removed *overwrite* parameter from `get_turbine_data_from_oedb()`

4.1.2 Other changes

- Power curves and nominal power of wind turbines are now saved in file in W instead of kW to be consistent with internal units.
- Restructured csv reading for offline usage of windpowerlib. The nominal power of wind turbines is now saved to a separate file along with other turbine data from the oedb turbine library.
- get_turbine_types() can now be used to get provided turbine types of oedb turbine library as well as provided turbine types of local files.

4.1.3 Documentation

- Improved documentation of `ModelChain` and `TurbineClusterModelChain` parameters (PR 64).
- Added info in README and getting started section on how to contribute to the oedb wind turbine library.

4.1.4 Contributors

- Birgit Schachler
- Sabine Haas
- Uwe Krien

4.2 v0.1.3 (September 2, 2019)

4.2.1 Bug fixes

- Acces to renamed oedb table has been fixed (Issue #71).

4.2.2 Contributors

- Birgit Schachler
- Uwe Krien

4.3 v0.1.2 (June 24, 2019)

4.3.1 New features

- new attribute *nominal_power* in `WindFarm` and `WindTurbineCluster` classes (PR #53)
- use properties and setters for *nominal_power* and *installed_power* in `WindFarm` and `WindTurbineCluster` classes
- made windpowerlib work offline: added csv files containing turbine data from OpenEnergy Database (oedb) to the repository for offline usage (PR #52)

4.3.2 Bug fixes

- fixed issue with pandas Multiindex labels and codes attributes (PR #51)

4.3.3 Other changes

- made `get_turbine_types()` also accessible via *get_turbine_types()* –> from windpowerlib import get_turbine_types
- added kwargs in init of `WindTurbine`, `WindFarm` and `WindTurbineCluster`
- we are working with deprecation warnings to draw our user's attention to important changes (PR #53)

4.3.4 Deprecations

wind_farm and wind_turbine_cluster:

- Parameter coordinates is deprecated. In the future the parameter can only be set after instantiation of WindFarm object.
- installed_power is deprecated, use nominal_power instead

wind_turbine:

- Parameters name, data_source and fetch_curve are deprecated. The parameter name will be renamed to turbine_type. Data source and fetching will be defined by the parameters power_coefficient_curve, power_curve and nominal_power in the future.
- Parameter coordinates is deprecated. In the future the parameter can only be set after instantiation of WindTurbine object.

power_curves:

- wake_losses_model is deprecated, will be defined by the type of wind_farm_efficiency

4.3.5 Contributors

- Sabine Haas
- Birgit Schachler
- Uwe Krien

4.4 v0.1.1 (January 31, 2019)

4.4.1 Other changes

- Adapted csv reading to pandas API change
- Removed logging message for successful database connection
- Added system exit and error message in case a non-existing power (coefficient) curve of an existing turbine type is tried to be used

4.4.2 Contributors

- Uwe Krien
- Sabine Haas

4.5 v0.1.0 (January 17, 2019)

ATTENTION: From v0.1.0 on power (coefficient) curves are provided by the OpenEnergy Database (oedb) instead of in csv files (v0.6.0 and lower) due to legal reasons. Use `get_turbine_types()` to check whether the turbine types you need are included in the database. If your turbine type is not included you can either use your own csv file or open an issue.

4.5.1 New classes

- `WindFarm` class for modelling a wind farm. Defines a standard set of wind farm attributes, for example aggregated power curve and wind farm efficiency to take wake losses into account.
- `WindTurbineCluster` class for modelling a turbine cluster that contains several wind turbines and/or wind farms. This class is useful for gathering all wind turbines in a weather data grid cell. An aggregated power curve can be calculated which considers the wake losses of the wind farms by a set efficiency if desired.
- `TurbineClusterModelChain` class shows the usage of new functions and classes of windpowerlib v.0.1 and is based on the ModelChain class.

4.5.2 New functions

- `smooth_power_curve()` for taking into account the spatial distribution of wind speed
- `wake_losses_to_power_curve()`: application of wake losses to a power curve
- `reduce_wind_speed()`: application of wake losses to a wind speed time series by using wind efficiency curves which are provided in the data directory
- `logarithmic_interpolation_extrapolation()` for wind speed time series
- `gauss_distribution()` needed for power curve smoothing
- `estimate_turbulence_intensity()` by roughness length
- `get_turbine_data_from_oedb()` for retrieving power curves from OpenEnergy Database

4.5.3 Testing

- added continuous integration to automatically test the windpowerlib for different python versions and to check the test coverage

4.5.4 Documentation

- added second example section and jupyter notebook
- added model description

4.5.5 API changes

- renamed attribute *turbine_name* of WindTurbine class to *name* to match with *name* attribute of WindFarm and WindTurbineCluster class
- renamed *basic_example* to *modelchain_example*
- renamed column 'values' of power (coefficient) curves to 'value' to prevent errors using df.value(s)
- renamed *run_basic_example()* to *run_example()* in *modelchain_example*
- renamed parameter *wind_turbine* of *ModelChain* object to *power_plant*
- removed parameter *density_correction* from *power_plant.power_coefficient()*

4.5.6 Other changes

- removed deprecated attributes (.ix)
- added *turbine_cluster_modelchain_example* showing the usage of the TurbineClusterModelChain

4.5.7 Contributors

- Sabine Haas
- Uwe Krien
- Birgit Schachler

4.6 v0.0.6 (July 07, 2017)

4.6.1 Documentation

- added basic usage section and jupyter notebook

4.6.2 Testing

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

4.6.3 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

4.6.4 Contributors

- Sabine Haas
- Birgit Schachler
- Uwe Krien

4.7 v0.0.5 (April 10, 2017)

4.7.1 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

4.7.2 Documentation

- added references

4.7.3 Testing

- tests for all modules were added

4.7.4 Contributors

- Sabine Haas
- Birgit Schachler
- Uwe Krien

CHAPTER 5

API

5.1 Classes

wind_turbine.WindTurbine(hub_height[, ...])	Defines a standard set of wind turbine attributes.
wind_farm.WindFarm(wind_turbine_fleet[, ...])	Defines a standard set of wind farm attributes.
wind_turbine_cluster.WindTurbineCluster(...)	Defines a standard set of wind turbine cluster attributes.
modelchain.ModelChain(power_plant[, ...])	Model to determine the output of a wind turbine
turbine_cluster_modelchain.TurbineClusterModelChain(...)	Model to determine the output of a wind farm or wind turbine cluster.

5.1.1 windpowerlib.wind_turbine.WindTurbine

class windpowerlib.wind_turbine.**WindTurbine**(*hub_height*, *nominal_power=None*, *path='oedb'*, *power_curve=None*, *power_coefficient_curve=None*, *rotor_diameter=None*, *turbine_type=None*, ***kwargs*)

Defines a standard set of wind turbine attributes.

Parameters

- **hub_height** (*float*) – Hub height of the wind turbine in m.
- **power_curve** (pandas.DataFrame or dict (optional)) – If provided directly sets the power curve. DataFrame/dictionary must have 'wind_speed' and 'value' columns/keys with wind speeds in m/s and the corresponding power curve value in W. If not set the value is retrieved from 'power_curve.csv' file in *path*. In that case a *turbine_type* is needed. Default: None.
- **power_coefficient_curve** (pandas.DataFrame or dict (optional)) – If provided directly sets the power coefficient curve. DataFrame/dictionary must have 'wind_speed' and 'value' columns/keys with wind speeds in m/s and the corresponding power coefficient curve

value. If not set the value is retrieved from 'power_coefficient_curve.csv' file in *path*. In that case a *turbine_type* is needed. Default: None.

- **turbine_type** (`str (optional)`) – Name of the wind turbine type. Must be provided if power (coefficient) curve, nominal power or rotor diameter is retrieved from self-provided or oedb turbine library csv files. If turbine_type is None it is not possible to retrieve turbine data from file. Use `get_turbine_types()` to see a table of all wind turbines for which power (coefficient) curve data and other turbine data is provided in the oedb turbine library. Default: None.
- **rotor_diameter** (`float (optional)`) – Diameter of the rotor in m. If not set the value is retrieved from 'turbine_data.csv' file in *path*. In that case a *turbine_type* is needed. The rotor diameter only needs to be set if power output is calculated using the power coefficient curve. Default: None.
- **nominal_power** (`float (optional)`) – The nominal power of the wind turbine in W. If not set the value is retrieved from 'turbine_data.csv' file in *path*. In that case a *turbine_type* is needed. Default: None.
- **path** (`str (optional)`) – Directory where the turbine database files are located. The files need to be named 'power_coefficient_curve.csv', 'power_curve.csv', and 'turbine_data.csv'. By default the oedb turbine library files are used. Set path to *None* to ignore turbine data from files. Default: 'oedb'.

turbine_type
Name of the wind turbine.

Type str

hub_height
Hub height of the wind turbine in m.

Type float

rotor_diameter
Diameter of the rotor in m. Default: None.

