feedinlib Documentation

oemof developer group

Dec 04, 2019

Contents

Contents:

CHAPTER 1

Getting started

The feedinlib is designed to calculate feed-in time series of photovoltaic and wind power plants. It is part of the oemof group but works as a standalone application.

The feedinlib is ready to use but it definitely has a lot of space for further development, new and improved models and nice features.

1.1 Introduction

So far the feedinlib provides interfaces to download *open_FRED* and ERA5 weather data. *open_FRED* is a local reanalysis weather data set that provides weather data for Germany (and bounding box). *ERA5* is a global reanalysis weather data set that provides weather data for the whole world. The weather data can be used to calculate the electrical output of PV and wind power plants. At the moment the feedinlib provides interfaces to the pvlib and the windpowerlib. Furthermore, technical parameters for many PV modules and inverters, as well as wind turbines, are made available and can be easily used for calculations.

1.2 Installation

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

```
pip install feedinlib
```

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

We highly recommend to use virtual environments. Please see the installation page of the oemof documentation for complete instructions on how to install python and a virtual environment on your operating system.

1.3 Examples and basic usage

The basic usage of the feedinlib is shown in the *Examples* section. The examples are provided as jupyter notebooks that you can download here:

- `ERA5 weather data example`
- `open_FRED weather data example`
- `pvlib model example`
- `windpowerlib model example`

Furthermore, you have to install the feedinlib with additional packages needed to run the notebooks, e.g. *jupyter*.

```
pip install feedinlib[examples]
```

To launch jupyter notebook type `jupyter notebook` in the terminal. This will open a browser window. Navigate to the directory containing the notebook(s) to open it. See the jupyter notebook quick start guide for more information on how to run jupyter notebooks.

1.4 Contributing

We are warmly welcoming all who want to contribute to the feedinlib. If you are interested do not hesitate to contact us via github.

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

How to create a pull request:

- Fork the feedinlib repository to your own github account.
- Create a local clone of your fork and install the cloned repository using pip with -e option:

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

- 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 feedinlib directory:

```
pytest
```

1.5 Citing the feedinlib

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

1.6 License

CHAPTER 2

Examples

2.1 Example for open_FRED weather data download

This example shows you how to download open_FRED weather data from the OpenEnergy DataBase and store it locally. Furthermore, it shows how to convert the weather data to the format needed by the pvlib and windpowerlib.

[1]:
```
# imports
from shapely.geometry import Point, Polygon

from feedinlib.db import Weather
from feedinlib.db import defaultdb
```

2.1.1 Download data for single coordinate

[2]:
```
location = Point(13.5, 52.4)
```

Besides a location you have to specify a time period for which you would like to download the data as well as the weather variables you need. The feedinlib provides predefined sets of variables that are needed to use the pvlib and windpowerlib. These can be applied by setting the `variables` parameter to "pvlib" or "windpowerlib", as shown below.

[3]:
```
# download data for January 2015 (end date will not be included in the
# time period for which data is downloaded)
start_date, end_date = '2015-01-01', '2015-02-01'
variables = "windpowerlib"
```

As the open_FRED weather dataset provides some variables at different heights, such as wind speed and air pressure, it is possible to define which heights you want to retrieve the data for.

[4]:
```
heights = [80, 100]
```

Now we can retrieve the data:

[5]:
```
open_FRED_weather_windpowerlib = Weather(
    start=start_date, stop=end_date, locations=[location],
    heights=heights,
    variables=variables,
    **defaultdb())
```

2.1.2 Download data for a region

[6]:
```
lat_point_list = [51.936377, 51.936377, 51.744302, 51.744302, 51.936377]
lon_point_list = [12.621739, 13.005414, 13.005414, 12.621739, 12.621739]
region = Polygon(zip(lon_point_list, lat_point_list))
```

In this example we will retrieve weather data needed for pvlib calculations. We can do this by setting `variables` to "pvlib". In this case specifying the heights for which to retrieve the data is not necessary as irradiance data is only available at the surface and 10m wind speed is used per default.

The following code may take a while to execute!

[7]:
```
open_FRED_weather_pvlib = Weather(
    start='2015-06-01', stop='2015-06-05',
    locations=[], regions=[region],
    variables="pvlib",
    **defaultdb())
```

2.1.3 Convert data into pvlib and windpowerlib format

In order to use the weather data for your feed-in calculations using the pvlib and windpowerlib it has to be converted into the required format. This can easily be done as follows.

[8]:
```
# convert to windpowerlib dataframe
windpowerlib_df = open_FRED_weather_windpowerlib.df(location=location, lib=
↪"windpowerlib")
```

[9]:
```
# save dataframe as csv
windpowerlib_df.to_csv('windpowerlib_df.csv')
```

[11]:
```
# select point inside region
point = Point(12.7, 51.9)
# convert to pvlib dataframe
pvlib_df = open_FRED_weather_pvlib.df(location=point, lib="pvlib")
```

[ ]:
```

```

2.2 Example for ERA5 weather data download

This example shows you how to download ERA5 weather data from the Climate Data Store (CDS) and store it locally. Furthermore, it shows how to convert the weather data to the format needed by the pvlib and windpowerlib.

In order to download ERA5 weather data you need an account at the CDS. Furthermore, you need to install the cdsapi package. See here for installation details.

When downloading the data using the API your request gets queued and may take a while to be completed. All actual calls of the data download are therefore commented to avoid unintended download.

- *Download data for single coordinate*
- *Download data for a region*
- *Convert data into pvlib and windpowerlib format*

[1]:
```
from feedinlib import era5
```

2.2.1 Download data for single coordinate

To download data for a single location you have to specify latitude and longitude of the desired location. Data will be retrieved for the nearest weather data point to that location.

[2]:
```
latitude = 52.4
longitude = 13.5
```

Besides a location you have to specify a time period for which you would like to download the data as well as the weather variables you need. The feedinlib provides predefined sets of variables that are needed to use the pvlib and windpowerlib. These can be applied by setting the `variable` parameter to "pvlib" or "windpowerlib", as shown below. If you want to download data for both library you can set `variable` to 'feedinlib'.

[3]:
```
# set start and end date (end date will be included
# in the time period for which data is downloaded)
start_date, end_date = '2015-01-01', '2015-02-01'
# set variable set to download
variable = "windpowerlib"
```

If you want to store the downloaded data you may provide a filename (including path) to save data to.

[4]:
```
target_file = 'ERA5_weather_data.nc'
```

Now we can retrieve the data:

```
# get windpowerlib data for specified location
ds = era5.get_era5_data_from_datespan_and_position(
    variable=variable,
    start_date=start_date, end_date=end_date,
    latitude=latitude, longitude=longitude,
    target_file=target_file)
```

2.2.2 Download data for a region

When wanting to download weather data for a region instead of providing a single value for each latitude and longitude you have to provide latitude and longitude as lists in the following form:

[5]:
```
latitude = [51.0, 53.0]  # [latitude south, latitude north]
longitude = [13.5, 13.8]  # [longitude west, longitude east]
```

[6]:
```
variable = "pvlib"
```

```
# get pvlib data for specified area
ds = era5.get_era5_data_from_datespan_and_position(
    variable=variable,
    start_date=start_date, end_date=end_date,
    latitude=latitude, longitude=longitude,
    target_file=target_file)
```

If you want weather data for the whole world, you may leave latitude and longitude unspecified.

```
# get feedinlib data (includes pvlib and windpowerlib data)
# for the whole world
ds = era5.get_era5_data_from_datespan_and_position(
    variable="feedinlib",
    start_date=start_date, end_date=end_date,
    target_file=target_file)
```

2.2.3 Convert data into pvlib and windpowerlib format

In order to use the weather data for your feed-in calculations using the pvlib and windpowerlib it has to be converted into the required format. This can easily be done as follows.

[7]:
```
# filename (including path) downloaded ERA5 data was saved to
era5_netcdf_filename = 'ERA5_weather_data.nc'
```

```
# get weather data in windpowerlib format for all locations in
# netcdf file
windpowerlib_df = era5.weather_df_from_era5(
    era5_netcdf_filename=filename,
    lib='windpowerlib')
```

If the netcdf contains more than one location the `windpowerlib_df` will contain all of those locations and the index of the dataframe will be a multiindex with levels (time, latitude, longitude). To use it for windpowerlib calculations you need to select a **single location** first. To directly obtain data for a single location you need to specify it using the `area` parameter:

```
# get weather data in windpowerlib format for single location
# (weather data for nearest weather data point to specified
# location is returned)
area = [13.5, 52.4]
windpowerlib_df = era5.weather_df_from_era5(
    era5_netcdf_filename=filename,
    lib='windpowerlib', area=area)
```

You may also specify an area for which to retrieve weather data. Again, keep in mind that in that case the index of the returned dataframe will be a multiindex with levels (time, latitude, longitude) and cannot be directly used for windpowerlib or pvlib calculations.

