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 IV

Annexes

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

Call: June 2018

Índex

Annex A

All the code of the program is listed in this annex.

1.1. “critical_loads.m”

```
function
[nom_power_loads,min_power_loads,priority,load_name,load_identifier]=
critical_loads(time,S_SMG)
% Constants defined on the general program

persistent loads_data loads_data_str n_SMG time_points
nominal_power_points minimum_power_points
if isempty(time_points)

n_SMG=getappdata(0,'number_of_SMG');


% Data acquisition
for SMG = 1:n_SMG
    [loads_data{SMG},loads_data_str{SMG}] =
xlsread('loads_data.xlsx',num2str(SMG)); % Imports the data of buying
prices
end

% Data processing
% The first step is to find the time points where each load needs power
not
% very much precision is needed because the loads are assumed to have
% constant consumption.

for SMG = 1:n_SMG
    for i=1:length(loads_data{SMG}(:,1))
        k=0;
        for j=1:2:loads_data{SMG}(i,12)*2

time_points{SMG}{i}(j)=datenum(strcat(num2str(loads_data{SMG}(i,7)),'/',n
um2str(loads_data{SMG}(i,8)),'/',num2str(loads_data{SMG}(i,9)),'.',num2st
r(loads_data{SMG}(i,4)),':',num2str(loads_data{SMG}(i,5)),':',num2str(loa
ds_data{SMG}(i,6))),'dd/mm/yyyy.HH:MM:SS')+k*(loads_data{SMG}(i,11));
            time_points{SMG}{i}(j+1)=(loads_data{SMG}(i,10)-
0.0001)/24+time_points{SMG}{i}(j);
            k=k+1;
        end
    end
end

% The second step is to create the points where the loads will consume
% power, the odd points will be the power, the even points will be 0,
later
% with the 'previous' interpolation it will create the load curves.
```

```
for SMG = 1:n_SMG
    for i=1:length(loads_data{SMG}(:,1))
        for j=1:2:loads_data{SMG}(i,12)*2
            nominal_power_points{SMG}{i}(j)=loads_data{SMG}(i,2);
            nominal_power_points{SMG}{i}(j+1)=0;
        end
    end
end

% The same to create the points of minimum power points

for SMG = 1:n_SMG
    for i=1:length(loads_data{SMG}(:,1))
        for j=1:2:loads_data{SMG}(i,12)*2
            minimum_power_points{SMG}{i}(j)=loads_data{SMG}(i,3);
            minimum_power_points{SMG}{i}(j+1)=0;
        end
    end
end

% It is necessary to unite the points of the same loads with more than
one
% schedules
SMG=S_SMG;


for i=1:length(loads_data{SMG}(:,1))
    aux_nom_power_loads(i,:) =
interp1(time_points{SMG}{i}(:),nominal_power_points{SMG}{i}(:),time,'prev
ious');
    aux_min_power_loads(i,:) =
interp1(time_points{SMG}{i}(:),minimum_power_points{SMG}{i}(:),time,'prev
ious');
end


aux_nom_power_loads(isnan(aux_nom_power_loads))=0;
aux_min_power_loads(isnan(aux_min_power_loads))=0;


k=0;
for i=(unique(loads_data{SMG}(:,1)))'
    k=k+1;
    positions=find(loads_data{SMG}(:,1) == i);
    nom_power_loads(k,:)=max(aux_nom_power_loads(positions));
    min_power_loads(k,:)=max(aux_min_power_loads(positions));
    % The names and the priority of the loads have to be also in the
outputs
    % for the desitions or the plots
    load_name{k,:}=loads_data_str{SMG}{positions(1)+1,1};

load_identifier(k,:)=max(aux_nom_power_loads(positions(1))>0).*loads_data
{SMG}(positions(1),1);

priority(k,:)=max(aux_nom_power_loads(positions(1))>0).*loads_data{SMG}(p
ositions(1),13);
```

```
end



else
    SMG=S_SMG;

    for i=1:length(loads_data{SMG}(:,1))
        aux_nom_power_loads(i,:) =
interp1(time_points{SMG}{i}(:),nominal_power_points{SMG}{i}(:),time,'prev
ious');
        aux_min_power_loads(i,:) =
interp1(time_points{SMG}{i}(:),minimum_power_points{SMG}{i}(:),time,'prev
ious');
    end



    aux_nom_power_loads(isnan(aux_nom_power_loads))=0;
    aux_min_power_loads(isnan(aux_min_power_loads))=0;



    k=0;
    for i=(unique(loads_data{SMG}(:,1)))'
        k=k+1;
        positions=find(loads_data{SMG}(:,1) == i);
        nom_power_loads(k,:)=max(aux_nom_power_loads(positions));
        min_power_loads(k,:)=max(aux_min_power_loads(positions));
        % The names and the priority of the loads have to be also in the
outputs
        % for the desitions or the plots
        load_name{k,:}=loads_data_str{SMG}{positions(1)+1,1};

load_identifier(k,:)=max(aux_nom_power_loads(positions(1))>0).*loads_data
{SMG}(positions(1),1);

priority(k,:)=max(aux_nom_power_loads(positions(1))>0).*loads_data{SMG}(p
ositions(1),13);
    end


end
```

1.2. “grid.m”

```
function [P_max,buying_price,selling_price]=grid(time,S_SMG)

persistent n_SMG max_P start_moment_fall duration_fall
if isempty(n_SMG)

    n_SMG=getappdata(0,'number_of_SMG');
```

```
    max_P=getappdata(0,'maximum_power_allowed_grid');
    start_moment_fall=getappdata(0,'when_grid_falls');
    duration_fall=getappdata(0,'grid_fall_duration');

    % Determine the power availability
    SMG=S_SMG;

    flag=0;
    for i=1:length(start_moment_fall{SMG})
        if (time>=start_moment_fall{SMG}(i) &&
time<=(start_moment_fall{SMG}(i)+duration_fall{SMG}(i))) && flag==0
            P_max=0;
            flag=1;
        elseif flag==0
            P_max=max_P(SMG);
            flag=0;
        end
    end

    % Prices
    aux_buying_price=grid_purchasing_price_function(time);
    aux_selling_price=grid_selling_price_function(time);
    buying_price=aux_buying_price(SMG);
    selling_price=aux_selling_price(SMG);

else

    SMG=S_SMG;
    % Determine the power availability
    flag=0;
    for i=1:length(start_moment_fall{SMG})
        if (time>=start_moment_fall{SMG}(i) &&
time<=(start_moment_fall{SMG}(i)+duration_fall{SMG}(i))) && flag==0
            P_max=0;
            flag=1;
        elseif flag==0
            P_max=max_P(SMG);
            flag=0;
        end
    end


    % Prices
    aux_buying_price=grid_purchasing_price_function(time);
    aux_selling_price=grid_selling_price_function(time);
    buying_price=aux_buying_price(SMG);
    selling_price=aux_selling_price(SMG);

end
```

1.3. "grid_purchasing_price_function.m"

```
function [buying_price]= grid_purchasing_price_function(time)
% Constants defined on the general program


persistent period_points buying_price_points initial_date final_date
n_SMG
if isempty(period_points)

initial_date=datenum(getappdata(0,'initial_date_external_data'),'dd/mm/yy
yy');
final_date=datenum(getappdata(0,'final_date_external_data'),'dd/mm/yyyy')
;
n_SMG=getappdata(0,'number_of_SMG');
flag=1;

for SMG = 1:n_SMG
    [buying_price_data{SMG}] = xlsread('grid_data.xlsx',num2str(SMG)); %
Imports the data of buying prices
end
% Data processing
initial_year=datestr(initial_date,'yyyy');
initial_month=datestr(initial_date,'mm');
initial_day=datestr(initial_date,'dd');

        % Splitting the data in month's representative days
for SMG = 1:n_SMG
    start_day{SMG}=find(not(isnan(buying_price_data{SMG}(:,4))));

start_day{SMG}=[start_day{SMG}(1:(length(start_day{SMG})));length(buying_
price_data{SMG}(:,1))+1];
end
    % Correcting the time from excel to matlab time for the first day of
every
    % month

        % Searching the datenum for each first month day of the data
month=initial_month;
for i=1:length(start_day{SMG})
    if i == 1
        day=num2str(str2double(initial_day));
    else
        day='01';
    end

first_month_day(i)=datenum(strcat(day,'/',month,'/',initial_year),'dd/mm/
yyyy');
    month=num2str(str2double(month)+1);
end

for SMG = 1:n_SMG
    for i=1:length(start_day{SMG})-1

day_points{1,i,SMG}=first_month_day(i)+buying_price_data{SMG}(start_day{S
MG}(i):(start_day{SMG}(i+1)-1),1);
```

```
    end
end

        % Repeating the day to make a month
for SMG = 1:n_SMG
    for i=1:length(start_day{SMG})-1
        unitary_day=day_points{1,i,SMG}-first_month_day(i);
        date=first_month_day(i)+1;
        while date<first_month_day(i+1)
            day_points{1,i,SMG}=[day_points{1,i,SMG}(1:end)
                                 date+unitary_day];
            date=date+1;
        end
    end
end
        % Creating the time vector

for SMG = 1:n_SMG
    period_points{SMG}=day_points{1,1,SMG}(1:end);
    for i=1:length(day_points)-1
        period_points{SMG}=[period_points{SMG}(1:end)
                           day_points{1,i+1,SMG}(1:end)];
    end
end

% Arrange the buying price points
    %Split the vector in month's representative days
for SMG = 1:n_SMG
    for i=1:length(start_day{SMG})-1

day_buying_price_points{1,i,SMG}=buying_price_data{SMG}(start_day{SMG}(i)
:(start_day{SMG}(i+1)-1),2);
    end
end

        % Repeating the day to make a month
for SMG = 1:n_SMG
    for i=1:length(start_day{SMG})-1
        unitary_buying_price_day=day_buying_price_points{1,i,SMG};
        date=first_month_day(i)+1;
        while date<first_month_day(i+1)

day_buying_price_points{1,i,SMG}=[day_buying_price_points{1,i,SMG}(1:end)
                                 unitary_buying_price_day];
            date=date+1;
        end
    end
end
        % Creating the buying price vector
for SMG = 1:n_SMG
    buying_price_points{SMG}=day_buying_price_points{1,1,SMG}(1:end);
    for i=1:length(day_points)-1
        buying_price_points{SMG}=[buying_price_points{SMG}(1:end)
                           day_buying_price_points{1,i+1,SMG}(1:end)];
    end
end
```

```
for SMG = 1:n_SMG
    buying_price(SMG) =
interp1(period_points{SMG},buying_price_points{SMG},time,'previous');
end

else

    for SMG = 1:n_SMG
    buying_price(SMG) =
interp1(period_points{SMG},buying_price_points{SMG},time,'previous');
    end
    flag=0;
end
```

1.4. "grid_selling_price_function.m"

```
function [selling_price]= grid_selling_price_function(time)
% Constants defined on the general program


persistent period_points selling_price_points initial_date final_date
n_SMG
if isempty(period_points)

initial_date=datenum(getappdata(0,'initial_date_external_data'),'dd/mm/yy
yy');
final_date=datenum(getappdata(0,'final_date_external_data'),'dd/mm/yyyy')
;
n_SMG=getappdata(0,'number_of_SMG');
flag=1;

for SMG = 1:n_SMG
    [selling_price_data{SMG}] = xlsread('grid_data.xlsx',num2str(SMG)); %
Imports the data of selling prices
end
% Data processing
initial_year=datestr(initial_date,'yyyy');
initial_month=datestr(initial_date,'mm');
initial_day=datestr(initial_date,'dd');

        % Splitting the data in month's representative days
for SMG = 1:n_SMG
    start_day{SMG}=find(not(isnan(selling_price_data{SMG}(:,4))));

start_day{SMG}=[start_day{SMG}(1:(length(start_day{SMG})));length(selling
_price_data{SMG}(:,1))+1];
end
    % Correcting the time from excel to matlab time for the first day of
every
    % month
```

```
        % Searching the datenum for each first month day of the data
month=initial_month;
for i=1:length(start_day{SMG})
    if i == 1
        day=num2str(str2double(initial_day));
    else
        day='01';
    end

first_month_day(i)=datenum(strcat(day,'/',month,'/',initial_year),'dd/mm/
yyyy');
    month=num2str(str2double(month)+1);
end

for SMG = 1:n_SMG
    for i=1:length(start_day{SMG})-1

day_points{1,i,SMG}=first_month_day(i)+selling_price_data{SMG}(start_day{
SMG}(i):(start_day{SMG}(i+1)-1),1);
    end
end

        % Repeating the day to make a month
for SMG = 1:n_SMG
    for i=1:length(start_day{SMG})-1
        unitary_day=day_points{1,i,SMG}-first_month_day(i);
        date=first_month_day(i)+1;
        while date<first_month_day(i+1)
            day_points{1,i,SMG}=[day_points{1,i,SMG}(1:end)
                                 date+unitary_day];
            date=date+1;
        end
    end
end
        % Creating the time vector

for SMG = 1:n_SMG
    period_points{SMG}=day_points{1,1,SMG}(1:end);
    for i=1:length(day_points)-1
        period_points{SMG}=[period_points{SMG}(1:end)
                           day_points{1,i+1,SMG}(1:end)];
    end
end

% Arrange the selling price points
    %Split the vector in month's representative days
for SMG = 1:n_SMG
    for i=1:length(start_day{SMG})-1

day_selling_price_points{1,i,SMG}=selling_price_data{SMG}(start_day{SMG}(
i):(start_day{SMG}(i+1)-1),3);
    end
end

        % Repeating the day to make a month
for SMG = 1:n_SMG
    for i=1:length(start_day{SMG})-1
```

```
        unitary_selling_price_day=day_selling_price_points{1,i,SMG};
        date=first_month_day(i)+1;
        while date<first_month_day(i+1)

day_selling_price_points{1,i,SMG}=[day_selling_price_points{1,i,SMG}(1:en
d)
                             unitary_selling_price_day];
            date=date+1;
        end
    end
end
        % Creating the selling price vector
for SMG = 1:n_SMG
    selling_price_points{SMG}=day_selling_price_points{1,1,SMG}(1:end);
    for i=1:length(day_points)-1
        selling_price_points{SMG}=[selling_price_points{SMG}(1:end)
                        day_selling_price_points{1,i+1,SMG}(1:end)];
    end
end


for SMG = 1:n_SMG
    selling_price(SMG) =
interp1(period_points{SMG},selling_price_points{SMG},time,'previous');
end

else

    for SMG = 1:n_SMG
    selling_price(SMG) =
interp1(period_points{SMG},selling_price_points{SMG},time,'previous');
    end
    flag=0;
end
```

