****

*Smart system of renewable energy storage based on* ***IN****tegrated E****V****s and b****A****tteries to empower mobile,* ***D****istributed and centralised* ***E****nergy storage in the distribution grid*

|  |
| --- |
| Deliverable nº: **XXXXX**  Deliverable name: Optimal Flexibility Power Flow (DSO-FR)  Version: **1.0**  Release date: **13/03/2019**  Dissemination level: **Public** (Public, Confidential)  Status: **Draft** (Draft, Peer-reviewed, Submitted, Approved)  Author: **Íngrid Munné-Collado (UPC)** |

Table of contents

[Executive summary 5](#_Toc11679744)

[1 DSO case study 6](#_Toc11679745)

[1.1 Spanish case study 6](#_Toc11679746)

[1.2 Flowchart 15](#_Toc11679747)

[1.3 Communication scheme between agents in the FR calculation 16](#_Toc11679748)

Abbreviations and Acronyms

*List of abbreviations* ***only*** *related to this deliverable.*

*Please complete the following proposed one, deleting the unnecessary items.*

| **Acronym** | **Description** |
| --- | --- |
| CA | Consortium Agreement |
| DoA | Description of Action (annex I of the Grant Agreement) |
| EC | European Commission |
| GA | Grant Agreement |
| PC | Project Coordinator |
| PCC | Project Coordination Committee |
| PO | Project Officer |
| QM | Quality Management |
| TCC | Technical Coordination Committee |
| TL | Task Leader |
| ToC | Table of Contents |
| WP | Work Package |
| WPL | Work Package Leader |

# Executive summary

This document intends to compile all the work done in the period March-June 2019 concerning power flows and panda power tool.

Firstly, an explanation of panda power together with a comparison to Matpower is carried out. A preliminary overview of basic concepts before starting with the programs will be done.

# Panda Power vs. Matpower

## Principal differences

|  |  |
| --- | --- |
| **Pandapower** | **Matpower** |
| Loading percentage |  |
|  | Differenciates P/Q buses Generation/Load |
| Branch data power (from/to) one node  (sum of everything in one node) | Branch data power indicates  from bus #/to bus # |
|  | Min/Max Voltage magnitude, angle,  P losses, Q losses |
| Impedance magnitudes/unit lenght | Impedances in p.u. |
| kV, MW, MVAr, | kV, MW, MVAr |

Table 1: Panda Power and Matpower main differences

In Pandapower the SLACK bus needs to be implemented as the external grid.

“External grids represent the higher level power grid connection and are modelled as the slack bus in the power flow calculation.” – *Pandapower documentation*

Also in the external grid we can add the 0 degree reference, whereas in the bus design (if we try to implement a slack bus as a regular kind of bus) it’s not possible to do so.

From matpower to pandapower:

**Matpower (p.u.)** **Pandapower (units/km)**

Base values

Input values pandapower:

# Methodology and steps to build and run a grid

## Kjnkjnkjh

Sets

|  |  |
| --- | --- |
| K | Set of nodes of the MV distribution grid |
| L | Set of lines of the MV distribution grid |
| T | Set of time periods in the optimization problem |

Parameters

|  |  |
| --- | --- |
|  | Bus nominal voltage value of the nodes and [S] |
|  | Bus transversal conductance value between nodes and [S] |
|  | Current at node in period [A] |
|  | Maximum allowed current at line |
|  | Total active power at node in period [kW] |
|  | Total reactive power at node in period [kW] |
|  | Apparent power at node in period [kVA] |
|  | Voltage at node in period [kV] |
|  | Admittance value for line [S] |
|  | Transversal admittance value for line , first component [S]. |
|  | Transversal admittance value for line , second component [S] |
|  | Voltage angle difference between nodes in period [rad] |
|  | Voltage angle at node in period [rad] |
|  | Minimum allowed voltage angle at nodein period [rad] |
|  | Maximum allowed voltage angle at nodein period [rad] |
|  | Impedance value for line [Ω] |

