# Comprehensive Library of Photovoltaic Functions on Python for Academic and Educational Purposes

by

Pedro Reguera Rodriguez

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2018 by the
Graduate Supervisory Committee:

Chistiana Honsberg, Chair
Stuart Bowden
Richard King

ARIZONA STATE UNIVERSITY

May 2018

ABSTRACT

This comprehensive library of photovoltaic functions (PVSimLib) is an attempt to help the photovoltaics community to solve one of its long-lasting problems, the lack of a simple, flexible and comprehensive tool that can be used for photovoltaic calculations. The library contains a collection of useful functions and detailed examples that will show the user how to take advantage of the resources present in this library. The results will show how in combination with other Python libraries (Matplotlib), this library becomes a powerful tool for anyone involved in solar power.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

# INTRODUCTION

When it comes to the study of any science, modeling and simulations are inevitably necessary. The case of photovoltaics is no different, and most students and researchers of solar cells and modules wind up creating an excel sheet with thousands of rows and columns that does the necessary math or the equivalent. In the end, solar scientists and scholars often invest hundreds of hours to create some sort of automated calculator that satisfies their own needs but no one else's. The objective of PVSimLib is to provide the ultimate resource for anyone interested on solar power. It contains all the resources that any photovoltaics student needs to complete his/her courses, as well as serving as a baseline for any scholar planning to do research in this field.

As an example of the power of this tool, it will be shown that all the projects required on solar classes can be solved in a few tens of lines by calling different functions from this library. In case the student is not proficient in programing, the library includes several examples to help the student to figure out how to use the library. Yet, to extract the maximum potential of this library, the student is required to have basic knowledge of programming. Including how to write a loop, display a variable or import a library.

A library like PVSimLib will be in an ever-lasting developing phase as there will always be new effects to include, more complex functions to add or simply new functions coming from the latest advances in solar science. Therefore, PVSimLib serves as a baseline for any scholar researching on solar power. Although scholars will need to tweak some functions or improve them according to their particular needs.

## METHODOLOGY

To achieve the flexibility necessary to be useful for both scholars and students, PVSimLib has been designed as a collection of functions with several examples included. This structure will allow the user to apprehend with a quick look how to use the functions and cherry pick the functions that they need. Ideally, the user will write a script, imitating the examples provided.

### Collection of Functions

This collection has separated in 6 different modules: Common, Constants, Radiation, Semiconductors, Solar Cells and Systems. Files and Examples are folders. Examples contains all the examples enclosed in this library.

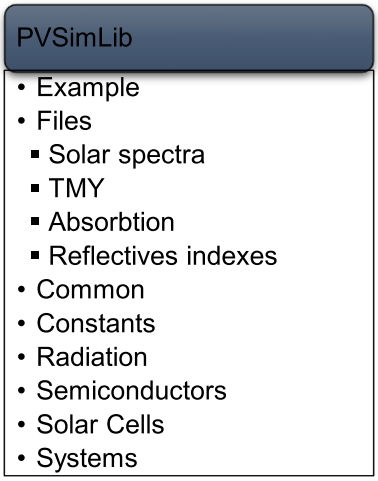


Figure 1: Structure of PVSimLib.

Each of these modules includes all the functions necessary to do the most frequent calculations and will be explained in further detail later. However, I want to bring attention to “Files”, which has not been mention earlier as it is not a Python module. It is actually just a folder that contains 3 types of files: Solar Spectrum files, absorption files and TMY files. If any user wants to add any of these types of files, it is mandatory to place them in the mentioned folder. Also, it is critical that the added files follow the same structure of its kind located there. The reason is that the file reading functions are really sensitive to changes in the structure and location of files. Therefore, a modification in the location or the structure of the file could end in a run-time error.

Some of the functions require as input, values that can be calculated from functions present in other modules. This has been done on purpose to allow maximum flexibility in the usage. This way, any function can be called using experimental results or using values previously calculated with other functions. Following the standard approach on computed science, more complex functions have encapsulated the more basic functions, making impossible using experimental values.

