windpowerlib Documentation

oemof developer group

Jan 18, 2019

Contents

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. We highly recommend to use virtual environments.

```
pip install windpowerlib
```

The windpowerlib is designed for Python 3 and tested on Python >= 3.5. 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.

For retrieving power (coefficient) curves from the OpenEnergy Database (oedb) the python package requests will be installed with your windpowerlib installation. The windpowerlib was tested with requests version 2.20.1 but might work with lower versions.

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 pip3 though some Linux users reported that it is easier and more stable to use the pre-built packages of your Linux distribution.

1.4 Examples and basic usage

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

- `ModelChain example (Python script)`
- `ModelChain example (Jupyter notebook)`
- `Example data file`

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

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 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 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
```

1.6 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.

1.7 License

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

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

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

    """

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


# Read weather data from csv
```

(continues on next page)

(continued from previous page)

```
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 specify your own turbine, as done below for 'my_turbine', or fetch power and/or power coefficient curve data from data files provided by the windpowerlib, as done for the 'enercon_e126', or provide your turbine data in csv files as done for the 'dummy_turbine' with an example file.

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
```

[33]:
```
# specification of own wind turbine (Note: power coefficient values and
# nominal power have to be in Watt)
my_turbine = {
    'name': 'myTurbine',
    'nominal_power': 3e6,  # in W
    'hub_height': 105,  # in m
    'rotor_diameter': 90,  # in m
    'power_curve': pd.DataFrame(
            data={'power': [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
    }
# initialisze WindTurbine object
my_turbine = WindTurbine(**my_turbine)
```

[6]:
```
# specification of wind turbine where power curve is provided
# if you want to use the power coefficient curve change the value of
# 'fetch_curve' to 'power_coefficient_curve'
enercon_e126 = {
        'name': 'E-126/4200',  # turbine type as in register #
        'hub_height': 135,  # in m
        'rotor_diameter': 127,  # in m
        'fetch_curve': 'power_curve',  # fetch power curve #
        'data_source': 'oedb'  # data source oedb or name of csv file
    }
# initialize WindTurbine object
e126 = WindTurbine(**enercon_e126)
```

```
DEBUG:urllib3.connectionpool:Starting new HTTP connection (1): oep.iks.cs.ovgu.de:80
DEBUG:urllib3.connectionpool:http://oep.iks.cs.ovgu.de:80 "GET //api/v0/schema/
↪model_draft/tables/openfred_windpower_powercurve/rows/ HTTP/1.1" 301 438
DEBUG:urllib3.connectionpool:Starting new HTTP connection (1): openenergy-platform.
↪org:80
DEBUG:urllib3.connectionpool:http://openenergy-platform.org:80 "GET /api/v0/schema/
↪model_draft/tables/openfred_windpower_powercurve/rows/ HTTP/1.1" 200 150623
INFO:root:Data base connection successful.
```

[35]:
```
# specification of wind turbine where power coefficient curve is provided
# by a csv file
source = 'data/example_power_coefficient_curves.csv'
dummy_turbine = {
    'name': 'DUMMY 1',  # turbine type as in file #
    'hub_height': 100,  # in m
    'rotor_diameter': 70,  # in m
    'fetch_curve': 'power_coefficient_curve',  # fetch cp curve #
    'data_source': source  # data source
}
# 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. If you use the 'run_model' method first the wind speed and density (if necessary) at hub height are calculated and then used to calculate the power output. You can either use the default methods for the calculation steps, as done for 'my_turbine',

or choose different methods, as done for the 'e126'. Of course, you can also use the default methods while only changing one or two of them, as done for 'dummy_turbine'.

[7]:
```
# 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.
```

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

# 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.
```

[38]:
```
# 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

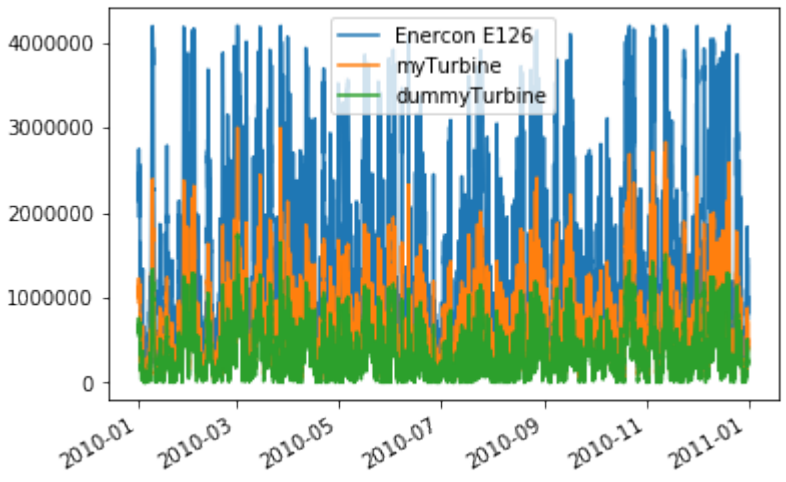
If you have matplotlib installed you can visualize the calculated power output and used power (coefficient) curves.

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

