PV PERFORMANCE MODELLING WITH PVPMC/PVLIB

Steve Ransome[1]

Josh Stein[2], Will Holmgren[3] & Juergen Sutterlueti[4]

[1]Steve Ransome Consulting Limited, London UK

[2]Sandia National Laboratories, Albuquerque NM, USA

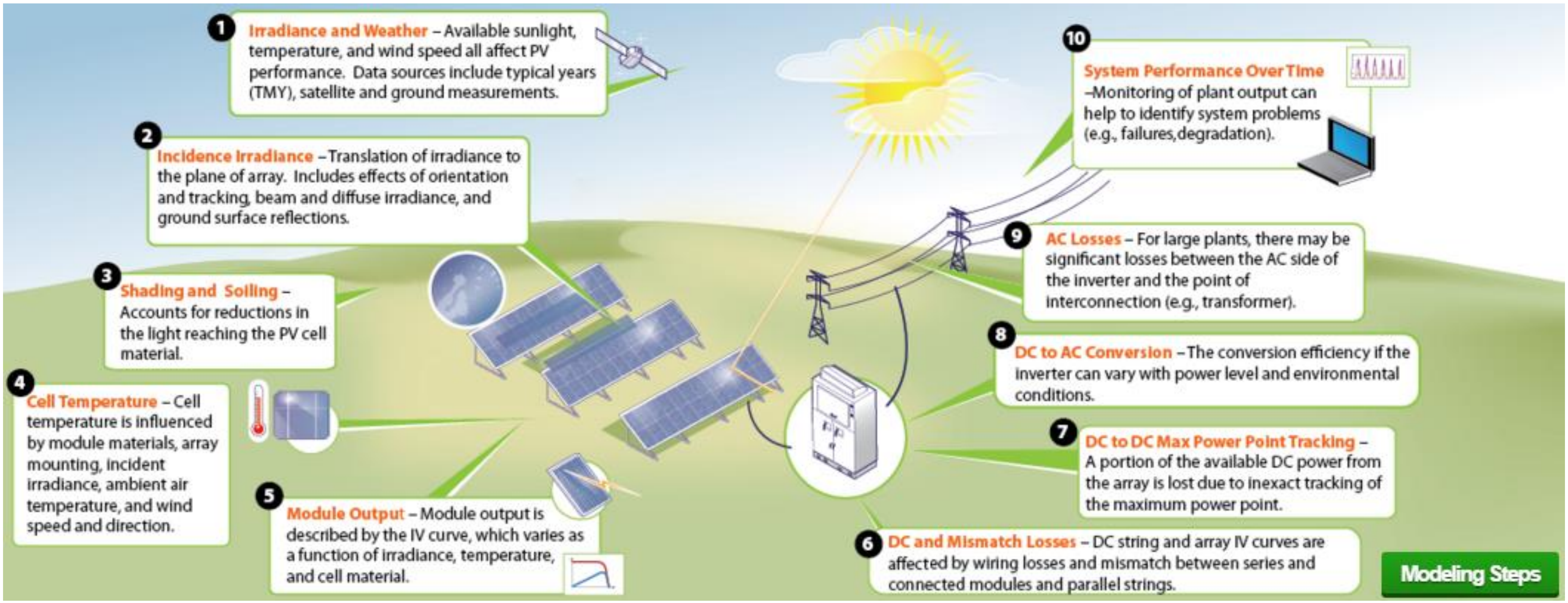
[3]University of Arizona, USA

[4]Gantner Instruments, Germany

PVSAT 12 Liverpool UK 6-8 Apr 2016

PV Performance Modelling steps

Site weather, Design, Component values & quantities → kWh/y



The present status of PV Performance modelling

- **Users cannot see all the algorithms and assumptions used**
  in most simulation programs (such as PVSyst, PVSol …)

- **It's difficult to simulate more complicated systems than authors allowed for**
  e.g. multiple array orientations, varied design, differing PV panels or inverters.

- **Bug fixing can be time consuming**
  as it needs to be done by original authors and redistributed.

- **Users might need to exceed default input limitations**
  e.g. V_{MP} tracking limits or P_{DC}/P_{AC} ratios.

- **New algorithms can be difficult to validate**
  unless all loss stages are modelled and analysed

PVPMC (PV Performance Modelling Collaborative) and PVLIB (PV Library)

- The PVPMC aims to **improve the accuracy of PV performance models** for
  - **instantaneous PV performance** (e.g. P_{OUT} W vs. weather data)
  - **predicting energy yield** (YA or YF kWh / time)
  - **calculating investment risk** (with cost assumptions).
- PVLIB is a **standard repository** for **high quality PV algorithms**.
- Code is **open-source** and is **collaboratively developed** and validated.
- PVLIB is available in both **MATLAB** (license=£1600) and **Python** (license=free!) language versions.