1.5. “irradiance_function.m”

```
function [irradiance_value]= irradiance_function(time)
% Constants defined on the general program


persistent period_points irradiance_points initial_date final_date n_SMG
if isempty(period_points)

initial_date=datenum(getappdata(0,'initial_date_external_data'),'dd/mm/yy
yy');
final_date=datenum(getappdata(0,'final_date_external_data'),'dd/mm/yyyy')
;
n_SMG=getappdata(0,'number_of_SMG');
flag=1;

for SMG = 1:n_SMG
```

```
    [irradiance_data{SMG}] =
xlsread('environmental_data.xlsx',num2str(SMG)); % Imports the data of
irradiance
end
% Data processing
initial_year=datestr(initial_date,'yyyy');
initial_month=datestr(initial_date,'mm');
initial_day=datestr(initial_date,'dd');

        % Splitting the data in month's representative days
for SMG = 1:n_SMG
    start_day{SMG}=find(not(isnan(irradiance_data{SMG}(:,4))));

start_day{SMG}=[start_day{SMG}(1:(length(start_day{SMG})));length(irradia
nce_data{SMG}(:,1))+1];
end
    % Correcting the time from excel to matlab time for the first day of
every
    % month

        % Searching the datenum for each first month day of the data
month=initial_month;
for i=1:length(start_day{SMG})
    if i == 1
        day=num2str(str2double(initial_day));
    else
        day='01';
    end

first_month_day(i)=datenum(strcat(day,'/',month,'/',initial_year),'dd/mm/
yyyy');
    month=num2str(str2double(month)+1);
end

for SMG = 1:n_SMG
    for i=1:length(start_day{SMG})-1

day_points{1,i,SMG}=first_month_day(i)+irradiance_data{SMG}(start_day{SMG
}(i):(start_day{SMG}(i+1)-1),1);
    end
end
        % Adding the points where irradiance is zero
time_step=day_points{1,1,SMG}(2)-day_points{1,1,SMG}(1);

for SMG = 1:n_SMG
    for i=1:length(start_day{SMG})-1
        day_points{1,i,SMG}=[day_points{1,i,SMG}(1)-time_step
                             day_points{1,i,SMG}(1:end)
                             day_points{1,i,SMG}(end)+time_step]; %previous 0
point
    end
end

        % Repeating the day to make a month
for SMG = 1:n_SMG
    for i=1:length(start_day{SMG})-1
        unitary_day=day_points{1,i,SMG}-first_month_day(i);
```

```
        date=first_month_day(i)+1;
        while date<first_month_day(i+1)
            day_points{1,i,SMG}=[day_points{1,i,SMG}(1:end)
                                 date+unitary_day];
            date=date+1;
        end
    end
end
        % Creating the time vector

for SMG = 1:n_SMG
    period_points{SMG}=day_points{1,1,SMG}(1:end);
    for i=1:length(day_points)-1
        period_points{SMG}=[period_points{SMG}(1:end)
                           day_points{1,i+1,SMG}(1:end)];
    end
end
        %Adding the initial and final point
for SMG = 1:n_SMG
    period_points{SMG}=[initial_date
                   period_points{SMG}(1:end)
                   final_date];
end

% Arrange the irradiance points
    %Split the vector in month's representative days
for SMG = 1:n_SMG
    for i=1:length(start_day{SMG})-1

day_irradiance_points{1,i,SMG}=irradiance_data{SMG}(start_day{SMG}(i):(st
art_day{SMG}(i+1)-1),2);
    end
end

    % Adding the points where irradiance is zero
for SMG = 1:n_SMG
    for i=1:length(start_day{SMG})-1
        day_irradiance_points{1,i,SMG}=[0
                                 day_irradiance_points{1,i,SMG}(1:end)
                                 0]; %previous 0 point
    end
end
        % Repeating the day to make a month
for SMG = 1:n_SMG
    for i=1:length(start_day{SMG})-1
        random_value(i,:,SMG)=1 + 0.3*rand(1,30);
        unitary_irradiance_day=day_irradiance_points{1,i,SMG};
        date=first_month_day(i)+1;
        k_random=1;
        while date<first_month_day(i+1)

day_irradiance_points{1,i,SMG}=[day_irradiance_points{1,i,SMG}(1:end)

unitary_irradiance_day.*random_value(i,k_random,1)];
            date=date+1;
            k_random=1+k_random;
        end
```

```
    end
end
        % Creating the irradiance vector
for SMG = 1:n_SMG
    irradiance_points{SMG}=day_irradiance_points{1,1,SMG}(1:end);
    for i=1:length(day_points)-1
        irradiance_points{SMG}=[irradiance_points{SMG}(1:end)
                               day_irradiance_points{1,i+1,SMG}(1:end)];
    end
end
        %Adding the initial and final point
for SMG = 1:n_SMG
    irradiance_points{SMG}=[0
                           irradiance_points{SMG}(1:end)
                           0];
end

for SMG = 1:n_SMG
    irradiance_value(SMG) =
interp1(period_points{SMG},irradiance_points{SMG},time);
end

else

    for SMG = 1:n_SMG
    irradiance_value(SMG) =
interp1(period_points{SMG},irradiance_points{SMG},time);
    end
    flag=0;
end
```

1.6. “irradiance_forecast_function.m”

```
function [irradiance_value]= irradiance_forecast_function(time)
% Constants defined on the general program


persistent period_points irradiance_points initial_date final_date n_SMG
if isempty(period_points)

initial_date=datenum(getappdata(0,'initial_date_external_data'),'dd/mm/yy
yy');
final_date=datenum(getappdata(0,'final_date_external_data'),'dd/mm/yyyy')
;
n_SMG=getappdata(0,'number_of_SMG');
flag=1;

for SMG = 1:n_SMG
    [irradiance_data{SMG}] =
xlsread('environmental_data.xlsx',num2str(SMG)); % Imports the data of
irradiance
end
```

```
% Data processing
initial_year=datestr(initial_date,'yyyy');
initial_month=datestr(initial_date,'mm');
initial_day=datestr(initial_date,'dd');

        % Splitting the data in month's representative days
for SMG = 1:n_SMG
    start_day{SMG}=find(not(isnan(irradiance_data{SMG}(:,4))));

start_day{SMG}=[start_day{SMG}(1:(length(start_day{SMG})));length(irradia
nce_data{SMG}(:,1))+1];
end
    % Correcting the time from excel to matlab time for the first day of
every
    % month

        % Searching the datenum for each first month day of the data
month=initial_month;
for i=1:length(start_day{SMG})
    if i == 1
        day=num2str(str2double(initial_day));
    else
        day='01';
    end

first_month_day(i)=datenum(strcat(day,'/',month,'/',initial_year),'dd/mm/
yyyy');
    month=num2str(str2double(month)+1);
end

for SMG = 1:n_SMG
    for i=1:length(start_day{SMG})-1

day_points{1,i,SMG}=first_month_day(i)+irradiance_data{SMG}(start_day{SMG
}(i):(start_day{SMG}(i+1)-1),1);
    end
end
        % Adding the points where irradiance is zero
time_step=day_points{1,1,SMG}(2)-day_points{1,1,SMG}(1);

for SMG = 1:n_SMG
    for i=1:length(start_day{SMG})-1
        day_points{1,i,SMG}=[day_points{1,i,SMG}(1)-time_step
                             day_points{1,i,SMG}(1:end)
                             day_points{1,i,SMG}(end)+time_step]; %previous 0
point
    end
end

        % Repeating the day to make a month
for SMG = 1:n_SMG
    for i=1:length(start_day{SMG})-1
        unitary_day=day_points{1,i,SMG}-first_month_day(i);
        date=first_month_day(i)+1;
        while date<first_month_day(i+1)
            day_points{1,i,SMG}=[day_points{1,i,SMG}(1:end)
                                 date+unitary_day];
```

```
            date=date+1;
        end
    end
end
        % Creating the time vector

for SMG = 1:n_SMG
    period_points{SMG}=day_points{1,1,SMG}(1:end);
    for i=1:length(day_points)-1
        period_points{SMG}=[period_points{SMG}(1:end)
                       day_points{1,i+1,SMG}(1:end)];
    end
end
        %Adding the initial and final point
for SMG = 1:n_SMG
    period_points{SMG}=[initial_date
                   period_points{SMG}(1:end)
                   final_date];
end

% Arrange the irradiance points
    %Split the vector in month's representative days
for SMG = 1:n_SMG
    for i=1:length(start_day{SMG})-1

day_irradiance_points{1,i,SMG}=irradiance_data{SMG}(start_day{SMG}(i):(st
art_day{SMG}(i+1)-1),2);
    end
end

    % Adding the points where irradiance is zero
for SMG = 1:n_SMG
    for i=1:length(start_day{SMG})-1
        day_irradiance_points{1,i,SMG}=[0
                         day_irradiance_points{1,i,SMG}(1:end)
                         0]; %previous 0 point
    end
end
        % Repeating the day to make a month
for SMG = 1:n_SMG
    for i=1:length(start_day{SMG})-1
        random_value=ones(1,30);%1 + 0.3*rand(1,30);
        unitary_irradiance_day=day_irradiance_points{1,i,SMG};
        date=first_month_day(i)+1;
        k_random=1;
        while date<first_month_day(i+1)

day_irradiance_points{1,i,SMG}=[day_irradiance_points{1,i,SMG}(1:end)

unitary_irradiance_day.*random_value(k_random)];
            date=date+1;
            k_random=1+k_random;
        end
    end
end
        % Creating the irradiance vector
for SMG = 1:n_SMG
```

```
    irradiance_points{SMG}=day_irradiance_points{1,1,SMG}(1:end);
    for i=1:length(day_points)-1
        irradiance_points{SMG}=[irradiance_points{SMG}(1:end)
                               day_irradiance_points{1,i+1,SMG}(1:end)];
    end
end
        %Adding the initial and final point
for SMG = 1:n_SMG
    irradiance_points{SMG}=[0
                           irradiance_points{SMG}(1:end)
                           0];
end

for SMG = 1:n_SMG
    irradiance_value(SMG) =
interp1(period_points{SMG},irradiance_points{SMG},time);
end

else

    for SMG = 1:n_SMG
    irradiance_value(SMG) =
interp1(period_points{SMG},irradiance_points{SMG},time);
    end
    flag=0;
end
```

1.7. “temperature_function.m”

```
function [temperature_value]= temperature_function(time)
% Constants defined on the general program


persistent period_points temperature_points initial_date final_date n_SMG
if isempty(period_points)

flag=1;
initial_date=datenum(getappdata(0,'initial_date_external_data'),'dd/mm/yy
yy');
final_date=datenum(getappdata(0,'final_date_external_data'),'dd/mm/yyyy')
;
n_SMG=getappdata(0,'number_of_SMG');

for SMG = 1:n_SMG
    [temperature_data{SMG}] =
xlsread('environmental_data.xlsx',num2str(SMG)); % Imports the data of
irradiation
end
% Data processing
initial_year=datestr(initial_date,'yyyy');
initial_month=datestr(initial_date,'mm');
initial_day=datestr(initial_date,'dd');
```

```
        % Splitting the data in month's representative days
for SMG = 1:n_SMG
    start_day{SMG}=find(not(isnan(temperature_data{SMG}(:,4))));

start_day{SMG}=[start_day{SMG}(1:(length(start_day{SMG})));length(tempera
ture_data{SMG}(:,1))+1];
end
    % Correcting the time from excel to matlab time for the first day of
every
    % month

        % Searching the datenum for each first month day of the data
month=initial_month;
for i=1:length(start_day{SMG})
    if i == 1
        day=num2str(str2double(initial_day));
    else
        day='01';
    end

first_month_day(i)=datenum(strcat(day,'/',month,'/',initial_year),'dd/mm/
yyyy');
    month=num2str(str2double(month)+1);
end

for SMG = 1:n_SMG
    for i=1:length(start_day{SMG})-1

day_points{1,i,SMG}=first_month_day(i)+temperature_data{SMG}(start_day{SM
G}(i):(start_day{SMG}(i+1)-1),1);
    end
end
        % Adding the points where irradiance is zero



        % Repeating the day to make a month
for SMG = 1:n_SMG
    for i=1:length(start_day{SMG})-1
        unitary_day=day_points{1,i,SMG}-first_month_day(i);
        date=first_month_day(i)+1;
        while date<first_month_day(i+1)
            day_points{1,i,SMG}=[day_points{1,i,SMG}(1:end)
                                 date+unitary_day];
            date=date+1;
        end
    end
end
        % Creating the time vector

for SMG = 1:n_SMG
    period_points{SMG}=day_points{1,1,SMG}(1:end);
    for i=1:length(day_points)-1
        period_points{SMG}=[period_points{SMG}(1:end)
                           day_points{1,i+1,SMG}(1:end)];
    end
```

```
end
        %Adding the initial and final point
%for SMG = 1:n_SMG
    %period_points{SMG}=[initial_date
                   %period_points{SMG}(1:end)
                   %final_date];
%end

% Arrange the temperature points
    %Split the vector in month's representative days
for SMG = 1:n_SMG
    for i=1:length(start_day{SMG})-1

day_temperature_points{1,i,SMG}=temperature_data{SMG}(start_day{SMG}(i):(
start_day{SMG}(i+1)-1),3);
    end
end

    % Adding the points where irradiance is zero

        % Repeating the day to make a month
for SMG = 1:n_SMG
    for i=1:length(start_day{SMG})-1
        random_value=1 + 0.3*rand(1,30);
        unitary_irradiance_day=day_temperature_points{1,i,SMG};
        date=first_month_day(i)+1;
        k_random=1;
        while date<first_month_day(i+1)

day_temperature_points{1,i,SMG}=[day_temperature_points{1,i,SMG}(1:end)

unitary_irradiance_day.*random_value(k_random)];
            date=date+1;
            k_random=1+k_random;
        end
    end
end
        % Creating the irradiation vector
for SMG = 1:n_SMG
    temperature_points{SMG}=day_temperature_points{1,1,SMG}(1:end);
    for i=1:length(day_points)-1
        temperature_points{SMG}=[temperature_points{SMG}(1:end)
                               day_temperature_points{1,i+1,SMG}(1:end)];
    end
end
        %Adding the initial and final point
%for SMG = 1:n_SMG
    %temperature_points{SMG}=[0
                   %temperature_points{SMG}(1:end)
                   %0];
%end

for SMG = 1:n_SMG
    temperature_value(SMG) =
interp1(period_points{SMG},temperature_points{SMG},time,'pchip');
end
```

```
else

    for SMG = 1:n_SMG
    temperature_value(SMG) =
interp1(period_points{SMG},temperature_points{SMG},time,'pchip');
    end
    flag=0;
end
```

1.8. "temperature_forecast_function.m"

```
function [temperature_value]= temperature_forecast_function(time)
% Constants defined on the general program


persistent period_points temperature_points initial_date final_date n_SMG
if isempty(period_points)

flag=1;
initial_date=datenum(getappdata(0,'initial_date_external_data'),'dd/mm/yy
yy');
final_date=datenum(getappdata(0,'final_date_external_data'),'dd/mm/yyyy')
;
n_SMG=getappdata(0,'number_of_SMG');

for SMG = 1:n_SMG
    [temperature_data{SMG}] =
xlsread('environmental_data.xlsx',num2str(SMG)); % Imports the data of
irradiation
end
% Data processing
initial_year=datestr(initial_date,'yyyy');
initial_month=datestr(initial_date,'mm');
initial_day=datestr(initial_date,'dd');

        % Splitting the data in month's representative days
for SMG = 1:n_SMG
    start_day{SMG}=find(not(isnan(temperature_data{SMG}(:,4))));

start_day{SMG}=[start_day{SMG}(1:(length(start_day{SMG})));length(tempera
ture_data{SMG}(:,1))+1];
end
    % Correcting the time from excel to matlab time for the first day of
every
    % month

        % Searching the datenum for each first month day of the data
month=initial_month;
for i=1:length(start_day{SMG})
    if i == 1
        day=num2str(str2double(initial_day));
    else
```

```
        day='01';
    end

first_month_day(i)=datenum(strcat(day,'/',month,'/',initial_year),'dd/mm/
yyyy');
    month=num2str(str2double(month)+1);
end

for SMG = 1:n_SMG
    for i=1:length(start_day{SMG})-1

day_points{1,i,SMG}=first_month_day(i)+temperature_data{SMG}(start_day{SM
G}(i):(start_day{SMG}(i+1)-1),1);
    end
end
        % Adding the points where irradiance is zero


        % Repeating the day to make a month
for SMG = 1:n_SMG
    for i=1:length(start_day{SMG})-1
        unitary_day=day_points{1,i,SMG}-first_month_day(i);
        date=first_month_day(i)+1;
        while date<first_month_day(i+1)
            day_points{1,i,SMG}=[day_points{1,i,SMG}(1:end)
                                 date+unitary_day];
            date=date+1;
        end
    end
end
        % Creating the time vector

for SMG = 1:n_SMG
    period_points{SMG}=day_points{1,1,SMG}(1:end);
    for i=1:length(day_points)-1
        period_points{SMG}=[period_points{SMG}(1:end)
                           day_points{1,i+1,SMG}(1:end)];
    end
end
        %Adding the initial and final point
%for SMG = 1:n_SMG
    %period_points{SMG}=[initial_date
                        %period_points{SMG}(1:end)
                        %final_date];
%end

% Arrange the temperature points
    %Split the vector in month's representative days
for SMG = 1:n_SMG
    for i=1:length(start_day{SMG})-1

day_temperature_points{1,i,SMG}=temperature_data{SMG}(start_day{SMG}(i):(
start_day{SMG}(i+1)-1),3);
    end
end
```

```
    % Adding the points where irradiance is zero

        % Repeating the day to make a month
for SMG = 1:n_SMG
    for i=1:length(start_day{SMG})-1
        random_value=ones(1,30);%1 + 0.3*rand(1,30);
        unitary_irradiance_day=day_temperature_points{1,i,SMG};
        date=first_month_day(i)+1;
        k_random=1;
        while date<first_month_day(i+1)

day_temperature_points{1,i,SMG}=[day_temperature_points{1,i,SMG}(1:end)

unitary_irradiance_day.*random_value(k_random)];
            date=date+1;
            k_random=1+k_random;
        end
    end
end
        % Creating the irradiation vector
for SMG = 1:n_SMG
    temperature_points{SMG}=day_temperature_points{1,1,SMG}(1:end);
    for i=1:length(day_points)-1
        temperature_points{SMG}=[temperature_points{SMG}(1:end)
                               day_temperature_points{1,i+1,SMG}(1:end)];
    end
end
        %Adding the initial and final point
%for SMG = 1:n_SMG
    %temperature_points{SMG}=[0
                   %temperature_points{SMG}(1:end)
                   %0];
%end

for SMG = 1:n_SMG
    temperature_value(SMG) =
interp1(period_points{SMG},temperature_points{SMG},time,'pchip');
end

else

    for SMG = 1:n_SMG
    temperature_value(SMG) =
interp1(period_points{SMG},temperature_points{SMG},time,'pchip');
    end
    flag=0;
end
```