```
DEBUG:matplotlib:CACHEDIR=/home/sabine/.cache/matplotlib
DEBUG:matplotlib.font_manager:Using fontManager instance from /home/sabine/.cache/
↪matplotlib/fontlist-v300.json
DEBUG:matplotlib.pyplot:Loaded backend module://ipykernel.pylab.backend_inline␣
↪version unknown.
DEBUG:matplotlib.pyplot:Loaded backend module://ipykernel.pylab.backend_inline␣
↪version unknown.
DEBUG:matplotlib.pyplot:Loaded backend module://ipykernel.pylab.backend_inline␣
↪version unknown.
```

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



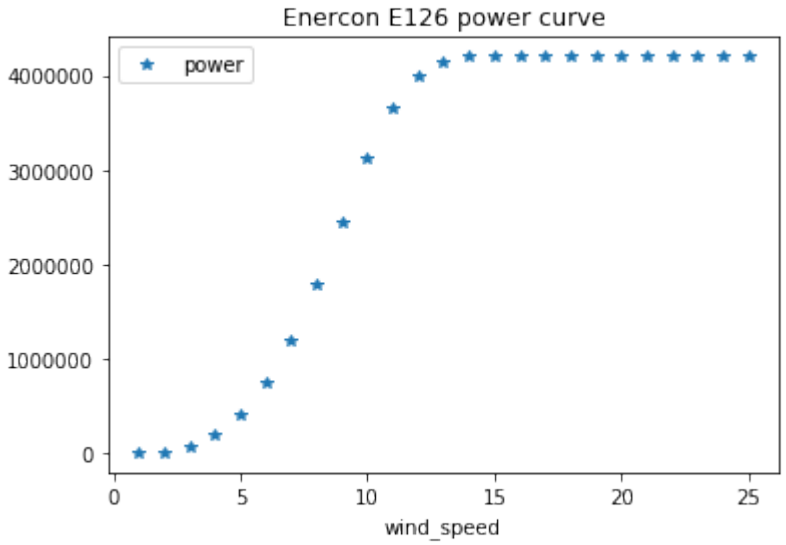
[11]:
```
# plot power (coefficient) curves
if plt:
    if e126.power_coefficient_curve is not None:
        e126.power_coefficient_curve.plot(
            x='wind_speed', y='power coefficient', style='*',
            title='Enercon E126 power coefficient curve')
```

(continues on next page)

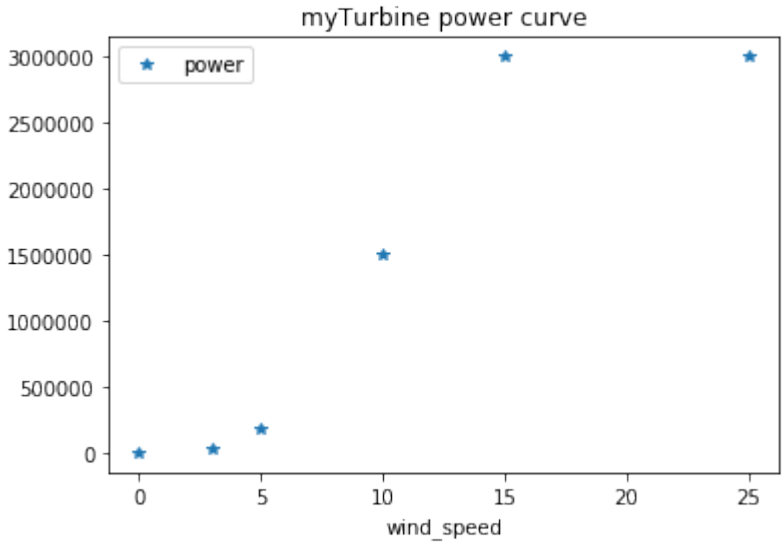
(continued from previous page)

```
        plt.show()
    if e126.power_curve is not None:
        e126.power_curve.plot(x='wind_speed', y='power', style='*',
                              title='Enercon E126 power curve')
        plt.show()
    if my_turbine.power_coefficient_curve is not None:
        my_turbine.power_coefficient_curve.plot(
            x='wind_speed', y='power coefficient', style='*',
            title='myTurbine power coefficient curve')
        plt.show()
    if my_turbine.power_curve is not None:
        my_turbine.power_curve.plot(x='wind_speed', y='power', style='*',
                                    title='myTurbine power curve')
        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 with 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 explanations.

