PVMOS manual

B.E. Pieters[*]
Institut für Energie und Klimaforschung - IEK5 Photovoltaik,
Forschungszentrum Jülich, 52425 Jülich, Germany
(Dated: October 29, 2014)

I. INTRODUCTION

This is (or rather will be as the current state of this document is far from finished) the manual for the Photo-Voltaic MOdule Simulator (PVMOS). PVMOS is an ordinary differential equation solver using finite-differences specifically designed to electrically model solar modules. For more information on how that works I refer the reader to [1]. The purpose of this document is to document how PVMOS is operated and installed. In the following sections I discuss in order, the installation, basic usage and operating principles and finally a detailed discussion of all available functions.

Before we continue, here are some legalities:

In order to stress the ABSOLUTELY NO WARRANTY bit, here a quote from R. Freund:

```
For all intent and purpose, any description of what the codes are doing should be construed as
being a note of what we thought the codes did on our machine on a particular Tuesday of last
year.  If you're really lucky, they might do the same for you someday.  Then again, do you really
feel *that* lucky?
```

II. INSTALLATION INSTRUCTIONS

PVMOS is a command-line application written entirely in C. For the operation PVMOS depends on several libraries, most notably `cholmod`, for the solving of sparse linear systems. PVMOS has been tested on Linux and Windows systems. To install PVMOS you need to compile the source, for which a Makefile is provided. To install PVMOS you thus need to

1. Install PVMOS's dependencies (`cholmod`,`BLAS`)
2. Edit the Makefile
3. Compile the code (type make)
4. test the executable

The performance of PVMOS is typically strongly dependent on the performance of the sparse linear solver (i.e. `cholmod`). For an optimal performance an optimized `BLAS` library must be used (the reference `BLAS` is comparatively slow). For an optimized `BLAS` there are several options. One option is to use ATLAS which is available for all common CPU architectures and gives a very decent performance. On some architectures you can use `OpenBLAS`, which gives a very good performance (`OpenBLAS` is an actively developed fork of the now unmaintained `GoTo BLAS`). Some CPU manufacturers also publish their own optimized `BLAS` libraries for their CPU's (e.g. Intel and AMD). Per default the makefiles are set up to use `OpenBLAS` as it provides a good performance and is freely available.

[*]Electronic address: `b.pieters@fz-juelich.de`

III. BASIC USAGE AND OPERATING PRINCIPLES

The input for PVMOS is a plain text file with commands. To solve a problem PVMOS is typically called from the command line with as an argument the filename describing the problem. With these input files you can specify the geometry of your solar cell/module including the local properties such as electrode sheet resistances and solar cell properties. You can also specify which calculations you want PVMOS should perform and what data to save. A call to PVMOS from the command line looks like this[3]:

```
pvmos [verbose-option] <input-file>
```

where `[verbose-option]` is an option which how much information PVMOS outputs to stdout, and `<input-file>` is the plain text input file. We first describe the mesh data structure in more detail in the next section.

In its essence PVMOS is a Poisson solver. The Poisson equation is solved for several stacked, 2D "electrodes", where each electrode is coupled to the electrodes above and below (note that a stack of 2D electrodes makes a 3D structure). The electrodes itself simply conductors and behave linearly (Ohmic). However, the connection between the electrodes can be non-linear (e.g. a diode). Now ther are several limitations of the structures that PVMOS handles. The main limitation is that the meshes are 2D as in PVMOS all electrodes share the same 2D mesh. This means that PVMOS is specifically designed for flat layered structures, a less than optimal performance is to be expected for different that flat layered geometries. PVMOS uses straight forward finite-differences to solve the coupled Poisson equations. As such the 2D meshes in PVMOS divide the 2D surface in rectangular elements. Now every element in the mesh must be associated with certain properties. Storing properties on a per element basis woul put a heavy burdon on the memory resources. For that reason elements are grouped into "area's" where each ares constitures a certain combination of properties. A mesh therfore also contains a list of all area's and every element is a member of one of the area's in the mesh (note that an element is always assigned to one area, i.e. you need at least one area for every unique combination of local properties).