1.9. “PV.m”

```
function [V1,I,P]= PV(time,S_SMG)
```

```
persistent n_SMG N_s_i n_s_i N_p_i NOCT A K_i K_v V_oc_i V_mpp I_mpp n_s
N_p E_g k q G_stc T_ref I_sc V_oc
if isempty(n_SMG)
% Parameters
n_SMG=getappdata(0,'number_of_SMG');
N_s_i=getappdata(0,'series_connected_panels');
n_s_i=getappdata(0,'series_connected_cells');
N_p_i=getappdata(0,'parallel_connected_panels');
NOCT=getappdata(0,'NOCT');
A=getappdata(0,'ideality_factor');
K_i=getappdata(0,'temp_coeficient_current');
K_v=getappdata(0,'temp_coeficient_voltage');
I_sc=getappdata(0,'Shortcut_current');
V_oc_i=getappdata(0,'Open_circuit_voltage');
V_mpp=getappdata(0,'MPP_voltage');
I_mpp=getappdata(0,'MPP_current');
n_s=n_s_i.*N_s_i; %Number of panels in series
N_p=N_p_i; %Number of panels in paralel
E_g=1.12; %
k=1.38065e-23; %Boltzman constant
q=1.602e-19; %Electron charge
G_stc=1000;
T_ref=273+25; %Temperature in standard conditions
V_oc=V_oc_i.*N_s_i;

% Variables
T_a = temperature_function(time);
G = irradiance_function(time);



SMG=S_SMG;

    % Intermediate equations
    T_c=(T_a(SMG)+(NOCT(SMG)-20)/800*G(SMG))+273; %Temperature of the
cell
    V_tn=n_s(SMG)*(k*T_ref/q);
    I_on=I_sc(SMG)/((exp(V_oc(SMG)/(A(SMG)*V_tn)))-1);
    I_o=I_on*((T_c/T_ref).^3)*exp((((q*E_g/(A(SMG)*k))*((1/T_ref)-
(1/T_c))));
    I_pv=(I_sc(SMG)+K_i(SMG)*(T_c-T_ref))*(G(SMG)/G_stc);
    R_s=(A(SMG)*V_tn*log(1-
I_mpp(SMG)*N_p_i(SMG)/(I_sc(SMG)*N_p_i(SMG)))+V_oc(SMG)-
V_mpp(SMG)*N_s_i(SMG))/(I_mpp(SMG)*N_p_i(SMG));

R_p=(V_mpp(SMG)*N_s_i(SMG)+I_mpp(SMG)*N_p_i(SMG)*R_s)/(I_pv*N_p_i(SMG)-
I_mpp(SMG)*N_p_i(SMG)-
I_on*N_p_i(SMG)*(exp((V_mpp(SMG)*N_s_i(SMG)+I_mpp(SMG)*N_p_i(SMG)*R_s)/(A
(SMG)*V_tn))-1))*n_s(SMG)*N_s_i(SMG);
    V_t=n_s(SMG)*(k*T_c/q);

    I=zeros(1,(round(V_oc(SMG)*1.1)+1)*10);
    V1=zeros(1,(round(V_oc(SMG)*1.1)+1)*10);
    P=zeros(1,(round(V_oc(SMG)*1.1)+1)*10);
    I(1)=I_sc(SMG)*N_p(SMG);
    i=1;
    V=0;
```

```
    for i = 1:(round(V_oc(SMG)*1.1)+1)*10
        I_part=N_p(SMG)*I_o*(exp((V+(I(i)*R_s/N_p(SMG)))/(V_t*A(SMG)))-
1)+((V*N_p(SMG)+(R_s*I(i)))/R_p);
        I(i+1)=N_p(SMG)*I_pv-I_part;
        if I(i+1)<0
            I(i+1)=0;
        end
        V1(i)=V;
        P(i)=V*I(i);
        V=V+0.1;
    end
    i=i+1;
    V1(i)=V1(i-1);
    P(i)=P(i-1);

else
% Variables
T_a = temperature_function(time);
G = irradiance_function(time);

SMG=S_SMG;

    % Intermediate equations
    T_c=(T_a(SMG)+(NOCT(SMG)-20)/800*G(SMG))+273; %Temperature of the
cell
    V_tn=n_s(SMG)*(k*T_ref/q);
    I_on=I_sc(SMG)/((exp(V_oc(SMG)/(A(SMG)*V_tn)))-1);
    I_o=I_on*((T_c/T_ref).^3)*exp(((q*E_g/(A(SMG)*k))*((1/T_ref)-
(1/T_c))));
    I_pv=(I_sc(SMG)+K_i(SMG)*(T_c-T_ref))*(G(SMG)/G_stc);
    R_s=(A(SMG)*V_tn*log(1-
I_mpp(SMG)*N_p_i(SMG)/(I_sc(SMG)*N_p_i(SMG)))+V_oc(SMG)-
V_mpp(SMG)*N_s_i(SMG))/(I_mpp(SMG)*N_p_i(SMG));

R_p=(V_mpp(SMG)*N_s_i(SMG)+I_mpp(SMG)*N_p_i(SMG)*R_s)/(I_pv*N_p_i(SMG)-
I_mpp(SMG)*N_p_i(SMG)-
I_on*N_p_i(SMG)*(exp((V_mpp(SMG)*N_s_i(SMG)+I_mpp(SMG)*N_p_i(SMG)*R_s)/(A
(SMG)*V_tn))-1))*n_s(SMG)*N_s_i(SMG);
    V_t=n_s(SMG)*(k*T_c/q);

    I=zeros(1,(round(V_oc(SMG)*1.1)+1)*10);
    V1=zeros(1,(round(V_oc(SMG)*1.1)+1)*10);
    P=zeros(1,(round(V_oc(SMG)*1.1)+1)*10);
    I(1)=I_sc(SMG)*N_p(SMG);
    i=1;
    V=0;

    for i = 1:(round(V_oc(SMG)*1.1)+1)*10
        I_part=N_p(SMG)*I_o*(exp((V+(I(i)*R_s/N_p(SMG)))/(V_t*A(SMG)))-
1)+((V*N_p(SMG)+(R_s*I(i)))/R_p);
        I(i+1)=N_p(SMG)*I_pv-I_part;
        if I(i+1)<0
            I(i+1)=0;
        end
        V1(i)=V;
```

```
        P(i)=V*I(i);
        V=V+0.1;
    end
    i=i+1;
    V1(i)=V1(i-1);
    P(i)=P(i-1);
end
```

1.10. "PV_inverter.m"

```
function [P_max]= PV_inverter(time,S_SMG)
% [P_max,P_inst]= PV_inverter(time,step,commands)
% The time variable is the current instant.
% The step variable must be 1 when only requesting the possibility of
% giving energy in the current moment
% The commands variable is the demanded active power output


persistent n_SMG inverter_efficiency_curve P_nom
if isempty(n_SMG)
    SMG=S_SMG;
    % Parameters
    n_SMG=getappdata(0,'number_of_SMG');

inverter_efficiency_curve=getappdata(0,'pv_inverter_efficiency_curve');
%(1,:)Efficiency vector in %, (2,:)Output/nominal power vector in %
    P_nom=getappdata(0,'nominal_power_inverter');

    % Variables
    [V1,I,P]=PV(time,SMG);

    % Functionality code
    aux_P_max=(max(P)/1000);

    if aux_P_max<0.020
        aux_P_max=0;
    end


inverter_efficiency=interp1(inverter_efficiency_curve{SMG}(2,:),inverter_
efficiency_curve{SMG}(1,:),100*aux_P_max/P_nom(SMG))/100;
    P_max=aux_P_max*inverter_efficiency;
    if isnan(P_max)
        P_max=0;
    end


else
    SMG=S_SMG;
    % Parameters

    % Variables
```

```
    [V1,I,P]=PV(time,SMG);

    % Functionality code

    aux_P_max=(max(P)/1000);

    if aux_P_max<0.005
        aux_P_max=0;
    end


inverter_efficiency=interp1(inverter_efficiency_curve{SMG}(2,:),inverter_
efficiency_curve{SMG}(1,:),100*aux_P_max/P_nom(SMG))/100;
    P_max=aux_P_max*inverter_efficiency;
    if isnan(P_max)
        P_max=0;
    end

end
```

1.11. “battery.m”

```
function
[power_vector,autonomy_vector,SoC]=battery(time,S_SMG,step,command)
% This is a simplified model of the battery

persistent n_SMG time_steps max_power initial_SoC efficiency_curve p_SoC
...
    battery_voltage rated_capacity bulk_power charger_efficiency

if isempty(n_SMG)
    n_SMG=getappdata(0,'number_of_SMG');
    time_steps=getappdata(0,'time_steps');
    max_power=getappdata(0,'battery_max_power'); %Demanded in kW
    bulk_power=getappdata(0,'bulk_power'); %Demanded in kW
    efficiency_curve=getappdata(0,'battery_efficiency_curve');
%(1)Temperature matrix in °C, (2)SoC matrix in % (3) Efficiency matrix in
%
    battery_voltage=getappdata(0,'battery_voltage');
    rated_capacity=getappdata(0,'battery_rated_capacity');
    initial_SoC=getappdata(0,'initial_SoC');
    p_SoC=initial_SoC./100;
    charger_efficiency=getappdata(0,'charger_efficiency')./100;
    SMG=S_SMG;

    if nargin == 2
        step = 1;
    end

    if step==1
        %Determining the power it can absorb in the specific SoC
        if p_SoC(SMG)<0.8
            max_charge_power=-bulk_power(SMG);
        else
```

```
            max_charge_power=-
interp1([0.8,1],[bulk_power(SMG),0],p_SoC(SMG));
        end


power_vector=[max_power(SMG),max_power(SMG)*2/3,max_power(SMG)/3,max_char
ge_power/3,max_charge_power*2/3,max_charge_power];

autonomy_vector=((p_SoC(SMG)*rated_capacity(SMG)*battery_voltage(SMG))/10
00)./(power_vector([1,2,3])./0.95);
        SoC=p_SoC(SMG)*100;

    elseif step==2

        power_vector=command;
        if command>0
            command=(power_vector/0.95/charger_efficiency(SMG));

autonomy_vector=((p_SoC(SMG)*rated_capacity(SMG)*battery_voltage(SMG))/10
00)./command;
        else
            efficiency =
interp2(efficiency_curve{SMG}(:,:,1),efficiency_curve{SMG}(:,:,2),efficie
ncy_curve{SMG}(:,:,3),temperature_function(time),p_SoC(SMG)*100,'linear',
95)/100;
            autonomy_vector=inf;
            command=command*efficiency(SMG)*charger_efficiency(SMG);
        end
        SoC=p_SoC(SMG)*100;

p_SoC(SMG)=(((p_SoC(SMG)*rated_capacity(SMG)*battery_voltage(SMG))/1000)-
command*time_steps*24)/((rated_capacity(SMG)*battery_voltage(SMG))/1000);
    end
else
    SMG=S_SMG;

    if nargin == 2
        step = 1;
    end

    if step==1
        %Determining the power it can absorb in the specific SoC
        if p_SoC(SMG)<0.8
            max_charge_power=-bulk_power(SMG);
        else
            max_charge_power=-
interp1([0.8,1],[bulk_power(SMG),0],p_SoC(SMG));
        end


power_vector=[max_power(SMG),max_power(SMG)*2/3,max_power(SMG)/3,max_char
ge_power/3,max_charge_power*2/3,max_charge_power];

autonomy_vector=((p_SoC(SMG)*rated_capacity(SMG)*battery_voltage(SMG))/10
00)./(power_vector([1,2,3])./0.95);
        SoC=p_SoC(SMG)*100;
```

```
    elseif step==2

        power_vector=command;
        if command>0
            command=(power_vector/0.95/charger_efficiency(SMG));

autonomy_vector=((p_SoC(SMG)*rated_capacity(SMG)*battery_voltage(SMG))/10
00)./command;
        else
            efficiency=
interp2(efficiency_curve{SMG}(:,:,1),efficiency_curve{SMG}(:,:,2),efficie
ncy_curve{SMG}(:,:,3),temperature_function(time),p_SoC(SMG)*100,'linear',
95)/100;
            autonomy_vector=NaN;
            command=command*efficiency(SMG)*charger_efficiency(SMG);
        end
        SoC=p_SoC(SMG)*100;

p_SoC(SMG)=(((p_SoC(SMG)*rated_capacity(SMG)*battery_voltage(SMG))/1000)-
(command*time_steps*24))/((rated_capacity(SMG)*battery_voltage(SMG))/1000
);
    end
end
```

