Introduction to the Open Source PV_LIB for Python Photovoltaic System Modelling Package

Robert W. Andrews [1], Joshua S. Stein [2], Clifford Hansen[2], and Daniel Riley [2]
[1] Calama Consulting, Toronto, Ontario, M5T1B3, Canada
[2]Sandia National Laboratories, Albuquerque, NM, 87185, USA

***Abstract*—The proper modeling of Photovoltaic(PV) systems is critical for their financing, design, and operation. PV_LIB provides a flexible toolbox to perform advanced data analysis and research into the performance modeling and operations of PV assets, and this paper presents the extension of the PV_LIB toolbox into the python programming language. PV_LIB provides a common repository for the release of published modeling algorithms, and thus can also help to improve the quality and frequency of model validation and inter comparison studies. Overall, the goal of PV_LIB is to accelerate the pace of innovation in the PV sector.**

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

I. Introduction

THE proper modeling of a photovoltaic (PV) system is critical to its technical and financial success. Many commercial integrated software packages are available and accepted in industry: for example PVsyst, Helioscope, SAM, PVWatts and others provide the ability for designers to asses the potential output and revenue from a PV system. These packages are particularly adept at answering the following design questions:

- Overall system output given irradiance information.
- Sizing and specification of major components (inverters, panels, combiners).
- Shading and loss factors analysis.

And they are able to provide these results in a form which is accepted in the technical and financial industries (i.e bankable). This is due to the traceability of their algorithms and the familiarity of the industry with these outputs.

However, because of the integrated nature of these simulations packages, there is a lack in flexibility of simulations which the package can perform. This means that designers and researchers cannot create customized and integrated tools for exploring advanced topics in PV performance analysis.These advanced tools are often required to perform detailed analyses of system performance, and to allow innovation in the deployment of PV technologies. Examples of questions which are not easily answered with integrated modeling packages are:

- Comparisons of performance outputs using varying models and assumptions
- Large scale parametric analyses of multiple datasets (though some newer models are allowing this)
- Ability to modify, customize, and update modeling algorithms
- Automated integration into external workflows, eg. automated performance assessments
- Ability to view intermediary modeling results and perform on-the-fly statistical analyses

Thus, those requiring the ability to perform more advanced data analysis will generally create custom algorithms in programs such as MATLAB, R, Python, or in spreadsheet programs. These algorithms are generally based on published literature or understood industry best practice, however the process of creating new algorithms based on these sources is time consuming and can lead to errors or mis-interpretations, and are generally difficult to individually validate.

In order to address these issues, Sandia National Laboratories has developed PV_LIB [1], a collaboratively developed open source PV modeling environment. This environment consists of modeling algorithms which can be used in isolation or in combination to analyze all portions of the PV system modeling work flow.

II. The PV_LIB concept

The vision for PV_LIB is to become a standard repository for high quality PV related analysis algorithms. Put simply, an algorithm can be published in a journal, conference, or white paper, and the code can be published through PV_LIB.

This code is open-source, collaboratively developed and validated, and it is hoped that the release of this code will accelerate the development of the PV industry for the following reasons:

- **Create a linkage between researchers and implementors** Authors of new algorithms can present them in a format which can be adopted into modeling work flows quickly and and at a low cost
- **Provide validated tools to groups which do not have the ability to develop them** Many industry and research groups do not have the ability to develop algorithms for advanced data analysis, and are limited by the capabilities of spreadsheet software and commercial integrated analysis packages
- **Increase the efficiency of PV research** Many research projects in PV performance require the development of an initial testing and modeling suite before new research can begin. By providing a high quality and validated toolset for the basis of research, the rate of innovation can be accelerated
- **Enable faster, reliable model inter-comparisons and validations** Models integrated into the PV_LIB Package can be easily compared and validated in a reproducible way.
- **Increase the quality of communication between developers, Independent Engineers (IEs), and financiers**

By providing a common modeling platform for advanced data analysis, the communication of complex data analysis can be simplified and shared in a reproducible way.
- **Provide tools to efficiently handle large datasets** The analysis of data on hourly or minutely timescales can provide valuable insights into system performance, and spreadsheet programs are generally not able to handle the volume of data associated with these datasets. Using efficient code and the ability to parallelize some operations means that high resolution data can be analyzed.
- **Collaboration can increase the pace of technology innovation in the sector** Because the future expansion of PV_LIB will be based on the contributions of the PV community, new analysis methodologies and best practices can be quickly adopted by the sector.

