UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

FINAL DEGREE PROJECT

**Degree in Energetic Engineering**

# ENERGY-LEVEL SIMULATOR FOR MICRO-GRIDS

## Volume III

## User Manual


**Author:** Adrià Bové Salat
**Director:** Herminio Martínez García
**Co-Director**: Guillermo Velasco Quesada

**Call:** June 2018

# Índex

# 1. User manual

## 1.1. How to call the models' functions?

The created functions can be classified in two different groups in terms of calling them.

- The ones that only give information and don't receive any kind of modification, these are the majority of them and are listed below:
  - "critical_loads.m" (code on Annexes)
  - "grid.m" (code on Annexes)
  - "grid_purchasing_price_function.m" (code on Annexes)
  - "grid_selling_price_function.m" (code on Annexes)
  - "irradiance_function.m" (code on Annexes)
  - "irradiance_forecast_function.m" (code on Annexes)
  - "temperature_function.m" (code on Annexes)
  - "temperature_forecast_function.m" (code on Annexes)
  - "PV.m" (code on Annexes)
  - "PV_inverter.m" (code on Annexes)
- The ones that can give information, receive and be internally modified.
  - "battery.m" (code on Annexes)
  - "genset.m" (code on Annexes)

In the following sections they will be explained one by one how to call them and how to save the output data of them.

### 1.1.1. The one direction functions

An example of calling them is:

```
[nom_power_loads{SMG}(:,i),min_power_loads{SMG}(:,i),priority{SMG}(:,i),
load_name{SMG},load_identifier{SMG}(:,i)]= critical_loads(time,SMG);
```

Two parts can be differentiated, the part before the "=" sign and the part after the equal.

The part after " `critical_loads(time,SMG);` " is the function calling doing this the matlab will run the function of the loads. In parenthesis there are the inputs. The first one is the "time" that is automatically assigned by the time for-loop (see section 4.4.2 of the Technical Report) and the "SMG" that is the number of the Smart Micro Grid being studied this value has to be assigned before calling

the function. If the function is used in a SMG for-loop will be also automatically assigned. The ";" is used to not display in every iteration the value on the "Command Window".

This function has five outputs, these outputs can be stored in memory or not in the shown code they would be stored.

To have the five outputs is necessary to write a vector before the "=" with the same number of variables in this way:

[variable1, variable2, variable3, variable4, variable5] `= critical_loads(time,SMG);`

Doing it this way will not store the values along all the simulation, this maybe is interesting or maybe not it depends. The names of the variables have to be easy to understand what do they store.

How does this of storing the variables work?

After the variable name will be necessary to insert this complementary code "`{SMG}(:,i)`".

{SMG} creates a cell and if after this ( : , i ) is written will create a matrix in the cell. Then everything will be well stored, one cell per each SMG and the matrix will have one column per each instant and one line per each load in this case. The "i" variable is the variable that says the iteration number of the time for-loop. This variable should be untouchable and unmodifiable in the algorithm. In the example the variable loads_name only has {SMG}, why? Because it's only interesting to store the name of the loads once and not for every instant.

The grid function can be called this way:

```
[P_max_grid{SMG}(i),grid_purchasing_price{SMG}(i),grid_selling_price{SMG}
(i)]=grid(time,SMG);
```

It has three outputs and the principal difference between the loads function is that they are values and not vectors and when they have to be stored only writing (i) in enough.

The others can be called as is presented below.

```
aux_buying_price=grid_purchasing_price_function(time);
aux_selling_price=grid_selling_price_function(time);
[temperature(:,i)] = temperature_function(time);
[temperature_f(:,i)] = temperature_forecast_function(time);
[irradiance(:,i)] = irradiance_function(time);
[irradiance_f(:,i)] = irradiance_forecast_function(time);
[P_max{SMG}(i)]= PV_inverter(time,SMG);
[V1{SMG}{i},I{SMG}{i},P{SMG}{i}]=PV(time,SMG);
```

The functions purchasing price, selling price, temperature, irradiance and PV are internally called in other functions, they can be called in the algorithm with the aim to have more information to take decisions but is possible to work without calling them.