1.12. “genset.m”

```
function
[power_vector,emission_vector,cost_vector,autonomy_vector,state,fuel_leve
l]=genset(time,S_SMG,step,command)
%
[power_vector,emission_vector,cost_vector,autonomy_vector,state,fuel_leve
l]=genset(time,step,state_change,power_command)
%OUTPUTS
% The power vector is a vector of 3 points (why 3? to interpolate in the
EMS algorithm)
% where the 1st one is the maximum reachable power by the genset from the
current
% state, the 3rd point is the minimum reachable power and the 2nd is the
medium. In kW.
% The emission vector is similar to the power vector but in kg of CO2
% emitted by the correspondant power values in the specified time step
% period.
% The cost vector is the same as the emission vector but in money. Takes
% into acount the variable cost
% The autonomy vector is the time in hours the genset will be able to
work in each
% power.
% The state is 0 when stopped, 1 is turned on but not connected to the
% grid, 2 when connected to the grid and giving power, 3 when
disconnected
% and in process to stop.
% The fuel level gives the % of the fuel against the maximum.
%INPUTS
% Time is the variable time in the simulator
% S_SMG is the Smart Micro Grid of the genset
```

```
% The step sets the objective of the function calling, 1 to know the
output
% variables but without making any change on the genset variables
(normally
% in the fuel_level,autonomy...), 2 to make vinculant changes on the
genset
% setting the demanded power, 3 is to make vinculant changes on the
% genset as change the genset state (turn it on or off)
% 4 is to turn on an alarm to refill the fuel tank (the
% genset will automatically refill it after the specified days).
% The variable command is a variable that can be used only when step=2 or
3, and can
% take the values of power demanded in kW always between the values
previously given
% on the power_vecto and when step=3 can take the values 1 or 3 to turn
the
% genset on(1), turn it off (and disconnect it) (3)

    persistent n_SMG time_steps start_time stop_time ramping_increase
ramping_decrease ...
        max_power initial_fuel fuel_capacity fuel_cost fuel_emission
fuel_density efficiency_curve ...
        refill_period c_state p_command p_fuel_level time_reference_1
time_reference_3 time_reference_refilling

    if isempty(n_SMG)
        n_SMG=getappdata(0,'number_of_SMG');
        time_steps=getappdata(0,'time_steps');
        start_time=getappdata(0,'time_reach_rpm');
        stop_time=getappdata(0,'time_reach_0rpm');
        ramping_increase=getappdata(0,'ramping_power_increase');
%kW/minute
        ramping_decrease=getappdata(0,'ramping_power_decrease');
%kW/minute
        max_power=getappdata(0,'genset_max_power');
        initial_fuel=getappdata(0,'initial_fuel');
        fuel_capacity=getappdata(0,'fuel_capacity');
        fuel_cost=getappdata(0,'fuel_cost'); % €/litter
        fuel_emission=getappdata(0,'fuel_emission'); %kgCO2/litter
        fuel_density=getappdata(0,'fuel_density'); %kWh/litter
        efficiency_curve=getappdata(0,'genset_efficiency_curve');
%(1)Efficiency vetor in %, (2)Output power vector
        refill_period=getappdata(0,'refilling_periods');
        for SMG=1:n_SMG
        c_state(SMG)=0;
        p_command(SMG)=NaN;
        time_reference_1(SMG)=NaN;
        time_reference_3(SMG)=NaN;
        time_reference_refilling(SMG)=NaN;
        end
        p_fuel_level=initial_fuel./fuel_capacity;

        SMG=S_SMG;

        if nargin == 2
            step = 1;
        end
```

```
        if step==1
            if c_state(SMG)==0
                power_vector=[0,0,0];
                emission_vector=[0,0,0];
                cost_vector=[0,0,0];
                autonomy_vector=[NaN,NaN,NaN];
                state=0;
                fuel_level=p_fuel_level(SMG)*100;

            elseif c_state(SMG)==1
                power_vector=[0,0,0];
                emission_vector=[0,0,0];
                cost_vector=[0,0,0];
                autonomy_vector=[NaN,NaN,NaN];
                state=1;
                fuel_level=p_fuel_level(SMG)*100;

            elseif c_state(SMG)==2
                if isnan(p_command(SMG))
                    power_vector=[max_power(SMG),(max_power(SMG)-
max_power(SMG)*0.1)/2+(max_power(SMG)*0.1),max_power(SMG)*0.1];

emission_vector=[(fuel_emission(SMG)/fuel_density(SMG)).*power_vector./(i
nterp1(efficiency_curve{SMG}(2,:),efficiency_curve{SMG}(1,:),power_vector
)./100).*(time_steps*24)];

cost_vector=[(fuel_cost(SMG)/fuel_density(SMG)).*power_vector./(interp1(e
fficiency_curve{SMG}(2,:),efficiency_curve{SMG}(1,:),power_vector)./100).
*(time_steps*24)];
                    autonomy_vector=[
(p_fuel_level(SMG)*fuel_capacity(SMG)) ./
(power_vector./(interp1(efficiency_curve{SMG}(2,:),efficiency_curve{SMG}(
1,:),power_vector)./100)./fuel_density(SMG)) ];
                    state=2;
                    fuel_level=p_fuel_level(SMG)*100;
                else
                    % searching the p_max
                    if
(p_command(SMG)+(ramping_increase(SMG)*60*24*time_steps))>=max_power(SMG)
                        p_max=max_power(SMG);
                    else

p_max=p_command(SMG)+(ramping_increase(SMG)*60*24*time_steps);
                    end
                    % searching the p_min
                    if (p_command(SMG)-
(ramping_decrease(SMG)*60*24*time_steps))<=max_power(SMG)*0.1
                        p_min=max_power(SMG)*0.1;
                    else
                        p_min=p_command(SMG)-
(ramping_decrease(SMG)*60*24*time_steps);
                    end
                    power_vector=[p_max,(p_max-p_min)/2+p_min,p_min];

emission_vector=[(fuel_emission(SMG)/fuel_density(SMG)).*power_vector./(i
```

```
nterp1(efficiency_curve{SMG}(2,:),efficiency_curve{SMG}(1,:),power_vector
)./100).*(time_steps*24)];

cost_vector=[(fuel_cost(SMG)/fuel_density(SMG)).*power_vector./(interp1(e
fficiency_curve{SMG}(2,:),efficiency_curve{SMG}(1,:),power_vector)./100).
*(time_steps*24)];
                    autonomy_vector=[
(p_fuel_level(SMG)*fuel_capacity(SMG)) ./
(power_vector./(interp1(efficiency_curve{SMG}(2,:),efficiency_curve{SMG}(
1,:),power_vector)./100)./fuel_density(SMG)) ];
                    state=2;
                    fuel_level=p_fuel_level(SMG)*100;
                end

            elseif c_state(SMG)==3
                power_vector=[0,0,0];
                emission_vector=[0,0,0];
                cost_vector=[0,0,0];
                autonomy_vector=[NaN,NaN,NaN];
                state=3;
                fuel_level=p_fuel_level(SMG)*100;
            end

        elseif step==2
            if c_state(SMG)==0
                power_vector=0;
                emission_vector=0;
                cost_vector=0;
                autonomy_vector=NaN;
                state=0;
                fuel_level=p_fuel_level(SMG)*100;

            elseif c_state(SMG)==1
                power_vector=0;
                emission_vector=0;
                cost_vector=0;
                autonomy_vector=NaN;
                state=1;
                fuel_level=p_fuel_level(SMG)*100;

            elseif c_state(SMG)==2
                power_vector=command;

emission_vector=[(fuel_emission(SMG)/fuel_density(SMG)).*power_vector./(i
nterp1(efficiency_curve{SMG}(2,:),efficiency_curve{SMG}(1,:),power_vector
)./100).*(time_steps*24)];

cost_vector=[(fuel_cost(SMG)/fuel_density(SMG)).*power_vector./(interp1(e
fficiency_curve{SMG}(2,:),efficiency_curve{SMG}(1,:),power_vector)./100).
*(time_steps*24)];
                autonomy_vector=[ (p_fuel_level(SMG)*fuel_capacity(SMG))
./
(power_vector./(interp1(efficiency_curve{SMG}(2,:),efficiency_curve{SMG}(
1,:),power_vector)./100)./fuel_density(SMG)) ];
                state=2;
                fuel_level=p_fuel_level(SMG)*100;
```

```
                p_fuel_level(SMG)=p_fuel_level(SMG)-
(power_vector./(interp1(efficiency_curve{SMG}(2,:),efficiency_curve{SMG}(
1,:),power_vector)./100)./fuel_density(SMG))*time_steps*24/fuel_capacity(
SMG);
                p_command(SMG)=command;

            elseif c_state(SMG)==3
                power_vector=0;
                emission_vector=0;
                cost_vector=0;
                autonomy_vector=NaN;
                state=3;
                fuel_level=p_fuel_level(SMG)*100;
            end


        elseif step==3
            if c_state(SMG)==0
                if command==1
                    c_state(SMG)=command;
                    time_reference_1(SMG)=time;
                else
                    waitfor(msgbox('Impossible to go from state 0 to the
states 2 or 3'));
                end
            elseif c_state(SMG)==2
                if command==3
                    c_state(SMG)=command;
                    time_reference_3(SMG)=time;
                    p_command(SMG)=NaN;
                else
                    waitfor(msgbox('Impossible to go from state 2 to the
states 0 or 1'));
                end
            end

        elseif step==4
            time_reference_refilling(SMG)=time;

        end

    else

        SMG=S_SMG;
        % state evaluation
        if c_state(SMG)==1
            if time-time_reference_1(SMG)>=start_time(SMG)
                c_state(SMG)=2;
                time_reference_1(SMG)=NaN;
            else
                c_state(SMG)=1;
            end
        elseif c_state(SMG)==3
            if time-time_reference_3(SMG)>=stop_time(SMG)
                c_state(SMG)=0;
                time_reference_3(SMG)=NaN;
```

```
            else
                c_state(SMG)=3;
            end
        end

        % refilling evaluation
        if time-time_reference_refilling(SMG)>=refill_period(SMG)
            p_fuel_level(SMG)=1;
            time_reference_refilling(SMG)=NaN;
        end

        if nargin == 2
            step = 1;
        end

        if step==1
            if c_state(SMG)==0
                power_vector=[0,0,0];
                emission_vector=[0,0,0];
                cost_vector=[0,0,0];
                autonomy_vector=[NaN,NaN,NaN];
                state=0;
                fuel_level=p_fuel_level(SMG)*100;

            elseif c_state(SMG)==1
                power_vector=[0,0,0];
                emission_vector=0;
                cost_vector=0;
                autonomy_vector=[NaN,NaN,NaN];
                state=1;
                fuel_level=p_fuel_level(SMG)*100;

            elseif c_state(SMG)==2
                if isnan(p_command(SMG))
                    power_vector=[max_power(SMG),(max_power(SMG)-
max_power(SMG)*0.1)/2+(max_power(SMG)*0.1),max_power(SMG)*0.1];

emission_vector=[(fuel_emission(SMG)/fuel_density(SMG)).*power_vector./(i
nterp1(efficiency_curve{SMG}(2,:),efficiency_curve{SMG}(1,:),power_vector
)./100).*(time_steps*24)];

cost_vector=[(fuel_cost(SMG)/fuel_density(SMG)).*power_vector./(interp1(e
fficiency_curve{SMG}(2,:),efficiency_curve{SMG}(1,:),power_vector)./100).
*(time_steps*24)];
                    autonomy_vector=[
(p_fuel_level(SMG)*fuel_capacity(SMG)) ./
(power_vector./(interp1(efficiency_curve{SMG}(2,:),efficiency_curve{SMG}(
1,:),power_vector)./100)./fuel_density(SMG)) ];
                    state=2;
                    fuel_level=p_fuel_level(SMG)*100;
                else
                    % searching the p_max
                    if
(p_command(SMG)+(ramping_increase(SMG)*60*24*time_steps))>=max_power(SMG)
                        p_max=max_power(SMG);
                    else
```

```
p_max=p_command(SMG)+(ramping_increase(SMG)*60*24*time_steps);
                    end
                    % searching the p_min
                    if (p_command(SMG)-
(ramping_decrease(SMG)*60*24*time_steps))<=max_power(SMG)*0.1
                        p_min=max_power(SMG)*0.1;
                    else
                        p_min=p_command(SMG)-
(ramping_decrease(SMG)*60*24*time_steps);
                    end
                    power_vector=[p_max,(p_max-p_min)/2+p_min,p_min];

emission_vector=[(fuel_emission(SMG)/fuel_density(SMG)).*power_vector./(i
nterp1(efficiency_curve{SMG}(2,:),efficiency_curve{SMG}(1,:),power_vector
)./100).*(time_steps*24)];

cost_vector=[(fuel_cost(SMG)/fuel_density(SMG)).*power_vector./(interp1(e
fficiency_curve{SMG}(2,:),efficiency_curve{SMG}(1,:),power_vector)./100).
*(time_steps*24)];
                    autonomy_vector=[
(p_fuel_level(SMG)*fuel_capacity(SMG)) ./
(power_vector./(interp1(efficiency_curve{SMG}(2,:),efficiency_curve{SMG}(
1,:),power_vector)./100)./fuel_density(SMG)) ];
                    state=2;
                    fuel_level=p_fuel_level(SMG)*100;
                end

            elseif c_state(SMG)==3
                power_vector=[0,0,0];
                emission_vector=[0,0,0];
                cost_vector=[0,0,0];
                autonomy_vector=[NaN,NaN,NaN];
                state=3;
                fuel_level=p_fuel_level(SMG)*100;
            end

        elseif step==2
            if c_state(SMG)==0
                power_vector=0;
                emission_vector=0;
                cost_vector=0;
                autonomy_vector=NaN;
                state=0;
                fuel_level=p_fuel_level(SMG)*100;

            elseif c_state(SMG)==1
                power_vector=0;
                emission_vector=0;
                cost_vector=0;
                autonomy_vector=NaN;
                state=1;
                fuel_level=p_fuel_level(SMG)*100;

            elseif c_state(SMG)==2
                power_vector=command;
```

```
emission_vector=[(fuel_emission(SMG)/fuel_density(SMG)).*power_vector./(i
nterp1(efficiency_curve{SMG}(2,:),efficiency_curve{SMG}(1,:),power_vector
)./100).*(time_steps*24)];

cost_vector=[(fuel_cost(SMG)/fuel_density(SMG)).*power_vector./(interp1(e
fficiency_curve{SMG}(2,:),efficiency_curve{SMG}(1,:),power_vector)./100).
*(time_steps*24)];
                autonomy_vector=[ (p_fuel_level(SMG)*fuel_capacity(SMG))
./
(power_vector./(interp1(efficiency_curve{SMG}(2,:),efficiency_curve{SMG}(
1,:),power_vector)./100)./fuel_density(SMG)) ];
                state=2;
                fuel_level=p_fuel_level(SMG)*100;
                p_fuel_level(SMG)=p_fuel_level(SMG)-
(power_vector./(interp1(efficiency_curve{SMG}(2,:),efficiency_curve{SMG}(
1,:),power_vector)./100)./fuel_density(SMG))*time_steps*24/fuel_capacity(
SMG);
                p_command(SMG)=command;

            elseif c_state(SMG)==3
                power_vector=0;
                emission_vector=0;
                cost_vector=0;
                autonomy_vector=NaN;
                state=3;
                fuel_level=p_fuel_level(SMG)*100;
            end


        elseif step==3
            if c_state(SMG)==0
                if command==1
                    c_state(SMG)=command;
                    time_reference_1(SMG)=time;
                else
                    waitfor(msgbox('Impossible to go from state 0 to the
states 2 or 3'));
                end
            elseif c_state(SMG)==2
                if command==3
                    c_state(SMG)=command;
                    time_reference_3(SMG)=time;
                    p_command(SMG)=NaN;
                else
                    waitfor(msgbox('Impossible to go from state 2 to the
states 0 or 1'));
                end
            end

        elseif step==4
            time_reference_refilling(SMG)=time;

        end
    end
end
```