[8]:
```
# specify rectangular area
area = [(13.5, 13.8), (51.0, 53.0)]
```

[9]:
```
# specify area giving a Polygon
from shapely.geometry import Polygon
lat_point_list = [51.936377, 51.936377, 51.744302, 51.744302, 51.936377]
lon_point_list = [12.621739, 13.005414, 13.005414, 12.621739, 12.621739]
area = Polygon(zip(lon_point_list, lat_point_list))
```

Furthermore, it is possible to specify a start and end date to retrieve data for. They must be provided as something that can be converted to a timestamp, i.e. '2013-07-02'.

```
# get weather data in pvlib format starting January 15th
start = '2015-01-15'
windpowerlib_df = era5.weather_df_from_era5(
    era5_netcdf_filename=filename,
    lib='pvlib', area=area, start=start)
```

[ ]:

2.3 Example for using the Pvlib model

The `Pvlib` model can be used to determine the feed-in of a photovoltaic module using the pvlib. The pvlib is a python library for simulating the performance of photovoltaic energy systems. For more information check out the documentation of the pvlib.

The following example shows you how to use the `Pvlib` model.

- *Set up Photovoltaic object*
- *Get weather data*
- *Calculate feed-in*

2.3.1 Set up Photovoltaic object

To calculate the feed-in using the `Pvlib` model you have to set up a `Photovoltaic` object. You can import it as follows:

[1]:
```
from feedinlib import Photovoltaic
```

To set up a Photovoltaic system you have to provide all PV system parameters required by the `PVlib` model. The required parameters can be looked up in the model's documentation. For the `Pvlib` model these are the **azimuth** and **tilt** of the module as well as the **albedo or surface type**. Furthermore, the **name of the module and inverter** are needed to obtain technical parameters from the provided module and inverter databases. For an overview of the provided modules and inverters you can use the function `get_power_plant_data()`.

[2]:
```
from feedinlib import get_power_plant_data
```

[3]:
```
# get modules
module_df = get_power_plant_data(dataset='sandiamod')
# print the first four modules
module_df.iloc[:, 1:5]
```

[3]:
```
                                Advent_Solar_Ventura_210___2008_  \
Vintage                                                     2008
Area                                                       1.646
Material                                                  mc-Si
Cells_in_Series                                              60
Parallel_Strings                                              1
Isco                                                       8.34
Voco                                                      35.31
Impo                                                       7.49
```

(continues on next page)

(continued from previous page)

```
Vmpo                                                    27.61
Aisc                                                  0.00077
Aimp                                                 -0.00015
C0                                                      0.937
C1                                                      0.063
Bvoco                                                  -0.133
Mbvoc                                                       0
Bvmpo                                                  -0.135
Mbvmp                                                       0
N                                                       1.495
C2                                                     0.0182
C3                                                    -10.758
A0                                                     0.9067
A1                                                    0.09573
A2                                                    -0.0266
A3                                                    0.00343
A4                                                 -0.0001794
B0                                                          1
B1                                                  -0.002438
B2                                                    0.00031
B3                                                 -1.246e-05
B4                                                   2.11e-07
B5                                                  -1.36e-09
DTC                                                         3
FD                                                          1
A                                                       -3.45
B                                                      -0.077
C4                                                      0.972
C5                                                      0.028
IXO                                                      8.25
IXXO                                                      5.2
C6                                                      1.067
C7                                                     -0.067
Notes             Source: Sandia National Laboratories Updated 9...

                                  Advent_Solar_Ventura_215___2009_  \
Vintage                                                     2009
Area                                                       1.646
Material                                                   mc-Si
Cells_in_Series                                               60
Parallel_Strings                                               1
Isco                                                        8.49
Voco                                                       35.92
Impo                                                        7.74
Vmpo                                                       27.92
Aisc                                                     0.00082
Aimp                                                    -0.00013
C0                                                         1.015
C1                                                        -0.015
Bvoco                                                     -0.135
Mbvoc                                                          0
Bvmpo                                                     -0.136
Mbvmp                                                          0
N                                                          1.373
C2                                                        0.0036
C3                                                       -7.2509
A0                                                        0.9323
```

(continues on next page)

(continued from previous page)

```
A1                                                      0.06526
A2                                                     -0.01567
A3                                                      0.00193
A4                                                    -9.81e-05
B0                                                            1
B1                                                     -0.002438
B2                                                      0.00031
B3                                                    -1.246e-05
B4                                                     2.11e-07
B5                                                    -1.36e-09
DTC                                                           3
FD                                                            1
A                                                         -3.47
B                                                        -0.087
C4                                                        0.989
C5                                                        0.012
IXO                                                        8.49
IXXO                                                       5.45
C6                                                        1.137
C7                                                       -0.137
Notes             Source: Sandia National Laboratories Updated 9...

                                           Aleo_S03_160__2007__E__  \
Vintage                                                2007 (E)
Area                                                       1.28
Material                                                   c-Si
Cells_in_Series                                              72
Parallel_Strings                                              1
Isco                                                        5.1
Voco                                                       43.5
Impo                                                       4.55
Vmpo                                                       35.6
Aisc                                                     0.0003
Aimp                                                    -0.00025
C0                                                         0.99
C1                                                         0.01
Bvoco                                                    -0.152
Mbvoc                                                         0
Bvmpo                                                    -0.158
Mbvmp                                                         0
N                                                          1.25
C2                                                        -0.15
C3                                                        -8.96
A0                                                        0.938
A1                                                      0.05422
A2                                                    -0.009903
A3                                                    0.0007297
A4                                                    -1.907e-05
B0                                                            1
B1                                                     -0.002438
B2                                                     0.0003103
B3                                                    -1.246e-05
B4                                                     2.11e-07
B5                                                    -1.36e-09
DTC                                                           3
FD                                                            1
A                                                         -3.56
```

(continues on next page)

(continued from previous page)

```
B                                                        -0.075
C4                                                        0.995
C5                                                        0.005
IXO                                                        5.04
IXXO                                                       3.16
C6                                                         1.15
C7                                                        -0.15
Notes             Source: Sandia National Laboratories Updated 9...

                                              Aleo_S03_165__2007__E__
Vintage                                                2007 (E)
Area                                                       1.28
Material                                                   c-Si
Cells_in_Series                                              72
Parallel_Strings                                              1
Isco                                                        5.2
Voco                                                       43.6
Impo                                                       4.65
Vmpo                                                       35.8
Aisc                                                     0.0003
Aimp                                                    -0.00025
C0                                                         0.99
C1                                                         0.01
Bvoco                                                    -0.152
Mbvoc                                                         0
Bvmpo                                                    -0.158
Mbvmp                                                         0
N                                                          1.25
C2                                                        -0.15
C3                                                        -8.96
A0                                                        0.938
A1                                                      0.05422
A2                                                    -0.009903
A3                                                    0.0007297
A4                                                   -1.907e-05
B0                                                            1
B1                                                    -0.002438
B2                                                    0.0003103
B3                                                   -1.246e-05
B4                                                    2.11e-07
B5                                                   -1.36e-09
DTC                                                           3
FD                                                            1
A                                                         -3.56
B                                                        -0.075
C4                                                        0.995
C5                                                        0.005
IXO                                                        5.14
IXXO                                                       3.25
C6                                                         1.15
C7                                                        -0.15
Notes             Source: Sandia National Laboratories Updated 9...
```

[4]:
```
# get inverter data
inverter_df = get_power_plant_data(dataset='cecinverter')
# print the first four inverters
inverter_df.iloc[:, 1:5]
```

