User's Guide

Overview

Table of Contents

The aim of this library is to provide the I-V characteristic of a simulated photovoltaic generator under mismatched conditions. The final goal is to help understand how mismatched conditions can affect the power production performance and how the implementation of bypass diodes modify the electrical behaviour of the panel. This is a hands-on guide with the recommended procedures to use this library. This section is focused on the proper usage and understanding of the functions and objects contained in the project. For any concept-related question refer to the **Theoretical Documentation** section.

Usage of the project

The library contains 5 different classes:

- **SolarCell.** Represents a PV cell.
- **BypassDiode.** Represents a bypass diode.
- **SolarString.** Represents a string of PV cells and (maybe) a bypass diode.
- **SolarPanel.** Represents a PV panel composed by more than one PV string.
- **SolarSolver.** Object in charge of solving the state of the PV panels.

The first three classes are not actually meant to be instantiated by the final user. However, it is important for the proper usage of the library to understand their hierarchical structure.

The usage of the library can be separated in two parts.

1. Creation of the panel

The first step is to define the structure and properties of the PV panel to be simulated. The structure and external factors of the panel are defined in the **input file** that must be introduced in the SolarPanel constructor.

Once instantiated, it is also possible to modify the parameters of the PV cells that compose the panel. To do this, the SolarPanel class has accessors (defined in the Classes documentation). In the following example the SolarPanel object is instantiated and some of the parameters of it are modified later.

```cpp
#include <iostream>
#include <pv_panel.h>

int main() {
    std::string input_path = "input_data.csv";

    SolarPanel my_panel{input_path};

    my_panel.setCellResistanceSeries(0.001);
    my_panel.setCellResistanceShunt(1000);

    return 0;
}
```

Fig. 1. Example of instantiation of SolarPanel object.

By using these accessors and the information contained in the input file the panel is completely defined.

2. Resolution of the panel

Once the panel is defined, the SolarSolver class can be instantiated by sending the SolarPanel object as an argument in the constructor. The objects that represent the panel (cell, diode and string) are then created.

Then, the SolarSolver object has two main methods that provide simulations of the panel.

- **void calcState(std::string, double).** This method solve every variable in the panel for a given value of voltage. The location of the output file that will be created or overwritten is also given as an argument. This resulting file contains the current through every diode and then, in columns following this order, for every cell:
  - Position of the string the cell is in.
  - Position of the cell in the string.
  - Irradiance in the cell [W/m^2].
  - Temperature of the cell [ºC].
  - Current through the cell [A].
  - Voltage between the terminals of the cell [V].

```cpp
#include <iostream>
#include <pv_panel.h>
#include <pv_solver.h>

int main() {
    std::string input_path = "input_data.csv";
    std::string output_path = "output_data.csv";

    SolarPanel my_panel{input_path};

    my_panel.setCellResistanceSeries(0.001);
    my_panel.setCellResistanceShunt(1000);

    SolarSolver my_solver(my_panel);

    double voltage{17.2};
    my_solver.calcState(output_path, voltage);

    return 0;
}
```

Fig. 2. Example of the calcState() method of the SolarSolver object.

- **void calcIVcharacteristic(std::string).** This method creates the current-voltage I-V characteristic of the panel. If the limits and number of points are not included in the constructor arguments, the limits are dynamically calculated and the characteristic contains 250 points. The location of the output file that will be created or overwritten must be given as an argument. The resulting file contains as rows as points in the characteristic. Each row contains the values of voltage and current, correspondently. The values are separated by a semi-colon ';'.

```cpp
#include <iostream>
#include <pv_panel.h>
#include <pv_solver.h>

int main() {
    std::string input_path = "input_data.csv";
    std::string output_path = "output_data.csv";

    SolarPanel my_panel{input_path};

    my_panel.setCellResistanceSeries(0.001);
    my_panel.setCellResistanceShunt(1000);

    SolarSolver my_solver(my_panel);

    my_solver.calcIVcharacteristic(output_path);

    return 0;
}
```

Fig. 3. Example of the calcIVcharacteristic() method of the SolarSolver object.

