NTNU
Norwegian University of
Science and Technology

Multi-terminal HVDC and power flow analysis

Esther Gil Colmenero

Specialization Project of Electric Power Engineering - TET5500

Submission date: December 2011
Supervisor: Kjetil Uhlen
Co-supervisor: Temesgen Haileselassie

Norwegian University of Science and Technology
Department of Electric Power Engineering

Summary

The aim of this Specialization Project is to implement and demonstrate a general AC/DC power flow solution in the Matlab environment. This task is interesting from the point of view of the increasing development of the integration of offshore wind power, especially in the North Sea area.

The solution proposed in this paper, is valid for systems consisting on one DC grid, each of its buses is connected to different AC grids. Specifically, this study focuses on a general DC grid of three nodes and three consequent AC grids.

In this project, a complete procedure on how to set-up the power flow model is developed and described. The first step taken in this process consists on the calculation of the power flows of all the AC grids, in a sequential way. In addition to the calculation of all the involved AC systems, the DC power flow has to be solved as well. The calculations of both power flows, AC and DC, have been implemented using the Newton-Raphson solution algorithm. A key challenge in this procedure has been to model the connection between the AC and DC grid through the HVDC converters. Several possibilities have been studied, mainly depending on the type of bus in question. A study and comparison between a converter with or without losses have been done as well.

The final achievements of this Specialization Project consist on a Matlab code solving successfully this matter, highlighting that it is programmed in a general format, so that it enables modifications on the previous explained configuration between the DC and AC connection by simple changes. This enables the future connection of more AC grids if needed. Additionally, it seems obvious that the election of the slack node is a critical parameter that will influence the results; but in this work, an investigation and discussion is made about the possibility of controlling other parameters of the grid, as for example the AC and DC power at the nodes.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

During recent years the international community has shown a greater environmental responsibility, as an example, important investments in wind power generation have taken place during the last years. The majority of wind farms are situated onshore. However, causes like noise, visual impacts and land disputes are inducing the move of its development from onshore to offshore, where these problems are avoided. In fact, the European Wind Energy Association estimated in 2009 that around 50GW of offshore wind power will be installed in the European countries by 2020, increasing close to 150GW by 2030[1]. In addition to these problems, the following advantages help to promote offshore development:

- Availability of large and continuous areas suitable for major stations.
- Higher wind speed, generally increased with distance from the coast.
- Less turbulences, that reduces the fatigue loads on the turbine and makes it more efficient.

For this matter, other power transmission systems should be considered, where systems based on high voltage direct current transmission are obvious candidates. Multi-terminal HVDC systems have become an interesting technical solution for integrating offshore wind power and the power systems in the North Sea area. Such a solution demands extensive technical economic studies, where basic power flow analyses are common to many of them.

[1] Dr. Nicolas Fichaux and Justin Wilkes. Oceans of Opportunity - Harnessing Europe’s largest domestic energy resource. Technical report, EWEA, September 2009.

This project work deals with the study of these power flows, analyzing the different possibilities of configuration that can be achieved in case of connecting several AC grids, with one DC grid. In this document, this investigation has been made with three AC systems, connected as it is shown in the next image.

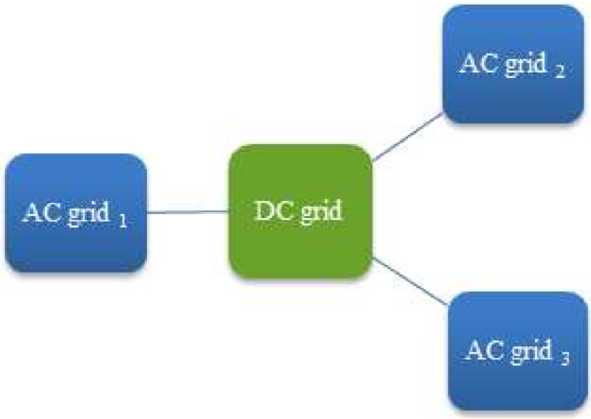

Figure 1. Sketch connection between DC and AC grids

The transmission system converter which enables the connection between the AC and the DC grid constitutes a key factor in this study. It model will be implemented in the Matlab environment too, and a subsequent research about the possibility of which parameters should be previously chosen will be done.

Chapter 2

Power flow analysis theory

2.1 The Power Flow Solution

As it has been said in the summary, the first step is to calculate the power flow of all the grids. The final goal of this procedure is to indicate an operator or a planner of a system the voltage value and it phase angle at each bus/node of the system, as well as the active and reactive power flow in each line.

The power flow solution begins by identifying the known and unknown state variables in the system, which depends on the type of node. Thus, each AC bus is defined by the following four variables:

- Voltage value
- Voltage phase angle
- Real power injection
- Reactive power injection.

In the AC system, there are three possible types of buses, and at each of them two of the previous state variables are defined or given in advance, as it can be seen in the following table.

Bus type	Known parameters	Unknown parameters
Slack bus	Voltage magnitude Voltage angle	Active power Reactive power
Regulated bus (PV)	Real power Voltage magnitude	Reactive power Voltage angle
Load bus (PQ)	Active power Reactive powers	Voltage magnitude Voltage angle

Table 1. Relation between type of bus and known or unknown parameters

In the case of the DC grid this table is simplified, due to the fact that the total number of state variables, and consequently the possible types of nodes is reduced to two: slack bus or power bus. This is because the no consideration of reactive power (Q) and voltage angle in the DC power flow, as it will be explained in the next chapter.

There are several known methods for solving power equations. The more popular numerical ones are the Gauss-Seidel and Newton Raphson method, last one with an added approximate but faster variation called fast decoupled method. For this project the chosen one was the iterative Newton Raphson method, superior to Gauss algorithm in it accuracy and because it exhibits a faster convergence characteristic. On the other hand the drawbacks of this method are the elaborated programming logic and the complex calculations to be done. The first of these disadvantages deals with long and complex codes, as it will be seen in the finals results of this project, and the second drawback is solved by the use of Matlab software, which enables the running of the codes in a short time despite the complexity of the operations.