1.13. "simulator_starter.m"

```
function
[initial_simulation_time,final_simulation_time,time_steps,n_SMG]=simulato
r_starter()
% Some previous code to reset functions

clear irradiance_function
clear temperature_function
clear irradiance_forecast_function
clear temperature_forecast_function
clear PV
clear PV_inverter
clear grid
clear grid_selling_price_function
clear grid_buying_price_function
clear critical_loads
clear genset
clear battery

% Objective of the execution
options_program={'Create new simulation parameters.','Load previously
inserted parameters.'};

v=1:length(options_program);

prpt_program_choice=sprintf('Choose an option concerning on the
parameters of the simulation.');
choice_program = menu(prpt_program_choice,options_program{v});

switch(choice_program)
    case 1
% Some information about the system necessary for the simulation

waitfor(msgbox('Now I will ask you the system parameters, there are some
default values to help you.'))
prpt_system{1}=('Insert the initial date of the data provided in the
Excels using this format: 01/01/2017');
prpt_system{2}=('Insert the final date of the data provided in the Excels
using this format: 31/12/2017');
prpt_system{3}=('Number of SMG in the study');

system_default={'01/01/2017','31/12/2017','2'};
linies_system=[1,50;1,50;1,50];
system_parameters=inputdlg(prpt_system,'System
parameters',linies_system,system_default);

initial_data_date=system_parameters{1}; % The format must be the same
final_data_date=system_parameters{2}; % The format must be the same
n_SMG=str2double(system_parameters{3}); % The format must be the same
```

```
% Parameters of the PV system
waitfor(msgbox(sprintf('Now I will ask you the PV parameters, there are
some default values to help you. I''ll do it %d times.',n_SMG)))
for SMG=1:n_SMG
    prpt_PV{1}=('Insert the number of PV panels connected in series');
    prpt_PV{2}=('Insert the number of PV panels connected in paralel');
    prpt_PV{3}=('Insert the number of cells has the PV panel');
    prpt_PV{4}=('Introduce the NOCT, "Nominal Operating Cell Temperature
in °C');
    prpt_PV{5}=('Introduce the ideality factor.');
    prpt_PV{6}=('Introduce the temperature coeficient for current, not
expressed as a percentage! In K^-1');
    prpt_PV{7}=('Introduce the temperature coeficient for voltage, not
expressed as a percentage! In K^-1');
    prpt_PV{8}=('Introduce the shortcut current of the panel in A.');
    prpt_PV{9}=('Introduce the open-circuit voltage of the panel in V.');
    prpt_PV{10}=('Introduce the mpp current of the panel in A.');
    prpt_PV{11}=('Introduce the mpp voltage of the panel in V.');
    PV_default={'6','1','60','46','1.2','0.0007','-
0.003','8.89','37.9','31.7','8.36'};
    linies_PV=[1,50;1,50;1,50;1,50;1,50;1,50;1,50;1,50;1,50;1,50;1,50];
    PV_parameters=inputdlg(prpt_PV,sprintf('SMG%d: PV
parameters',SMG),linies_PV,PV_default);

    N_s(SMG)=str2double(PV_parameters{1});
    N_p(SMG)=str2double(PV_parameters{2});
    n_s(SMG)=str2double(PV_parameters{3});
    NOCT(SMG)=str2double(PV_parameters{4});
    A(SMG)=str2double(PV_parameters{5});
    K_i(SMG)=str2double(PV_parameters{6});
    K_v(SMG)=str2double(PV_parameters{7});
    I_sc(SMG)=str2double(PV_parameters{8});
    V_oc(SMG)=str2double(PV_parameters{9});
    V_mpp(SMG)=str2double(PV_parameters{10});
    I_mpp(SMG)=str2double(PV_parameters{11});

end

% Parameters of the PV_inverter

for SMG=1:n_SMG

    prpt_pv_inverter{1}=('Insert the maximum power it can provide. In
kW.');
    prpt_pv_inverter{2}=('Insert the efficiency points of the efficiency
curve of the PV inverter, in %. Use a vector format.');
    prpt_pv_inverter{3}=('Insert the points P_pv/P_rated, in %. Use a
vector format.');


pv_inverter_default={'1.5','[50,91,95,97,97.2,97.2,96.8]','[0,5,10,20,40,
60,105]'};
    linies_pv_inverter=[1,100;1,100;1,100];
    pv_inverter_parameters=inputdlg(prpt_pv_inverter,sprintf('SMG%d: PV
inverter parameters',SMG),linies_pv_inverter,pv_inverter_default);
```

```
    nominal_pv_inverter_power(SMG)=str2double(pv_inverter_parameters{1});

pv_inverter_efficiency_curve{SMG}(1,:)=str2num(pv_inverter_parameters{2})
; %(1)Efficiency vetor in %, (2)Output power vector

pv_inverter_efficiency_curve{SMG}(2,:)=str2num(pv_inverter_parameters{3})
;

end

% Parameters of the grid
waitfor(msgbox(sprintf('Now I will ask you the grid parameters, there are
some default values to help you. I''ll do it %d times.',n_SMG)))
for SMG=1:n_SMG
    prpt_grid{1}=('Insert the maximum power the grid can provide.');
    prpt_grid{2}=('Insert the number of grid falls');


    grid_default={'2','0'};
    linies_grid=[1,50;1,50];
    grid_parameters=inputdlg(prpt_grid,sprintf('SMG%d: Grid
parameters',SMG),linies_grid,grid_default);

    max_P_grid(SMG)=str2double(grid_parameters{1});
    number_grid_falls(SMG)=str2double(grid_parameters{2});
end


for SMG=1:n_SMG
    if number_grid_falls(SMG)==0;
        grid_fall{SMG}=datenum('00/00/0000 00:00:00','dd/mm/yyyy
HH:MM:SS');
        grid_fall_duration{SMG}=0; % In days
    else
        clear prpt_grid2 prpt_grid3
        for i=1:number_grid_falls(SMG)
            prpt_grid2{i}=('Insert the times when the grid falls:
01/01/2017 14:00:00. If there are more points use a cell format.');
            prpt_grid3{i}=('Insert the duration of each fall in days. If
there are more than one use a vector format.');
        end

        grid_parameters2=inputdlg(prpt_grid2,sprintf('SMG%d: Grid
parameters',SMG));
        grid_parameters3=inputdlg(prpt_grid3,sprintf('SMG%d: Grid
parameters',SMG));

        grid_fall{SMG}=datenum(grid_parameters2,'dd/mm/yyyy HH:MM:SS');
        grid_fall_duration{SMG}=str2double(grid_parameters3); % In days
    end
end

% Parameters of the batteries
```

```
waitfor(msgbox(sprintf('Now I will ask you the battery parameters, there 
are some default values to help you. I''ll do it %d times.',n_SMG)))

for SMG=1:n_SMG
    prpt_battery{1}=('Insert the maximum power the battery can provide. 
In kW.');
    prpt_battery{2}=('Insert the voltage of the batteries'' bank. In 
V.');
    prpt_battery{3}=('Insert the reted capacity of the battery. In Ah. 
Use the capacity for the most common use of this bank in therms of 
discharge rate.');
    prpt_battery{4}=('Insert the maximum bulk power the battery can be 
charged with. In kW.');
    prpt_battery{5}=('Insert the temperature points of the efficiency 
curve of the battery, in °C. Use a vector format and introduce at least 
two curves.');
    prpt_battery{6}=('Insert the SoC points of the efficiency curve of 
the battery, in %. Use a vector format and introduce at least two 
curves.');
    prpt_battery{7}=('Insert the efficiency points of the efficiency 
curve of the battery, in %. Use a vector format and introduce at least 
two curves.');
    prpt_battery{8}=('Insert the SoC at the beginning of the simulation. 
In %.');
    prpt_battery{9}=('Insert the battery charger/inverter efficiency. In 
%.');
    battery_default={'2.4','24','303','1.1','[-20,-10,0,10,20,30,40;-20,-
10,0,10,20,30,40]','[30,30,30,30,30,30,30;90,90,90,90,90,90,90]','[75,85,
92,97,100,97,86;60,71,80,87,90,85,58]','80','95'};
    linies_battery=[1,50;1,50;1,50;1,50;1,50;1,50;1,50;1,50;1,50];
    battery_parameters=inputdlg(prpt_battery,sprintf('SMG%d: Battery 
parameters',SMG),linies_battery,battery_default);

    battery_max_power(SMG)=str2double(battery_parameters{1}); %Demanded 
in kW
    battery_voltage(SMG)=str2double(battery_parameters{2});
    battery_rated_capacity(SMG)=str2double(battery_parameters{3});
    bulk_power(SMG)=str2double(battery_parameters{4}); %Demanded in kW
    battery_efficiency_curve{SMG}(:,:,1)=str2num(battery_parameters{5});
    battery_efficiency_curve{SMG}(:,:,2)=str2num(battery_parameters{6});
    battery_efficiency_curve{SMG}(:,:,3)=str2num(battery_parameters{7});
    initial_SoC(SMG)=str2double(battery_parameters{8});
    charger_efficiency(SMG)=str2double(battery_parameters{9})

end



% Parameters of the gensets

for SMG=1:n_SMG
    prpt_genset{1}=('Insert the time the genset needs to start and reach 
the needed rpm. In days.');
    prpt_genset{2}=('Insert the time the genset needs to stop. In 
days.');
```

```
    prpt_genset{3}=('Insert the ramping of increasing power the genset
can achieve. In kW/min.');
    prpt_genset{4}=('Insert the ramping of decreasing power the genset
can achieve. In kW/min.');
    prpt_genset{5}=('Insert the maximum power it can provide. In kW.');
    prpt_genset{6}=('Insert the initial liters of fuel in the genset.');
    prpt_genset{7}=('Insert the capacity in liters of the genset.');
    prpt_genset{8}=('Insert the fuel cost. In €/liter.');
    prpt_genset{9}=('Insert the fuel emissions. In kgCO2/liter.');
    prpt_genset{10}=('Insert the fuel energy density. In kWh/liter.');
    prpt_genset{11}=('Insert the period of time it takes to refill the
tank till the alarm is activated. In days.');
    prpt_genset{12}=('Insert the efficiency points of the efficiency
curve of the genset, in %. Use a vector format.');
    prpt_genset{13}=('Insert the output power points of the efficiency
curve of the genset, in kW. Use a vector format.');


genset_default={'0.002','0.003','2','2','5','30','35','1','2.64','14','1'
,'[0,5,15,20,25,28,30]','[0,0.1,1,2,3,4,5]'};

linies_genset=[1,100;1,100;1,100;1,100;1,100;1,100;1,100;1,100;1,100;1,10
0;1,100;1,100;1,100];
    genset_parameters=inputdlg(prpt_genset,sprintf('SMG%d: Genset
parameters',SMG),linies_genset,genset_default);

    start_time(SMG)=str2double(genset_parameters{1});
    stop_time(SMG)=str2double(genset_parameters{2});
    ramping_increase(SMG)=str2double(genset_parameters{3}); %kW/minute
    ramping_decrease(SMG)=str2double(genset_parameters{4}); %kW/minute
    max_genset_power(SMG)=str2double(genset_parameters{5});
    initial_fuel(SMG)=str2double(genset_parameters{6});% liters
    fuel_capacity(SMG)=str2double(genset_parameters{7});% liters
    fuel_cost(SMG)=str2double(genset_parameters{8}); % €/liter
    fuel_emission(SMG)=str2double(genset_parameters{9}); %kgCO2/liter
    fuel_density(SMG)=str2double(genset_parameters{10}); %kWh/liter
    refill_period(SMG)=str2double(genset_parameters{11});
    genset_efficiency_curve{SMG}(1,:)=str2num(genset_parameters{12});
%(1)Efficiency vetor in %, (2)Output power vector
    genset_efficiency_curve{SMG}(2,:)=str2num(genset_parameters{13});

end



saveit = questdlg('Do you want to save the introduced parameters to load
them for other simulations?','Save data?','default');

if strcmp(saveit,'Yes')
    file_name=inputdlg({'Write the file name below'},'Naming data
file',[1,100],{'parameters_simulation_1.mat'});
    save(char(file_name))
end

case 2
```

```
    file_name=inputdlg({'Write the file name below'},'Naming data
file',[1,100],{'parameters_simulation_1.mat'});
    load(char(file_name))

end
% Introduce variable parameters such simulations time and time_step
waitfor(msgbox('Finally insert the times for this specific simulation.'))
prpt_times{1}=('Insert the initial simulation time using this format:
01/01/2017 00:00:00');
prpt_times{2}=('Insert the initial simulation time using this format:
02/01/2017 23:59:03');
prpt_times{3}=('Indicate the time step expressed in days. Can be a float
as 0.001');

times_default={'01/01/2017 00:00:00','02/01/2017 00:00:00','0.01'};
linies_times=[1,50;1,50;1,50];
times_parameters=inputdlg(prpt_times,'Times
parameters',linies_times,times_default);

initial_simulation_time=times_parameters{1}; % The format must be the
same
final_simulation_time=times_parameters{2}; % The format must be the same
time_steps=str2double(times_parameters{3}); % Expressed in days

% Untouchable code
% Setting system parameters
setappdata(0,'initial_date_external_data',initial_data_date) % Inicial
date of the data of irradiance provided
setappdata(0,'final_date_external_data',final_data_date) % Final date of
the data of irradiance provided
setappdata(0,'number_of_SMG',n_SMG) % Final date of the data of
irradiance provided
setappdata(0,'time_steps',time_steps) % Expressed in days
% Setting PV parameters
setappdata(0,'series_connected_panels',N_s);
setappdata(0,'series_connected_cells',n_s);
setappdata(0,'parallel_connected_panels',N_p);
setappdata(0,'NOCT',NOCT);
setappdata(0,'ideality_factor',A);
setappdata(0,'temp_coeficient_current',K_i);
setappdata(0,'temp_coeficient_voltage',K_v);
setappdata(0,'Shortcut_current',I_sc);
setappdata(0,'Open_circuit_voltage',V_oc);
setappdata(0,'MPP_voltage',V_mpp);
setappdata(0,'MPP_current',I_mpp);
% Setting grid parameters
setappdata(0,'maximum_power_allowed_grid',max_P_grid);
setappdata(0,'when_grid_falls',grid_fall);
setappdata(0,'grid_fall_duration',grid_fall_duration);
% Setting battery parameters
setappdata(0,'battery_max_power',battery_max_power);
setappdata(0,'bulk_power',bulk_power); %Demanded in kW
setappdata(0,'battery_efficiency_curve',battery_efficiency_curve);
%(1)Temperature matrix in °C, (2)SoC matrix in % (3) Efficiency matrix in
%
setappdata(0,'battery_voltage',battery_voltage);
setappdata(0,'battery_rated_capacity',battery_rated_capacity);
```

```
setappdata(0,'initial_SoC',initial_SoC);
setappdata(0,'charger_efficiency',charger_efficiency);
% Setting genset parameters
setappdata(0,'time_reach_rpm',start_time);
setappdata(0,'time_reach_0rpm',stop_time);
setappdata(0,'ramping_power_increase',ramping_increase); %kW/minute
setappdata(0,'ramping_power_decrease',ramping_decrease); %kW/minute
setappdata(0,'genset_max_power',max_genset_power);
setappdata(0,'initial_fuel',initial_fuel);
setappdata(0,'fuel_capacity',fuel_capacity);
setappdata(0,'fuel_cost',fuel_cost); % €/litter
setappdata(0,'fuel_emission',fuel_emission); %kgCO2/litter
setappdata(0,'fuel_density',fuel_density); %kWh/litter
setappdata(0,'genset_efficiency_curve',genset_efficiency_curve);
%(1,:)Efficiency vector in %, (2,:)Output power vector
setappdata(0,'refilling_periods',refill_period);
% Setting inverter parameters
setappdata(0,'pv_inverter_efficiency_curve',pv_inverter_efficiency_curve)
; %(1,:)Efficiency vector in %, (2,:)Output power vector
setappdata(0,'nominal_power_inverter',nominal_pv_inverter_power);

initial_simulation_time=datenum(initial_simulation_time,'dd/mm/yyyy
HH:MM:SS');
final_simulation_time=datenum(final_simulation_time,'dd/mm/yyyy
HH:MM:SS');

end
```

1.14. “simulator.m”

```
% This function contains all the code that initializes the simulator, is
% where is programed all the interface that appears at the beggining
clear all
clc

tic
[initial_simulation_time,final_simulation_time,time_steps,n_SMG]=simulato
r_starter();
toc

% Core of the program, where all the algorithms must be inserted,

i=0; %This index will be the time counter, but in integers, its units
will be the time steps if i=3 three time steps.

for SMG=1:n_SMG %Space for initialization and preallocation
    ies=length(initial_simulation_time:time_steps:final_simulation_time);

end
time_vector=zeros(1,ies);
```

```
% Space for initializations


tic

waitb = waitbar(0,'Please wait...');


for time=initial_simulation_time:time_steps:final_simulation_time %
Untouchable
    i=i+1;          % Untouchable





    time_vector(:,i)=time;
    waitbar(i/ies,waitb)
end
toc
close(waitb)
points=8;
x_labels
={datestr(time_vector(1:round(length(time_vector)/points):length(time_vec
tor)),'dd/mm/yyyy HH:MM:SS')};
period=(final_simulation_time-initial_simulation_time)/365;
```