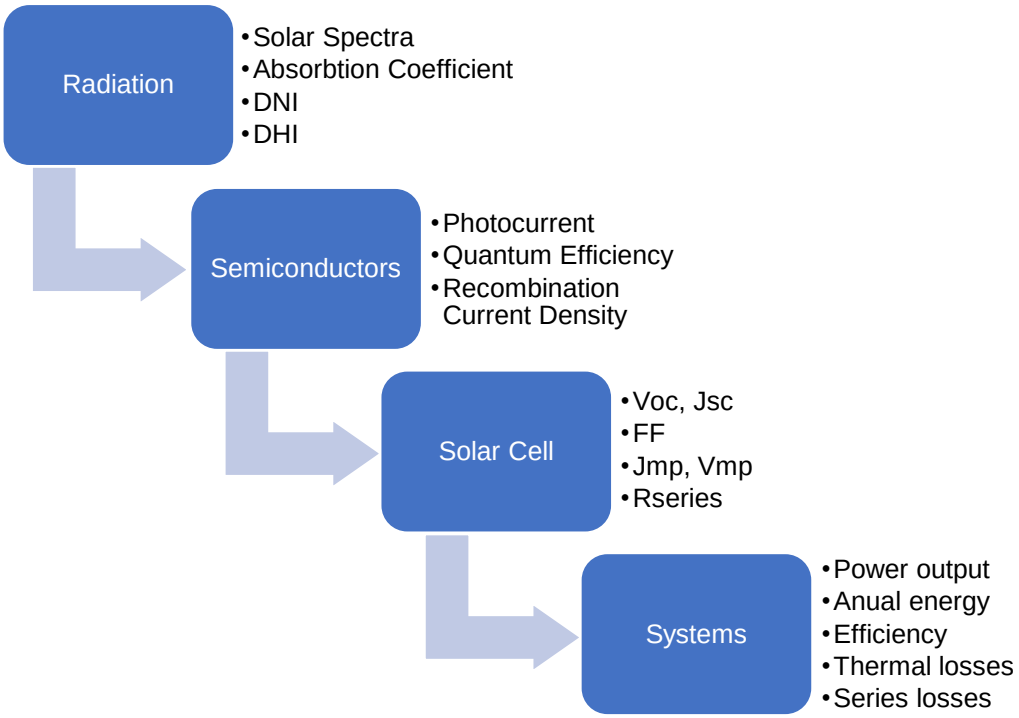


Figure 2. Example Flowchart of Possible Execution.

Also, it is worth noticing that some functions have been overloaded. One example of this overload are the functions that calculate intrinsic carrier concentration or the carrier's mobility. The reason behind the overload is that, often, students learn in class more simplistic equations that do not include effects that are considered when doing research. This approach will allow the students to use the more basic functions, that implements the equations that they might study in their classes, and the scholars to use the more complex and accurate functions.

Modules Description

The code has been placed in modules attending to the concept that they relate. Also, the name of the module has been chosen to relate to that same concept. The number of modules is certainly arbitrary, although it has been my intention to keep the structure simple, so anyone interested can easily navigate through the code.

- Common: Contains the function that helps with conversion of units or simple calculations like thermal voltage, sine or cosine that can be useful in any other module. As well as all the functions to read files.
- Constants: it is simply a group of useful constants put together.
- Radiation: Includes function to read, manipulate and make calculations on the solar spectra.
- Semiconductors: Contains all the functions related to semiconductor parameters, such as recombination, mobility, recombination current, lifetimes, quantum efficiency, etc.
- Solar Cells: Includes all the functions necessary to calculate solar cell parameters, such as open-circuit voltage, short-circuit current density, series resistance, maximum power point, etc.
- Systems: Allow to calculate efficiency and power output of solar cells and photovoltaics panels. It is possible to take into account series resistance losses, thermal losses and shading losses.

Dependency

In the diagram below it is represented the dependency relationship between the different modules within the library. In blue, the modules that belong to PVSimLib. In green, other Python libraries that are necessary for the well-functioning of this library. Those libraries (scipy and Numpy) need to be downloaded through pip or any other method in order to be able to use PVSimLib. Last, in orange, a Python built-in function that is needed as well. Built-in functions do not need to be downloaded, as they come with Python.

