PVLIB_Python Documentation

Release 0.3.0

Sandia National Labs, Rob Andrews, University of Arizona, github

January 19, 2016
## Contents

1 Installation .......................................................... 3

2 Contents .................................................................. 5
   2.1 What’s New ...................................................... 5
   2.2 Comparison with PVLIB_MATLAB ..................... 9
   2.3 Forecast Module Background .............................. 10
   2.4 Modules .......................................................... 22

3 Indices and tables ....................................................... 71

Python Module Index .................................................... 73
pvlib-python provides a set of documented functions for simulating the performance of photovoltaic energy systems. The toolbox was originally developed in MATLAB at Sandia National Laboratories and it implements many of the models and methods developed at the Labs. More information on Sandia Labs PV performance modeling programs can be found at https://pypmc.sandia.gov/.

The source code for pvlib-python is hosted on github.

The github page also contains a valuable wiki with information on how you can contribute to pvlib-python development!

Please see the links above for details on the status of the pvlib-python project. We are at an early stage in the development of this project, so expect to see significant API changes in the next few releases.

This documentation focuses on providing a reference for all of the modules and functions available in pvlib-python. For examples of how to use pvlib-python, please see the tutorials. Some of the tutorials were written with older versions of pvlib-python and we would greatly appreciate your help updating them!

**Note:** This documentation assumes general familiarity with Python, NumPy, and Pandas. Google searches will yield many excellent tutorials for these packages.

Please see our [PVSC 2014 paper](#) and [PVSC 2015 paper](#) (and the notebook to reproduce the figures) for more information.

The GitHub wiki also has a page on [Projects and publications that use pvlib python](#) for inspiration and listing of your application.
1. Follow Pandas’ instructions for installing the scientific python stack, including pip.

2. `pip install pvlib-python`
2.1 What’s New

These are new features and improvements of note in each release.

2.1.1 v0.2.2 (November 13, 2015)

This is a minor release from 0.2.1. We recommend that all users upgrade to this version.

Enhancements

- Adds Python 3.5 compatibility (GH87)
- Moves the Linke turbidity lookup into clearsky.lookup_linke_turbidity. The API for clearsky.ineichen remains the same. (GH95)

Bug fixes

- irradiance.total_irrad had a typo that required the Klucher model to be accessed with ‘klucher’. Both spellings will work for the remaining 0.2.* versions of pvlib, but the misspelled method will be removed in 0.3. (GH97)
- Fixes an import and KeyError in the IPython notebook tutorials (GH94).
- Uses the logging module properly by replacing format calls with args. This results in a 5x speed increase for tracking.singleaxis (GH89).
- Adds a link to the 2015 PVSC paper (GH81)

Contributors

- Will Holmgren
- jetheurer
- dacoex
2.1.2 v0.2.1 (July 16, 2015)

This is a minor release from 0.2. It includes a large number of bug fixes for the IPython notebook tutorials. We recommend that all users upgrade to this version.

Enhancements

- Update component info from SAM (csvs dated 2015-6-30) (GH75)

Bug fixes

- Fix incorrect call to Perez irradiance function (GH76)
- Fix numerous bugs in the IPython notebook tutorials (GH30)

Contributors

- Will Holmgren
- Jessica Forbess

2.1.3 v0.2.0 (July 6, 2015)

This is a major release from 0.1 and includes a large number of API changes, several new features and enhancements along with a number of bug fixes. We recommend that all users upgrade to this version.

Due to the large number of API changes, you will probably need to update your code.

API changes

- Change variable names to conform with new Variables and style rules wiki. This impacts many function declarations and return values. Your existing code probably will not work! (GH37, GH54).
- Move dirint and disc algorithms from clearsky.py to irradiance.py (GH42)
- Mark some pvsystem.py methods as private (GH20)
- Make output of pvsystem.sapm_celltemp a DataFrame (GH54)

Enhancements

- Add conda installer
- PEP8 fixups to solarposition.py and spa.py (GH50)
- Add optional projection_ratio keyword argument to the haydavies calculator. Speeds calculations when irradiance changes but solar position remains the same (GH58)
- Improved installation instructions in README.
Bug fixes

• fix local build of the documentation (GH49, GH56)
• The release date of 0.1 was fixed in the documentation (see v0.1.0 (April 20, 2015))
• fix casting of DateTimeIndex to int64 epoch timestamp on machines with 32 bit python int (GH63)
• fixed some docstrings with failing doctests (GH62)

Contributors

• Will Holmgren
• Rob Andrews
• bmu
• Tony Lorenzo

2.1.4 v0.1.0 (April 20, 2015)

This is the first official release of the pvlib-python project. As such, a “What’s new” document is a little hard to write. There will be significant overlap with the to-be-written document that describes the differences between pvlib-python and PVLIB_Matlab.

API changes

• Remove pvl_ from module names.
• Consolidation of similar modules. For example, functions from pvl_clearsky_ineichen.py and pvl_clearsky_haurwitz.py have been consolidated into clearsky.py.
• Return one DataFrame instead of a tuple of DataFrames.
• Change function and module names so that they do not conflict.

New features

• Library is Python 3.3 and 3.4 compatible
• Add What’s New section to docs (GH10)
• Add PyEphem option to solar position calculations.
• Add a Python translation of NREL’s SPA algorithm.
• irradiance.py has more AOI, projection, and irradiance sum and calculation functions
• TMY data import has a coerce_year option
• TMY data can be loaded from a url (GH5)
• Locations are now pvlib.location.Location objects, not “structs”.
• Specify time zones using a string from the standard IANA Time Zone Database naming conventions or using a pytz.timezone instead of an integer GMT offset. We may add dateutils support in the future.
• clearsky.ineichen supports interpolating monthly Linke Turbidities to daily resolution.
Other changes

- Removed `Vars=Locals(); Expect...; var=pvl\_tools.Parse(Vars,Expect);` pattern. Very few tests of input validity remain. Garbage in, garbage or nan out.
- Removing unnecessary and sometimes undesired behavior such as setting maximum zenith=90 or airmass=0. Instead, we make extensive use of nan values.
- Adding logging calls, removing print calls.
- Improved PEP8 compliance.
- Added `/pvlib/data` for lookup tables, test, and tutorial data.
- Limited the scope of `clearsky.py`'s `scipy` dependency. `clearsky.ineichen` will work without `scipy` so long as the Linke Turbidity is supplied as a keyword argument. (GH13)
- Removed NREL's SPA code to comply with their license (GH9).
- Revised the globalinplane function and added a test_globalinplane (GH21, GH33).

Documentation

- Using readthedocs for documentation hosting.
- Many typos and formatting errors corrected (GH16)
- Documentation source code and tutorials live in `/` rather than `/pvlib/docs`.
- Additional tutorials in `/docs/tutorials`.
- Clarify `pvsystem.systemdef` input (GH17)

Testing

- Tests are cleaner and more thorough. They are still nowhere near complete.
- Using Coveralls to measure test coverage.
- Using TravisCI for automated testing.
- Using `nose` for more concise test code.

Bug fixes

- Fixed DISC algorithm bugs concerning modifying input zenith Series (GH24), the $K_t$ conditional evaluation (GH6), and ignoring the input pressure (GH25).
- Many more bug fixes were made, but you’ll have to look at the detailed commit history.
- Fixed inconsistent azimuth angle in the ephemeris function (GH40)

Contributors

This list includes all (I hope) contributors to `pvlib/pvlib-python`, `Sandia-Labs/PVLIB_Python`, and `UARENForecasting/PVLIB_Python`.

- Rob Andrews
- Will Holmgren
2.2 Comparison with PVLIB_MATLAB

This document is under construction. Please see our PVSC 2014 paper and PVSC 2015 abstract for more information. The pvlib-python license is BSD 3-clause, the PVLIB_MATLAB license is ??.

We want to keep developing the core functionality and algorithms of the Python and MATLAB projects roughly in parallel, but we’re not making any promises at this point. The PVLIB_MATLAB and pvlib-python projects are currently developed by different teams that do not regularly work together. We hope to grow this collaboration in the future. Do not expect feature parity between the libraries, only similarity.

Here are some of the major differences between the latest pvlib-python build and the original Sandia PVLIB_Python project, but many of these comments apply to the difference between pvlib-python and PVLIB_MATLAB.

2.2.1 Library wide changes

• Remove `pvl_` from module names.
• Consolidation of similar modules. For example, functions from `pvl_clearsky_ineichen.py` and `pvl_clearsky_haurwitz.py` have been consolidated into `clearsky.py`.
• Removed `Vars=Locals(); Expect...; var=pvl\_tools.Parse(Vars,Expect);` pattern. Very few tests of input validity remain. Garbage in, garbage or `nan` out.
• Removing unnecessary and sometimes undesired behavior such as setting maximum zenith=90 or airmass=0. Instead, we make extensive use of `nan` values.
• Changing function and module names so that they do not conflict.
• Added `/pvlib/data` for lookup tables, test, and tutorial data.
• Added “forecast.py” module, for downloading UNIDATA weather forecast data.

2.2.2 More specific changes

• Add PyEphem option to solar position calculations.
• `irradiance.py` has more AOI, projection, and irradiance sum and calculation functions
• Locations are now `pvlib.location.Location` objects, not structs.
• Specify time zones using a string from the standard JANA Time Zone Database naming conventions or using a `pytz.timezone` instead of an integer GMT offset. We may add dateutil support in the future.
• `clearsky.ineichen` supports interpolating monthly Linke Turbidities to daily resolution.
Instead of requiring effective irradiance as an input, `pvsystem.sapm` calculates and returns it based on input POA irradiance, AM, and AOI.

### 2.2.3 Documentation

- Using readthedocs for documentation hosting.
- Many typos and formatting errors corrected.
- Documentation source code and tutorials live in `/` rather than `/pvlib/docs`.
- Additional tutorials in `/docs/tutorials`.

### 2.2.4 Testing

- Tests are cleaner and more thorough. They are still no where near complete.
- Using Coveralls to measure test coverage.
- Using TravisCI for automated testing.
- Using `nose` for more concise test code.

### 2.3 Forecast Module Background

This document is under construction.

Here is the background behind the development of the forecast module.

#### 2.3.1 PV System Power Calculation with pvlib-python

Here is the process that is used to produce a power output estimate using pvlib library.
2.3.2 Forecasted Meteorological Data

Listed in the chart below are the forecast models and their intervals for which data is available on the Unidata THREDDS server.
Where does the data come from?

Unidata hosts a THREDDS Data Server that contains the forecast model data from NCAR, NCEP, and FNMOC.

**Thematic Real-time Environmental Distributed Data Services (THREDDS)**

- [http://thredds.ucar.edu](http://thredds.ucar.edu)
- [http://thredds.ucar.edu/thredds/catalog.html](http://thredds.ucar.edu/thredds/catalog.html)
- XML based online data repository
- Multiple data formats supported (netCDF, HDF5, GRIB, NEXRAD, OPeNDAP, etc.)
2.3.4 Unidata Siphon

What is Siphon?
Open source Python library for downloading data from Unidata technologies THREDDS Data Server (TDS).
https://github.com/Unidata/siphon

How to download the data? Using the NCSS (netCDF subset service) module which accesses the xml catalogs on the TDS.

What versions of Python are supported? Python 2.7 & Python >= 3.3
Available via pypi, Binstar, and Github

2.3.5 pvlib-python Forecast Module Objectives

The criteria for the module development were as follows (many were focused around object-oriented programming).
- Simple and easy to use
- Comprehensive
- Flexible
- Integrated
- Standardized
2.3.6 Challenges

There were several challenges that were addressed when putting together the forecast module.

- **Data format dissimilarities between forecast models**
  - **Forecast period** Many of the forecasts come at different intervals and span different lengths of time.
  - **Variables provided** The model share many of the same quantities, however they are labeled using different terms or need to be converted into useful values.
– **Data availability**  The models are updated at different intervals and also are sometimes missing data.

  - **Irradiance**

    – **Cloud cover and radiation**  Many of the forecast models do not have radiation observations or if they do they are incorrect. Since it is necessary to have accurate radiation values to calculate power output the model by default uses a basic radiation function (Liu and Jordan, 1960) to generate more appropriate radiation values based on cloud cover.

\[
DNI = \tau^m DNI_{extraterrestrial}
\]
\[
DHI = 0.3(1 - \tau^m)\cos\psi DNI_{extraterrestrial}
\]


### 2.3.7 Forecast Module Structure

Hosted on GitHub [https://github.com/MoonRaker/pvlib-python](https://github.com/MoonRaker/pvlib-python)

**Existing functionality**

- location
- Air mass number (atmosphere.py)
- Direct Radiation (clearsky.py)
- POA diffuse radiation (irradiance.py)
- Solar angles (solarposition.py)
- Conversion to power (pvsystem.py)

**Added functionality**

- Unidata Forecast data (forecast.py)
- Cloud cover to diffuse radiation (irradiance.py)
2.3.8 Model subclass

Each forecast model has its own subclass. These subclasses belong to a more comprehensive parent class that holds many of the methods used by every model.

Within each subclass model specific variables are assigned to common variable labels that are available from each forecast model.

Here are the subclasses for two models.
class GFS(ForecastModel):
...
    Subclass of the ForecastModel class representing GFS forecast model.
    
    Model data corresponds to 0.25 degree resolution forecasts.
    
    Attributes
    ----------
    cols: list
        Common names for variables.
    data_labels: dictionary
        Dictionary where the common variable name references the model specific variable name.
    idx: list
        Indices of the variables corresponding to their common name.
    model: string
        Name of the UNIDATA forecast model.
    model_type: string
        UNIDATA category in which the model is located.
    variables: list
        Names of default variables specific to the model.
    ...

    def __init__(self, res='half', set_type='best'):
        model_type = 'Forecast Model Data'
        if res == 'half':
            model = 'GFS Half Degree Forecast'
        elif res == 'quarter':
            model = 'GFS Quarter Degree Forecast'
        self.variables = {
            'temperature':'Temperature_surface',
            'wind_speed_gust':'Wind_speed_gust_surface',
            'wind_speed_u':'u-component_of_wind_isobaric',
            'wind_speed_v':'v-component_of_wind_isobaric',
            '# pressure':'Pressure_surface',
            'total_clouds':
                'Total_cloud_cover_entire_atmosphere_Mixed_intervals_Average',
                'low_clouds':
                'Total_cloud_cover_low_cloud_Mixed_intervals_Average',
                'mid_clouds':
                'Total_cloud_cover_middle_cloud_Mixed_intervals_Average',
                'high_clouds':
                'Total_cloud_cover_high_cloud_Mixed_intervals_Average',
                'boundary_clouds':
                'Total_cloud_cover_boundary_layer_cloud_Mixed_intervals_Average',
                'convect_clouds':
                'Total_cloud_cover_convective_cloud',
                'ghi':
                Downward_Short-Wave_Radiation_Flux_surface_Mixed_intervals_Average',
        }
        self.modelvariables = self.variables.keys()
        self.queryvariables = [self.variables[key] for key in 
            self.modelvariables]
        self.dataframe_variables = [
            'temperature',
            '# pressure',
            'total_clouds',
            'low_clouds',
            'mid_clouds',
            'high_clouds',
            'boundary_clouds',
            'convect_clouds',
            'ghi',
        ]
        super(GFS, self).__init__(model_type, model, set_type)
2.3.9 ForecastModel class

The following code is part of the parent class that each forecast model belongs to.
Upon instantiation of a forecast model, several assignments are made and functions called to initialize values and objects within the class.