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`.

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

import modelchain_example as mc_e
from windpowerlib.turbine_cluster_modelchain import TurbineClusterModelChain
from windpowerlib.wind_turbine_cluster import WindTurbineCluster
from windpowerlib.wind_farm import 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
my_turbine, e126, dummy_turbine = mc_e.initialize_wind_turbines()
print()
print('nominal power of my_turbine: {}'.format(my_turbine.nominal_power))
```

```
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
```

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

```
nominal power of my_turbine: 3000000.0
```

2.2.2 Initialize wind farm

To initialize a specific wind farm you can use a dictionary that contains the basic parameters. A wind farm is defined by its name, wind turbine fleet, and optionally also by a wind farm efficiency and the wind farm's location.

A wind farm efficiency can be a constant value or be dependent on the wind speed. The coordinates are not being used here but are necessary if you need to assign your wind farm to a certain weather data point.

[3]:
```
# specification of wind farm data
example_farm_data = {
    'name': 'example_farm',
    'wind_turbine_fleet': [{'wind_turbine': my_turbine,
                            'number_of_turbines': 6},
                           {'wind_turbine': e126,
                            'number_of_turbines': 3}
                           ]}

# initialize WindFarm object
example_farm = WindFarm(**example_farm_data)
```

[4]:
```
# specification of wind farm data (2) containing a wind farm efficiency
# and coordinates
example_farm_2_data = {
    'name': 'example_farm_2',
    'wind_turbine_fleet': [{'wind_turbine': my_turbine,
                            'number_of_turbines': 6},
                           {'wind_turbine': e126,
                            'number_of_turbines': 3}],
    'efficiency': 0.9,
    'coordinates': [52.2, 13.1]}

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

print('nominal power of first turbine type of example_farm_2: {}'.format(
    example_farm_2.wind_turbine_fleet[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 name, wind farms and optionally by its location.

[5]:
```
# specification of cluster data
example_cluster_data = {
    'name': 'example_cluster',
    'wind_farms': [example_farm, example_farm_2],
    'coordinates': [52.2, 13.1]}

# 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 here 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 calulated.

[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 smoothed by method: turbulence_intensity
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': 'constant_efficiency',  #
                                            # 'dena_mean' (default), None,
                                            # 'power_efficiency_curve',
                                            # 'constant_efficiency' or name of
                                            #  a wind efficiency curve
                #  see :py:func:`~.wake_losses.get_wind_efficiency_curve`
        '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'
```

(continues on next page)

(continued from previous page)

```
        '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 constant_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

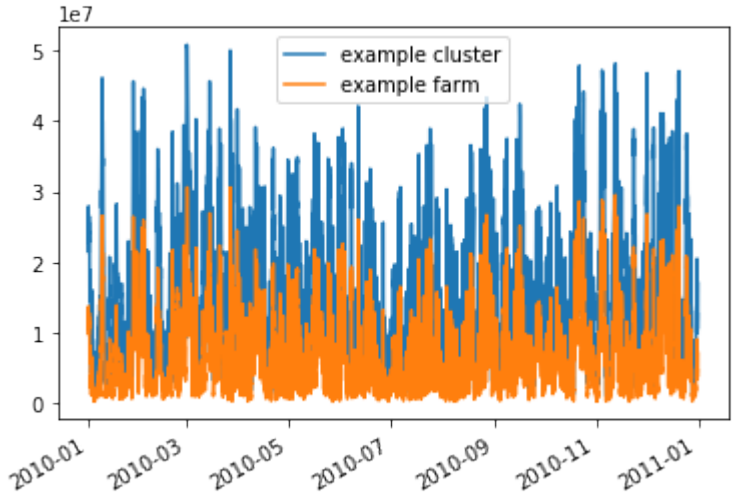
If you have matplotlib installed you can visualize the calculated power output.

[8]:
```
# try to import matplotlib
try:
    from matplotlib import pyplot as plt
    # matplotlib inline needed in notebook to plot inline
    %matplotlib inline
except ImportError:
    plt = None
```

```
DEBUG:matplotlib.pyplot:Loaded backend module://ipykernel.pylab.backend_inline␣
↪version unknown.
```

[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.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
```


1e7
5
4
3
2
1
0
example cluster
example farm
2010-01
2010-03
2010-05
2010-07
2010-09
2010-11
2011-01

[ ]:

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`).

Desciptions 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 of considering wake losses is applying 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 to feed-in time series has the advantage that the power curves can further be aggregated to achieve 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

4.1 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.

4.1.1 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.