Understanding the dependency within the library is not useful for a user that is not interested in developing new content in the library, but it will become critical if the user intends to do so.

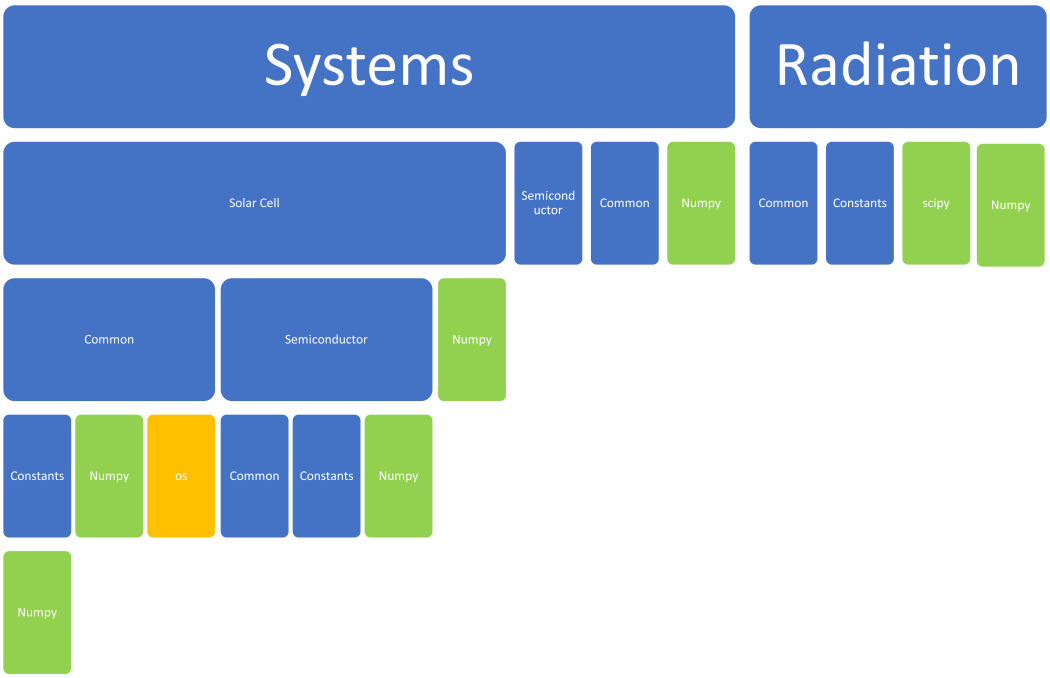


Figure 3: Dependency diagram

Dependency means that modules above import the modules or libraries right underneath them, as shown in the diagram above. It is extremely important that the dependency always happen in only one direction. If someone introduces a function in one the modules at the bottom (i.e. Solar Cell) and that function requires another function from a module in a level above (i.e. System), it will require an import and that import will necessarily create an issue known as “circular dependency”. This is not only a common mistake in programming, it could also force a run-time error. Most compilers would highlight this as a compilation error, so the developer has to solve it. However, Python is an interpreted language, therefore there is a real danger of the program crashing if this type of dependency is introduced.

Examples

The examples included in PVSimLib are meant to ease the learning curve necessary to be able to use this library. Any unexperienced user should be able to learn how to use the library by simply replicating what the examples show and substituting some functions or parameters for another.

Among the examples provided, there are implemented cases of use that are expected to be the most frequent. Therefore, allowing in many cases to simply execute the example with the desired input values.