2.2 Newton Raphson method

As is has been said before, Newton-Raphson is the chosen method used for the calculation of the power flow solution.

This iterative method is based on the linear approximation idea. When considering the function $f(x)$, it basically consists on solving:

$$f(x)=0 \tag{1}$$

Assuming that the zero of this function is near the point $(x_0, f(x_0))$, following the Taylor's expansion about x_0 yields:

$$f(x)=f(x_0)+\left[\frac{df}{dx}\right]\cdot(x-x_0)+\frac{1}{2}\cdot\left[\frac{d^2f}{dx^2}\right]\cdot(x-x_0)^2+... \tag{2}$$

Staying in the first order, then the resultant equation represents the tangent at the curve of the function at the point (x_0, y_0). It is also equal to the slope of the tangent line at the point $(x_0, f(x_0))$.

$$f(x)=f(x_0)+\left[\frac{df}{dx}\right]\cdot(x-x_0)\Rightarrow f'(x_0)=\frac{f(x)-f(x_0)}{(x-x_0)} \tag{3}$$

For the first approximation, point $(x_1, 0)$:

$$0=f(x_0)+\left[\frac{df}{dx}\right]\cdot(x-x_0)\Rightarrow f'(x_0)=\frac{0-f(x_0)}{(x_1-x_0)}\Rightarrow x_1-x_0=\frac{-f(x_0)}{f'(x_0)}\Rightarrow x_1=x_0-\frac{f(x_0)}{f'(x_0)} \tag{4}$$

On the same way, generalizing this result we obtain the following general equation for the iterative process:

$$x_{n+1}=x_n-\frac{f(x_n)}{f'(x_n)} \tag{5}$$

A graphical illustration consisting on three steps of Newton Raphson method can be seen in the next illustration:

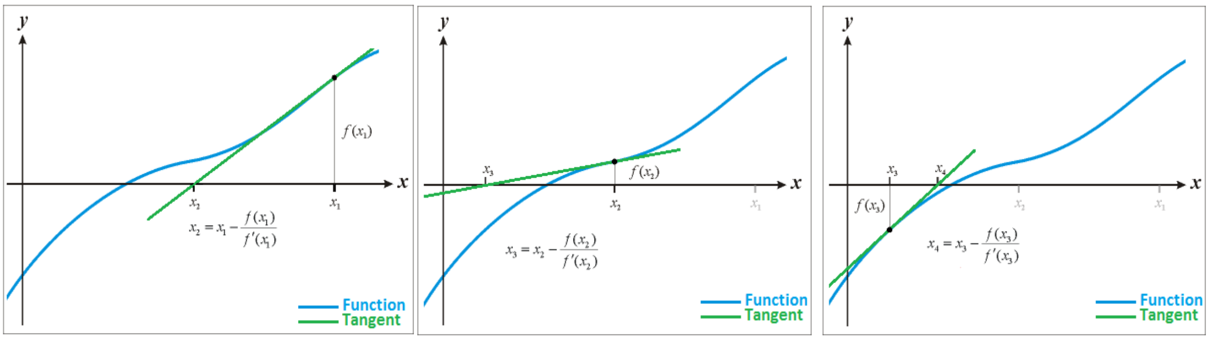

Figure 2. Evolution on the solution procedure by Newton Raphson method

2.2.1 Newton Raphson method applied to power flow equations

Appling the previous equations to the power flow problem, the next equivalences between matrix functions of the system of equations can be made for this goal, where k is the iteration index.

$$x^{[k]} = \begin{bmatrix} \delta^{[k]} \\ V^{[k]} \end{bmatrix} \qquad f(x^{[k]}) = \begin{bmatrix} \Delta P(x^{[k]}) \\ \Delta Q(x^{[k]}) \end{bmatrix} \tag{6}$$

Then, the real and reactive power injection can be expressed as follows:

$$\begin{gathered} P_i^{[k]} = \sum_{j=1}^{n} \left|V_i^{[k]}\right| \cdot \left|V_j^{[k]}\right| \cdot \left|Y_{ij}\right| \cdot \cos(\theta_{ij} - \delta_i^{[k]} + \delta_j^{[k]}) \Rightarrow \\ \Rightarrow \Delta P_i^{[k]} = \sum_{j=1}^{n} \left|V_i^{[k]}\right| \cdot \left|V_j^{[k]}\right| \cdot \left|Y_{ij}\right| \cdot \cos(\theta_{ij} - \delta_i^{[k]} + \delta_j^{[k]}) - P_i^{[k]} \end{gathered} \tag{7}$$

$$\begin{gathered} Q_i^{[k]} = -\sum_{j=1}^{n} \left|V_i^{[k]}\right| \cdot \left|V_j^{[k]}\right| \cdot \left|Y_{ij}\right| \cdot \sin(\theta_{ij} - \delta_i^{[k]} + \delta_j^{[k]}) \Rightarrow \\ \Rightarrow \Delta Q_i^{[k]} = -\sum_{j=1}^{n} \left|V_i^{[k]}\right| \cdot \left|V_j^{[k]}\right| \cdot \left|Y_{ij}\right| \cdot \sin(\theta_{ij} - \delta_i^{[k]} + \delta_j^{[k]}) - Q_i^{[k]} \end{gathered} \tag{8}$$

For the solution of the power flow, we need to form a Jacobian matrix, whose elements are obtained considering previous equations presented.