4.1.2 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

4.1.3 Testing

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

4.1.4 Documentation

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

4.1.5 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()*

4.1.6 Other changes

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

4.1.7 Contributors

- Sabine Haas
- Uwe Krien
- Birgit Schachler

4.2 v0.0.6 (July 07, 2017)

4.2.1 Documentation

- added basic usage section and jupyter notebook

4.2.2 Testing

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

4.2.3 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

4.2.4 Contributors

- Sabine Haas
- Birgit Schachler
- Uwe Krien

4.3 v0.0.5 (April 10, 2017)

4.3.1 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

4.3.2 Documentation

- added references

4.3.3 Testing

- tests for all modules were added

4.3.4 Contributors

- Sabine Haas
- Birgit Schachler
- Uwe Krien

CHAPTER 5

API

5.1 Classes

wind_turbine.WindTurbine(name, hub_height[, ...])	Defines a standard set of wind turbine attributes.
wind_farm.WindFarm(name, 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**(*name*, *hub_height*, *rotor_diameter=None*, *power_coefficient_curve=None*, *power_curve=None*, *nominal_power=None*, *fetch_curve=None*, *coordinates=None*, *data_source='oedb'*)

Defines a standard set of wind turbine attributes.

Parameters

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

- **power_curve** (*None, pandas.DataFrame or dictionary*) – Power curve of the wind turbine. DataFrame/dictionary must have 'wind_speed' and 'value' columns/keys with wind speeds in m/s and the corresponding power curve value in W. Default: None.
- **nominal_power** (*None or float*) – The nominal output of the wind turbine in W. Default: None.
- **fetch_curve** (*string*) – Parameter to specify whether a power or power coefficient curve should be retrieved from the provided turbine data. Valid options are 'power_curve' and 'power_coefficient_curve'. Default: None.
- **coordinates** (*list or None*) – List of coordinates [lat, lon] of location for loading data. Default: None.
- **data_source** (*string*) – Specifies whether turbine data (f.e. nominal power, power curve, power coefficient curve) is loaded from the OpenEnergy Database ('oedb') or from a csv file ('<path including file name>'). Default: 'oedb'. See *example_power_curves.csv' and 'example_power_coefficient_curves.csv* in example/data for the required form of a csv file (more columns can be added).

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

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

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

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

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

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

coordinates
list or None – List of coordinates [lat, lon] of location for loading data. Default: None.

power_output
pandas.Series – The calculated power output of the wind turbine. Default: None.

Notes

Your wind turbine object should have a power coefficient or power curve. You can set the *fetch_curve* parameter and the *data_source* parameter if you want to automatically fetch a curve from a data set provided in the OpenEnergy Database (oedb) or want to read a csv file that you provide. See *example_power_curves.csv' and 'example_power_coefficient_curves.csv* in example/data for the required form of such a csv file (more columns can be added).

Examples

```
>>> from windpowerlib import wind_turbine
>>> enerconE126 = {
...     'hub_height': 135,
...     'rotor_diameter': 127,
...     'name': 'E-126/4200',
...     'fetch_curve': 'power_curve',
...     'data_source': 'oedb'}
>>> e126 = wind_turbine.WindTurbine(**enerconE126)
>>> print(e126.nominal_power)
4200000.0
```

__init__(*name*, *hub_height*, *rotor_diameter=None*, *power_coefficient_curve=None*, *power_curve=None*, *nominal_power=None*, *fetch_curve=None*, *coordinates=None*, *data_source='oedb'*)

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

Methods

__init__(name, hub_height[, rotor_diameter, ...])	Initialize self.
fetch_turbine_data(fetch_curve, data_source)	Fetches data of the requested wind turbine.

5.1.2 windpowerlib.wind_farm.WindFarm

class windpowerlib.wind_farm.**WindFarm**(*name*, *wind_turbine_fleet*, *coordinates=None*, *efficiency=None*)

Defines a standard set of wind farm attributes.

Parameters

- **name** (*string*) – Name of the wind farm.
- **wind_turbine_fleet** (*list of dictionaries*) – Wind turbines of wind farm. Dictionaries 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 keys.
- **coordinates** (*list or None*) – List of coordinates [lat, lon] of location for loading data. Default: None.
- **efficiency** (*float or pd.DataFrame*) – Efficiency of the wind farm. Either constant (float) power efficiency curve (pd.DataFrame) containing 'wind_speed' and 'efficiency' columns/keys with wind speeds in m/s and the corresponding dimensionless wind farm efficiency. Default: None.

name

string – Name of the wind farm.

wind_turbine_fleet

list of dictionaries – Wind turbines of wind farm. Dictionaries 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 keys.

coordinates
list or None – List of coordinates [lat, lon] of location for loading data. Default: None.

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

hub_height
float – The calculated mean hub height of the wind farm.

installed_power
float – The calculated installed power of the wind farm.

power_curve
pandas.DataFrame or None – The calculated power curve of the wind farm.