PVLIB code is hosted on a web based repository (GitHub)

Allowing developers to collaborate easily

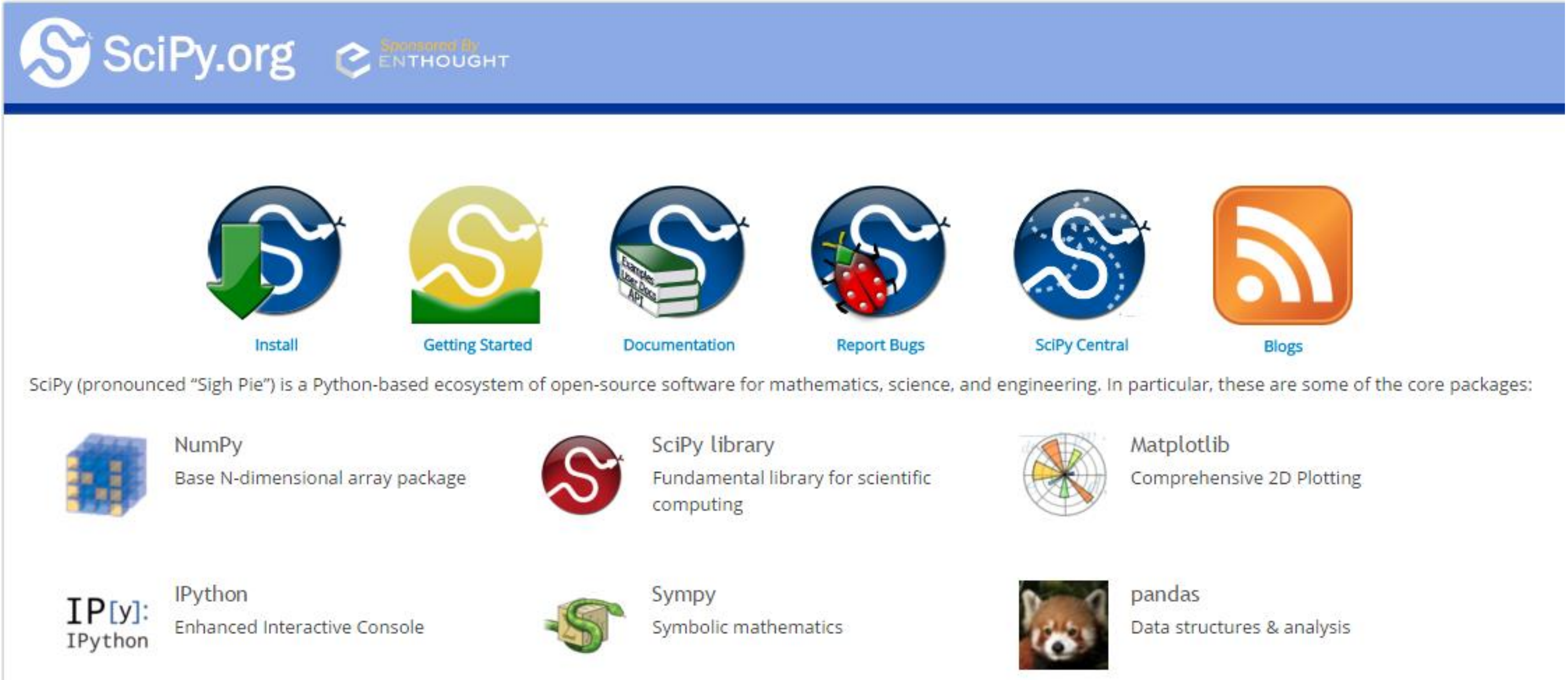
Source code is separated into these PV LIB code Modules

• tools	trigonometry, time functions and maths
• location	latitude, longitude, time zone and altitude
• tmy	hourly weather files (typical meteorological year)
• solar position	solar altitude and azimuth
• pv tracking	single or two axis tracking if not fixed orientation
• atmosphere	air mass etc.
• clear sky	extraterrestrial direct,normal clear-sky irradiance
• irradiance	angle of incidence; beam, diffuse, global irradiance
• pv system	array orientation, reflectivity, pv models