III. The PV_LIB Package

The PV_LIB software package currently exists as a set of Python and MATLAB scripts which perform system performance calculations, and represent a large portion of currently published system performance and atmospheric functions.

These algorithms can be used in isolation to investigate a certain portion of PV performance evaluation, or can be used in combination to fully simulate a PV array, and are summarized in Table I.

A. Python PV_LIB

The Python programming language is a free and open source programming environment which has been shown to be very well suited to scientific computations [2], [3], [4]. A key feature of Python is it's flexibility to handle computation from small scale to large highly parallelized high performance computing operations [5] and because of this is able to scale easily, with no costs, to a users demands and requirements.

In addition, the Python language is supported by a vibrant community of developers, and new features are actively added and maintained. Because of this, integration with web, database and graphically intensive processes are relatively simple and intuitive in the system, allowing for a greater flexibility when developing models and algorithms. [6]

Python is also designed to be easily written and interpreted, and because of the high-level nature of the code is easily learned and understood by those with a basic understanding of programming syntax.

As PV_LIB develops into the Python programming language, it will feature three main principles:

- Take advantage of the Python programming language, to ensure free access to academic and commercial users
- Designed for collaborative development, and backed by a rigorous method to include the contributions of authors and researchers into the package
- 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.

IV. Collaborative software development

The Python version of PV_LIB will be designed and maintained to allow for collaborative development of the software using the open source model. [7], [8]

The full set of code is maintained on the Git Hub collaborative revision control system and it is possible for those interested in using or modifying the code base to create a 'fork' of the code onto their local systems. The users can then use or modify the code on their local machines. If a bug is fixed or a new function is developed, a request can be made through the GitHub system to integrate it into the main code base.

Requests to merge a change into the code base are then reviewed by the maintainers of the code, and if accepted can be integrated and merged into the code of those utilizing the code. At specified milestones, official version updates will be released which will integrate the latest updates into a down loadable package.

A. User demographics

There is a distinction made between two categories of expected users of the PV_LIB package: **Users** and **Developers**

Users Users are concerned with using high level modules to develop analysis workflows. They would take advantage of pre-defined modules, but wouldn't generally dive into their back end function or development. It is assumed that they will generally interact with PV_LIB through the 'Ipython notebook' environment, shown in Figure 1

Developers Though developers would be interested in the end result of developing new analysis workflows, they would also be concerned with the back end development of analysis modules. This would include the development of algorithms and the creation of algorithmic and physical testing suites.

B. Testing requirements

Algorithms integrated into the PV_LIB Python package will require **algorithmic** and **physical** testing packages. The specific requirements for these are published on the PV_LIB website.

Algorithmic testing packages ensure that the module will operate properly under all reasonable conditions imposed on it by users , including logical error handling.

Physical physical testing should include a validation dataset which demonstrates the implied function of the algorithm. This provides users with an accessible method of ensuring the accuracy of results and ensuring the physical validity of their models. Validation datasets must be included directly in the testing file, and should not rely on external datasets to ensure stability of the validation over time.

In order to ensure repeatable results, a version tracking system is also included in PV_LIB which will provide a standard method to document the algorithms and versions which were used in a particular simulation.