Type None or float

power_coefficient_curve
Power coefficient curve of the wind turbine. DataFrame/dictionary containing 'wind_speed' and 'value' columns/keys with wind speeds in m/s and the corresponding power coefficients. Default: None.

Type None, pandas.DataFrame or dictionary

power_curve
Power curve of the wind turbine. DataFrame/dictionary containing 'wind_speed' and 'value' columns/keys with wind speeds in m/s and the corresponding power curve value in W. Default: None.

Type None, pandas.DataFrame or dictionary

nominal_power
The nominal output of the wind turbine in W. Default: None.

Type None or float

Notes

Your wind turbine object needs to have a power coefficient or power curve. By default they are fetched from the oedb turbine library that is provided along with the windpowerlib. In that case *turbine_type* must be specified. You can also set the curves directly or provide your own csv files with power coefficient and power curves.

See *example_power_curves.csv', 'example_power_coefficient_curves.csv* and *example_turbine_data.csv* in example/data for the required format of such csv files.

Examples

```
>>> import os
>>> from windpowerlib import WindTurbine
>>> enerconE126 = {
...     'hub_height': 135,
...     'turbine_type': 'E-126/4200'}
>>> e126 = WindTurbine(**enerconE126)
>>> print(e126.nominal_power)
4200000.0
>>> # Example with own path
>>> path = os.path.join(os.path.dirname(__file__), '../example/data')
>>> example_turbine = {
...     'hub_height': 100,
...     'rotor_diameter': 70,
...     'turbine_type': 'DUMMY 3',
...     'path' : path}
>>> e_t_1 = WindTurbine(**example_turbine)
>>> print(e_t_1.power_curve['value'][7])
18000.0
>>> print(e_t_1.nominal_power)
1500000.0
```

__init__(*hub_height*, *nominal_power=None*, *path='oedb'*, *power_curve=None*, *power_coefficient_curve=None*, *rotor_diameter=None*, *turbine_type=None*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(hub_height[, nominal_power, path, ...])	Initialize self.
to_group([number_turbines, total_capacity])	Creates a *WindTurbineGroup*, a NamedTuple data container with the fields 'number_of_turbines' and 'wind_turbine'.

5.1.2 windpowerlib.wind_farm.WindFarm

class windpowerlib.wind_farm.**WindFarm**(*wind_turbine_fleet*, *efficiency=None*, *name="*, ***kwargs*)
Defines a standard set of wind farm attributes.

Parameters

- **wind_turbine_fleet** (pandas.DataFrame or list(*WindTurbineGroup*)) – The wind turbine fleet specifies the turbine types in the wind farm and their corresponding number or total installed capacity. There are different options to provide the wind turbine fleet (see also examples below):
  - pandas.DataFrame - DataFrame must have columns 'wind_turbine' containing a *WindTurbine* object and either 'number_of_turbines' (number of wind turbines of the same turbine type in the wind farm, can be a float) or 'total_capacity' (installed capacity

of wind turbines of the same turbine type in the wind farm).

  – list(*WindTurbineGroup*) - A *WindTurbineGroup* can be created from a *WindTurbine* using the *to_group()* method.

  – list(dict) - It is still possible to use a list of dictionaries (see example) but we recommend to use one of the other options above.

- **efficiency** (float or pandas.DataFrame or None (optional)) – Efficiency of the wind farm. Provide as either constant (float) or power efficiency curve (pd.DataFrame) containing 'wind_speed' and 'efficiency' columns with wind speeds in m/s and the corresponding dimensionless wind farm efficiency. Default: None.

- **name** (*str (optional)*) – Can be used as an identifier of the wind farm. Default: ''.

wind_turbine_fleet

Wind turbines of wind farm. DataFrame must have 'wind_turbine' (contains a *WindTurbine* object) and 'number_of_turbines' (number of wind turbines of the same turbine type in the wind farm) as columns.

Type pandas.DataFrame

efficiency

Efficiency of the wind farm. Either constant (float) power efficiency curve (pd.DataFrame) containing 'wind_speed' and 'efficiency' columns with wind speeds in m/s and the corresponding dimensionless wind farm efficiency. Default: None.

Type float or pandas.DataFrame or None

name

If set this is used as an identifier of the wind farm.

Type str

hub_height

The calculated mean hub height of the wind farm. See *mean_hub_height()* for more information.

Type float

power_curve

The calculated power curve of the wind farm. See *assign_power_curve()* for more information.

Type pandas.DataFrame or None

Examples

```
>>> from windpowerlib import wind_farm
>>> from windpowerlib import WindTurbine
>>> import pandas as pd
>>> enerconE126 = {
...    'hub_height': 135,
...    'rotor_diameter': 127,
...    'turbine_type': 'E-126/4200'}
>>> e126 = WindTurbine(**enerconE126)
>>> vestasV90 = {
...     'hub_height': 90,
...     'turbine_type': 'V90/2000',
...     'nominal_power': 2e6}
>>> v90 = WindTurbine(**vestasV90)
>>> # turbine fleet as DataFrame
>>> wind_turbine_fleet = pd.DataFrame(
```

(continues on next page)

(continued from previous page)

```
...     {'wind_turbine': [e126, v90],
...      'number_of_turbines': [6, None],
...      'total_capacity': [None, 3 * 2e6]})
>>> example_farm = wind_farm.WindFarm(wind_turbine_fleet, name='my_farm')
>>> print(example_farm.nominal_power)
31200000.0
>>> # turbine fleet as a list of WindTurbineGroup objects using the
>>> # 'to_group' method.
>>> wind_turbine_fleet = [e126.to_group(6),
...                       v90.to_group(total_capacity=3 * 2e6)]
>>> example_farm = wind_farm.WindFarm(wind_turbine_fleet, name='my_farm')
>>> print(example_farm.nominal_power)
31200000.0
>>> # turbine fleet as list of dictionaries (not recommended)
>>> example_farm_data = {
...    'name': 'my_farm',
...    'wind_turbine_fleet': [{'wind_turbine': e126,
...                            'number_of_turbines': 6},
...                           {'wind_turbine': v90,
...                            'total_capacity': 3 * 2e6}]}
>>> example_farm = wind_farm.WindFarm(**example_farm_data)
>>> print(example_farm.nominal_power)
31200000.0
```

__init__(*wind_turbine_fleet*, *efficiency=None*, *name=''*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(wind_turbine_fleet[, efficiency, name])	Initialize self.
assign_power_curve([wake_losses_model, ...])	Calculates the power curve of a wind farm.
check_and_complete_wind_turbine_fleet()	Function to check wind turbine fleet user input.
mean_hub_height()	Calculates the mean hub height of the wind farm.

Attributes

nominal_power	The nominal power is the sum of the nominal power of all turbines.

5.1.3 windpowerlib.wind_turbine_cluster.WindTurbineCluster

class windpowerlib.wind_turbine_cluster.**WindTurbineCluster**(*wind_farms*, *name=''*, ***kwargs*)

Defines a standard set of wind turbine cluster attributes.

Parameters

- **wind_farms** (list(*WindFarm*)) – List of wind farms in cluster.
- **name** (*str (optional)*) – Can be used as an identifier of the wind turbine cluster. Default: ''.

wind_farms
List of wind farms in cluster.

Type list(*WindFarm*)

name
If set this is used as an identifier of the wind turbine cluster.

Type str

hub_height
The calculated average hub height of the wind turbine cluster. See *mean_hub_height()* for more information.

Type float

power_curve
The calculated power curve of the wind turbine cluster. See *assign_power_curve()* for more information.

Type pandas.DataFrame or None

__init__(*wind_farms*, *name=''*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(wind_farms[, name])	Initialize self.
assign_power_curve([wake_losses_model, . . .])	Calculates the power curve of a wind turbine cluster.
mean_hub_height()	Calculates the mean hub height of the wind turbine cluster.

Attributes

nominal_power	The nominal power is the sum of the nominal power of all turbines in the wind turbine cluster.

5.1.4 windpowerlib.modelchain.ModelChain

class windpowerlib.modelchain.**ModelChain**(*power_plant*, *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*, ***kwargs*)

Model to determine the output of a wind turbine

The ModelChain class provides a standardized, high-level interface for all of the modeling steps necessary for calculating wind turbine power output from weather time series inputs.

Parameters

- **power_plant** (*WindTurbine*) – A *WindTurbine* object representing the wind turbine.
- **wind_speed_model** (*str*) – Parameter to define which model to use to calculate the

wind speed at hub height. Valid options are:

- 'logarithmic' - See `logarithmic_profile()` for more information. The parameter *obstacle_height* can be used to set the height of obstacles in the surrounding area of the wind turbine.
- 'hellman' - See `hellman()` for more information.
- 'interpolation_extrapolation' - See `linear_interpolation_extrapolation()` for more information.
- 'log_interpolation_extrapolation' - See `logarithmic_interpolation_extrapolation()` for more information.

Default: 'logarithmic'.

- **temperature_model** (*str*) – Parameter to define which model to use to calculate the temperature of air at hub height. Valid options are:
  - 'linear_gradient' - See `linear_gradient()` for more information.
  - 'interpolation_extrapolation' - See `linear_interpolation_extrapolation()` for more information.

