Pypvcell: An Open-Source Solar Cell Modeling Library in Python

Kan-Hua Lee, Kenji Araki, Omar Elleuch, Nobuaki Kojima and Masafumi Yamaguchi

***Abstract*—We announced a open source solar cell modeling and analysis toolkit written in Python. The standard off-the-shelf solar cell simulation software is often difficult to modify or reuse some of its functionality into new solar cell models. In this software package, we wrap the solar cell simulations into individual modules and application programming interfaces to make them very user-friendly. This library contains a wide range of functions to do the heavy-lifting, error-prone jobs of modeling solar cells, such as unit conversions and arithmetic operations of spectrum data, absorption-emission reciprocity or solving the I-Vs of multi-junction cells. This allows the users to rapidly adapt and build their own models with these modularized and verified components. Source codes, detailed documentation and examples can be found in https://kanhua.github.io/pypvcell.**

***Index Terms*—solar cells, simulation**

I. Introduction

Modeling solar cells is an important part in designing and optimizing solar cell structure. The most common option is using off-the-shelf solutions to model the solar cells, such as PC1D and its descendants [1] [2] [3]. Their graphical user interface lowers the slope of learning curve, however, it is difficult to automate large amount of iterative simulation processes or tailor the model to suit their research purposes. Researchers therefore choose to develop their own codes from scratch. However, this slows down the development of solar cell modeling because significant amount efforts were spent on reinventing the wheels that may have been built by others. Therefore, in our opinion, an ideal solar cell modeling software package for advanced users should meet the following requirements:

1) It is formed by a rich set of carefully designed application programming interfaces (APIs), so the users can easily plug any parts of the functionalities of the software to their own model.
2) The code is well-documented and open-sourced, allowing everyone to validate and adapt the codes for their own research
3) Each functions are validated and tested comprehensively.

Pypvcell was therefore designed and implemented to meet these requirements. Our goal is to define a standard of APIs so that everyone can build their own solar cell model rapidly and share their models under a common framework.

Pypvcell is written in Python [4], a general-purpose programming language that recently become very popular in scientific computing. Implementing the solar cell modeling library in Python also makes it easy to streamline the energy yield prediction with PVLIB-Python [5], a renowned Python softwarepackage for predicting the energy yields.

This paper will present an overview of the design concept, followed by the progress and the validation of Pypvcell. We would like to emphasize that although these design concepts are implemented in python in this work, these can also be implemented in other modern programming languages that supports object-oriented programming.

II. Design of this package

A. Architecture of Pypvcell

The design architecture of Pypvcell is illustrated in Fig. 1. Pypvcell focuses on the simulation of the solar cell. Most of the low-level numerical computation such as matrix operations and interpolation are rely on Numpy [6] and Scipy [7], which are the definitive packages for numerical calculation in Python. A rich set of functions that performs the operations related to solar cells, including optics, photogeneration and carrier recombination, were built upon Numpy and Scipy. These function are further encapsulated into different solar cell models. In Pypvcell, every solar cell model is defined as a class. For example, the following codes initializes an instance of solar cell object that implements the radiative-limit model:

```
gaas=SQCell(eg=1.42,cell_T=292)
```

The above code initialize a radiative-limit cell with a band gap of 1.42 eV. The solar cell models that are currently implemented in Pypvcell are listed in TABLE I.

These solar cell classes are all inherited from the same base class and share the same interfaces of retrieving the electrical and optical output. These standardized interface let the switching between different the solar cell model very easy.

Pypvcell currently simulate two-terminal series connected multi-junction solar cell by solving the I-V of series connected individual subcells. One can establish this multi-junction cell by putting different solar cell instances in a pipeline. This approach can accommodate subcells with different models. For instance, a multi-junction solar cell can be set up like this:

```
gaas=SQCell(eg=1.42,cell_T=292)
si=DBCell(qe,rad_eta)
mj=MJCell([gaas,si])
```

In this example, the multi-junction cell has a GaAs top cell at Shockley-Queisser limit, and a silicon bottom cell with user-defined quantum efficiency and radiative efficiencies. Users can easily write the model for their own solar cell and incorporate it into the multi-junction cell as long as they use the same API interface to name the functions.