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

Examples

```
>>> from windpowerlib import wind_farm
>>> from windpowerlib import wind_turbine
>>> enerconE126 = {
...     'hub_height': 135,
...     'rotor_diameter': 127,
...     'name': 'E-126/4200',
...     'fetch_curve': 'power_curve',
...     'data_source': 'oedb'}
>>> e126 = wind_turbine.WindTurbine(**enerconE126)
>>> example_farm_data = {
...     'name': 'example_farm',
...     'wind_turbine_fleet': [{'wind_turbine': e126,
...                             'number_of_turbines': 6}]}
>>> example_farm = wind_farm.WindFarm(**example_farm_data)
>>> example_farm.installed_power = example_farm.get_installed_power()
>>> print(example_farm.installed_power)
25200000.0
```

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

Methods

__init__(name, wind_turbine_fleet[, ...])	Initialize self.
assign_power_curve([wake_losses_model, ...])	Calculates the power curve of a wind farm.
get_installed_power()	Calculates the installed power of the wind farm.
mean_hub_height()	Calculates the mean hub height of the wind farm.

5.1.3 windpowerlib.wind_turbine_cluster.WindTurbineCluster

class windpowerlib.wind_turbine_cluster.**WindTurbineCluster**(*name*, *wind_farms*, *coordinates=None*)

Defines a standard set of wind turbine cluster attributes.

Parameters

- **name** (*string*) – Name of the wind turbine cluster.
- **wind_farms** (*list*) – Contains objects of the *WindFarm*.
- **coordinates** (*list or None*) – Coordinates of location [lat, lon]. Can be practical for loading weather data. Default: None.

name
string – Name of the wind turbine cluster.

wind_farms
list – Contains objects of the *WindFarm*.

coordinates
list or None – Coordinates of location [lat, lon]. Can be practical for loading weather data. Default: None.

hub_height
float – The calculated average hub height of the wind turbine cluster.

installed_power
float – The calculated installed power of the wind turbine cluster.

power_curve
pandas.DataFrame or None – The calculated power curve of the wind turbine cluster.

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

__init__(*name*, *wind_farms*, *coordinates=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

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

Model to determine the output of a wind turbine

Parameters

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

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

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

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

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

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

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

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

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

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

Examples

```
>>> from windpowerlib import modelchain
>>> from windpowerlib import wind_turbine
>>> enerconE126 = {
...     'hub_height': 135,
...     'rotor_diameter': 127,
...     'name': 'E-126/4200',
...     'fetch_curve': 'power_curve',
...      'data_source': 'oedb'}
>>> 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*)
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** (*string*) – Defines the method for talking wake losses within the farm into consideration. Options: None, 'power_efficiency_curve' or 'constant_efficiency' or the name of a wind efficiency curve like 'dena_mean'. Default: 'dena_mean'. Use *get_wind_efficiency_curve()* for all provided wind efficiency curves.
- **smoothing** (*boolean*) – 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** (*string*) – Method for calculating the standard deviation for the Gauss distribution. Options: 'turbulence_intensity', 'Staffell_Pfenninger'. Default: 'turbulence_intensity'.
- **smoothing_order** (*string*) – 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'.

Other Parameters

- **wind_speed_model** (*string*) – Parameter to define which model to use to calculate the wind speed at hub height. Valid options are 'logarithmic', 'hellman' and 'interpolation_extrapolation'.
- **temperature_model** (*string*) – Parameter to define which model to use to calculate the temperature of air at hub height. Valid options are 'linear_gradient' and 'interpolation_extrapolation'.
- **density_model** (*string*) – Parameter to define which model to use to calculate the density of air at hub height. Valid options are 'barometric', 'ideal_gas' and 'interpolation_extrapolation'.
- **power_output_model** (*string*) – Parameter to define which model to use to calculate the turbine power output. Valid options are 'power_curve' and 'power_coefficient_curve'.
- **density_correction** (*boolean*) – If the parameter is True the density corrected power curve is used for the calculation of the turbine power output.
- **obstacle_height** (*float*) – Height of obstacles in the surrounding area of the wind turbine in m. Set *obstacle_height* to zero for wide spread obstacles.
- **hellman_exp** (*float*) – The Hellman exponent, which combines the increase in wind speed due to stability of atmospheric conditions and surface roughness into one constant.

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

wake_losses_model
string – Defines the method for talking wake losses within the farm into consideration. Options: None, 'power_efficiency_curve' or 'constant_efficiency' or the name of a wind efficiency curve like 'dena_mean'. Default: 'dena_mean'. Use *get_wind_efficiency_curve()* for all provided wind efficiency curves.

smoothing
boolean – 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
string – Method for calculating the standard deviation for the Gauss distribution. Options: 'turbulence_intensity', 'Staffell_Pfenninger'. Default: 'turbulence_intensity'.

smoothing_order
string – 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'.

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

pandas.DataFrame or None
The calculated power curve of the wind farm.

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

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

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

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

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

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

hellman_exp
float – The Hellman exponent, which combines the increase in wind speed due to stability of atmospheric conditions and surface roughness into one constant.

__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 6 – 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.WindTurbine.fetch_turbine_data(...)	Fetches data of the requested wind turbine.
wind_turbine.get_turbine_data_from_file()	Fetches power (coefficient) curve data from a csv file.
wind_turbine.get_turbine_data_from_oedb()	Gets turbine data from the OpenEnergy Database (oedb).
wind_turbine.load_turbine_data_from_oedb()	Loads turbine data from the OpenEnergy Database (oedb).
wind_turbine.get_turbine_types([print_out])	Get the names of all possible wind turbine types for which the power coefficient curve or power curve is provided in the OpenEnergy Data Base (oedb).

5.5.1 windpowerlib.wind_turbine.WindTurbine.fetch_turbine_data

WindTurbine.**fetch_turbine_data**(*fetch_curve*, *data_source*)

Fetches data of the requested wind turbine.

Method fetches nominal power as well as power coefficient curve or power curve from a data set provided in the OpenEnergy Database (oedb). You can also import your own power (coefficient) curves from a file. For that the wind speeds in m/s have to be in the first row and the corresponding power coefficient curve values or power curve values in W in a row where the first column contains the turbine name. See *example_power_curves.csv' and 'example_power_coefficient_curves.csv* in example/data for the required form of a csv file (more columns can be added). See *get_turbine_data_from_file()* for an example reading data from a csv file.

Parameters

- **fetch_curve** (*string*) – Parameter to specify whether a power or power coefficient curve should be retrieved from the provided turbine data. Valid options are 'power_curve' and 'power_coefficient_curve'. Default: None.
- **data_source** (*string*) – Specifies whether turbine data (f.e. nominal power, power curve, power coefficient curve) is loaded from the OpenEnergy Database ('oedb') or from a csv file ('<path including file name>'). Default: 'oedb'.

Returns

Return type self

Examples