[4]:
```
           ABB__MICRO_0_25_I_OUTD_US_208__208V__208V__CEC_2018_  \
Vac                                                   208.000000
Paco                                                  250.000000
Pdco                                                  259.589000
Vdco                                                   40.000000
Pso                                                     2.089610
C0                                                     -0.000041
C1                                                     -0.000091
C2                                                      0.000494
C3                                                     -0.013171
Pnt                                                     0.020000
Vdcmax                                                 50.000000
Idcmax                                                  6.489710
Mppt_low                                               30.000000
Mppt_high                                              50.000000

           ABB__MICRO_0_25_I_OUTD_US_240_240V__CEC_2014_  \
Vac                                               240.000000
Paco                                              250.000000
Pdco                                              259.552697
Vdco                                               39.982246
Pso                                                 1.931194
C0                                                 -0.000027
C1                                                 -0.000158
C2                                                  0.001480
C3                                                 -0.034600
Pnt                                                 0.050000
Vdcmax                                             65.000000
Idcmax                                             10.000000
Mppt_low                                           20.000000
Mppt_high                                          50.000000

           ABB__MICRO_0_25_I_OUTD_US_240__240V__240V__CEC_2018_  \
Vac                                                   240.000000
Paco                                                  250.000000
Pdco                                                  259.492000
Vdco                                                   40.000000
Pso                                                     2.240410
C0                                                     -0.000039
C1                                                     -0.000132
C2                                                      0.002418
C3                                                     -0.014926
Pnt                                                     0.050000
Vdcmax                                                 50.000000
Idcmax                                                  6.487300
Mppt_low                                               30.000000
Mppt_high                                              50.000000

           ABB__MICRO_0_3_I_OUTD_US_208_208V__CEC_2014_
Vac                                           208.000000
Paco                                          300.000000
Pdco                                          311.714554
Vdco                                           40.227111
Pso                                             1.971053
C0                                             -0.000036
C1                                             -0.000256
C2                                             -0.000833
```

(continues on next page)

(continued from previous page)

```
C3                                          -0.039100
Pnt                                          0.020000
Vdcmax                                      65.000000
Idcmax                                      10.000000
Mppt_low                                    30.000000
Mppt_high                                   50.000000
```

Now you can set up a PV system to calculate feed-in for, using for example the first module and converter in the databases:

[5]:
```
system_data = {
    'module_name': 'Advent_Solar_Ventura_210___2008_',  # module name as in database
    'inverter_name': 'ABB__MICRO_0_25_I_OUTD_US_208__208V__208V__CEC_2018_',  #␣
↪inverter name as in database
    'azimuth': 180,
    'tilt': 30,
    'albedo': 0.2}
pv_system = Photovoltaic(**system_data)
```

Optional power plant parameters

Besides the required PV system parameters you can provide optional parameters such as the number of modules per string, etc. Optional PV system parameters are specific to the used model and how to find out about the possible optional parameters is documented in the model's `feedin` method under `power_plant_parameters`. In case of the `Pvlib` model see here.

[6]:
```
system_data['modules_per_string'] = 2
pv_system_with_optional_parameters = Photovoltaic(**system_data)
```

2.3.2 Get weather data

Besides setting up your PV system you have to provide weather data the feed-in is calculated with. This example uses open_FRED weather data. For more information on the data and download see the *load_open_fred_weather_data Notebook*.

[7]:
```
from feedinlib.db import Weather
from feedinlib.db import defaultdb
from shapely.geometry import Point
```

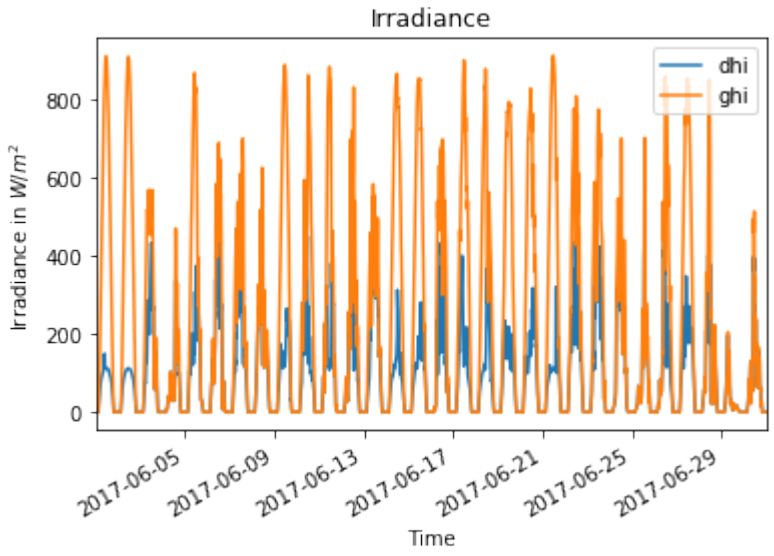
[8]:
```
# specify latitude and longitude of PV system location
lat = 52.4
lon = 13.5
location = Point(lon, lat)
```

[9]:
```
# download weather data for June 2017
open_FRED_weather_data = Weather(
    start='2017-06-01', stop='2017-07-01',
    locations=[location],
    variables="pvlib",
    **defaultdb())
```

[10]:
```
# get weather data in pvlib format
weather_df = open_FRED_weather_data.df(location=location, lib="pvlib")
```

```
/home/birgit/virtualenvs/feedinlib/lib/python3.6/site-packages/pandas/core/sorting.py:
↪257: FutureWarning: Converting timezone-aware DatetimeArray to timezone-naive␣
↪ndarray with 'datetime64[ns]' dtype. In the future, this will return an ndarray␣
↪with 'object' dtype where each element is a 'pandas.Timestamp' with the correct 'tz
↪'.
        To accept the future behavior, pass 'dtype=object'.
        To keep the old behavior, pass 'dtype="datetime64[ns]"'.
  items = np.asanyarray(items)
```

[11]:
```
# plot irradiance
import matplotlib.pyplot as plt
%matplotlib inline
weather_df.loc[:, ['dhi', 'ghi']].plot(title='Irradiance')
plt.xlabel('Time')
plt.ylabel('Irradiance in $W/m^2$');
```

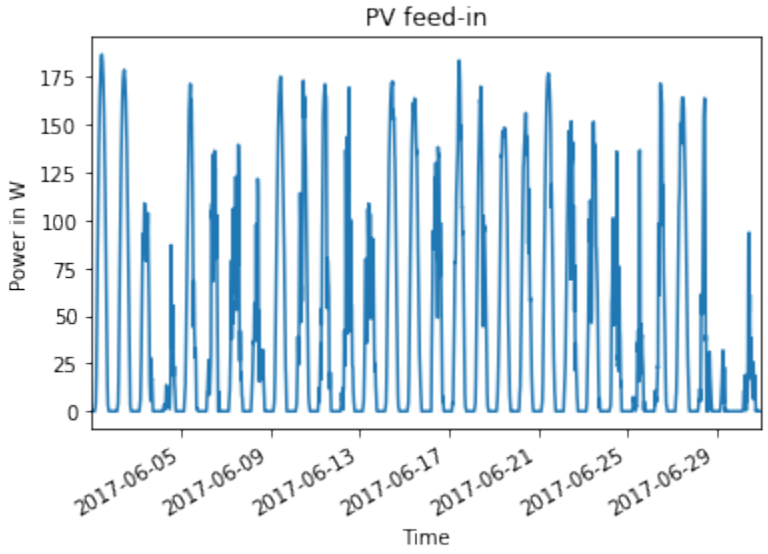


2.3.3 Calculate feed-in

The feed-in can be calculated by calling the `Photovoltaic`'s `feedin` method with the weather data. For the `Pvlib` model you also have to provide the location of the PV system.

[12]:
```
feedin = pv_system.feedin(
    weather=weather_df,
    location=(lat, lon))
```

[13]:
```
# plot calculated feed-in
import matplotlib.pyplot as plt
%matplotlib inline
feedin.plot(title='PV feed-in')
plt.xlabel('Time')
plt.ylabel('Power in W');
```



Scaled feed-in

The PV feed-in can also be automatically scaled by the PV system's area or peak power. The following example shows how to scale feed-in by area.

[14]:
```
feedin_scaled = pv_system.feedin(
    weather=weather_df,
    location=(lat, lon),
    scaling='area')
```

To scale by the peak power use `scaling=peak_power`.