1. Blackbody_plot.py: Plots the blackbody spectrum for a certain temperature between 200nm and 3000nm.
2. Plot_solar_reference_spectra.py: Plots the spectral irradiance of AM0, AM1.5G and AM1.5G between 200nm and 2000nm of the spectrum.
3. Sunlight_examples.py: Calculates and displays a few parameters related to the sun position depending on a certain location and module parameters.
4. Current_in_band.py: Calculates and plots the current density available in each band of 0.5nm from 200nm to 1400nm of the spectrum. The bands are not constant on the file provided.
5. Plot_declination.py: Plots the declination in degrees along the year according to different methods of calculation.
6. Plot_DLARC_with_slider.py: Plots the reflectivity between 2 dielectrics, based on their indexes and thickness. It can be modified on the fly and it updates the graph automatically.
7. Plot_mobility_resistivity.py: Plots the resistivity of a semiconductor as a function of doping.
8. Plot_silicon_mobility.py: Plots the mobility of electrons in silicon as a function of doping according to two different models available in the library (Masetti and Thurber).
9. Semiconductor_examples.py: Calculates and displays the band-gap narrowing effect based on doping and temperature.
10. Quantum_efficiency.py: Plots the quantum efficiency of a solar cell by regions.

11. Plot_ideal_cell.py: Calculates and plots the IV curve of an ideal solar cell.
12. Solar_cell_parameters.py: Calculates and displays all the base parameters (Jsc, Voc, J0, FF, Jmp, Vmp and Efficiency) of an ideal solar cell based on the detail balance.
13. Series_resistance.py: Calculates the series resistances and the power losses caused by this series resistance. The calculation is done by regions (base, emitter, fingers and busses).
14. Module_ser_temp.py: Calculates and displays the effect of series resistance and temperature on the yearly energy produced by a module and its efficiency.
15. One_cell_shaded.py: plots the IV curve of a series of solar cells with 1 shaded. Also, calculates and displays the yearly energy produced by the system and its efficiency.

## RESULTS

In this section, I will show some results that can be achieved by using a script that call the functions from PVSimLib to calculate and plot them using the library Matplotlib. PVSimLib does not include plotting functions as there are already several available to use on Python.

All the results shown in this section come from the execution of the script listed in the section “2. EXAMPLES” under the chapter “METHODOLOGY”. The input parameters used to obtain all the results in this section are shown in the figure 4.

```
# General Parameters
T = 300                 # Temperature (K)
E_Si = 11.7               # Relative electric permittivity of Silicon (No dimension)
W = 15.6                 # Solar cell width (cm)
Z = 15.6                 # Solar cell height (cm)

# Emitter parameters
We = 0.25e-4       # emitter thickness (cm)
Le = 3.87e-4       # emitter diffusion length (cm)
De = 3           # emitter diffusivity (cm2/s)
Se = 1000          # front surface recomb (cm/s)

# Base Parameters
Wb = 250e-4        # Base thickness (cm)
Lb = 158.11e-4     # Base diffusion length (cm)
Db = 5           # Base diffusivity (cm2/s)
Sb = 2000          # Back surface recomb (cm/s)

# Concentration parameters
Nd = 2e19          # Majority carrier concentration on N side
Na = 3e16          # Majority carrier concentration on P side

# Front contacts parameters
bus_resistivity = 1.6e-6    # Bus resistivity (Ohm*cm)
bus_width = 0.2           # Bus width (cm)
bus_height = 400e-4        # Bus height (cm)
bus_number = 3           # Number of buses

finger_resistivity = 4.8e-6              # Finger resistivity (Ohm*cm)
finger_spacing = 0.2                    # Finger spacing (cm)
finger_width = 120e-4                    # Finger width (cm)
finger_length = Z/(2*bus_number)          # Finger length (cm)
finger_height = 20e-4                   # Finger height (cm)
finger_number = round(W/finger_spacing)     # Number of fingers
```

Figure 4: Input parameters needed for results section.

In the example 4, it is shown the current density available in each band of 0.5nm from 200nm to 1400nm of the spectrum. This window has been chosen because is the band that is normally absorbed on solar cells. The result is show on figure 5.