1.15. Simulator with the algorithms applied

```
% This function contains all the code that initializes the simulator, is
% where is programed all the interface that appears at the beggining
clear all
clc

tic
[initial_simulation_time,final_simulation_time,time_steps,n_SMG]=simulato
r_starter();
toc

% Core of the program, where all the algorithms must be inserted,

i=0; %This index will be the time counter, but in integers, its units
will be the time steps if i=3 three time steps.

for SMG=1:n_SMG %Space for initialization and preallocation
    ies=length(initial_simulation_time:time_steps:final_simulation_time);
    bat_power_vector1{SMG}=zeros(1,6);
    bat_autonomy_vector1{SMG}=zeros(1,3);
    SoC{SMG}=zeros(1,ies);
    P_max_grid{SMG}=0;
    grid_purchasing_price{SMG}=0;
    grid_selling_price{SMG}=0;
```

```
    P_max_pv{SMG}=0;
    P_max_charge_bat{SMG}=0;
    P_max_discharge_bat{SMG}=0;
    P_nom_load{SMG}=zeros(1,ies);
    P_min_load{SMG}=zeros(1,ies);
    P_load{SMG}=zeros(1,ies);
    P_pv{SMG}=zeros(1,ies);
    P_grid{SMG}=zeros(1,ies);
    P_bat{SMG}=zeros(1,ies);
    ok{SMG}=zeros(1,ies);
    unbalance{SMG}=zeros(1,ies);
    state{SMG}=zeros(1,ies);
    bat_power{SMG}=zeros(1,ies);
    bat_autonomy{SMG}=zeros(1,ies);
    Grid_cost{SMG}=zeros(1,ies);
    Grid_income{SMG}=zeros(1,ies);

end
time_vector=zeros(1,ies);


% Space for initializations

min_SoC=[50,30,50,30,30];
SoC_grid=[80,60,80,60,0];
price=0.08;

tic

waitb = waitbar(0,'Please wait...');


for time=initial_simulation_time:time_steps:final_simulation_time %
Untouchable
    i=i+1;          % Untouchable


    for SMG=[1,3]


        %Step one getting the information

[bat_power_vector1{SMG},bat_autonomy_vector1{SMG},SoC{SMG}(i)]=battery(ti
me,SMG);

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

[P_nom_load_sep{SMG}(:,i),P_min_load_sep{SMG}(:,i),priority{SMG}(:,i),loa
d_name{SMG},load_identifier{SMG}]= critical_loads(time,SMG);
        [P_max_pv{SMG}]= PV_inverter(time,SMG);

        %Applying loses
```

```
bat_power_vector1{SMG}=[bat_power_vector1{SMG}(1:3)*0.95,bat_power_vector
1{SMG}(4:6)];
        P_max_grid{SMG}=P_max_grid{SMG}*0.95;
        P_max_pv{SMG}=P_max_pv{SMG}*0.81;

        %Assigning variables
        P_max_charge_bat{SMG}=bat_power_vector1{SMG}(6);
        P_max_discharge_bat{SMG}=bat_power_vector1{SMG}(1);
        P_nom_load{SMG}(i)=sum(P_nom_load_sep{SMG}(:,i));
        P_min_load{SMG}(i)=sum(P_min_load_sep{SMG}(:,i));

        %Step two
        %batteries
        step=2;

        if P_max_pv{SMG} - P_nom_load{SMG}(i) - (-P_max_charge_bat{SMG})
> P_max_grid{SMG}

            P_load{SMG}(i)=P_nom_load{SMG}(i);
            P_pv{SMG}(i)=0;
            if P_load{SMG}(i)>P_max_grid{SMG}
                P_grid{SMG}(i)=P_max_grid{SMG};
                P_bat{SMG}(i)=P_load{SMG}(i)-P_grid{SMG}(i);
            elseif P_load{SMG}(i)<=P_max_grid{SMG}
                P_grid{SMG}(i)=P_load{SMG}(i);
                P_bat{SMG}(i)=0;
            end

            if -
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i)==0
                ok{SMG}(i)=1;
                unbalance{SMG}(i)=-
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i);
            else
                ok{SMG}(i)=0;
                unbalance{SMG}(i)=-
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i);
            end

            state{SMG}(i)=1;

        elseif P_max_pv{SMG} > P_nom_load{SMG}(i) + (-
P_max_charge_bat{SMG})

            P_load{SMG}(i)=P_nom_load{SMG}(i);
            P_pv{SMG}(i)=P_max_pv{SMG};
            P_bat{SMG}(i)=P_max_charge_bat{SMG};
            P_grid{SMG}(i)=-P_pv{SMG}(i)+(-P_bat{SMG}(i))+P_load{SMG}(i);

            if -
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i)==0
                ok{SMG}(i)=1;
                unbalance{SMG}(i)=-
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i);
            else
```

```
                ok{SMG}(i)=0;
                unbalance{SMG}(i)=-
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i);
            end

            state{SMG}(i)=2;

        elseif P_nom_load{SMG}(i) < P_max_pv{SMG}

            P_load{SMG}(i)=P_nom_load{SMG}(i);
            P_pv{SMG}(i)=P_max_pv{SMG};
            P_grid{SMG}(i)=0;
            P_bat{SMG}(i)=-P_pv{SMG}(i)-P_grid{SMG}(i)+P_load{SMG}(i);

            if -
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i)==0
                ok{SMG}(i)=1;
                unbalance{SMG}(i)=-
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i);
            else
                ok{SMG}(i)=0;
                unbalance{SMG}(i)=-
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i);
            end

            state{SMG}(i)=3;

        elseif P_nom_load{SMG}(i) < P_max_grid{SMG} + P_max_pv{SMG}

            P_load{SMG}(i)=P_nom_load{SMG}(i);
            P_pv{SMG}(i)=P_max_pv{SMG};
            P_grid{SMG}(i)=P_load{SMG}(i)-P_pv{SMG}(i);

            if SoC{SMG}(i)<SoC_grid(SMG) && P_max_pv{SMG}>0 &&
grid_purchasing_price{SMG}<=price

                if (P_max_grid{SMG}-P_grid{SMG}(i))>=-
P_max_charge_bat{SMG}
                    P_bat{SMG}(i)=P_max_charge_bat{SMG}/2;
                else
                    P_bat{SMG}(i)=-(P_max_grid{SMG}-P_grid{SMG}(i))/2;
                end

            elseif SoC{SMG}(i)<SoC_grid(SMG) && P_max_pv{SMG}==0 &&
grid_purchasing_price{SMG}<=price
                if (P_max_grid{SMG}-P_grid{SMG}(i))>=-
P_max_charge_bat{SMG}
                    P_bat{SMG}(i)=P_max_charge_bat{SMG};
                else
                    P_bat{SMG}(i)=-(P_max_grid{SMG}-P_grid{SMG}(i));
                end

            elseif SoC{SMG}(i)>=SoC_grid(SMG)
                P_bat{SMG}(i)=0;
            end
```

```
            P_grid{SMG}(i)=P_grid{SMG}(i)+(-P_bat{SMG}(i));

            if -
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i)==0
                ok{SMG}(i)=1;
                unbalance{SMG}(i)=-
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i);
            else
                ok{SMG}(i)=0;
                unbalance{SMG}(i)=-
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i);
            end

            state{SMG}(i)=4;

        elseif (P_nom_load{SMG}(i) < P_max_grid{SMG} + P_max_pv{SMG} +
P_max_discharge_bat{SMG}) && SoC{SMG}(i) > min_SoC(SMG)

            P_load{SMG}(i)=P_nom_load{SMG}(i);
            P_pv{SMG}(i)=P_max_pv{SMG};
            P_grid{SMG}(i)=P_max_grid{SMG};
            P_bat{SMG}(i)=-P_pv{SMG}(i)-P_grid{SMG}(i)+P_load{SMG}(i);

            if -
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i)==0
                ok{SMG}(i)=1;
                unbalance{SMG}(i)=-
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i);
            else
                ok{SMG}(i)=0;
                unbalance{SMG}(i)=-
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i);
            end

            state{SMG}(i)=5;

        elseif P_nom_load{SMG}(i) > (P_max_grid{SMG} + P_max_pv{SMG} +
P_max_discharge_bat{SMG}) || (P_nom_load{SMG}(i) >(P_max_grid{SMG} +
P_max_pv{SMG}) && SoC{SMG}(i) <= min_SoC(SMG))

            P_load{SMG}(i)=P_min_load{SMG}(i);

            while P_load{SMG}(i) > (P_max_grid{SMG} + P_max_pv{SMG} +
P_max_discharge_bat{SMG}) || (P_load{SMG}(i) >(P_max_grid{SMG} +
P_max_pv{SMG}) && SoC{SMG}(i) <= min_SoC(SMG))


P_min_load_sep{SMG}(:,i)=(priority{SMG}(:,i)<max(priority{SMG}(:,i))).*P_
min_load_sep{SMG}(:,i);

priority{SMG}(:,i)=(priority{SMG}(:,i)<max(priority{SMG}(:,i))).*priority
{SMG}(:,i);

                P_load{SMG}(i)=sum(P_min_load_sep{SMG}(:,i));
```

```
        end
        % Once the load demand is reduced then there is another check
        % to know in which situation the system is.
            if P_max_pv{SMG} - P_load{SMG}(i) - (-
P_max_charge_bat{SMG}) > P_max_grid{SMG}

                P_load{SMG}(i)=P_load{SMG}(i);
                P_pv{SMG}(i)=0;
                if P_load{SMG}(i)>P_max_grid{SMG}
                    P_grid{SMG}(i)=P_max_grid{SMG};
                    P_bat{SMG}(i)=P_load{SMG}(i)-P_grid{SMG}(i);
                elseif P_load{SMG}(i)<=P_max_grid{SMG}
                    P_grid{SMG}(i)=P_load{SMG}(i);
                    P_bat{SMG}(i)=0;
                end

                state{SMG}(i)=1;

            elseif P_max_pv{SMG} > P_load{SMG}(i) + (-
P_max_charge_bat{SMG})

                P_load{SMG}(i)=P_load{SMG}(i);
                P_pv{SMG}(i)=P_max_pv{SMG};
                P_bat{SMG}(i)=P_max_charge_bat{SMG};
                P_grid{SMG}(i)=-P_pv{SMG}(i)+(-
P_bat{SMG}(i))+P_load{SMG}(i);

                state{SMG}(i)=2;

            elseif P_load{SMG}(i) < P_max_pv{SMG}

                P_load{SMG}(i)=P_load{SMG}(i);
                P_pv{SMG}(i)=P_max_pv{SMG};
                P_grid{SMG}(i)=0;
                P_bat{SMG}(i)=-P_pv{SMG}(i)-
P_grid{SMG}(i)+P_load{SMG}(i);

                state{SMG}(i)=3;

            elseif P_load{SMG}(i) < P_max_grid{SMG} + P_max_pv{SMG}

                P_load{SMG}(i)=P_load{SMG}(i);
                P_pv{SMG}(i)=P_max_pv{SMG};
                P_grid{SMG}(i)=P_load{SMG}(i)-P_pv{SMG}(i);

                if SoC{SMG}(i)<SoC_grid(SMG) && P_max_pv{SMG}>0 &&
grid_purchasing_price{SMG}<=price

                    if (P_max_grid{SMG}-P_grid{SMG}(i))>=-
P_max_charge_bat{SMG}
                        P_bat{SMG}(i)=P_max_charge_bat{SMG}/2;
                    else
                        P_bat{SMG}(i)=-(P_max_grid{SMG}-
P_grid{SMG}(i))/2;
```

```
                        end

                    elseif SoC{SMG}(i)<SoC_grid(SMG) && P_max_pv{SMG}==0
&& grid_purchasing_price{SMG}<=price
                        if (P_max_grid{SMG}-P_grid{SMG}(i))>=-
P_max_charge_bat{SMG}
                            P_bat{SMG}(i)=P_max_charge_bat{SMG};
                        else
                            P_bat{SMG}(i)=-(P_max_grid{SMG}-
P_grid{SMG}(i));
                        end

                    elseif SoC{SMG}(i)>SoC_grid(SMG)
                        P_bat{SMG}(i)=0;
                    end
                    P_grid{SMG}(i)=P_grid{SMG}(i)+(-P_bat{SMG}(i));
                    state{SMG}(i)=4;

                elseif (P_load{SMG}(i) < P_max_grid{SMG} + P_max_pv{SMG}
+ P_max_discharge_bat{SMG}) && SoC{SMG}(i) > min_SoC(SMG)

                    P_load{SMG}(i)=P_load{SMG}(i);
                    P_pv{SMG}(i)=P_max_pv{SMG};
                    P_grid{SMG}(i)=P_max_grid{SMG};
                    P_bat{SMG}(i)=-P_pv{SMG}(i)-
P_grid{SMG}(i)+P_load{SMG}(i);

                    state{SMG}(i)=5;
                end

            if -
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i)==0
                ok{SMG}(i)=1;
                unbalance{SMG}(i)=-
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i);
            else
                ok{SMG}(i)=0;
                unbalance{SMG}(i)=-
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i);
            end

            state{SMG}(i)=6;

        elseif (P_max_grid{SMG} + P_max_pv{SMG}  == 0) && SoC{SMG}(i) <=
min_SoC(SMG)

            P_load{SMG}(i)=0;
            P_pv{SMG}(i)=0;
            P_grid{SMG}(i)=0;
            P_bat{SMG}(i)=0;

            if -
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i)==0
                ok{SMG}(i)=1;
                unbalance{SMG}(i)=-
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i);
```

```
            else
                ok{SMG}(i)=0;
                unbalance{SMG}(i)=-
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i);
            end

            state{SMG}(i)=7;
        end


        %Step 7 and 8 and 9

[bat_power{SMG}(i),bat_autonomy{SMG}(i)]=battery(time,SMG,2,P_bat{SMG}(i)
);

        %Variable Costs
            % Costs of grid energy
            if P_grid{SMG}(i)>0

Grid_cost{SMG}(i)=P_grid{SMG}(i)/0.95*time_steps*24*grid_purchasing_price
{SMG};
            else
                Grid_cost{SMG}(i)=0;
            end
            % Income from selling energy to the grid
            if P_grid{SMG}(i)<0

Grid_income{SMG}(i)=P_grid{SMG}(i)*time_steps*24*grid_selling_price{SMG};
            else
                Grid_income{SMG}(i)=0;
            end
    end

    for SMG=[2,4,5]


        %Step one getting the information

[bat_power_vector1{SMG},bat_autonomy_vector1{SMG},SoC{SMG}(i)]=battery(ti
me,SMG);

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

[P_nom_load_sep{SMG}(:,i),P_min_load_sep{SMG}(:,i),priority{SMG}(:,i),loa
d_name{SMG},load_identifier{SMG}]= critical_loads(time,SMG);
        [P_max_pv{SMG}]= PV_inverter(time,SMG);

        %Applying loses

bat_power_vector1{SMG}=[bat_power_vector1{SMG}(1:3)*0.95,bat_power_vector
1{SMG}(4:6)];
        P_max_grid{SMG}=P_max_grid{SMG}*0.95;
        P_max_pv{SMG}=P_max_pv{SMG}*0.81;

        %Assigning variables
```

```
        P_max_charge_bat{SMG}=bat_power_vector1{SMG}(6);
        P_max_discharge_bat{SMG}=bat_power_vector1{SMG}(1);
        P_nom_load{SMG}(i)=sum(P_nom_load_sep{SMG}(:,i));
        P_min_load{SMG}(i)=sum(P_min_load_sep{SMG}(:,i));

        %Step two
        %batteries
        step=2;

        if P_max_pv{SMG} - P_nom_load{SMG}(i) - (-P_max_charge_bat{SMG})
> P_max_grid{SMG}

            P_load{SMG}(i)=P_nom_load{SMG}(i);
            P_pv{SMG}(i)=0;
            if P_load{SMG}(i)>P_max_grid{SMG}
                P_grid{SMG}(i)=P_max_grid{SMG};
                P_bat{SMG}(i)=P_load{SMG}(i)-P_grid{SMG}(i);
            elseif P_load{SMG}(i)<=P_max_grid{SMG}
                P_grid{SMG}(i)=P_load{SMG}(i);
                P_bat{SMG}(i)=0;
            end

            if -
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i)==0
                ok{SMG}(i)=1;
                unbalance{SMG}(i)=-
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i);
            else
                ok{SMG}(i)=0;
                unbalance{SMG}(i)=-
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i);
            end

            state{SMG}(i)=1;

        elseif P_max_pv{SMG} > P_nom_load{SMG}(i) + (-
P_max_charge_bat{SMG})

            P_load{SMG}(i)=P_nom_load{SMG}(i);
            P_pv{SMG}(i)=P_max_pv{SMG};
            P_bat{SMG}(i)=P_max_charge_bat{SMG};
            P_grid{SMG}(i)=-P_pv{SMG}(i)+(-P_bat{SMG}(i))+P_load{SMG}(i);

            if -
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i)==0
                ok{SMG}(i)=1;
                unbalance{SMG}(i)=-
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i);
            else
                ok{SMG}(i)=0;
                unbalance{SMG}(i)=-
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i);
            end

            state{SMG}(i)=2;
```

```
        elseif P_nom_load{SMG}(i) < P_max_pv{SMG}

            P_load{SMG}(i)=P_nom_load{SMG}(i);
            P_pv{SMG}(i)=P_max_pv{SMG};
            P_grid{SMG}(i)=0;
            P_bat{SMG}(i)=-P_pv{SMG}(i)-P_grid{SMG}(i)+P_load{SMG}(i);

            if -
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i)==0
                ok{SMG}(i)=1;
                unbalance{SMG}(i)=-
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i);
            else
                ok{SMG}(i)=0;
                unbalance{SMG}(i)=-
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i);
            end

            state{SMG}(i)=3;

        elseif P_nom_load{SMG}(i) > P_max_pv{SMG}  && P_nom_load{SMG}(i)
< (P_max_pv{SMG} + P_max_discharge_bat{SMG}) && P_max_pv{SMG} > 0 &&
SoC{SMG}(i) > min_SoC(SMG) && (P_max_pv{SMG} - P_nom_load{SMG}(i)) <= (-
P_max_charge_bat{SMG})

            P_load{SMG}(i)=P_nom_load{SMG}(i);
            P_pv{SMG}(i)=P_max_pv{SMG};
            P_grid{SMG}(i)=0;
            P_bat{SMG}(i)=-P_pv{SMG}(i)+P_load{SMG}(i);

            if -
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i)==0
                ok{SMG}(i)=1;
                unbalance{SMG}(i)=-
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i);
            else
                ok{SMG}(i)=0;
                unbalance{SMG}(i)=-
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i);
            end

            state{SMG}(i)=8;


        elseif P_nom_load{SMG}(i) < P_max_grid{SMG} + P_max_pv{SMG}

            P_load{SMG}(i)=P_nom_load{SMG}(i);
            P_pv{SMG}(i)=P_max_pv{SMG};
            P_grid{SMG}(i)=P_load{SMG}(i)-P_pv{SMG}(i);

            if SoC{SMG}(i)<SoC_grid(SMG) && P_max_pv{SMG}>0 &&
grid_purchasing_price{SMG}<=price

                if (P_max_grid{SMG}-P_grid{SMG}(i))>=-
P_max_charge_bat{SMG}
```

```
                    P_bat{SMG}(i)=P_max_charge_bat{SMG}/2;
                else
                    P_bat{SMG}(i)=-(P_max_grid{SMG}-P_grid{SMG}(i))/2;
                end

            elseif SoC{SMG}(i)<SoC_grid(SMG) && P_max_pv{SMG}==0 &&
grid_purchasing_price{SMG}<=price
                if (P_max_grid{SMG}-P_grid{SMG}(i))>=-
P_max_charge_bat{SMG}
                    P_bat{SMG}(i)=P_max_charge_bat{SMG};
                else
                    P_bat{SMG}(i)=-(P_max_grid{SMG}-P_grid{SMG}(i));
                end

            elseif SoC{SMG}(i)>=SoC_grid(SMG)
                P_bat{SMG}(i)=0;
            end

            P_grid{SMG}(i)=P_grid{SMG}(i)+(-P_bat{SMG}(i));

            if -
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i)==0
                ok{SMG}(i)=1;
                unbalance{SMG}(i)=-
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i);
            else
                ok{SMG}(i)=0;
                unbalance{SMG}(i)=-
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i);
            end

            state{SMG}(i)=4;

        elseif (P_nom_load{SMG}(i) < P_max_grid{SMG} + P_max_pv{SMG} +
P_max_discharge_bat{SMG}) && SoC{SMG}(i) > min_SoC(SMG)

            P_load{SMG}(i)=P_nom_load{SMG}(i);
            P_pv{SMG}(i)=P_max_pv{SMG};
            P_grid{SMG}(i)=P_max_grid{SMG};
            P_bat{SMG}(i)=-P_pv{SMG}(i)-P_grid{SMG}(i)+P_load{SMG}(i);

            if -
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i)==0
                ok{SMG}(i)=1;
                unbalance{SMG}(i)=-
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i);
            else
                ok{SMG}(i)=0;
                unbalance{SMG}(i)=-
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i);
            end

            state{SMG}(i)=5;
```

```
        elseif P_nom_load{SMG}(i) > (P_max_grid{SMG} + P_max_pv{SMG} +
P_max_discharge_bat{SMG}) || (P_nom_load{SMG}(i) >(P_max_grid{SMG} +
P_max_pv{SMG}) && SoC{SMG}(i) <= min_SoC(SMG))

            P_load{SMG}(i)=P_min_load{SMG}(i);

            while P_load{SMG}(i) > (P_max_grid{SMG} + P_max_pv{SMG} +
P_max_discharge_bat{SMG}) || (P_load{SMG}(i) >(P_max_grid{SMG} +
P_max_pv{SMG}) && SoC{SMG}(i) <= min_SoC(SMG))


P_min_load_sep{SMG}(:,i)=(priority{SMG}(:,i)<max(priority{SMG}(:,i))).*P_
min_load_sep{SMG}(:,i);

priority{SMG}(:,i)=(priority{SMG}(:,i)<max(priority{SMG}(:,i))).*priority
{SMG}(:,i);

                P_load{SMG}(i)=sum(P_min_load_sep{SMG}(:,i));

            end
            % Once the load demand is reduced then there is another check
            % to know in which situation the system is.
                if P_max_pv{SMG} - P_load{SMG}(i) - (-
P_max_charge_bat{SMG}) > P_max_grid{SMG}

                    P_load{SMG}(i)=P_load{SMG}(i);
                    P_pv{SMG}(i)=0;
                    if P_load{SMG}(i)>P_max_grid{SMG}
                        P_grid{SMG}(i)=P_max_grid{SMG};
                        P_bat{SMG}(i)=P_load{SMG}(i)-P_grid{SMG}(i);
                    elseif P_load{SMG}(i)<=P_max_grid{SMG}
                        P_grid{SMG}(i)=P_load{SMG}(i);
                        P_bat{SMG}(i)=0;
                    end

                    state{SMG}(i)=1;

                elseif P_max_pv{SMG} > P_load{SMG}(i) + (-
P_max_charge_bat{SMG})

                    P_load{SMG}(i)=P_load{SMG}(i);
                    P_pv{SMG}(i)=P_max_pv{SMG};
                    P_bat{SMG}(i)=P_max_charge_bat{SMG};
                    P_grid{SMG}(i)=-P_pv{SMG}(i)+(-
P_bat{SMG}(i))+P_load{SMG}(i);

                    state{SMG}(i)=2;

                elseif P_load{SMG}(i) < P_max_pv{SMG}

                    P_load{SMG}(i)=P_load{SMG}(i);
                    P_pv{SMG}(i)=P_max_pv{SMG};
                    P_grid{SMG}(i)=0;
                    P_bat{SMG}(i)=-P_pv{SMG}(i)-
P_grid{SMG}(i)+P_load{SMG}(i);
```

```
                    state{SMG}(i)=3;

                elseif P_nom_load{SMG}(i) > P_max_pv{SMG}  &&
P_nom_load{SMG}(i) < (P_max_pv{SMG} + P_max_discharge_bat{SMG}) &&
P_max_pv{SMG} > 0 && SoC{SMG}(i) > min_SoC(SMG) && (P_max_pv{SMG} -
P_nom_load{SMG}(i)) <= (-P_max_charge_bat{SMG})

                    P_load{SMG}(i)=P_nom_load{SMG}(i);
                    P_pv{SMG}(i)=P_max_pv{SMG};
                    P_grid{SMG}(i)=0;
                    P_bat{SMG}(i)=-P_pv{SMG}(i)+P_load{SMG}(i);

                    if -
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i)==0
                        ok{SMG}(i)=1;
                        unbalance{SMG}(i)=-
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i);
                    else
                        ok{SMG}(i)=0;
                        unbalance{SMG}(i)=-
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i);
                    end

                    state{SMG}(i)=8;

                elseif P_load{SMG}(i) < P_max_grid{SMG} + P_max_pv{SMG}

                    P_load{SMG}(i)=P_load{SMG}(i);
                    P_pv{SMG}(i)=P_max_pv{SMG};
                    P_grid{SMG}(i)=P_load{SMG}(i)-P_pv{SMG}(i);

                    if SoC{SMG}(i)<SoC_grid(SMG) && P_max_pv{SMG}>0 &&
grid_purchasing_price{SMG}<=price

                        if (P_max_grid{SMG}-P_grid{SMG}(i))>=-
P_max_charge_bat{SMG}
                            P_bat{SMG}(i)=P_max_charge_bat{SMG}/2;
                        else
                            P_bat{SMG}(i)=-(P_max_grid{SMG}-
P_grid{SMG}(i))/2;
                        end

                    elseif SoC{SMG}(i)<SoC_grid(SMG) && P_max_pv{SMG}==0
&& grid_purchasing_price{SMG}<=price
                        if (P_max_grid{SMG}-P_grid{SMG}(i))>=-
P_max_charge_bat{SMG}
                            P_bat{SMG}(i)=P_max_charge_bat{SMG};
                        else
                            P_bat{SMG}(i)=-(P_max_grid{SMG}-
P_grid{SMG}(i));
                        end

                    elseif SoC{SMG}(i)>SoC_grid(SMG)
                        P_bat{SMG}(i)=0;
```

```
                    end
                    P_grid{SMG}(i)=P_grid{SMG}(i)+(-P_bat{SMG}(i));
                    state{SMG}(i)=4;

                elseif (P_load{SMG}(i) < P_max_grid{SMG} + P_max_pv{SMG}
+ P_max_discharge_bat{SMG}) && SoC{SMG}(i) > min_SoC(SMG)

                    P_load{SMG}(i)=P_load{SMG}(i);
                    P_pv{SMG}(i)=P_max_pv{SMG};
                    P_grid{SMG}(i)=P_max_grid{SMG};
                    P_bat{SMG}(i)=-P_pv{SMG}(i)-
P_grid{SMG}(i)+P_load{SMG}(i);

                    state{SMG}(i)=5;
                end

            if -
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i)==0
                ok{SMG}(i)=1;
                unbalance{SMG}(i)=-
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i);
            else
                ok{SMG}(i)=0;
                unbalance{SMG}(i)=-
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i);
            end

            state{SMG}(i)=6;

        elseif (P_max_grid{SMG} + P_max_pv{SMG}  == 0) && SoC{SMG}(i) <=
min_SoC(SMG)

            P_load{SMG}(i)=0;
            P_pv{SMG}(i)=0;
            P_grid{SMG}(i)=0;
            P_bat{SMG}(i)=0;

            if -
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i)==0
                ok{SMG}(i)=1;
                unbalance{SMG}(i)=-
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i);
            else
                ok{SMG}(i)=0;
                unbalance{SMG}(i)=-
P_load{SMG}(i)+P_pv{SMG}(i)+P_bat{SMG}(i)+P_grid{SMG}(i);
            end

            state{SMG}(i)=7;
        end


        %Step 7 and 8 and 9
```

```
[bat_power{SMG}(i),bat_autonomy{SMG}(i)]=battery(time,SMG,2,P_bat{SMG}(i)
);

        %Variable Costs
            % Costs of grid energy
            if P_grid{SMG}(i)>0

Grid_cost{SMG}(i)=P_grid{SMG}(i)/0.95*time_steps*24*grid_purchasing_price
{SMG};
            else
                Grid_cost{SMG}(i)=0;
            end
            % Income from selling energy to the grid
            if P_grid{SMG}(i)<0

Grid_income{SMG}(i)=P_grid{SMG}(i)*time_steps*24*grid_selling_price{SMG};
            else
                Grid_income{SMG}(i)=0;
            end
    end

    time_vector(:,i)=time;
    waitbar(i/ies,waitb)
end
toc
close(waitb)
points=8;
x_labels
={datestr(time_vector(1:round(length(time_vector)/points):length(time_vec
tor)),'dd/mm/yyyy HH:MM:SS')};
period=(final_simulation_time-initial_simulation_time)/365;
```