The PV system area and peak power can be retrieved as follows:

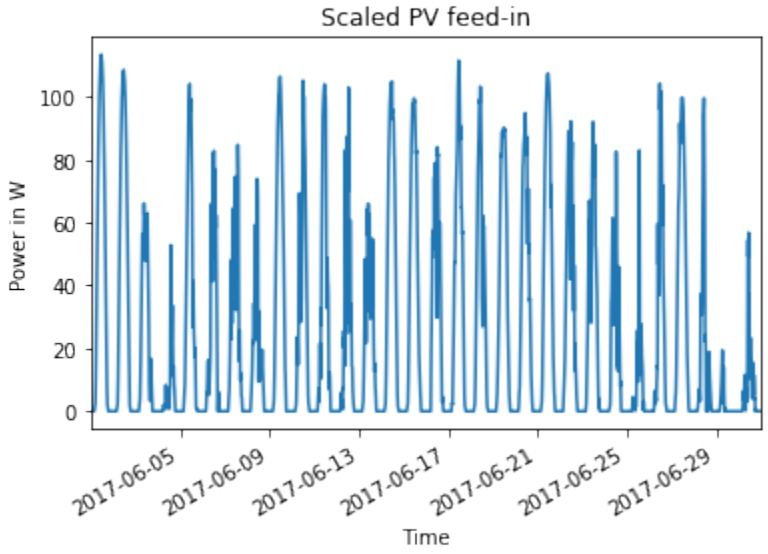
[15]:
```
pv_system.area
```

[15]:
```
1.646
```

[16]:
```
pv_system.peak_power
```

[16]:
```
206.7989
```

[17]:
```
# plot calculated feed-in
import matplotlib.pyplot as plt
%matplotlib inline
feedin_scaled.plot(title='Scaled PV feed-in')
plt.xlabel('Time')
plt.ylabel('Power in W');
```

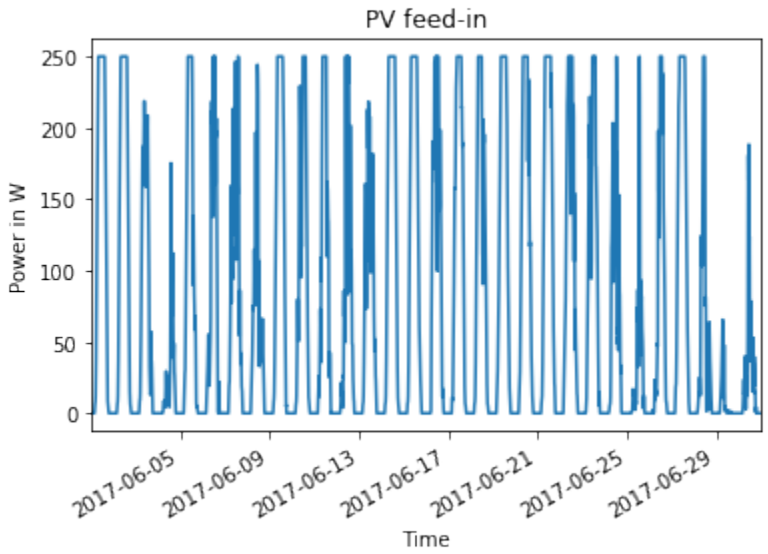


Feed-in for PV system with optional parameters

In the following example the feed-in is calculated for the PV system with optional system parameters (with 2 modules per string, instead of 1, which is the default). It was chosen to demonstrate the importantance of choosing a suitable converter.

[18]:
```
feedin_ac = pv_system_with_optional_parameters.feedin(
    weather=weather_df,
    location=(lat, lon))
```

[19]:
```
# plot calculated feed-in
import matplotlib.pyplot as plt
%matplotlib inline
feedin_ac.plot(title='PV feed-in')
plt.xlabel('Time')
plt.ylabel('Power in W');
```



As the above plot shows the feed-in is cut off at 250 W. That is because it is limited by the inverter. So while the area is as expected two times greater as for the PV system without optional parameters, the peak power is only around 1.2 times higher.

[20]:
```
pv_system_with_optional_parameters.peak_power / pv_system.peak_power
```

[20]:
```
1.208903915833208
```

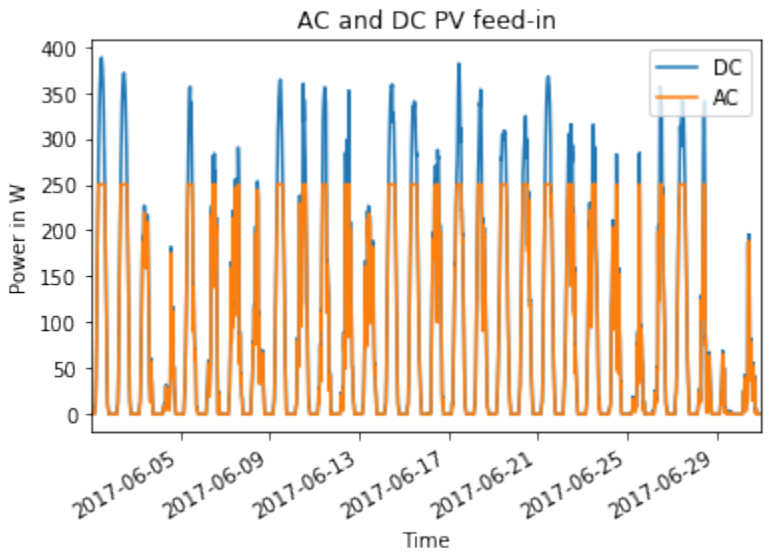
[21]:
```
pv_system_with_optional_parameters.area / pv_system.area
```

[21]:
```
2.0
```

If you are only interested in the modules power output without the inverter losses you can have the `Pvlib` model return the DC feed-in. This is done as follows:

[22]:
```
feedin_dc = pv_system_with_optional_parameters.feedin(
    weather=weather_df,
    location=(lat, lon),
    mode='dc')
```

[23]:
```
# plot calculated feed-in
import matplotlib.pyplot as plt
%matplotlib inline
feedin_dc.plot(label='DC', title='AC and DC PV feed-in', legend=True)
feedin_ac.plot(label='AC', legend=True)
plt.xlabel('Time')
plt.ylabel('Power in W');
```

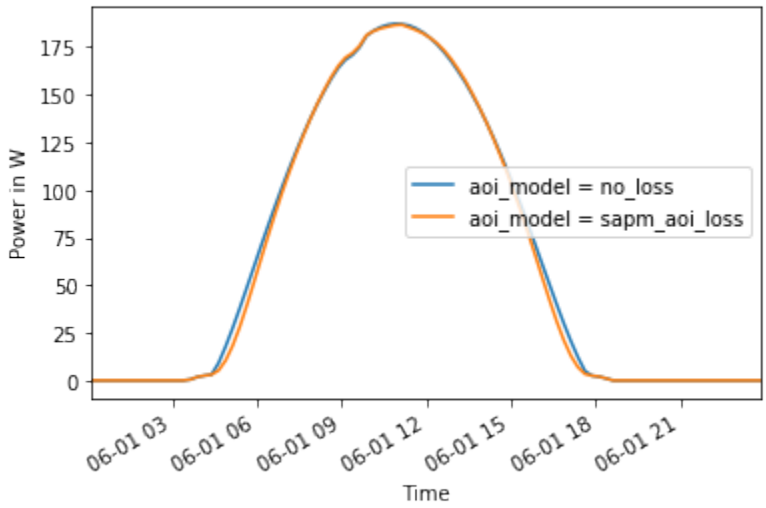


Feed-in with optional model parameters

In order to change the default calculation configurations of the `Pvlib` model to e.g. choose a different model to calculate losses or the solar position you can pass further parameters to the `feedin` method. An overview of which further parameters may be provided is documented under the feedin method's kwargs.

[24]:
```
feedin_no_loss = pv_system.feedin(
    weather=weather_df,
    location=(lat, lon),
    aoi_model='no_loss')
```

[25]:
```
# plot calculated feed-in
import matplotlib.pyplot as plt
%matplotlib inline
feedin_no_loss.iloc[0:96].plot(label='aoi_model = no_loss', legend=True)
feedin.iloc[0:96].plot(label='aoi_model = sapm_aoi_loss', legend=True)
plt.xlabel('Time')
plt.ylabel('Power in W');
```



[ ]:

CHAPTER 3

What's New

These are new features and improvements of note in each release

Releases

3.1 v0.0.11 (November 22, 2016)

3.1.1 New features

- Using model of windpowerlib instead of internal model. This will be the future of the feedinlib.