TABLE I
FUNCTIONS INCLUDED IN VERSION 1.1 OF PV_LIB . ALL REFERENCES ARE INCLUDED IN THE ON LINE PV_LIB DOCUMENTATION

Irradiance and atmospheric functions	
`pvlib.pvl_alt2pres`(altitude)	Determine site pressure from altitude
`pvlib.pvl_pres2alt`(pressure)	Determine altitude from site pressure
`pvlib.pvl_getaoi`(SurfTilt, SurfAz, SunZen, SunAz)	Determine angle of incidence from surface tilt/azimuth and apparent sun zenith/azimuth
`pvlib.pvl_disc`(GHI, SunZen, Time[, pressure])	Estimate Direct Normal Irradiance from Global Horizontal Irradiance using the DISC model
`pvlib.pvl_ephemeris`(Time, Location[, ...])	Calculates the position of the sun given time, location, and optionally pressure and temperature
`pvlib.pvl_spa`(Time, Location)	Calculate the solar position using the PySolar package
`pvlib.pvl_extraradiation`(doy)	Determine extraterrestrial radiation from day of year
`pvlib.pvl_globalinplane`(SurfTilt, SurfAz, ...)	Determine the three components on in-plane irradiance
`pvlib.pvl_grounddiffuse`(SurfTilt, GHI, Albedo)	Estimate diffuse irradiance from ground reflections given irradiance, albedo, and surface tilt
`pvlib.pvl_makelocationstruct`(latitude, ...)	Create a structure to define a site location
`pvlib.pvl_relativeairmass`(z[, model])	Gives the relative (not pressure-corrected) airmass
`pvlib.pvl_absoluteairmass`(AMrelative, Pressure)	Determine absolute (pressure corrected) airmass from relative airmass and pressure
`pvlib.pvl_clearsky_ineichen`(Time, Location)	Determine clear sky GHI, DNI, and DHI from Ineichen/Perez model
`pvlib.pvl_clearsky_haurwitz`(ApparentZenith)	Determine clear sky GHI from Haurwitz model

Irradiance Translation Functions	
`pvlib.pvl_perez`(SurfTilt, SurfAz, DHI, DNI, ...)	Determine diffuse irradiance from the sky on a tilted surface using one of the Perez models
`pvlib.pvl_haydavies1980`(SurfTilt, SurfAz, ...)	Determine diffuse irradiance from the sky on a tilted surface using Hay & Davies' 1980 model
`pvlib.pvl_isotropicsky`(SurfTilt, DHI)	Determine diffuse irradiance from the sky on a tilted surface using isotropic sky model
`pvlib.pvl_kingdiffuse`(SurfTilt, DHI, GHI, SunZen)	Determine diffuse irradiance from the sky on a tilted surface using the King model
`pvlib.pvl_klucher1979`(SurfTilt, SurfAz, DHI, ...)	Determine diffuse irradiance from the sky on a tilted surface using Klucher's 1979 model
`pvlib.pvl_reindl1990`(SurfTilt, SurfAz, DHI, ...)	Determine diffuse irradiance from the sky on a tilted surface using Reindl's 1990 model

Data Handling	
`pvlib.pvl_readtmy2`(FileName)	Read a TMY2 file in to a DataFrame
`pvlib.pvl_readtmy3`(FileName)	Read a TMY3 file in to a DataFrame

System Modeling functions	
`pvlib.pvl_physicaliam`(K, L, n, theta)	Determine the incidence angle modifier using refractive models
`pvlib.pvl_ashraeiam`(b, theta)	Determine the incidence angle modifier using the ASHRAE transmission model
`pvlib.pvl_calcparams_desoto`(S, Tcell, ...[, ...])	Applies the temperature and irradiance corrections to inputs for pvl_singlediode
`pvlib.pvl_retreiveSAM`(name[, FileLoc])	Retrieve latest module and inverter info from SAM website
`pvlib.pvl_sapm`(Module, Eb, Ediff, Tcell, AM, AOI)	Performs Sandia PV Array Performance Model to get 5 points on IV curve given SAPM module parameters, Ee, andcell temperature
`pvlib.pvl_sapmcelltemp`(E, Wspd, Tamb[, modelt])	Estimate cell temperature from irradiance, wind speed, ambient temperature, and module parameters (SAPM)
`pvlib.pvl_singlediode`(Module, IL, I0, Rs, ...)	Solve the single-diode model to obtain a photovoltaic IV curve
`pvlib.pvl_snlinverter`(Inverter, Vmp, Pmp)	Converts DC power and voltage to AC power using Sandia's Grid-Connected PV Inverter model
`pvlib.pvl_systemdef`(TMYmeta, SurfTilt, ...)	Generates a dictionary of system parameters used throughout a simulation