The query function is responsible for completing the retrieval of data from the Unidata THREDDS server using the Unidata siphon THREDDS server API.
```python
def get_query_data(self, latitude, longitude, time, vert_level=None, variables=None):
    ...
    Submits a query to the UNIDATA servers using siphon NCCS and
    converts the netcdf data to a pandas DataFrame.

    Parameters
    ----------

    variables: dictionary
        Variables and common names being queried.
    latlon: list
        Latitude and longitude of query location.
    time: pd.datetimeindex
        Time range of interest.
    vert_level: float or integer
        Vertical altitude of interest.

    Returns
    -------
    pd.DataFrame
    ...

    if vert_level is not None:
        self.vert_level = vert_level
    if variables is not None:
        self.variables = variables
        self.modelvariables = list(self.variables.keys())
        self.queryvariables = [self.variables[key] for key in \
                                self.modelvariables]
        self.columns = self.modelvariables
        self.dataframe_variables = self.modelvariables

    self.latitude = latitude
    self.longitude = longitude
    self.set_query_latlon()
    self.time = time
    self.set_query_time()
    self.query.vertical_level(self.vert_level)
    self.query.variables(*self.queryvariables)
    self.query.accept(self.data_format)
    netcdf_data = self.ncss.get_data(self.query)
    self.set_time(netcdf_data.variables['time'], self.time.tzinfo)
    self.data = self.netcdf2pandas(netcdf_data)
    self.set_variable_units(netcdf_data)
    self.set_variable_stddnames(netcdf_data)
    self.convert_temperature(netcdf_data)

    self.calc_wind(netcdf_data)
    self.calc_radiation(netcdf_data)

    netcdf_data.close()

    return self.data

def netcdf2pandas(self, data):
    ...
    Transforms data from netcdf to pandas DataFrame.
```
The ForecastModel class also contains miscellaneous functions that process raw netcdf data from the THREDDS server and create containers for all the processed data.

```python
def test_temp_convert():
    amodel = ...models['GFS']
    amodel.data = pd.DataFrame({'temperature': [273.15], 'temperature_iso': [273.15]})
    amodel.convert_temperature()

    assert amodel.data['temperature'].values == 0.0
    assert amodel.data['temperature_iso'].values == 0.0
```

### 2.3.10 Tests

The nose library is used to perform tests on the library modules to ensure that the library performs as expected and as it was intended.
2.4 Modules

2.4.1 atmosphere module

The atmosphere module contains methods to calculate relative and absolute airmass and to determine pressure from altitude or vice versa.

```
pvlib.atmosphere.absoluteairmass(airmass_relative, pressure=101325.0)
```
Determine absolute (pressure corrected) airmass from relative airmass and pressure

Gives the airmass for locations not at sea-level (i.e. not at standard pressure). The input argument “AMrelative” is the relative airmass. The input argument “pressure” is the pressure (in Pascals) at the location of interest and must be greater than 0. The calculation for absolute airmass is

\[
\text{absolute airmass} = (\text{relative airmass}) \times \frac{\text{pressure}}{101325}
\]

**Parameters**

- `airmass_relative` : scalar or Series
  The airmass at sea-level.
- `pressure` : scalar or Series
  The site pressure in Pascal.

**Returns**

- scalar or Series
  Absolute (pressure corrected) airmass

**References**


`pvlib.atmosphere.alt2pres(altitude)`

Determine site pressure from altitude.

**Parameters**

- `Altitude` : scalar or Series
  Altitude in meters above sea level

**Returns**

- `Pressure` : scalar or Series
  Atmospheric pressure (Pascals)

**Notes**

The following assumptions are made

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base pressure</td>
<td>101325 Pa</td>
</tr>
<tr>
<td>Temperature at zero altitude</td>
<td>288.15 K</td>
</tr>
<tr>
<td>Gravitational acceleration</td>
<td>9.80665 m/s^2</td>
</tr>
<tr>
<td>Lapse rate</td>
<td>-6.5E-3 K/m</td>
</tr>
<tr>
<td>Gas constant for air</td>
<td>287.053 J/(kgK)</td>
</tr>
<tr>
<td>Relative Humidity</td>
<td>0%</td>
</tr>
</tbody>
</table>

**References**


`pvlib.atmosphere.pres2alt(pressure)`

Determine altitude from site pressure.

**Parameters**

- `pressure` : scalar or Series
  Atmospheric pressure (Pascals)
Returns **altitude**: scalar or Series

Altitude in meters above sea level

**Notes**

The following assumptions are made

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base pressure</td>
<td>101325 Pa</td>
</tr>
<tr>
<td>Temperature at zero altitude</td>
<td>288.15 K</td>
</tr>
<tr>
<td>Gravitational acceleration</td>
<td>9.80665 m/s^2</td>
</tr>
<tr>
<td>Lapse rate</td>
<td>-6.5E-3 K/m</td>
</tr>
<tr>
<td>Gas constant for air</td>
<td>287.053 J/(kgK)</td>
</tr>
<tr>
<td>Relative Humidity</td>
<td>0%</td>
</tr>
</tbody>
</table>

**References**


`pvlib.atmosphere.relativeairmass(zenith, model='kastenyoung1989')`

Gives the relative (not pressure-corrected) airmass

Gives the airmass at sea-level when given a sun zenith angle, $z$ (in degrees). The “model” variable allows selection of different airmass models (described below). “model” must be a valid string. If “model” is not included or is not valid, the default model is ‘kastenyoung1989’.

**Parameters**

- **zenith**: float or Series
  
  Zenith angle of the sun in degrees. Note that some models use the apparent (refraction corrected) zenith angle, and some models use the true (not refraction-corrected) zenith angle. See model descriptions to determine which type of zenith angle is required. Apparent zenith angles must be calculated at sea level.

- **model**: String
  
  Available models include the following:
  
  - ‘simple’ - secant(apparent zenith angle) - Note that this gives -inf at zenith=90

**Returns**

- **airmass_relative**: float or Series
  
  Relative airmass at sea level. Will return NaN values for any zenith angle greater than 90 degrees.
References


\[ \text{pvlib.atmosphere.transmittance(cloud_prct)} \]

Calculates transmittance.

Based on observations by Liu and Jordan, 1960 as well as Gates 1980.

**Parameters**

- cloud_prct: float or int
  
  Percentage of clouds covering the sky.

**Returns**

- value: float
  
  Shortwave radiation transmittance.

References


2.4.2 clearsky module

The clearsky module contains several methods to calculate clear sky GHI, DNI, and DHI.

\[ \text{pvlib.clearsky.haurwitz(apparent_zenith)} \]

Determine clear sky GHI from Haurwitz model.

Implements the Haurwitz clear sky model for global horizontal irradiance (GHI) as presented in [1, 2]. A report on clear sky models found the Haurwitz model to have the best performance of models which require only zenith angle [3]. Extreme care should be taken in the interpretation of this result!

**Parameters**

- apparent_zenith : Series
  
  The apparent (refraction corrected) sun zenith angle in degrees.
Returns pd.Series
The modeled global horizontal irradiance in W/m^2 provided
by the Haurwitz clear-sky model.
Initial implementation of this algorithm by Matthew Reno.

References


pvlib.clearsky.ineichen(time, location, linke_turbidity=None, solarposition_method='pyephem', zenith_data=None, airmass_model='young1994', airmass_data=None, interp_turbidity=True)

Determine clear sky GHI, DNI, and DHI from Ineichen/Perez model
Implements the Ineichen and Perez clear sky model for global horizontal irradiance (GHI), direct normal irradiance (DNI), and calculates the clear-sky diffuse horizontal (DHI) component as the difference between GHI and DNI*cos(zenith) as presented in [1, 2]. A report on clear sky models found the Ineichen/Perez model to have excellent performance with a minimal input data set [3].
Default values for monthly Linke turbidity provided by SoDa [4, 5].

Parameters time : pandas.DatetimeIndex
location : pvlib.Location
linke_turbidity : None or float
    If None, uses LinkeTurbidities.mat lookup table.
solarposition_method : string
    Sets the solar position algorithm. See solarposition.get_solarposition()
zenith_data : None or Series
    If None, ephemeris data will be calculated using solarposition_method.
airmass_model : string
    See pvlib.airmass.relativeairmass().
airmass_data : None or Series
    If None, absolute air mass data will be calculated using airmass_model and location.alitude.
interp_turbidity : bool
    If True, interpolates the monthly Linke turbidity values found in LinkeTurbidities.mat to daily values.

Returns DataFrame with the following columns: ghi, dni, dhi.
Notes

If you are using this function in a loop, it may be faster to load LinkeTurbidities.mat outside of the loop and feed it in as a keyword argument, rather than having the function open and process the file each time it is called.

References


`pvlib.clearsky.lookup_linke_turbidity(time, latitude, longitude, filepath=None, interp_turbidity=True)`

Look up the Linke Turbidity from the LinkeTurbidities.mat data file supplied with pvlib.

**Parameters**

- `time` : pandas.DatetimeIndex
  - latitude : float
  - longitude : float
  - filepath : string
  - `interp_turbidity` : bool
    - If True, interpolates the monthly Linke turbidity values found in LinkeTurbidities.mat to daily values.

**Returns**

- `turbidity` : Series

2.4.3 forecast module

The ‘forecast’ module contains class definitions for retrieving forecasted data from UNIDATA Thredd servers.

```python
class pvlib.forecast.ForecastModel(model_type, model_name, set_type)
```

An object for holding forecast model information for use within the pvlib library.

Simplifies use of siphon library on a THREDDS server.

**Parameters**

- `model_type` : string
  - UNIDATA category in which the model is located.

- `model_name` : string
  - Name of the UNIDATA forecast model.

- `set_type` : string
Model dataset type.

**Attributes**

<table>
<thead>
<tr>
<th>access_url: string</th>
<th>URL specifying the dataset from data will be retrieved.</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_tds_url: string</td>
<td>(string) The top level server address.</td>
</tr>
<tr>
<td>catalog_url: string</td>
<td>(string) The url path of the catalog to parse.</td>
</tr>
<tr>
<td>columns: list</td>
<td>List of headers used to create the data DataFrame.</td>
</tr>
<tr>
<td>data: pd.DataFrame</td>
<td>Data returned from the query.</td>
</tr>
<tr>
<td>data_format: string</td>
<td>Format of the forecast data being requested from UNIDATA.</td>
</tr>
<tr>
<td>dataset: Dataset</td>
<td>Object containing information used to access forecast data.</td>
</tr>
<tr>
<td>dataframe_variables: list</td>
<td>Model variables that are present in the data.</td>
</tr>
<tr>
<td>datasets_list: list</td>
<td>List of all available datasets.</td>
</tr>
<tr>
<td>fm_models: Dataset</td>
<td>Object containing all available forecast models.</td>
</tr>
<tr>
<td>fm_models_list: list</td>
<td>List of all available forecast models from UNIDATA.</td>
</tr>
<tr>
<td>latitude: list</td>
<td>A list of floats containing latitude values.</td>
</tr>
<tr>
<td>location: Location</td>
<td>A pvlib Location object containing geographic quantities.</td>
</tr>
<tr>
<td>longitude: list</td>
<td>A list of floats containing longitude values.</td>
</tr>
<tr>
<td>lbox: boolean</td>
<td>Indicates the use of a location bounding box.</td>
</tr>
<tr>
<td>ncss: NCSS object</td>
<td>NCSS model_name: string Name of the UNIDATA forecast model.</td>
</tr>
<tr>
<td>model: Dataset</td>
<td>A dictionary of Dataset object, whose keys are the name of the dataset’s name.</td>
</tr>
<tr>
<td>model_url: string</td>
<td>The url path of the dataset to parse.</td>
</tr>
<tr>
<td>modelvariables: list</td>
<td>Common variable names that correspond to queryvariables.</td>
</tr>
<tr>
<td>query: NCSS query object</td>
<td>NCSS object used to complete the forecast data retrieval.</td>
</tr>
<tr>
<td>queryvariables: list</td>
<td>Variables that are used to query the THREDDS Data Server.</td>
</tr>
<tr>
<td>rad_type: dictionary</td>
<td>Dictionary labeling the method used for calculating radiation values.</td>
</tr>
<tr>
<td>time: datetime</td>
<td>Time range specified for the NCSS query.</td>
</tr>
<tr>
<td>utctime: DatetimeIndex</td>
<td>Time range in UTC.</td>
</tr>
<tr>
<td>var_stdnames: dictionary</td>
<td>Dictionary containing the standard names of the variables in the query, where the keys are the common names.</td>
</tr>
<tr>
<td>var_units: dictionary</td>
<td>Dictionary containing the unites of the variables in the query, where the keys are the common names.</td>
</tr>
<tr>
<td>variables: dictionary</td>
<td>Dictionary that translates model specific variables to common named variables.</td>
</tr>
<tr>
<td>vert_level: float or integer</td>
<td>Vertical altitude for query data.</td>
</tr>
<tr>
<td>wind_type: string</td>
<td>Quantity that was used to calculate wind_speed.</td>
</tr>
<tr>
<td>zenith: numpy.array</td>
<td>Solar zenith angles for the given time range.</td>
</tr>
</tbody>
</table>

**access_url_key** = ‘NetcdfSubset’

**base_tds_url** = ‘http://thredds.ucar.edu’

**calc_radiation** *(data, cloud_type='total_clouds')*

Determines shortwave radiation values if they are missing from the model data.

**Parameters**

- **data**: netcdf

Query data formatted in netcdf format.

**cloud_type**: string
Type of cloud cover to use for calculating radiation values.

**calc_temperature**(data)
Calculates temperature (in degrees C) from isobaric temperature.

**Parameters**
- **data**: netcdf

  Query data in netcdf format.

**calc_wind**(data)
Computes wind speed.
In some cases only gust wind speed is available. The wind_type attribute will indicate the type of wind speed that is present.

**Parameters**
- **data**: netcdf

  Query data in netcdf format.

**catalog_url** = ‘http://thredds.ucar.edu/thredds/catalog.xml’


**convert_temperature**()
Converts Kelvin to celsius.

**data_format** = ‘netcdf’

**get_query_data**(latitude, longitude, time, vert_level=None, variables=None)
Submits a query to the UNIDATA servers using siphon NCSS and converts the netcdf data to a pandas DataFrame.

**Parameters**
- **latitude**: list
  A list of floats containing latitude values.

- **longitude**: list
  A list of floats containing longitude values.