Variables

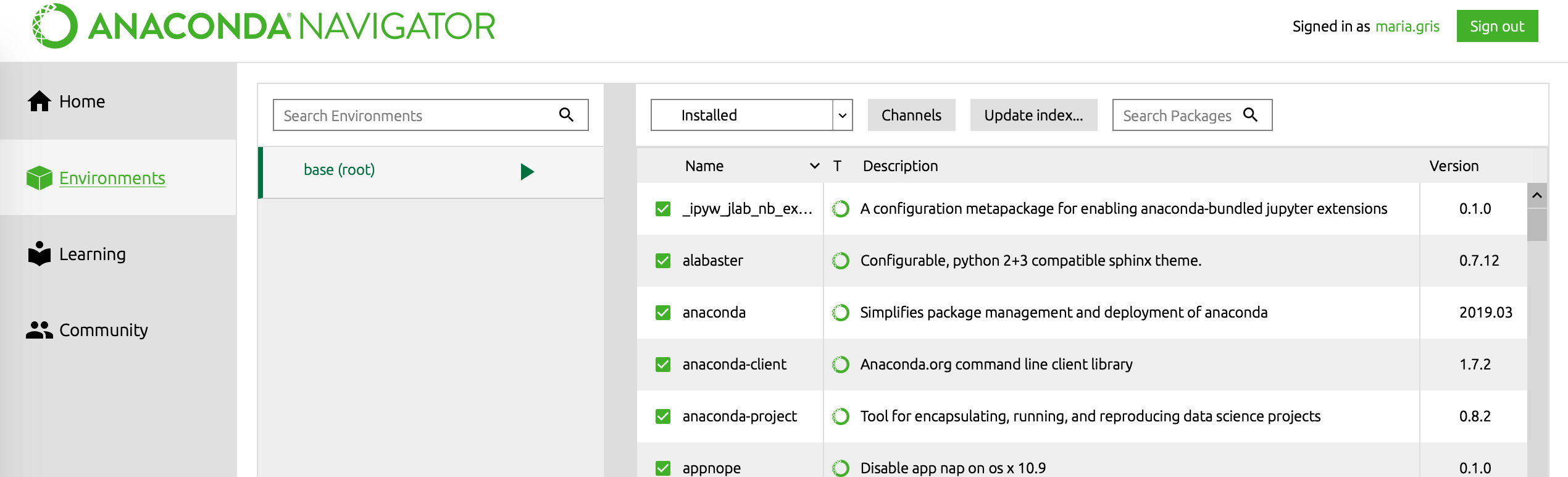
|  |  |
| --- | --- |
|  | Voltage absolute value or modulus at node i and time period t [pu] |
|  | Voltage angle at node i and time period t [rad] |
|  | Total active power at node i and time period t [pu?] |
|  | Total reactive power at node i and time period t [pu?] |

Segons el que posi ja acabaré de concreter.

**2.1. Previous concepts and prerequisites**

Before starting, we need to install Anaconda navigator from the Anaconda Distributor, from the following link: <https://www.anaconda.com/distribution/>

Download the appropriate version according to your Python one. However, it is strongly recommended to also download the newest Python version in case you don’t have the latest one. Make sure you have all the packages installed in your working environment:



If you need a new one, just install it by searching in the “search packages” box, selecting it and clicking in the “Apply” button that will appear in the bottom right part of the window.

Next, to install Pandapower write the following command in the computer terminal:

**pip install pandapower**

In order to plot the grid later, igraph must be installed similarly:

**conda install -c conda-forge python-igraph**

Not sure if we need to import plotting(sometimes it is required but sometimes no). Also, occasionally it takes two attempts to draw the graph properly, as it was too slowly to run the script properly. If needed we will import plotting command as follows:

**from pandapower import plotting**

This command uses internally the igraph library, so we do not have to import igraph.

**2.2. Grid**

**2.2.1. Grid structure and power flow**

First of all, we need to learn how to create a grid with all necessary elements and run the power flow. Afterwards, we will create a loop and iterate over the loads in the given json file.

The steps to build the desired grid, run the corresponding power flow, plot the network and save the overload lines are explained in this section:

**Import libraries:**

*import pandas as pd*

*import os from pathlib import Path*

*import pandapower as pp from pandapower*

*from pandapower import plotting*

1. **Import the excel** file and make it readable for any working directory. **Import json file** for the loads

*file = 'Sitel\_Invade\_MV\_Topology.xlsx'*

*xl = pd.ExcelFile(Path(str(os.getcwd()) + '/'+ file))*

*dfload = pd.read\_json(Path(str(os.getcwd()) +'/ofpfs\_sent.json', orient='rows'))*

1. Create an **empty network**.

*net = pp.create\_empty\_network()*

1. Generate a **Data Frame** with all the **line data** available in the excel file.

*MVNetwork = xl.parse('Linies')*

1. **Assign** the appropriate data to the variables **X, C, R**.