C. Code examples

Developers wising to contribute to the PV_LIB package will be require to adhere to PV_LIB coding standards for documentation and function creation. Docstrings (function descriptive headers) should be included for all functions and formatted according to the current google python docstring standard, and functions should adhere to the google python style guide where possible. An example of the basic layout of a function and it's test function is shown below. Note that the data included in a physical function test must be included in the same file as the test script to ensure stability of the testing package over time.

Function Code

```
'''
Attribution block
Developed by: name, organization
Date:
Revision 1: name, organization
Date:
'''
import modules
def pvl_Fcn(Inputs,**kwargs):
    '''
    Function Description
    Parameters
    ----------
    Input name: format
        Input Description
    Returns
    -------
    Output name: format
        Output Description
    '''
    Vars=locals()
    Expect={'Input1':('num','x>0')
              #Numerical logical contsraints
           'Input2':('str',('option1','option2')
              #String dependancy constraints
           'Inputopt':('Optional')}
```

```
                        #Kwargs input
        var=pvlib.pvl_tools.Parse(Vars,Expect)

        function_body_producing_output

        return Output
```

Testing Code

```
'''
Attribution block
Developed by: name, organization
        Date:
Revision 1: name, organization
      Date:
'''
from nose.tools import *
from .. import pvl_fcn
import modules
def test_proper_algorithmicl():
    #Run the function with multipe of inputs
    #to ensure it handles edge cases properly
    pvl_fcn(Inputs)
    assert(algorithmic test passes)

def test_proper_physical():
    #inputs representing a physical test case, can be
    #numpy array or DataFrame
    Inputs=np.array([Physcial data])
    #physically correct outputs, can be numpy ar
    #DataFrame
    output=np.array([Physcial data])
    #run the function
    output_algorithm=pvl_fcn(Inputs)
    #check it's validity
    assert(output_algorithm==output)
def main():
    unittest.main()
if __name__ == '__main__':
    main()
```

D. Licenses

PV_LIB is licensed under the BSD 3 clause which is quoted in part below:

> *Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:*
>
> *1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.*
>
> *2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.*
>
> *3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.*

The intention is to enable users to integrate the code into other projects or programs while ensuring that proper traceability and attribution of the code is maintained.

V. Using PV_LIB for Python

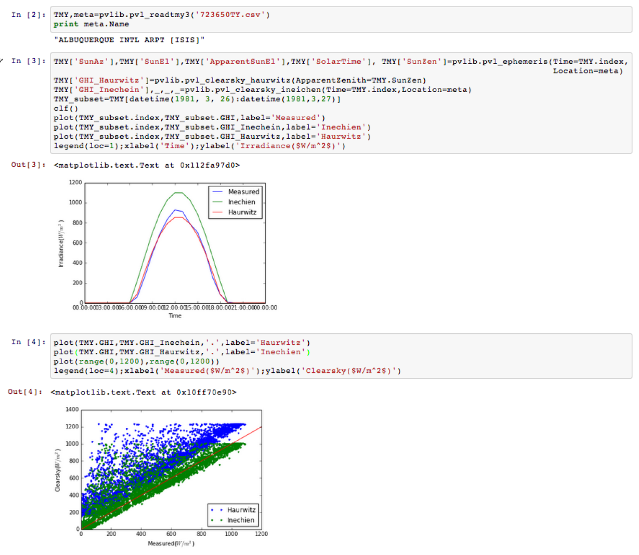
PV_LIB is created in the Python programming language, and can used alongside a powerful suite of Python modules to enable powerful options for data analysis including:

