PVLIB: Open Source Photovoltaic Performance Modeling Functions for Matlab and Python

Joshua S. Stein[1], William F. Holmgren[2], Jessica Forbess[3], and Clifford W. Hansen[1]

[1]Sandia National Laboratories, Albuquerque, NM 87122, USA;

[2]Department of Atmospheric Sciences, University of Arizona, Tucson, AZ, 85721, USA
[3]Sunshine Analytics, 288 3rd Street, Oakland, CA 94607, USA

***Abstract* — PVLIB is a set of open source modeling functions that allow users to simulate most aspects of PV system performance. The functions, in Matlab and Python, are freely available under a BSD 3 clause open source license. The Matlab version is maintained by Sandia and is available on the PV Performance Modeling Collaborative (PVPMC) website (pvpmc.sandia.gov). The Python version is available on GitHub with packages easily installable through conda and pip. New functions were released on the Matlab version 1.3 in January 2016 and are actively being ported to Python.**

I. Introduction

The PVLIB Toolbox began at Sandia National Laboratories in 2009 as an in-house project aimed at standardizing analysis methods used across Sandia's PV research groups. Previously, each researcher coded his own version of modeling functions and frequently the results differed between versions, either because of minor coding errors or differences in the interpretation of the original algorithms. By standardizing formats, applying version controls, making the codes open source, and distributing to a larger community of users, it was thought that the PVLIB could become a de-facto standard in the PV performance modeling community for understanding and validating models. In addition, a set of high level modeling and utility functions could be used to build application specific analysis tools. After more than five years and several versions later this vision has largely come to pass. PVLIB is used by more than a thousand users from academia and the commercial sector. In fact, the PVLIB effort helped to spawn the creation of the PV Performance Modeling Collaborative (PVPMC) [1], an open group of PV performance modelers that share ideas, information and help the PV community to improve the science of predicting PV system performance. The PVPMC has held numerous workshops in the US and has recently expanded its influence internationally as an activity of the International Energy Agency PVPS Task 13 on PV performance and reliability. New PVLIB functions are added either by contributions sent to the PVPMC (for the Matlab version) or added directly by users to the GitHub site (for the Python version). This paper reviews and summarizes the newest features of the PVLIB family of functions and is intended to introduce the packages to a new group of users. The first User's Group meeting for PVLIB was held in Santa Clara, CA as part of Sandia and EPRI's PV System Symposium. Over 40 people participated in this one-day meeting and contributed many ideas for keeping this project alive.

PVLIB offers functions that generally follow a standardized set of PV Performance Modeling steps that are outlined on the pvpmc.sandia.gov website. The general categories used for the Matlab version of the toolbox are as follows:

1. Time and location utilities
2. Irradiance and atmospheric functions
3. Irradiance translation functions
4. Photovoltaic system functions
5. Functions for parameter estimation for PV module models
6. Numerical utilities
7. Example scripts

II. PVLIB for Matlab

The latest version of PVLIB for Matlab (Version1.3) was released in January 2016. It includes the addition of a number of new functions that include the following:

- •*pvl_FSspeccorr* – Spectral mismatch modifier function contributed by First Solar based on precipitable water.
- •*pvl_calcPwat* = function to estimate precipitable water content
- •*pvl_huld* – PV performance model of Huld et al., 2011
- •*pvl_PVsyst_parameter_estimation* – function to estimate PVsyst module parameters from IV curves.
- •*pvl_calcparams_PVsyst* – Calculates the five parameters for an IV curve using the PVsyst model.
- •*pvl_desoto_parameter_estimation* - function to estimate Desoto module parameters from IV curves.

- •*pvl_getISDdata* - Functions to access ground measured weather data from NOAA's Integrated Surface Data network

An example using the first two functions to estimate the effect of changing relative humidity on spectral mismatch for x-Si and CdTe PV technologies is shown below. For both technologies an increase in relative humidity leads to an increase in relative performance in the form of a higher spectral mismatch modifier value. Note that the performance enhancement is greater for x-Si than for CdTe.