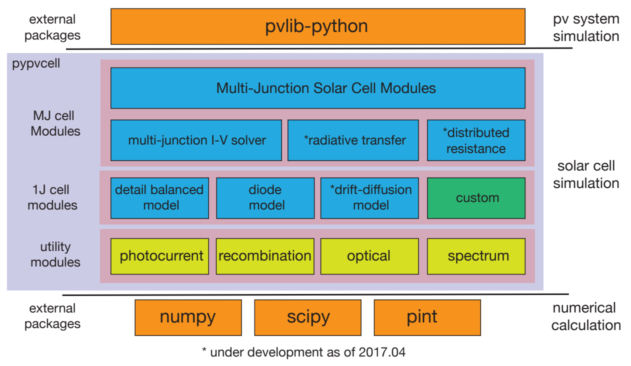

Fig. 1. The architecture of Pypvcell.

B. Data Structure

In general, Pypvcell uses `ndarray` defined in Numpy as the main data structure in the calculation. `ndarray` is the standard data structure for storing the values of vector and matrix. Internally, the physical quantities are converted to SI units for calculation. In the current version, the user has to explicitly convert the physical quantities to SI units when inputting the values to the APIs. The capability of handling physical quantities with dimensions will be supported in future versions.

Modeling solar cells involves a lot of arithmetic operations, interpolation and unit conversions of spectrum data, such as solar spectrum, quantum efficiency, absorption coefficients and so on. These operations can be very troublesome and error prone. We therefore implemented a class called `Spectrum` to do these operations. An instance of `Spectrum` class stores two arrays, the abscissa and the ordinate of a spectrum, which are denoted as `x` and `y` in later discussion, respectively. Despite that it may seem inconvenient to bundle the `x` and `y` data in a class instance, it can save tremendous efforts of debugging especially when doing unit conversions, since the values of `y` are often coupled with `x`. Also, we have created many functions in `Spectrum` to reduce these inconveniences and make the syntax as intuitive as possible.

An instance of the class can be initialized by the following code:

```
sp=Spectrum(x_data=wavelength,
            y_data=spec,x_unit='nm')
```

Arithmetic operations in `Spectrum` is very intuitive, for example,

```
# multiply every element of y by 0.8
new_sp=sp*0.8

# sum up the spectrum of sp1 and sp2
sp_sum=sp1+sp2
```

In the above code snippet, if `x` in `sp1` and `sp2` do no match, `Spectrum` class will first interpolates `sp2` and perform then perform addition on `sp1`.

The value of `spectrum` is retrieved by `get_spectrum` method for plotting or exporting. When retrieving the values of `x` and `y`, the user has to specify the units of `x`. The user can also elect to convert the spectrum from energy to photon flux by specifying the parameter `to_photon_flux`:

```
x1,y1=sp.get_spectrum(to_x_unit='eV',
                      to_photon_flux=True)
```

`Spectrum` class can also deal with `y` that comes with the unit of per wavelength or per photon energy, such as $\mathrm{W/m^2}$. In other words, the integral of the spectrum $\int y(x)dx$ would give a legitimate physical quantity, such as total power.

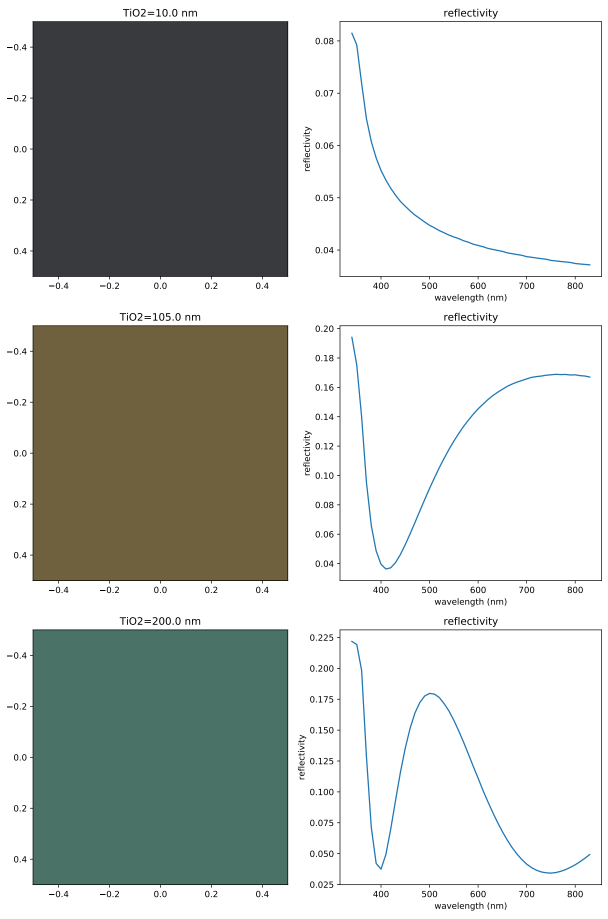

Fig. 2. The modelled color appearence of the surface of glass substrates with different thicknesses of TiO2 coated on the top. The left column shows the colors and the right column is the reflectivity.