Numpy Numerical and statistical analysis tool-set

Scipy Scientific computing tool-sets including optimization and advanced regression

Pandas Time series analysis tool-set, including timezone and daylight savings handling, and time series statistical operations

The predominant method used by users to interact with PV_LIB is through the IPython Notebook [3] which is shown in Figure 1, and allows users to create shareable workflows like the one demonstrated below. These workflows can contain all processing steps from data input, to modeling, to statistical analysis. In addition, these workbooks can be shared as individual program files, or can be converted to HTML and easily shared as a static web-page.



```
In [2]: TMY,meta=pvlib.pvl_readtmy3('723650TY.csv')
        print meta.Name

        "ALBUQUERQUE INTL ARPT [ISIS]"

In [3]: TMY['SunAz'],TMY['SunEl'],TMY['ApparentSunEl'],TMY['SolarTime'], TMY['SunZen']=pvlib.pvl_ephemeris(Time=TMY.index,
                                                                                                                Location=meta)
        TMY['GHI_Haurwitz']=pvlib.pvl_clearsky_haurwitz(ApparentZenith=TMY.SunZen)
        TMY['GHI_Inechein'],_,_,_=pvlib.pvl_clearsky_ineichen(Time=TMY.index,Location=meta)
        TMY_subset=TMY[datetime(1981, 3, 26):datetime(1981,3,27)]
        clf()
        plot(TMY_subset.index,TMY_subset.GHI,label='Measured')
        plot(TMY_subset.index,TMY_subset.GHI_Inechein,label='Inechien')
        plot(TMY_subset.index,TMY_subset.GHI_Haurwitz,label='Haurwitz')
        legend(loc=1);xlabel('Time');ylabel('Irradiance($W/m^2$)')

Out[3]: <matplotlib.text.Text at 0x112fa97d0>

In [4]: plot(TMY.GHI,TMY.GHI_Inechein,'.',label='Haurwitz')
        plot(TMY.GHI,TMY.GHI_Haurwitz,'.',label='Inechien')
        plot(range(0,1200),range(0,1200))
        legend(loc=4);xlabel('Measured($W/m^2$)');ylabel('Clearsky($W/m^2$)')

Out[4]: <matplotlib.text.Text at 0x10ff70e90>
```

Fig: 1. Example of PV_LIB tools being used in IPython to perform analysis. This analysis shows a comparison of TMY GHI in Golden, Colorado to the clearsky GHI calculated from two models .

VI. Conclusion

This paper introduces the PV_LIB for Python software package developed by Sandia National Laboratories. The intention of this open source package is to accelerate the pace of innovation in the PV sector through providing a powerful, validated set of tools for the modeling of PV system performance. The intention of this package is to develop along with the industry and to serve as a common repository for new and developing techniques prior to their inclusion in commercial integrated modeling software, and to enable users to gain deeper insight into their data analysis.

Acknowledgements

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

References

[1] J. S. Stein, "The Photovoltaic Performance Modeling Collaborative (PVPMC)," *Photovoltaic Specialists Conference (PVSC)*, 2012.

[2] K. Hinsen, "High-Level Scientific Programming with Python," *Computational Science ICCS*, pp. 691–700, 2002.

[3] F. Pérez and B. E. Granger, "IPython: a system for interactive scientific computing," *Computing in Science & Engineering*, 2007.

[4] 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.

[5] X. Cai, H. P. Langtangen, and H. Moe, "On the performance of the Python programming language for serial and parallel scientific computations," *Scientific Programming*, 2005.

[6] K. J. Millman and M. Aivazis, "Python for Scientists and Engineers," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 9–12, 2011.

[7] B. Kogut and A. Metiu, "OpenSource Software Development and Distributed Innovation," *Oxford Review of Economic Policy*, vol. 17, no. 2, pp. 248–264, Jun. 2001.

[8] K. R. Lakhani and E. Von Hippel, "How open source software works:"free" user-to-user assistance," *Research policy*, 2003.