- **time**: pd.datetimeindex
  Time range of interest.

- **vert_level**: float or integer
  Vertical altitude of interest.

- **variables**: dictionary
  Variables and common names being queried.

**Returns**
- **pd.DataFrame**

**netcdf2pandas**(data)
Transforms data from netcdf to pandas DataFrame.
Currently only supports one-dimensional netcdf data.

**Parameters**
- **data**: netcdf

  Data returned from UNIDATA NCSS query.

**Returns**
- **pd.DataFrame**

**set_dataset**()
Retreives the designated dataset, creates NCSS object, and initiates a NCSS query.
**set_location** *(time)*
Sets the location for

**Parameters**
- **time**: datetime or DatetimeIndex
  Time range of the query.

**set_query_latlon** ()
Sets the NCSS query location latitude and longitude.

**set_query_time** ()
Sets the NCSS query time range.
- **as**: single or range

**set_time** *(time)*
Converts time data into a pandas date object.

**Parameters**
- **time**: netcdf
  Contains time information.

**Returns**
- pandas.DatetimeIndex

**set_variable_stdnames** *(data)*
Extracts standard names from netcdf data.

**Parameters**
- **data**: netcdf
  Contains queried variable information.

**set_variable_units** *(data)*
Extracts variable unit information from netcdf data.

**Parameters**
- **data**: netcdf
  Contains queried variable information.

**vert_level** = 100000

**class** `pvlib.forecast.GFS` *(res='half', set_type='best')*

**Bases**: `pvlib.forecast.ForecastModel`

Subclass of the ForecastModel class representing GFS forecast model.

Model data corresponds to 0.25 degree resolution forecasts.

**Parameters**
- **res**: string
  Resolution of the model.

- **set_type**: string
  Type of model to pull data from.

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataframe_variables</td>
<td>Common variables present in the final set of data.</td>
</tr>
<tr>
<td>model</td>
<td>Name of the UNIDATA forecast model.</td>
</tr>
<tr>
<td>model_type</td>
<td>UNIDATA category in which the model is located.</td>
</tr>
<tr>
<td>modelvariables</td>
<td>Common variable names.</td>
</tr>
<tr>
<td>queryvariables</td>
<td>Names of default variables specific to the model.</td>
</tr>
<tr>
<td>variables</td>
<td>Dictionary of common variables that reference the model specific variables.</td>
</tr>
</tbody>
</table>
class `pvlib.forecast.HRRR`(`set_type='best'`)  
Bases: `pvlib.forecast.ForecastModel`  
Subclass of the `ForecastModel` class representing HRRR forecast model.  
Model data corresponds to NCEP HRRR CONUS 2.5km resolution forecasts.  

Parameters  
set_type: string  
Type of model to pull data from.

Attributes

<table>
<thead>
<tr>
<th>dataframe_variables: list</th>
<th>Common variables present in the final set of data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>model: string</td>
<td>Name of the UNIDATA forecast model.</td>
</tr>
<tr>
<td>model_type: string</td>
<td>UNIDATA category in which the model is located.</td>
</tr>
<tr>
<td>modelVariables: list</td>
<td>Common variable names.</td>
</tr>
<tr>
<td>queryvariables: list</td>
<td>Names of default variables specific to the model.</td>
</tr>
<tr>
<td>variables: dictionary</td>
<td>Dictionary of common variables that reference the model specific variables.</td>
</tr>
</tbody>
</table>

class `pvlib.forecast.HRRR_ESRL`(`set_type='best'`)  
Bases: `pvlib.forecast.ForecastModel`  
Subclass of the `ForecastModel` class representing NOAA/GSD/ESRL’s HRRR forecast model. This is not an operational product.  
Model data corresponds to NOAA/GSD/ESRL HRRR CONUS 3km resolution surface forecasts.  

Parameters  
set_type: string  
Type of model to pull data from.

Attributes

<table>
<thead>
<tr>
<th>dataframe_variables: list</th>
<th>Common variables present in the final set of data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>model: string</td>
<td>Name of the UNIDATA forecast model.</td>
</tr>
<tr>
<td>model_type: string</td>
<td>UNIDATA category in which the model is located.</td>
</tr>
<tr>
<td>modelVariables: list</td>
<td>Common variable names.</td>
</tr>
<tr>
<td>queryvariables: list</td>
<td>Names of default variables specific to the model.</td>
</tr>
<tr>
<td>variables: dictionary</td>
<td>Dictionary of common variables that reference the model specific variables.</td>
</tr>
</tbody>
</table>

class `pvlib.forecast.NAM`(`set_type='best'`)  
Bases: `pvlib.forecast.ForecastModel`  
Subclass of the `ForecastModel` class representing NAM forecast model.  
Model data corresponds to NAM CONUS 12km resolution forecasts from CONDUIT.  

Parameters  
set_type: string  
Type of model to pull data from.
Attributes

<table>
<thead>
<tr>
<th>dataframe_variables: list</th>
<th>Common variables present in the final set of data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>model: string</td>
<td>Name of the UNIDATA forecast model.</td>
</tr>
<tr>
<td>model_type: string</td>
<td>UNIDATA category in which the model is located.</td>
</tr>
<tr>
<td>modelVariables: list</td>
<td>Common variable names.</td>
</tr>
<tr>
<td>queryVariables: list</td>
<td>Names of default variables specific to the model.</td>
</tr>
<tr>
<td>variables: dictionary</td>
<td>Dictionary of common variables that reference the model specific variables.</td>
</tr>
</tbody>
</table>

class pvlib.forecast.NDFD (set_type='best')

Bases: pvlib.forecast.ForecastModel

Subclass of the ForecastModel class representing NDFD forecast model.
Model data corresponds to NWS CONUS CONDUIT forecasts.

Parameters

set_type: string
Type of model to pull data from.

Attributes

<table>
<thead>
<tr>
<th>dataframe_variables: list</th>
<th>Common variables present in the final set of data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>model: string</td>
<td>Name of the UNIDATA forecast model.</td>
</tr>
<tr>
<td>model_type: string</td>
<td>UNIDATA category in which the model is located.</td>
</tr>
<tr>
<td>modelVariables: list</td>
<td>Common variable names.</td>
</tr>
<tr>
<td>queryVariables: list</td>
<td>Names of default variables specific to the model.</td>
</tr>
<tr>
<td>variables: dictionary</td>
<td>Dictionary of common variables that reference the model specific variables.</td>
</tr>
</tbody>
</table>

class pvlib.forecast.RAP (set_type='best')

Bases: pvlib.forecast.ForecastModel

Subclass of the ForecastModel class representing RAP forecast model.
Model data corresponds to Rapid Refresh CONUS 20km resolution forecasts.

Parameters

set_type: string
Type of model to pull data from.

Attributes

<table>
<thead>
<tr>
<th>dataframe_variables: list</th>
<th>Common variables present in the final set of data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>model: string</td>
<td>Name of the UNIDATA forecast model.</td>
</tr>
<tr>
<td>model_type: string</td>
<td>UNIDATA category in which the model is located.</td>
</tr>
<tr>
<td>modelVariables: list</td>
<td>Common variable names.</td>
</tr>
<tr>
<td>queryVariables: list</td>
<td>Names of default variables specific to the model.</td>
</tr>
<tr>
<td>variables: dictionary</td>
<td>Dictionary of common variables that reference the model specific variables.</td>
</tr>
</tbody>
</table>

2.4.4 irradiance module

The irradiance module contains functions for modeling global horizontal irradiance, direct normal irradiance, diffuse horizontal irradiance, and total irradiance under various conditions.
pvlib.irradiance.aoi(surface_tilt, surface_azimuth, solar_zenith, solar_azimuth)
Calculates the angle of incidence of the solar vector on a surface. This is the angle between the solar vector and the surface normal.

Input all angles in degrees.

Parameters

- **surface_tilt**: float or Series.
  Panel tilt from horizontal.

- **surface_azimuth**: float or Series.
  Panel azimuth from north.

- **solar_zenith**: float or Series.
  Solar zenith angle.

- **solar_azimuth**: float or Series.
  Solar azimuth angle.

Returns

float or Series. Angle of incidence in degrees.

pvlib.irradiance.aoi_projection(surface_tilt, surface_azimuth, solar_zenith, solar_azimuth)
Calculates the dot product of the solar vector and the surface normal.

Input all angles in degrees.

Parameters

- **surface_tilt**: float or Series.
  Panel tilt from horizontal.

- **surface_azimuth**: float or Series.
  Panel azimuth from north.

- **solar_zenith**: float or Series.
  Solar zenith angle.

- **solar_azimuth**: float or Series.
  Solar azimuth angle.

Returns

float or Series. Dot product of panel normal and solar angle.

pvlib.irradiance.beam_component(surface_tilt, surface_azimuth, solar_zenith, solar_azimuth, dni)
Calculates the beam component of the plane of array irradiance.

Parameters

- **surface_tilt**: float or Series.
  Panel tilt from horizontal.

- **surface_azimuth**: float or Series.
  Panel azimuth from north.

- **solar_zenith**: float or Series.
  Solar zenith angle.

- **solar_azimuth**: float or Series.
  Solar azimuth angle.

- **dni**: float or Series
  Direct Normal Irradiance
Returns Series

`pvlib.irradiance.cloudy_day_check(zenith, cloud_prct, pressure=101325.0)`
Determine if the sky is overcast.

Returns logical: bool
Is the sky is overcast.

References


`pvlib.irradiance.dirint(ghi, zenith, times, pressure=101325, use_delta_kt_prime=True, temp_dew=None)`
Determine DNI from GHI using the DIRINT modification of the DISC model.

Implements the modified DISC model known as “DIRINT” introduced in [1]. DIRINT predicts direct normal irradiance (DNI) from measured global horizontal irradiance (GHI). DIRINT improves upon the DISC model by using time-series GHI data and dew point temperature information. The effectiveness of the DIRINT model improves with each piece of information provided.

Parameters

ghi : pd.Series
Global horizontal irradiance in W/m^2.

zenith : pd.Series
True (not refraction-corrected) zenith angles in decimal degrees. If Z is a vector it must be of the same size as all other vector inputs. Z must be >=0 and <=180.

times : DatetimeIndex

pressure : float or pd.Series
The site pressure in Pascal. Pressure may be measured or an average pressure may be calculated from site altitude.

use_delta_kt_prime : bool
Indicates if the user would like to utilize the time-series nature of the GHI measurements. A value of False will not use the time-series improvements, any other numeric value will use time-series improvements. It is recommended that time-series data only be used if the time between measured data points is less than 1.5 hours. If none of the input arguments are vectors, then time-series improvements are not used (because it’s not a time-series).

temp_dew : None, float, or pd.Series
Surface dew point temperatures, in degrees C. Values of temp_dew may be numeric or NaN. Any single time period point with a DewPtTemp=NaN does not have dew point improvements applied. If DewPtTemp is not provided, then dew point improvements are not applied.

Returns dni : pd.Series.
The modeled direct normal irradiance in W/m^2 provided by the DIRINT model.
**References**


DIRINT model requires time series data (i.e. one of the inputs must be a vector of length >2.

`pvlib.irradiance.disc(ghi, zenith, times, pressure=101325)`

Estimate Direct Normal Irradiance from Global Horizontal Irradiance using the DISC model.

The DISC algorithm converts global horizontal irradiance to direct normal irradiance through empirical relationships between the global and direct clearness indices.

**Parameters**

- **ghi**: Series
  - Global horizontal irradiance in \( \text{W/m}^2 \).

- **solar_zenith**: Series
  - True (not refraction - corrected) solar zenith angles in decimal degrees.

- **times**: DatetimeIndex

- **pressure**: float or Series
  - Site pressure in Pascal.

**Returns**

Dataframe with the following keys:

- **dni**: The modeled direct normal irradiance in \( \text{W/m}^2 \) provided by the Direct Insolation Simulation Code (DISC) model.

- **kt**: Ratio of global to extraterrestrial irradiance on a horizontal plane.

- **airmass**: Airmass

**See also:**

`atmosphere.alt2pres, dirint`

**References**


`pvlib.irradiance.extraradiation(date_or_doy, solar_constant=1366.1, method='spencer')`

Determine extraterrestrial radiation from day of year.

**Parameters**

- **datetime_or_doy**: int, float, array, pd.DatetimeIndex
  - Day of year, array of days of year e.g. pd.DatetimeIndex.dayofyear, or pd.DatetimeIndex.

- **solar_constant**: float
  - The solar constant.

- **method**: string
The method by which the ET radiation should be calculated. Options include 'pyephem', 'spencer', 'asce'.

**Returns** float or Series

The extraterrestrial radiation present in watts per square meter on a surface which is normal to the sun. Ea is of the same size as the input doy.

'pyephem’ always returns a series.

**See also:**

`pvlib.clearsky.disc`

**Notes**

The Spencer method contains a minus sign discrepancy between equation 12 of [1]. It’s unclear what the correct formula is.

**References**


`pvlib.irradiance.globalinplane(aoi, dni, poa_sky_diffuse, poa_ground_diffuse)`

Determine the three components on in-plane irradiance

Combines in-plane irradiance components from the chosen diffuse translation, ground reflection and beam irradiance algorithms into the total in-plane irradiance.

**Parameters**

- **aoi** : float or Series
  Angle of incidence of solar rays with respect to the module surface, from `aoi()`.

- **dni** : float or Series
  Direct normal irradiance (W/m²), as measured from a TMY file or calculated with a clearsky model.

- **poa_sky_diffuse** : float or Series
  Diffuse irradiance (W/m²) in the plane of the modules, as calculated by a diffuse irradiance translation function

- **poa_ground_diffuse** : float or Series
  Ground reflected irradiance (W/m²) in the plane of the modules, as calculated by an albedo model (eg. `grounddiffuse()`)

**Returns** DataFrame with the following keys:

- **poa_global** : Total in-plane irradiance (W/m²)
- **poa_direct** : Total in-plane beam irradiance (W/m²)
- **poa_diffuse** : Total in-plane diffuse irradiance (W/m²)
Notes

Negative beam irradiation due to aoi > 90° or AOI < 0° is set to zero.

**pvlib.irradiance.grounddiffuse** *(surface_tilt, ghi, albedo=0.25, surface_type=None)*

Estimate diffuse irradiance from ground reflections given irradiance, albedo, and surface tilt

Function to determine the portion of irradiance on a tilted surface due to ground reflections. Any of the inputs may be DataFrames or scalars.

**Parameters**

- **surface_tilt**: float or DataFrame
  Surface tilt angles in decimal degrees. SurfTilt must be >=0 and <=180. The tilt angle is defined as degrees from horizontal (e.g. surface facing up = 0, surface facing horizon = 90).

- **ghi**: float or DataFrame
  Global horizontal irradiance in W/m^2.

- **albedo**: float or DataFrame
  Ground reflectance, typically 0.1-0.4 for surfaces on Earth (land), may increase over snow, ice, etc. May also be known as the reflection coefficient. Must be >=0 and <=1. Will be overridden if surface_type is supplied.