In PVMOS we can define more than one mesh at the same time (this is useful as you can build meshes from several meshes, e.g. join two meshes for single cells to one mesh with two series connected cells). In order to reference one particular mesh, each mesh has a name. To reference an area within a mesh you can refer to `<mesh-name>.<area-name>`, i.e. the name of the mesh followed by a dot and the name of the area within that particular mesh. Sometimes we need to select elements in a mesh (for example when we want to assign a set of elements to a certain area). To this end each mesh has a list of selected elements. If you select elements in a mesh the list is occupied by the element ID's, after which you can do operations on the selected elements. Note that elements are selected on a per mesh basis.

The input file is parsed by PVMOS, which sequentially processes the file. PVMOS provides functions to generate and manipulate meshes such that you can generate meshes describing the problem by a sequence of commands. To this end each mesh you define has a name to reference it by. The following section provides a command-reference.

IV. PVMOS COMMAND REFERENCE

Creating Meshes

Keyword	Description
`newmesh`	Create a new, rectangular mesh. The command takes six arguments: 1. `x1`, x-coordinate of the lower left corner 2. `y1`, y-coordinate of the lower left corner 3. `x2`, x-coordinate of the upper right corner 4. `y2`, y-coordinate of the upper right corner 5. `Nx`, Number of elements in x direction 6. `Ny`, Number of elements in y direction 7. `mesh-name`, Name of the new mesh

`joinmesh`	Create new mesh by joining two meshes. Make sure the meshes touch but do not overlap. The function takes offset values as input which allow you to "shift" the second mesh to align it to the first. The command takes 5 arguments: 1. `x_off`, x-offset in coordinate system of the second mesh 2. `y_off`, y-offset in coordinate system of the second mesh 3. `mesh1-name`, Name of the first mesh 4. `mesh2-name`, Name of the second mesh 5. `mesh3-name`, Name of the resulting mesh
`joinmesh_h`	Create new mesh by joining two meshes. Make sure the meshes touch but do not overlap. The function takes a y-offset value as input which allow you to "shift" the second mesh in the y-direction to align it to the first. The x-offset value is the maximal x-value found in the first mesh. The command takes 4 arguments: 1. `y_off`, y-offset in coordinate system of the second mesh 2. `mesh1-name`, Name of the first mesh 3. `mesh2-name`, Name of the second mesh 4. `mesh3-name`, Name of the resulting mesh
`joinmesh_v`	Create new mesh by joining two meshes. Make sure the meshes touch but do not overlap. The function takes an x-offset value as input which allow you to "shift" the second mesh in the x-direction to align it to the first. The y-offset value is the maximal y-value found in the first mesh. The command takes 4 arguments: 1. `x_off`, x-offset in coordinate system of the second mesh 2. `mesh1-name`, Name of the first mesh 3. `mesh2-name`, Name of the second mesh 4. `mesh3-name`, Name of the resulting mesh
`dupmesh`	Duplicate a mesh. The command takes 2 arguments: 1. `mesh1-name`, Name of the mesh to be duplicated 2. `mesh2-name`, Name of the resulting copy
`add_electrode`	Adds an electrode to a certain mesh. The command takes arguments: 1. `mesh-name`, Name of the mesh

Selecting Elements

Keyword	Description
`select_rect`	Select a rectangular area in a mesh. The command takes five arguments: 1. `x1`, x-coordinate of the lower left corner of the selected rectangle 2. `y1`, y-coordinate of the lower left corner of the selected rectangle 3. `x2`, x-coordinate of the upper right corner of the selected rectangle 4. `y2`, y-coordinate of the upper right corner of the selected rectangle 5. `mesh-name`, Name of the mesh
`select_circ`	Select a circular area in a mesh. The command takes four arguments: 1. `x_c`, center x-coordinate of the selected circle 2. `y_c`, center y-coordinate of the selected circle 3. `r`, radius of the selected circle 4. `mesh-name`, Name of the mesh
`select_poly`	Select an area within a polygon-contour. In order to use this command you must first load a polygon from file with the `load_poly` command. The command takes one argument. 1. `mesh-name`, Name of the mesh