1.16. Interesting plots for the algorithm

```
%% Battery power
nbins=50;
clear categ_bat_power
for SMG=1:n_SMG

    figure

    histogram(P_bat{SMG}(P_bat{SMG}>0),nbins);
    title(strcat('Power released by the batteries for SMG',num2str(SMG),'
in',num2str(period*365),' days'))
    xlabel('Power (kW)')
    ylabel('Frequency')
end

for SMG=1:n_SMG
    mean_bat_power(SMG)=mean(P_bat{SMG}(P_bat{SMG}>0));
    categ_bat_power (SMG) = categorical({strcat('SMG',num2str(SMG),'
total of ',num2str(length(P_bat{SMG}(P_bat{SMG}>0))),' power points')});
```

```
end
figure

bar(categ_bat_power,mean_bat_power)
ylabel('Power (kW)')

title(strcat('Mean batteries power for every SMG during
',num2str(period*365),' days'))

%% States

for SMG=1:n_SMG

    C_states = categorical(state{SMG},[1:8]);
    figure
    h = histogram(C_states,'BarWidth',0.5);
    title(strcat('State for SMG',num2str(SMG),' in',num2str(period*365),'
days'))
    ylabel('Frequency')
    xlabel('State')
end

%% DoD
nbins=30;
for SMG=1:n_SMG

    SoC1{SMG}=[0,SoC{SMG}(1:length(SoC{SMG})),100];
    LMax= islocalmax(SoC1{SMG}).*SoC1{SMG};
    Lmin= islocalmin(SoC1{SMG}).*SoC1{SMG};
    LMax(LMax==0) = [];
    Lmin(Lmin==0) = [];
    vMax{SMG}=LMax;
    vmin{SMG}=Lmin;
    DoD{SMG}=vMax{SMG}-vmin{SMG};

    figure
    h = histogram(DoD{SMG},nbins);
    title(strcat('DoD of batteries for SMG',num2str(SMG),'
in',num2str(period*365),' days'))
    ylabel('Frequency')
    xlabel('DoD (%)')
end

clear categ

for SMG=1:n_SMG
    mean_DoD(SMG)=mean(DoD{SMG});
    categ (SMG) = categorical({strcat('SMG',num2str(SMG),' total of
',num2str(length(DoD{SMG})),' cycles')});
end
figure

bar(categ,mean_DoD)
```

```
title(strcat('Mean DoD for every SMG during ',num2str(period*365),'
days'))


life_cycles=[25000,15000,8000,5250,3800,2900,2250,1850,1500];
life_DoD=[0,10,20,30,40,50,60,70,80];

for SMG=1:n_SMG
    k{SMG}=1./interp1(life_DoD,life_cycles,DoD{SMG});
    mean_w_DoD(SMG)=sum(DoD{SMG}.*k{SMG})/sum(k{SMG});
end
figure

bar(categ,mean_w_DoD)

title(strcat('Proportional mean DoD for every SMG during
',num2str(period*365),' days'))

%% Economic analysis
Elements_name{1}={'PV array + inverter + all','Battery','Inv/Charger'};
Elements_name{2}={'PV array + inverter + all','Battery','Inv/Charger'};
Elements_name{3}={};

Elements_cost{1}=[4750,2400,1500,2000];
Elements_cost{2}=[6100,2400,1500,2000];
Elements_cost{3}=[4750,2400,1500,2000];
Elements_cost{4}=[6100,2400,1500,2000];
Elements_cost{5}=[];

Elements_life{1}=[30,24,20,30]; %years
Elements_life{2}=[30,9,20,30]; %years
Elements_life{3}=[30,24,20,30]; %years
Elements_life{4}=[30,9,20,30]; %years
Elements_life{5}=[]; %years

Power_price=38; %€/kW/year
Contracted_power=[1.75,1.75,1.75,1.75,5];

bat_Power_price=6; %€/kW/year
bat_Contracted_power=[2.4,2.4,2.4,2.4,0];

for SMG=1:n_SMG

Fix_element_costs{SMG}=(Elements_cost{SMG}./Elements_life{SMG}).*period;
    Total_fix_costs(SMG)=sum(Fix_element_costs{SMG});

Power_costs(SMG)=(Contracted_power(SMG)*Power_price+bat_Contracted_power(
SMG)*bat_Power_price)*period;
    Energy_costs(SMG)=sum(Grid_cost{SMG});
    Energy_income(SMG)=sum(Grid_income{SMG});
    Taxes_costs(SMG)=(Power_costs(SMG)+Energy_costs(SMG))*(0.05113+0.21);

end
figure
c = categorical({'SMG1','SMG2','SMG3','SMG4','SMG5'});
```

```
costs=[Total_fix_costs(:),Power_costs(:),Energy_costs(:),Taxes_costs(:),E
nergy_income(:)];
bar(c,costs,'stacked')

legend('Total fix costs','Power costs','Energy costs','Taxes
costs','Energy income')
title(strcat('SMG costs for ',num2str(period*365),' days'))

%% Energetic mix analysis
clear E_pv_used E_pv_togrid E_grid c
for SMG=1:n_SMG
    E_pv_used(SMG)=sum(P_pv{SMG}.*time_steps.*24)+sum(P_grid{SMG}(
P_grid{SMG}<0 ).*time_steps.*24);
    E_pv_togrid(SMG)=-sum(P_grid{SMG}( P_grid{SMG}<0 ).*time_steps.*24);
    E_grid(SMG)=sum(P_grid{SMG}( P_grid{SMG}>=0 ).*time_steps.*24);

end
c_mix = categorical({'SMG1','SMG2','SMG3','SMG4','SMG5'});
%,'SMG2','SMG3'
E_mix=[E_pv_used(:),E_grid(:),E_pv_togrid(:)];
bar(c_mix,E_mix,'stacked')
ylabel('kWh')
legend('Total used PV energy','Total used grid energy','Total not used PV
energy')
title(strcat('SMG mix production for ',num2str(period*365),' days'))

figure

for SMG=1:n_SMG
    a=P_bat{SMG}<0;
    b=P_pv{SMG}==0;
    E_bat_grid(SMG)=-sum((a.*b).*P_bat{SMG}.*time_steps.*24);
    E_bat_pv(SMG)=-sum(P_bat{SMG}( P_bat{SMG}<0 ).*time_steps.*24)-
E_bat_grid(SMG);

end
c_bat_energy = categorical({'SMG1','SMG2','SMG3','SMG4','SMG5'});
%,'SMG2','SMG3'
E_bat=[E_bat_grid(:),E_bat_pv(:)];
bar(c_bat_energy,E_bat,'stacked')
ylabel('kWh')
legend('Energy from grid','Energy from PV')
title(strcat('Battery energy origin for ',num2str(period*365),' days'))

figure
kg_kWh_grid=0.245;
for SMG=1:n_SMG

    Emissions(SMG)=sum(P_grid{SMG}( P_grid{SMG}>0
).*time_steps.*24)*kg_kWh_grid;
    saved_Emissions(SMG)=sum(P_grid{SMG}( P_grid{SMG}<0
).*time_steps.*24)*kg_kWh_grid;

end
c_Emissions = categorical({'SMG1','SMG2','SMG3','SMG4','SMG5'});
%,'SMG2','SMG3'
```

```
Emiss=[Emissions(:),saved_Emissions(:)];
bar(c_Emissions,Emiss,'stacked')
legend('Emission from grid consumption','Emission saved from energy
injection')
ylabel('kg CO_2')
title(strcat('Emissions produced by each SMG during
',num2str(period*365),' days'))

%% Energetic demand analysis
period=(final_simulation_time-initial_simulation_time)/365;
for SMG=1:n_SMG
    E_load(SMG)=sum(P_load{SMG}.*time_steps.*24);
end
figure
c_demand = categorical({'SMG1','SMG2'});
demand=[E_load(:)];
bar(c_demand,demand)

legend('Total demand')
title(strcat('SMG demand for ',num2str(period*365),' days'))

%% Time power plots

for SMG=1:n_SMG
    figure
    plot(1:ies,P_pv{SMG})
    hold on
    plot(1:ies,P_grid{SMG})
    plot(1:ies,P_bat{SMG})
    plot(1:ies,P_load{SMG})
    plot(1:ies,state{SMG})
    title(strcat('Power of SMG',num2str(SMG)))
    legend('PV','Grid','Battery','Loads','state')
    ylabel('Power (kW)')
    xlabel('Time')
    set(gca,
'Xtick',time_vector(1:round(length(time_vector)/points):length(time_vecto
r)),'XTickLabel',x_labels);
    xtickangle(45)
    hold off
end

%% Plotting inverters efficiency curve

inverter_efficiency_curve=getappdata(0,'pv_inverter_efficiency_curve');
%(1,:)Efficiency vector in %, (2,:)Output/nominal power vector in %
P_nom=getappdata(0,'nominal_power_inverter');
inverter_efficiency_curve=getappdata(0,'pv_inverter_efficiency_curve');
%(1,:)Efficiency vector in %, (2,:)Output/nominal power vector in %

figure
for SMG=1:n_SMG
plot(inverter_efficiency_curve{SMG}(2,:),inverter_efficiency_curve{SMG}(1
,:))
hold on
end
```

```
legend('SMG1','SMG2','SMG3')
ylabel('Efficiency (%)')
xlabel('P_p_v/P_i_n_v_e_r_t_e_r')

%% Plotting the batteries
figure
for SMG=1:n_SMG

    plot(1:ies,P_bat{SMG})
    hold on
end

title('Batteries power output')
ylabel('Power (kW)')
xlabel('Time')
legend('SMG1','SMG2')
set(gca,
'Xtick',time_vector(1:round(length(time_vector)/points):length(time_vecto
r)),'XTickLabel',x_labels);
xtickangle(45)

figure
for SMG=1:n_SMG

    plot(1:ies,SoC{SMG})
    hold on

end

title('Batteries SoC')
ylabel('SoC (%)')
xlabel('Time')
legend('SMG1','SMG2')
set(gca,
'Xtick',time_vector(1:round(length(time_vector)/points):length(time_vecto
r)),'XTickLabel',x_labels);
xtickangle(45)

figure
for SMG=1:n_SMG

    plot(1:ies,bat_autonomy{SMG})
    hold on

end

title('Batteries autonomy')
ylabel('Autonomy (h)')
xlabel('Time')
legend('SMG1','SMG2')
set(gca,
'Xtick',time_vector(1:round(length(time_vector)/points):length(time_vecto
r)),'XTickLabel',x_labels);
xtickangle(45)
```