### 1.1.2. The battery function

The battery function is a little bit trickier because it has two steps or in other words two ways of calling it.

The first way is for information and is called in the step 1 (see **¡Error! No se encuentra el origen de la referencia.**) like this:

```
[bat_power_vector1{SMG},bat_autonomy_vector1{SMG},SoC1{SMG}] =
battery(time,SMG);
```

The variables usually are not enough interesting to store them for more than one instant. But enough to store them instantly for each SMG.

The first output "bat_power_vector1" is a vector of 6 values, the first three are positive and represent the power the battery can provide, the last 3 are negative and represent the power the battery can absorb. They are three to do an interpolation if needed. The case of the battery is lineal but in case that the model is improved, this will be useful.

bat_power_vector1 = 5.0000  3.3333  1.6667  -3.3333  -6.6667  -10.0000

The second output "bat_autonomy_vector1" is a vector of three values that tell for the first three values of "bat_power_vector1" the duration of the battery until it becomes flat. It is expressed in hours.

bat_autonomy_vector1 = 3.9630  5.9445  11.8890

The third output "SoC1" is the current State of Charge of the battery in %. It is only one value.

The second way is called in the step 7 and 8 (see 5.1.7 and 5.1.8 TR) like this:

```
[bat_power{SMG}(i),bat_autonomy{SMG}(i),SoC{SMG}(i)] =
battery(time,SMG,step,command);
```

This time the variables are stored in time using (i) and two inputs are added:

- step, indicates to the function which part of code has to activate, because it's the second way to call the function this value must be "2". step=2; If this input is not completed automatically the function assumes it is "1".
- command, indicates to the battery the power demanded in kW, if it is positive the battery will be discharged and if is negative will be charged.

The outputs now are only a value and not a vector, the “bat_power” is the same power indicated in the “command”. The “bat_autonomy” is the time in hours at the “bat_power” value taking into account the efficiency, the value is a “NaN” when charging. And the “SoC” is the State of Charge at the beginning of the instant. The SoC will change for the next instant. It is important not to call this function in this way more times than one in the same instant because it will discharge or charge the battery more times.

### 1.1.3. The genset function

The genset function is the trickiest one because there are 4 ways to call it and also has 4 different states (see 4.1.4. on Technical Report).

The first way is informative as the battery function is called in step 1.

```
[genset_power_vector1{SMG},genset_emission_vector1{SMG},
genset_cost_vector1{SMG},genset_autonomy_vector1{SMG},genset_state1{SMG},
fuel_level1{SMG}] = genset(time,SMG);
```

The outputs are:

genset_power_vector1 = 10 5.5 1 in kW.

genset_emission_vector1 = 0.021 0.016 0.005

genset_cost_vector1 = 0.03 0.024 0.007

genset_autonomy_vector1 = 250 150 20 in hours.

genset_state1 = 2

fuel_level1 = 100 in %.

The second way to call is the binding one and only has to be called once at the instant. It is called in steps 7 and 8 (see 5.1.7 and 5.1.8 of TR). It’s called like this:

```
[genset_power{SMG}(i),genset_emission {SMG}(i), genset_cost{SMG}(i),
genset_autonomy {SMG}(i),genset_state{SMG}(i),genset_fuel_level{SMG}(i)]=
genset(time,SMG,gstep,gcommand);
```

The new inputs are:

gstep=2;

gcommand=5; Is the power demanded in kW.

And the outputs are single values instead of vectors all of them depend on the output power demanded by the EMS. As can be seen they are stored.

The third way to call the function is setting gstep=3 and is used to change the state of the generator for further information see 4.1.4 TR. This step can be called in steps 7 and 8 (see 5.1.7 and 5.1.8 of TR) and before the binding way. It's called like this:

```
genset(time,SMG,3,1)
```

or

```
genset(time,SMG,3,3)
```

There are no outputs, it just changes internally the state of the generator.