3.1.2 Bug fixes

- removed 'vernetzen'-server because it is down

3.1.3 Contributors

- Uwe Krien

3.2 v0.0.10 (November 18, 2016)

3.2.1 Other changes

Move wind power calculations to windpowerlib Allow installation of windpowerlib for python versions >3.4 Import requests package instead of urllib5

3.2.2 Contributors

- Uwe Krien
- Stephen Bosch
- Birgit Schachler

3.3 v0.0.9 (August 23, 2016)

3.3.1 Bug fixes

- Adapt API due to changes in the pvlib
- Avoid pandas future warning running the pv model

3.3.2 Contributors

- Uwe Krien

3.4 v0.0.8 (Mai 2, 2016)

3.4.1 New features

- add a geometry attribute for shapely.geometry objects to the weather class
- add lookup table for the sandia pv modules

3.4.2 Documentation

- add link to the developer rules of oemof

3.4.3 Bug fixes

- Adapt url to sandia's module library

3.4.4 Contributors

- Uwe Krien

3.5 v0.0.7 (October 20, 2015)

3.5.1 New features

- add a weather class to define the structure of the weather data input
- add example file to pass your own model class to the feedinlib

3.5.2 Documentation

- correct some typos
- some distribtions are clearer now
- describe the used units

3.5.3 Testing

- add more doctests
- removed obsolete tests

3.5.4 Bug fixes

- does not overwrite class attributes (issue 7)

3.5.5 Other changes

- rename classes to more describing names
- initialisation of a power plant changed (see README for details)

3.5.6 Contributors

- Uwe Krien
- Stephan Günther
- Cord Kaldemeyer

CHAPTER 4

API

4.1 Power plant classes

Power plant classes for specific weather dependent renewable energy resources.

feedinlib.powerplants.Photovoltaic([model])	Class to define a standard set of PV system attributes.
feedinlib.powerplants.WindPowerPlant([model])	Class to define a standard set of wind power plant attributes.

4.1.1 feedinlib.powerplants.Photovoltaic

class feedinlib.powerplants.**Photovoltaic**(*model=<class 'feedinlib.models.Pvlib'>*, ***attributes*)

Class to define a standard set of PV system attributes.

The Photovoltaic class serves as a data container for PV system attributes. Actual calculation of feed-in provided by the PV system is done by the chosen PV model. So far there is only one PV model, *Pvlib*.

Parameters

- **model** (A subclass or instance of subclass of *PhotovoltaicModelBase*) – The *model* parameter defines the feed-in model used to calculate the PV system feed-in. It defaults to *Pvlib* which is currently the only implemented photovoltaic model.

  model is used as the *model* parameter for *Base*.

- ****attributes** – PV system parameters. See *power_plant_parameters* parameter in respective model's *feedin()* method for further information on the model's required and optional plant parameters.

  As the *Pvlib* model is currently the only implemented photovoltaic model see *power_plant_parameters* parameter *feedin()* for further information.

Methods

__init__([model])	
feedin(weather[, scaling])	Calculates PV system feed-in in Watt.

Attributes

area	Area of PV system in m^2.
peak_power	Peak power of PV system in Watt.
required	The power plant parameters the specified model requires.

__init__ (*model=<class 'feedinlib.models.Pvlib'>*, ***attributes*)

feedin (*weather*, *scaling=None*, ***kwargs*)
Calculates PV system feed-in in Watt.

The feed-in can further be scaled by PV system area or peak power using the *scaling* parameter.

This method delegates the actual computation to the model's *feedin()* method while giving you the opportunity to override some of the inputs used to calculate the feed-in. As the *Pvlib* model is currently the only implemented photovoltaic model see *feedin()* for further information on feed-in calculation.

If the respective model does calculate AC and DC feed-in, AC feed-in is returned by default. See the model's *feedin()* method for information on how to overwrite this default behaviour.

Parameters

- **weather** – Weather data to calculate feed-in. Check the *weather* parameter of the respective model's *feedin()* method for required weather data parameters and format.
- **scaling** (*str*) – Specifies what feed-in is scaled by. Possible options are 'peak_power' and 'area'. Defaults to None in which case feed-in is not scaled.
- ****kwargs** – Keyword arguments for respective model's feed-in calculation. Check the keyword arguments of the model's *feedin()* method for further information.

Returns Series with PV system feed-in in Watt.

Return type pandas.Series

area
Area of PV system in m^2.

See `pv_system_area` attribute of your chosen model for further information on how the area is calculated.

peak_power
Peak power of PV system in Watt.

See `pv_system_peak_power` attribute of your chosen model for further information and specifications on how the peak power is calculated.

required
The power plant parameters the specified model requires.

Check the model's `power_plant_requires` attribute for further information.

4.1.2 feedinlib.powerplants.WindPowerPlant

class feedinlib.powerplants.**WindPowerPlant**(*model=<class 'feedinlib.models.WindpowerlibTurbine'>*, ***attributes*)

Class to define a standard set of wind power plant attributes.

The WindPowerPlant class serves as a data container for wind power plant attributes. Actual calculation of feed-in provided by the wind power plant is done by the chosen wind power model. So far there are two wind power models, *WindpowerlibTurbine* and *WindpowerlibTurbineCluster*. The *WindpowerlibTurbine* model should be used for single wind turbines, whereas the *WindpowerlibTurbineCluster* model can be used for wind farm and wind turbine cluster calculations.

Parameters

- **model** (A subclass or instance of subclass of *feedinlib.models.WindpowerModelBase*) – The *model* parameter defines the feed-in model used to calculate the wind power plant feed-in. It defaults to *WindpowerlibTurbine*.

  model is used as the *model* parameter for *Base*.

- ****attributes** – Wind power plant parameters. See *power_plant_parameters* parameter in respective model's *feedin()* method for further information on the model's required and optional plant parameters.

Methods

__init__([model])	
feedin(weather[, scaling])	Calculates wind power plant feed-in in Watt.

Attributes

nominal_power	Nominal power of wind power plant in Watt.
required	The power plant parameters the specified model requires.

__init__(*model=<class 'feedinlib.models.WindpowerlibTurbine'>*, ***attributes*)

feedin(*weather*, *scaling=None*, ***kwargs*)

Calculates wind power plant feed-in in Watt.

The feed-in can further be scaled by the nominal power of the wind power plant using the *scaling* parameter.

This method delegates the actual computation to the model's meth:*feedin* method while giving you the opportunity to override some of the inputs used to calculate the feed-in. See model's *feedin()* method for further information on feed-in calculation.

Parameters

- **weather** – Weather data to calculate feed-in. Check the *weather* parameter of the respective model's *feedin()* method for required weather data parameters and format.
- **scaling** (*str*) – Specifies what feed-in is scaled by. Possible option is 'nominal_power'. Defaults to None in which case feed-in is not scaled.

- ****kwargs** – Keyword arguments for respective model's feed-in calculation. Check the keyword arguments of the model's *feedin()* method for further information.

Returns Series with wind power plant feed-in in Watt.

Return type pandas.Series

nominal_power
Nominal power of wind power plant in Watt.

See *nominal_power* attribute of your chosen model for further information on how the nominal power is derived.

required
The power plant parameters the specified model requires.

Check the model's `power_plant_requires` attribute for further information.

4.2 Feed-in models

Feed-in models take in power plant and weather data to calculate power plant feed-in. So far models using the python libraries pvlib and windpowerlib to calculate photovoltaic and wind power feed-in, respectively, have been implemented.

feedinlib.models.Pvlib(**kwargs)	Model to determine the feed-in of a photovoltaic module using the pvlib.
feedinlib.models.WindpowerlibTurbine(**kwargs)	Model to determine the feed-in of a wind turbine using the windpowerlib.
feedinlib.models.WindpowerlibTurbineCluster(...)	Model to determine the feed-in of a wind turbine cluster using the windpowerlib.

4.2.1 feedinlib.models.Pvlib

class feedinlib.models.**Pvlib**(***kwargs*)
Model to determine the feed-in of a photovoltaic module using the pvlib.

The pvlib[1] is a python library for simulating the performance of photovoltaic energy systems. For more information about the photovoltaic model check the documentation of the pvlib[2].

Notes

In order to use this model various power plant and model parameters have to be provided. See *power_plant_requires* as well as *requires* for further information. Furthermore, the weather data used to calculate the feed-in has to have a certain format. See *feedin()* for further information.

References

See also:

Base PhotovoltaicModelBase

[1] pvlib on github
[2] pvlib documentation

Methods

__init__(**kwargs)		
feedin(weather, **kwargs)	power_plant_parameters,	Calculates power plant feed-in in Watt.
instantiate_module(**kwargs)		Instantiates a pvlib.PVSystem object.

Attributes

power_plant_requires	The power plant parameters this model requires to calculate a feed-in.
pv_system_area	Area of PV system in m^2.
pv_system_peak_power	Peak power of PV system in Watt.
requires	The parameters this model requires to calculate a feed-in.

__init__(***kwargs*)

feedin(*weather*, *power_plant_parameters*, ***kwargs*)

Calculates power plant feed-in in Watt.

This function uses the pvlib.ModelChain to calculate the feed-in for the given weather time series and PV system. By default the AC feed-in is returned. Set *mode* parameter to 'dc' to retrieve DC feed-in.