*x\_ohm\_per\_km = MVNetwork['Reactancia\_ohm\_km'][0]*

*c\_nf\_per\_km = MVNetwork['Capacitat\_uF\_km'][0]*

*r\_ohm\_per\_km = MVNetwork['Resistencia\_ohm\_km'][0]*

1. If the **line type** is **not defined** as a standard type, **create a new standard type** for your own line data.

*line\_data = {"c\_nf\_per\_km": c\_nf\_per\_km, "r\_ohm\_per\_km": r\_ohm\_per\_km, "x\_ohm\_per\_km": x\_ohm\_per\_km, "max\_i\_ka":0.415}*

*pp.create\_std\_type(net, line\_data, "line\_ESTABANELL", element='line')*

1. Generate a **Data Frame** with all the **busses** in the excel file (also those at the LV part of the TRAFOS, which can be found in the excel TRAFOS page.

*MVNetworkbusses = xl.parse('Busses')*

*MVNetworkbussesTrafos = xl.parse(‘Trafos’)*

1. **Create the busses** (stored in the previous data frames) with its **voltage**, **origin-end**, **name** and indicating the **network where they belong** to. We can do this by iterating in the data frame [‘name’] created with the excel file.

*for i in MVNetworkbusses['name']:*

*pp.create\_bus(net, vn\_kv=20.5,name=i, max\_vm\_pu=1.1, min\_vm\_pu=0.9)*

*for i in range(len(MVNetworkbussesTrafos['name'])):*

*pp.create\_bus(net, vn\_kv= MVNetworkbussesTrafos['vn\_kv'][i], name=MVNetworkbussesTrafos['name'][i], max\_vm\_pu=1.1, min\_vm\_pu=0.9)*

1. The **busses data frame can be edited** and we could include any other columns we may need, for example an identifier for the busses (all, including LV).

Previously, in order to get the IDs both from the joints at MV and the knots at the LV part of the transformers, we’ll create a common data frame that will later become a list to be able to add it in the data bus data frame as an additional column:

*l=MVNetworkbusses['Id Bus'] #Data Frame*

*r=l.append(MVNetworkbussesTrafos['Id Bus']) # r still a Data Frame*

*r=r.values.tolist() # r list*

We add in the net.bus (a pandas.core.frame.DataFrame type) a column for the Id Bus. Then, we can set the Id Bus column as the index of the net.bus and decide whether or not we keep the Id Bus column to be able to access to it\*.

\*In a data frame we cannot access the index as a number. We first need to indicate which column we want to pick and then access to the [i] item of that column.

*net.bus.insert(5, 'Id Bus', r)*

*net.bus = net.bus.set\_index(‘Id Bus’, drop = False)*

1. The **identifiers** used for the **busses** are:
   1. Transformer nodes: the one given
   2. Other nodes: 200-226 (own generated code to identify the non-transformers busses)
   3. Both identifiers in the “Busses” shit in the excel file.
2. **Create trafos** between the MV and LV busses with the parameters provided by Estabanell and also the standard type most similar to our voltages and power.

*pp.create\_transformer\_from\_parameters(net, hv\_bus=202, lv\_bus=75, sn\_mva=0.4, vn\_hv\_kv=20.5, vn\_lv\_kv=0.23, vk\_percent=4.,* ***vkr\_percent=0.04****, pfe\_kw=0.75, i0\_percent=1.8, shift\_degree=150, in\_service=True, parallel=1, name='E.T. NODE 1', tap\_side='hv', tap\_neutral=0, tap\_min=-2, tap\_max=2, tap\_step\_percent=2.5, tap\_step\_degree=0, tap\_phase\_shifter=False, df=1, index=75)*

Pay attention to the units, as may change depending on the program version.

1. **Create the lines** by selecting the **start and end node** and also its **standard type** already defined. Moreover, at this point, the length and the name of the line can be set.
   1. **Length:** MVNetwork['Length\_km'][i]
   2. **Name:** bus origin
   3. **Line identifier:** Id TedisNet

*for i in range(len(MVNetwork['Id TedisNet'])):*

*pp.create\_line(net, from\_bus=MVNetworkbusses['Id Bus'][i], to\_bus=MVNetworkbusse ['Id Bus'][i+1], length\_km=MVNetwork['Length\_km'][i], std\_type="line\_ESTABANELL", name= MVNetwork['origen'] [i])*

Add the line identifier but without making it become the line index:

*net.line.insert(14, 'Id linia', MVNetwork['Id TedisNet'])*

1. We need to define an **external grid**, which will take the place of **one of the existing busses**, making it **become the Slack bus**.

*pp.create\_ext\_grid(net, bus=201, vm\_pu=1.0, va\_degree=0.0, in\_service=True, name="Grid Connection")*

1. **Create loads** from the json file first iteration:

To understand the structure of the json file and where the inputs, outputs and simulation are, we will create different data frames.