  Default: 'linear_gradient'.
- **density_model** (*str*) – Parameter to define which model to use to calculate the density of air at hub height. Valid options are:
  - 'barometric' - See `barometric()` for more information.
  - 'ideal_gas' - See `ideal_gas()` for more information.
  - 'interpolation_extrapolation' - See `linear_interpolation_extrapolation()` for more information.

  Default: 'barometric'.
- **power_output_model** (*str*) – Parameter to define which model to use to calculate the turbine power output. Valid options are:
  - 'power_curve' - See `power_curve()` for more information. In order to use the density corrected power curve to calculate the power output set parameter *density_correction* to True.
  - 'power_coefficient_curve' - See `power_coefficient_curve()` for more information.

  Default: 'power_curve'.
- **density_correction** (*bool*) – This parameter is only used if the parameter *power_output_model* is 'power_curve'. For more information on this parameter see parameter *density_correction* in `power_curve()`. Default: False.
- **obstacle_height** (*float*) – This parameter is only used if the parameter *wind_speed_model* is 'logarithmic'. For more information on this parameter see parameter *obstacle_height* in `logarithmic()`. Default: 0.
- **hellman_exp** (*float*) – This parameter is only used if the parameter *wind_speed_model* is 'hellman'. For more information on this parameter see parameter *hellman_exponent* in `hellman()`. Default: None.

power_plant
A `WindTurbine` object representing the wind turbine.

Type *WindTurbine*

wind_speed_model
Defines which model is used to calculate the wind speed at hub height.

Type str

temperature_model
Defines which model is used to calculate the temperature of air at hub height.

Type str

density_model
Defines which model is used to calculate the density of air at hub height.

Type str

power_output_model
Defines which model is used to calculate the turbine power output.

Type str

density_correction
Used to set *density_correction* parameter in `power_curve()`.

Type bool

obstacle_height
Used to set *obstacle_height* in `logarithmic()`.

Type float

hellman_exp
Used to set *hellman_exponent* in `hellman()`.

Type float

power_output
Electrical power output of the wind turbine in W.

Type pandas.Series

Examples

```
>>> from windpowerlib import modelchain
>>> from windpowerlib import wind_turbine
>>> enerconE126 = {
...    'hub_height': 135,
...    'rotor_diameter': 127,
...    'turbine_type': 'E-126/4200'}
>>> 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__(*power_plant, 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, **kwargs*)
Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__`(power_plant[, wind_speed_model, …])	Initialize self.
`calculate_power_output`(wind_speed_hub, …)	Calculates the power output of the wind power plant.
`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.
`wind_speed_hub`(weather_df)	Calculates the wind speed at hub height.

5.1.5 windpowerlib.turbine_cluster_modelchain.TurbineClusterModelChain

class windpowerlib.turbine_cluster_modelchain.**TurbineClusterModelChain**(*power_plant, wake_losses_model='dena_m smoothing=False, block_width=0.5, standard_deviation_method='tur smoothing_order='wind_farm_powe **kwargs*)

Model to determine the output of a wind farm or wind turbine cluster.

Parameters

- **power_plant** (*WindFarm* or *WindTurbineCluster*) – A *WindFarm* object representing the wind farm or a *WindTurbineCluster* object representing the wind turbine cluster.
- **wake_losses_model** (*str or None*) –

  Defines the method for taking wake losses within the farm into consideration.

  – None - Wake losses are not taken into account.

  – 'wind_farm_efficiency' - The values of the wind farm power curve(s) are reduced by the wind farm efficiency, which needs to be set in the *WindFarm* class. Note: The wind farm efficiency has no effect if *wake_losses_model* is not set to 'wind_farm_efficiency'. See `wake_losses_to_power_curve()` for more information.

  – 'dena_mean' or name of other wind efficiency curve - The values of the wind speed time series are reduced by the chosen wind efficiency curve in `run_model()` before the power output calculations. See `reduce_wind_speed()` for more information. Use `get_wind_efficiency_curve()` to get a DataFrame of all provided wind efficiency curves and see the provided example on how to plot the wind efficiency curves.

  Default: 'dena_mean'.
- **smoothing** (*bool*) – If True the power curves will be smoothed to account for the distribution of wind speeds over space. Depending on the parameter *smoothing_order* the power curves are smoothed before or after aggregating wind turbine power curves to one representative power curve of the wind farm or cluster. See `smooth_power_curve()` for more information.

Default: False.

- **block_width** (*float*) – Width between the wind speeds in the sum of the equation in *smooth_power_curve()*. This parameter is only used if *smoothing* is True. To achieve a smooth curve without steps a value not much higher than the step width between the power curve wind speeds should be chosen.

  Default: 0.5.

- **standard_deviation_method** (*str*) – Method for calculating the standard deviation for the Gauss distribution if *smoothing* is True.

  – 'turbulence_intensity' - See *smooth_power_curve()* for more information.

  – 'Staffell_Pfenninger' - See *smooth_power_curve()* for more information.

  Default: 'turbulence_intensity'.

- **smoothing_order** (*str*) – Defines when the smoothing takes place if *smoothing* is True.

  – 'turbine_power_curves' - Smoothing is applied to wind turbine power curves.

  – 'wind_farm_power_curves' - Smoothing is applied to wind farm power curves.

  Default: 'wind_farm_power_curves'.

Other Parameters

- **wind_speed_model** – See *ModelChain* for more information.
- **temperature_model** – See *ModelChain* for more information.
- **density_model** – See *ModelChain* for more information.
- **power_output_model** – See *ModelChain* for more information.
- **density_correction** – See *ModelChain* for more information.
- **obstacle_height** – See *ModelChain* for more information.
- **hellman_exp** – See *ModelChain* for more information.

power_plant

A *WindFarm* object representing the wind farm or a *WindTurbineCluster* object representing the wind turbine cluster.

Type *WindFarm* or *WindTurbineCluster*

wake_losses_model

Defines the method for taking wake losses within the farm into consideration.

Type str or None

smoothing

If True the power curves are smoothed.

Type bool

block_width

Width between the wind speeds in the sum of the equation in *smooth_power_curve()*.

Type float

standard_deviation_method

Method for calculating the standard deviation for the Gauss distribution.

Type str

smoothing_order
Defines when the smoothing takes place if *smoothing* is True.

Type str

power_output
Electrical power output of the wind turbine in W.

Type pandas.Series

power_curve
The calculated power curve of the wind farm.

Type pandas.Dataframe or None

wind_speed_model
Defines which model is used to calculate the wind speed at hub height.

Type str

temperature_model
Defines which model is used to calculate the temperature of air at hub height.

Type str

density_model
Defines which model is used to calculate the density of air at hub height.

Type str

power_output_model
Defines which model is used to calculate the turbine power output.

Type str

density_correction
Used to set *density_correction* parameter in `power_curve()`.

Type bool

obstacle_height
Used to set *obstacle_height* in `logarithmic()`.

Type float

hellman_exp
Used to set *hellman_exponent* in `hellman()`.

Type float

__init__(*power_plant, wake_losses_model='dena_mean', smoothing=False, block_width=0.5, standard_deviation_method='turbulence_intensity', smoothing_order='wind_farm_power_curves', **kwargs*)
Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__`(power_plant[, wake_losses_model, ...])	Initialize self.
`assign_power_curve`(weather_df)	Calculates the power curve of the wind turbine cluster.

Continued on next page

Table 8 – continued from previous page

calculate_power_output(wind_speed_hub, ...)	Calculates the power output of the wind power plant.
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.
wind_speed_hub(weather_df)	Calculates the wind speed at hub height.

5.2 Temperature

Function for calculating air temperature at hub height.

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

5.2.1 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)

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

References

5.3 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.

5.3.1 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.

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

[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

References

5.3.2 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.

Return type pandas.Series or numpy.array

Notes

The following equations are used[1][2][3]:

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

and[4]:

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

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

5.4 Wind speed

Functions for calculating wind speed at hub height.

[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] Knorr, K.: "Modellierung von raum-zeitlichen Eigenschaften der Windenergieeinspeisung für wetterdatenbasierte Windleistungssimulationen". Universität Kassel, Diss., 2016, p. 97

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

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.

5.4.1 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.
- **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[1,2,3]:

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

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.

[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

References

5.4.2 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.
- **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.

[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

References

5.5 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.get_turbine_data_from_file(...)	Fetches turbine data from a csv file.
wind_turbine.load_turbine_data_from_oedb([...])	Loads turbine library from the OpenEnergy database (oedb).
wind_turbine.get_turbine_types([...])	Get all provided wind turbine types provided.

5.5.1 windpowerlib.wind_turbine.get_turbine_data_from_file

windpowerlib.wind_turbine.**get_turbine_data_from_file**(*turbine_type*, *path*)

Fetches turbine data from a csv file.

See *example_power_curves.csv', 'example_power_coefficient_curves.csv* and *example_turbine_data.csv* in example/data for the required format of a csv file. Make sure to provide wind speeds in m/s and power in W or convert units after loading the data.

Parameters

- **turbine_type** (*str*) – Specifies the turbine type data is fetched for.
- **path** (*str*) – Specifies the source of the turbine data. See the example below for how to use the example data.

Returns Power curve or power coefficient curve (pandas.DataFrame) or nominal power (float) of one wind turbine type. Power (coefficient) curve DataFrame contains power coefficient curve values (dimensionless) or power curve values (in dimension given in file) with the corresponding wind speeds (in dimension given in file).

Return type pandas.DataFrame or float

Examples