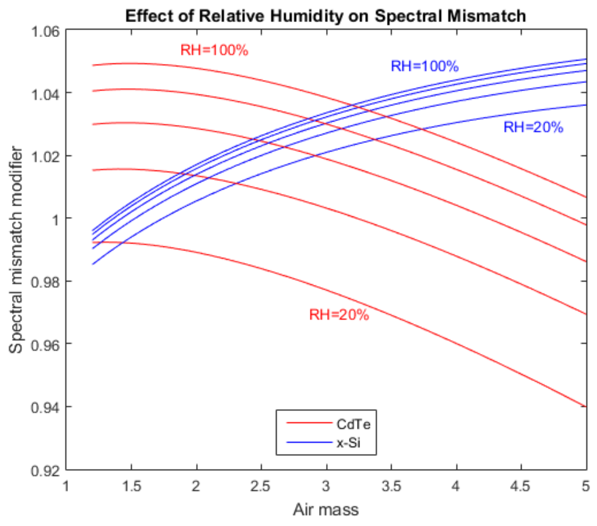

Fig 1. Example result using the *pvl_FSspeccorr* and *pvl_calcPwat* functions. A nearly identical plot can be created with the pvlib-python functions *calc_pw* and *first_solar_spectral_correction*.

The Matlab code used to make this plot is shown below (text annotations were added using Plot Tools in Matlab):

```
figure
index =0;
AMa = 1.2:0.1:5;
for rh = 20:20:100
    index=index+1;
    Pwat(index) = pvl_calcPwat(25,rh);
    MCdTe(:,index) =
pvl_FSspeccorr(Pwat(index), AMa, 'CdTe');
    MxSi(:,index) =
pvl_FSspeccorr(Pwat(index), AMa, 'xSi');
    plot(AMa,MCdTe(:,index),'r-')
    hold all
    plot(AMa,MxSi(:,index),'b-')
end
xlabel('Air mass')
ylabel('Spectral mismatch modifier')
title('Effect of Relative Humidity on
Spectral Mismatch')
legend('CdTe','x-Si','Location','South')
```

III. PVLIB FOR PYTHON

PVLIB-Python is a collaborative project that aims to bring the functionality of PVLIB for Matlab to the Python ecosystem. PVLIB-Python was initially released as an open source project by Sandia National Laboratories in 2014 [2], and attracted enough users and developers to become an independently administered project in 2015 [3]. PVLIB-Python is developed on GitHub.com and uses modern development practices such as version control, continuous integration testing, and automated documentation. These tools are described in more detail in [3].

The latest version of PVLIB-Python, 0.3.2, was released in May, 2016. It is available as a package using the *conda* and *pip* installation programs, as well being directly available on GitHub. The development of code in PVLIB-Python generally follows the implementation of the functions available in Matlab, as well as including additional functions and classes as needed and contributed by the PVLIB-Python user base. Some of these contributions make use of the broad Python ecosystem. For example, user Tony Lorenzo contributed a solar position calculation module that uses the Numba library to create faster compiled code with minimal changes to the standard Python syntax.

PVLIB-Python expands the standardization of solar performance modeling into the large system modeling space, reflecting the needs of its primary developers. PVLIB-Python allows researchers and operators of even large portfolios to automate sophisticated performance modeling for fleets of PV plants. Other PV plant analysis tools are typically based on straightforward performance ratios that result in inaccurate models due to shading or inverter clipping, among many additional system complexities. Using PVLIB-Python, operators can define a system in as much detail as desired to minimize discrepancies between measured and modeled performance. Using an open-source tool like PVLIB-Python helps operators maintain control over the modeling algorithms used, rather than relying on a black box process provided by a third-party monitoring system.

PVLIB-Python allows a user to define locations and system configurations including specific PV module and inverter model characteristics available from Sandia and the CEC. The system may be modeled with typical weather data or measured weather data. The measured irradiance can be compared to a clear sky model, as a data quality check on the sensor calibration. Loss factors similar to those found in PVsyst may be applied as needed to get a final expected energy, which can be compared to the actual measured energy.