```
>>> from windpowerlib import wind_turbine
>>> enerconE126 = {
...     'hub_height': 135,
...     'rotor_diameter': 127,
...     'name': 'E-126/4200',
...     'fetch_curve': 'power_coefficient_curve',
...     'data_source': 'oedb'}
```

(continues on next page)

(continued from previous page)

```
>>> e126 = wind_turbine.WindTurbine(**enerconE126)
>>> print(e126.power_coefficient_curve['value'][5])
0.44
>>> print(e126.nominal_power)
4200000.0
```

5.5.2 windpowerlib.wind_turbine.get_turbine_data_from_file

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

Fetches power (coefficient) curve data from a csv file.

See *example_power_curves.csv' and 'example_power_coefficient_curves.csv* in example/data for the required format of a csv file. The self-provided csv file may contain more columns than the example files. Only columns containing wind speed and the corresponding power or power coefficient as well as the column 'nominal_power' are taken into account.

Parameters

- **turbine_type** (*string*) – Specifies the turbine type data is fetched for.
- **file** (*string*) – 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) and nominal power (float). Power (coefficient) curve DataFrame contains power coefficient curve values (dimensionless) or power curve values in W as column names with the corresponding wind speeds in m/s.

Return type Tuple (pandas.DataFrame, float)

Examples

```
>>> from windpowerlib import wind_turbine
>>> import os
>>> source = os.path.join(os.path.dirname(__file__), '../example/data',
...                       'example_power_curves.csv')
>>> example_turbine = {
...    'hub_height': 100,
...    'rotor_diameter': 70,
...    'name': 'DUMMY 3',
...    'fetch_curve': 'power_curve',
...    'data_source': source}
>>> e_t_1 = wind_turbine.WindTurbine(**example_turbine)
>>> print(e_t_1.power_curve['value'][7])
18000.0
>>> print(e_t_1.nominal_power)
150000
```

5.5.3 windpowerlib.wind_turbine.get_turbine_data_from_oedb

windpowerlib.wind_turbine.**get_turbine_data_from_oedb**(*turbine_type*, *fetch_curve*)

Gets turbine data from the OpenEnergy Database (oedb).

Parameters

- **turbine_type** (*string*) – Specifies the turbine type data is fetched for. Use *get_turbine_types()* to see a table of all wind turbines for which power (coefficient) curve data is provided.
- **fetch_curve** (*string*) – Parameter to specify whether a power or power coefficient curve should be retrieved from the provided turbine data. Valid options are 'power_curve' and 'power_coefficient_curve'. Default: None.

Returns Power curve or power coefficient curve (pandas.DataFrame) and nominal power (float). Power (coefficient) curve DataFrame contains power coefficient curve values (dimensionless) or power curve values in W with the corresponding wind speeds in m/s.

Return type Tuple (pandas.DataFrame, float)

5.5.4 windpowerlib.wind_turbine.load_turbine_data_from_oedb

windpowerlib.wind_turbine.**load_turbine_data_from_oedb**()

Loads turbine data from the OpenEnergy Database (oedb).

Returns **turbine_data** – Contains turbine data of different turbines such as 'manufacturer', 'turbine_type', nominal power ('installed_capacity_kw').

Return type pd.DataFrame

5.5.5 windpowerlib.wind_turbine.get_turbine_types

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

Get the names of all possible wind turbine types for which the power coefficient curve or power curve is provided in the OpenEnergy Data Base (oedb).

Parameters **print_out** (*boolean*) – 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.

Returns **curves_df** – 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 pd.DataFrame

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
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
```

