PVLIB Python 2015

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

*Department of Atmospheric Sciences, University of Arizona, Tucson, AZ, United States
†Institute Name, City, State/Region, Mail/Zip Code, Country
‡College of Optical Sciences, University of Arizona, Tucson, AZ, United States
§Institute Name, City, State/Region, Mail/Zip Code, Country

***Abstract*—We describe improvements to the open source PVLIB-Python modeling package. PVLIB-Python currently provides most of the functionality of its parent PVLIB-Matlab package. The package is hosted on GitHub.com and co-developed by GitHub contributors. We also describe several PVLIB-Python use cases.**

***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 [3]. 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 [4][more? see Andrews]. It is elegant, easy to read and write, portable across platforms, powerful, free and open source, and it has a large scientific computing community. With the appropriate scientific packages installed (NumPy, SciPy, Matplotlib, statsmodels, pandas), Python can easily replicate most Matlab and R functionality. This means a single language can be used for the entire data collection, processing, and analysis workflow reducing the mental overhead of programming in multiple languages.

Andrews et. al. [2] 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.

The first PVLIB-Python implementation succeeded in providing nearly all of the PVLIB-Matlab functionality in python. It also succeeded in establishing collaborative development environment on GitHub [5]. 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.

However, the original implementation suffered from several deficiencies that we addressed, or are actively working to address, and describe below.

II. Improvements to PVLIB-Python

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 discuss the largest changes and improvements relevant to these issues below, but we encourage readers to visit the PVLIB-Python GitHub issues page for a more comprehensive discussion.

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* [9], or conforming to the *Zen of Python* [7]. 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 [2]. 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 well-tested and built-in language's functionality. This makes the code easier to understand and less likely to contain hidden bugs.

We made PVLIB-Python more pythonic in many ways. The most obvious change may be the package structure. The original PVLIB-Python, as well as the current PVLIB-Matlab, contains all functions in files, or modules, that are name 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.

Next, we replaced several design patterns that are 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, so it can interact with the rest of PVLIB in expected ways, but also has additional user-defined attributes and functionality.

Additional pythonic changes include using python's logging capabilities instead of print statements, relying on built-in python exceptions to alert the user to problems, making extensive use of `NaN` or `inf` values instead of setting values to 0, and applying PEP8 style and naming conventions [10] to more of the code.

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 in the form of webpages and, perhaps most importantly, IPython notebooks that demonstrate where to find key functionality, how to tie together different components, and how to write good python in the context of problems that PVLIB-Matlab users are familiar with. Second, we propose a wildcard import statement pattern that puts the common PVLIB-Matlab functions in the namespace of the python script. For example, the TMY import functions described below can be accessed in the traditional manner with the command `from pvlib.api import *`, or more specifically, `from pvlib.tmy 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.

B. 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 [2]. Unfortunately, the first version of the library had few meaningful tests of either type. These tests are often called unittests, 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.

Of course, there is no substitute for simply taking the time to systematically write more tests, and we have done this. We adopted the `nose` package [8] to make writing tests faster and easier. We estimate that fewer than 10% of the code was functionally tested in the initial release, but this is now at 75%.

Next, we adopted a *continuous integration* service, TravisCI, to automatically test the code every time that a user pushes a new commit to a GitHub repository or submits a pull request to merge their changes into another repository. For an example of TravisCI applied to PVLIB, please see the UARENForecasting/PVLIB_Python fork [6]. This service will soon be applied to the official repository.

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.

C. 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 useable 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. This also enables PVLIB-Python to be placed on the Python Package Index (PyPI), the most common resource for installing python packages.

III. Roadmap

As of this writing, the roadmap for PVLIB-Python is unfinished. We propose one possible roadmap, but we encourage the community suggest improvements. We believe that the following roadmap may reasonably take between 6 and 9 months.

1) Initial PVLIB-Python implementation released on GitHub (completed in June, 2014).
2) Initial PVLIB-Python released to PyPI and documentation released to readthedocs.org and/or pvpmc.sandia.gov.
3) Establish a new GitHub organization, perhaps named `pvlib`, to administer the official PVLIB-Python project.
4) Merge the developments described here into the master branch.
5) Release updated package on PyPI, release updated documentation on readthedocs.org and/or pvpmc.sandia.gov.
6) Establish PVLIB-Python governance rules. Scientific python packages, in particular IPython IPEP 29 [11], may be a good resource.
7) Bring PVLIB-Python up to date with PVLIB-Matlab 1.2.
8) Reach 100% functional test coverage.

9) Release new version.
10) Develop and implement common physical tests for PVLIB-Python and PVLIB-Matlab.
11) Release new version.

Due to the major changes in governance proposed above, any further discussion would be pure speculation.

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* [12].

V. Conclusion

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

Acknowledgment

The authors gratefully acknowledge contributions of many others to PVLIB-Matlab and PVLIB-Python. A list of PVLIB-Python contributors may be found on its GitHub repository [5]. 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. 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 Energys National Nuclear Security Administration under contract DE-AC04-94AL85000.

References

[1] **WH: Order references correctly.**
[2] Robert W. Andrews, Joshua S. Stein, Clifford Hansen, and Daniel Riley, "Introduction to the Open Source PV LIB for Python Photovoltaic System Modelling Package", *in 40th IEEE Photovoltaic Specialist Conference*, 2014.
[3] J. S. Stein, "The Photovoltaic Performance Modeling Collaborative (PVPMC)", *Photovoltaic Specialists Conference*, 2012.
[4] P. Guo, "Python is Now the Most Popular Introductory Teaching Language at Top U.S. Universities", http://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-us-universities/fulltext Accessed January 22, 2015.
[5] GitHub contributors, "Sandia-Labs/PVLIB_Python", https://github.com/Sandia-Labs/PVLIB_Python Accessed January 22, 2015.
[6] W. F. Holmgren, A. T. Lorenzo, and GitHub contributors, "UARENForecasting/PVLIB_Python", https://github.com/UARENForecasting/PVLIB_Python Accessed January 22, 2015.
[7] T. Peters, "PEP 20 – The Zen of Python", https://www.python.org/dev/peps/pep-0020/ Accessed January 22, 2015.
[8] nose contributors, https://nose.readthedocs.org/en/latest/ Accessed January 22, 2015.
[9] M. Faassen, "What is Pythonic?", http://blog.startifact.com/posts/older/what-is-pythonic.html Accessed January 22, 2015.
[10] G. van Rossum and N. Coghlan, "PEP 8 – Style Guide for Python Code", https://www.python.org/dev/peps/pep-0008/ Accessed January 22, 2015.
[11] IPython IPEP 29 contributors, "IPEP 29: Project Governance", https://github.com/ipython/ipython/wiki/IPEP-29%3A-Project-Governance Accessed January 22, 2015.
[12] IPython wiki contributors, "A gallery of interesting IPython Notebooks" and "Projects using IPython", https://github.com/ipython/ipython/wiki/ Accessed January 22, 2015.