As an example of the PVLIB-Python library, the following code can be used to nearly exactly reproduce Figure 1:

```
import numpy as np
import matplotlib.pyplot as plt
from pvlib.atmosphere import calc_pw,
first_solar_spectral_correction

airmass = np.linspace(1.2, 5)
rhs = np.linspace(20, 100, 5)
pws = calc_pw(25, rhs)
for pw in pws:
    cdte =
first_solar_spectral_correction(pw,
airmass, 'CdTe')
    xsi =
first_solar_spectral_correction(pw,
airmass, 'xSi')
    plt.plot(airmass, cdte, 'r-')
    plt.plot(airmass, xsi, 'b-')

plt.xlabel('Air mass')
plt.ylabel('Spectral mismatch modifier')
plt.title('Effect of Relative Humidity on
Spectral Mismatch')
plt.legend(['CdTe','x-Si'], loc='lower
center')
```

One of the goals for the developers is to try and keep both the Matlab and Python version as compatible as possible. This means that when a new function is added or existing function changed, it spawns a work flow that leads to the addition and or change to be reflected in the other version. This level of effort is not currently supported.

IV. PVSYST PARAMETER ESTIMATION EXAMPLE

PVsyst is the most used performance modeling application for commercial and utility-scale PV projects. However, the process of adding new modules to their performance database is somewhat shrouded in mystery, with each test lab using their own proprietary methods. In order to make this process more transparent Sandia added functions to PVLIB Matlab to estimate PVsyst module parameters from IV curves measured on a 2-axis tracker pointing at the sun. The procedure is described in detail elsewhere [4].

The PVLIB function, *pvl_PVsyst_paraeter_estimation()* is used in this example. It takes several inputs including:

- *IVCurves*: a structure containing IV measurements, effective irradiance, and cell temperatures.
- *Specs*: a structure containing the number of cells in series and the temperature coefficient of Isc.
- *Const*: a structure containing physical constants and reference conditions.
- Optional inputs include max number of iterations and tolerances for the calculations.

The function outputs a structure of PVsyst parameters. These parameters can be used to calculate IV curves for any combination of effective irradiance and cell temperature using the function, *pvl_calcparams_PVsyst()* and then *pvl_singlediode()*.

As an example of how this works, we analyzed a set of 3,585 IV curves measured in Albuquerque, NM over a period of five days on a 36 cell Mitsubishi c-Si module (Fig 2).

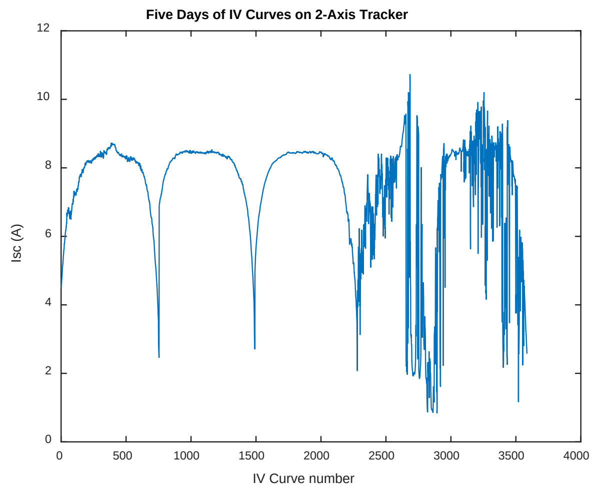

Fig. 2. Time series of short circuit current from 3,585 IV curves measured on a 33 cell Mitsubishi c-Si module on a 2-axis tracker in Albuquerque NM.