Parameters

- **weather** (pandas.DataFrame) – Weather time series used to calculate feed-in. See *weather* parameter in pvlib's Modelchain run_model method for more information on required variables, units, etc.
- **power_plant_parameters** (*dict*) – Dictionary with power plant specifications. Keys of the dictionary are the power plant parameter names, values of the dictionary hold the corresponding value. The dictionary must at least contain the required power plant parameters (see *power_plant_requires*) and may further contain optional power plant parameters (see pvlib.PVSystem).
- **location** (tuple or shapely.Point) – Geo location of the PV system. Can either be provided as a tuple with first entry being the latitude and second entry being the longitude or as a shapely.Point.
- **mode** (*str (optional)*) – Can be used to specify whether AC or DC feed-in is returned. By default *mode* is 'ac'. To retrieve DC feed-in set *mode* to 'dc'.

  mode also influences the peak power of the PV system. See *pv_system_peak_power* for more information.
- ****kwargs** – Further keyword arguments can be used to overwrite pvlib.ModelChain parameters.

Returns Power plant feed-in time series in Watt.

Return type pandas.Series

instantiate_module(***kwargs*)

Instantiates a pvlib.PVSystem object.

Parameters ****kwargs** – See *power_plant_parameters* parameter in *feedin()* for more information.

Returns PV system to calculate feed-in for.

Return type pvlib.PVSystem

power_plant_requires
The power plant parameters this model requires to calculate a feed-in.

The required power plant parameters are:

module_name, *inverter_name*, *azimuth*, *tilt*, *albedo/surface_type*

module_name (str) Name of the PV module as in the Sandia module database. Use *get_power_plant_data()* with *dataset* = 'sandiamod' to get an overview of all provided modules. See the data set documentation[3] for further information on provided parameters.

inverter_name (str) Name of the inverter as in the CEC inverter database. Use *get_power_plant_data()* with *dataset* = 'cecinverter' to get an overview of all provided inverters. See the data set documentation[4] for further information on provided parameters.

azimuth (float) Azimuth angle of the module surface (South=180).

See also PVSystem.surface_azimuth in pvlib documentation.

tilt (float) Surface tilt angle in decimal degrees. The tilt angle is defined as degrees from horizontal (e.g. surface facing up = 0, surface facing horizon = 90).

See also PVSystem.surface_tilt in pvlib documentation.

albedo (float) The ground albedo. See also PVSystem.albedo in pvlib documentation.

surface_type (str) The ground surface type. See *SURFACE_ALBEDOS* in pvlib.irradiance module for valid values.

References

pv_system_area
Area of PV system in m^2.

pv_system_peak_power
Peak power of PV system in Watt.

The peak power of the PV system can either be limited by the inverter or the PV module(s), wherefore in the case the *mode* parameter, which specifies whether AC or DC feed-in is calculated, is set to 'ac' (which is the default), the minimum of AC inverter power and maximum power of the module(s) is returned. In the case that *mode* is set to 'dc' the inverter power is not considered and the peak power is equal to the maximum power of the module(s).

requires
The parameters this model requires to calculate a feed-in.

The required model parameters are:

location

location (tuple or shapely.Point) Geo location of the PV system. Can either be provided as a tuple with first entry being the latitude and second entry being the longitude or as a shapely.Point.

[3] Sandia module database documentation
[4] CEC inverter database documentation

4.2.2 feedinlib.models.WindpowerlibTurbine

class feedinlib.models.**WindpowerlibTurbine**(***kwargs*)

Model to determine the feed-in of a wind turbine using the windpowerlib.

The windpowerlib[1] is a python library for simulating the performance of wind turbines and farms. For more information about the model check the documentation of the windpowerlib[2].

Notes

In order to use this model various power plant and model parameters have to be provided. See *power_plant_requires* as well as *requires* for further information. Furthermore, the weather data used to calculate the feed-in has to have a certain format. See *feedin()* for further information.

References

See also:

Base WindpowerModelBase

Methods

__init__(**kwargs)	
feedin(weather, power_plant_parameters, **kwargs)	Calculates power plant feed-in in Watt.
instantiate_turbine(**kwargs)	Instantiates a windpowerlib.WindTurbine object.

Attributes

nominal_power_wind_power_plant	Nominal power of wind turbine in Watt.
power_plant_requires	The power plant parameters this model requires to calculate a feed-in.
requires	The parameters this model requires to calculate a feed-in.

__init__(***kwargs*)

feedin(*weather*, *power_plant_parameters*, ***kwargs*)

Calculates power plant feed-in in Watt.

This function uses the windpowerlib's ModelChain to calculate the feed-in for the given weather time series and wind turbine.

Parameters

- **weather** (pandas.DataFrame) – Weather time series used to calculate feed-in. See *weather_df* parameter in windpowerlib's Modelchain run_model method for more information on required variables, units, etc.
- **power_plant_parameters** (*dict*) – Dictionary with power plant specifications. Keys of the dictionary are the power plant parameter names, values of the dictionary hold

[1] windpowerlib on github
[2] windpowerlib documentation

the corresponding value. The dictionary must at least contain the required power plant parameters (see *power_plant_requires*) and may further contain optional power plant parameters (see windpowerlib.WindTurbine).

- ****kwargs** – Keyword arguments can be used to overwrite the windpowerlib's ModelChain parameters.

Returns Power plant feed-in time series in Watt.

Return type pandas.Series

instantiate_turbine(***kwargs*)

Instantiates a windpowerlib.WindTurbine object.

Parameters ****kwargs** – See *power_plant_parameters* parameter in *feedin()* for more information.

Returns Wind turbine to calculate feed-in for.

Return type windpowerlib.WindTurbine

nominal_power_wind_power_plant

Nominal power of wind turbine in Watt.

See WindTurbine.nominal_power in windpowerlib for further information.

power_plant_requires

The power plant parameters this model requires to calculate a feed-in.

The required power plant parameters are:

hub_height, *power_curve/power_coefficient_curve/turbine_type*

hub_height (float) Hub height in m.

See also WindTurbine.hub_height in windpowerlib documentation.

power_curve (pandas.DataFrame or dict) DataFrame/dictionary with wind speeds in m/s and corresponding power curve value in W.

See also WindTurbine.power_curve in windpowerlib documentation.

power_coefficient_curve (pandas.DataFrame or dict) DataFrame/dictionary with wind speeds in m/s and corresponding power coefficient.

See also WindTurbine.power_coefficient_curve in windpowerlib documentation.

turbine_type (str) Name of the wind turbine type as in the oedb turbine library. Use *get_power_plant_data()* with *dataset* = 'oedb_turbine_library' to get an overview of all provided turbines. See the data set metadata[3] for further information on provided parameters.

References

requires

The parameters this model requires to calculate a feed-in.

This model does not require any additional model parameters.

[3] oedb wind turbine library

4.2.3 feedinlib.models.WindpowerlibTurbineCluster

class feedinlib.models.**WindpowerlibTurbineCluster**(***kwargs*)

Model to determine the feed-in of a wind turbine cluster using the windpowerlib.

The windpowerlib[1] is a python library for simulating the performance of wind turbines and farms. For more information about the model check the documentation of the windpowerlib[2].

Notes

In order to use this model various power plant and model parameters have to be provided. See *power_plant_requires* as well as *requires* for further information. Furthermore, the weather data used to calculate the feed-in has to have a certain format. See *feedin()* for further information.

See also:

Base WindpowerModelBase

References

Methods

__init__(**kwargs)	
feedin(weather, power_plant_parameters, **kwargs)	Calculates power plant feed-in in Watt.
instantiate_turbine(**kwargs)	Instantiates a windpowerlib.WindTurbine object.
instantiate_turbine_cluster(**kwargs)	Instantiates a windpowerlib.WindTurbineCluster object.
instantiate_windfarm(**kwargs)	Instantiates a windpowerlib.WindFarm object.

Attributes

nominal_power_wind_power_plant	Nominal power of wind turbine cluster in Watt.
power_plant_requires	The power plant parameters this model requires to calculate a feed-in.
requires	The parameters this model requires to calculate a feed-in.

__init__(***kwargs*)

feedin(*weather*, *power_plant_parameters*, ***kwargs*)

Calculates power plant feed-in in Watt.

This function uses the windpowerlib's TurbineClusterModelChain to calculate the feed-in for the given weather time series and wind farm or cluster.

Parameters

- **weather** (pandas.DataFrame) – Weather time series used to calculate feed-in. See *weather_df* parameter in windpowerlib's TurbineClusterModelChain run_model method for more information on required variables, units, etc.