$$\left. df(x) \middle/ dx \right. \Rightarrow \begin{bmatrix} \Delta P \\ \Delta Q \end{bmatrix} = \begin{bmatrix} \partial P / \partial \delta & \partial P / \partial |V| \\ \partial Q / \partial \delta & \partial Q / \partial |V| \end{bmatrix} \cdot \begin{bmatrix} \Delta \delta \\ \Delta |V| \end{bmatrix} \qquad (9)$$

$$-\begin{bmatrix} \Delta P_1 \\ \vdots \\ \Delta P_{n-1} \\ \Delta Q_1 \\ \vdots \\ \Delta Q_{n-m} \end{bmatrix} = \underbrace{\begin{bmatrix} \frac{\partial P_1}{\partial \delta_1} & \cdots & \frac{\partial P_1}{\partial \delta_{n-1}} & \frac{|V_1| \cdot \partial P_1}{\partial |V_1|} & \cdots & \frac{|V_{n-m}| \cdot \partial P_1}{\partial |V_{n-m}|} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial P_{n-1}}{\partial \delta_1} & \cdots & \frac{\partial P_{n-1}}{\partial \delta_{n-1}} & \frac{|V_1| \cdot \partial P_{n-1}}{\partial |V_1|} & \cdots & \frac{|V_{n-m}| \cdot \partial P_{n-1}}{\partial |V_{n-m}|} \\ \frac{\partial Q_1}{\partial \delta_1} & \cdots & \frac{\partial Q_1}{\partial \delta_{n-1}} & \frac{|V_1| \cdot \partial Q_1}{\partial |V_1|} & \cdots & \frac{|V_{n-m}| \cdot \partial Q_1}{\partial |V_{n-m}|} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial Q_{n-m}}{\partial \delta_1} & \cdots & \frac{\partial Q_{n-m}}{\partial \delta_{n-1}} & \frac{|V_1| \cdot \partial Q_{n-m}}{\partial |V_1|} & \cdots & \frac{|V_{n-m}| \cdot \partial Q_{n-m}}{\partial |V_{n-m}|} \end{bmatrix}}_{\text{Jacobian matrix}} \cdot \begin{bmatrix} \Delta \delta_1 \\ \vdots \\ \Delta \delta_{n-1} \\ \Delta |V_1| / |V_1| \\ \vdots \\ \Delta |V_{n-m}| / |V_{n-m}| \end{bmatrix} \qquad (10)$$

The following table shows the equations for the calculation of these jacobian terms. As it can be observed, it depends on the position of the element in this matrix, specifically it influences if the element correspond to the principal diagonal of the Jacobian matrix or not. Next table is valid for calculating the terms depending on the voltage angle derivative.

		Voltage angle	
Active power (P)	Diagonal	$\frac{\partial P_i}{\partial \delta_i} = \sum_{j \neq i} \lvert V_i \rvert \cdot \lvert V_j \rvert \cdot \lvert Y_{ij} \rvert \cdot \sin\left(\theta_{ij} - \delta_i + \delta_j\right)$	(11)
	Not diagonal, $i \neq j$	$\frac{\partial P_i}{\partial \delta_j} = -\lvert V_i \rvert \cdot \lvert V_j \rvert \cdot \lvert Y_{ij} \rvert \cdot \sin\left(\theta_{ij} - \delta_i + \delta_j\right)$	(12)
Reactive power (Q)	Diagonal	$\frac{\partial Q_i}{\partial \delta_i} = \sum_{j \neq i} \lvert V_i \rvert \cdot \lvert V_j \rvert \cdot \lvert Y_{ij} \rvert \cdot \cos\left(\theta_{ij} - \delta_i + \delta_j\right)$	(13)
	Not diagonal, $i \neq j$	$\frac{\partial Q_i}{\partial \delta_j} = -\lvert V_i \rvert \cdot \lvert V_j \rvert \cdot \lvert Y_{ij} \rvert \cdot \cos\left(\theta_{ij} - \delta_i + \delta_j\right)$	(14)

Table 2. Equations for jacobian terms depending on the voltage angle derivative

The rest of the jacobian terms are calculated with the next equations shown in the following table, depending again on its position at the Jacobian matrix.

		Voltage module	
Active power (P)	Diagonal	$\frac{\partial P_i}{\partial \lvert V_i \rvert} = 2 \cdot \lvert V_i \rvert \cdot \lvert Y_{ii} \rvert \cdot \cos\theta_{ii} + \sum_{j \neq i} \lvert V_j \rvert \cdot \lvert Y_{ij} \rvert \cdot \cos\left(\theta_{ij} - \delta_i + \delta_j\right)$	(15)
	Not diagonal, $i \neq j$	$\frac{\partial P_i}{\partial \lvert V_j \rvert} = \lvert V_i \rvert \cdot \lvert Y_{ij} \rvert \cdot \cos\left(\theta_{ij} - \delta_i + \delta_j\right)$	(16)
Reactive power (Q)	Diagonal	$\frac{\partial Q_i}{\partial \lvert V_i \rvert} = -2 \cdot \lvert V_i \rvert \cdot \lvert Y_{ii} \rvert \cdot \sin\theta_{ii} + \sum_{j \neq i} \lvert V_j \rvert \cdot \lvert Y_{ij} \rvert \cdot \sin\left(\theta_{ij} - \delta_i + \delta_j\right)$	(17)
	Not diagonal, $i \neq j$	$\frac{\partial Q_i}{\partial \lvert V_j \rvert} = -\lvert V_i \rvert \cdot \lvert Y_{ij} \rvert \cdot \sin\left(\theta_{ij} - \delta_i + \delta_j\right)$	(18)

Table 3. Equations for jacobian terms depending on the voltage module derivative

Next step consist on actualizing the values of parameters, for the next iteration, in case that the error of the obtained result is bigger than the tolerance selected for the necessary accuracy. Those parameters to be actualized are the voltage value (module and angle), as well as active and reactive power.

$$\delta_i^{[k+1]} = \delta_i^{[k]} - \Delta\delta_i^{[k]} \quad (19)$$

$$\left|V_i^{[k+1]}\right| = \left|V_i^{[k]}\right| - \Delta\left|V_i^{[k]}\right| \quad (20)$$

$$\Delta P_i^{[k]} = P_i - P_i^{[k]} \quad (21)$$

$$\Delta Q_i^{[k]} = Q_i - Q_i^{[k]} \quad (22)$$

All this steps to follow for the power flow solution by Newton Raphson method, will be explained in detail in the next chapter, due to the necessary classification between AC and DC power flow, which have different steps to implement, because of the no consideration of some parameters in the case of the DC grid.