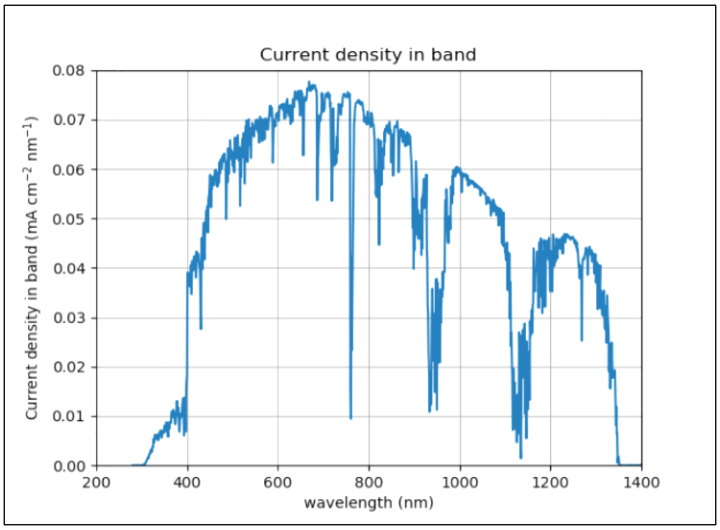


Figure 5. Current in band between 200nm and 1400nm.

From the execution of example 8, it is obtained a comparison between both models implemented to calculate electron mobility from doping (show on figure 6). It is interesting to see how both models predict almost identical mobilities until the doping is heavier than $10^{20}$ $cm^{-3}$. Once the doping crosses this threshold, both models differ in their predictions. Since doping rarely goes beyond this limit, it does not seem too relevant which model to choose. Unless the user is interested on studying limit cases.

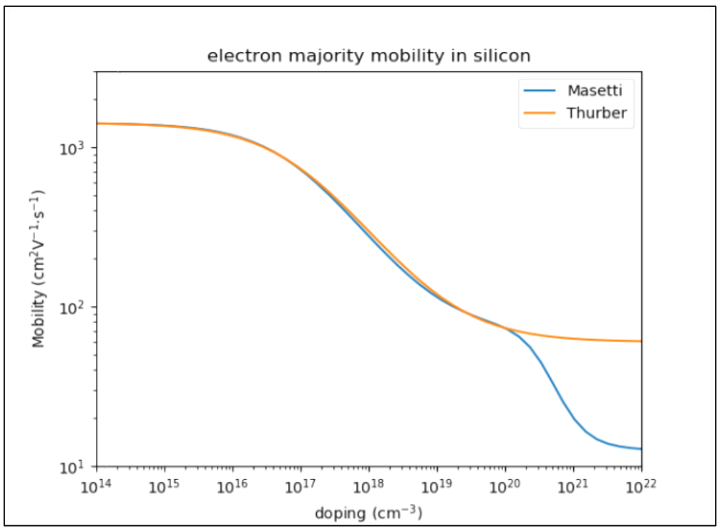


Figure 6: Comparison between Masetti and Thurber mobility functions.

After running example 10, two plots are obtained and shown on figure 7. The graph on the left has been generated using an absorption coefficient input file that has less points than the spectrum. This is corrected by an interpolation function included in PVSimLib that adapts one vector number of points to another that has been specified. The graph on the left is the result of using an absorption coefficient with the same accuracy as the spectrum file. It is easy to see that the interpolation function introduces a small ripple on the output. The error introduced by this interpolation is minimal.

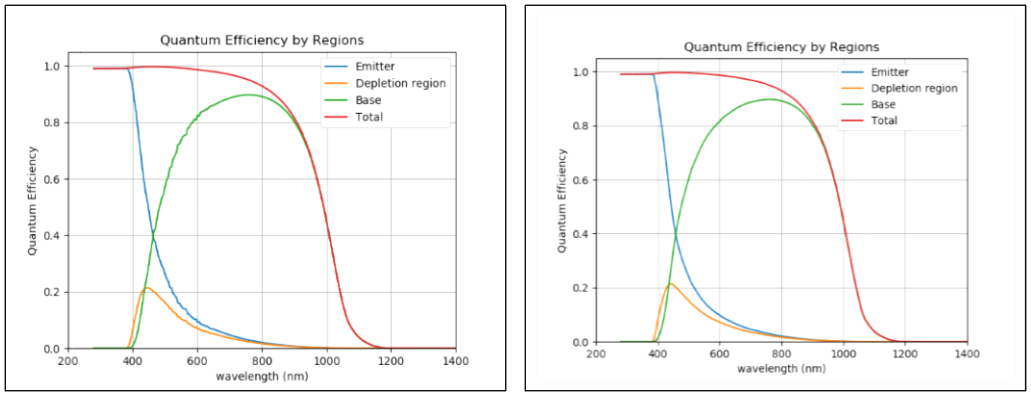


Figure 7: Comparison of quantum efficiency by regions.