```
>>> from windpowerlib import wind_turbine
>>> import os
>>> path = os.path.join(os.path.dirname(__file__), '../example/data',
...     'power_curves.csv')
>>> d3 = get_turbine_data_from_file('DUMMY 3', path)
>>> print(d3['value'][7])
18000.0
>>> print(d3['value'].max())
1500000.0
```

5.5.2 windpowerlib.wind_turbine.load_turbine_data_from_oedb

windpowerlib.wind_turbine.**load_turbine_data_from_oedb**(*schema='supply'*, *table='wind_turbine_library'*)

Loads turbine library from the OpenEnergy database (oedb).

Turbine data is saved to csv files ('oedb_power_curves.csv', 'oedb_power_coefficient_curves.csv' and 'oedb_nominal_power') for offline usage of the windpowerlib. If the files already exist they are overwritten.

Parameters

- **schema** (*str*) – Database schema of the turbine library.
- **table** (*str*) – Table name of the turbine library.

Returns Turbine data of different turbines such as 'manufacturer', 'turbine_type', 'nominal_power'.

Return type pandas.DataFrame

5.5.3 windpowerlib.wind_turbine.get_turbine_types

windpowerlib.wind_turbine.**get_turbine_types**(*turbine_library='local'*, *print_out=True*, *filter_=True*)

Get all provided wind turbine types provided.

Choose by *turbine_library* whether to get wind turbine types provided by the OpenEnergy Database ('oedb') or wind turbine types provided in your local file(s) ('local'). By default only turbine types for which a power coefficient curve or power curve is provided are returned. Set *filter_=False* to see all turbine types for which any data (e.g. hub height, rotor diameter, ...) is provided.

Parameters

- **turbine_library** (*str*) – Specifies if the oedb turbine library ('oedb') or your local turbine data file ('local') is evaluated. Default: 'local'.
- **print_out** (*bool*) – Directly prints a tabular containing the turbine types in column 'turbine_type', the manufacturer in column 'manufacturer' and information about whether a power (coefficient) curve exists (True) or not (False) in columns 'has_power_curve' and 'has_cp_curve'. Default: True.
- **filter** (*bool*) – If True only turbine types for which a power coefficient curve or power curve is provided in the oedb turbine library are returned. Default: True.

Returns Contains turbine types in column 'turbine_type', the manufacturer in column 'manufacturer' and information about whether a power (coefficient) curve exists (True) or not (False) in columns 'has_power_curve' and 'has_cp_curve'.

Return type pandas.DataFrame

Notes

If the power (coefficient) curve of the desired turbine type (or the turbine type itself) is missing you can contact us via github or windpowerlib@rl-institut.de. You can help us by providing data in the format as shown in the data base.

Examples

```
>>> from windpowerlib import wind_turbine
>>> df = wind_turbine.get_turbine_types(print_out=False)
>>> print(df[df["turbine_type"].str.contains("E-126")].iloc[0])
manufacturer          Enercon
turbine_type       E-126/4200
has_power_curve          True
```

(continues on next page)

(continued from previous page)

```
has_cp_curve                True
Name: 5, dtype: object
>>> print(df[df["manufacturer"].str.contains("Enercon")].iloc[0])
manufacturer             Enercon
turbine_type          E-101/3050
has_power_curve             True
has_cp_curve                True
Name: 1, dtype: object
```

5.6 Data Container

Create data container to be used as an input in classes und functions.

wind_turbine.WindTurbineGroup	A simple data container to define more than one turbine of the same type.
wind_turbine.WindTurbine.to_group([…])	Creates a *WindTurbineGroup*, a NamedTuple data container with the fields 'number_of_turbines' and 'wind_turbine'.

5.6.1 windpowerlib.wind_turbine.WindTurbineGroup

class windpowerlib.wind_turbine.**WindTurbineGroup**

A simple data container to define more than one turbine of the same type. Use the *to_group()* method to easily create a WindTurbineGroup from a *WindTurbine* object.

Parameters

- **'wind_turbine'** (WindTurbine) – A WindTurbine object with all necessary attributes.
- **'number_of_turbines'** (*float*) – The number of turbines. The number is not restricted to integer values.

Create new instance of WindTurbineGroup(wind_turbine, number_of_turbines)

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

count	Return number of occurrences of value.
index	Return first index of value.

Attributes

number_of_turbines	Number of turbines of type WindTurbine
wind_turbine	A WindTurbine object.

5.6.2 windpowerlib.wind_turbine.WindTurbine.to_group

WindTurbine.**to_group**(*number_turbines=None*, *total_capacity=None*)

Creates a *WindTurbineGroup*, a NamedTuple data container with the fields 'number_of_turbines' and 'wind_turbine'. If no parameter is passed the number of turbines is set to one.

It can be used to calculate the number of turbines for a given total capacity or to create a namedtuple that can be used to define a *WindFarm* object.

Parameters

- **number_turbines** (*float*) – Number of turbines of the defined type. Default: 1
- **total_capacity** (*float*) – Total capacity of the group of wind turbines of the same type.

Returns A namedtuple with two fields: 'number_of_turbines' and 'wind_turbine'.

Return type *WindTurbineGroup*

Examples