The fourth way is setting gstep=4. It is used to set an alarm or notification to refill the tank. And the function automatically after the specified time will refill the tank. It's called like this:

```
genset(time,SMG,4)
```

There are not outputs either.

## 1.2. How to plot results?

There are some types of results and them can be plotted in many ways the most commonly used are showed below and the code is pasted too.

### 1.2.1. Time evolving results

The values that change over the time and it is interesting to see their results can be plotted as follows (see Figure 1.1. ).

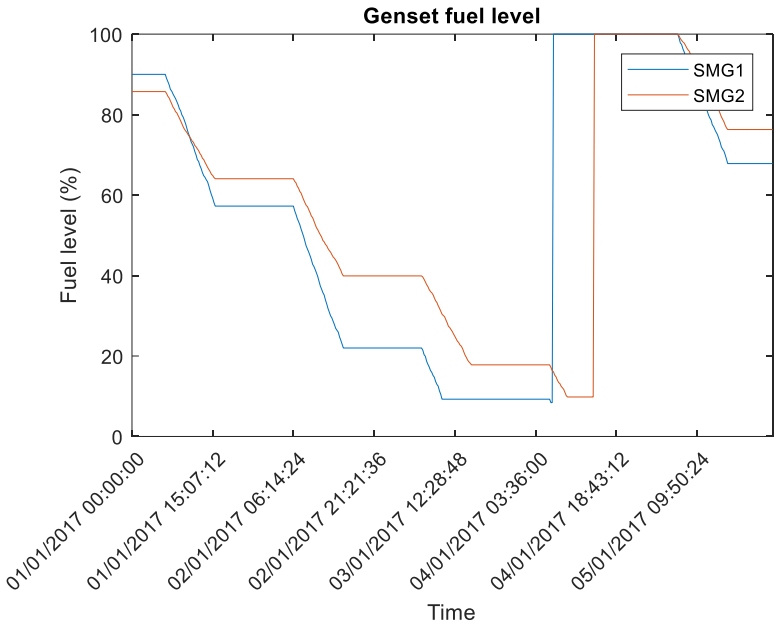


**Figure 1.1.** Fuel level results

The code used here is:

```
figure
for SMG=1:n_SMG

    plot(time_vector,genset_fuel_level{SMG})
    hold on

end

title('Genset fuel level')
ylabel('Fuel level (%)')
xlabel('Time')
legend('SMG1','SMG2')
set(gca, 'Xtick',
time_vector(1:round(length(time_vector)/points):length(time_vector)),
'XTickLabel',x_labels);
xtickangle(45)
```

This code can be fully reused only changing the parameter we want to plot (inside the plot function) and changing the names of the title, ylabel and xlabel.

### 1.2.2. Long vector results per instant

Is the case of the PV panels that have an output of a long vector each instant. The result of the plot is as follows.

