PVLIB Python 2015

William F. Holmgren*, Robert W. Andrews†, Antonio T. Lorenzo‡,
Joshua S. Stein§

*Department of Atmospheric Sciences, University of Arizona, Tucson, AZ, 85721, United States
†Heliolytics, 483 Bay St. Toronto, ON, M5G2C9, Canada
‡College of Optical Sciences, University of Arizona, Tucson, AZ, 85721, United States
§Sandia National Laboratories, Albuquerque, NM, 87185, USA

***Abstract*—We describe improvements to the open source PVLIB-Python modeling package. PVLIB-Python provides most of the functionality of its parent PVLIB-MATLAB package and now follows standard Python design patterns and conventions, has improved unit test coverage, and is installable. PVLIB-Python is hosted on GitHub.com and co-developed by GitHub contributors. We also describe a roadmap for the future of the PVLIB-Python package.**

***Index Terms*—PV modeling, software, data analysis, performance modeling**

I. Introduction

The PVLIB Toolbox is a well-established MATLAB library for photovoltaic modeling and analysis [1]. It was originally developed at Sandia National Laboratories and has been expanded by contributions from members of the Photovoltaic Performance and Modeling Collaboration (PVPMC). While MATLAB remains a common choice in many public and private laboratories, the popularity of Python has grown tremendously in the last decade. Python is now the language used in introductory programming courses at a number of top universities [2], [3]. It is elegant, easy to read and write, portable across platforms, free and open source, and it has a large scientific computing community. With the appropriate scientific packages installed (NumPy, SciPy, Matplotlib, statsmodels, pandas), Python provides a powerful alternative to MATLAB and R. The scientific Python stack also enables the use of a single language for the entire data collection, processing, and analysis workflow, which can result in faster development with fewer bugs.

Andrews et. al. [4] introduced the PVLIB-Python toolbox in 2014 and outlined its three main principles:

1) Take advantage of the Python programming language, to ensure free access to academic and commercial users.
2) Designed for collaborative development, and backed by a rigorous method to include the contributions of authors and researchers into the package.
3) Backed by a full testing and validation suite to ensure stability of the package and to allow for validation of model results against real-world performance data.

Overall, the goal of this package is not to supersede other established commercial and public PV modeling packages, such as Helioscope, PVSyst, SAM, and PVWatts. Instead, PVLIB-Python adds a flexible, accessible, and collaboratively-developed analysis package which can be utilized to derive deep insights about the performance of PV systems and the tools used to model them. By providing a code-level and modular approach to system modeling, users are able to model and analyze each portion of the PV system performance chain, and are able to utilize the significant data analysis capabilities of Python to analyze large data sets. The PVLIB-Python source code is hosted on GitHub [5]. The source code to generate the figures in this manuscript and poster is also hosted on GitHub [6].

The first PVLIB-Python implementation [7] succeeded in providing nearly all of the PVLIB-MATLAB functionality in Python. It also succeeded in establishing collaborative development environment on GitHub. Over a dozen forks (independent copies) of the project now exist on GitHub. Many of these users have contributed substantial changes to the source code, cataloged issues, or discussed ways of improving the code through GitHub. Users do not need significant experience with Python to make significant contributions to PVLIB-Python. In fact, making small contributions to PVLIB-Python source code, unit tests, and documentation can be an effective way for new users to learn how to use Python and GitHub.

II. Growth of the PVLIB-Python package

The initial release of the package provided a direct translation and adaptation of most of the PVLIB-MATLAB version of the code. Though functional, this translation exposed a number of problems for using PVLIB in a Python environment. We focused on improving the following three issues that existed in the initial release:

- The package did not conform to standard Python design patterns and conventions.
- The package test coverage was poor.
- The package did not have an install script.

We summarize the largest changes and improvements relevant to these issues below, but we encourage readers to visit the PVLIB-Python GitHub issues webpage for comprehensive discussions regarding these changes.