III. Optical Model

Pypvcell implements a transfer matrix model to calculate the transmission, reflection and absorption described in [8] and [9]. As an example, Fig. 2 models the reflectivity of a glass substrate with different thicknesses of TiO2 along with it color appearances. In this example, pypvcell calculated the reflected spectrum of the TiO2-coated glass substrate and then feed the reflected spectrum into python-colormath to calculate the color. This example demonstrates the flexibility and extendability of using a script-based solar cell model package instead of a graphic-user-interface-based one.

TABLE I
SOLAR CELL MODELS IMPLEMENTED IN PYPVCELL

Model	Description
`SQCell`	Solar cell at Shockley-Queisser limit.
`DBCell`	Generalized detailed balance model with arbitrary absorptivity and external radiative efficiency.
`MJCell`	Multi-junction solar cell by solving I-V of individual cell.
`TwoDiodeCell`	Two-diode model with specified J_{01}, J_{02}, n_1, n_2 ,R_s and R_{sh}
`AnaPNCell`	Solar cell model with analytical solver of drift-diffusion equations (in progress).
`NumPNCell`	Solar cell model with numerical drift-diffusion equation solver (in progress).

IV. SOLAR CELL MODELS AND VALIDATIONS

In this section we briefly describe underlying algorithms of several the models that are implemented in Pypvcell, which is then followed by some validations by comparing Pypvcell's results to the value reported in the literature.

A. Generalized detailed balance model

Pypvcell has a detailed balance model for modeling single junction solar cells [10] [11]. This model is widely used for predicting the efficiency limits of solar cells. Implementations of this model can be found on any code repository, such as EtaOpt [12]. Most of these packages can only deal with bulk material with stepped absorption. Our implementation generalized this model for arbitrary absorptivity or quantum efficiency, making it easy to project the efficiency limit of solar cell with indirect band gaps or nanostructure. Mathematical details of the model can be found elsewhere, such as [12].

B. Series-connected-diode multi-junction model

A series-connected-diode model is currently implemented in Pypvcell. In this model, the I-V characteristics of each subcell is first calculated individually. After that, an I-V solver is launched to find the I-V characteristics of the multi-junction cell from the I-V characteristics of the individual subcells. The details of these two steps are described as below.

The illumination of the i-th junction is assumed to be the sum of the transmitted photons from its upward junction and other contribution, including the reflection from its downward junction and the radiative emission from its upward junction, which can be formulated by:

$$\phi_i(E) = \phi_{i-1}(E)(1 - \mathrm{EQE}_{i-1}(E)) + \phi'_i(E) \tag{1}$$

where $\phi_{i-1}(E)$ and $\mathrm{EQE}_{i-1}(E)$ are the incident photons and the EQE of the junction stacked above the i-th junction. $\phi'_i(E)$ accounts for those photons contributions from the reflection of $i+1$-th junction or the radiative emission from $i-1$-th junction. Note that this assumption only applies to thick subcells. For thin subcells, more accurate model such transfer matrix model should be used to calculate the illumination of each subcell properly. Once the illumination is determined, the I-V characteristics of i-th subcell $V_i(J)$ can then be calculated by any single-junction solar cell model provided in Pypvcell or a custom model.

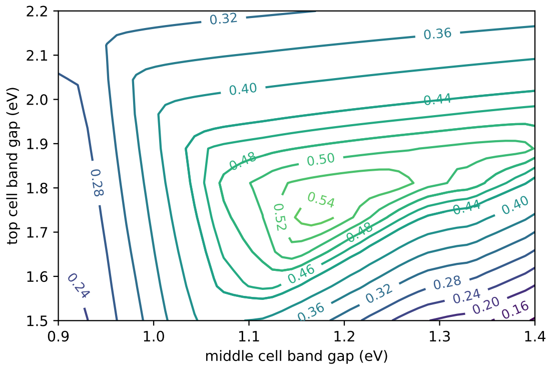

Fig. 3. Radiative-limit efficiency contour of 3J $Eg_x/Eg_y/0.67$eV against Eg_x and Eg_y. This figure matches Figure 1 of [13].