- **void calcIVcharacteristic(std::string, double, double, int).** As explained above, this method can be called including the start and end voltage limits and the number of points in the characteristic.

```cpp
#include <iostream>
#include <pv_panel.h>
#include <pv_solver.h>

int main() {
    std::string input_path = "input_data.csv";
    std::string output_path = "output_data.csv";

    SolarPanel my_panel{input_path};

    my_panel.setCellResistanceSeries(0.001);
    my_panel.setCellResistanceShunt(1000);

    SolarSolver my_solver(my_panel);

    double initial_voltage{-5.0}, final_voltage{35.5};
    int number_points{250};
    my_solver.calcIVcharacteristic(output_path,
            initial_voltage, final_voltage, number_points);

    return 0;
}
```

Fig. 4. Example of the overloaded calcIVcharacteristic() method of the SolarSolver object.

For more information on the output files, check the **Output file format** section. These are the main objects and methods meant to be used by final users.

Input file format

This section specifies the format and structure for the input file required to provide the physical information of a solar panel. This file contains the following information:

- Number of strings.
- State of the bypass diode (0-broken / 1-ok).
- Number of cells in every string.
- Irradiance [W/m^2] and temperature [ºC] of every cell.

Attention

The format recommended for the file is a **.csv** file. Other file formats may not being recognised by this library.

The structure to introduce the information in this file must follow these rules:

- Every string is composed by:
  - The first line refers to the bypass diode. It only contains 1 value: 1 if the diode is present and ok, 0 if there is no diode or it is broken.
  - The following lines represent the cells in the string. Each line represent one cell and they are sorted following their physical order. Cell lines contain two double values: the first one belongs to the irradiance [W/m^2] and the second one to the cell temperature [ºC].
  - The string ends when it finds the next bypass diode line (next string) or an empty line (end of the file).
- The library understands the file order (of cells and strings) as their corresponding physical order.
- There can only be bypass diode lines (1 value) and cell lines (2 values).
- Bypass diode lines can only contain a Boolean value.
- Cell lines can only contain two consecutive double values.
- The first empty line will be interpreted as the end of the file and the rest of the file won't be taken into account.

The following example represents a panel made out with 2 strings with 3 and 5 cells correspondingly. The first string doesn't have bypass diode and the second one is partially shaded.

```
0
1000.0;25.0
1000.0;25.0
1000.0;25.0
1
1000.0;25.0
250.5;23.6
213.4;20.7
250.5;23.6
1000.0;25.0
```

Please, consider that the ';' represents the division symbol of the .csv file used. Adapt this symbol according to your .csv editing software configuration.

Output file format

This section explains the format and structure of the output files generated by the library. The files are expected to be .txt or .csv files. Other formats may lead to a unreadable file.

There are two possible formats for the output files, both generated by member methods od the SolarSolver object:

- The calcState() method generates a file with:

- The current through every diode, each one in a new row. A text string is used here:

  > Idiode(x) = value A

- A blank row.
- A header for the rest of the file:

  > String,Cell,Irrad.,Temper.,Curr. (A),Tens. (V)

- Position of the string the cell is in.
- Position of the cell in the string.
- Irradiance in the cell [W/m^2].
- Temperature of the cell [ºC].
- Current through the cell [A].
- Voltage between the terminals of the cell [V]. All the values are separated by comas ‘,’ and have points ‘.’ as decimal separators. For example, the file can result into:

```
Idiode(0) = -5.6e-06 A

String,Cell,Irrad.,Temper.,Curr. (A),Tens. (V)
0,0,1000,298,-4.49165,0.875
0,1,1000,298,-4.49165,0.875
0,2,1000,298,-4.49165,0.875
0,3,1000,298,-4.49165,0.875
```

- The calcIVvharacteristic() method generates a file with:
  - The total voltage in the panel [V].
  - The total current through the panel [A]. The values are separated by semicolons ‘;’. For example, the file can result into:

```
-2;1.06395e+06
-1.85;151713
-1.7;21636.3
-1.55;3088.31
-1.4;443.505
-1.25;66.3648
-1.1;12.5751
-0.95;4.89717
-0.8;3.87092
-0.65;3.80168
-0.5;3.80003
```