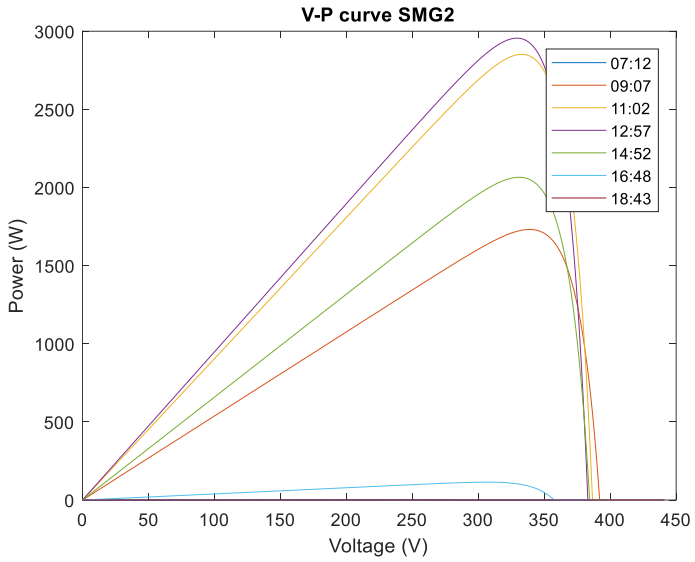


**Figure 1.2.** PV V-P curve

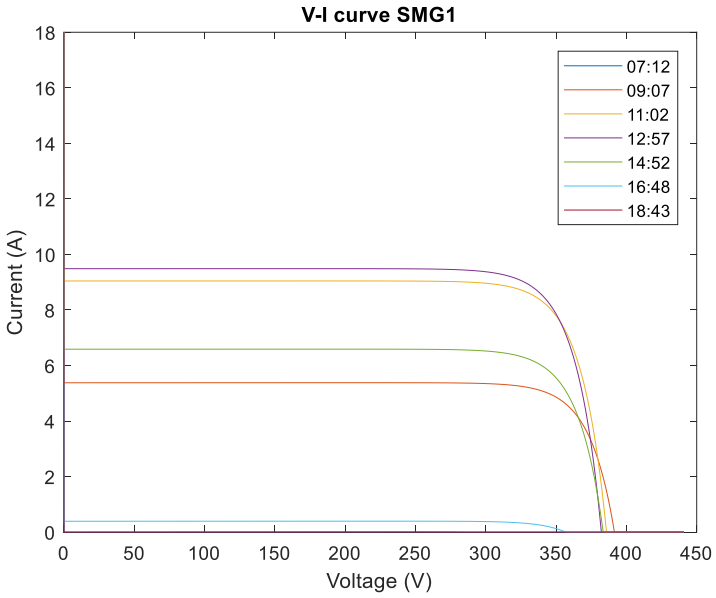


**Figure 1.3.** PV V-I curve

The code used is pasted below, this code returns the two figures for each SMG.

```
%% Plotting PV curves use it with responsability
starting=31; %Is the the position of the instant we want to start
plotting
jump=8; %Is the number of time steps between curve and curve
ending=80; %Is the the position of the instant we want to start plotting
for SMG=1:n_SMG
    figure
    k=1;
```

```
    for j=starting:jump:ending
        plot(V1{SMG}{j},I{SMG}{j})
        hold on
        times(k)=time-(i-j)*time_steps;
        k=k+1;
    end
    title(strcat('V-I curve SMG',num2str(SMG)))
    xlabel('Voltage (V)')
    ylabel('Current (A)')
    legend(datestr(times,'HH:MM'))
    hold off
end

for SMG=1:n_SMG
    figure
    k=1;
    for j=starting:jump:ending
        plot(V1{SMG}{j},P{SMG}{j})
        hold on
        times(k)=time-(i-j)*time_steps;
        k=k+1;
    end
    title(strcat('V-P curve SMG',num2str(SMG)))
    xlabel('Voltage (V)')
    ylabel('Power (W)')
    legend(datestr(times,'HH:MM'))
    hold off
end
```

There are three interesting variables (“starting”, “jump”, “ending”) at the beginning of the code that allow to set when to start plotting, when to end and the intervals in between, always referenced on vector positions for the index “i”.

### 1.2.3. Accumulative results

Sometimes is interesting to compare the total of similar variables of different SMG or generators, then a bar plot can be interesting.

For example the figures Figure 1.4. Figure 1.5. and Figure 1.6. are related with the generator.