```
>>> from windpowerlib import WindTurbine
>>> enerconE126 = {
...    'hub_height': 135,
...    'turbine_type': 'E-126/4200'}
>>> e126 = WindTurbine(**enerconE126)
>>> e126.to_group(5).number_of_turbines
5
>>> e126.to_group().number_of_turbines
1
>>> e126.to_group(number_turbines=7).number_of_turbines
7
>>> e126.to_group(total_capacity=12600000).number_of_turbines
3.0
>>> e126.to_group(total_capacity=14700000).number_of_turbines
3.5
>>> e126.to_group(total_capacity=12600000).wind_turbine.nominal_power
4200000.0
>>> type(e126.to_group(5))
<class 'windpowerlib.wind_turbine.WindTurbineGroup'>
>>> e126.to_group(5)  # doctest: +NORMALIZE_WHITESPACE
WindTurbineGroup(wind_turbine=Wind turbine: E-126/4200 ['nominal
power=4200000.0 W', 'hub height=135 m', 'rotor diameter=127.0 m',
'power_coefficient_curve=True', 'power_curve=True'],
number_of_turbines=5)
```

5.7 Wind farm calculations

Functions and methods to calculate the mean hub height, installed power as well as the aggregated power curve of a *WindFarm* object.

wind_farm.WindFarm.check_and_complete_wind_turbine_fleet()	Function to check wind turbine fleet user input.

Continued on next page

Table 16 – continued from previous page

wind_farm.WindFarm.nominal_power	The nominal power is the sum of the nominal power of all turbines.
wind_farm.WindFarm.mean_hub_height()	Calculates the mean hub height of the wind farm.
wind_farm.WindFarm.assign_power_curve([…])	Calculates the power curve of a wind farm.

5.7.1 windpowerlib.wind_farm.WindFarm.check_and_complete_wind_turbine_fleet

WindFarm.**check_and_complete_wind_turbine_fleet**()

Function to check wind turbine fleet user input.

Besides checking if all necessary parameters to fully define the wind turbine fleet are provided, this function also fills in the number of turbines or total capacity of each turbine type and checks if they are consistent.

5.7.2 windpowerlib.wind_farm.WindFarm.nominal_power

WindFarm.**nominal_power**

The nominal power is the sum of the nominal power of all turbines.

Returns Nominal power of the wind farm in W.

Return type float

5.7.3 windpowerlib.wind_farm.WindFarm.mean_hub_height

WindFarm.**mean_hub_height**()

Calculates the mean hub height of the wind farm.

The mean hub height of a wind farm is necessary for power output calculations with an aggregated wind farm power curve containing wind turbines with different hub heights. Hub heights of wind turbines with higher nominal power weigh more than others. After the calculations the mean hub height is assigned to the attribute *hub_height*.

Returns self

Return type *WindFarm*

Notes

The following equation is used[1]:

$$h_{WF} = e^{\sum\limits_{k} ln(h_{WT,k}) \frac{P_{N,k}}{\sum\limits_{k} P_{N,k}}}$$

with: h_{WF}: mean hub height of wind farm, $h_{WT,k}$: hub height of the k-th wind turbine of a wind farm, $P_{N,k}$: nominal power of the k-th wind turbine

[1] Knorr, K.: "Modellierung von raum-zeitlichen Eigenschaften der Windenergieeinspeisung für wetterdatenbasierte Windleistungssimulationen". Universität Kassel, Diss., 2016, p. 35

References

5.7.4 windpowerlib.wind_farm.WindFarm.assign_power_curve

WindFarm.**assign_power_curve**(*wake_losses_model='wind_farm_efficiency'*, *smoothing=False*, *block_width=0.5*, *standard_deviation_method='turbulence_intensity'*, *smoothing_order='wind_farm_power_curves'*, *turbulence_intensity=None*, ***kwargs*)

Calculates the power curve of a wind farm.

The wind farm power curve is calculated by aggregating the power curves of all wind turbines in the wind farm. Depending on the parameters the power curves are smoothed (before or after the aggregation) and/or a wind farm efficiency (power efficiency curve or constant efficiency) is applied after the aggregation. After the calculations the power curve is assigned to the attribute `power_curve`.

Parameters

- **wake_losses_model** (*str*) – Defines the method for taking wake losses within the farm into consideration. Options: 'wind_farm_efficiency' or None. Default: 'wind_farm_efficiency'.
- **smoothing** (*bool*) – If True the power curves will be smoothed before or after the aggregation of power curves depending on *smoothing_order*. Default: False.
- **block_width** (*float*) – Width between the wind speeds in the sum of the equation in `smooth_power_curve()`. Default: 0.5.
- **standard_deviation_method** (*str*) – Method for calculating the standard deviation for the Gauss distribution. Options: 'turbulence_intensity', 'Staffell_Pfenninger'. Default: 'turbulence_intensity'.
- **smoothing_order** (*str*) – Defines when the smoothing takes place if *smoothing* is True. Options: 'turbine_power_curves' (to the single turbine power curves), 'wind_farm_power_curves'. Default: 'wind_farm_power_curves'.
- **turbulence_intensity** (*float*) – Turbulence intensity at hub height of the wind farm for power curve smoothing with 'turbulence_intensity' method. Can be calculated from *roughness_length* instead. Default: None.
- **roughness_length** (*float (optional)*) – Roughness length. If *standard_deviation_method* is 'turbulence_intensity' and *turbulence_intensity* is not given the turbulence intensity is calculated via the roughness length.

Returns self

Return type `WindFarm`

5.8 Wind turbine cluster calculations

Functions and methods to calculate the mean hub height, nominal power as well as the aggregated power curve of a `WindTurbineCluster` object. This is realized in a new module as the functions differ from the functions in the `WindFarm` class.

`wind_turbine_cluster.WindTurbineCluster.nominal_power`	The nominal power is the sum of the nominal power of all turbines in the wind turbine cluster.

Continued on next page

Table 17 – continued from previous page

wind_turbine_cluster.WindTurbineCluster.mean_hub_height()	Calculates the mean hub height of the wind turbine cluster.
wind_turbine_cluster.WindTurbineCluster.assign_power_curve([…])	Calculates the power curve of a wind turbine cluster.

5.8.1 windpowerlib.wind_turbine_cluster.WindTurbineCluster.nominal_power

`WindTurbineCluster.`**`nominal_power`**

The nominal power is the sum of the nominal power of all turbines in the wind turbine cluster.

Returns Nominal power of the wind turbine cluster in W.

Return type float

5.8.2 windpowerlib.wind_turbine_cluster.WindTurbineCluster.mean_hub_height

`WindTurbineCluster.`**`mean_hub_height`**`()`

Calculates the mean hub height of the wind turbine cluster.

The mean hub height of a wind turbine cluster is necessary for power output calculations with an aggregated wind turbine cluster power curve. Hub heights of wind farms with higher nominal power weigh more than others. After the calculations the mean hub height is assigned to the attribute *hub_height*.

Returns self

Return type *WindTurbineCluster*

Notes

The following equation is used[1]:

$$h_{WTC} = e^{\sum\limits_{k} ln(h_{WF,k}) \frac{P_{N,k}}{\sum\limits_{k} P_{N,k}}}$$

with: h_{WTC}: mean hub height of wind turbine cluster, $h_{WF,k}$: hub height of the k-th wind farm of the cluster, $P_{N,k}$: installed power of the k-th wind farm

References

5.8.3 windpowerlib.wind_turbine_cluster.WindTurbineCluster.assign_power_curve

`WindTurbineCluster.`**`assign_power_curve`**`(`*wake_losses_model='wind_farm_efficiency', smoothing=False, block_width=0.5, standard_deviation_method='turbulence_intensity', smoothing_order='wind_farm_power_curves', turbulence_intensity=None, **kwargs*`)`

Calculates the power curve of a wind turbine cluster.

The turbine cluster power curve is calculated by aggregating the wind farm power curves of wind farms within the turbine cluster. Depending on the parameters the power curves are smoothed (before or after the aggregation)

[1] Knorr, K.: "Modellierung von raum-zeitlichen Eigenschaften der Windenergieeinspeisung für wetterdatenbasierte Windleistungssimulationen". Universität Kassel, Diss., 2016, p. 35

and/or a wind farm efficiency is applied before the aggregation. After the calculations the power curve is assigned to the attribute `power_curve`.

Parameters

- **`wake_losses_model`** (*str*) – Defines the method for taking wake losses within the farm into consideration. Options: 'wind_farm_efficiency' or None. Default: 'wind_farm_efficiency'.
- **`smoothing`** (*bool*) – If True the power curves will be smoothed before or after the aggregation of power curves depending on *smoothing_order*. Default: False.
- **`block_width`** (*float*) – Width between the wind speeds in the sum of the equation in `smooth_power_curve()`. Default: 0.5.
- **`standard_deviation_method`** (*str*) – Method for calculating the standard deviation for the Gauss distribution. Options: 'turbulence_intensity', 'Staffell_Pfenninger'. Default: 'turbulence_intensity'.
- **`smoothing_order`** (*str*) – Defines when the smoothing takes place if *smoothing* is True. Options: 'turbine_power_curves' (to the single turbine power curves), 'wind_farm_power_curves'. Default: 'wind_farm_power_curves'.
- **`turbulence_intensity`** (*float*) – Turbulence intensity at hub height of the wind farm or wind turbine cluster for power curve smoothing with 'turbulence_intensity' method. Can be calculated from *roughness_length* instead. Default: None.
- **`roughness_length`** (*float (optional)*) – Roughness length. If *standard_deviation_method* is 'turbulence_intensity' and *turbulence_intensity* is not given the turbulence intensity is calculated via the roughness length.

Returns self

Return type `WindTurbineCluster`

5.9 Power output

Functions for calculating power output of a wind power plant.

`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.

5.9.1 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*)

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'.

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/m³.

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[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/m³], 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

5.9.2 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/m³. This parameter is needed if *density_correction* is True. Default: None.

[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

- **density_correction** (*bool*) – If the parameter is True the density corrected power curve (see *power_curve_density_correction()*) 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.

5.9.3 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 site specific power curve wind speeds[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]

[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

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.

References

5.10 Alteration of power curves

Functions for smoothing power curves or applying wake losses to a power curve.

`power_curves.smooth_power_curve`(...[, ...])	Smoothes a power curve by using a Gauss distribution.
`power_curves.wake_losses_to_power_curve`(...)	Reduces the power values of a power curve by an efficiency (curve).

5.10.1 windpowerlib.power_curves.smooth_power_curve

windpowerlib.power_curves.**smooth_power_curve**(*power_curve_wind_speeds, power_curve_values, block_width=0.5, wind_speed_range=15.0, standard_deviation_method='turbulence_intensity', mean_gauss=0, **kwargs*)

Smoothes a power curve by using a Gauss distribution.

The smoothing serves for taking the distribution of wind speeds over space into account.

Parameters

- **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*.
- **block_width** (*float*) – Width between the wind speeds in the sum of equation (5.1). Default: 0.5.
- **wind_speed_range** (*float*) – The sum in the equation below is taken for this wind speed range below and above the power curve wind speed. Default: 15.0.
- **standard_deviation_method** (*str*) – Method for calculating the standard deviation for the Gauss distribution. Options: 'turbulence_intensity', 'Staffell_Pfenninger'. Default: 'turbulence_intensity'.
- **mean_gauss** (*float*) – Mean of the Gauss distribution in `gauss_distribution()`. Default: 0.

Other Parameters **turbulence intensity** (*float, optional*) – Turbulence intensity at hub height of the wind turbine, wind farm or wind turbine cluster the power curve is smoothed for.

Returns Smoothed power curve. DataFrame has 'wind_speed' and 'value' columns with wind speeds in m/s and the corresponding power curve value in W.

Return type pandas.DataFrame

Notes

The following equation is used to calculated the power curves values of the smoothed power curve[1]:

$$P_{smoothed}(v_{std}) = \sum_{v_i} \Delta v_i \cdot P(v_i) \cdot \frac{1}{\sigma \sqrt{2\pi}} \exp\left[-\frac{(v_{std} - v_i - \mu)^2}{2\sigma^2}\right] \tag{5.1}$$

with: P: power [W], v: wind speed [m/s], σ: standard deviation (Gauss), μ: mean (Gauss)

$P_{smoothed}$ is the smoothed power curve value, v_{std} is the standard wind speed in the power curve, Δv_i is the interval length between v_i and $v_\text{i+1}$

Power curve smoothing is applied to take account of the spatial distribution of wind speed. This way of smoothing power curves is also used in[2] and[3].

The standard deviation σ of the above equation can be calculated by the following methods.

'turbulence_intensity'[2]:

$$\sigma = v_\text{std} \cdot \sigma_\text{n} = v_\text{std} \cdot TI$$

with: TI: turbulence intensity

'Staffell_Pfenninger'[4]:

$$\sigma = 0.6 \cdot 0.2 \cdot v_\text{std}$$

References

5.10.2 windpowerlib.power_curves.wake_losses_to_power_curve

`windpowerlib.power_curves.`**`wake_losses_to_power_curve`**(*power_curve_wind_speeds*, *power_curve_values*, *wind_farm_efficiency*)

Reduces the power values of a power curve by an efficiency (curve).

Parameters

- **`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*.
- **`wind_farm_efficiency`** (float or pandas.DataFrame) – Efficiency of the wind farm. Either constant (float) or efficiency curve (pd.DataFrame) containing 'wind_speed' and 'efficiency' columns with wind speeds in m/s and the corresponding dimensionless wind farm efficiency (reduction of power). Default: None.

Returns Power curve with power values reduced by a wind farm efficiency. DataFrame has 'wind_speed' and 'value' columns with wind speeds in m/s and the corresponding power curve value in W.

Return type pandas.DataFrame

[1] Knorr, K.: "Modellierung von raum-zeitlichen Eigenschaften der Windenergieeinspeisung für wetterdatenbasierte Windleistungssimulationen". Universität Kassel, Diss., 2016, p. 106

[2] Nørgaard, P. and Holttinen, H.: "A Multi-Turbine and Power Curve Approach". Nordic Wind Power Conference, 1.–2.3.2004, 2000, p. 5

[3] Kohler, S. and Agricola, A.-Cl. and Seidl, H.: "dena-Netzstudie II. Integration erneuerbarer Energien in die deutsche Stromversorgung im Zeitraum 2015 – 2020 mit Ausblick 2025". Technical report, 2010.

[4] Staffell, I. and Pfenninger, S.: "Using Bias-Corrected Reanalysis to Simulate Current and Future Wind Power Output". 2005, p. 11

5.11 Wake losses

Functions for applying wake losses to a wind speed time series.

wake_losses.reduce_wind_speed(wind_speed[, ...])	Reduces wind speed by a wind efficiency curve.
wake_losses.get_wind_efficiency_curve([...])	Reads wind efficiency curve(s) specified in *curve_name*.

5.11.1 windpowerlib.wake_losses.reduce_wind_speed

windpowerlib.wake_losses.**reduce_wind_speed**(*wind_speed*, *wind_efficiency_curve_name='dena_mean'*)

Reduces wind speed by a wind efficiency curve.

The wind efficiency curves are provided in the windpowerlib and were calculated in the dena-Netzstudie II and in the work of Knorr (see[1] and[2]).

Parameters

- **wind_speed** (pandas.Series or numpy.array) – Wind speed time series.
- **wind_efficiency_curve_name** (*str*) – Name of the wind efficiency curve. Use *get_wind_efficiency_curve()* to get all provided wind efficiency curves. Default: 'dena_mean'.

Returns *wind_speed* reduced by wind efficiency curve.

Return type pandas.Series or np.array

References

5.11.2 windpowerlib.wake_losses.get_wind_efficiency_curve

windpowerlib.wake_losses.**get_wind_efficiency_curve**(*curve_name='all'*)

Reads wind efficiency curve(s) specified in *curve_name*.

Parameters **curve_name** (*str or list(str)*) – Specifies the curve. Use 'all' to get all curves in a MultiIndex DataFrame or one of the curve names to retrieve a single curve. Default: 'all'.

Returns Wind efficiency curve. Contains 'wind_speed' and 'efficiency' columns with wind speed in m/s and wind efficiency (dimensionless). If *curve_name* is 'all' or a list of strings a MultiIndex DataFrame is returned with curve names in the first level of the columns.

Return type pandas.DataFrame

[1] Kohler et.al.: "dena-Netzstudie II. Integration erneuerbarer Energien in die deutsche Stromversorgung im Zeitraum 2015 – 2020 mit Ausblick 2025.", Deutsche Energie-Agentur GmbH (dena), Tech. rept., 2010, p. 101

[2] Knorr, K.: "Modellierung von raum-zeitlichen Eigenschaften der Windenergieeinspeisung für wetterdatenbasierte Windleistungssimulationen". Universität Kassel, Diss., 2016, p. 124

Notes

The wind efficiency curves were generated in the “Dena Netzstudie”[1] and in the work of Kaspar Knorr[2]. The mean wind efficiency curve is an average curve from 12 wind farm distributed over Germany[1] or respectively an average from over 2000 wind farms in Germany[2]. Curves with the appendix ‘extreme’ are wind efficiency curves of single wind farms that are extremely deviating from the respective mean wind efficiency curve. For more information see[1] and[2].

References

Examples