1.17. Other plots

```
%% Economic analysis
Elements_name{1}={'PV array + inverter + all','Battery','Inv/Charger'};
Elements_name{2}={'PV array','Battery','PV Inverter','Inv/Charger'};
Elements_name{3}={};

Elements_cost{1}=[4750,2400,1500];
Elements_cost{2}=[8500,1500,5000,3000];
Elements_cost{3}=[];

Elements_life{1}=[30,18,20]; %years
Elements_life{2}=[30,30,18,20]; %years
Elements_life{3}=[]; %years

Power_price=38; %€/kW/year
Contracted_power=[1.75,1,5];

period=(final_simulation_time-initial_simulation_time)/365;
for SMG=1:n_SMG

Fix_element_costs{SMG}=(Elements_cost{SMG}./Elements_life{SMG}).*period;
    Total_fix_costs(SMG)=sum(Fix_element_costs{SMG});
    Power_costs(SMG)=Contracted_power(SMG)*Power_price*period;
    Energy_costs(SMG)=sum(Grid_cost{SMG});
    Energy_income(SMG)=sum(Grid_income{SMG});
    Taxes_costs(SMG)=(Power_costs(SMG)+Energy_costs(SMG))*(0.05113+0.21);

end
figure
c = categorical({'SMG1','SMG2','SMG3'});
costs=[Total_fix_costs(:),Power_costs(:),Energy_costs(:),Taxes_costs(:),E
nergy_income(:)];
bar(c,costs,'stacked')

legend('Total fix costs','Power costs','Energy costs','Taxes
costs','Energy income')
title(strcat('SMG costs for ',num2str(period*365),' days'))


%% Energetic mix analysis
period=(final_simulation_time-initial_simulation_time)/365;
for SMG=1:n_SMG
    E_pv(SMG)=sum(P_pv{SMG}.*time_steps.*24)+sum(P_grid{SMG}(
P_grid{SMG}<0 ).*time_steps.*24);
    E_grid(SMG)=sum(P_grid{SMG}( P_grid{SMG}>=0 ).*time_steps.*24);
end
c = categorical({'SMG1','SMG2','SMG3'});
costs=[E_pv(:),E_grid(:)];
bar(c,costs,'stacked')

legend('Total PV energy','Total grid energy')
```

```
title(strcat('SMG mix production for ',num2str(period*365),' days'))

%% Energetic demand analysis
period=(final_simulation_time-initial_simulation_time)/365;
for SMG=1:n_SMG
    E_load(SMG)=sum(P_load{SMG}.*time_steps.*24);
end
figure
c = categorical({'SMG1','SMG2','SMG3'});
costs=[E_load(:)];
bar(c,costs,'stacked')

legend('Total demand')
title(strcat('SMG demand for ',num2str(period*365),' days'))

%% Time power plots

for SMG=1:n_SMG
    figure
    plot(1:i,P_pv{SMG})
    hold on
    plot(1:i,P_grid{SMG})
    plot(1:i,P_bat{SMG})
    plot(1:i,P_load{SMG})
    plot(1:i,state{SMG})
    title(strcat('Power of SMG',num2str(SMG)))
    legend('PV','Grid','Battery','Loads','state')
    ylabel('Power (kW)')
    xlabel('Time')
    set(gca,
'Xtick',time_vector(1:round(length(time_vector)/points):length(time_vecto
r)),'XTickLabel',x_labels);
    xtickangle(45)
    hold off
end

%% Plotting irrandiance
figure
plot(time_vector,irradiance)
set(gca,
'Xtick',time_vector(1:round(length(time_vector)/points):length(time_vecto
r)),'XTickLabel',x_labels);
xtickangle(45)
ylabel('Irradiance (W/m^2)')
hold on
plot(time_vector,irradiance_f)
legend('SMG1','SMG2','SMG3','forecast SMG1','forecast SMG2','forecast
SMG3')

%% Plotting temperature
figure
plot(time_vector,temperature)
set(gca,
'Xtick',time_vector(1:round(length(time_vector)/points):length(time_vecto
r)),'XTickLabel',x_labels);
xtickangle(45)
```

```
ylabel('Temperature (°C)')
%hold on
%plot(time_vector,temperature_f)
legend('SMG1','SMG2','SMG3')%,'forecast SMG1','forecast SMG2'

%% Ploting PV
for SMG=1:n_SMG
plot(time_vector,P_pv{SMG})
hold on
end
ylabel('Output power (kW)')
xlabel('Time')
set(gca,
'Xtick',time_vector(1:round(length(time_vector)/points):length(time_vecto
r)),'XTickLabel',x_labels);
xtickangle(45)
title('Maximum power from PV inverter')
legend('SMG1 1.5 kW','SMG2 1.5 kW','SMG3 0 kW')

%% Plotting inverters efficiency curve

inverter_efficiency_curve=getappdata(0,'pv_inverter_efficiency_curve');
%(1,:)Efficiency vector in %, (2,:)Output/nominal power vector in %
P_nom=getappdata(0,'nominal_power_inverter');
inverter_efficiency_curve=getappdata(0,'pv_inverter_efficiency_curve');
%(1,:)Efficiency vector in %, (2,:)Output/nominal power vector in %

figure
for SMG=1:n_SMG
plot(inverter_efficiency_curve{SMG}(2,:),inverter_efficiency_curve{SMG}(1
,:))
hold on
end

legend('SMG1','SMG2','SMG3')
ylabel('Efficiency (%)')
xlabel('P_p_v/P_i_n_v_e_r_t_e_r')

%% 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

%% Plotting the batteries
figure
for SMG=1:n_SMG

    plot(1:i,P_bat{SMG})
    hold on
    plot(1:i,SoC{SMG}/100)
end

title('Batteries power output')
ylabel('Power (kW)')
xlabel('Time')
legend('SMG1','SMG2')
set(gca,
'Xtick',time_vector(1:round(length(time_vector)/points):length(time_vecto
r)),'XTickLabel',x_labels);
xtickangle(45)

figure
for SMG=1:n_SMG

    plot(1:i,SoC{SMG})
    hold on

end

title('Batteries SoC')
ylabel('SoC (%)')
xlabel('Time')
legend('SMG1','SMG2')
set(gca,
'Xtick',time_vector(1:round(length(time_vector)/points):length(time_vecto
r)),'XTickLabel',x_labels);
xtickangle(45)

figure
for SMG=1:n_SMG
```

```
    plot(1:i,bat_autonomy{SMG})
    hold on

end

title('Batteries autonomy')
ylabel('Autonomy (h)')
xlabel('Time')
legend('SMG1','SMG2')
set(gca,
'Xtick',time_vector(1:round(length(time_vector)/points):length(time_vecto
r)),'XTickLabel',x_labels);
xtickangle(45)

%% Plotting the genset
figure
for SMG=1:n_SMG

    plot(time_vector,genset_power{SMG})
    hold on

end

title('Genset power output')
ylabel('Power (kW)')
xlabel('Time')
legend('SMG1','SMG2','SMG3')
set(gca,
'Xtick',time_vector(1:round(length(time_vector)/points):length(time_vecto
r)),'XTickLabel',x_labels);
xtickangle(45)

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_vecto
r)),'XTickLabel',x_labels);
xtickangle(45)

figure
for SMG=1:n_SMG

    plot(time_vector,genset_state{SMG})
    hold on
```

```
end

title('Genset state')
ylabel('State')
xlabel('Time')
legend('SMG1','SMG2','SMG3')
set(gca,
'Xtick',time_vector(1:round(length(time_vector)/points):length(time_vecto
r)),'XTickLabel',x_labels);
xtickangle(45)

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','SMG3'});
bar(c,emissions)
title('Genset emissions')
ylabel('CO_2 emissions (kg)')

figure
c = categorical({'SMG1','SMG2','SMG3'});
bar(c,genset_costs)
title('Genset variable costs')
ylabel('Variable costs (€)')

figure
c = categorical({'SMG1','SMG2','SMG3'});
bar(c,genset_produced_energy)
title('Genset energy provided')
ylabel('Energy provided (kWh)')
%% Plotting the grid
for SMG=1:n_SMG
    figure
    plot(time_vector,P_max_grid{SMG})
    set(gca,
'Xtick',time_vector(1:round(length(time_vector)/points):length(time_vecto
r)),'XTickLabel',x_labels);
    xtickangle(45)
    title('Maximum power of the grid')
    ylabel('Maximum power (kW)')
    figure
    plot(time_vector,grid_selling_price{SMG})
    title(strcat('Selling prices, SMG',num2str(SMG)))
    ylabel('Price (€)')
    ylim([0.02 0.07])
    set(gca,
'Xtick',time_vector(1:round(length(time_vector)/points):length(time_vecto
r)),'XTickLabel',x_labels);
    xtickangle(45)
    figure
```

```
    plot(time_vector,grid_purchasing_price{SMG})
    title(strcat('Purchasing prices, SMG',num2str(SMG)))
    ylabel('Price (€)')
    ylim([0.0 0.2])
    set(gca,
'Xtick',time_vector(1:round(length(time_vector)/points):length(time_vecto
r)),'XTickLabel',x_labels);
    xtickangle(45)
end
%% Plotting the load curves

for SMG=1:n_SMG

    figure
    plot(time_vector,min_power_loads{SMG})
    legend(char(load_name{SMG}))
    title(strcat('Separatelly minimum load curves, SMG',num2str(SMG)))
    set(gca,
'Xtick',time_vector(1:round(length(time_vector)/points):length(time_vecto
r)),'XTickLabel',x_labels);
    xtickangle(45)
    ylabel('Load power (kW)')

    figure
    plot(time_vector,nom_power_loads{SMG})
    legend(char(load_name{SMG}))
    title(strcat('Separatelly nominal load curves, SMG',num2str(SMG)))
    set(gca,
'Xtick',time_vector(1:round(length(time_vector)/points):length(time_vecto
r)),'XTickLabel',x_labels);
    xtickangle(45)
    ylabel('Load power (kW)')

    figure
    plot(time_vector,sum(min_power_loads{SMG}))
    c
    plot(time_vector,sum(nom_power_loads{SMG}))
    legend('minimum','nominal')
    title(strcat('Load curves, SMG',num2str(SMG)))
    set(gca,
'Xtick',time_vector(1:round(length(time_vector)/points):length(time_vecto
r)),'XTickLabel',x_labels);
    xtickangle(45)
    ylabel('Load power (kW)')
end

for SMG=1:n_SMG


total_min_demand(SMG)=sum(sum(min_power_loads{SMG}).*(time_steps*24));

total_nom_demand(SMG)=sum(sum(nom_power_loads{SMG}).*(time_steps*24));

end
figure
c = categorical({'SMG1','SMG2','SMG3'});
```

```
bar(c,total_min_demand)
title('Total minimum energy demanded')
ylabel('Energy (kWh)')

figure
c = categorical({'SMG1','SMG2','SMG3'});
bar(c,total_nom_demand)

title('Total nominal energy demanded')
ylabel('Energy (kWh)')
```