`load_poly`	Load a polygon from file. This command is used in conjunction with the `select_poly` command. The command takes one argument. 1. `file-name`, Name of the file describing the polygon (one 2D coordinate per line, i.e., two columns, 1: x-coordinate, 2: y-coordinate)
`deselect`	deselects a selection within a mesh. The command takes one argument. 1. `mesh-name`, Name of the mesh

Manually changing the mesh topology

Keyword	Description
`split_x`	Split selected elements in x-direction. If no elements are selected, all elements are split. As the topology of the mesh changed all selected nodes in the mesh are un-selected after this command. The command takes one argument 1. `mesh-name`, Name of the mesh
`split_y`	Split selected elements in y-direction. If no elements are selected, all elements are split. As the topology of the mesh changed all selected nodes in the mesh are un-selected after this command. The command takes one argument 1. `mesh-name`, Name of the mesh
`split_xy`	Split selected elements in both x- and y-direction. If no elements are selected, all elements are split. As the topology of the mesh changed all selected nodes in the mesh are un-selected after this command. The command takes one argument 1. `mesh-name`, Name of the mesh
`split_long`	Split selected elements in thier longest direction. If no elements are selected, all elements are split. As the topology of the mesh changed all selected nodes in the mesh are un-selected after this command. The command takes one argument 1. `mesh-name`, Name of the mesh
`split_coarse`	Split selected elements until the node-edges are all smaller than a given length. If no elements are selected, all elements are split. As the topology of the mesh changed all selected nodes in the mesh are un-selected after this command. The command takes two arguments 1. `mesh-name`, Name of the mesh 2. `l`, Maximum edge length
`simplify`	Attempt to simplify a mesh. If elements are selected they are un-selected as the topology of the mesh changed. The command takes one argument 1. `mesh-name`, Name of the mesh

Saving and loading meshes

Keyword	Description
`savemesh`	Save a mesh to file in the PVMOS binary format, so it can be loaded again at a later time (see the `loadmesh` command). The command takes two arguments. 1. `mesh-name`, Name of themesh to be saved. 2. `file-name`, filename to save the mesh to.
`loadmesh`	Load a mesh saved to file in the PVMOS binary format (see the `savemesh` command). The command takes two arguments. 1. `file-name`, filename of the file containing the mesh data. 2. `mesh-name`, Name to assign to the loaded mesh

Element-wise export of data

Keyword	Description

`printmesh` Export the mesh in a manner that is plottable with the gnuplot program (www.gnuplot.info/). The resulting plot draws the contour of each element in the mesh. The command takes two arguments:

1. `mesh-name`, Name of the mesh
2. `file-name`, filename to save the data in.

The output file will contain coordinates in columns. For each element the file contains the coordinates of the lower left- and the upper right corners empty line:

```
x1    y1
x2    y1
x2    y2
x1    y2
<empty line>
```

`printconn` Print lateral connections in the electrodes in a format plottable with gnuplot (www.gnuplot.info/). When plotting the file (with vectors) a vector is drawn between the center of each element to the center of the adjacent elements to which it is connected. This routine may be useful when inspecting generated meshes. The command takes two arguments:

1. `mesh-name`, Name of the mesh
2. `file-name`, filename to save the data in.

The output is coordinates in columns. For each element the file contains the following data where `xc`, `yc` is the center of the current element and `xca_i`, `yca_i` is the center coordinate of the i-th adjacent element:

```
xc yc xca_1 yca_1
xc yc xca_2 yca_2
...
```

`printarea` Print the geometry of the mesh which identifies each element and the area it belongs to. The fileformat is laid out such that it is plottable with a surface plot in gnuplot (www.gnuplot.info/). The command takes two arguments:

1. `mesh-name`, Name of the mesh
2. `file-name`, filename to save the data in.

The output file will contain data in columns. The file contains coordinates, the element ID and the corresponding area ID. Note that the parameters for each area can be exported with the `printpars` command. For each element it plots 2 times 2 data lines with an empty line inbetween. Between the data of two elements are two empty lines. This file is formatted such that when plotted with "splot" in gnuplot you can plot a surface for each element in the mesh, which allows you to see the areas in the defined geometry. For each element the folowing data is printed to the file:

```
x1    y1    element-ID area-ID
x1    y2    element-ID area-ID
<empty line>
x2    y2    element-ID area-ID
x2    y1    element-ID area-ID
<empty line>
<empty line>
```

`printV` Print the electrode potentials per element for each stored solution. The output is formatted for gnuplot's splot command, such that a surface plot plots each element individually. The command takes two arguments:

1. `mesh-name`, Name of the mesh
2. `file-name`, filename to save the data in.