A. Python design patterns, conventions, and the Zen of Python

The original PVLIB-Python package implemented nearly all of PVLIB-MATLAB in Python, but it did not generally

use idiomatic Python. Idiomatic Python is often referred to as *pythonic* [8], or conforming to the *Zen of Python* [9]. These fanciful words may imply simply a matter of taste or preference for how software should be written, but the implications for the package are actually much larger. First, we chose to implement PVLIB in the Python language for reasons that centered on the fundamental nature of Python and the associated scientific Python libraries, described above and in [4]. It is reasonable for new users and developers to expect a Python package to look and behave like other Python packages, and for it to be written using the same design patterns. Second, pythonic Python is usually simpler and uses more of the language's well-tested and built-in functionality. This makes the code easier to understand and less likely to contain hidden bugs. In order to address this, we made PVLIB-Python more pythonic in the following ways.

1) Package structure: The original PVLIB-Python, as well as the current PVLIB-MATLAB, contains all functions in files, or modules, that are named the same as the function. This pattern causes serious module import problems in Python, and it neglects the advantages of grouping similar functions into a single module for logical consistency. We now group functions of a similar type into one module. For example, the TMY reading functions `pvl_readtmy2` and `pvl_readtmy3` now reside in a single module `tmy`. Next, we removed `pvl_` from all functions and modules – after all, they are all contained in a packaged named `pvlib`. A TMY reading function then becomes `pvlib.tmy.readtmy3`. Similar changes have been applied library-wide, yielding a more pythonic library structure that improves readability and organization. Several additional PVLIB-MATLAB features, such as single axis tracker models (see Figure 1) were also ported to PVLIB-Python.

2) Pythonic modification of design patterns: Several design patterns existed in the Python package which were holdovers from PVLIB-MATLAB. For example, PVLIB-MATLAB uses a struct to represent a location. Python does not have a C-like struct, but the initial PVLIB-Python attempted to keep the same pattern by creating a struct-like object. By replacing the awkward Python version of a struct with a simple `pvlib.location.Location` class we obtained more readable and more functional code. Objects of this `Location` class have added functionality such as more flexible constructors, better timezone handling, and nicer string representations. Most importantly, the `Location` class can be extended by users for their own purposes. For example, a user may define her own `SolarPlant` class that inherits from the `Location` class. `SolarPlant` objects could then interact with the rest of PVLIB in the same ways as `Location` objects, but also have additional user-defined attributes and functionality. Additional changes include using making extensive use of `NaN` or `inf` values instead of setting values to 0, and applying PEP8 style and naming conventions [10] to some of the code. Furthermore, most automatic input variable checking on design functions was removed as it is largely unnecessary in a Python environment and increased the complexity of the code.

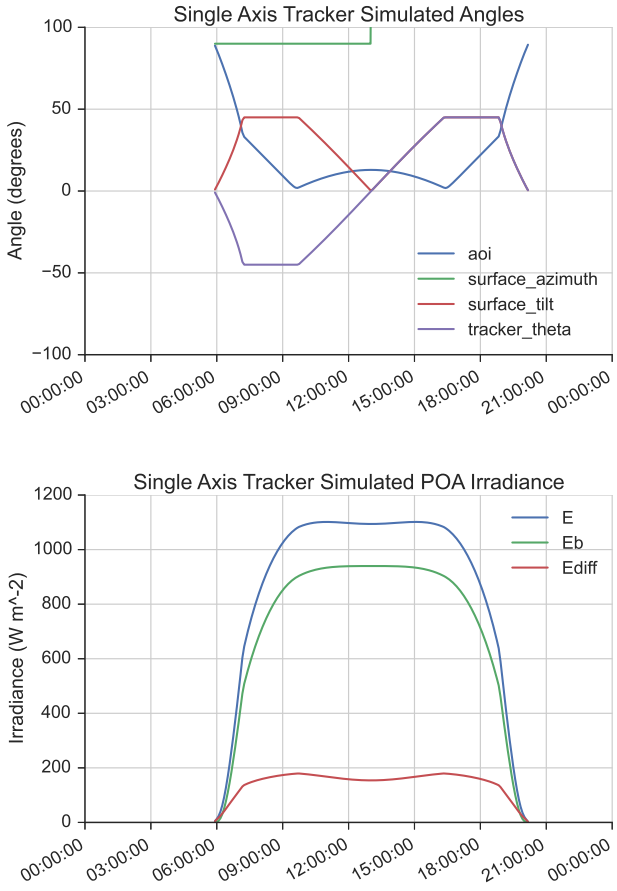

Fig. 1. PVLIB-Python simulation of a single axis tracker, with backtracking, located near Albuquerque, NM, for June 1, 2015. The simulation outputs include (top) the angle of incidence (blue), panel surface azimuth (green), panel surface tilt (red), and angle of rotation about the tracking axis (purple) and (bottom) total plane of array irradiance (blue), beam component of plane of array irradiance (green), and diffuse plane of array irradiance (red). This example simulation used the Ineichen model to generate clear sky DNI, GHI, and DHI, the Hay-Davies model to generate the diffuse plane of array irradiance, and an isotropic ground diffuse model with an albedo of 0.25. This example may be compared to a similar simulation in the PVLIB-MATLAB documentation.