*dfload\_output0= dfload['output'][0]*

*dfload\_output0\_input= dfload['output'][0]['input']*

*dfload\_output0\_simulation= dfload['output'][0]['simulation']*

*dfload\_output0\_timestamp= dfload['output'][0]['timestamp']*

We will create an empty list where we will append the IDs from the busses that have a load [1]. The loads then will be created from the data in the first iteration input [2]. Also, free busses will be assigned a 0 power load [3].

*[1] l=[ ]*

*[2] for i in range(len(dfload['output'][0]['input'])):*

*pp.create\_load(net, bus=dfload['output'][0]['input'][i]['IdTedisNet'], p\_mw = (dfload['output'][0] ['input'][i]['ActivePower'])/1000, q\_mvar = (dfload['output'][0]['input'][i]['ReactivePower'])/1000)*

*l.append(dfload['output'][0]['input'][i]['IdTedisNet'])*

*[3] for i in net.bus['Id Bus']:*

*if i not in l:*

*pp.create\_load(net, bus=i, p\_mw=0)*

Pay attention to units again, as you may be given loads in “kilo” and pandapower request them in “Mega”.

1. **Running the net:**

There are many algorithms to solve power flow problems. The following algorithms are available:

**“nr”** Newton-Raphson (pypower implementation with numba accelerations)

**“iwamoto\_nr”** Newton-Raphson with Iwamoto multiplier (maybe slower than NR but more robust)

**“bfsw”** backward/forward sweep (specially suited for radial and weakly-meshed networks)

**“gs”** gauss-seidel (pypower implementation)

**“fdbx”** fast-decoupled (pypower implementation)

**“fdxb”** fast-decoupled (pypower implementation)

The algorithm chosen is **bfsw**, our grid is radial and weakly meshed.

*pp.runpp(net, algorithm='bfsw')*

1. **Plotting the net:**

To plot the network built, the next steps need to be followed:

*[1] conda install -c conda-forge python-igraph*

*import igraph*

*[2] pp.plotting.simple\_plot(net) necessary!*

1. **Congestion in the grid:**

We will add the bus origin to the data frame of line results and make it its index column for both line results data frame and line data frame.

*net.res\_line.insert (14, 'from\_bus', net.line['from\_bus'])*

*net.res\_line = net.res\_line.set\_index('from\_bus', drop = False)*

*net.line = net.line.set\_index("from\_bus", drop = True)*

Moreover, a dictionary will be initialized to store the overloaded lines, which will be outlined by a maximum load percentage (modified as wished). The parameters kept will be the loading percentage of the line, the origin bus name and the bus ID:

*lines\_overloaded={}*

*for i in net.res\_line['from\_bus']:*

*if net.res\_line['loading\_percent'][i]>2: 🡪* ***[2 it’s just an example]***

*a=net.bus['name'][i]*

*b=net.bus['Id Bus'][i]*

*c=net.res\_line['loading\_percent'][i]*

*lines\_overloaded[(net.line['Id linia'][i])]=(a,b,c)*

Convert the dictionary in a data frame:

*df\_lines\_overloaded = pd.DataFrame.from\_dict(lines\_overloaded, orient='index')*

We need to overwrite the variable when renaming it. If we don’t do so, we won’t be able to refer / access it.

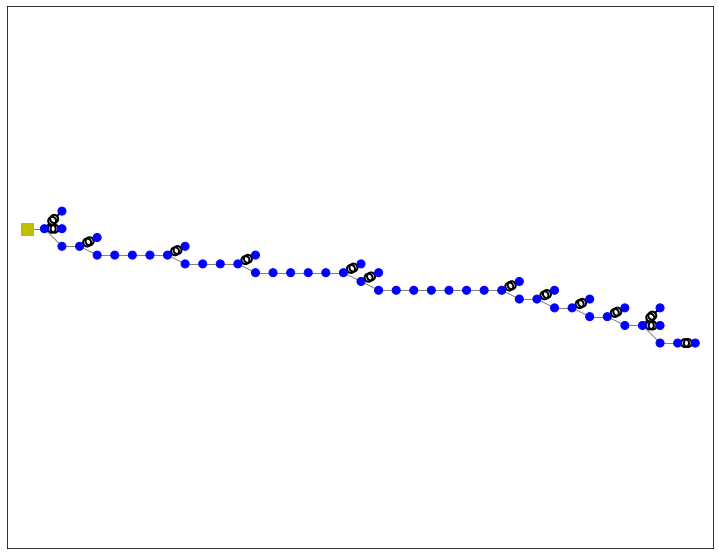
*df\_lines\_overloaded=df\_lines\_overloaded.rename({0:'From Bus', 1:'Id Bus', 2:'overload\_percentatge'}, axis='columns')*

1. **Create excel file:** from the data frame obtained we will provide Estabanell with an excel file only containing the origin bus and the ID of the overloaded line. Thus, we will exclude the Bus ID.