- **surface_type**: None or string in

**Returns**

- float or DataFrame
  Ground reflected irradiances in W/m^2.

**References**


The calculation is the last term of equations 3, 4, 7, 8, 10, 11, and 12.


**pvlib.irradiance.haydavies** *(surface_tilt, surface_azimuth, dni, dhi, dni_extra, solar_zenith=None, solar_azimuth=None, projection_ratio=None)*

Determine diffuse irradiance from the sky on a tilted surface using Hay & Davies’ 1980 model

\[ I_d = DHI(A R_b + (1 - A)(\frac{1 + \cos \beta}{2})) \]

Hay and Davies’ 1980 model determines the diffuse irradiance from the sky (ground reflected irradiance is not included in this algorithm) on a tilted surface using the surface tilt angle, surface azimuth angle, diffuse horizontal irradiance, direct normal irradiance, extraterrestrial irradiance, sun zenith angle, and sun azimuth angle.

**Parameters**

- **surface_tilt**: float or Series
Surface tilt angles in decimal degrees. The tilt angle is defined as degrees from horizontal (e.g. surface facing up = 0, surface facing horizon = 90)

**surface_azimuth**: float or Series

Surface azimuth angles in decimal degrees. The azimuth convention is defined as degrees east of north (e.g. North=0, South=180, East=90, West=270).

**dhi**: float or Series

Diffuse horizontal irradiance in W/m^2.

**dni**: float or Series

Direct normal irradiance in W/m^2.

**dni_extra**: float or Series

Extraterrestrial normal irradiance in W/m^2.

**solar_zenith**: None, float or Series

Solar apparent (refraction-corrected) zenith angles in decimal degrees. Must supply solar_zenith and solar_azimuth or supply projection_ratio.

**solar_azimuth**: None, float or Series

Solar azimuth angles in decimal degrees. Must supply solar_zenith and solar_azimuth or supply projection_ratio.

**projection_ratio**: None, float or Series

Ratio of angle of incidence projection to solar zenith angle projection. Must supply solar_zenith and solar_azimuth or supply projection_ratio.

**Returns** **sky_diffuse**: float or Series

The diffuse component of the solar radiation on an arbitrarily tilted surface defined by the Perez model as given in reference [3]. Does not include the ground reflected irradiance or the irradiance due to the beam.

**References**


**pvlib.irradiance.isotropic(surface_tilt, dhi)**

Determine diffuse irradiance from the sky on a tilted surface using the isotropic sky model.

\[ I_d = DHI \left(1 + \frac{\cos \beta}{2}\right) \]

Hottel and Woertz’s model treats the sky as a uniform source of diffuse irradiance. Thus the diffuse irradiance from the sky (ground reflected irradiance is not included in this algorithm) on a tilted surface can be found from the diffuse horizontal irradiance and the tilt angle of the surface.

**Parameters** **surface_tilt**: float or Series
Surface tilt angle in decimal degrees. surface_tilt must be >=0 and <=180. The tilt angle is defined as degrees from horizontal (e.g. surface facing up = 0, surface facing horizon = 90)

\[ \text{dhi} : \text{float or Series} \]

Diffuse horizontal irradiance in W/m^2. DHI must be >=0.

Returns float or Series

The diffuse component of the solar radiation on an arbitrarily tilted surface defined by the isotropic sky model as given in Loutzenhiser et. al (2007) equation 3.

SkyDiffuse is the diffuse component ONLY and does not include the ground reflected irradiance or the irradiance due to the beam.

SkyDiffuse is a column vector vector with a number of elements equal to the input vector(s).

References


pvlib.irradiance.king (surface_tilt, dhi, ghi, solar_zenith)

Determine diffuse irradiance from the sky on a tilted surface using the King model.

King’s model determines the diffuse irradiance from the sky (ground reflected irradiance is not included in this algorithm) on a tilted surface using the surface tilt angle, diffuse horizontal irradiance, global horizontal irradiance, and sun zenith angle. Note that this model is not well documented and has not been published in any fashion (as of January 2012).

Parameters surface_tilt : float or Series

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

\[ \text{dhi} : \text{float or Series} \]

Diffuse horizontal irradiance in W/m^2.

\[ \text{ghi} : \text{float or Series} \]

Global horizontal irradiance in W/m^2.

\[ \text{solar_zenith} : \text{float or Series} \]

Apparent (refraction-corrected) zenith angles in decimal degrees.

Returns poa_sky_diffuse : float or Series

The diffuse component of the solar radiation on an arbitrarily tilted surface as given by a model developed by David L. King at Sandia National Laboratories.

\[ I_d = DHI \frac{1 + \cos \beta}{2} (1 + F' \sin^3(\beta/2))(1 + F' \cos^2 \theta \sin^3 \theta_z) \]

2.4. Modules 39
where

\[ F' = 1 - \left( \frac{I_{d0}}{GHI} \right) \]

Klucher’s 1979 model determines the diffuse irradiance from the sky (ground reflected irradiance is not included in this algorithm) on a tilted surface using the surface tilt angle, surface azimuth angle, diffuse horizontal irradiance, direct normal irradiance, global horizontal irradiance, extraterrestrial irradiance, sun zenith angle, and sun azimuth angle.

**Parameters**

- **surface_tilt**: float or Series
  
  Surface tilt angles in decimal degrees. surface_tilt must be >=0 and <=180. The tilt angle is defined as degrees from horizontal (e.g. surface facing up = 0, surface facing horizon = 90)

- **surface_azimuth**: float or Series
  
  Surface azimuth angles in decimal degrees. surface_azimuth must be >=0 and <=360. The Azimuth convention is defined as degrees east of north (e.g. North = 0, South=180 East = 90, West = 270).

- **dhi**: float or Series
  
  Diffuse horizontal irradiance in W/m^2. DHI must be >=0.

- **ghi**: float or Series
  
  Global irradiance in W/m^2. DNI must be >=0.

- **solar_zenith**: float or Series
  
  Apparent (refraction-corrected) zenith angles in decimal degrees. solar_zenith must be >=0 and <=180.

- **solar_azimuth**: float or Series
  
  Sun azimuth angles in decimal degrees. solar_azimuth must be >=0 and <=360. The Azimuth convention is defined as degrees east of north (e.g. North = 0, East = 90, West = 270).

**Returns** float or Series.

The diffuse component of the solar radiation on an arbitrarily tilted surface defined by the Klucher model as given in Loutzenhiser et al (2007) equation 4.

SkyDiffuse is the diffuse component ONLY and does not include the ground reflected irradiance or the irradiance due to the beam.

SkyDiffuse is a column vector vector with a number of elements equal to the input vector(s).

**References**


pvlib.irradiance.liujordan(zenith, cloud_prct, pressure=101325.0)

Determine DNI, DHI, GHI from extraterrestrial flux, transmittance, and optical air mass number.

Liu and Jordan, 1960, developed a simplified direct radiation model. DHI is from an empirical equation for diffuse radiation from Liu and Jordan, 1960.

**Parameters**
- **zenith**: pd.Series
  - True (not refraction-corrected) zenith angles in decimal degrees. If Z is a vector it must be of the same size as all other vector inputs. Z must be >=0 and <=180.
- **cloud_prct**: integer or float
  - Cloud coverage in percentage, %.

**Returns**
- Pandas.DataFrame
  - Modeled direct normal irradiance, direct horizontal irradiance, and global horizontal irradiance in W/m^2

**References**


pvlib.irradiance.perez(surface_tilt, surface_azimuth, dhi, dni, dni_extra, solar_zenith, solar_azimuth, airmass, modelt='allsitescomposite1990')

Determine diffuse irradiance from the sky on a tilted surface using one of the Perez models.

Perez models determine the diffuse irradiance from the sky (ground reflected irradiance is not included in this algorithm) on a tilted surface using the surface tilt angle, surface azimuth angle, diffuse horizontal irradiance, direct normal irradiance, extraterrestrial irradiance, sun zenith angle, sun azimuth angle, and relative (not pressure-corrected) airmass. Optionally a selector may be used to use any of Perez’s model coefficient sets.

**Parameters**
- **surface_tilt**: float or Series
  - Surface tilt angles in decimal degrees. surface_tilt must be >=0 and <=180. The tilt angle is defined as degrees from horizontal (e.g. surface facing up = 0, surface facing horizon = 90)
- **surface_azimuth**: float or Series
  - Surface azimuth angles in decimal degrees. surface_azimuth must be >=0 and <=360. The Azimuth convention is defined as degrees east of north (e.g. North = 0, South=180 East = 90, West = 270).
- **dhi**: float or Series
  - Diffuse horizontal irradiance in W/m^2. DHI must be >=0.
- **dni**: float or Series
  - Direct normal irradiance in W/m^2. DNI must be >=0.
- **dni_extra**: float or Series
  - Extraterrestrial normal irradiance in W/m^2.
- **solar_zenith**: float or Series
apparent (refraction-corrected) zenith angles in decimal degrees. solar_zenith must be >=0 and <=180.

**solar_zenith** : float or Series

Sun azimuth angles in decimal degrees. solar_zenith must be >=0 and <=360. The Azimuth convention is defined as degrees east of north (e.g. North = 0, East = 90, West = 270).

**airmass** : float or Series

Relative (not pressure-corrected) airmass values. If AM is a DataFrame it must be of the same size as all other DataFrame inputs. AM must be >=0 (careful using the 1/sec(z) model of AM generation)

**model** : string (optional, default='allsitescomposite1990')

A string which selects the desired set of Perez coefficients. If model is not provided as an input, the default, '1990' will be used. All possible model selections are:

- '1990'
- 'allsitescomposite1990' (same as '1990')
- 'allsitescomposite1988'
- 'sandiacomposite1988'
- 'usacomposite1988'
- 'france1988'
- 'phoenix1988'
- 'elmonte1988'
- 'osage1988'
- 'albuquerque1988'
- 'capecanaveral1988'
- 'albany1988'

**Returns** float or Series

The diffuse component of the solar radiation on an arbitrarily tilted surface defined by the Perez model as given in reference [3]. SkyDiffuse is the diffuse component ONLY and does not include the ground reflected irradiance or the irradiance due to the beam.

**References**


pvlib.irradiance.poa_horizontal_ratio(surface_tilt, surface_azimuth, solar_zenith, solar_azimuth)

Calculates the ratio of the beam components of the plane of array irradiance and the horizontal irradiance. Input all angles in degrees.

Parameters

- **surface_tilt**: float or Series.
  Panel tilt from horizontal.
- **surface_azimuth**: float or Series.
  Panel azimuth from north.
- **solar_zenith**: float or Series.
  Solar zenith angle.
- **solar_azimuth**: float or Series.
  Solar azimuth angle.

Returns

- float or Series. Ratio of the plane of array irradiance to the horizontal plane irradiance

pvlib.irradiance.reindl(surface_tilt, surface_azimuth, dhi, dni, ghi, dni_extra, solar_zenith, solar_azimuth)

Determine diffuse irradiance from the sky on a tilted surface using Reindl’s 1990 model

\[ I_d = DHI(AR_b + (1 - A)(\frac{1 + \cos \beta}{2})(1 + \sqrt{\frac{I_{hb}}{I_h} \sin^3(\beta/2)}) \]  

Reindl’s 1990 model determines the diffuse irradiance from the sky (ground reflected irradiance is not included in this algorithm) on a tilted surface using the surface tilt angle, surface azimuth angle, diffuse horizontal irradiance, direct normal irradiance, global horizontal irradiance, extraterrestrial irradiance, sun zenith angle, and sun azimuth angle.

Parameters

- **surface_tilt**: float or Series.
  Surface tilt angles in decimal degrees. The tilt angle is defined as degrees from horizontal (e.g. surface facing up = 0, surface facing horizon = 90)
- **surface_azimuth**: float or Series.
  Surface azimuth angles in decimal degrees. The Azimuth convention is defined as degrees east of north (e.g. North = 0, South = 180 East = 90, West = 270).
- **dhi**: float or Series.
  Diffuse horizontal irradiance in W/m^2.
- **dni**: float or Series.
  Direct normal irradiance in W/m^2.
- **ghi**: float or Series.
  Global irradiance in W/m^2.
- **dni_extra**: float or Series.
  Extraterrestrial normal irradiance in W/m^2.
- **solar_zenith**: float or Series.
  Apparent (refraction-corrected) zenith angles in decimal degrees.
solar_azimuth : float or Series.

Sun azimuth angles in decimal degrees. The Azimuth convention is defined as degrees east of north (e.g. North = 0, East = 90, West = 270).

Returns poa_sky_diffuse : float or Series.

The diffuse component of the solar radiation on an arbitrarily tilted surface defined by the Reindl model as given in Loutzenhiser et. al (2007) equation 8. SkyDiffuse is the diffuse component ONLY and does not include the ground reflected irradiance or the irradiance due to the beam. SkyDiffuse is a column vector vector with a number of elements equal to the input vector(s).

Notes

The poa_sky_diffuse calculation is generated from the Loutzenhiser et al. (2007) paper, equation 8. Note that I have removed the beam and ground reflectance portion of the equation and this generates ONLY the diffuse radiation from the sky and circumsolar, so the form of the equation varies slightly from equation 8.

References


pvlib.irradiance.total_irrad(surface_tilt, surface_azimuth, solar_zenith, solar_azimuth, dni, ghi, dhi, dni_extra=None, airmass=None, albedo=0.25, surface_type=None, model='isotropic', model_perez='allsitescomposite1990')

Determine diffuse irradiance from the sky on a tilted surface.

\[ I_{tot} = I_{beam} + I_{sky} + I_{ground} \]

Parameters surface_tilt : float or Series.

Panel tilt from horizontal.

surface_azimuth : float or Series.

Panel azimuth from north.

solar_zenith : float or Series.

Solar zenith angle.

solar_azimuth : float or Series.

Solar azimuth angle.

dni : float or Series

Direct Normal Irradiance

ghi : float or Series

Global horizontal irradiance

dhi : float or Series
Diffuse horizontal irradiance

\texttt{dni\_extra} : float or Series

Extraterrestrial direct normal irradiance

\texttt{airmass} : float or Series

Airmass

\texttt{albedo} : float

Surface albedo

\texttt{surface\_type} : String

Surface type. See grounddiffuse.

\texttt{model} : String

Irradiance model.

\texttt{model\_perez} : String

See perez.

\textbf{Returns} DataFrame with columns ‘total’, ‘beam’, ‘sky’, ‘ground’.

\section*{References}


\section*{2.4.5 location module}

This module contains the Location class.

\begin{verbatim}
class pvlib.location.Location (latitude, longitude, tz='US/Mountain', altitude=100, name=None)
Bases: object

Location objects are convenient containers for latitude, longitude, timezone, and altitude data associated with a particular geographic location. You can also assign a name to a location object.

Location objects have two timezone attributes:

• \texttt{location.tz} is a IANA timezone string.
• \texttt{location.pytz} is a pytz timezone object.