Chapter 3

Solution proposal

3.1 Solution proposal

The solution proposed is divided into three blocks, as it can be seen in the sketch at the *figure 3.*

There will be a DC block, which calculates the power flow of the DC grid, by Newton Raphson method, followed by an interface block, which makes the function of implementing the converter between the AC and the DC grid. This second block will analyze the values and directions of the powers at each node, and depending on the characteristics of each one (type of node) it will establish some constrains, for the next iterations of the algorithm. The function of the third and last block is to solve the power flow of all the AC grids by Newton Raphson method.

For the implementation of these blocks, several software options where first considered, as Simpow or Matpower, the package of Matlab M-files. Finally, the selected decision was to program with Matlab, but without using any predefined file. Furthermore, a general programming was used, enabling future modifications in relation with the number of AC systems considering.

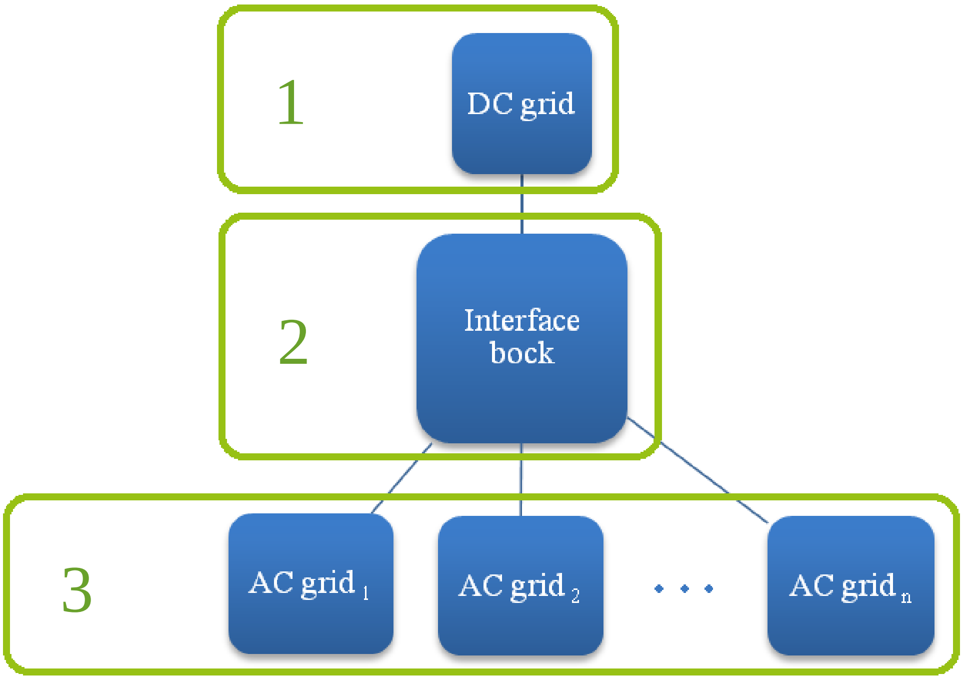

Figure 3. Sketch of the solution proposal

3.2 AC Power flow calculation by Newton-Raphson method

The number of AC systems has to be equal to the number of nodes of the DC grid, which are three in our case. The size of each system, that is the number of buses, in addition to the configuration and number of the lines connecting them, is determined by the user as well. In our studied case all the AC systems are composed of three nodes.

The program needs as data input the number of AC systems, and from each system it needs: data from the nodes (voltage, angle of voltage, type of node, active power generated, reactive power generated, active power demanded, reactive power demanded and suscepstance) and input data from the lines of the system (R_{ij}, X_{ij}, B_{ij}).

For the organization of the processing of all these input data, two matrices have been created. One of them will contain the data of the nodes, and the other one will store the input data of the lines are its connections. Next two images show in detail this idea.

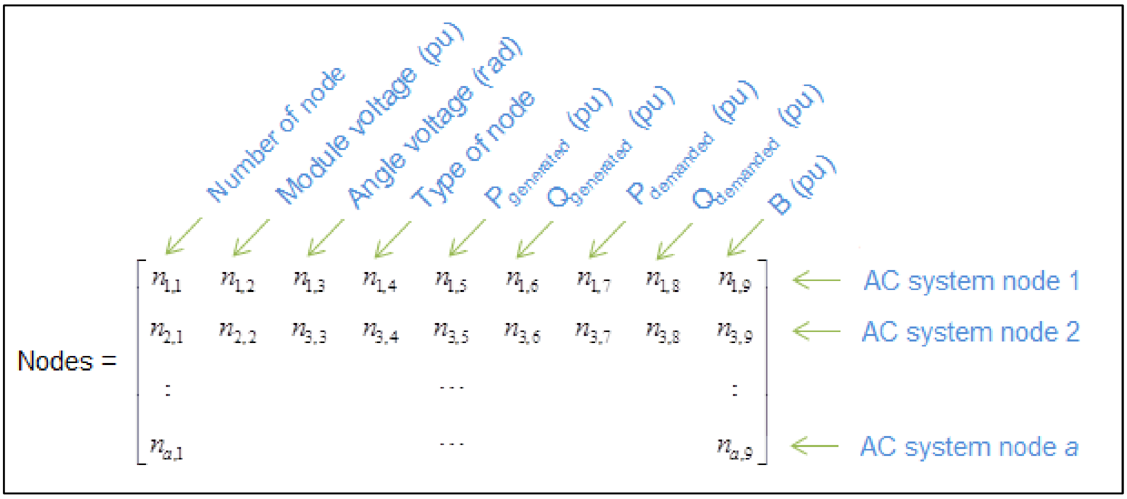

Figure 4. Sketch of the matrix with the nodes information, where *a* is the total number of nodes of the system

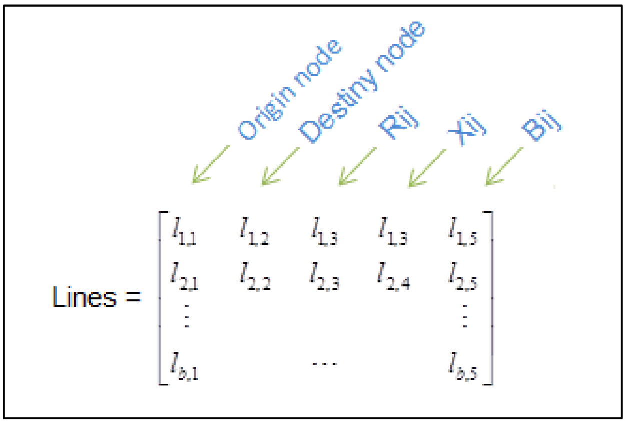

Figure 5. Sketch of the matrix with the lines information

For a successful implementation, the previous matrices must fit some constrains.