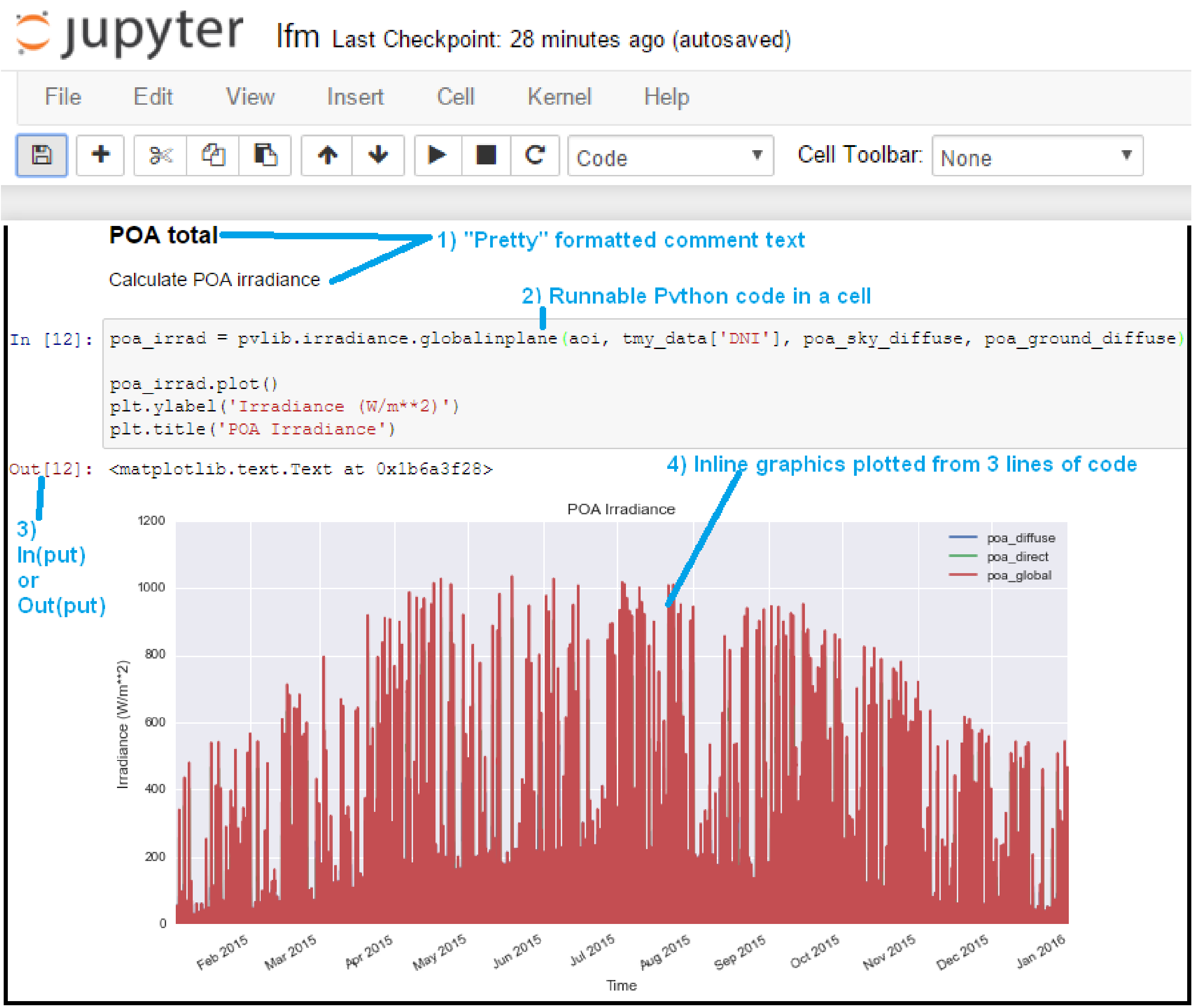
Example scripts provided take the user through all of the modelling stages from site and weather data input to AC power output.

Many Python Libraries add language functionality

(named after Monty Python's Flying Circus) →



PVLIB code is run in a web browser such as Chrome in an iPython Notebook Environment



Menu bar

(run, stop, cut, paste, step, save …)

Main area

Comments, code, results, graphics

Example PVLIB code tutorial "TMY(weather) to Power"

From setup (formatted comments)

To Output power (built in graphics modules)

TMY to Power Tutorial

This tutorial will walk through the process of going from TMY data to AC power using the SAPM.

Table of contents:

1) Formatted comments

This tutorial has been tested against the following package versions:

- pvlib 0.2.0
- Python 2.7.10
- IPython 3.2
- pandas 0.16.2

It should work with other Python and Pandas versions. It requires pvlib >= 0.2.0 and IPython >= 3.0.

Authors:

- Will Holmgren (@wholmgren), University of Arizona, July 2015
- Rob Andrews (@Calama-Consulting), Heliolytics, June 2014

Setup

These are just your standard interactive scientific python imports that you'll get very used to using.

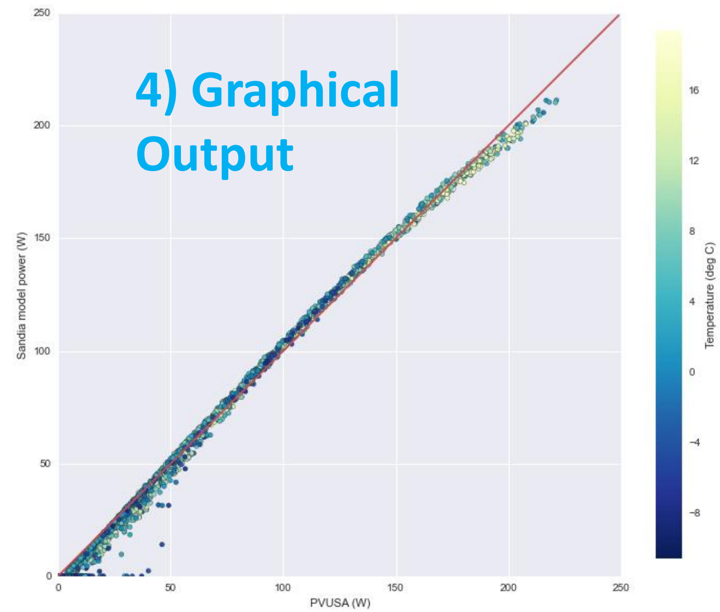
In [1]:
```
# built-in python modules
import os
import inspect

# scientific python add-ons
import numpy as np
import pandas as pd

# plotting stuff
# first line makes the plots appear in the notebook
%matplotlib inline
import matplotlib.pyplot as plt
```

2) Setup

In [44]:
```
power_pvusa = pvusa(pvusa_data, *popt)

fig, ax = sapm_other_scatter(tmy_data.DryBulb, power_pvusa, clabel='Temperature (deg C)',
                             aspect_equal=True, xlabel='PVUSA (W)')

maxmax = max(ax.get_xlim()[1], ax.get_ylim()[1])
ax.set_ylim(None, maxmax)
ax.set_xlim(None, maxmax)
ax.plot(np.arange(maxmax), np.arange(maxmax), 'r')
```