Location objects support the print method.

\textbf{Parameters} latitude : float.

Positive is north of the equator. Use decimal degrees notation.

longitude : float.

Positive is east of the prime meridian. Use decimal degrees notation.

tz : string or pytz.timezone.

See http://en.wikipedia.org/wiki/List_of_tz_database_time_zones for a list of valid time zones. pytz.timezone objects will be converted to strings.

altitude : float.
\end{verbatim}
Altitude from sea level in meters.

**name**: None or string.

Sets the name attribute of the Location object.

## 2.4.6 pvsystem module

The `pvsystem` module contains functions for modeling the output and performance of PV modules and inverters.

```python
pvlib.pvsystem.ashraeiam(b, aoi)
```

Determine the incidence angle modifier using the ASHRAE transmission model.

ashraeiam calculates the incidence angle modifier as developed in [1], and adopted by ASHRAE (American Society of Heating, Refrigeration, and Air Conditioning Engineers) [2]. The model has been used by model programs such as PVSyst [3].

Note: For incident angles near 90 degrees, this model has a discontinuity which has been addressed in this function.

**Parameters**

- `b` : float
  - A parameter to adjust the modifier as a function of angle of incidence. Typical values are on the order of 0.05 [3].

- `aoi` : Series
  - The angle of incidence between the module normal vector and the sun-beam vector in degrees.

**Returns**

- `IAM` : Series
  - The incident angle modifier calculated as $1 - b \times (\sec(aoi) - 1)$ as described in [2,3].

  Returns nan for all $\text{abs}(aoi) \geq 90$ and for all IAM values that would be less than 0.

**See also:**

- `irradiance.aoi`
- `physicaliam`

**References**

- [2] ASHRAE standard 93-77

```python
pvlib.pvsystem.calcparams_desoto(poa_global, temp_cell, alpha_isc, module_parameters, EgRef, dEgdT, M=1, irrad_ref=1000, temp_ref=25)
```

Applies the temperature and irradiance corrections to inputs for singlediode.

Applies the temperature and irradiance corrections to the IL, I0, Rs, Rsh, and a parameters at reference conditions (IL_ref, I0_ref, etc.) according to the De Soto et. al description given in [1]. The results of this correction procedure may be used in a single diode model to determine IV curves at irradiance = S, cell temperature = Tcell.

**Parameters**

- `poa_global` : float or Series
  - The irradiance (in W/m^2) absorbed by the module.
**temp_cell**: float or Series

The average cell temperature of cells within a module in C.

**alpha_isc**: float

The short-circuit current temperature coefficient of the module in units of 1/C.

**module_parameters**: dict

Parameters describing PV module performance at reference conditions according to DeSoto's paper. Parameters may be generated or found by lookup. For ease of use, retrieve_sam can automatically generate a dict based on the most recent SAM CEC module database. The module_parameters dict must contain the following 5 fields:

- **a_ref**: modified diode ideality factor parameter at reference conditions (units of eV). 
  a_ref can be calculated from the usual diode ideality factor (n), number of cells in series (Ns), and cell temperature (Tcell) per equation (2) in [1].
- **I_L_ref**: Light-generated current (or photocurrent) in amperes at reference conditions. This value is referred to as Iph in some literature.
- **I_o_ref**: diode reverse saturation current in amperes, under reference conditions.
- **R_sh_ref**: shunt resistance under reference conditions (ohms).
- **R_s**: series resistance under reference conditions (ohms).

**EgRef**: float

The energy bandgap at reference temperature (in eV). 1.121 eV for silicon. EgRef must be >0.

**dEgdT**: float

The temperature dependence of the energy bandgap at SRC (in 1/C). May be either a scalar value (e.g. -0.0002677 as in [1]) or a DataFrame of dEgdT values corresponding to each input condition (this may be useful if dEgdT is a function of temperature).

**M**: float or Series (optional, default=1)

An optional airmass modifier, if omitted, M is given a value of 1, which assumes absolute (pressure corrected) airmass = 1.5. In this code, M is equal to M/Mref as described in [1] (i.e. Mref is assumed to be 1). Source [1] suggests that an appropriate value for M as a function absolute airmass (AMa) may be:

```python
>>> M = np.polyval([-0.000126, 0.002816, -0.024459, 0.086257, 0.918093], AMa)
```

M may be a Series.

**irrad_ref**: float (optional, default=1000)

Reference irradiance in W/m^2.

**temp_ref**: float (optional, default=25)

Reference cell temperature in C.

**Returns** Tuple of the following results:

- **photocurrent**: float or Series
  Light-generated current in amperes at irradiance=S and cell temperature=Tcell.
- **saturation_current**: float or Series

2.4. Modules
Diode saturation current in amperes at irradiance $S$ and cell temperature $T_{cell}$.

**resistance_series**: float

Series resistance in ohms at irradiance $S$ and cell temperature $T_{cell}$.

**resistance_shunt**: float or Series

Shunt resistance in ohms at irradiance $S$ and cell temperature $T_{cell}$.

**nNsVth**: float or Series

Modified diode ideality factor at irradiance $S$ and cell temperature $T_{cell}$. Note that in source [1] $nNsVth = a$ (equation 2). $nNsVth$ is the product of the usual diode ideality factor ($n$), the number of series-connected cells in the module ($Ns$), and the thermal voltage of a cell in the module ($Vth$) at a cell temperature of $T_{cell}$.

See also:
sapm, sapm_celltemp, singlediode, retrieve_sam

Notes

If the reference parameters in the ModuleParameters struct are read from a database or library of parameters (e.g. System Advisor Model), it is important to use the same $EgRef$ and $dEgdT$ values that were used to generate the reference parameters, regardless of the actual bandgap characteristics of the semiconductor. For example, in the case of the System Advisor Model library, created as described in [3], $EgRef$ and $dEgdT$ for all modules were 1.121 and -0.0002677, respectively.

This table of reference bandgap energies ($EgRef$), bandgap energy temperature dependence ($dEgdT$), and “typical” airmass response ($M$) is provided purely as reference to those who may generate their own reference module parameters ($a_{ref}$, $I_{L_{ref}}$, $I_{0_{ref}}$, etc.) based upon the various PV semiconductors. Again, we stress the importance of using identical $EgRef$ and $dEgdT$ when generating reference parameters and modifying the reference parameters (for irradiance, temperature, and airmass) per DeSoto’s equations.

**Silicon (Si):**

- $EgRef = 1.121$
- $dEgdT = -0.0002677$

```python
>>> M = np.polyval([-1.26E-4, 2.816E-3, -0.024459, 0.086257, 0.918093], AMa)
```

Source: [1]

**Cadmium Telluride (CdTe):**

- $EgRef = 1.475$
- $dEgdT = -0.0003$

```python
>>> M = np.polyval([-2.46E-5, 9.607E-4, -0.0134, 0.0716, 0.9196], AMa)
```

Source: [4]

**Copper Indium diSelenide (CIS):**

- $EgRef = 1.010$
- $dEgdT = -0.00011$
```
>>> M = np.polyval([-3.74E-5, 0.00125, -0.01462, 0.0718, 0.9210],
                  AMa)
```

Source: [4]

**Copper Indium Gallium diSelenide (CIGS):**

- \(E_{g,\text{Ref}} = 1.15\)
- \(dE_g/dT = ????\)

```
>>> M = np.polyval([-9.07E-5, 0.0022, -0.0202, 0.0652, 0.9417],
                  AMa)
```


**Gallium Arsenide (GaAs):**

- \(E_{g,\text{Ref}} = 1.424\)
- \(dE_g/dT = -0.000433\)
- \(M = \text{unknown}\)

Source: [4]

**References**


```
pvlib.pvsystem.i_from_v(resistance_shunt, resistance_series, nNsVth, voltage, saturation_current, photocurrent)
```

Calculates current from voltage per Eq 2 Jain and Kapoor 2004 [1].

**Parameters**

- **resistance_series**: float or Series
  Series resistance in ohms under desired IV curve conditions. Often abbreviated \(R_s\).

- **resistance_shunt**: float or Series
  Shunt resistance in ohms under desired IV curve conditions. Often abbreviated \(R_{sh}\).

- **saturation_current**: float or Series
  Diode saturation current in amperes under desired IV curve conditions. Often abbreviated \(I_0\).

- **nNsVth**: float or Series
  The product of three components. 1) The usual diode ideal factor (n), 2) the number of cells in series (N_s), and 3) the cell thermal voltage under the desired IV curve conditions (V_{th}). The thermal voltage of the cell (in volts) may be calculated as \(k*\text{temp}_{\text{cell}}/q\), where \(k\) is Boltzmann’s constant (J/K), \(\text{temp}_{\text{cell}}\) is the temperature of the p-n junction in Kelvin, and \(q\) is the charge of an electron (coulombs).

---

2.4. Modules 49
**photocurrent** : float or Series

Light-generated current (photocurrent) in amperes under desired IV curve conditions. Often abbreviated \( I_L \).

**Returns current** : np.array

**References**


```python
pvlib.pvsystem.physicaliam(K, L, n, aoi)
```

Determine the incidence angle modifier using refractive index, glazing thickness, and extinction coefficient.

physicaliam calculates the incidence angle modifier as described in De Soto et al. “Improvement and validation of a model for photovoltaic array performance”, section 3. The calculation is based upon a physical model of absorption and transmission through a cover. Required information includes, incident angle, cover extinction coefficient, cover thickness.

Note: The authors of this function believe that eqn. 14 in [1] is incorrect. This function uses the following equation in its place: \( \theta_r = \arcsin\left(\frac{1}{n} \times \sin(\theta)\right) \)

**Parameters**

- **K** : float
  
The glazing extinction coefficient in units of 1/meters. Reference [1] indicates that a value of 4 is reasonable for “water white” glass. K must be a numeric scalar or vector with all values \( \geq 0 \). If K is a vector, it must be the same size as all other input vectors.

- **L** : float
  
The glazing thickness in units of meters. Reference [1] indicates that 0.002 meters (2 mm) is reasonable for most glass-covered PV panels. L must be a numeric scalar or vector with all values \( \geq 0 \). If L is a vector, it must be the same size as all other input vectors.

- **n** : float
  
The effective index of refraction (unitless). Reference [1] indicates that a value of 1.526 is acceptable for glass. n must be a numeric scalar or vector with all values \( \geq 0 \). If n is a vector, it must be the same size as all other input vectors.

- **aoi** : Series
  
The angle of incidence between the module normal vector and the sun-beam vector in degrees.

**Returns**

- **IAM** : float or Series
  
The incident angle modifier as specified in eqns. 14-16 of [1]. IAM is a column vector with the same number of elements as the largest input vector.

Theta must be a numeric scalar or vector. For any values of theta where \( \text{abs}(\text{aoi}) \geq 90 \), IAM is set to 0. For any values of aoi where \(-90 < \text{aoi} < 0\), theta is set to \( |\text{aoi}| \) and evaluated.

**See also:**

getaoi, ephemeris, spa, ashraeiam
References


`pvlib.pvsystem.retrieve_sam(name=None, samfile=None)`

Retrieve latest module and inverter info from SAM website.

This function will retrieve either:

- CEC module database
- Sandia Module database
- CEC Inverter database

and return it as a pandas dataframe.

Parameters

- **name** : String
  
  Name can be one of:
  
  - ‘CECMod’ - returns the CEC module database
  - ‘CECInverter’ - returns the CEC Inverter database
  - ‘SandiaInverter’ - returns the CEC Inverter database (CEC is only current inverter db available; tag kept for backwards compatibility)
  - ‘SandiaMod’ - returns the Sandia Module database

- **samfile** : String

  Absolute path to the location of local versions of the SAM file. If file is specified, the latest versions of the SAM database will not be downloaded. The selected file must be in .csv format.

  If set to ‘select’, a dialogue will open allowing the user to navigate to the appropriate page.

Returns

A DataFrame containing all the elements of the desired database.

Each column represents a module or inverter, and a specific dataset can be retrieved by the command

Examples

```python
>>> from pvlib import pvsystem
>>> invdb = pvsystem.retrieve_sam(name='CECInverter')
>>> inverter = invdb.AE_Solar_Energy__AE6_0__277V__277V__CEC_2012_
>>> inverter
Vac    277.000000
Paco   6000.000000
Pdco   6165.670000
Vdco   361.123000
Pso    36.792300
C0     -0.000002
C1     -0.000047
C2     -0.001861
```

2.4. Modules
pvlib.pvsystem.sapm(module, poa_direct, poa_diffuse, temp_cell, airmass_absolute, aoi)

The Sandia PV Array Performance Model (SAPM) generates 5 points on a PV module's I-V curve (Voc, Isc, Ix, Ixx, Vmp/Imp) according to SAND2004-3535. Assumes a reference cell temperature of 25 C.

**Parameters**

- **module**: Series or dict
  
  A DataFrame defining the SAPM performance parameters.

- **poa_direct**: Series
  
  The direct irradiance incident upon the module (W/m^2).

- **poa_diffuse**: Series
  
  The diffuse irradiance incident on module.

- **temp_cell**: Series
  
  The cell temperature (degrees C).

- **airmass_absolute**: Series
  
  Absolute airmass.

- **aoi**: Series
  
  Angle of incidence (degrees).

**Returns**

A DataFrame with the columns:

- **i_sc**: Short-circuit current (A)
- **I_mp**: Current at the maximum-power point (A)
- **v_oc**: Open-circuit voltage (V)
- **v_mp**: Voltage at maximum-power point (V)
- **p_mp**: Power at maximum-power point (W)
- **i_x**: Current at module V = 0.5Voc, defines 4th point on I-V curve for modeling curve shape
- **i_xx**: Current at module V = 0.5(Voc+Vmp), defines 5th point on I-V curve for modeling curve shape
- **effective_irradiance**: Effective irradiance

**See also:**

`retrieve_sam`, `sapm_celltemp`

**Notes**

The coefficients from SAPM which are required in `module` are:
References


```
import pvlib.pvsystem

pvlib.pvsystem.sapm_celltemp(irrad, wind, temp, model='open_rack_cell_glassback')
```

Estimate cell and module temperatures per the Sandia PV Array Performance Model (SAPM, SAND2004-3535), from the incident irradiance, wind speed, ambient temperature, and SAPM module parameters.

**Parameters**

- **irrad**: float or Series
  Total incident irradiance in W/m^2.

- **wind**: float or Series
  Wind speed in m/s at a height of 10 meters.

- **temp**: float or Series
  Ambient dry bulb temperature in degrees C.

- **model**: string or list
  Model to be used.
  If string, can be:
  - `open_rack_cell_glassback` (default)
  - `roof_mount_cell_glassback`
  - `open_rack_cell_polymerback`
  - `insulated_back_polymerback`
  - `open_rack_polymer_thinfilm_steel`
  - `22x_concentrator_tracker`
  If list, supply the following parameters in the following order:
  - **a** [float] SAPM module parameter for establishing the upper limit for module temperature at low wind speeds and high solar irradiance.
  - **b** [float] SAPM module parameter for establishing the rate at which the module temperature drops as wind speed increases (see SAPM eqn. 11).
  - **deltaT** [float] SAPM module parameter giving the temperature difference between the cell and module back surface at the reference irradiance, E0.