The output file will contain data in columns. The file contains coordinates followed by the potential in each electrode for each solution. For each element it plots 2 times 2 data lines with an empty line inbetween. Between the data of two elements are two empty lines. This file is formatted such that when plotted with "splot" in gnuplot you can plot a surface for each electrode in each element in the mesh. For each element the folowing data is printed to the file, where the subscripts indicate the electrode index and the superscript the solution index:

x1 y1 V_0^1 V_1^1 $V_{...}^1$ V_N^1 V_0^2 V_1^2 $V_{...}^2$ V_N^2 ...
x1 y2 V_0^1 V_1^1 $V_{...}^1$ V_N^1 V_0^2 V_1^2 $V_{...}^2$ V_N^2 ...
<empty line>
x2 y2 V_0^1 V_1^1 $V_{...}^1$ V_N^1 V_0^2 V_1^2 $V_{...}^2$ V_N^2 ...
x2 y1 V_0^1 V_1^1 $V_{...}^1$ V_N^1 V_0^2 V_1^2 $V_{...}^2$ V_N^2 ...
<empty line>
<empty line>

`printpar` Print a summary of the parameters per area, including both area-name and area-ID. The command takes two arguments:

1. `mesh-name`, Name of the mesh
2. `file-name`, filename to save the data in.

`printmesh_sel` Same as `printmesh` except that it only exports a selected area specified by its lower left and upper right corners. The command takes six arguments:

1. `mesh-name`, Name of the mesh
2. `x1`, x-coordinate of the lower left corner of the selected rectangle
3. `y1`, y-coordinate of the lower left corner of the selected rectangle
4. `x2`, x-coordinate of the upper right corner of the selected rectangle
5. `y2`, y-coordinate of the upper right corner of the selected rectangle
6. `file-name`, filename to save the data in.

`printconn_sel` Same as `printconn` except that it only exports a selected area specified by its lower left and upper right corners. The command takes six arguments:

1. `mesh-name`, Name of the mesh
2. `x1`, x-coordinate of the lower left corner of the selected rectangle
3. `y1`, y-coordinate of the lower left corner of the selected rectangle
4. `x2`, x-coordinate of the upper right corner of the selected rectangle
5. `y2`, y-coordinate of the upper right corner of the selected rectangle
6. `file-name`, filename to save the data in.

`printarea_sel` Same as `printarea` except that it only exports a selected area specified by its lower left and upper right corners. The command takes six arguments:

1. `mesh-name`, Name of the mesh
2. `x1`, x-coordinate of the lower left corner of the selected rectangle
3. `y1`, y-coordinate of the lower left corner of the selected rectangle
4. `x2`, x-coordinate of the upper right corner of the selected rectangle
5. `y2`, y-coordinate of the upper right corner of the selected rectangle
6. `file-name`, filename to save the data in.

`printV_sel` Same as `printV` except that it only exports a selected area specified by its lower left and upper right corners. The command takes six arguments:

1. `mesh-name`, Name of the mesh
2. `x1`, x-coordinate of the lower left corner of the selected rectangle
3. `y1`, y-coordinate of the lower left corner of the selected rectangle
4. `x2`, x-coordinate of the upper right corner of the selected rectangle
5. `y2`, y-coordinate of the upper right corner of the selected rectangle
6. `file-name`, filename to save the data in.

`printIV` Export the IV characteristics of the device. Exports a file with two columns, the first contains all the simulated applied voltages and the second the corresponding total currents. The command takes two arguments:

1. `mesh-name`, Name of the mesh
2. `file-name`, filename to save the data in.