3) Code

Out[44]: [<matplotlib.lines.Line2D at 0x222ae390>]



Typical simplified Python code with *explanations*

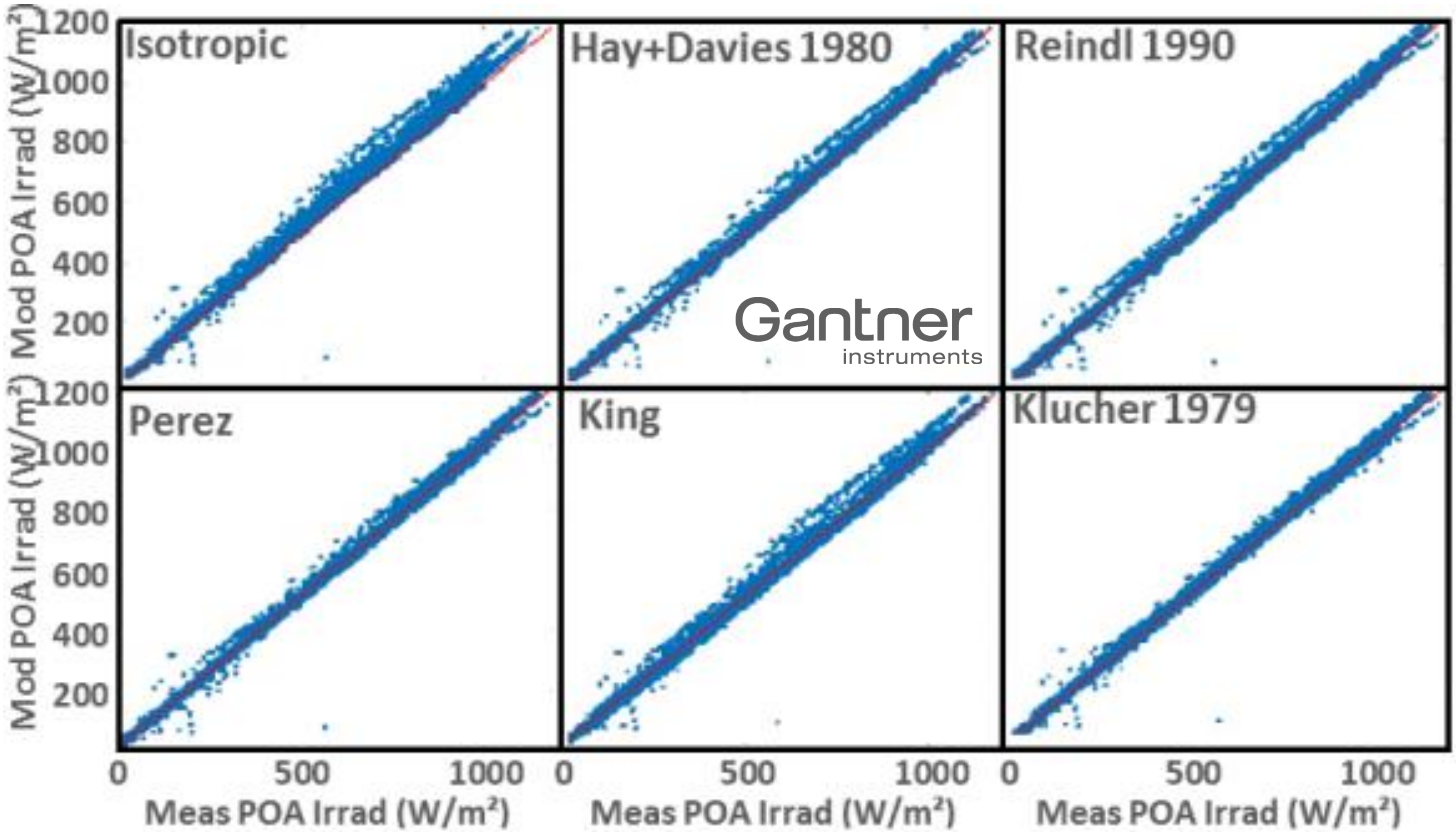
To estimate cell and module temperatures per SAPM

```
def sapm_celltemp(                                                      function name
    irrad, wind, temp, model='open_rack_cell_glassback'):                   and input series

    temp_models = {'open_rack_cell_glassback': [-3.47, -.0594, 3],      list models and
                   'roof_mount_cell_glassback': [-2.98, -.0471, 1],         their values
                   etc. }

    a = model[0] b = model[1] deltaT = model[2]                         get model values
    E0 = 1000. # Reference irradiance                                   set reference value

    temp_module = pd.Series(irrad*np.exp(a + b*wind) + temp)            calc module temp (C)
    temp_cell = temp_module + (irrad / E0)*(deltaT)                     calc cell temp (C)

    return pd.DataFrame(
        {'temp_cell': temp_cell, 'temp_module': temp_module})           output all data series
```

Validating PVLIB functions 1/2

e.g. Predicted vs. Measured tilted plane Irradiance kW/m²

Plane of Array Irradiance calculated vs. measured for six sky models (Gantner Instruments at their Tempe site).