[1] windpowerlib on github
[2] windpowerlib documentation

- **power_plant_parameters** (*dict*) – Dictionary with either wind farm or wind turbine cluster specifications. For more information on wind farm parameters see *kwargs* in *instantiate_windfarm()*. For information on turbine cluster parameters see *kwargs* in *instantiate_turbine_cluster()*.
- ****kwargs** – Keyword arguments can be used to overwrite the windpowerlib's TurbineClusterModelChain parameters.

Returns Power plant feed-in time series in Watt.

Return type pandas.Series

instantiate_turbine(***kwargs*)
Instantiates a windpowerlib.WindTurbine object.

Parameters ****kwargs** – Dictionary with wind turbine specifications. Keys of the dictionary are the power plant parameter names, values of the dictionary hold the corresponding value. The dictionary must at least contain the required turbine parameters (see *power_plant_requires*) and may further contain optional power plant parameters (see windpowerlib.WindTurbine).

Returns Wind turbine in wind farm or turbine cluster.

Return type windpowerlib.WindTurbine

instantiate_turbine_cluster(***kwargs*)
Instantiates a windpowerlib.WindTurbineCluster object.

Parameters ****kwargs** – Dictionary with turbine cluster specifications. Keys of the dictionary are the parameter names, values of the dictionary hold the corresponding value. The dictionary must at least contain a list of wind farms (see 'wind_farms' specifications in *power_plant_requires*) and may further contain optional wind turbine cluster parameters (see windpowerlib.WindTurbineCluster).

Returns

Return type windpowerlib.WindTurbineCluster

instantiate_windfarm(***kwargs*)
Instantiates a windpowerlib.WindFarm object.

Parameters ****kwargs** – Dictionary with wind farm specifications. Keys of the dictionary are the parameter names, values of the dictionary hold the corresponding value. The dictionary must at least contain a wind turbine fleet (see 'wind_turbine_fleet' specifications in *power_plant_requires*) and may further contain optional wind farm parameters (see windpowerlib.WindFarm).

Returns

Return type windpowerlib.WindFarm

nominal_power_wind_power_plant
Nominal power of wind turbine cluster in Watt.

The nominal power is the sum of the nominal power of all turbines. See *nominal_power* of windpowerlib.WindFarm or windpowerlib.WindTurbineCluster for further information.

power_plant_requires
The power plant parameters this model requires to calculate a feed-in.

The required power plant parameters are:

wind_turbine_fleet/wind_farms

The windpowerlib differentiates between wind farms as a group of wind turbines (of the same or different type) in the same location and wind turbine clusters as wind farms and turbines that are assigned the same weather data point to obtain weather data for feed-in calculations and can therefore be clustered to speed up calculations. The *WindpowerlibTurbineCluster* class can be used for both windpowerlib.WindFarm and windpowerlib.WindTurbineCluster calculations. To set up a windpowerlib.WindFarm please provide a *wind_turbine_fleet* and to set up a windpowerlib.WindTurbineCluster please provide a list of *wind_farms*. See below for further information.

wind_turbine_fleet (pandas.DataFrame) The wind turbine fleet specifies the turbine types and their corresponding number or total installed capacity in the wind farm. DataFrame must have columns 'wind_turbine' and either 'number_of_turbines' (number of wind turbines of the same turbine type in the wind farm, can be a float) or 'total_capacity' (installed capacity of wind turbines of the same turbine type in the wind farm in Watt).

The wind turbine in column 'wind_turbine' can be provided as a *WindPowerPlant* object, a dictionary with power plant parameters (see *power_plant_requires* for required parameters) or a windpowerlib.WindTurbine.

See also *wind_turbine_fleet* parameter of windpowerlib.WindFarm.

The wind turbine fleet may also be provided as a list of windpowerlib.WindTurbineGroup as described there.

wind_farms (list(dict) or list(windpowerlib.WindFarm)) List of wind farms in cluster. Wind farms in the list can either be provided as windpowerlib.WindFarm or as dictionaries where the keys of the dictionary are the wind farm parameter names and the values of the dictionary hold the corresponding value. The dictionary must at least contain a wind turbine fleet (see 'wind_turbine_fleet' parameter specifications above) and may further contain optional wind farm parameters (see windpowerlib.WindFarm).

requires

The parameters this model requires to calculate a feed-in.

This model does not require any additional model parameters.

4.3 Weather data

The feedinlib enables download of open_FRED weather data (local reanalysis data for Germany) and ERA5 weather data (global reanalysis data for the whole world).

feedinlib.db.Weather
feedinlib.era5.weather_df_from_era5
feedinlib.era5.get_era5_data_from_datespan_and_position

4.4 Tools

feedinlib.models.get_power_plant_data(...)	Function to retrieve power plant data sets provided by feed-in models.

4.4.1 feedinlib.models.get_power_plant_data

feedinlib.models.**get_power_plant_data**(*dataset*, ***kwargs*)

Function to retrieve power plant data sets provided by feed-in models.

This function can be used to retrieve power plant data from data sets and to get an overview of which modules, inverters and turbine types are provided and can be used in feed-in calculations.

Parameters

- **dataset** (*str*) – Specifies data set to retrieve. Possible options are:
  - pvlib PV module and inverter datasets: 'sandiamod', 'cecinverter'

    The original data sets are hosted here: https://github.com/NREL/SAM/tree/develop/deploy/libraries

    See retrieve_sam for further information.
  - windpowerlib wind turbine dataset: 'oedb_turbine_library'

    See get_turbine_types for further information.
- ****kwargs** – See referenced functions for each dataset above for further optional parameters.

Example

```
>>> from feedinlib import get_power_plant_data
>>> data = get_power_plant_data('sandiamod')
>>> # list of all provided PV modules
>>> pv_modules = data.columns
>>> print(data.loc["Area", data.columns.str.contains('Aleo_S03')])
Aleo_S03_160__2007__E__    1.28
Aleo_S03_165__2007__E__    1.28
Name: Area, dtype: object
```

4.5 Abstract classes

The feedinlib uses abstract classes for power plant and feed-in models that serve as blueprints for classes that implement those models. This ensures that new models provide required implementations that make it possible to easily exchange the model used in your calculation. They are important for people who want to implement new power plant and model classes rather than for users.

feedinlib.powerplants.Base(**attributes)	The base class of feedinlib power plants.
feedinlib.models.Base(**kwargs)	The base class of feedinlib models.
feedinlib.models.PhotovoltaicModelBase(**kwargs)	Expands model base class *Base* by PV specific attributes.
feedinlib.models.WindpowerModelBase(**kwargs)	Expands model base class *Base* by wind power specific attributes.

4.5.1 feedinlib.powerplants.Base

class feedinlib.powerplants.**Base**(***attributes*)

The base class of feedinlib power plants.

The class mainly serves as a data container for power plant attributes. Actual calculation of feed-in provided by the power plant is done by the chosen model. See model.py module for implemented models.

This base class is an abstract class serving as a blueprint for classes that implement weather dependent renewable energy power plants. It forces implementors to implement certain properties and methods.

Parameters

- **model** (A subclass or instance of subclass of *Base*) – The *model* parameter defines the feed-in model used to calculate the power plant feed-in.

  If a class (or in general, any instance of type) is provided, it is used to create the model instance encapsulating the actual mathematical model used to calculate the feed-in provided by this power plant.

  In any other case, the provided object is used directly. Note though, that a reference to this power plant is saved in the provided object, so sharing model instances between two power plant objects is not a good idea, as the second power plant will overwrite the reference to the first.

  The non-class version is only provided for users who need the extra flexibility of controlling model instantiation and who know what they are doing. In general, you'll want to provide a class for this parameter or just go with the default for the specific subclass you are using.

- ***attributes** – Besides *model* parameter provided attributes hold the technical specification used to define the power plant. See *power_plant_parameters* parameter in respective model's *feedin()* method for further information on the model's required and optional plant parameters.

Raises AttributeError – In case an attribute listed in the given model's required parameters is not present in the *parameters* parameter.

Methods

__init__(**attributes)	
feedin(weather, **kwargs)	Calculates power plant feed-in in Watt.

Attributes

required	The power plant parameters the specified model requires.

__init__(***attributes*)

feedin(*weather*, ***kwargs*)

Calculates power plant feed-in in Watt.

This method delegates the actual computation to the model's *feedin()* method while giving you the opportunity to override some of the inputs used to calculate the feed-in.

If the respective model does calculate AC and DC feed-in, AC feed-in is returned by default. See the model's *feedin()* method for information on how to overwrite this default behaviour.

Parameters

- **weather** – Weather data to calculate feed-in. Check the *weather* parameter of the respective model's *feedin()* method for required weather data parameters and format.
- ****kwargs** – Keyword arguments for respective model's feed-in calculation. Check the keyword arguments of the model's *feedin()* for further information.