`surfVplot`	Export the front and back electrode voltages for a specific solution. Unlike the `print`-commands like `printV` the data is interpollated and mapped on a regular mesh. The command takes eight arguments: 1. `mesh-name`, Name of the mesh 2. `x1`, x-coordinate of the lower left corner of the selected rectangle 3. `y1`, y-coordinate of the lower left corner of the selected rectangle 4. `x2`, x-coordinate of the upper right corner of the selected rectangle 5. `y2`, y-coordinate of the upper right corner of the selected rectangle 6. `Nx`, Number of points in the regular mesh along the x-direction 7. `Ny`, Number of points in the regular mesh along the y-direction 8. `Va`, Applied voltage (if the sepcified voltage is not available the closest value will be taken 9. `file-name`, filename to save the data in.
`surfPplot`	Export the local power density for a specific solution. Just like in the `surfVplot` command the data is interpollated and mapped on a regular mesh. The command takes eight arguments: 1. `mesh-name`, Name of the mesh 2. `x1`, x-coordinate of the lower left corner of the selected rectangle 3. `y1`, y-coordinate of the lower left corner of the selected rectangle 4. `x2`, x-coordinate of the upper right corner of the selected rectangle 5. `y2`, y-coordinate of the upper right corner of the selected rectangle 6. `Nx`, Number of points in the regular mesh along the x-direction 7. `Ny`, Number of points in the regular mesh along the y-direction 8. `Va`, Applied voltage (if the sepcified voltage is not available the closest value will be taken 9. `file-name`, filename to save the data in.

Manipulating local properties

Keyword	Description
`assign_properties`	Assign nodes to a defined area. If no nodes are selected all nodes in the mesh are assigned to the specified area. The command takes one arguments: 1. `area-name`, Name of the area (<mesh-name>.<area-name>)
`set_Rel`	Set an electrode resistance. If the specified area does not exist it will be newly created. The command takes three arguments: 1. `area-name`, Name of the area (<mesh-name>.<area-name>) 2. `electrode-index`, Index of the electrode. The first electrode has index 0 3. `value`, Sheet resistance value (Ω)
`set_Rvp`	Set the contact resistance between the positive node and an electrode. Together with `set_Rvn` this command allows the application of an extranal voltage. The command takes three arguments: 1. `area-name`, Name of the area (<mesh-name>.<area-name>) 2. `electrode-index`, Index of the electrode. The first electrode has index 0 3. `value`, Contact resistance ($\Omega\mathrm{cm}^2$)
`set_Rvn`	Set the contact resistance between the negative node and an electrode. Together with `set_Rvp` this command allows the application of an extranal voltage.The command takes three arguments: 1. `area-name`, Name of the area (<mesh-name>.<area-name>) 2. `electrode-index`, Index of the electrode. The first electrode has index 0 3. `value`, Contact resistance ($\Omega\mathrm{cm}^2$)

`set_JV`

Specify a tabular data set to use as a JV characteristics. The command takes three arguments:

1. `area-name`, Name of the area (<mesh-name>.<area-name>)
2. `connection-index`, Index of the connection. The first connection, between electrode 0 and 1, has index 0
3. `file-name`, Name of a file containing two columns, voltage and current density (V, $A\mathrm{cm}^{-2}$)

`set_2DJV`

Specify a two-diode model for the JV characteristics. The command takes eight arguments:

1. `area-name`, Name of the area (<mesh-name>.<area-name>)
2. `connection-index`, Index of the connection. The first connection, between electrode 0 and 1, has index 0
3. `J01`, Saturation current density for the first diode (ideality factor one) ($A\mathrm{cm}^{-2}$)
4. `J02`, Saturation current density for the second diode (ideality factor two) ($A\mathrm{cm}^{-2}$)
5. `Jph`, Photo current density ($A\mathrm{cm}^{-2}$)
6. `Rs`, Series resistance ($\Omega\mathrm{cm}^2$)
7. `Rsh`, Shunt resistance ($\Omega\mathrm{cm}^2$)
8. `Eg`, Band gap (eV)

`set_1DJV`

Specify a one-diode model for the JV characteristics. The command takes eight arguments:

1. `area-name`, Name of the area (<mesh-name>.<area-name>)
2. `connection-index`, Index of the connection. The first connection, between electrode 0 and 1, has index 0
3. `J0`, Saturation current density ($A\mathrm{cm}^{-2}$)
4. `nid`, Ideality factor
5. `Jph`, Photo current density ($A\mathrm{cm}^{-2}$)
6. `Rs`, Series resistance ($\Omega\mathrm{cm}^2$)
7. `Rsh`, Shunt resistance ($\Omega\mathrm{cm}^2$)
8. `Eg`, Band gap (eV)