3) Logging and debugging: It is standard Python practice to use Python's built-in logging capabilities instead of print statements, so we replaced print statements with logging calls. We are also now relying on built-in Python exceptions to alert the user to problems. Using Python exceptions makes the PVLIB-Python library easier to integrate into a user's own programs.

B. Accessibility for MATLAB users

Rewriting the code to make it more pythonic leaves us with a significant problem: the package becomes less familiar to PVLIB-MATLAB users. We have attempted to lessen this problem in several ways. First, we have provided extensive documentation, discussed more below, in the form of webpages and IPython notebooks that demonstrate where to find key functionality, how to tie together different components,

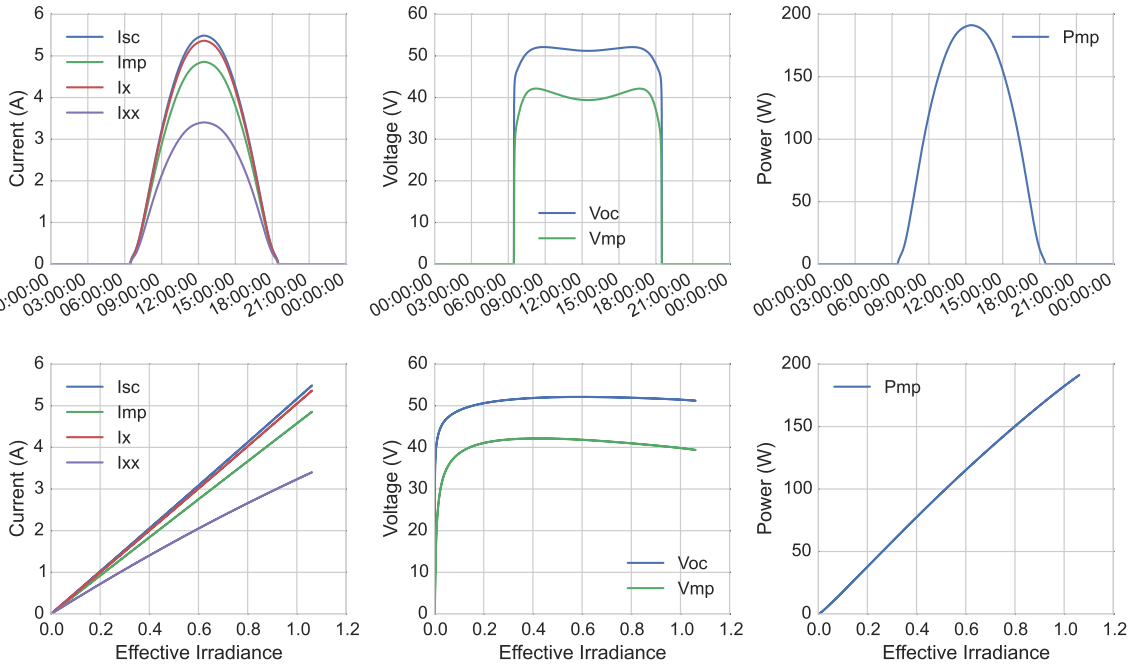

Fig. 2. Simulation of a fixed-tilt PV system on April 1, 2015, in Tucson, AZ, using PVLIB-Python's implementation of the Sandia Array Performance Model. The top row of subfigures shows currents, voltages, and maximum power as a function of time of day, while the bottom row shows the same as a function of effective irradiance. PVLIB-Python was used to load the Sandia Module Database from NREL's website, calculate solar position, clear sky data, airmass, cell temperature, and module temperature, and finally run the Sandia Array Performance Model in 9 lines of code. Detailed simulation parameters may be found online [6].

and how to write good Python in the context of problems that PVLIB-MATLAB users are familiar with. Second, wildcard import statements can be used to put the common PVLIB-MATLAB functions in the namespace of the Python script. For example, the TMY reading functions can be imported using `from pvlib.tmy import *`. The PVLIB-Python GitHub community has discussed adding the ability to import all of the frequently used top-level functions using a command such as `from pvlib.api import *`. Wildcard imports can sometimes lead to namespace problems and are often avoided in Python, but this functionality can provide PVLIB-MATLAB users with a more familiar interface while gradually becoming accustomed to Python design patterns.