The calculation steps are rather simple and straightforward. After defining the inputs it is really just three lines of Matlab code to estimate parameters and generate predicted IV curves. The first statement results a structure PVsyst, that contains the estimated parameters.

```
[PVsyst oflag] =
pvl_PVsyst_parameter_estimation(IVCurves,
Specs, Const, maxiter, eps1, graphic);
```

The second statement performs the forward calculation of the five single diode parameters (some of which vary as a function of effective irradiance (`IVCurves.Ee)` and cell temperature `(IVCurves.Tc).`

```
[IL, Io, Rs, Rsh, nNsVth] =
pvl_calcparams_PVsyst([IVCurves.Ee],[IVCu
rves.Tc],Specs.aIsc,PVsyst);
```

The third statement simply calculated the IV curve (and associated maximum power point) for each irradiance and temperature condition.

```
Modeled = pvl_singlediode(IL, Io, Rs,
Rsh, nNsVth);
```

After running the parameter estimation process using these measured IV curves, irradiance, and temperature inputs, the PVsyst module parameters were calculated. The PVsyst model was then run for those same conditions and the results compared to what was measured. These results are shown below in Fig 3.

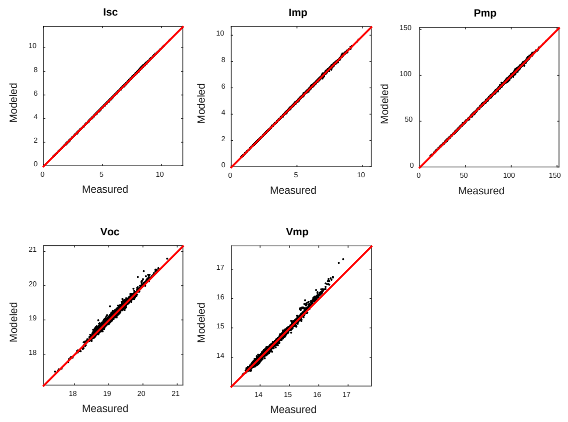

Fig. 3. Comparison between measured and modeled point on the IV curves. Good agreement indicates that the parameter estimation process was successful.

To test the effect of using clear or cloudy data on the goodness of fit, we ran two variants on the parameter estimation for this example. In the first case, we fit the model to data from the first three days, which were clear and then tested to against the data from all the days. The results of this are shown graphically in Fig. 4.

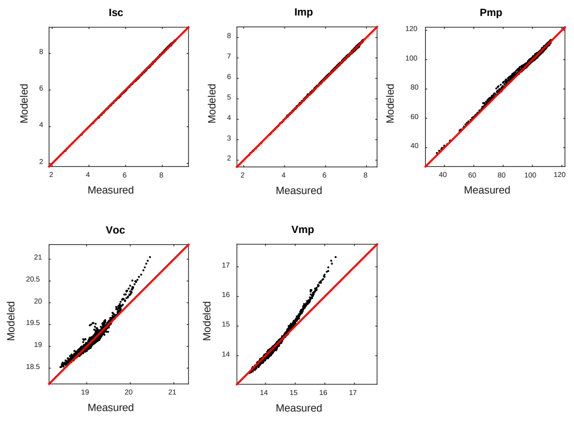

Fig. 4. Comparison between measured and modeled fit to first three days, which were clear.

The second variant fit the model using the data from the last two days, which were partly cloudy and then tested to against the data from all the days. The results of this are shown graphically in Fig. 5.

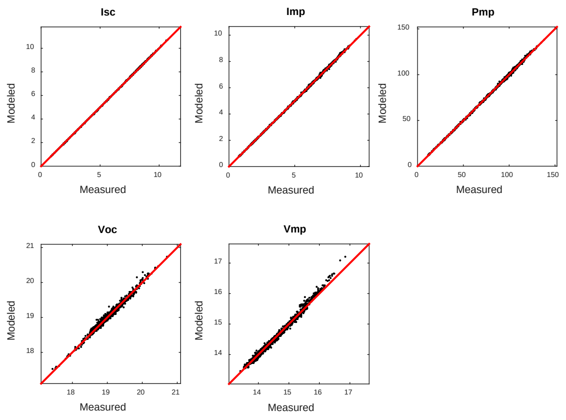

Fig. 5. Comparison between measured and modeled fit to first three days, which were partly cloudy.