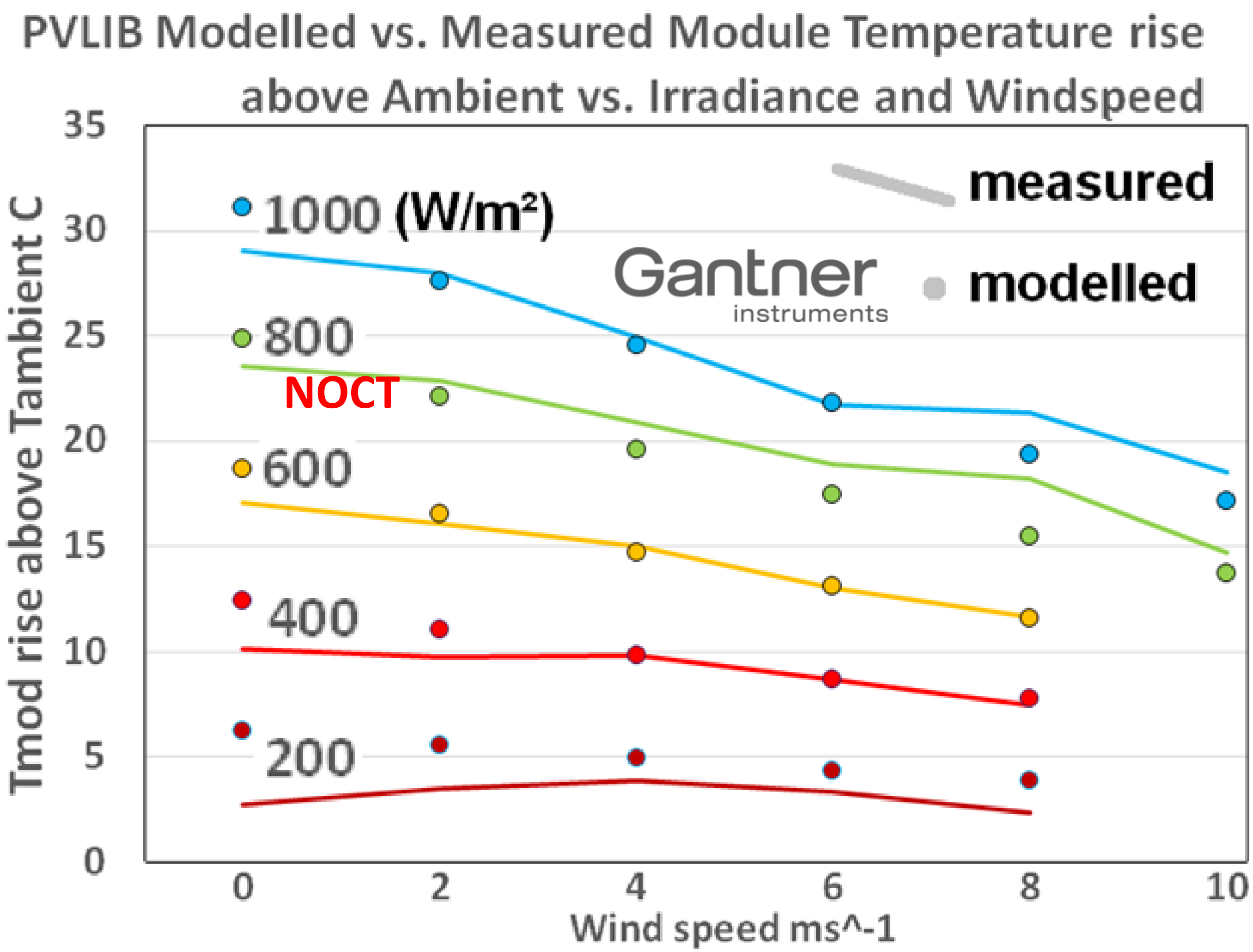
GI data, Tempe AZ

Validating PVLIB functions 2/2

e.g. Module Temperature rise vs. Irradiance & Wind speed

T_{MODULE} ~ function(
- $T_{AMBIENT}$,
- Irradiance,
- Manufacturing technology,
- Mounting method)

Using default PV_LIB coefficients gives a good overall agreement with discrepancies generally <±2C.