(continues on next page)

(continued from previous page)

```
has_cp_curve              True
Name: 1, dtype: object
```

5.6 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.mean_hub_height()	Calculates the mean hub height of the wind farm.
wind_farm.WindFarm.get_installed_power()	Calculates the installed power of the wind farm.
wind_farm.WindFarm.assign_power_curve([...])	Calculates the power curve of a wind farm.

5.6.1 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. Assigns the hub height to the wind farm object.

Returns

Return type self

Notes

The following equation is used[1]:

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

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

References

5.6.2 windpowerlib.wind_farm.WindFarm.get_installed_power

WindFarm.**get_installed_power**()

Calculates the installed power of the wind farm.

The installed power of wind farms is necessary when a *WindTurbineCluster* object is used and it's power weighed mean hub height is calculated with `mean_hub_height()`.

Returns Installed power of the wind farm.

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

Return type float

5.6.3 windpowerlib.wind_farm.WindFarm.assign_power_curve

WindFarm.**assign_power_curve**(*wake_losses_model='power_efficiency_curve'*, *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 wind farm object.

Parameters

- **wake_losses_model** (*string*) – Defines the method for taking wake losses within the farm into consideration. Options: 'power_efficiency_curve', 'constant_efficiency' or None. Default: 'power_efficiency_curve'.
- **smoothing** (*boolean*) – 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** (*string*) – Method for calculating the standard deviation for the Gauss distribution. Options: 'turbulence_intensity', 'Staffell_Pfenninger'. Default: 'turbulence_intensity'.
- **smoothing_order** (*string*) – 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.

Other Parameters **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

Return type self

5.7 Wind turbine cluster calculations

Functions and methods to calculate the mean hub height, installed 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.mean_hub_height()	Calculates the mean hub height of the wind turbine cluster.
wind_turbine_cluster.WindTurbineCluster.get_installed_power()	Calculates the installed power of a wind turbine cluster.
wind_turbine_cluster.WindTurbineCluster.assign_power_curve([...])	Calculates the power curve of a wind turbine cluster.

5.7.1 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. Assigns the hub height to the turbine cluster object.

Returns

Return type self

Notes

The following equation is used[1]:

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

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

5.7.2 windpowerlib.wind_turbine_cluster.WindTurbineCluster.get_installed_power

WindTurbineCluster.**get_installed_power**()

Calculates the installed power of a wind turbine cluster.

Returns Installed power of the wind turbine cluster.

Return type float

5.7.3 windpowerlib.wind_turbine_cluster.WindTurbineCluster.assign_power_curve

WindTurbineCluster.**assign_power_curve**(*wake_losses_model='power_efficiency_curve'*, *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.

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

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) 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** (*string*) – Defines the method for taking wake losses within the farm into consideration. Options: 'power_efficiency_curve', 'constant_efficiency' or None. Default: 'power_efficiency_curve'.
- **smoothing** (*boolean*) – 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** (*string*) – Method for calculating the standard deviation for the Gauss distribution. Options: 'turbulence_intensity', 'Staffell_Pfenninger'. Default: 'turbulence_intensity'.
- **smoothing_order** (*string*) – 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.

Other Parameters **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

Return type self

5.8 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.

5.8.1 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/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 if the parameter *density_corr* is False[1],[2]:

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

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

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

References

5.8.2 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*.

[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

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

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

Return type pandas.Series or numpy.array

Notes

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

5.8.3 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]

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

5.9 Alteration of power curves

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

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

5.9.1 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*)

Smooths the input power curve values 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*.