C. Testing and continuous integration

Andrews et. al. described the need for both functional (does the code return a value or crash) and physical (does the code return the correct value) tests of the PVLIB code [4]. These tests are often called unit tests, in reference to the fact that there is ideally one test for each unit of code. Here, a unit of code is typically a function, or a function called with a specific set of parameters. PVLIB-Python now uses the `nose` package [11] to make writing tests faster and easier. This package has been used to increase the testing coverage of the code to above 90%.

Next, we adopted a continuous integration service, TravisCI, to automatically test the code every time a contributor submits a pull request to merge their changes into the master PVLIB-Python. This helps to ensure that a contributor's changes to one part of the library does not inadvertently break another part of the library. A comprehensive test suite coupled with continuous integration services also makes it relatively easy to test a library against many permutations of user environments. As of June 1, 2015, we automatically test the PVLIB-Python package against Python versions 2.7 through 3.4, and pandas versions 0.13.1 through 0.16.1.

Writing new tests is a great way for new users to contribute to PVLIB-Python. We particularly need physical tests, ideally benchmarked against other PV modeling programs.

D. Installation

Most popular Python packages use one of several tools to enable users to install the package into their Python environment. Once installed, a Python package can be accessed regardless of the directory in which the user started the Python interpreter. The original PVLIB-Python did not contain a

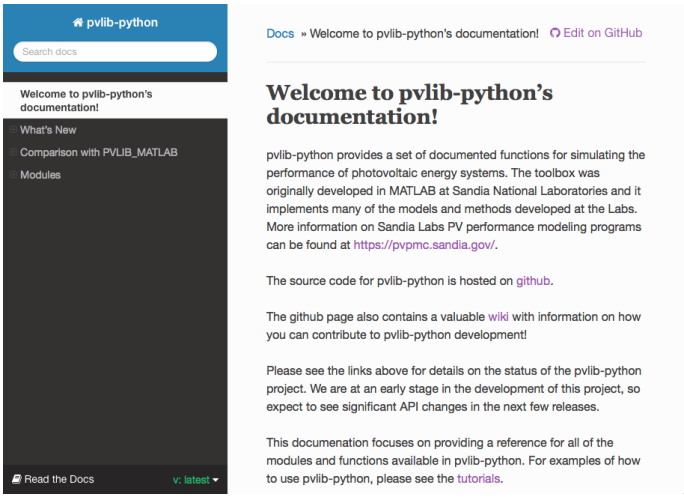

Fig. 3. Screenshot of PVLIB-Python documentation homepage hosted at http://pvlib-python.readthedocs.org.

usable installation script, which made using it and developing it more challenging. We wrote a `setup.py` script to enable PVLIB-Python to be installed and to be developed more effectively. We also added PVLIB-Python to the Python Package Index (PyPI), the most common resource for installing Python packages [12]. PVLIB-Python can now be installed with a simple `pip install pvlib-python` command. Future versions of PVLIB-Python will be installable using the `conda` package manager.

E. Documentation

Good documentation is essential to the success of any software. We documented PVLIB-Python in two ways. First, we created html and pdf documentation of the Python modules and functions. Most of this documentation is an edited version of the PVLIB-MATLAB documentation. We also documented the major differences between the Python and MATLAB projects. PVLIB-Python uses the standard Python tools sphinx and numpydoc to build html and pdf documentation from function and module docstrings. We use readthedocs.org to automatically build new versions of the documentation every time a change is made to the source code in the GitHub repository. Figure 3 shows a screenshot of the documentation homepage. The second form of documentation is a collection of IPython notebooks. IPython notebooks provide an informative combination of explanatory text and inline, executable code. These notebooks can be viewed and downloaded at the GitHub project page or using the nbviewer.org tool. We strongly encourage the community to contribute to both forms of the PVLIB-Python documentation.