The model parameters derived from all the data and from the cloudy data result in the best models for all key points on the IV curves. Model parameters derived from only clear days exhibit higher errors for both Voc and Vmp, especially at high values of voltage. This is likely due the fact that during partly cloudy periods, effective irradiance values reach higher values and thus help to represent voltage behavior, which is correlated with the log of irradiance.

The function: *pvl_desoto_parameter_estimation* operates in a very similar way as the one for the PVsyst. The only difference is in the set of equations and parameters that describe how the five single diode model parameters vary with irradiance and temperature. These equations and parameters differ between the PVsyst and DeSoto models.

This brief example shows the value of having an open source set of tools for PV performance modeling tasks. They allow critical examination of routine tasks, which ensure that the most accurate results are obtained.

V. Example finding weather data using PVLIB

Version 1.3 of PVLIB-Matlab includes new functions, *pvl_getISDdata* and *pvl_reasISH*, which allow the user to find measured weather data for a selected year anywhere in the workd. The first function takes as input a latitude, longitude, and year and returns measured weather data that is available from the nearest station from the NOAA's Integrated Surface Database (ISD) [6] for that year. The ISD covers the entire world (Fig 7).

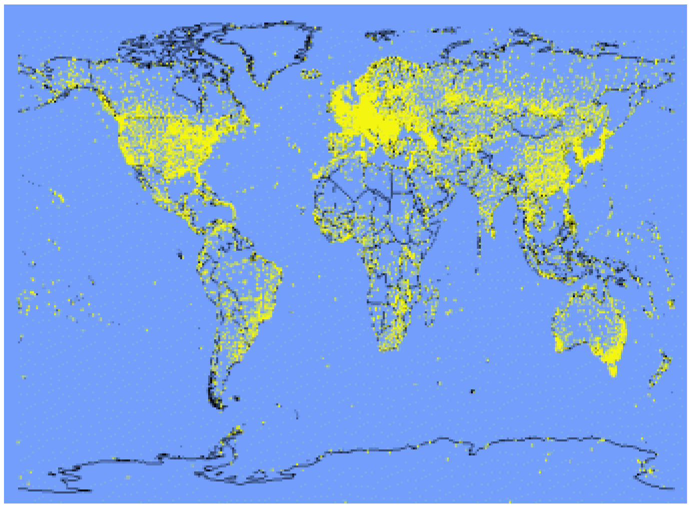
Fig. 6. Map showing the locations of ISD stations [7]

As an example, we chose 2009 for a site in Williston, Vermont at 44.465 deg N and -73.105 deg W. The commands are listed and explained below:

```
archive = '..\Example Data';

fname = pvl_getISDdata(44.465,-
73.105,2009,archive);
```

These commands assign an archive path and then find the nearest ISD station in the archive to the specified coordinates. In this example we use the archive that is included with PVLIB in the Example Data folder. If it was not included, the function would download a new archive. The function returns the file name for the nearest station. In this case:

```
fname = ‘726170-14742-2009’
```

To read the data file we use the function: *pvl_readISH*:

```
data = pvl_readISH([archive '\' fname]);
```

This file contains all the available data for the nearest station and selected year. Figure 7 is an example plot of the air temperature data retrieved from a station located 4 km away at the Burlington, VT airport. The time interval between measurements is not uniform but varies from several minutes to 1 hour. Figure 7 plots the data vs. an index.

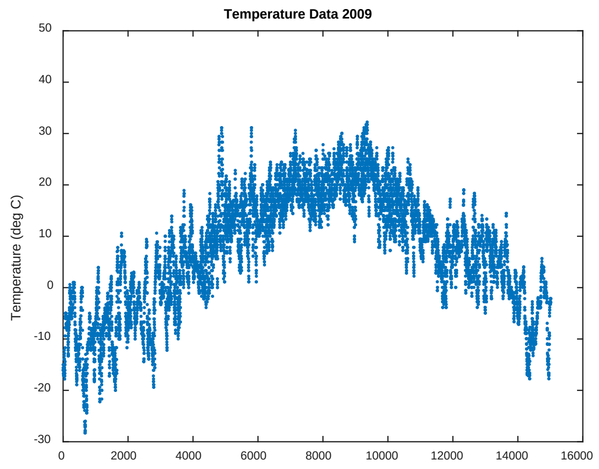

Fig. 7. Plot of air temperature data near the point of interest from 2009 in Williston, VT.