`set_R`

Specify a resistance for the JV characteristics. The command takes three arguments:

1. `area-name`, Name of the area (<mesh-name>.<area-name>)
2. `connection-index`, Index of the connection. The first connection, between electrode 0 and 1, has index 0
3. `R`, Resistance ($\Omega\mathrm{cm}^2$)

`set_T`

Specify a local temperature. The command takes two arguments:

1. `area-name`, Name of the area (<mesh-name>.<area-name>)
2. `T`, Temperature (K)

Numerical Settings

Keyword Description

`set_SplitX`

It is sometimes useful to prevent the adaptive meshing algorithms from splitting certain nodes in x- or y-direction. This commands toggles the splitting of nodes in x-direction for a specified area (per default all nodes can be split in all directions). The command takes one argument:

1. `area-name`, Name of the area (<mesh-name>.<area-name>)

`set_SplitY`

It is sometimes useful to prevent the adaptive meshing algorithms from splitting certain nodes in x- or y-direction. This commands toggles the splitting of nodes in y-direction for a specified area (per default all nodes can be split in all directions). The command takes one argument:

1. `area-name`, Name of the area (<mesh-name>.<area-name>)

`maxiter`	Set the maximum number of iterations for solving the non-linear system. The command takes one argument: 1. `maxiter`, Maximum number of iterations (default: 25)
`tol_V`	Absolute voltage tolerance for the break-off criterion. The command takes one argument: 1. `tol_V`, Absolute voltage tolerance V (default: 10^{-5}V)
`rel_tol_V`	Relative voltage tolerance for the break-off criterion. The command takes one argument: 1. `tol_V`, Relative voltage tolerance $-$ (default: 10^{-5})
`tol_kcl`	Absolute current tolerance for the break-off criterion (KCL stands for Kirchhoff's Current Law). The command takes one argument: 1. `tol_kcl`, KCL tolerance A (default: 10^{-5}A)
`rel_tol_kcl`	Relative current tolerance for the break-off criterion (KCL stands for Kirchhoff's Current Law). The command takes one argument: 1. `tol_kcl`, Relative KCL tolerance $-$ (default: 10^{-5})

Solving

Keyword	Description
`solve`	Solve the system (do a voltage sweep). The command takes four arguments: 1. `mesh-name`, Name of the mesh 2. `V_start`, Start voltage 3. `V_end`, End voltage 4. `N_step`, Number of voltage steps
`adaptive_solve`	Solve the system and adapt the mesh at one specified voltage. The command takes four arguments: 1. `mesh-name`, Name of the mesh 2. `V_a`, Applied voltage 3. `threshold`, Ralative threshold for node splitting, a parameter between 0 and 1 that controls how agressive the mesh is adapted, where lower values lead to a more agressive mesh adaption (typical values betwwen 0.3 and 0.5). 4. `N_step`, Number adaptive meshing iterations

Verbosity levels

Keyword	Description
`out_quiet`	Set verbosity to the minimum (only says something when it crashes). The command takes no arguments.
`out_normal`	Set verbosity to the normal level. The command takes no arguments.
`out_verbose`	Output additional data that may be interesting. The command takes no arguments.
`out_degug`	Output additional data that is only interesting for someone who is chasing bugs in the code. The command takes no arguments.