**Returns**

DataFrame with columns ‘temp_cell’ and ‘temp_module’.
Values in degrees C.

See also:

- sapm

References

Solve the single-diode model to obtain a photovoltaic IV curve.

Singlediode solves the single diode equation [1]

\[ I = I_L - I_0 * \left( \exp\left(\frac{(V + I * R_s)}{(nN_sV_{th})}\right) - 1 \right) - \frac{(V + I * R_s)}{R_{sh}} \]

for I and V when given \( I_L, I_0, R_s, R_{sh}, \) and \( nN_sV_{th} \) which are described later. Returns a DataFrame which contains the 5 points on the I-V curve specified in SAND2004-3535 [3]. If all \( I_L, I_0, R_s, R_{sh}, \) and \( nN_sV_{th} \) are scalar, a single curve will be returned, if any are Series (of the same length), multiple IV curves will be calculated.

The input parameters can be calculated using calcparams_desoto from meteorological data.

**Parameters**

- **module**: DataFrame
  A DataFrame defining the SAPM performance parameters.

- **photocurrent**: float or Series
  Light-generated current (photocurrent) in amperes under desired IV curve conditions. Often abbreviated \( I_L \).

- **saturation_current**: float or Series
  Diode saturation current in amperes under desired IV curve conditions. Often abbreviated \( I_0 \).

- **resistance_series**: float or Series
  Series resistance in ohms under desired IV curve conditions. Often abbreviated \( R_s \).

- **resistance_shunt**: float or Series
  Shunt resistance in ohms under desired IV curve conditions. Often abbreviated \( R_{sh} \).

- **nNsVth**: float or Series
  The product of three components. 1) The usual diode ideal factor (n), 2) the number of cells in series (Ns), and 3) the cell thermal voltage under the desired IV curve conditions (Vth). The thermal voltage of the cell (in volts) may be calculated as \( \frac{k*temp\_cell}{q} \), where \( k \) is Boltzmann’s constant (J/K), \( temp\_cell \) is the temperature of the p-n junction in Kelvin, and \( q \) is the charge of an electron (coulombs).

**Returns**

If **photocurrent** is a Series, a DataFrame with the following columns.

All columns have the same number of rows as the largest input DataFrame.

If **photocurrent** is a scalar, a dict with the following keys.

- **i_sc** - short circuit current in amperes.
- **v_oc** - open circuit voltage in volts.
- **i_mp** - current at maximum power point in amperes.
- **v_mp** - voltage at maximum power point in volts.
- **p_mp** - power at maximum power point in watts.
- **i_x** - current, in amperes, at \( V = 0.5*v_{oc} \).
- **i_xx** - current, in amperes, at \( V = 0.5*(v_{oc}+v_{mp}) \).

**See also:**

sapm, calcparams_desoto
Notes

The solution employed to solve the implicit diode equation utilizes the Lambert W function to obtain an explicit function of \( V = f(i) \) and \( I = f(V) \) as shown in [2].

References


\texttt{pvlib.pvsystem.snlinverter (inverter, v\_dc, p\_dc)}

Converts DC power and voltage to AC power using Sandia’s Grid-Connected PV Inverter model. Determines the AC power output of an inverter given the DC voltage, DC power, and appropriate Sandia Grid-Connected Photovoltaic Inverter Model parameters. The output, ac\_power, is clipped at the maximum power output, and gives a negative power during low-input power conditions, but does NOT account for maximum power point tracking voltage windows nor maximum current or voltage limits on the inverter.

**Parameters**

- \texttt{inverter} : DataFrame
  
  A DataFrame defining the inverter to be used, giving the inverter performance parameters according to the Sandia Grid-Connected Photovoltaic Inverter Model (SAND 2007-5036) [1]. A set of inverter performance parameters are provided with pvlib, or may be generated from a System Advisor Model (SAM) [2] library using retrievesam.

  Required DataFrame columns are:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pac0</td>
<td>AC-power output from inverter based on input power and voltage (W)</td>
</tr>
<tr>
<td>Pdc0</td>
<td>DC-power input to inverter, typically assumed to be equal to the PV array maximum power (W)</td>
</tr>
<tr>
<td>Vdc0</td>
<td>DC-voltage level at which the AC-power rating is achieved at the reference operating condition (V)</td>
</tr>
<tr>
<td>Ps0</td>
<td>DC-power required to start the inversion process, or self-consumption by inverter, strongly influences inverter efficiency at low power levels (W)</td>
</tr>
<tr>
<td>C0</td>
<td>Parameter defining the curvature (parabolic) of the relationship between ac-power and dc-power at the reference operating condition, default value of zero gives a linear relationship (1/W)</td>
</tr>
<tr>
<td>C1</td>
<td>Empirical coefficient allowing Pdc0 to vary linearly with dc-voltage input, default value is zero (1/V)</td>
</tr>
<tr>
<td>C2</td>
<td>Empirical coefficient allowing Ps0 to vary linearly with dc-voltage input, default value is zero (1/V)</td>
</tr>
<tr>
<td>C3</td>
<td>Empirical coefficient allowing C0 to vary linearly with dc-voltage input, default value is zero (1/V)</td>
</tr>
<tr>
<td>Pnt</td>
<td>AC-power consumed by inverter at night (night tare) to maintain circuitry required to sense PV array voltage (W)</td>
</tr>
</tbody>
</table>

- \texttt{v\_dc} : float or Series
  
  DC voltages, in volts, which are provided as input to the inverter. Vdc must be \( \geq 0 \).

- \texttt{p\_dc} : float or Series
A scalar or DataFrame of DC powers, in watts, which are provided as input to the inverter. Pdc must be >= 0.

Returns ac_power : float or Series

Modeled AC power output given the input DC voltage, Vdc, and input DC power, Pdc. When ac_power would be greater than Pac0, it is set to Pac0 to represent inverter “clipping”. When ac_power would be less than Ps0 (startup power required), then ac_power is set to -1*abs(Pnt) to represent nightly power losses. ac_power is not adjusted for maximum power point tracking (MPPT) voltage windows or maximum current limits of the inverter.

See also:
sapm, singlediode

References


pvlib.pvsystem.systemdef (meta, surface_tilt, surface_azimuth, albedo, series_modules, parallel_modules)

Generates a dict of system parameters used throughout a simulation.

Parameters meta : dict

meta dict either generated from a TMY file using readtmy2 or readtmy3, or a dict containing at least the following fields:

<table>
<thead>
<tr>
<th>meta field</th>
<th>format</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>meta.altitude</td>
<td>Float</td>
<td>site elevation</td>
</tr>
<tr>
<td>meta.latitude</td>
<td>Float</td>
<td>site latitude</td>
</tr>
<tr>
<td>meta.longitude</td>
<td>Float</td>
<td>site longitude</td>
</tr>
<tr>
<td>meta.Name</td>
<td>String</td>
<td>site name</td>
</tr>
<tr>
<td>meta.State</td>
<td>String</td>
<td>state</td>
</tr>
<tr>
<td>meta.TZ</td>
<td>Float</td>
<td>timezone</td>
</tr>
</tbody>
</table>

surface_tilt : float or Series

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

surface_azimuth : float or Series

Surface azimuth angles in decimal degrees. The azimuth convention is defined as degrees east of north (North=0, South=180, East=90, West=270).

albedo : float or Series

Ground reflectance, typically 0.1-0.4 for surfaces on Earth (land), may increase over snow, ice, etc. May also be known as the reflection coefficient. Must be >=0 and <=1.

series_modules : int

Number of modules connected in series in a string.

parallel_modules : int

Number of strings connected in parallel.
Returns Result: dict
A dict with the following fields.
• 'surface_tilt'
• 'surface_azimuth'
• 'albedo'
• 'series_modules'
• 'parallel_modules'
• 'latitude'
• 'longitude'
• 'tz'
• 'name'
• 'altitude'

See also:
pvlib.tmy.readtmy3, pvlib.tmy.readtmy2

2.4.7 solarposition module

Calculate the solar position using a variety of methods/packages.

```
pvlib.solarposition.calc_time(lower_bound, upper_bound, location, attribute, value, pressure=101325, temperature=12, xtol=1e-12)
```

Calculate the time between lower_bound and upper_bound where the attribute is equal to value. Uses PyEphem for solar position calculations.

Parameters

- **lower_bound**: datetime.datetime
- **upper_bound**: datetime.datetime
- **location**: pvlib.Location object
- **attribute**: str
  The attribute of a pyephem.Sun object that you want to solve for. Likely options are ‘alt’ and ‘az’ (which must be given in radians).
- **value**: int or float
  The value of the attribute to solve for
- **pressure**: int or float, optional
  Air pressure in Pascals. Set to 0 for no atmospheric correction.
- **temperature**: int or float, optional
  Air temperature in degrees C.
- **xtol**: float, optional
  The allowed error in the result from value

Returns datetime.datetime

Raises ValueError
If the value is not contained between the bounds.

**AttributeError**

If the given attribute is not an attribute of a PyEphem.Sun object.

```python
pvlib.solarposition.ephemeris(time, location, pressure=101325, temperature=12)
```

Python-native solar position calculator. The accuracy of this code is not guaranteed. Consider using the built-in spa_c code or the PyEphem library.

**Parameters**

- **time**: pandas.DatetimeIndex
- **location**: pvlib.Location
- **pressure**: float or Series
  - Ambient pressure (Pascals)
- **temperature**: float or Series
  - Ambient temperature (C)

**Returns**

DataFrame with the following columns:

- `apparent_elevation`: apparent sun elevation accounting for atmospheric refraction.
- `elevation`: actual elevation (not accounting for refraction) of the sun in decimal degrees, 0 = on horizon. The complement of the zenith angle.
- `azimuth`: Azimuth of the sun in decimal degrees East of North. This is the complement of the apparent zenith angle.
- `apparent_zenith`: apparent sun zenith accounting for atmospheric refraction.
- `zenith`: Solar zenith angle
- `solar_time`: Solar time in decimal hours (solar noon is 12.00).

**See also**

`pyephem`, `spa_c`, `spa_python`

**References**


```python
pvlib.solarposition.get_solarposition(time, location, method='nrel_numpy', pressure=101325, temperature=12, **kwargs)
```

A convenience wrapper for the solar position calculators.

**Parameters**

- **time**: pandas.DatetimeIndex
- **location**: pvlib.Location object
- **method**: string
  - ‘pyephem’ uses the PyEphem package: `pyephem()`
  - ‘nrel_c’ uses the NREL SPA C code [3]: `spa_c()`
  - ‘nrel_numpy’ uses an implementation of the NREL SPA algorithm described in [1] (default): `spa_python()`
  - ‘nrel_numba’ uses an implementation of the NREL SPA algorithm described in [1], but also compiles the code first: `spa_python()`
‘ephemeris’ uses the pvlib ephemeris code: `ephemeris()

**pressure** : float
Pascals.

**temperature** : float
Degrees C.

Other keywords are passed to the underlying solar position function.

**References**


```python
pvlib.solarposition.get_sun_rise_set_transit(time, location, how='numpy', delta_t=None, numthreads=4)
```

Calculate the sunrise, sunset, and sun transit times using the NREL SPA algorithm described in [1].

If numba is installed, the functions can be compiled to machine code and the function can be multithreaded. Without numba, the function evaluates via numpy with a slight performance hit.

**Parameters**

- **time** : pandas.DatetimeIndex
  Only the date part is used

- **location** : pvlib.Location object

- **delta_t** : float, optional
  Difference between terrestrial time and UT1. By default, use USNO historical data and predictions

- **how** : str, optional
  Options are ‘numpy’ or ‘numba’. If numba >= 0.17.0 is installed, how=’numba’ will compile the spa functions to machine code and run them multithreaded.

- **numthreads** : int, optional
  Number of threads to use if how == ‘numba’.

**Returns**

DataFrame
The DataFrame will have the following columns: sunrise, sunset, transit

**References**


```python
pvlib.solarposition.pyephem(time, location, pressure=101325, temperature=12)
```

Calculate the solar position using the PyEphem package.
**Parameters**

- **time**: pandas.DatetimeIndex
  - location : pvlib.Location object
  - pressure : int or float, optional
    - air pressure in Pascals.
  - temperature : int or float, optional
    - air temperature in degrees C.

**Returns** DataFrame

The DataFrame will have the following columns: apparent_elevation, elevation, apparent_azimuth, azimuth, apparent_zenith, zenith.

**See also:**

- spa_python, spa_c, ephemeris

**pvlib.solarposition.pyephem_earthsun_distance**(*time*)

Calculates the distance from the earth to the sun using pyephem.

**Parameters**

- **time**: pd.DatetimeIndex

**Returns** pd.Series. Earth-sun distance in AU.

**pvlib.solarposition.spa_c**(*time*, location, pressure=101325, temperature=12, delta_t=67.0, raw_spa_output=False)

Calculate the solar position using the C implementation of the NREL SPA code

The source files for this code are located in './spa_c_files/', along with a README file which describes how the C code is wrapped in Python. Due to license restrictions, the C code must be downloaded separately and used in accordance with its license.

**Parameters**

- **time**: pandas.DatetimeIndex
  - location : pvlib.Location object
  - pressure : float
    - Pressure in Pascals
  - temperature : float
    - Temperature in C
  - delta_t : float
    - Difference between terrestrial time and UT1. USNO has previous values and predictions.
  - raw_spa_output : bool
    - If true, returns the raw SPA output.

**Returns** DataFrame

The DataFrame will have the following columns: elevation, azimuth, zenith, apparent_elevation, apparent_zenith.

**See also:**

- pyephem, spa_python, ephemeris
References

NREL SPA code: http://rredc.nrel.gov/solar/codesandalgorithms/spa/

USNO delta T: http://www.usno.navy.mil/USNO/earth-orientation eo-products/long-term

pvlib.solarposition.spa_python(time, location, pressure=101325, temperature=12, delta_t=None, atmos_refract=None, how='numpy', numthreads=4)

Calculate the solar position using a python implementation of the NREL SPA algorithm described in [1].

If numba is installed, the functions can be compiled to machine code and the function can be multithreaded. Without numba, the function evaluates via numpy with a slight performance hit.

Parameters

- **time**: pandas.DatetimeIndex
- **location**: pvlib.Location object
- **pressure**: int or float, optional
- **temperature**: int or float, optional
- **delta_t**: float, optional
- **atmos_refrac**: float, optional
- **how**: str, optional
- **numthreads**: int, optional

Returns DataFrame

The DataFrame will have the following columns: apparent_zenith (degrees), zenith (degrees), apparent_elevation (degrees), elevation (degrees), azimuth (degrees), equation_of_time (minutes).