III. Roadmap

We propose one possible short-term roadmap for PVLIB-Python, and encourage the community to suggest improvements. We hope that PVLIB-Python administrative duties will rotate through the community on an annual to biannual basis.

1) Initial PVLIB-Python implementation released on GitHub under Sandia organization (completed in June, 2014) [7].
2) Establish a new GitHub organization, `pvlib`, to collaboratively administer the official PVLIB-Python project (completed February, 2015) [5].
3) Put documentation on readthedocs.org [13] and use TravisCI (completed February, 2015).
4) First PVLIB-Python release on PyPI [12] (completed April, 2015).
5) Establish PVLIB-Python governance rules. Scientific Python packages, in particular IPython IPEP 29 [14], may be a good resource.
6) Bring PVLIB-Python up to date with PVLIB-MATLAB 1.2.
7) Reach 100% functional test coverage.
8) Release new version.
9) Develop and implement common physical tests for PVLIB-Python and PVLIB-MATLAB.
10) Encourage users to contribute new models, improve existing model implementations, testing, and documentation.
11) Release new version.
12) Rotate PVLIB-Python administrative duties.

IV. Use cases

We intend to establish a GitHub wiki to catalog PVLIB-Python use cases. Such a wiki may be modeled on the IPython project's *A gallery of interesting IPython Notebooks* and *Projects using IPython* [15].

V. Conclusion

We described significant improvements to the PVLIB-Python package between the period June 2014 and May 2015 that make the library easier to use, easier to maintain, better tested, and better documented. Much more work remains, and we encourage readers to visit the GitHub project development website [5].

Acknowledgment

The authors gratefully acknowledge Sandia National Laboratories for the initial development of PVLIB-MATLAB and PVLIB-Python and the ongoing contributions of many others to the project. A list of PVLIB-Python contributors may be found on the GitHub repository [5] and in the online documentation [13]. 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. WFH thanks the Department of Energy (DOE) Office of Energy Efficiency and Renewable Energy (EERE) Postdoctoral Research Award for support. ATL thanks the University of Arizona Renewable Energy Network for support.

REFERENCES

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

[2] F. Pérez, B. E. Granger, and J. D. Hunter, "Python: An ecosystem for scientific computing," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 13–21, 2011.

[3] P. Guo. Python is now the most popular introductory teaching language at top U.S. universities. [Online]. Available: http://goo.gl/2wxOk6

[4] R. W. Andrews, J. S. Stein, C. Hansen, and D. Riley, "Introduction to the open source PV LIB for Python Photovoltaic system modelling package," in *40th IEEE Photovoltaic Specialist Conference*, 2014.

[5] W. F. Holmgren, B. Mueller, R. W. Andrews, and GitHub. pvlib/pvlib-python. [Online]. Available: https://github.com/pvlib/pvlib-python

[6] W. F. Holmgren, R. W. Andrews, A. T. Lorenzo, and J. S. Stein. pvlib/pvsc2015. [Online]. Available: https://github.com/pvlib/pvsc2015

[7] R. W. Andrews and GitHub. Sandia-Labs/PVLIB_Python. [Online]. Available: https://github.com/Sandia-Labs/PVLIB_Python

[8] M. Faassen. What is pythonic? [Online]. Available: http://blog.startifact.com/posts/older/what-is-pythonic.html

[9] T. Peters. PEP 20 – the zen of Python. [Online]. Available: https://www.python.org/dev/peps/pep-0020/

[10] G. van Rossum and N. Coghlan. PEP 8 – style guide for Python code. [Online]. Available: https://www.python.org/dev/peps/pep-0008/

[11] nose. [Online]. Available: https://nose.readthedocs.org/en/latest/

[12] W. F. Holmgren, B. Mueller, R. W. Andrews, and GitHub. PyPI/pvlib-python. [Online]. Available: https://pypi.python.org/pypi/pvlib

[13] ——. pvlib-python documentation. [Online]. Available: http://pvlib-python.readthedocs.org

[14] IPython. IPEP 29: Project governance. [Online]. Available: https://github.com/ipython/ipython/wiki/IPEP-29%3A-Project-Governance

[15] IPython. A gallery of interesting IPython notebooks and projects using IPython. [Online]. Available: https://github.com/ipython/ipython/wiki/