```
# Example to plot all curves
fig, ax = plt.subplots() /n
df = get_wind_efficiency_curve(curve_name='all')
for t in df.columns.get_level_values(0).unique():
    p = df[t].set_index('wind_speed')['efficiency']
    p.name = t
    ax = p.plot(ax=ax, legend=True)
plt.show()
```

5.12 ModelChain

Creating a ModelChain object.

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

Running the ModelChain.

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

5.12.1 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

[1] Kohler et.al.: “dena-Netzstudie II. Integration erneuerbarer Energien in die deutsche Stromversorgung im Zeitraum 2015 – 2020 mit Ausblick 2025.”, Deutsche Energie-Agentur GmbH (dena), Tech. rept., 2010, p. 101

[2] Knorr, K.: “Modellierung von raum-zeitlichen Eigenschaften der Windenergieeinspeisung für wetterdatenbasierte Windleistungssimulationen”. Universität Kassel, Diss., 2016, p. 124

Return type *ModelChain*

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.calculate_power_output(…)	Calculates the power output of the wind power plant.

5.12.2 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 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.

5.12.3 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 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.

5.12.4 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 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.

5.12.5 windpowerlib.modelchain.ModelChain.calculate_power_output

`ModelChain.calculate_power_output(wind_speed_hub, density_hub)`

Calculates the power output of the wind power plant.

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

5.13 TurbineClusterModelChain

The TurbineClusterModelChain inherits all functions from the ModelChain.

Creating a TurbineClusterModelChain object.

turbine_cluster_modelchain.TurbineClusterModelChain(...)	Model to determine the output of a wind farm or wind turbine cluster.

Running the TurbineClusterModelChain.

turbine_cluster_modelchain.TurbineClusterModelChain.run_model(...)	Runs the model.

5.13.1 windpowerlib.turbine_cluster_modelchain.TurbineClusterModelChain.run_model

TurbineClusterModelChain.**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, density *density* in kg/m^3 and turbulence intensity *turbulence_intensity* depending on *power_output_model*, *density_model* and *standard_deviation_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',
```

(continues on next page)

(continued from previous page)

```
...                                              '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 TurbineClusterModelChain object.

turbine_cluster_modelchain. TurbineClusterModelChain. assign_power_curve(...)	Calculates the power curve of the wind turbine cluster.
turbine_cluster_modelchain. TurbineClusterModelChain. temperature_hub(...)	Calculates the temperature of air at hub height.
turbine_cluster_modelchain. TurbineClusterModelChain. density_hub(...)	Calculates the density of air at hub height.
turbine_cluster_modelchain. TurbineClusterModelChain. wind_speed_hub(...)	Calculates the wind speed at hub height.
turbine_cluster_modelchain. TurbineClusterModelChain. calculate_power_output(...)	Calculates the power output of the wind power plant.

5.13.2 windpowerlib.turbine_cluster_modelchain.TurbineClusterModelChain.assign_power_cur

TurbineClusterModelChain.**assign_power_curve**(*weather_df*)

Calculates the power curve of the wind turbine cluster.

The power curve is aggregated from the wind farms' and wind turbines' power curves by using `power_plant.assign_power_curve()`. Depending on the parameters of the WindTurbineCluster power curves are smoothed and/or wake losses are taken into account.

Parameters **weather_df** (pandas.DataFrame) – DataFrame with weather data time series. If power curve smoothing *smoothing* is True and chosen method for calculating the standard deviation *standard_deviation_method* is *turbulence_intensity* the weather dataframe needs to either contain the turbulence intensity in column 'turbulence_intensity' or the roughness length in m in column 'roughness_length'. The turbulence intensity should be provided at hub height or at least at a height close to the hub height, as it cannot be inter- or extrapolated.

Returns

Return type self

5.13.3 windpowerlib.turbine_cluster_modelchain.TurbineClusterModelChain.temperature_hub

TurbineClusterModelChain.**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 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.

5.13.4 windpowerlib.turbine_cluster_modelchain.TurbineClusterModelChain.density_hub

`TurbineClusterModelChain.`**`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 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.

5.13.5 windpowerlib.turbine_cluster_modelchain.TurbineClusterModelChain.wind_speed_hub

`TurbineClusterModelChain.`**`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 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.

5.13.6 windpowerlib.turbine_cluster_modelchain.TurbineClusterModelChain.calculate_power_o

TurbineClusterModelChain.**calculate_power_output**(*wind_speed_hub*, *density_hub*)

Calculates the power output of the wind power plant.

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

5.14 Tools

Additional functions used in the windpowerlib.

tools.linear_interpolation_extrapolation(df, ...)	Linearly inter- or extrapolates between the values of a data frame.
tools.logarithmic_interpolation_extrapolation(df, ...)	Logarithmic inter- or extrapolation between the values of a data frame.
tools.gauss_distribution(function_variable, ...)	Gauss distribution.
tools.estimate_turbulence_intensity(height, ...)	Estimate turbulence intensity by the roughness length.

5.14.1 windpowerlib.tools.linear_interpolation_extrapolation

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

Linearly inter- or extrapolates between the values of a data frame.

This function can be used for the linear 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])])
>>> value = linear_interpolation_extrapolation(
...     weather_df['wind_speed'], 100)[0]
```