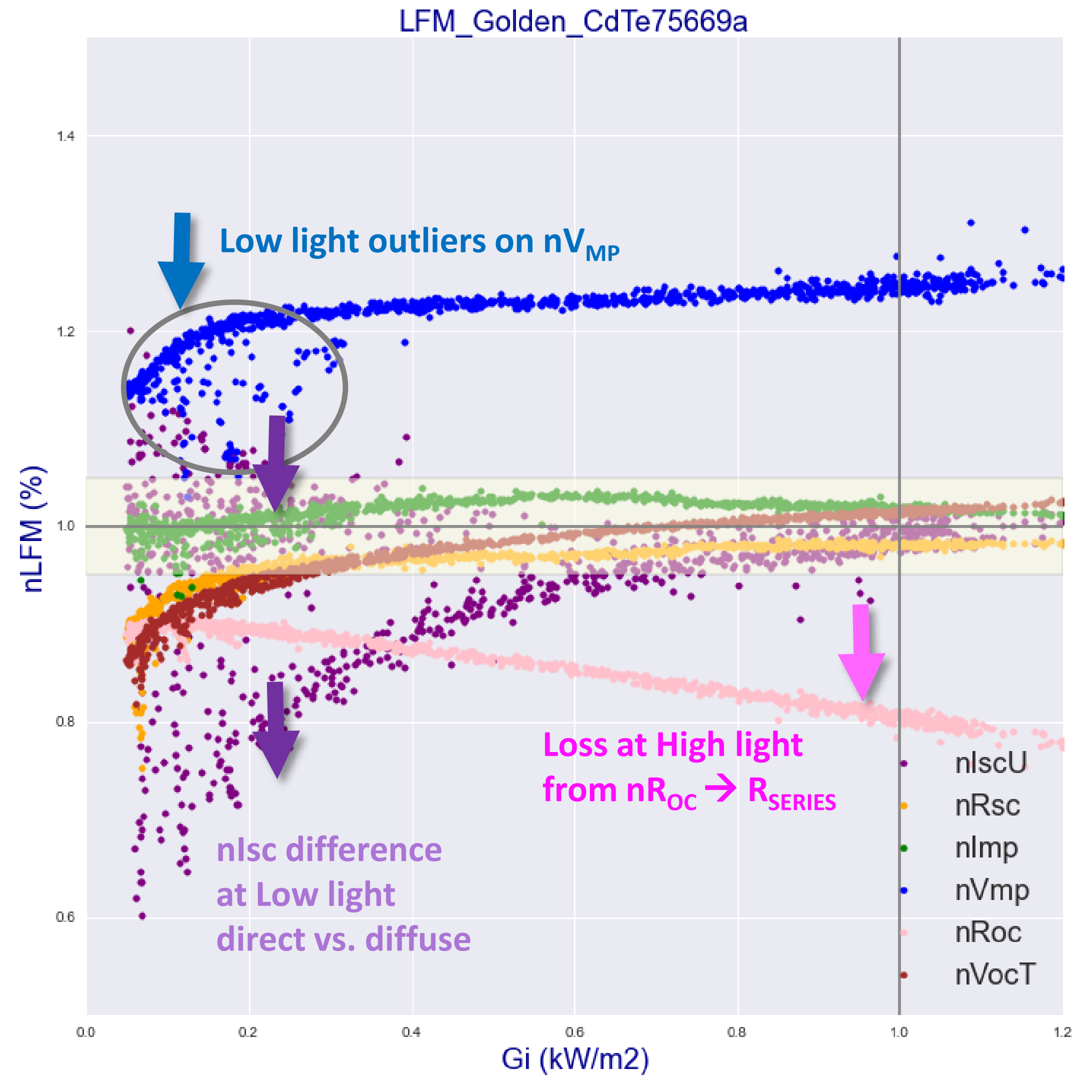
GI data, Tempe AZ

3rd Party iPython coding with PVLIB

SRCL analysis of NREL data using GI/SRCL LFM

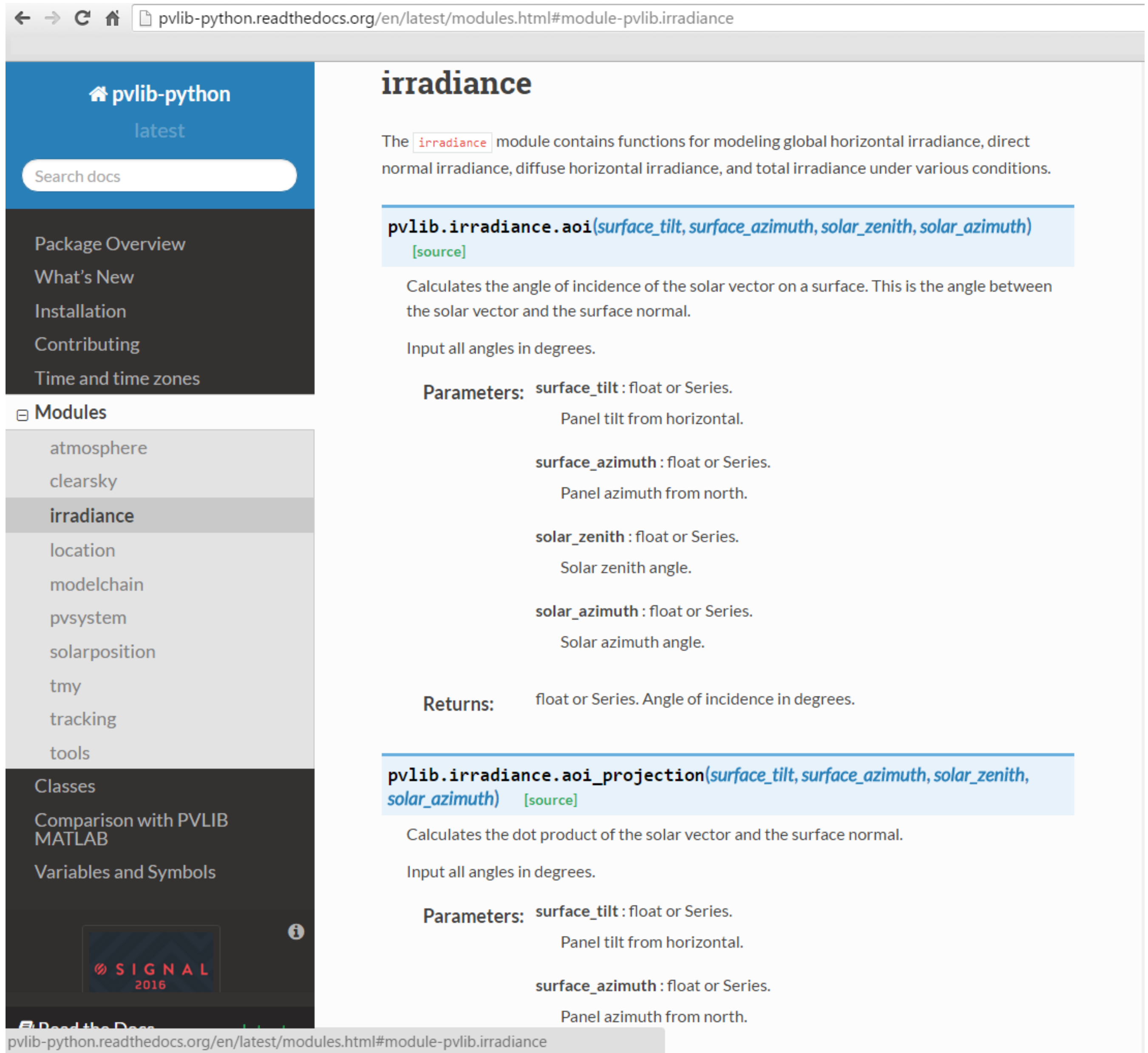
- Many algorithms and a rich programming environment are available to enable users to develop their own solutions which can be shared to help the community