Returns **feedin** – Series with power plant feed-in in Watt.

Return type pandas.Series

required

The power plant parameters the specified model requires.

Check the model's `power_plant_requires` attribute for further information.

4.5.2 feedinlib.models.Base

class feedinlib.models.**Base**(***kwargs*)

The base class of feedinlib models.

This base class is an abstract class serving as a blueprint for classes that implement feed-in models for weather dependent renewable energy resources. It forces implementors to implement certain properties and methods.

Methods

__init__(**kwargs)		
feedin(weather, **kwargs)	power_plant_parameters,	Calculates power plant feed-in in Watt.

Attributes

power_plant_requires	The (names of the) power plant parameters this model requires in order to calculate the feed-in.
requires	The (names of the) parameters this model requires in order to calculate the feed-in.

__init__(***kwargs*)

feedin(*weather*, *power_plant_parameters*, ***kwargs*)

Calculates power plant feed-in in Watt.

As this is an abstract method you have to override it in a subclass so that the power plant feed-in using the respective model can be calculated.

Parameters

- **weather** – Weather data to calculate feed-in. Format and required parameters depend on the model.
- **power_plant_parameters** (*dict*) – Dictionary with power plant specifications. Keys of the dictionary are the power plant parameter names, values of the dictionary hold the corresponding value. The dictionary must at least contain the power plant parameters required by the respective model and may further contain optional power plant parameters. See *power_plant_requires* property of the respective model for futher information.

- ****kwargs** – Keyword arguments for respective model's feed-in calculation.

Returns **feedin** – Series with power plant feed-in for specified time span in Watt. If respective model does calculate AC and DC feed-in, AC feed-in should be returned by default. *mode* parameter can be used to overwrite this default behavior and return DC power output instead (for an example see *feedin()*).

Return type pandas.Series

power_plant_requires

The (names of the) power plant parameters this model requires in order to calculate the feed-in.

As this is an abstract property you have to override it in a subclass so that the model can be instantiated. This forces implementors to make the required power plant parameters for a model explicit, even if they are empty, and gives them a good place to document them.

By default, this property is settable and its value can be specified via an argument upon construction. If you want to keep this functionality, simply delegate all calls to the superclass.

Parameters **names** (*list(str), optional*) – Containing the names of the required power plant parameters.

requires

The (names of the) parameters this model requires in order to calculate the feed-in.

As this is an abstract property you have to override it in a subclass so that the model can be instantiated. This forces implementors to make the required model parameters explicit, even if they are empty, and gives them a good place to document them.

By default, this property is settable and its value can be specified via an argument upon construction. If you want to keep this functionality, simply delegate all calls to the superclass.

Parameters **names** (*list(str), optional*) – Containing the names of the required power plant parameters.

4.5.3 feedinlib.models.PhotovoltaicModelBase

class feedinlib.models.**PhotovoltaicModelBase**(***kwargs*)

Expands model base class *Base* by PV specific attributes.

Methods

__init__(**kwargs)		
feedin(weather, **kwargs)	power_plant_parameters,	Calculates power plant feed-in in Watt.

Attributes

power_plant_requires	The (names of the) power plant parameters this model requires in order to calculate the feed-in.
pv_system_area	Area of PV system in m^2.
pv_system_peak_power	Peak power of PV system in Watt.
requires	The (names of the) parameters this model requires in order to calculate the feed-in.

__init__(***kwargs*)

feedin(*weather*, *power_plant_parameters*, ***kwargs*)
Calculates power plant feed-in in Watt.

As this is an abstract method you have to override it in a subclass so that the power plant feed-in using the respective model can be calculated.

Parameters

- **weather** – Weather data to calculate feed-in. Format and required parameters depend on the model.
- **power_plant_parameters** (*dict*) – Dictionary with power plant specifications. Keys of the dictionary are the power plant parameter names, values of the dictionary hold the corresponding value. The dictionary must at least contain the power plant parameters required by the respective model and may further contain optional power plant parameters. See *power_plant_requires* property of the respective model for futher information.
- ****kwargs** – Keyword arguments for respective model's feed-in calculation.

Returns **feedin** – Series with power plant feed-in for specified time span in Watt. If respective model does calculate AC and DC feed-in, AC feed-in should be returned by default. *mode* parameter can be used to overwrite this default behavior and return DC power output instead (for an example see *feedin()*).

Return type pandas.Series

power_plant_requires
The (names of the) power plant parameters this model requires in order to calculate the feed-in.

As this is an abstract property you have to override it in a subclass so that the model can be instantiated. This forces implementors to make the required power plant parameters for a model explicit, even if they are empty, and gives them a good place to document them.

By default, this property is settable and its value can be specified via an argument upon construction. If you want to keep this functionality, simply delegate all calls to the superclass.

Parameters **names** (*list(str), optional*) – Containing the names of the required power plant parameters.

pv_system_area
Area of PV system in m^2.

As this is an abstract property you have to override it in a subclass so that the model can be instantiated. This forces implementors to provide a way to retrieve the area of the PV system that is e.g. used to scale the feed-in by area.

pv_system_peak_power
Peak power of PV system in Watt.

As this is an abstract property you have to override it in a subclass so that the model can be instantiated. This forces implementors to provide a way to retrieve the peak power of the PV system that is e.g. used to scale the feed-in by installed capacity.

requires
The (names of the) parameters this model requires in order to calculate the feed-in.

As this is an abstract property you have to override it in a subclass so that the model can be instantiated. This forces implementors to make the required model parameters explicit, even if they are empty, and gives them a good place to document them.

By default, this property is settable and its value can be specified via an argument upon construction. If you want to keep this functionality, simply delegate all calls to the superclass.

Parameters **names** (*list(str), optional*) – Containing the names of the required power plant parameters.

4.5.4 feedinlib.models.WindpowerModelBase

class feedinlib.models.**WindpowerModelBase**(***kwargs*)

Expands model base class *Base* by wind power specific attributes.

Methods

__init__(**kwargs)		
feedin(weather, **kwargs)	power_plant_parameters,	Calculates power plant feed-in in Watt.

Attributes

nominal_power_wind_power_plant	Nominal power of wind power plant in Watt.
power_plant_requires	The (names of the) power plant parameters this model requires in order to calculate the feed-in.
requires	The (names of the) parameters this model requires in order to calculate the feed-in.

__init__(***kwargs*)

feedin(*weather*, *power_plant_parameters*, ***kwargs*)

Calculates power plant feed-in in Watt.

As this is an abstract method you have to override it in a subclass so that the power plant feed-in using the respective model can be calculated.

Parameters

- **weather** – Weather data to calculate feed-in. Format and required parameters depend on the model.
- **power_plant_parameters** (*dict*) – Dictionary with power plant specifications. Keys of the dictionary are the power plant parameter names, values of the dictionary hold the corresponding value. The dictionary must at least contain the power plant parameters required by the respective model and may further contain optional power plant parameters. See *power_plant_requires* property of the respective model for futher information.
- ****kwargs** – Keyword arguments for respective model's feed-in calculation.

Returns **feedin** – Series with power plant feed-in for specified time span in Watt. If respective model does calculate AC and DC feed-in, AC feed-in should be returned by default. *mode* parameter can be used to overwrite this default behavior and return DC power output instead (for an example see *feedin()*).

Return type pandas.Series

nominal_power_wind_power_plant

Nominal power of wind power plant in Watt.

As this is an abstract property you have to override it in a subclass so that the model can be instantiated. This forces implementors to provide a way to retrieve the nominal power of the wind power plant that is e.g. used to scale the feed-in by installed capacity.

power_plant_requires

The (names of the) power plant parameters this model requires in order to calculate the feed-in.

As this is an abstract property you have to override it in a subclass so that the model can be instantiated. This forces implementors to make the required power plant parameters for a model explicit, even if they are empty, and gives them a good place to document them.

By default, this property is settable and its value can be specified via an argument upon construction. If you want to keep this functionality, simply delegate all calls to the superclass.

Parameters **names** (*list(str), optional*) – Containing the names of the required power plant parameters.

requires

The (names of the) parameters this model requires in order to calculate the feed-in.

As this is an abstract property you have to override it in a subclass so that the model can be instantiated. This forces implementors to make the required model parameters explicit, even if they are empty, and gives them a good place to document them.

By default, this property is settable and its value can be specified via an argument upon construction. If you want to keep this functionality, simply delegate all calls to the superclass.

Parameters **names** (*list(str), optional*) – Containing the names of the required power plant parameters.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Index

Symbols

A

B

F

G

I

N

P

R

W