[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

- **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** (*string*) – 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_df** – 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 pd.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 $v_{\mathrm{i+1}}$

Power curve smoothing is applied to take account for 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]:

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

with: TI: turbulence intensity

'Staffell_Pfenninger'[4]:

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

[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

References

5.9.2 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*, *wake_losses_model='power_efficiency_curve'*)

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 pd.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.
- **wake_losses_model** (*String*) – Defines the method for taking wake losses within the farm into consideration. Options: 'power_efficiency_curve', 'constant_efficiency'. Default: 'power_efficiency_curve'.

Returns **power_curve_df** – 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 pd.DataFrame

5.10 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*.

5.10.1 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

[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

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

Returns **reduced_wind_speed** – *wind_speed* reduced by wind efficiency curve.

Return type pd.Series or np.array

References

5.10.2 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*) – 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 **efficiency_curve** – 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 pd.DataFrame

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()
```

[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

5.11 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.

5.11.1 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.

Other Parameters

- **roughness_length** (*Float, optional.*) – Roughness length.
- **turbulence_intensity** (*Float, optional.*) – Turbulence intensity.

Returns

Return type self

Examples

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

Methods of the ModelChain object.

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

5.11.2 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_hub** – Temperature of air in K at hub height.

Return type pandas.Series or numpy.array

Notes

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

5.11.3 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_hub** – Density of air in kg/m^3 at hub height.

Return type pandas.Series or numpy.array

Notes

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

5.11.4 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_hub** – Wind speed in m/s at hub height.

Return type pandas.Series or numpy.array

Notes

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

5.11.5 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

5.12 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.

5.12.1 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',
...                                              '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.

Continued on next page

Table 21 – continued from previous page

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.

5.12.2 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 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 documentation of *TurbineClusterModelChain.run_model()* for an example on how to create the weather_df DataFrame.

Returns

Return type self

5.12.3 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_hub** – Temperature of air in K at hub height.

Return type pandas.Series or numpy.array

Notes

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

5.12.4 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_hub** – Density of air in kg/m^3 at hub height.

Return type pandas.Series or numpy.array

Notes

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

5.12.5 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_hub** – Wind speed in m/s at hub height.

Return type pandas.Series or numpy.array

Notes

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

5.12.6 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

5.13 Tools

Additional functions used in the windpowerlib.

tools.linear_interpolation_extrapolation(df, ...)	Linear inter- or extrapolates between the values of a data frame.
tools.logarithmic_interpolation_extrapolation(df, ...)	Logarithmic (...) inter- or extrapolates 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.

5.13.1 windpowerlib.tools.linear_interpolation_extrapolation

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

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

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

Parameters

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

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

Return type pandas.Series

Notes

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

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

Examples

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

5.13.2 windpowerlib.tools.logarithmic_interpolation_extrapolation

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

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

This function can be used for the 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

5.13.3 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

5.13.4 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

5.14 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 two *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.

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

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

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

Return type pandas.DataFrame

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

5.14.2 example.modelchain_example.initialize_wind_turbines

example.modelchain_example.**initialize_wind_turbines**()
Initializes two *WindTurbine* objects.

Function shows three ways to initialize a WindTurbine object. You can either specify your own turbine, as done below for 'my_turbine', or fetch power and/or power coefficient curve data from the OpenEnergy Database (oedb), as done for the 'enercon_e126', or provide your turbine data in csv files as done for 'dummy_turbine' with an example file. Execute `windpowerlib.wind_turbine.get_turbine_types()` to get a table including all wind turbines for which power and/or power coefficient curves are provided.

Returns

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

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

5.14.4 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 data file provided by the windpowerlib.
- **dummy_turbine** (WindTurbine) – WindTurbine object with power coefficient curve from example file.

5.14.5 example.modelchain_example.run_example

example.modelchain_example.**run_example**()
Runs the basic example.

5.15 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.

5.15.1 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. You need to provide at least a name and a the wind farm's wind turbine fleet as done below for 'example_farm'. Optionally you can provide a wind farm efficiency (which can be constant or dependent on the wind speed) and coordinates as done for 'example_farm_2'. In this example the coordinates are not being used as just a single weather data set is provided as example data.

Parameters

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

Returns

Return type Tuple (*WindFarm*, *WindFarm*)

5.15.2 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. In this case the cluster only contains two wind farms.

Parameters

- **example_farm** (WindFarm) – WindFarm object.
- **example_farm_2** (WindFarm) – WindFarm object constant wind farm efficiency and coordinates.

Returns

Return type *WindTurbineCluster*

5.15.3 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** (*pd.DataFrame*) – Contains weather data time series.
- **example_farm** (WindFarm) – WindFarm object.
- **example_cluster** (WindTurbineCluster) – WindTurbineCluster object.

5.15.4 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.
- **example_farm_2** (WindFarm) – WindFarm object constant wind farm efficiency and coordinates.

5.15.5 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

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

W