VI. Future Plans for Updates

The PVLIB-Python community is designing and implementing new features to simplify basic full system modeling tasks, simplify some function inputs, and make the library more flexible. The next version of the library will support fully procedural programming (no objects needed), as well as fully object-oriented programming. While this would be difficult or impossible in many languages, the flexible design of the core Python language makes it feasible to accomplish this goal. We believe that this flexibility will attract a broader user and developer base and therefore improve the long term health of the library. This development is taking place through public issues and pull requests on our GitHub page, and we encourage readers to contribute.

One example of a tool that is currently being developed at Sandia that makes use of PVLIB-Python is Pecos [5]. It is an open source utility designed for monitoring time series based datasets, especially from PV systems. This tool allows an analyst to easily monitor the performance of a fleet of PV systems using the advanced features of PVLIB-Python. It includes the ability to group similar data streams (e.g., plane-of-array irradiance, or string currents) and design custom analyses and plots for displaying results. It provides a wide variety of powerful data quality checks and alarms and generates graphic reports that are delivered as either webpages or attached to email notifications.

PVLIB-Matlab continues to be developed at Sandia and accepts new code contributions through the PVPMC. The most recent version of the code is now available from GitHub (https://github.com/sandialabs/MATLAB_PV_LIB) and from the PV Performance Modeling Collaborative website (https://pvpmc.sandia.gov/applications/pv_lib-toolbox/). Many of the new functions from Sandia reflect current funded projects being supported at the labs. Current research projects at Sandia include developing models for predicting bifacial PV module and system performance and modeling the IV characteristics of CIGS thin film modules, which are not that well represented by current equivalent circuit diode models. External contributions are also always welcome.

A challenge for sustaining this code base and continuing to make improvements lies in ensuring there are enough resources to support the writing of new functions, documentation, and test cases as well as integrating these functions into new releases and responding to bug reports. At present this work is primarily being supported by Sandia (Matlab) and University of Arizona (Python), although users from other institutions have also made contributions. In the future, this loose organization may need to change. The developers are looking into alternative models to ensure that the community supported software can thrive. These models may include industry donations of developer time, industry contracts with freelance developers, universities, and labs to implement specific features, and grants from government agencies.

VII. Acknowledgements

PVLIB in both Matlab and Python would not be successful if it were not for its active and generous development and user community. Specific contributions are attributed in the function code and help files and a list of PVLIB-Python contributors is available in the GitHub site.

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy’s National Nuclear Security Administration under contract DE-AC04-94AL85000.

WF Holmgren thanks the Department of Energy (DOE) Office of Energy Efficiency and Renewable Energy (EERE) Postdoctoral Research Award and Tucson Electric Power for support.

References

[1] J. S. Stein, “The photovoltaic performance modeling collaborative (PVPMC),” in *Photovoltaic Specialists Conference*, 2012.

[2] R.W. Andrews, J.S. Stein, C. Hansen, and D. Riley, “Introduction to the open source pvlib for python photovoltaic system modelling package,” in *40th IEEE Photovoltaic Specialist Conference*, 2014.

[3] W.F. Holmgren, R.W. Andrews, A.T. Lorenzo, and J.S. Stein, “PVLIB Python 2015,” in *Photovoltaic Specialists Conference*, 2015.

[4] Hansen, C. Estimating Parameters for the PVsyst Version 6 Photovoltaic Module Performance Model. Albuquerque, NM, Sandia National Laboratories. SAND2015-8598, 2015.

[5] K.A. Klise and J.S. Stein, “Automated Performance Monitoring for PV Systems using Pecos” in *43rd IEEE Photovoltaic Specialist Conference*, 2016 (abstract submitted).