PV PERFORMANCE MODELLING WITH PVPMC/PVLIB

Steve Ransome¹

Josh Stein², Will Holmgren³ & Juergen Sutterlueti⁴

¹Steve Ransome Consulting Limited, London UK
²Sandia National Laboratories, Albuquerque NM, USA
³University of Arizona, USA
⁴Gantner 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)

SciPy.org

SciPy (pronounced “Sigh Pie”) is a Python-based ecosystem of open-source software for mathematics, science, and engineering. In particular, these are some of the core packages:

- **NumPy**: Base N-dimensional array package
- **SciPy library**: Fundamental library for scientific computing
- **Matplotlib**: Comprehensive 2D Plotting
- **IPython**: Enhanced interactive Console
- **Sympy**: Symbolic mathematics
- **pandas**: Data structures & analysis
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)

1) Formatted comments

2) Setup

3) Code

4) Graphical Output

To Output power
(built in graphics modules)
Typical simplified Python code with explanations
To estimate cell and module temperatures per SAPM

```python
def sapm_celltemp(
    irrad, wind, temp, model='open_rack_cell_glassback'):

    temp_models = {'open_rack_cell_glassback': [-3.47, -0.0594, 3],
                   'roof_mount_cell_glassback': [-2.98, -0.0471, 1],
                   etc. }

    a = model[0]  b = model[1]  deltaT = model[2]
    E0 = 1000.  # Reference irradiance
    temp_module = pd.Series(irrad*np.exp(a + b*wind) + temp)
    temp_cell = temp_module + (irrad / E0)*(deltaT)

    return pd.DataFrame(
        {'temp_cell': temp_cell, 'temp_module': temp_module})
```

3-Jun-16

www.steveransome.com
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

\[ \text{T}_{\text{MODULE}} \sim \text{function}(\text{T}_{\text{AMBIENT}}, \text{Irradiance, Manufacturing technology, Mounting method}) \]

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

---

irradiance

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

- **`pvlirradianciaoil(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.

- **`pvlirradianciaoiprojection(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.
  - **Returns:**
    - float or Series. Dot product of the solar vector and the surface normal.
PVLIB Help files
Help for functions and source code

irradiance

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

Input all angles in degrees.

**Parameters:**
- `surface_tilt`: float or Series.
- `surface_azimuth`: float or Series.
- `solar_zenith`: float or Series.
- `solar_azimuth`: float or Series.

**Returns:**
float or Series. Angle of incidence in degrees.

**Source code for pvlib.irradiance**

```python
from pvlib.irradiance import irradiance

# Example usage
irradiance = irradiance(surface_tilt, surface_azimuth, solar_zenith, solar_azimuth)
```

**pvlib.irradiance.aoi Projection**

Calculates the dot product of the solar vector and the surface normal.

Input all angles in degrees.

**Parameters:**
- `surface_tilt`: float or Series.
- `surface_azimuth`: float or Series.
- `solar_zenith`: float or Series.
- `solar_azimuth`: float or Series.

**Returns:**
float or Series. Angle of incidence in degrees.
PVLIB Help files

What’s new?

• 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.ABERRATION_CORRECTION` and `atmosphere.ABERRATION_CORRECTION` to enable code that can automatically determine which type of zenith data to use e.g. `location.get_airmass`

• Modify `solar` 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 `readm2`

• Fixes the import of tkinter on Python 3 (GH112)

• Add a decorator to skip `test_calparams_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 whatswnew/v0.2.2.txt and a table format in pvsystem. (GH123)

Contributors

• Will Holmgren

• pyElena21

• D.E.P.
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

• 5th: 9th May, 2016 Santa Clara, USA  ← SR not going
• 6th: 24-25th 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!
• Spare slides