*df\_lines\_overloaded\_ESTABANELL=df\_lines\_overloaded*

*del(df\_lines\_overloaded\_ESTABANELL['Id Bus'])*

*df\_lines\_overloaded.to\_excel(Path(str(os.getcwd()) +'/PowerFlow.xlsx'))*



**2.2.2. Grid fuction**

Once the grid is built, we can proceed to iterate over the different time periods. In order to do so, we need an empty network for each iteration as we would accumulate all time period loads.

As the previous code is quite extended to write it every time we need it in a loop, the solution considered as the most suitable is to generate a function with it.

By creating such function, we will obtain a new and empty grid each time we iterate over the loads json file. We can use this method because the grid structure is the same and does not vary for any iteration.

The grid function would be a follows:

*def grid():*

*net = pp.create\_empty\_network()*

*file = 'Sitel\_Invade\_MV\_Topology.xlsx'*

*xl = pd.ExcelFile(Path(str(os.getcwd()) + '/'+ file))*

*MVNetwork = xl.parse('Linies')*

*x\_ohm\_per\_km = MVNetwork['Reactancia\_ohm\_km'][0]*

*c\_nf\_per\_km = MVNetwork['Capacitat\_nF\_km'][0]*

*r\_ohm\_per\_km = MVNetwork['Resistencia\_ohm\_km'][0]*

*line\_data = {"c\_nf\_per\_km": c\_nf\_per\_km , "r\_ohm\_per\_km": r\_ohm\_per\_km, "x\_ohm\_per\_km": x\_ohm\_per\_km, "max\_i\_ka":0.415}*

*pp.create\_std\_type(net, line\_data, "line\_ESTABANELL", element='line')*

*MVNetworkbusses = xl.parse('Busses')*

*for i in MVNetworkbusses['name']:*

*pp.create\_bus(net, vn\_kv=20.5,name=i, max\_vm\_pu=1.1, min\_vm\_pu=0.9)*

*MVNetworkbussesTrafos = xl.parse('Trafos')*

*for i in range(len(MVNetworkbussesTrafos['name'])):*

*pp.create\_bus(net, vn\_kv= MVNetworkbussesTrafos['vn\_lv\_kv'][i], name=MVNetworkbussesTrafos['name'][i], max\_vm\_pu=1.1, min\_vm\_pu=0.9)*

*l=MVNetworkbusses['Id Bus']*

*r=l.append(MVNetworkbussesTrafos['Id Bus'])*

*r=r.values.tolist()*

*net.bus.insert(5, 'Id Bus', r)*

*net.bus = net.bus.set\_index("Id Bus", drop = False)*

*pp.create\_transformer\_from\_parameters(net, hv\_bus=202, lv\_bus=75, sn\_mva=0.4, vn\_hv\_kv=20.5, vn\_lv\_kv=0.23, vk\_percent=4., vkr\_percent=0.04, pfe\_kw=0.75,*

*i0\_percent=1.8, shift\_degree=150, in\_service=True, parallel=1, name='E.T. NODE 1', tap\_side='hv', tap\_pos=0, tap\_neutral=0, tap\_min=-2, tap\_max=2,*

*tap\_step\_percent=2.5, tap\_step\_degree=0, tap\_phase\_shifter=False, df=1, index=75)* ***(…) All trafos***

*for i in range(len(MVNetwork['Id TedisNet'])):*

*pp.create\_line(net, from\_bus=MVNetworkbusses['Id Bus'][i], to\_bus=MVNetworkbusses['Id Bus'][i+1],*

*length\_km=MVNetwork['Length\_km'][i], std\_type="line\_ESTABANELL", name=MVNetwork['origen'][i])*

*net.line.insert(14, 'Id linia', MVNetwork['Id TedisNet'])*

*pp.create\_ext\_grid(net, bus=201, vm\_pu=1.0, va\_degree=0.0, in\_service=True, name="Grid Connection")*

*return(net)*

**2.2.3. Iterative loop**

In this code all the structure necessary to define the grid is contained. However, the run is not run as loads are not defined yet. With this function we can now generate the iterative loop where the power flow will be run for each of them and the results will be saved.