5.14.2 windpowerlib.tools.logarithmic_interpolation_extrapolation

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

Logarithmic inter- or extrapolation between the values of a data frame.

This function can be used for the logarithmic inter-/extrapolation of the wind speed if it is 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* ModelChain class is 'log_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 in *linear_interpolation_extrapolation()* 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 logarithmic inter- and extrapolation the following equation is used[1]:

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

References

5.14.3 windpowerlib.tools.gauss_distribution

`windpowerlib.tools.`**`gauss_distribution`**(*function_variable*, *standard_deviation*, *mean=0*)

Gauss distribution.

The Gauss distribution is used in the function `smooth_power_curve()` for power curve smoothing.

Parameters

- **`function_variable`** (*`float`*) – Variable of the gaussian distribution.
- **`standard_deviation`** (*`float`*) – Standard deviation of the Gauss distribution.
- **`mean`** (*`float`*) – Defines the offset of the Gauss distribution. Default: 0.

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

Return type pandas.Series or numpy.array

Notes

The following equation is used[1]:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right]$$

with: σ: standard deviation, μ: mean

References

5.14.4 windpowerlib.tools.estimate_turbulence_intensity

`windpowerlib.tools.`**`estimate_turbulence_intensity`**(*height*, *roughness_length*)

Estimate turbulence intensity by the roughness length.

Parameters

- **`height`** (*`float`*) – Height above ground in m at which the turbulence intensity is calculated.
- **`roughness_length`** (*`pandas.Series or numpy.array or float`*) – Roughness length.

[1] Knorr, K.: "Modellierung von raum-zeitlichen Eigenschaften der Windenergieeinspeisung für wetterdatenbasierte Windleistungssimulationen". Universität Kassel, Diss., 2016, p. 83

[1] Berendsen, H.: "A Student's Guide to Data and Error Analysis". New York, Cambridge University Press, 2011, p. 37

Notes

The following equation is used[1]:

$$TI = \frac{1}{\ln \left(\frac{h}{z_0} \right)}$$

with: TI: turbulence intensity, h: height, z_0: roughness length

References

5.15 ModelChain example

The `modelchain_example` consists of the following functions.

modelchain_example.get_weather_data([filename])	Imports weather data from a file.
modelchain_example.initialize_wind_turbines()	Initializes three *WindTurbine* objects.
modelchain_example.calculate_power_output(…)	Calculates power output of wind turbines using the *ModelChain*.
modelchain_example.plot_or_print(my_turbine, …)	Plots or prints power output and power (coefficient) curves.
modelchain_example.run_example()	Runs the basic example.

5.15.1 example.modelchain_example.get_weather_data

example.modelchain_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** (*str*) – Filename of the weather data file. Default: 'weather.csv'.

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

Returns 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

[1] Knorr, K.: "Modellierung von raum-zeitlichen Eigenschaften der Windenergieeinspeisung für wetterdatenbasierte Windleistungssimulationen". Universität Kassel, Diss., 2016, p. 88

5.15.2 example.modelchain_example.initialize_wind_turbines

example.modelchain_example.**initialize_wind_turbines**()

Initializes three *WindTurbine* objects.

This function shows three ways to initialize a WindTurbine object. You can either use turbine data from the OpenEnergy Database (oedb) turbine library that is provided along with the windpowerlib, as done for the 'enercon_e126', or specify your own turbine by directly providing a power (coefficient) curve, as done below for 'my_turbine', or provide your own turbine data in csv files, as done for 'dummy_turbine'.

To get a list of all wind turbines for which power and/or power coefficient curves are provided execute ' *windpowerlib.wind_turbine.get_turbine_types()*'.

Returns *WindTurbine*, *WindTurbine*)

Return type Tuple (*WindTurbine*,

5.15.3 example.modelchain_example.calculate_power_output

example.modelchain_example.**calculate_power_output**(*weather*, *my_turbine*, *e126*, *dummy_turbine*)

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'. Of course, you can also use the default methods while only changing one or two of them, as done for 'dummy_turbine'.

Parameters

- **weather** (pandas.DataFrame) – Contains weather data time series.
- **my_turbine** (*WindTurbine*) – WindTurbine object with self provided power curve.
- **e126** (*WindTurbine*) – WindTurbine object with power curve from the OpenEnergy Database turbine library.
- **dummy_turbine** (*WindTurbine*) – WindTurbine object with power coefficient curve from example file.

5.15.4 example.modelchain_example.plot_or_print

example.modelchain_example.**plot_or_print**(*my_turbine*, *e126*, *dummy_turbine*)

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 the OpenEnergy Database turbine library.
- **dummy_turbine** (*WindTurbine*) – WindTurbine object with power coefficient curve from example file.

5.15.5 example.modelchain_example.run_example

`example.modelchain_example.run_example()`

Runs the basic example.

5.16 TurbineClusterModelChain example

The `turbine_cluster_modelchain_example` consists of the following functions as well as it uses functions of the `modelchain_example`.

turbine_cluster_modelchain_example.initialize_wind_farms(...)	Initializes two *WindFarm* objects.
turbine_cluster_modelchain_example.initialize_wind_turbine_cluster(...)	Initializes a *WindTurbineCluster* object.
turbine_cluster_modelchain_example.calculate_power_output(...)	Calculates power output of wind farms and clusters using the *TurbineClusterModelChain*.
turbine_cluster_modelchain_example.plot_or_print(...)	Plots or prints power output and power (coefficient) curves.
turbine_cluster_modelchain_example.run_example()	Runs the example.

5.16.1 example.turbine_cluster_modelchain_example.initialize_wind_farms

`example.turbine_cluster_modelchain_example.initialize_wind_farms(my_turbine, e126)`

Initializes two *WindFarm* objects.

This function shows how to initialize a WindFarm object. A WindFarm needs a wind turbine fleet specifying the wind turbines and their number or total installed capacity (in Watt) in the farm. Optionally, you can provide a wind farm efficiency (which can be constant or dependent on the wind speed) and a name as an identifier. See *WindFarm* for more information.

Parameters

- **my_turbine** (*WindTurbine*) – WindTurbine object with self provided power curve.
- **e126** (*WindTurbine*) – WindTurbine object with power curve from the OpenEnergy Database turbine library.

Returns

Return type tuple(*WindFarm*, *WindFarm*)

5.16.2 example.turbine_cluster_modelchain_example.initialize_wind_turbine_cluster

`example.turbine_cluster_modelchain_example.initialize_wind_turbine_cluster(example_farm, example_farm_2)`

Initializes a *WindTurbineCluster* object.

Function shows how to initialize a WindTurbineCluster object. A WindTurbineCluster consists of wind farms that are specified through the *wind_farms* parameter. Optionally, you can provide a name as an identifier.

Parameters

- **example_farm** (*WindFarm*) – WindFarm object without provided efficiency.
- **example_farm_2** (*WindFarm*) – WindFarm object with constant wind farm efficiency.

Returns

Return type *WindTurbineCluster*

5.16.3 example.turbine_cluster_modelchain_example.calculate_power_output

example.turbine_cluster_modelchain_example.**calculate_power_output**(*weather, example_farm, example_cluster*)

Calculates power output of wind farms and clusters using the *TurbineClusterModelChain*.

The *TurbineClusterModelChain* is a class that provides all necessary steps to calculate the power output of a wind farm or cluster. You can either use the default methods for the calculation steps, as done for 'example_farm', or choose different methods, as done for 'example_cluster'.

Parameters

- **weather** (pandas.DataFrame) – Contains weather data time series.
- **example_farm** (*WindFarm*) – WindFarm object without provided efficiency.
- **example_cluster** (*WindTurbineCluster*) – WindTurbineCluster object.