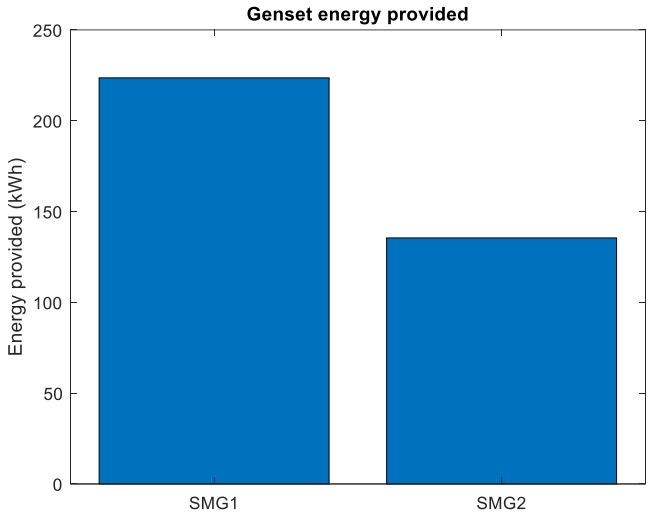


**Figure 1.4.** Genset energy provided

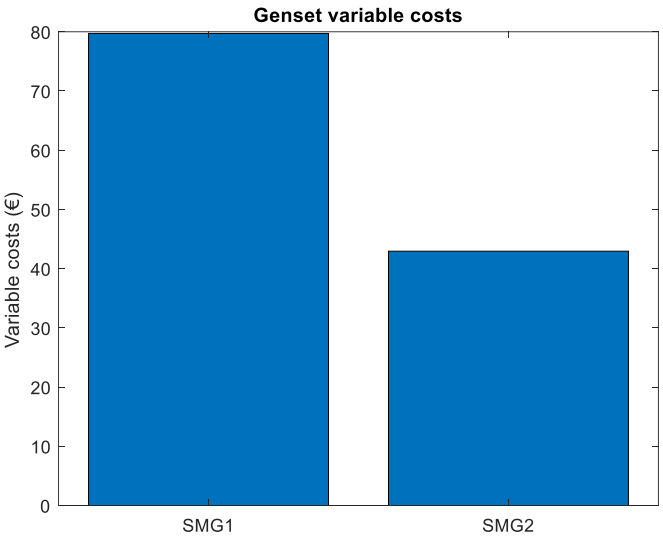


**Figure 1.5.** Genset variable costs

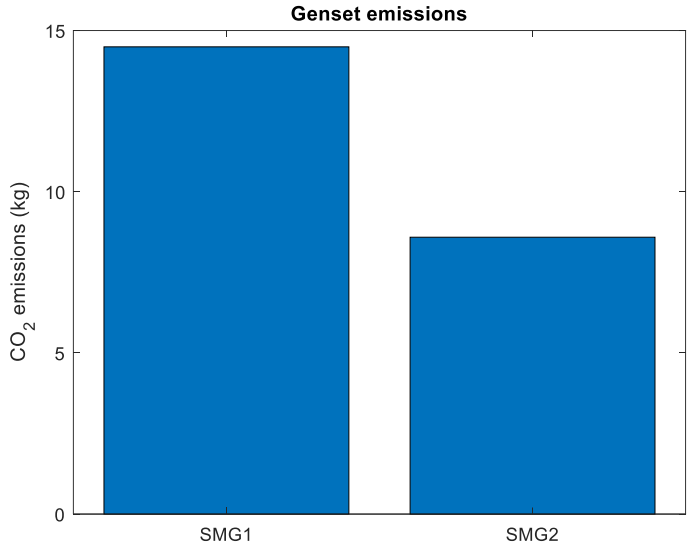


**Figure 1.6.** PV Genset $CO_2$ emissions

The code used to plot this bar is:

```
for SMG=1:n_SMG

    emissions(SMG)=sum(genset_emission{SMG});
    genset_costs(SMG)=sum(genset_cost{SMG});
    genset_produced_energy(SMG)=sum(genset_power{SMG}.*(time_steps*24));

end
figure
c = categorical({'SMG1','SMG2'});
bar(c,emissions)
title('Genset emissions')
ylabel('CO_2 emissions (kg)')

figure
c = categorical({'SMG1','SMG2'});
bar(c,genset_costs)
title('Genset variable costs')
ylabel('Variable costs (€)')

figure
c = categorical({'SMG1','SMG2'});
bar(c,genset_produced_energy)
title('Genset energy provided')
ylabel('Energy provided (kWh)')
```