*dataframe\_lines\_overloaded=pd.DataFrame()*

*dataframe\_loads=pd.DataFrame()*

*q=0*

*for i in dfload['output']:*

*power=i['input']*

*net=grid()*

*l=[]*

*for e in power:*

*pp.create\_load(net, bus=e['IdTedisNet'], p\_mw = (e['ActivePower'])/1000000, q\_mvar = (e['ReactivePower'])/1000000)*

*l.append(e['IdTedisNet'])*

*for n in net.bus['Id Bus']:*

*if n not in l:*

*pp.create\_load(net, bus=n, p\_mw=0, q\_mvar=0)*

*pp.runpp(net, algorithm='bfsw')*

*pp.plotting.simple\_plot(net)*

*lines\_overloaded=[ ]*

*for t in range(len(net.res\_line['loading\_percent'])):*

*if net.res\_line['loading\_percent'][t]>3:*

*lines\_overloaded.append((net.line['Idlinia'][t],net.res\_line['loading\_percent'][t]))*

*else:*

*lines\_overloaded.append('no congestio')*

*dataframe\_lines\_overloaded.insert(q,i['timestamp'],lines\_overloaded)*

*net.load = net.load.set\_index('bus', drop = False)*

*results\_loads=[ ]*

*if q==0:*

*dataframe\_loads.insert(0,'Id Bus',net.bus['Id Bus'])*

*dataframe\_loads = dataframe\_loads.set\_index('Id Bus', drop = False)*

*dataframe\_loads.insert(q,i['timestamp'],'')*

*for e in dataframe\_loads['Id Bus']: #net.load['p\_mw']:*

*dataframe\_loads[i['timestamp']][e]=net.load['p\_mw'][e]*

*q=q+1*

*print('Data i hora actuals:',pd.datetime.now())*

*print('Timestamp:',i['timestamp'])*

*globals()['net{}'.format(q)] = net*

*dataframe\_lines\_overloaded.to\_excel(Path(str(os.getcwd()) +'/PowerFlow.xlsx'))*

In the loop we call the grid function each iteration (*net=grid()*) and also the input load data corresponding to the “i” time period/iteration.

*for i in dfload['output']:*

*power=i['input']*

The loads are created for all the input data in *“i[‘input’]* and 0 value loads are generated for those buses without load assigned from the json.

At this point we can run the power flow and save the results desired.

*l=[]*

*for e in power:*

*pp.create\_load(net, bus=e['IdTedisNet'], p\_mw = (e['ActivePower'])/1000000, q\_mvar = (e['ReactivePower'])/1000000)*

*l.append(e['IdTedisNet'])*

*for n in net.bus['Id Bus']:*

*if n not in l:*

*pp.create\_load(net, bus=n, p\_mw=0, q\_mvar=0)*

To save the overloaded lines, we create an empty list for each iteration, which will be later filled with those lines overpassing the limit stablished (in this case 3%).

In order to save all time periods overloaded lines in the same data frame, one in each column, we need the length of the columns to be the same. To do so, we will add “no congestion” message if we do not have overload in that line. Whenever there is congestion, the Id and the overload percentage will be saved. The columns name will be the timestamp of each iteration so that we can easily identify them.