V. EXAMPLE:MONOLITHICALLY SERIES CONNECTED MINI-MODULE AND A DEFECT

```
####################################################################################################
# In this example we create an 8x8 cm a-Si:H mini module with a defect.
####################################################################################################


####################################################################################################
# create the meshes. The following meshes are created
# cell_a: mesh for the active part of a cell stripe (i.e., not including dead area)
# p1: mesh for the p1 laser line
# da: mesh for the area's between laser lines
# p2: mesh for the p2 laser line
# p3: mesh for the p3 laser line
# cp: mesh for a contact trip connected to the positive electrode
# cn: mesh for a contact strip to the negative electrode
###################################################################################################
#           x1      y1      x2       y2  Nx   Ny   name
newmesh     0.0     0.0     0.958    8   50   21   cell_a
newmesh     0.0     0.0     8e-3     8   1    21   p1
newmesh     0.0     0.0     12e-3    8   1    21   da
newmesh     0.0     0.0     5e-3     8   1    21   p2
newmesh     0.0     0.0     5e-3     8   1    21   p3
newmesh     -8e3    0.0     0        8   1    21   cp
newmesh     0.0     0.0     8e-3     8   1    21   cn
# Note I gave the cp mesh a negative x1 and 0 for x2, I will explain why later...

###################################################################################################
# A mesh consists of nodes. The nodes in a mesh belong to areas. All meshed are initialized with
# an area which is the same as the name of the mesh, and all nodes are assigned to it. You can
# change the properties by using various commands starting with set_XX. You can also define new
# areas and assign nodes to these new areas. More on that later. We first define the properties
# per mesh. To refer to a certain area you must specify the mesh and the area name like so:
# <mesh-name>.<area-name>
# Here we set the diode properties of the area cell_a in the mesh cell_a:
###################################################################################################
#           mesh.area       conn J0           n   Jph      Rs      Rsh          Eg
set_1DJV    cell_a.cell_a   0    2.4803e-12   1.5 0.015    7.6529  6.1273e+04   1.7

###################################################################################################
# here we set the sheet resistances for the same area, for the positive and negative electrodes
###################################################################################################
#           mesh.area       electrode    Rel
set_Rel     cell_a.cell_a   0            5
set_Rel     cell_a.cell_a   1            0.5

###################################################################################################
# here follow various areas for the various meshes
# set_R sets the resistance between front and back electrode, here we remove the contact
###################################################################################################
set_R       p1.p1   0   1e90
set_Rel     p1.p1   0   1e9
set_Rel     p1.p1   1   0.5
set_SplitX  p1.p1

set_1DJV    da.da   0   2.4803e-12   1.5 0.015    7.6529   6.1273e+04  1.7
set_Rel     da.da   0   5
set_Rel     da.da   1   0.5
set_SplitX  da.da

set_R       p2.p2   0   1e-2
set_Rel     p2.p2   0   5
set_Rel     p2.p2   1   0.5
set_SplitX  p2.p2

set_R       p3.p3   0   1e90
set_Rel     p3.p3   0   5
set_Rel     p3.p3   1   1e9
set_SplitX  p3.p3

set_R       cp.cp   0   1e90
set_Rel     cp.cp   0   1e-3
set_Rel     cp.cp   1   1e3
set_Rvp     cp.cp   0   1e-8
set_SplitX  cp.cp

set_R       cn.cn   0   1e90
set_Rel     cn.cn   0   1e-3
set_Rel     cn.cn   1   1e3
set_Rvn     cn.cn   0   1e-8
set_SplitX  cn.cn

####################################################################################################
# Now we start to assemble the complete mesh out of the various parts. When joining meshes we
# will use the coordinate system of the first mesh. The coordinate system of the second mesh can
# be shifted by using an x and y offset value. Note that you have to take care not to make the
# meshes overlap I have not (yet) inplemented error checking for this! In order to keep track
# of the dimensions and avoid overlapping meshes I keep track of the total width of the mesh on
# the comment lines below the join commands.
####################################################################################################
#           xoff    mesh1            mesh2    mesh_out
joinmesh_h  0.0     cell_a           p1       cell_p1
joinmesh_h  0.0     cell_p1          da       cell_p1da
joinmesh_h  0.0     cell_p1da        p2       cell_p1dap2
joinmesh_h  0.0     cell_p1dap2      da       cell_p1dap2da
joinmesh_h  0.0     cell_p1dap2da    p3       cell_p1dap2dap3
# should be 1 cm wide

####################################################################################################