5.16.4 example.turbine_cluster_modelchain_example.plot_or_print

example.turbine_cluster_modelchain_example.**plot_or_print**(*example_farm, example_cluster*)

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

Parameters

- **example_farm** (*WindFarm*) – WindFarm object without provided efficiency.
- **example_cluster** (*WindTurbineCluster*) – WindTurbineCluster object.

5.16.5 example.turbine_cluster_modelchain_example.run_example

example.turbine_cluster_modelchain_example.**run_example**()

Runs the example.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Index

Symbols

`__init__()` (*windpowerlib.modelchain.ModelChain method*), 34
`__init__()` (*windpowerlib.turbine_cluster_modelchain.TurbineClusterModelChain method*), 37
`__init__()` (*windpowerlib.wind_farm.WindFarm method*), 31
`__init__()` (*windpowerlib.wind_turbine.WindTurbine method*), 29
`__init__()` (*windpowerlib.wind_turbine.WindTurbineGroup method*), 45
`__init__()` (*windpowerlib.wind_turbine_cluster.WindTurbineCluster method*), 32

A

`assign_power_curve()` (*windpowerlib.turbine_cluster_modelchain.TurbineClusterModelChain method*), 60
`assign_power_curve()` (*windpowerlib.wind_farm.WindFarm method*), 48
`assign_power_curve()` (*windpowerlib.wind_turbine_cluster.WindTurbineCluster method*), 49

B

`barometric()` (*in module windpowerlib.density*), 39
`block_width` (*windpowerlib.turbine_cluster_modelchain.TurbineClusterModelChain attribute*), 36

C

`calculate_power_output()` (*in module example.modelchain_example*), 66
`calculate_power_output()` (*in module example.turbine_cluster_modelchain_example*), 68
`calculate_power_output()` (*windpowerlib.modelchain.ModelChain method*), 58
`calculate_power_output()` (*windpowerlib.turbine_cluster_modelchain.TurbineClusterModelChain method*), 62
`check_and_complete_wind_turbine_fleet()` (*windpowerlib.wind_farm.WindFarm method*), 47

D

`density_correction` (*windpowerlib.modelchain.ModelChain attribute*), 34
`density_correction` (*windpowerlib.turbine_cluster_modelchain.TurbineClusterModelChain attribute*), 37
`density_hub()` (*windpowerlib.modelchain.ModelChain method*), 58
`density_hub()` (*windpowerlib.turbine_cluster_modelchain.TurbineClusterModelChain method*), 61
`density_model` (*windpowerlib.modelchain.ModelChain attribute*), 34
`density_model` (*windpowerlib.turbine_cluster_modelchain.TurbineClusterModelChain attribute*), 37

E

`efficiency` (*windpowerlib.wind_farm.WindFarm attribute*), 30
`estimate_turbulence_intensity()` (*in module windpowerlib.tools*), 64

G

`gauss_distribution()` (*in module windpowerlib.tools*), 64
`get_turbine_data_from_file()` (*in module windpowerlib.wind_turbine*), 43
`get_turbine_types()` (*in module windpowerlib.wind_turbine*), 44

get_weather_data() (in module example.modelchain_example), 65
get_wind_efficiency_curve() (in module windpowerlib.wake_losses), 55

H

hellman() (in module windpowerlib.wind_speed), 42
hellman_exp (windpowerlib.modelchain.ModelChain attribute), 34
hellman_exp (windpowerlib.turbine_cluster_modelchain.TurbineClusterModelChain attribute), 37
hub_height (windpowerlib.wind_farm.WindFarm attribute), 30
hub_height (windpowerlib.wind_turbine.WindTurbine attribute), 28
hub_height (windpowerlib.wind_turbine_cluster.WindTurbineCluster attribute), 32

I

ideal_gas() (in module windpowerlib.density), 40
initialize_wind_farms() (in module example.turbine_cluster_modelchain_example), 67
initialize_wind_turbine_cluster() (in module example.turbine_cluster_modelchain_example), 67
initialize_wind_turbines() (in module example.modelchain_example), 66

L

linear_gradient() (in module windpowerlib.temperature), 38
linear_interpolation_extrapolation() (in module windpowerlib.tools), 62
load_turbine_data_from_oedb() (in module windpowerlib.wind_turbine), 43
logarithmic_interpolation_extrapolation() (in module windpowerlib.tools), 63
logarithmic_profile() (in module windpowerlib.wind_speed), 41

M

mean_hub_height() (windpowerlib.wind_farm.WindFarm method), 47
mean_hub_height() (windpowerlib.wind_turbine_cluster.WindTurbineCluster method), 49
ModelChain (class in windpowerlib.modelchain), 32

N

name (windpowerlib.wind_farm.WindFarm attribute), 30
name (windpowerlib.wind_turbine_cluster.WindTurbineCluster attribute), 32
nominal_power (windpowerlib.wind_farm.WindFarm attribute), 47
nominal_power (windpowerlib.wind_turbine.WindTurbine attribute), 28
nominal_power (windpowerlib.wind_turbine_cluster.WindTurbineCluster attribute), 49

O

obstacle_height (windpowerlib.modelchain.ModelChain attribute), 34
obstacle_height (windpowerlib.turbine_cluster_modelchain.TurbineClusterModelChain attribute), 37

P

plot_or_print() (in module example.modelchain_example), 66
plot_or_print() (in module example.turbine_cluster_modelchain_example), 68
power_coefficient_curve (windpowerlib.wind_turbine.WindTurbine attribute), 28
power_coefficient_curve() (in module windpowerlib.power_output), 50
power_curve (windpowerlib.turbine_cluster_modelchain.TurbineClusterModelChain attribute), 37
power_curve (windpowerlib.wind_farm.WindFarm attribute), 30
power_curve (windpowerlib.wind_turbine.WindTurbine attribute), 28
power_curve (windpowerlib.wind_turbine_cluster.WindTurbineCluster attribute), 32
power_curve() (in module windpowerlib.power_output), 51
power_curve_density_correction() (in module windpowerlib.power_output), 52
power_output (windpowerlib.modelchain.ModelChain attribute), 34
power_output (windpowerlib.turbine_cluster_modelchain.TurbineClusterModelChain attribute), 37
power_output_model (windpowerlib.modelchain.ModelChain attribute), 34
power_output_model (windpowerlib.turbine_cluster_modelchain.TurbineClusterModelChain attribute), 37

power_plant (*windpowerlib.modelchain.ModelChain attribute*), 33
power_plant (*windpowerlib.turbine_cluster_modelchain.TurbineClusterModelChain attribute*), 36

R

reduce_wind_speed() (*in module windpowerlib.wake_losses*), 55
rotor_diameter (*windpowerlib.wind_turbine.WindTurbine attribute*), 28
run_example() (*in module example.modelchain_example*), 67
run_example() (*in module example.turbine_cluster_modelchain_example*), 68
run_model() (*windpowerlib.modelchain.ModelChain method*), 56
run_model() (*windpowerlib.turbine_cluster_modelchain.TurbineClusterModelChain method*), 59

S

smooth_power_curve() (*in module windpowerlib.power_curves*), 53
smoothing (*windpowerlib.turbine_cluster_modelchain.TurbineClusterModelChain attribute*), 36
smoothing_order (*windpowerlib.turbine_cluster_modelchain.TurbineClusterModelChain attribute*), 36
standard_deviation_method (*windpowerlib.turbine_cluster_modelchain.TurbineClusterModelChain attribute*), 36

T

temperature_hub() (*windpowerlib.modelchain.ModelChain method*), 57
temperature_hub() (*windpowerlib.turbine_cluster_modelchain.TurbineClusterModelChain method*), 60
temperature_model (*windpowerlib.modelchain.ModelChain attribute*), 34
temperature_model (*windpowerlib.turbine_cluster_modelchain.TurbineClusterModelChain attribute*), 37
to_group() (*windpowerlib.wind_turbine.WindTurbine method*), 46
turbine_type (*windpowerlib.wind_turbine.WindTurbine attribute*), 28
TurbineClusterModelChain (*class in windpowerlib.turbine_cluster_modelchain*), 35

W

wake_losses_model (*windpowerlib.turbine_cluster_modelchain.TurbineClusterModelChain attribute*), 36
wake_losses_to_power_curve() (*in module windpowerlib.power_curves*), 54
wind_farms (*windpowerlib.wind_turbine_cluster.WindTurbineCluster attribute*), 31
wind_speed_hub() (*windpowerlib.modelchain.ModelChain method*), 58
wind_speed_hub() (*windpowerlib.turbine_cluster_modelchain.TurbineClusterModelChain method*), 61
wind_speed_model (*windpowerlib.modelchain.ModelChain attribute*), 34
wind_speed_model (*windpowerlib.turbine_cluster_modelchain.TurbineClusterModelChain attribute*), 37
wind_turbine_fleet (*windpowerlib.wind_farm.WindFarm attribute*), 30
WindFarm (*class in windpowerlib.wind_farm*), 29
WindTurbine (*class in windpowerlib.wind_turbine*), 27
WindTurbineCluster (*class in windpowerlib.wind_turbine_cluster*), 31
WindTurbineGroup (*class in windpowerlib.wind_turbine*), 45