See also:

pyephem, spa_c, ephemeris

References


2.4.8 tmy module

Import functions for TMY2 and TMY3 data files.

\texttt{pvlib.tmy.readtmy2(filename)}

Read a TMY2 file into a DataFrame.

Note that values contained in the DataFrame are unchanged from the TMY2 file (i.e. units are retained). Time/Date and location data imported from the TMY2 file have been modified to a “friendlier” form conforming to modern conventions (e.g. N latitude is positive, E longitude is positive, the “24th” hour of any day is technically the “0th” hour of the next day). In the case of any discrepancies between this documentation and the TMY2 User’s Manual [1], the TMY2 User’s Manual takes precedence.

**Parameters**

- **filename**: None or string
  - If None, attempts to use a Tkinter file browser. A string can be a relative file path, absolute file path, or url.

**Returns**

Tuple of the form (data, metadata).

- **data**: DataFrame
  - A dataframe with the columns described in the table below. For a more detailed descriptions of each component, please consult the TMY2 User’s Manual ([1]), especially tables 3-1 through 3-6, and Appendix B.

- **metadata**: dict
  - The site metadata available in the file.

**Notes**

The returned structures have the following fields.

<table>
<thead>
<tr>
<th>key</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SiteID</td>
<td>Site identifier code (WBAN number)</td>
</tr>
<tr>
<td>StationName</td>
<td>Station name</td>
</tr>
<tr>
<td>StationState</td>
<td>Station state 2 letter designator</td>
</tr>
<tr>
<td>SiteTimeZone</td>
<td>Hours from Greenwich</td>
</tr>
<tr>
<td>latitude</td>
<td>Latitude in decimal degrees</td>
</tr>
<tr>
<td>longitude</td>
<td>Longitude in decimal degrees</td>
</tr>
<tr>
<td>SiteElevation</td>
<td>Site elevation in meters</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TMYData field</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>Pandas timeseries object containing timestamps</td>
</tr>
<tr>
<td>year</td>
<td></td>
</tr>
<tr>
<td>month</td>
<td></td>
</tr>
<tr>
<td>day</td>
<td></td>
</tr>
<tr>
<td>hour</td>
<td></td>
</tr>
<tr>
<td>ETR</td>
<td>Extraterrestrial horizontal radiation rec’d during 60 minutes prior to timestamp, Wh/m^2</td>
</tr>
<tr>
<td>ETRN</td>
<td>Extraterrestrial normal radiation rec’d during 60 minutes prior to timestamp, Wh/m^2</td>
</tr>
<tr>
<td>GHI</td>
<td>Direct and diffuse horizontal radiation rec’d during 60 minutes prior to timestamp, Wh/m^2</td>
</tr>
<tr>
<td>GHISource</td>
<td>See [1], Table 3-3</td>
</tr>
<tr>
<td>GHIUncertainty</td>
<td>See [1], Table 3-4</td>
</tr>
<tr>
<td>DNI</td>
<td>Amount of direct normal radiation (modeled) rec’d during 60 mintues prior to timestamp, Wh/m^2</td>
</tr>
<tr>
<td>DNISource</td>
<td>See [1], Table 3-3</td>
</tr>
<tr>
<td>TMYData field</td>
<td>description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>DNIUncertainty</td>
<td>See [1], Table 3-4</td>
</tr>
<tr>
<td>DNI</td>
<td>Amount of diffuse horizontal radiation received during 60 minutes prior to timestamp, Wh/m²</td>
</tr>
<tr>
<td>DNIsource</td>
<td>See [1], Table 3-3</td>
</tr>
<tr>
<td>DNIUncertainty</td>
<td>See [1], Table 3-4</td>
</tr>
<tr>
<td>GHI</td>
<td>Avg. total horizontal illuminance received during the 60 minutes prior to timestamp, units of 100 lux (e.g. value of 50 = 5000 lux)</td>
</tr>
<tr>
<td>GHIsource</td>
<td>See [1], Table 3-3</td>
</tr>
<tr>
<td>GHIUncertainty</td>
<td>See [1], Table 3-4</td>
</tr>
<tr>
<td>DNillum</td>
<td>Avg. direct normal illuminance received during the 60 minutes prior to timestamp, units of 100 lux</td>
</tr>
<tr>
<td>DNillumsource</td>
<td>See [1], Table 3-3</td>
</tr>
<tr>
<td>DNillumUncertainty</td>
<td>See [1], Table 3-4</td>
</tr>
<tr>
<td>DHI</td>
<td>Avg. horizontal diffuse illuminance received during the 60 minutes prior to timestamp, units of 100 lux</td>
</tr>
<tr>
<td>DHIsource</td>
<td>See [1], Table 3-3</td>
</tr>
<tr>
<td>DHIUncertainty</td>
<td>See [1], Table 3-4</td>
</tr>
<tr>
<td>ZenithLum</td>
<td>Avg. luminance at the sky’s zenith during the 60 minutes prior to timestamp, units of 10 Cd/m² (e.g. value of 700 = 7,000 Cd/m²)</td>
</tr>
<tr>
<td>ZenithLumsource</td>
<td>See [1], Table 3-3</td>
</tr>
<tr>
<td>ZenithLumUncertainty</td>
<td>See [1], Table 3-4</td>
</tr>
<tr>
<td>TotCld</td>
<td>Amount of sky dome covered by clouds or obscuring phenomena at time stamp, tenths of sky</td>
</tr>
<tr>
<td>TotCldSource</td>
<td>See [1], Table 3-5, 8760x1 cell array of strings</td>
</tr>
<tr>
<td>TotCldUncertainty</td>
<td>See [1], Table 3-6</td>
</tr>
<tr>
<td>OpqCld</td>
<td>Amount of sky dome covered by clouds or obscuring phenomena that prevent observing the sky at time stamp, tenths of sky</td>
</tr>
<tr>
<td>OpqCldSource</td>
<td>See [1], Table 3-5, 8760x1 cell array of strings</td>
</tr>
<tr>
<td>OpqCldUncertainty</td>
<td>See [1], Table 3-6</td>
</tr>
<tr>
<td>DryBulb</td>
<td>Dry bulb temperature at the time indicated, in tenths of degree C (e.g. 352 = 35.2 C).</td>
</tr>
<tr>
<td>DryBulbSource</td>
<td>See [1], Table 3-5, 8760x1 cell array of strings</td>
</tr>
<tr>
<td>DryBulbUncertainty</td>
<td>See [1], Table 3-6</td>
</tr>
<tr>
<td>DewPoint</td>
<td>Dew-point temperature at the time indicated, in tenths of degree C (e.g. 76 = 7.6 C).</td>
</tr>
<tr>
<td>DewPointSource</td>
<td>See [1], Table 3-5, 8760x1 cell array of strings</td>
</tr>
<tr>
<td>DewPointUncertainty</td>
<td>See [1], Table 3-6</td>
</tr>
<tr>
<td>RHum</td>
<td>Relative humidity at the time indicated, percent</td>
</tr>
<tr>
<td>RHumSource</td>
<td>See [1], Table 3-5, 8760x1 cell array of strings</td>
</tr>
<tr>
<td>RHumUncertainty</td>
<td>See [1], Table 3-6</td>
</tr>
<tr>
<td>Pressure</td>
<td>Station pressure at the time indicated, 1 mbar</td>
</tr>
<tr>
<td>PressureSource</td>
<td>See [1], Table 3-5, 8760x1 cell array of strings</td>
</tr>
<tr>
<td>PressureUncertainty</td>
<td>See [1], Table 3-6</td>
</tr>
<tr>
<td>Wdir</td>
<td>Wind direction at time indicated, degrees from east of north (360 = 0 = north; 90 = East; 0 = undefined, calm)</td>
</tr>
<tr>
<td>WdirSource</td>
<td>See [1], Table 3-5, 8760x1 cell array of strings</td>
</tr>
<tr>
<td>WdirUncertainty</td>
<td>See [1], Table 3-6</td>
</tr>
<tr>
<td>Wspd</td>
<td>Wind speed at the time indicated, in tenths of meters/second (e.g. 212 = 21.2 m/s)</td>
</tr>
<tr>
<td>WspdSource</td>
<td>See [1], Table 3-5, 8760x1 cell array of strings</td>
</tr>
<tr>
<td>WspdUncertainty</td>
<td>See [1], Table 3-6</td>
</tr>
<tr>
<td>Hvis</td>
<td>Distance to discernable remote objects at time indicated (7777=unlimited, 9999=missing data), in tenths of kilometers (e.g. 341 = 34.1 km)</td>
</tr>
<tr>
<td>HvisSource</td>
<td>See [1], Table 3-5, 8760x1 cell array of strings</td>
</tr>
<tr>
<td>HvisUncertainty</td>
<td>See [1], Table 3-6</td>
</tr>
<tr>
<td>CeilingHgt</td>
<td>Height of cloud base above local terrain (7777=unlimited, 88888=cirroform, 99999=missing data), in meters</td>
</tr>
<tr>
<td>CeilingHgtSource</td>
<td>See [1], Table 3-5, 8760x1 cell array of strings</td>
</tr>
<tr>
<td>CeilingHgtUncertainty</td>
<td>See [1], Table 3-6</td>
</tr>
<tr>
<td>Pwat</td>
<td>Total precipitable water contained in a column of unit cross section from Earth to top of atmosphere, in millimeters</td>
</tr>
<tr>
<td>PwatSource</td>
<td>See [1], Table 3-5, 8760x1 cell array of strings</td>
</tr>
<tr>
<td>PwatUncertainty</td>
<td>See [1], Table 3-6</td>
</tr>
</tbody>
</table>
Table 2.1 – continued from previous page

<table>
<thead>
<tr>
<th>TMYData field</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOD</td>
<td>The broadband aerosol optical depth (broadband turbidity) in thousandths on the day indicated (e.g. 114 = 0.114).</td>
</tr>
<tr>
<td>AODSource</td>
<td>See [1], Table 3-5, 8760x1 cell array of strings</td>
</tr>
<tr>
<td>AODUncertainty</td>
<td>See [1], Table 3-6</td>
</tr>
<tr>
<td>SnowDepth</td>
<td>Snow depth in centimeters on the day indicated, (999 = missing data).</td>
</tr>
<tr>
<td>SnowDepthSource</td>
<td>See [1], Table 3-5, 8760x1 cell array of strings</td>
</tr>
<tr>
<td>SnowDepthUncertainty</td>
<td>See [1], Table 3-6</td>
</tr>
<tr>
<td>LastSnowfall</td>
<td>Number of days since last snowfall (maximum value of 88, where 88 = 88 or greater days; 99 = missing data).</td>
</tr>
<tr>
<td>LastSnowfallSource</td>
<td>See [1], Table 3-5, 8760x1 cell array of strings</td>
</tr>
<tr>
<td>LastSnowfallUncertainty</td>
<td>See [1], Table 3-6</td>
</tr>
<tr>
<td>PresentWeather</td>
<td>See [1], Appendix B, an 8760x1 cell array of strings. Each string contains 10 numeric values. The string can be parsed to determine</td>
</tr>
</tbody>
</table>

References


**pvlib.tmy.readtmy3** *(filename=None, coerce_year=None, recolumn=True)*

Read a TMY3 file into a pandas dataframe.

Note that values contained in the metadata dictionary are unchanged from the TMY3 file (i.e. units are retained). In the case of any discrepancies between this documentation and the TMY3 User’s Manual [1], the TMY3 User’s Manual takes precedence.

**Parameters**

- **filename**: None or string
  - If None, attempts to use a Tkinter file browser. A string can be a relative file path, absolute file path, or url.

- **coerce_year**: None or int
  - If supplied, the year of the data will be set to this value.

- **recolumn**: bool
  - If True, apply standard names to TMY3 columns. Typically this results in stripping the units from the column name.

**Returns**

Tuple of the form (data, metadata).

- **data**: DataFrame
  - A pandas dataframe with the columns described in the table below. For more detailed descriptions of each component, please consult the TMY3 User’s Manual ([1]), especially tables 1-1 through 1-6.

- **metadata**: dict
  - The site metadata available in the file.

**Notes**

The returned structures have the following fields.
<table>
<thead>
<tr>
<th>key</th>
<th>format</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>altitude</td>
<td>Float</td>
<td>site elevation</td>
</tr>
<tr>
<td>latitude</td>
<td>Float</td>
<td>site latitude</td>
</tr>
<tr>
<td>longitude</td>
<td>Float</td>
<td>site longitude</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>site name</td>
</tr>
<tr>
<td>State</td>
<td>String</td>
<td>state</td>
</tr>
<tr>
<td>TZ</td>
<td>Float</td>
<td>UTC offset</td>
</tr>
<tr>
<td>USAF</td>
<td>Int</td>
<td>USAF identifier</td>
</tr>
</tbody>
</table>

**TMYData field**

- **TMYData.Index**: A pandas datetime index. NOTE, the index is currently timezone unaware, and times are set to local standard time (daylight savings is not included).
- **TMYData.ETR**: Extraterrestrial horizontal radiation rec’d during 60 minutes prior to timestamp, Wh/m^2^2
- **TMYData.ETRN**: Extraterrestrial normal radiation rec’d during 60 minutes prior to timestamp, Wh/m^2^2
- **TMYData.GHI**: Direct and diffuse horizontal radiation rec’d during 60 minutes prior to timestamp, Wh/m^2^2
  - **TMYData.GHIUncertainty**: Uncertainty based on random and bias error estimates see [2]
- **TMYData.DNI**: Amount of direct normal radiation (modeled) rec’d during 60 minutes prior to timestamp, Wh/m^2^2
  - **TMYData.DNISource**: See [1], Table 1-4
- **TMYData.DNIUncertainty**: Uncertainty based on random and bias error estimates see [2]
  - **TMYData.DNISource**: See [1], Table 1-4
- **TMYData.DHI**: Amount of diffuse horizontal radiation rec’d during 60 minutes prior to timestamp, Wh/m^2^2
  - **TMYData.DHISource**: See [1], Table 1-4
- **TMYData.DHIUncertainty**: Uncertainty based on random and bias error estimates see [2]
  - **TMYData.DHIUncertainty**: See [1], Table 1-4
- **TMYData.DNillum**: Avg. direct normal illuminance rec’d during the 60 minutes prior to timestamp, lx
  - **TMYData.DNillumSource**: See [1], Table 1-4
- **TMYData.DNillumUncertainty**: Uncertainty based on random and bias error estimates see [2]
  - **TMYData.DNillumSource**: See [1], Table 1-4
- **TMYData.DHillum**: Avg. horizontal diffuse illuminance rec’d during the 60 minutes prior to timestamp, lx
  - **TMYData.DHillumSource**: See [1], Table 1-4
- **TMYData.DHillumUncertainty**: Uncertainty based on random and bias error estimates see [2]
  - **TMYData.DHillumSource**: See [1], Table 1-4
- **TMYData.Zenithlum**: Avg. luminance at the sky’s zenith during the 60 minutes prior to timestamp, cd/m^2^2
  - **TMYData.ZenithlumSource**: See [1], Table 1-4
- **TMYData.ZenithlumUncertainty**: Uncertainty based on random and bias error estimates see [1] section 2.10
  - **TMYData.ZenithSource**: See [1], Table 1-4
- **TMYData.TotCld**: Amount of sky dome covered by clouds or obscuring phenomena at time stamp, tenths of sky
  - **TMYData.TotCldSource**: See [1], Table 1-5, 8760x1 cell array of strings
  - **TMYData.TotCldUncertainty**: See [1], Table 1-6
- **TMYData.OpqCld**: Amount of sky dome covered by clouds or obscuring phenomena that prevent observing the sky at time stamp, tenths of sky
  - **TMYData.OpqCldSource**: See [1], Table 1-5, 8760x1 cell array of strings
  - **TMYData.OpqCldUncertainty**: See [1], Table 1-6
- **TMYData.DryBulb**: Dry bulb temperature at the time indicated, deg C
  - **TMYData.DryBulbSource**: See [1], Table 1-5, 8760x1 cell array of strings
  - **TMYData.DryBulbUncertainty**: See [1], Table 1-6
- **TMYData.DewPoint**: Dew-point temperature at the time indicated, deg C
  - **TMYData.DewPointSource**: See [1], Table 1-5, 8760x1 cell array of strings
  - **TMYData.DewPointUncertainty**: See [1], Table 1-6
- **TMYData.RHum**: Relative humidity at the time indicated, percent
  - **TMYData.RHumSource**: See [1], Table 1-5, 8760x1 cell array of strings
  - **TMYData.RHumUncertainty**: See [1], Table 1-6
- **TMYData.Pressure**: Station pressure at the time indicated, 1 mbar
  - **TMYData.PressureSource**: See [1], Table 1-5, 8760x1 cell array of strings