*lines\_overloaded=[ ]*

*for t in range(len(net.res\_line['loading\_percent'])):*

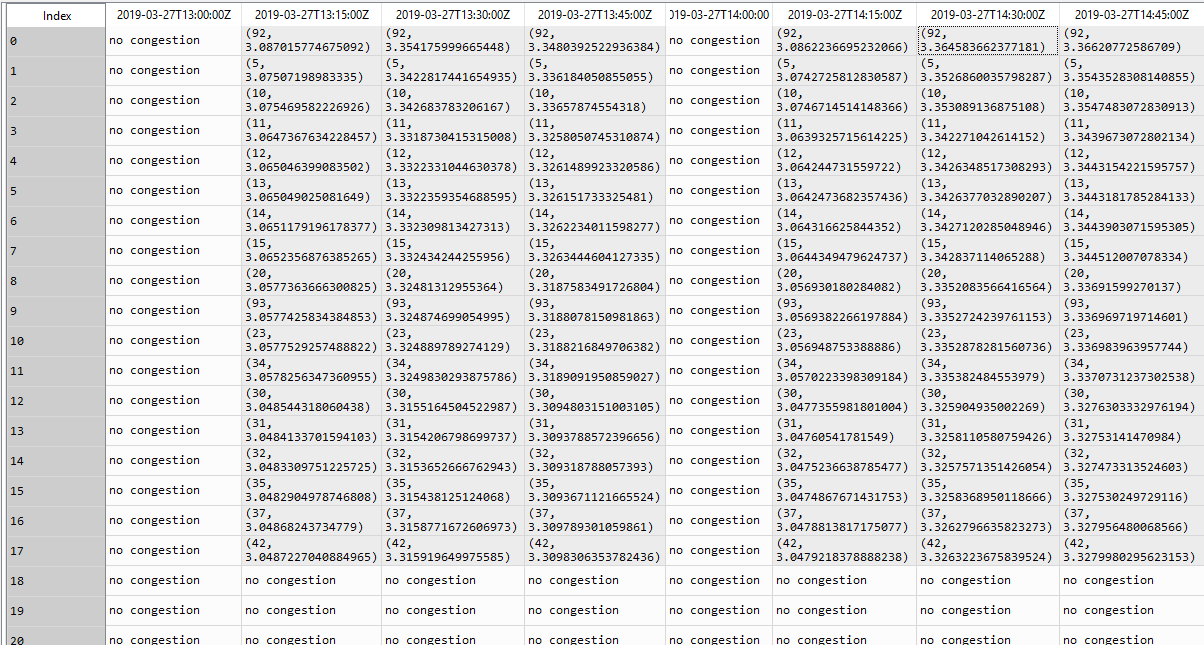
*if net.res\_line['loading\_percent'][t]>3:*

*lines\_overloaded.append((net.line['Idlinia'][t],net.res\_line['loading\_percent'][t]))*

*else:*

*lines\_overloaded.append('no congestion')*

*dataframe\_lines\_overloaded.insert(q,i['timestamp'],lines\_overloaded)*



Moreover, all loads will we saved with its timestamp in another Dataframe. As it can be observed, the “*Bus Id”* is set as the load index because the loads are specified for each bus. In the first iteration, we create a column with the Id Bus and set it as the index. Then, in each “q” iteration, we add a column with timestamp as title, in the position “q”, so that each column is created in its iteration position, respecting the time evolution.

After all this, an iteration over Id Bus can be carried out while adding the corresponding load, as each load is assigned to a bus.

*net.load = net.load.set\_index('bus', drop = False)*

*results\_loads=[]*

*if q==0:*

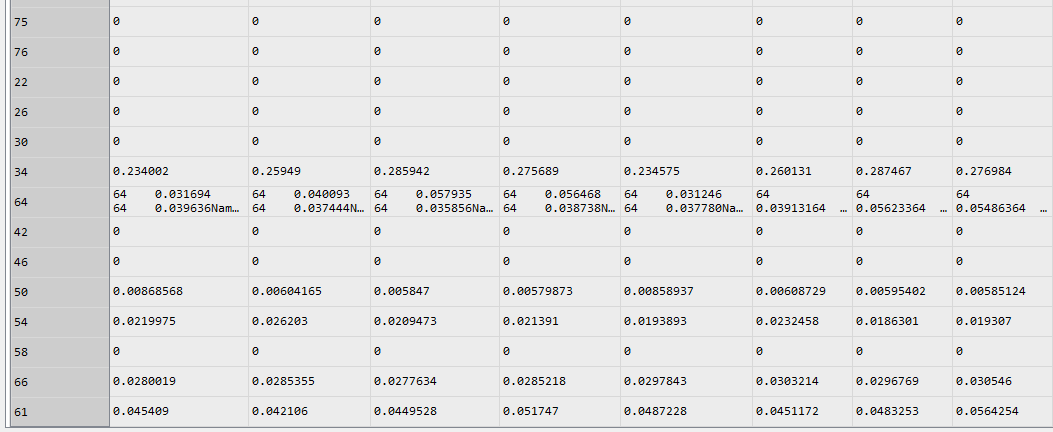
*dataframe\_loads.insert(0,'Id Bus',net.bus['Id Bus'])*

*dataframe\_loads = dataframe\_loads.set\_index('Id Bus', drop = False)*

*dataframe\_loads.insert(q,i['timestamp'],'')*

*for e in dataframe\_loads['Id Bus']:*

*dataframe\_loads[i['timestamp']][e]=net.load['p\_mw'][e]*



In addition, all created nets are saved as global variables, which can be later used and called.