- One the one hand the slack node has to be called as node 1. The remaining nodes can be named regardless its type.
- Additionally the numbers of the first column of the previous explained *nodes* matrix (the one that corresponds to the number of node) should be order in an increasing way: starting the first row with number one, and ending in the last row with the grater node index of the AC system.

Following the fundaments of the iterative Newton Raphson method explained in *chapter 2*, the resulting algorithm considered for solving the AC power flow can be represented as follows:

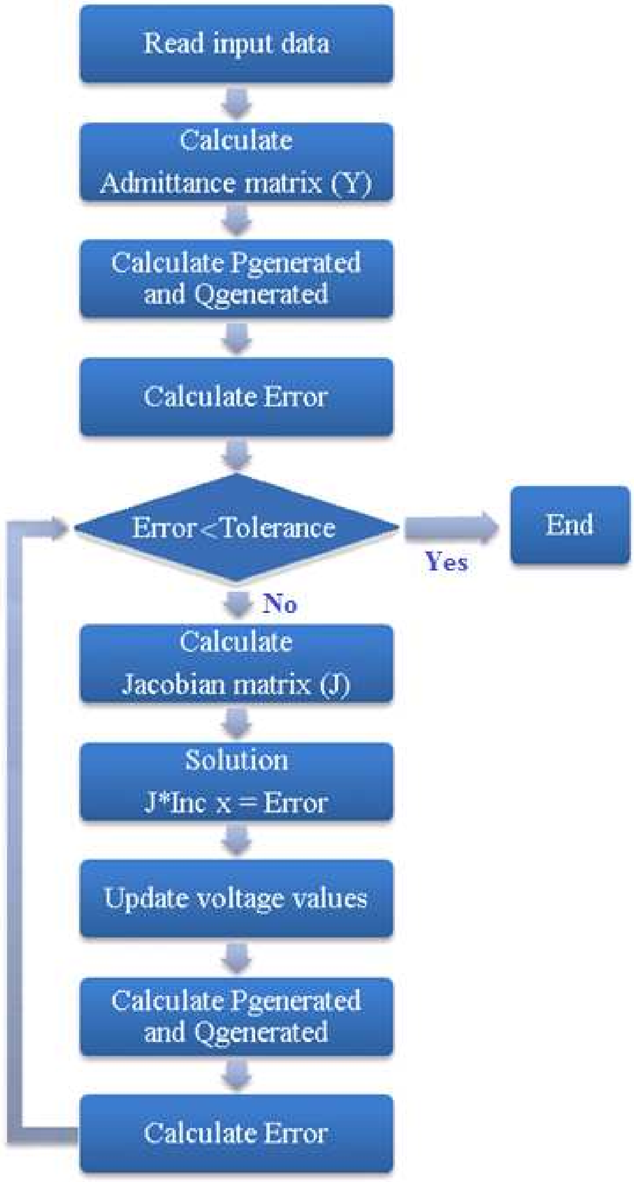

Figure 6. Flow diagram of the AC power flow calculation by Newton Raphson method

For the implementation of the Jacobian matrix, a simplification for the calculation has been made, by dividing the matrix in four sub matrices as follows:

$$J = \left[\begin{array}{ccc|ccc}
\frac{\partial P_1}{\partial \delta_1} & \cdots & \frac{\partial P_1}{\partial \delta_{n-1}} & \frac{|V_1| \cdot \partial P_1}{\partial |V_1|} & \cdots & \frac{|V_{n-m}| \cdot \partial P_1}{\partial |V_{n-m}|} \\
\vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
\frac{\partial P_{n-1}}{\partial \delta_1} & \cdots & \frac{\partial P_{n-1}}{\partial \delta_{n-1}} & \frac{|V_1| \cdot \partial P_{n-1}}{\partial |V_1|} & \cdots & \frac{|V_{n-m}| \cdot \partial P_{n-1}}{\partial |V_{n-m}|} \\
\hline
\frac{\partial Q_1}{\partial \delta_1} & \cdots & \frac{\partial Q_1}{\partial \delta_{n-1}} & \frac{|V_1| \cdot \partial Q_1}{\partial |V_1|} & \cdots & \frac{|V_{n-m}| \cdot \partial Q_1}{\partial |V_{n-m}|} \\
\vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
\frac{\partial Q_{n-m}}{\partial \delta_1} & \cdots & \frac{\partial Q_{n-m}}{\partial \delta_{n-1}} & \frac{|V_1| \cdot \partial Q_{n-m}}{\partial |V_1|} & \cdots & \frac{|V_{n-m}| \cdot \partial Q_{n-m}}{\partial |V_{n-m}|}
\end{array}\right] = \begin{bmatrix} \boxed{H} & \boxed{N} \\ \boxed{M} & \boxed{L} \end{bmatrix}$$

Figure 7. Division of the Jacobian matrix in four sub matrices

After the execution of the resulting Matlab code, the final solution of the power flow of each system is shown in the user screen. A last comment about the code obtain in this step, is the important advantage that it is programmed with a general format, not forcing the number of AC systems to three, but enabling the future connection of more AC grids if needed.

3.3 DC Power flow calculation by Newton-Raphson method

The second approach in this project is to develop a code that calculates the power flow of the DC grid.