---

2.4. Modules 65
Table 2.2 – continued from previous page

<table>
<thead>
<tr>
<th>TMYData field</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMYData.PressureUncertainty</td>
<td>See [1], Table 1-6</td>
</tr>
<tr>
<td>TMYData.Wdir</td>
<td>Wind direction at time indicated, degrees from north (360 = north; 0 = undefined, calm)</td>
</tr>
<tr>
<td>TMYData.WdirSource</td>
<td>See [1], Table 1-5, 8760x1 cell array of strings</td>
</tr>
<tr>
<td>TMYData.WdirUncertainty</td>
<td>See [1], Table 1-6</td>
</tr>
<tr>
<td>TMYData.Wspd</td>
<td>Wind speed at the time indicated, meter/second</td>
</tr>
<tr>
<td>TMYData.WspdSource</td>
<td>See [1], Table 1-5, 8760x1 cell array of strings</td>
</tr>
<tr>
<td>TMYData.WspdUncertainty</td>
<td>See [1], Table 1-6</td>
</tr>
<tr>
<td>TMYData.Hvis</td>
<td>Distance to discernable remote objects at time indicated (7777=unlimited), meter</td>
</tr>
<tr>
<td>TMYData.HvisSource</td>
<td>See [1], Table 1-5, 8760x1 cell array of strings</td>
</tr>
<tr>
<td>TMYData.HvisUncertainty</td>
<td>See [1], Table 1-6</td>
</tr>
<tr>
<td>TMYData.CeilHgt</td>
<td>Height of cloud base above local terrain (7777=unlimited), meter</td>
</tr>
<tr>
<td>TMYData.CeilHgtSource</td>
<td>See [1], Table 1-5, 8760x1 cell array of strings</td>
</tr>
<tr>
<td>TMYData.CeilHgtUncertainty</td>
<td>See [1], Table 1-6</td>
</tr>
<tr>
<td>TMYData.Pwat</td>
<td>Total precipitable water contained in a column of unit cross section from earth to top of atmosphere, cm</td>
</tr>
<tr>
<td>TMYData.PwatSource</td>
<td>See [1], Table 1-5, 8760x1 cell array of strings</td>
</tr>
<tr>
<td>TMYData.PwatUncertainty</td>
<td>See [1], Table 1-6</td>
</tr>
<tr>
<td>TMYData.AOD</td>
<td>The broadband aerosol optical depth per unit of air mass due to extinction by aerosol component of atmosphere, unitless</td>
</tr>
<tr>
<td>TMYData.AODSource</td>
<td>See [1], Table 1-5, 8760x1 cell array of strings</td>
</tr>
<tr>
<td>TMYData.AODUncertainty</td>
<td>See [1], Table 1-6</td>
</tr>
<tr>
<td>TMYData.Alb</td>
<td>The ratio of reflected solar irradiance to global horizontal irradiance, unitless</td>
</tr>
<tr>
<td>TMYData.AlbSource</td>
<td>See [1], Table 1-5, 8760x1 cell array of strings</td>
</tr>
<tr>
<td>TMYData.AlbUncertainty</td>
<td>See [1], Table 1-6</td>
</tr>
<tr>
<td>TMYData.Lprecipdepth</td>
<td>The amount of liquid precipitation observed at indicated time for the period indicated in the liquid precipitation quantity field, millimeter</td>
</tr>
<tr>
<td>TMYData.Lprecipquantity</td>
<td>The period of accumulatitudeion for the liquid precipitation depth field, hour</td>
</tr>
<tr>
<td>TMYData.LprecipSource</td>
<td>See [1], Table 1-5, 8760x1 cell array of strings</td>
</tr>
<tr>
<td>TMYData.LprecipUncertainty</td>
<td>See [1], Table 1-6</td>
</tr>
</tbody>
</table>

References


### 2.4.9 tracking module

**pvlid.tracking** **.singleaxis** *(apparent_zenith, apparent_azimuth, axis_tilt=0, axis_azimuth=0, max_angle=90, backtrack=True, gcr=0.2857142857142857)*

Determine the rotation angle of a single axis tracker using the equations in [1] when given a particular sun zenith and azimuth angle. Backtracking may be specified, and if so, a ground coverage ratio is required.

Rotation angle is determined in a panel-oriented coordinate system. The tracker azimuth axis_azimuth defines the positive y-axis; the positive x-axis is 90 degrees clockwise from the y-axis and parallel to the earth surface, and the positive z-axis is normal and oriented towards the sun. Rotation angle tracker_theta indicates tracker position relative to horizontal: tracker_theta = 0 is horizontal, and positive tracker_theta is a clockwise rotation around the y axis in the x, y, z coordinate system. For example, if tracker azimuth axis_azimuth is 180 (oriented south), tracker_theta = 30 is a rotation of 30 degrees towards the west, and tracker_theta = -90 is a rotation to the vertical plane facing east.

**Parameters** **apparent_zenith** : Series
Solar apparent zenith angles in decimal degrees.

**apparent_azimuth** : Series
Solar apparent azimuth angles in decimal degrees.

**axis_tilt** : float
The tilt of the axis of rotation (i.e, the y-axis defined by axis_azimuth) with respect to horizontal, in decimal degrees.

**axis_azimuth** : float
A value denoting the compass direction along which the axis of rotation lies. Measured in decimal degrees East of North.

**max_angle** : float
A value denoting the maximum rotation angle, in decimal degrees, of the one-axis tracker from its horizontal position (horizontal if axis_tilt = 0). A max_angle of 90 degrees allows the tracker to rotate to a vertical position to point the panel towards a horizon. max_angle of 180 degrees allows for full rotation.

**backtrack** : bool
Controls whether the tracker has the capability to “backtrack” to avoid row-to-row shading. False denotes no backtrack capability. True denotes backtrack capability.

**gcr** : float
A value denoting the ground coverage ratio of a tracker system which utilizes back-tracking; i.e. the ratio between the PV array surface area to total ground area. A tracker system with modules 2 meters wide, centered on the tracking axis, with 6 meters between the tracking axes has a gcr of 2/6=0.333. If gcr is not provided, a gcr of 2/7 is default. gcr must be <=1.

**Returns** DataFrame with the following columns:

- **tracker_theta**: The rotation angle of the tracker.
  tracker_theta = 0 is horizontal, and positive rotation angles are clockwise.
- **aoi**: The angle-of-incidence of direct irradiance onto the rotated panel surface.
- **surface_tilt**: The angle between the panel surface and the earth surface, accounting for panel rotation.
- **surface_azimuth**: The azimuth of the rotated panel, determined by projecting the vector normal to the panel’s surface to the earth’s surface.

**References**


**2.4.10 tools module**

Collection of functions used in pvlib_python
pvlib.tools.asind(number)
Inverse Sine returning an angle in degrees

Parameters number : float
          Input number

Returns result : float
          arcsin result

pvlib.tools.cosd(angle)
Cosine with angle input in degrees

Parameters angle : float
          Angle in degrees

Returns result : float
          Cosine of the angle

pvlib.tools.datetime_to_djd(time)
Converts a datetime to the Dublin Julian Day

Parameters time : datetime.datetime
          time to convert

Returns float
          fractional days since 12/31/1899+0000

pvlib.tools.djd_to_datetime(djd, tz='UTC')
Converts a Dublin Julian Day float to a datetime.datetime object

Parameters djd : float
          fractional days since 12/31/1899+0000

tz : str
          timezone to localize the result to

Returns datetime.datetime
          The resultant datetime localized to tz

pvlib.tools.localize_to_utc(time, location)
Converts or localizes a time series to UTC.

Parameters time : datetime.datetime, pandas.DatetimeIndex,
          or pandas.Series/DataFrame with a DatetimeIndex.

location : pvlib.Location object

Returns pandas object localized to UTC.

pvlib.tools.sind(angle)
Sine with angle input in degrees

Parameters angle : float
          Angle in degrees

Returns result : float
          Sin of the angle
pvlib.tools.tand(angle)
    Tan with angle input in degrees

    Parameters angle : float
        Angle in degrees

    Returns result : float
        Tan of the angle
CHAPTER 3

Indices and tables

- genindex
- modindex
- search
p

pvlib.atmosphere, 22
pvlib.clearsky, 25
pvlib.forecast, 27
pvlib.irradiance, 32
pvlib.location, 45
pvlib.pvsystem, 46
pvlib.solarposition, 57
pvlib.tmy, 62
pvlib.tools, 67
pvlib.tracking, 66
Index

A
absoluteairmass() (in module pvlib.atmosphere), 22
access_url_key (pvlib.forecast.ForecastModel attribute),
alt2pres() (in module pvlib.atmosphere), 23
aoi() (in module pvlib.irradiance), 32
aoi_projection() (in module pvlib.irradiance), 33
ashraeiam() (in module pvlib.pvsystem), 46
asind() (in module pvlib.tools), 67

B
base_tds_url (pvlib.forecast.ForecastModel attribute), 28
beam_component() (in module pvlib.irradiance), 33

C
calc_radiation() (pvlib.forecast.ForecastModel method), 28
calc_temperature() (pvlib.forecast.ForecastModel method), 29
calc_time() (in module pvlib.solarposition), 57
calc_wind() (pvlib.forecast.ForecastModel method), 29
calcpars_desoto() (in module pvlib.pvsystem), 46
catalog_url (pvlib.forecast.ForecastModel attribute), 29
cloudy_day_check() (in module pvlib.irradiance), 34
columns (pvlib.forecast.ForecastModel attribute), 29
convert_temperature() (pvlib.forecast.ForecastModel method), 29
cosd() (in module pvlib.tools), 68

d

e
ephemeris() (in module pvlib.solarposition), 58
extraradiation() (in module pvlib.irradiance), 35

F
ForecastModel (class in pvlib.forecast), 27

G
get_query_data() (pvlib.forecast.ForecastModel method),
get_solarposition() (in module pvlib.solarposition), 58
get_sunrise_sunset_transit() (in module pvlib.solarposition), 59
GFS (class in pvlib.forecast), 30
globalinplane() (in module pvlib.irradiance), 36
grounddiffuse() (in module pvlib.irradiance), 37

H
haurwitz() (in module pvlib.clearsky), 25
haydavies() (in module pvlib.irradiance), 37
HRRR (class in pvlib.forecast), 30
HRRR_ESRL (class in pvlib.forecast), 31

I
i_from_v() (in module pvlib.pvsystem), 49
ineichen() (in module pvlib.clearsky), 26
isotropic() (in module pvlib.irradiance), 38

K
king() (in module pvlib.irradiance), 39
klucher() (in module pvlib.irradiance), 39

L
liujordan() (in module pvlib.irradiance), 40
localize_to_utc() (in module pvlib.tools), 68
Location (class in pvlib.location), 45
lookup_linke_turbidity() (in module pvlib.clearsky), 27

N
NAM (class in pvlib.forecast), 31
NDFD (class in pvlib.forecast), 32
netcdf2pandas() (pvlib.forecast.ForecastModel method), 29
P

perez() (in module pvlib.irradiance), 41
physicaliam() (in module pvlib.pvsystem), 50
poa_horizontal_ratio() (in module pvlib.irradiance), 42
pres2alt() (in module pvlib.atmosphere), 23
pvlib.atmosphere (module), 22
pvlib.clearsky (module), 25
pvlib.forecast (module), 27
pvlib.irradiance (module), 32
pvlib.location (module), 45
pvlib.pvsystem (module), 46
pvlib.solarposition (module), 57
pvlib.tmy (module), 62
pvlib.tools (module), 67
pvlib.tracking (module), 66
pyephem() (in module pvlib.solarposition), 59
pyephem_earthsun_distance() (in module pvlib.solarposition), 60

V

vert_level (pvlib.forecast.ForecastModel attribute), 30

R

RAP (class in pvlib.forecast), 32
readtmy2() (in module pvlib.tmy), 62
readtmy3() (in module pvlib.tmy), 64
reindl() (in module pvlib.irradiance), 43
relativeairmass() (in module pvlib.atmosphere), 24
retrieve_sam() (in module pvlib.pvsystem), 51

S

sapm() (in module pvlib.pvsystem), 52
sapm_celltemp() (in module pvlib.pvsystem), 53
set_dataset() (pvlib.forecast.ForecastModel method), 29
set_location() (pvlib.forecast.ForecastModel method), 29
set_query_latitude() (pvlib.forecast.ForecastModel method), 30
set_query_time() (pvlib.forecast.ForecastModel method), 30
set_time() (pvlib.forecast.ForecastModel method), 30
set_variable_stdnames() (pvlib.forecast.ForecastModel method), 30
set_variable_units() (pvlib.forecast.ForecastModel method), 30
sind() (in module pvlib.tools), 68
singleaxis() (in module pvlib.tracking), 66
singlydiode() (in module pvlib.pvsystem), 53
snlinverter() (in module pvlib.pvsystem), 55
spa_c() (in module pvlib.solarposition), 60
spa_python() (in module pvlib.solarposition), 61
systemdef() (in module pvlib.pvsystem), 56

T

tand() (in module pvlib.tools), 68
total_irrad() (in module pvlib.irradiance), 44
transmittance() (in module pvlib.atmosphere), 25