Pypvcell provides two different modes to calculate the efficiency of the multi-junction cells. One is assuming a mechanical stack cell, in which case the efficiency is simply calculated by adding up the maximum power of the individual subcell. Another one is the two-terminal, series-connected multi-junction cell. To solve this numerically, a suitable range of J is discretized to get a series values of current, that is,

$$\{j_m\}, m = 0...N-1 \tag{2}$$

The I-V of the multi-junction device is thus solved by interpolating the voltages for discretized current density j_m for every subcell i and adding up the subcell voltages to obtain the I-V characteristics of the multi-junction cell $V_{tot}(j_m)$, namely,

$$V_{tot}(j_m) = \sum_{i=1}^{N} V_i(j_m) \tag{3}$$

where $V_i(J)$ is the voltage of the i-th subcell at the current density J.

C. Validation of the models

Our results of radiative-limit solar cells show reasonable agreements with others' work. We caluclated the efficiency contour of three-junction cell X/Y/Ge(0.67eV) cell against the band gap of subcell X and Y, as shown in Fig. 3. This calculation assumes all the cells at Shockley-Queisser limits. This figure also match nicely with the same calculation presented in [13]. The results shows that the detailed balance model and the I-V solver implemented in Pypvcell is robust and reliable. Other validations and test of the functions and components of Pypvcell can also be found in the package.

V. CONCLUSION AND FUTURE DEVELOPMENT

We presented the design concepts, data structure and implementation details of a solar cell modeling library in Python. The design of Pypvcell allows great extensibility for incorporating new or user-defined solar cell models. A number of solar cell models were implemented in Pypvcell, featuring a generalized detailed balance model and a robust multi-junction I-V solver. The development of more sophisticated model such as drift-diffusion model is currently underway.

Acknowledgment

The authors would like to thank Japan New Energy and Industrial Technology Development Organization (NEDO) for supporting this research (NEDO 15100731-0).

References

[1] D. A. Clugston and P. A. Basore, "PC1D version 5: 32-bit solar cell modeling on personal computers," in *Photovoltaic Specialists Conference, 1997., Conference Record of the Twenty-Sixth IEEE*. IEEE, 1997, pp. 207–210.

[2] H. Haug, J. Greulich, A. Kimmerle, and E. S. Marstein, "PC1Dmod 6.1 – state-of-the-art models in a well-known interface for improved simulation of Si solar cells," *Solar Energy Materials and Solar Cells*, vol. 142, no. C, pp. 47–53, Nov. 2015.

[3] H. Haug, A. Kimmerle, J. Greulich, A. Wolf, and E. S. Marstein, "Implementation of Fermi–Dirac statistics and advanced models in PC1D for precise simulations of silicon solar cells," *Solar Energy Materials and Solar Cells*, vol. 131, no. C, pp. 30–36, Dec. 2014.

[4] "Official Python Website."

[5] W. F. Holmgren, R. W. Andrews, A. T. Lorenzo, and J. S. Stein, "PVLIB Python 2015," in *2015 IEEE 42nd Photovoltaic Specialists Conference (PVSC)*. IEEE, 2015, pp. 1–5.

[6] [Online]. Available: http://www.numpy.org

[7] [Online]. Available: http://www.scipy.org

[8] L. A. A. Pettersson, L. S. Roman, and O. Inganäs, "Modeling photocurrent action spectra of photovoltaic devices based on organic thin films," *Journal of Applied Physics*, vol. 86, no. 1, pp. 487–496, Jul. 1999.

[9] P. Peumans, A. Yakimov, and S. R. Forrest, "Small molecular weight organic thin-film photodetectors and solar cells," *Journal of Applied Physics*, vol. 93, no. 7, pp. 3693–3723, Apr. 2003.

[10] G. L. Araújo and A. Martí, "Absolute limiting efficiencies for photovoltaic energy conversion," *Solar Energy Materials and Solar Cells*, vol. 33, no. 2, pp. 213–240, 1994.

[11] W. Shockley and H. J. Queisser, "Detailed Balance Limit of Efficiency of p-n Junction Solar Cells," *Journal of Applied Physics*, vol. 32, no. 3, pp. 510–519, 1961.

[12] G. Létay and A. Bett, "EtaOpt–a program for calculating limiting efficiency and optimum bandgap structure for multi-bandgap solar cells and TPV cells," in *The 17th EC-PVSEC*, 2001.

[13] R. R. King, D. C. Law, K. M. Edmondson, C. M. Fetzer, G. S. Kinsey, H. Yoon, R. A. Sherif, and N. H. Karam, "40GaInP/GaInAs/Ge multijunction solar cells," *Applied Physics Letters*, vol. 90, no. 18, pp. 183 516–3, 2007.