This task has been done with several modifications in the Matlab code resulting of the previous section (AC power flow). For doing it successfully, several considerations regarding the differences between DC and AC power flow should be taken into account:

- No consideration of reactive power (Q).
- No voltage angle values considered.
- Instead of three types of nodes as in the AC grid (slack node, PV node and PQ node), we will have only two possible types: slack node and power node.
- The admittance matrix is only composed by resistive part (no complex part).

In addition to these general constrains of the DC systems, an extra constrain has been taken in account for this project: the maximum number of DC grids on our case of study is fixed and is only one, with difference on the AC system, where the number of AC grids is an election of the user, and it could be greater than one. However, the number of buses of this DC grid is a free election of the user. In the solution shown in the *appendix 1*, the number of nodes of the DC system selected was three.

By modifications on the iterative Newton Raphson method used for the AC grid, the resulting algorithm for solving this problem can be represented as follows:

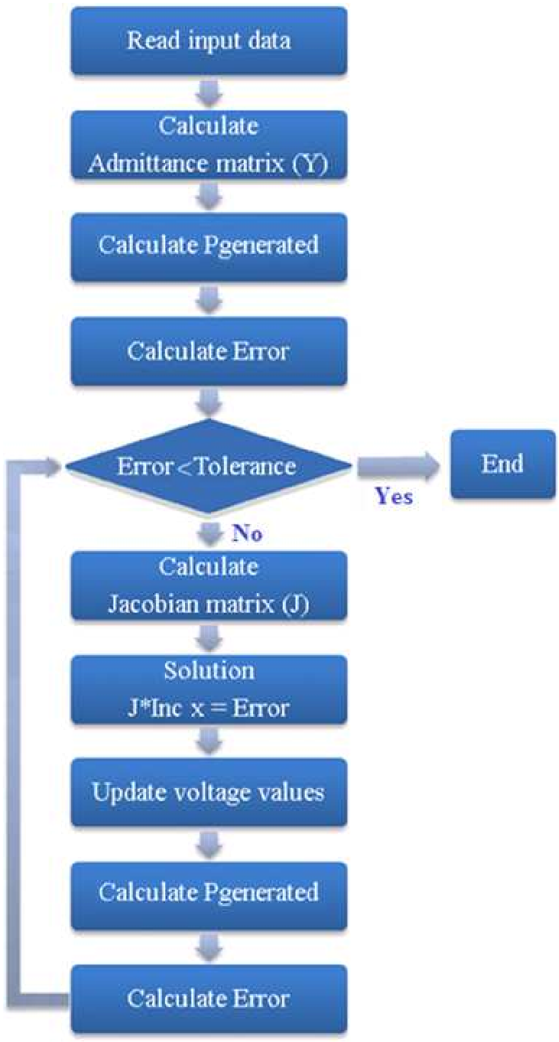

Figure 8. Flow diagram of the DC power flow calculation by Newton Raphson method

Finally, taking into account the previously explained considerations and following the upper flow diagram, the result of the DC power flow solving program ends on the code shown in the *appendix 1.*

3.4 Implementation of the Interface block

The main goal of the implementation block is to do the connection between the DC grid and the AC systems. This block considers the behavior of the AC/DC converter, to implement an algorithm that implements its functions.

It has been necessary the definition of a vector (*connection*), for indicating which node of each AC grid is connected to which node of the DC grid. Consequently, it size will be equal to the number of buses of the DC grid. The elements of the vector are introduced as follows: the first row indicates which bus of the AC system number one is connected to the bus one of the DC grid, the second element of the vector indicates the bus of the AC system two, connected to the DC bus number two, and so on. An example will be given in next chapter.

3.4.1 Converter model with no consideration of losses

Basically, it consists on analyzing all the possibilities in each node of the DC grid. Three cases are defined when defining bus arrangements for the converters:

- ***Case 1. The DC side is the slack node.***

  In this first case, the active power of the AC grid side has to take the value of the DC active power at each iteration step.

$$P_{AC\,i} = -P_{DC\,i} \tag{23}$$

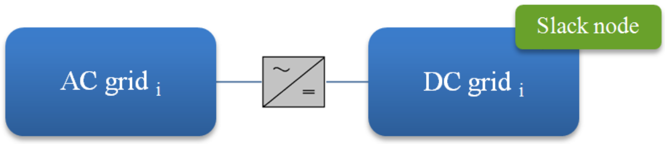

Figure 9. Sketch of the connection with the converter, when the DC grid is the slack node

- ***Case 2. The AC side is the slack node.***

In this second case, the AC side sets the value of the DC active power, so that the DC grid has to take the value from P_{AC} at each iteration.

$$P_{DC\,i} = -P_{AC\,i} \tag{24}$$

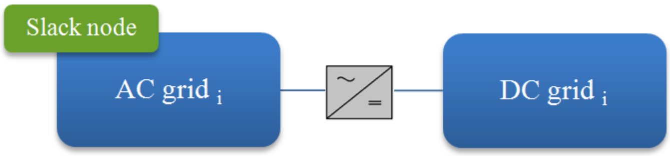

Figure 10. Sketch of the connection with the converter, when the AC grid is the slack node

- ***Case 3. None of the AC or DC sides corresponds to the slack node.***

In this case, the active power could be fixed in anyone of the two sides, AC or DC. Both possibilities have been studied in this project, resulting two different codes.

a) DC active power constant

In one of the codes the decision selected was that the DC grid sets the value of the active power of the AC grid, that is, the same as in the previous explained case 1. So, equation number 23 was the one used in this case.

b) AC active power constant

On the other hand, another code has been created considering that the AC grid sets the value of the active power of the DC node, which corresponds to case 2, and so the equation number 24 was the one applied.