In the figure 8, it is shown the result from the execution example 11. This is the ideal IV curve of a silicon solar cell. Parameters are shown at the beginning of this chapter.

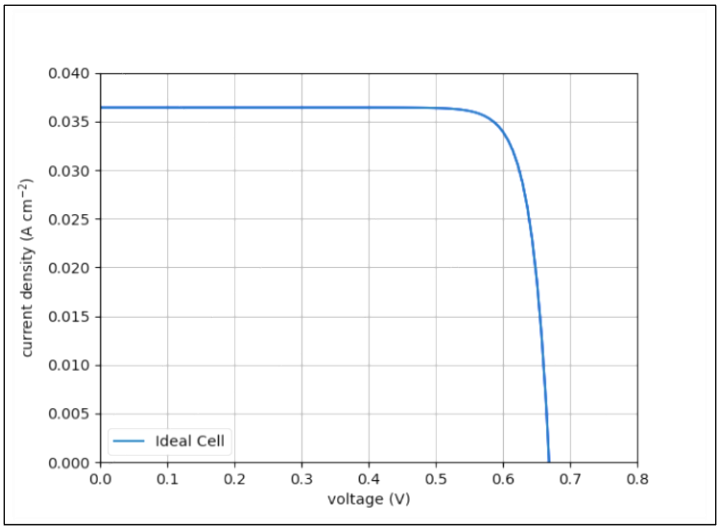


Figure 8: Ideal silicon solar cell IV curve.

The example 12 output is shown on figure 9. Each parameter shown is calculated from an individual function in PVSimLib. Jsc comes from the detail balance, J0 from the narrow base equation (including surface recombination), Voc from the diode equation, FF from the Voc approximation, MPP parameters from the fill factor and Voc and Jsc.

Built-in voltage: 0.94 V
Depletion region width: 0.20 um

Max photocurrent: 43.62 $\mathrm{mA/cm^2}$
J_SC: 36.43 $\mathrm{mA/cm^2}$
J_0(Narrow Base): 1.78e-10 $\mathrm{mA/cm^2}$

Voc: 0.67 V
FF: 0.84
Jmp: 33.41 $\mathrm{mA/cm^2}$
Vmp: 0.62 V
Efficiency: 20.64 %

Figure 9: Ideal solar cell parameters calculation results

Base resistivity: 0.54 Ωcm
Base Resistance: 0.01 $\Omega\mathrm{cm^2}$
Base Losses: 0.07 %

Emitter resistivity: 3.18e-03 Ωcm
Sheet resistance: 127.16 $\Omega/\square$
Emitter Resistance: 0.42 $\Omega\mathrm{cm^2}$
Emitter Losses: 2.28 %

Finger Resistance: 0.09 $\Omega\mathrm{cm^2}$
Finger Losses: 0.49 %

Bus Resistance: 0.08 $\Omega\mathrm{cm^2}$
Bus Losses: 0.45 %

transparency: 90.38 %

Rseries: 0.61 $\Omega\mathrm{cm^2}$

Figure 10: Series resistance calculations.

The example 13 output is shown on figure 10. The series resistances are calculated from the formulas of power losses and the dimensions, as well as the maximum power point parameters. The resistance is adjusted to area, since it is calculated from current density. This has been chosen this way because simplifies the calculations of losses in other modules.

The output of example 14, shown in figure 11, presents a comparison between the three different ways of calculating the yearly power output for the same solar module. This is done by the same function but introducing different set of parameters. The function has been overloaded to allow the user to call it with or without a vector of temperature. That feature allows to introduce the temperature effect or leave it with only the series resistance effect. Last, to avoid using any series resistance effect, a series resistance of 0Ω has been set as input.