*globals()['net{}'.format(q)] = net*

Finally, when the loop has finished, we create an excel file from the definitive completed Dataframe:

*dataframe\_lines\_overloaded.to\_excel(Path(str(os.getcwd()) +'/PowerFlow.xlsx'))*

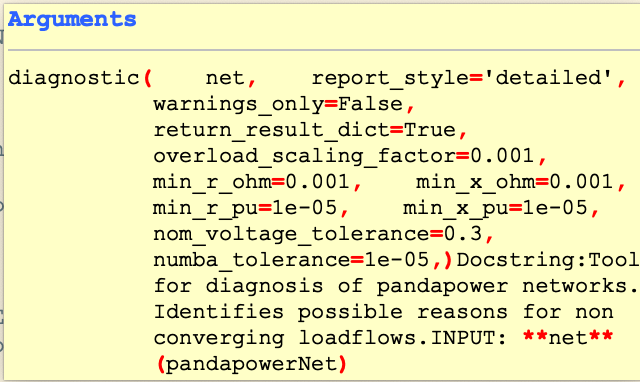
# Problems

Many problems with convergence have arisen. To understand them deep research has been made before fixing them.

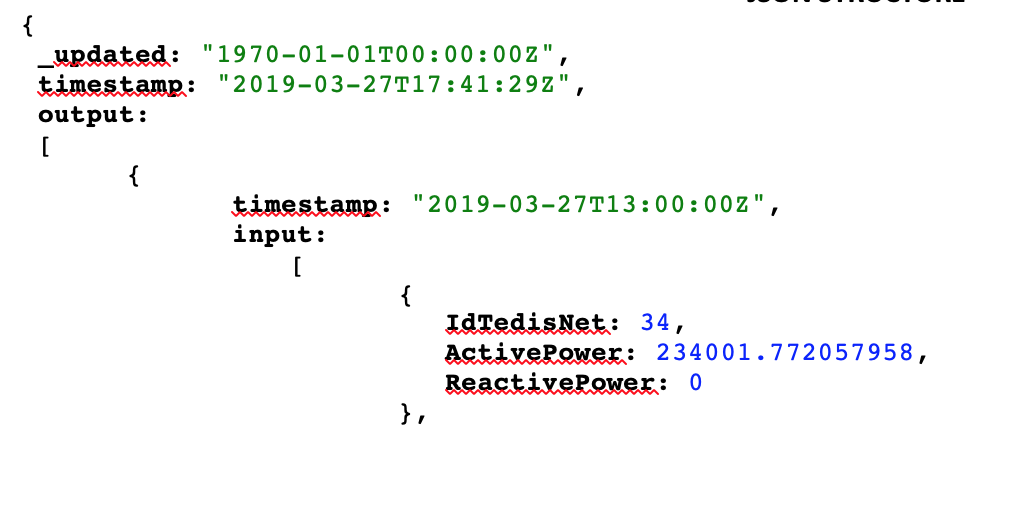
If NO CONVERGENCE message comes out after running the grid make sure you:

* Are using appropriate units
  + Sometimes unit mistakes may lead to out of the logical range results exceeding upper or lower limits for loads, voltages, etc., causing the power flow not to converge.
  + This was the problem we had with Transformers, as we initially thought units were of k instead of M. Actually, it was the first units the program asked for, but when reinstalling Anaconda (because of technical problems) it seems that a new version expected M and not k any more.
* Are asking for adequate range of values
  + Before running the program make sure all values you have put as input data are correctly written according to basic electric knowledge.
  + Be especially careful with very short lines that might produce impedances close to zero.
* Are receiving correct values from the client
  + We have had some issues with loads units, which are not still totally clear and we are waiting for their feedback.

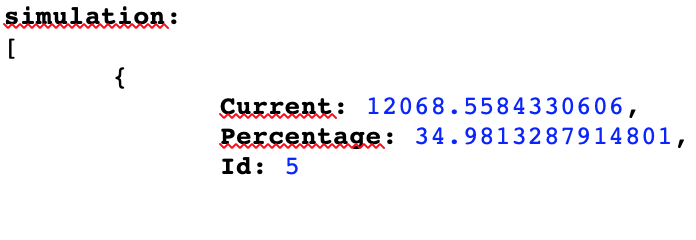
A useful tool for detecting problems is the panda power diagnostic tool:



# Json structure: loads



(e) in the iteration: for e in i[‘input’]

The Json file has three elements in the principal level. These are: Updated, timestamp and output. We are only going to deal with the output data, which inside has three different sections: **timestamp** (the one we are going to use)**, input** (the loads information), **simulation** (the theoretical result of the simulation carried out). We will only use *timestamp* and *input.* In the image above, it is presented the way we have called each part in the iterative loop to make it more understandable.

(i) in the iteration: for i in dfload [‘output’]