This previous considerations and equivalences are valid in the case that we consider the converter without losses, following the *equation 25*. But in next section, a more realistic converter will be implemented, by taking in account it losses.

$$P_{AC} + P_{DC} = 0 \tag{25}$$

3.4.2 Converter model with consideration of losses

In the case that the converter is considered with losses, the model to implement is the following one:

$$\mathrm{P_{AC}} + \mathrm{P_{DC}} + \mathrm{P_{losses}} = 0 \tag{26}$$

We take into consideration that the losses of the converter AC/DC obey the next simple model, where *k* is a constant of the converter, in our case with a selected value of 2%. The reason for taking the absolute value of the active power is because of the restriction that losses always have to be positive.

$$P_{losses} = k \cdot |P| \tag{27}$$

Two possibilities have to be studied, regarding the direction of the active power flow.

a) If the DC active power is positive, it means that the power flow goes from the AC to the DC grid, so the resulting AC active power will has negative sign. Consequently the converter is carrying out a rectifier operation. Applying *equations 26* and *27*, the following equality results:

$$P_{AC} = -P_{DC} - P_{losses} \Rightarrow P_{AC} = -P_{DC} - (k \cdot |P_{DC}|) \quad (28)$$

In the code of the interface block, with losses considerations, this previous equation has to be applied in the case that the DC side is the slack node, and consequently the AC grid side has to take the value of the DC active power at each iteration (case 1 of section 3.4.1).

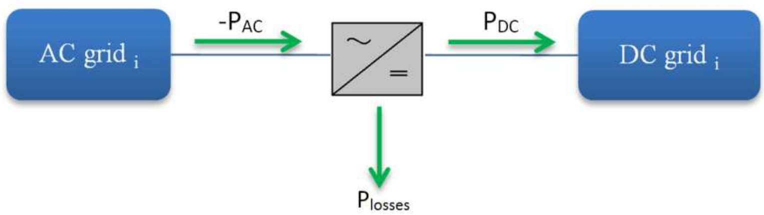

Figure 11. Power flow when P_{DC} is positive

b) In the case that the DC active power is negative, the power flow will go now from the DC to the AC grid, and the AC active power will be positive in this case.

$$P_{DC} = -P_{AC} - P_{losses} \Rightarrow P_{DC} = -P_{AC} - (k \cdot |P_{AC}|) \quad (29)$$

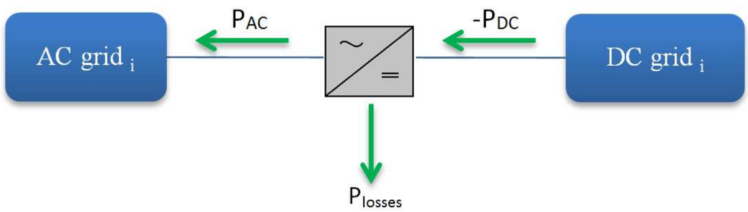

Figure 12. Power flow when P_{DC} is negative

Equation 28 has been used in the case that the AC side is the slack node (case 2 of section 3.4.1). In this case, the AC side sets the value of the DC active power, so that the DC grid has to take the value from the AC active power at each iteration.

The remaining case, where none AC or DC side are slack nodes, as in the previous section with no losses consideration, two alternatives has been studied: when the AC active power is the fixed one, and when the DC power is the fixed one.

Chapter 4

Results

4.1 DC grid

In this part, a sample of the running of the code attached at the *appendix 1*, corresponding to the DC power flow solution is shown.

For the DC system implemented, composed by three nodes, the arbitrary input values selected for the demonstration are summarized in the next table (information find in the definition of the matrices *nodes* and *lines* of the code enclosed).

NODE	Voltage (p.u.)	Type of node	$P_{generated}$ (p.u.)	$P_{demanded}$ (p.u.)	LINE	R_{ij} (p.u.)
Node 1	1	Slack	0	0	Line 1-2	0.0108
Node 2	1	Power	1	0	Line 1-3	0.0235
Node 3	0.9	Power	0	0.6	Line 2-3	0.0147

Table 4. Data input of the DC grid

The following image corresponds to the result obtained in the command window of Matlab after running the code. Some variables have been selected to show their final value after the execution.