# Note that we now have many meshes defined:
# 1.  cell_a
# 2.  p1
# 3.  da
# 4.  p3
# 5.  p3
# 6.  cp
# 7.  cn
# 8.  cell_p1
# 9.  cell_p1da
# 10. cell_p1dap2
# 11. cell_p1dap2da
# 12. cell_p1dap2dap3
# The last mesh is cell stripe including dead area. We can easily series connect several cells
# by joining several instances of the last mesh in a row. Here we series connect 40 cells
####################################################################################################
joinmesh_h  0.0 cell_p1dap2dap3  cell_p1dap2dap3  2cells
joinmesh_h  0.0 2cells           2cells           4cells
joinmesh_h  0.0 4cells           4cells           8cells
# 8x8 cm2, 8 cells


####################################################################################################
# Now we add the contacts. The resulting mesh will be called: minimodule
####################################################################################################
joinmesh_h  0.0 cp          8cells   cp8cells
# Remember that for the cp mesh x1 was nagative and x2 was 0. This means that after joining them
# I have the 8x8 cm^2 cells in the square (0,0) and (8,8)
joinmesh_h  0.0 cp8cells    cn       minimodule


####################################################################################################
# Now for some defects!
####################################################################################################
# First we create a new area with the defect properties
set_Rel     minimodule.shunt     0   5
set_Rel     minimodule.shunt     1   0.5
set_R       minimodule.shunt     0   0.0001

####################################################################################################
# we will make the defect in the center of the third cell
# First thing to take care of is that the mesh is fine around the defect so we can
# properly define it. Currently at the defect location the elements are
# 0.02x0.4 cm^2. We select an area with radius 2 cm and refine it a bit
###################################################################################################
select_circ 2.5 4   2   minimodule
split_coarse    minimodule  0.2

###################################################################################################
# now the elements in a radius of 2 cm around the defect are 0.02x0.2 cm^2
# we do another run, this time in an area with a 1 cm radius around the defect
###################################################################################################
select_circ 2.5 4   1   minimodule
split_coarse    minimodule  0.1

###################################################################################################
# repeat ...
###################################################################################################
select_circ 2.5 4   0.5   minimodule
split_coarse    minimodule  0.05

select_circ 2.5 4   0.25 minimodule
split_coarse    minimodule  0.025

###################################################################################################
# We create a defect with a radius of 1mm, a a rule of thumb I want at leat 10 elemnts along the
# radius, so this time I have a circle slightly larger than the defect with a maximum mesh distance
# of 0.01.
###################################################################################################
select_circ 2.5 4   0.125   minimodule
split_coarse    minimodule  0.01

###################################################################################################
# Time to create ourselves a defect
###################################################################################################
select_circ 2.5 4   0.1 minimodule
assign_properties   minimodule.shunt

###################################################################################################
# Solve stuff and export data
###################################################################################################
# Dump the initial mesh so you can plot it later
printmesh       minimodule  mesh0.dat

# Do 2 iterations of adaptive solving a@5 volt with a factor 0.3
adaptive_solve  minimodule  5    0.3 2

# Dump the refined mesh so you can plot it later and compare to the initial one
printmesh       minimodule  mesh1.dat

# solve IV from 0-35 V in 36 steps (1V steps including both 0 and 35 V)
solve       minimodule  0   8    41\n");
surfVplot   minimodule  0   0    8   8   200   200 4.4  minimodule_44.dat
surfVplot   minimodule  0   0    8   8   200   200 0    minimodule_00.dat
surfVplot   minimodule  0   0    8   8   200   200 6.4  minimodule_64.dat
printIV     minimodule  IV_minimodule.dat
```

---

[1] B. E. Pieters, "PVMOS: A Free and Open Source Simulation Tool for Solar Modules," *submitted to: Journal of Photovoltaics*, 2014.

[2] T. A. Davis. (retrieved 22 May 2014) BLAS performance bug and its effect on sparse "bench" in MATLAB 7.4. [Online]. Available: http://www.cise.ufl.edu/research/sparse/cholmod/blasbug.html

[3] Many BLAS versisons seem to suffer from a bug which makes that `cholmod` performance deteriorates when multi-threading is enabled [2]. If your BLAS library has multi-threading enabled, performance may improve if you set environment variables

such that BLAS will use only one thread. In case of `OpenBLAS` with OpenMP you need to set `OMP_NUM_THREADS=1`, e.g. `export OMP_NUM_THREADS=1` in bash or `SET OMP_NUM_THREADS=1` in DOS