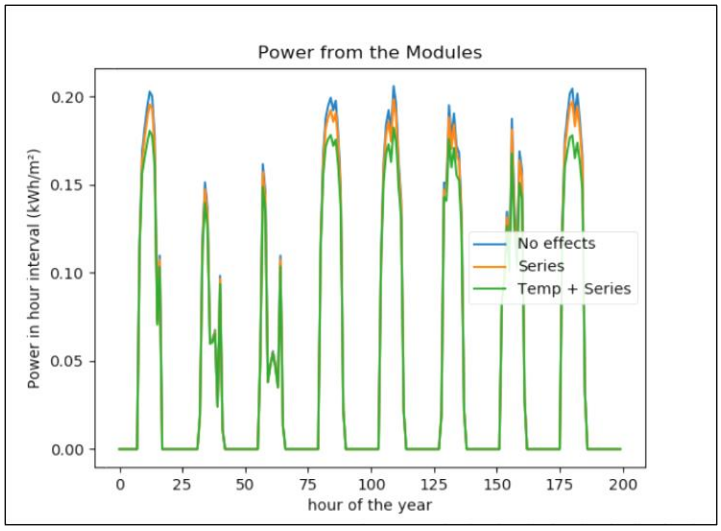


Figure 11: Comparison of the power of a module under different effects.

However, in the same example 14 there is another interesting result that shows more clearly the effect of temperature. In figure 12, it can be observed a change in the ripple of the efficiency calculated. The temperature vector mentioned and the incident power vector to the module comes from the TMY file selected.

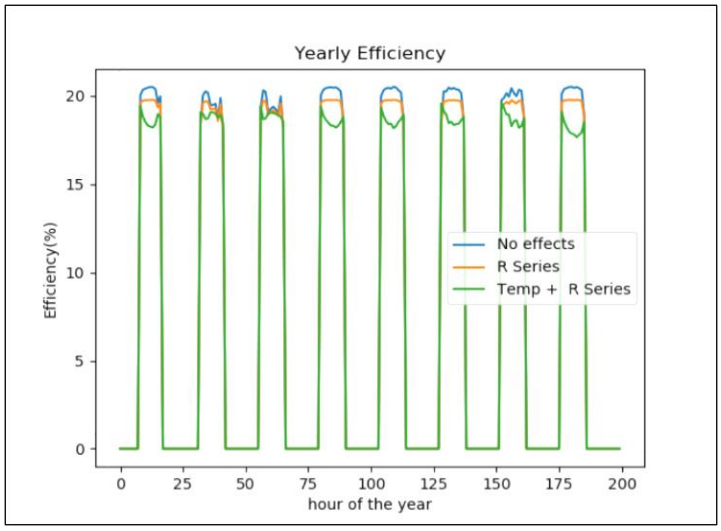


Figure 12: Comparison of the efficiency of a module under different effects.

Last, it is important to highlight that despite these calculations being done over the period of a year, only first 200 hours have been shown in the graphs to be able to show clearly the different effects involved.

Finally, example 15 has been coded to calculate IV curve for 9 solar cells in series with one of them shaded. Below, in figures 13 and 14, it is shown the output two different runs of that example 15.

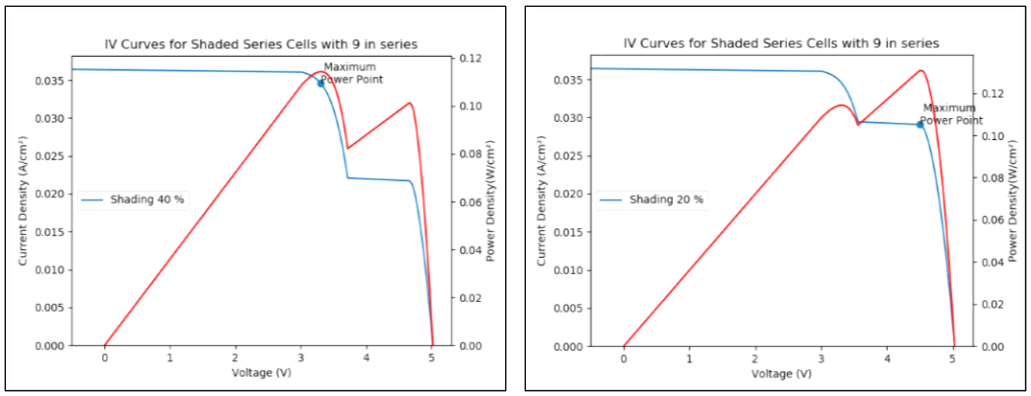


Figure 13: Comparison of IV curves of 9 solar cells in series with 1 shaded.