- SRCL Python code analysing an NREL measured CdTe module using GI/SRCL LFM.



PVLIB Help files

Help for functions



pvlib-python.readthedocs.org/en/latest/modules.html#module-pvlib.irradiance

pvlib-python
latest
Search docs

Package Overview
What's New
Installation
Contributing
Time and time zones
Modules
atmosphere
clearsky
irradiance
location
modelchain
pvsystem
solarposition
tmy
tracking
tools
Classes
Comparison with PVLIB MATLAB
Variables and Symbols

irradiance

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*) [source]

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*) [source]

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.

pvlib-python.readthedocs.org/en/latest/modules.html#module-pvlib.irradiance

Steve Ransome Consulting Limited

PVLIB Help files

Help for functions and source code

pvlib-python.readthedocs.org/en/latest/modules.html#module-pvlib.irradiance

pvlib-python

latest

Search docs

irradiance

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*) [source]

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*) [source]

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.

html#module-pvlib.irradiance

pvlib-python.readthedocs.org/en/latest/_modules/pvlib/irradiance.html#aoi

pvlib-python

latest

Search docs

- Package Overview
- What's New
- Installation
- Contributing
- Time and time zones
- Modules
- Classes
- Comparison with PVLIB MATLAB
- Variables and Symbols

Read the Docs v: latest

Docs » Module code » pvlib.irradiance   Edit on GitHub

Source code for pvlib.irradiance

```python
"""
The ``irradiance`` module contains functions for modeling
global horizontal irradiance, direct normal irradiance,
diffuse horizontal irradiance, and total irradiance
under various conditions.
"""

from __future__ import division

import logging
pvl_logger = logging.getLogger('pvlib')

import datetime

import numpy as np
import pandas as pd

from pvlib import tools
from pvlib import solarposition

SURFACE_ALBEDOS = {'urban': 0.18,
                   'grass': 0.20,
                   'fresh grass': 0.26,
                   'soil': 0.17,
                   'sand': 0.40,
                   'snow': 0.65,
                   'fresh snow': 0.75,
                   'asphalt': 0.12,
                   'concrete': 0.30,
                   'aluminum': 0.85,
                   'copper': 0.74,
                   'fresh steel': 0.35,
                   'dirty steel': 0.08}

# would be nice if this took pandas index as well.
# Use try:except of isinstance.


def extraradiation(datetime_or_doy, solar_constant=1366.1, method='spencer'):   [docs]
    """
    Determine extraterrestrial radiation from day of year.
```

PVLIB Help files

What's new ?

pvlib-python.readthedocs.org/en/latest/whatsnew.html#v0-3-0-2016

pvlib-python

latest

Search docs

- Added new sections to the documentation:
  - Package Overview (GH93)
  - Installation (GH135)
  - Contributing (GH46)
  - Time and time zones (GH47)
  - Variables and Symbols (GH102)
  - Classes (GH93)
- Adds support for Appveyor, a Windows continuous integration service. (GH111)
- The readthedocs documentation build now uses conda packages instead of mock packages. This enables code to be run and figures to be generated during the documentation builds. (GH104)
- Reconfigures TravisCI builds and adds e.g. `has_numba` decorators to the test suite. The result is that the TravisCI test suite runs almost 10x faster and users do not have to install all optional dependencies to run the test suite. (GH109)
- Adds more unit tests that test that the return values are actually correct.
- Add `atmosphere.APPARENT_ZENITH_MODELS` and `atmosphere.TRUE_ZENITH_MODELS` to enable code that can automatically determine which type of zenith data to use e.g. `Location.get_airmass`.
- Modify `sapm` documentation to clarify that it does not work with the CEC database. (GH122)
- Adds citation information to the documentation. (GH73)
- Updates the Comparison with PVLIB MATLAB documentation. (GH116)

Bug fixes

- Fixed the metadata key specification in documentation of the `readtmy2` function.
- Fixes the import of tkinter on Python 3 (GH112)
- Add a decorator to skip `test_calcparams_desoto` on pandas 0.18.0. (GH130)
- Fixes `i_from_v` documentation. (GH126)
- Fixes two minor sphinx documentation errors: a too short heading underline in whatsnew/v0.2.2.txt and a table format in pvsystem. (GH123)