| | |
|---|---|
| Energy from Sun over year 3087.997 | Energy from Sun over year 3087.997 |
| Energy from Module 506.31 kWh/m² | Energy from Module 506.31 kWh/m² |
| Energy from Module with shading 390.70 kWh/m² | Energy from Module with shading 446.47 kWh/m² |
| Average Efficiency = 12.65% | Average Efficiency = 14.46% |

Figure 14: Efficiency calculations of 9 solar cells in series with 1 shaded at 40% and 20%.

It seems obvious that increasing the percentage of shading over one cell, will reduce the voltage on that cell and therefore the MPP, yearly energy and efficiency. However, this will not continue happening once the shading percentage crosses a certain threshold. This threshold is set by the capacity of the solar cell to stay in forward bias, when this becomes impossible, the solar cell goes into reverse bias. Once the solar cell is on reverse bias, the cell does not participate in the power generation process, therefore the output becomes independent from it.

In order to calculate the shading threshold, mention above, a sweep of the shading percentage has been included in example 15. The outcome of the mentioned sweep is on figure 15 and shows clearly how around 32% shading is enough to push the solar cell out of forward bias.

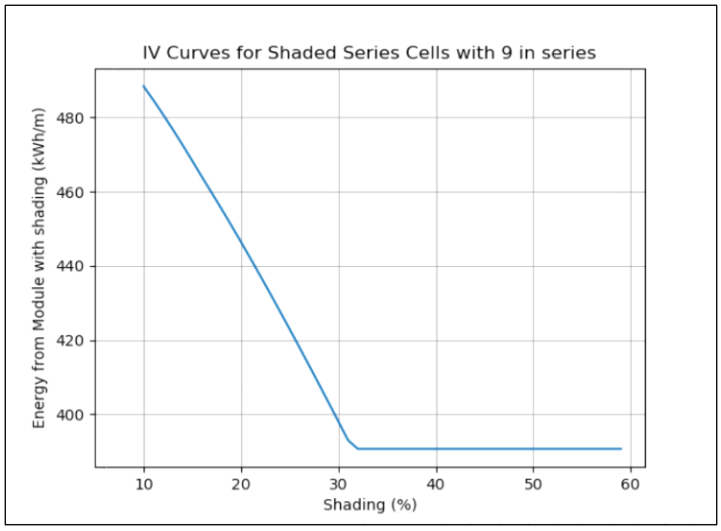


Figure 15: Yearly energy of 9 solar cells in series with 1 shaded.

# CONCLUSION

Along the present document and specially under the chapter “Results” has been shown how PVSimLib presents itself as a powerful and comprehensive tool to anyone interested on solar power. PVSimLib will save to thousands of people the precious time and the tedious work necessary to create their own excel sheet calculators, allowing them to spend that precious time to climb another step in the ladder of knowledge of photovoltaics.

Again, it is important to understand that this library is not final, and will probably never be, it serves as a baseline for everyone. That has been the intention from the beginning, to be as general as possible so it can server both students and scholars as just that, a baseline. For students it will probably be enough, but scholars are expected to get deep into the code and modify it to their needs if necessary. For that purpose, the code has been implemented without any encapsulation to simplify the discovery process that will lead to understanding the code. A necessary step to be able to modify it successfully.

REFERENCES

Honsberg, Christiana, and Stuart Bowden. “Photovoltaic Education Network.” *PVEducation*, www.pveducation.org.

Bowden, Stuart. “Photovoltaic.” *GitHub*, www.github.com/trautsned/photovoltaic.

# APPENDIX A

## PVSIMLIB LIBRARY SOURCE CODE IN PYTHON

[Consult Attached Files]