Contributors

- Will Holmgren
- pyElena21

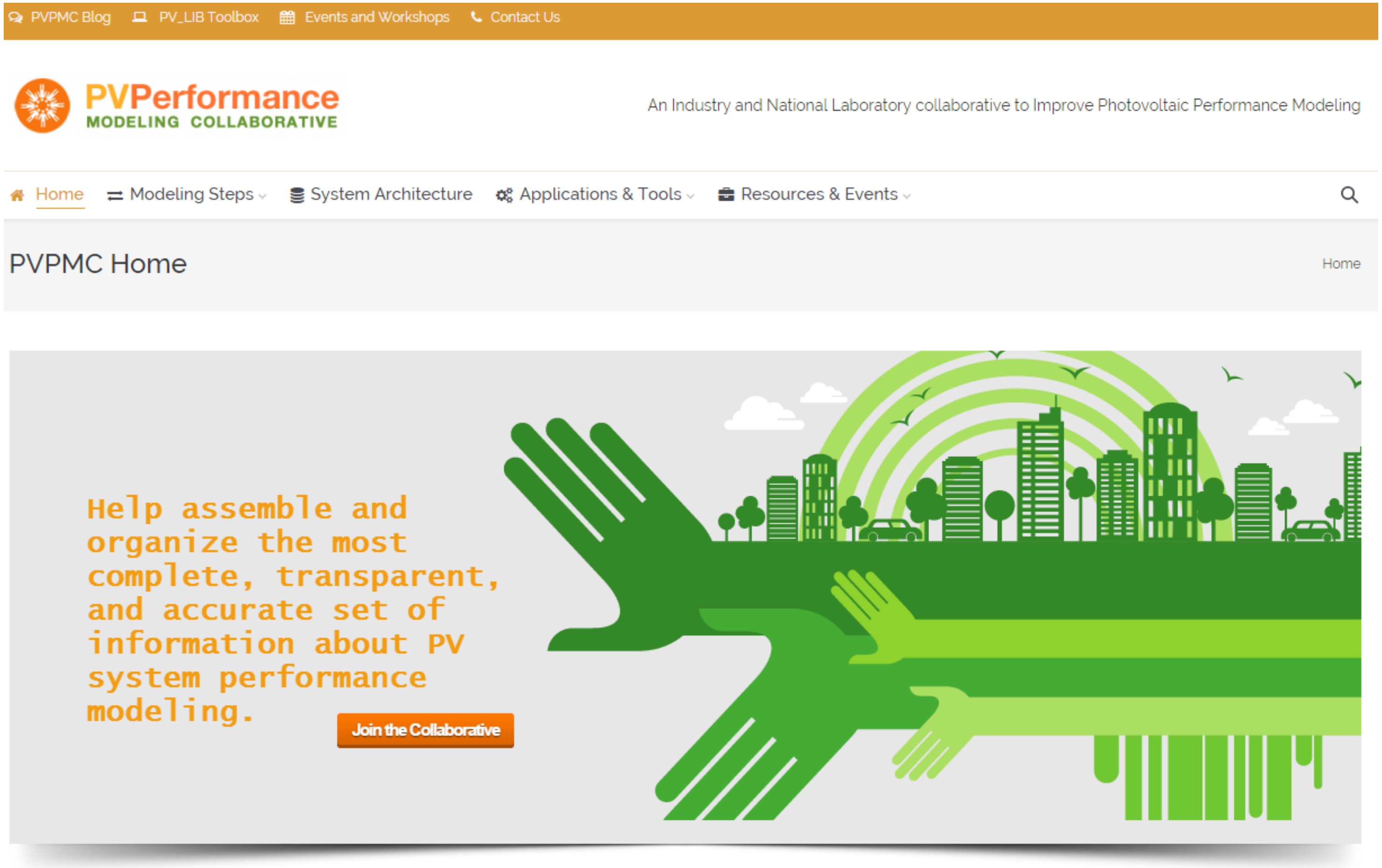
Some of the tasks that can be done by PVLIB code

- Basic simulation program
- Weather data analysis
- Angle of Incidence, Tracking
- Shading
- PV modelling
- Inverter modelling
- Curve fitting routines
- Statistics

Conclusions

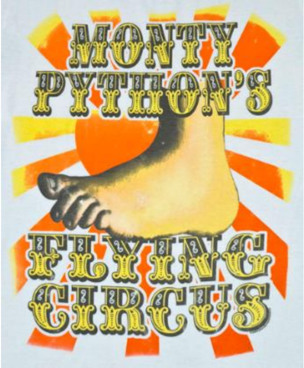
- PVMPC/PVLIB have been introduced with links to their website and details of their workshops.

- Anyone interested should download the toolbox and you are encouraged to learn Python and contribute.



Dates of next PVPMC Workshops

- 5^{th} : 9^{th} May, 2016 Santa Clara, USA ← SR not going
- 6^{th} : 24-25^{th} Oct, 2016 Freiburg, Germany ← SR going



Acknowledgements

- All PVPMC contributors on GitHub, Rob Andrews (Heliolytics)

www.pvpmc.org

Thanks for your attention and please get involved!

PVPerformance MODELING COLLABORATIVE

An Industry and National Laboratory collaborative to Improve Photovoltaic Performance Modeling

Gantner instruments · Heliolytics · Sandia National Laboratories · The University of Arizona

- Spare slides