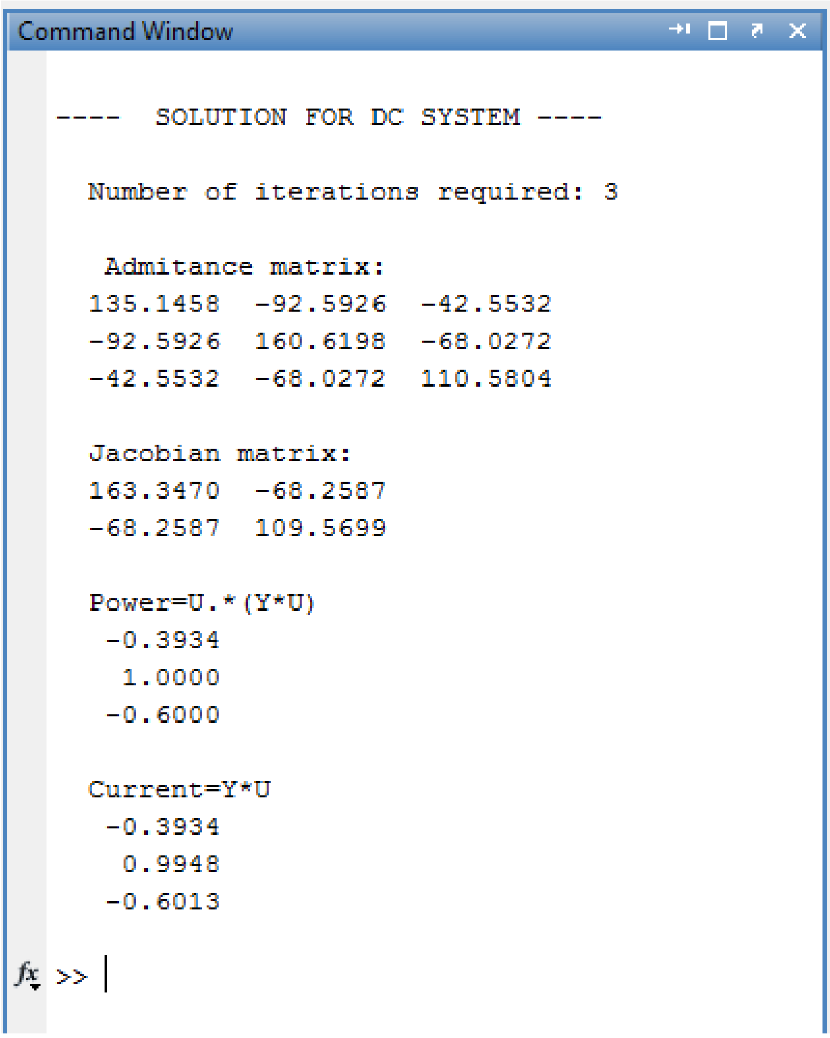

Figure 13. Final results of the DC power flow

The previous results indicates that three iterations were necessary for the convergence (performance of condition error<tolerance, for a selected tolerance of 10e-5).The jacobian matrix shown corresponds to the one calculated at the last iteration. As it can be checked, the sum of the power of the system is equal to 0.0066 p.u. This sign of this result is positive, as it was expected for a correct working.

4.2 AC grid

4.2.1 Single AC grid

For the demonstration of the working of the AC power flow solution, the input data of the system shown in the *figure 14* was introduced.

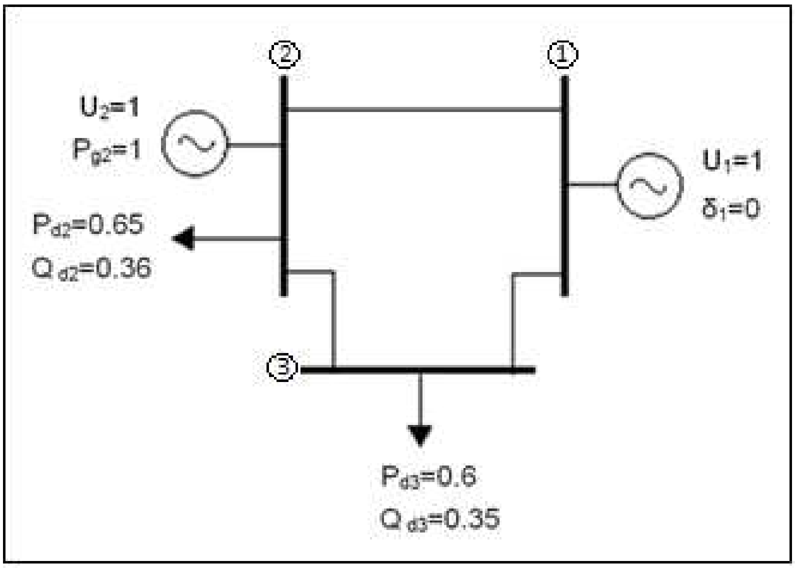

Figure 14. AC system example implemented

The information of the lines of this previous AC grid is summarized in the following table.

LINE	R_{ij} (p.u.)	X_{ij} (p.u.)	B (p.u.)
Line 1-2	0.0108	0.0649	0.066
Line 1-3	0.0235	0.0941	0.04
Line 2-3	0.0147	0.0566	0.08

Table 5. Lines input parameters of the AC grid

When running the Matlab code attached at *appendix 2,* the results displayed at *figure 15* are obtained. The admittance matrix, which does not change during the iterations, is a square matrix of dimension three (the number of nodes of the grid). The jacobian matrix has the same size, since there is one slack node, one regulated node (PV) and a load node (PQ). The sum of the final powers of the system is positive as well.

```
Command Window

----  SOLUTION FOR AC SYSTEM  ----

  Admitance matrix:
   4.9931 -24.9433i  -2.4950 +14.9931i  -2.4981 +10.0031i
  -2.4950 +14.9931i   6.7937 -31.4715i  -4.2987 +16.5514i
  -2.4981 +10.0031i  -4.2987 +16.5514i   6.7968 -26.4945i

  Jacobian matrix:
   31.3639  -16.2225   -3.9410
  -16.3741   26.0251    6.0112
    4.5249   -7.1741   25.3722

  Number of iterations required:
     3

  The solution after these iterations is the next repart of voltages:
   1.0000
   1.0000 - 0.0013i
   0.9836 - 0.0194i

  Absolutes values of voltages:
    1.0000
    1.0000
    0.9838

  Power=U.*(Y*U)
   0.2541 - 0.0591i
   0.3497 - 0.1240i
  -0.5857 + 0.3733i

  Current=Y*U
   0.2541 - 0.0591i
   0.3499 - 0.1235i
  -0.6027 + 0.3677i

fx >>
```

Figure 15. Final result of the single AC grid

4.2.2 Multiples AC grids

For the future converter implementation, the calculation of several AC grids has to be done. With this goal, some changes in the previous code of the AC grid have been made, in order to calculate with just one program, more than one AC system, with different input parameters between them.

In the example proposed, the code solves three AC power flows. The information of these systems in question is presented in *figures 16* and *17*, together with *table 6*, which contains the lines parameters values. The sketch of AC system number three is not indicated, because the node parameters value and its configuration is exactly the same as the one of system number one. The reason for this is the easier and faster verification of the correct working of the code, so that if the results of the power flow of the third system coincide with the ones of the first one, after running the second grid, means that none of the parameters take wrong values from the previous power flow calculation.

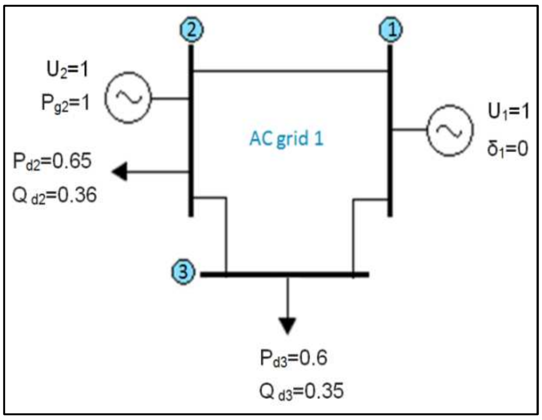

Figure 16. Configuration of AC systems one and three

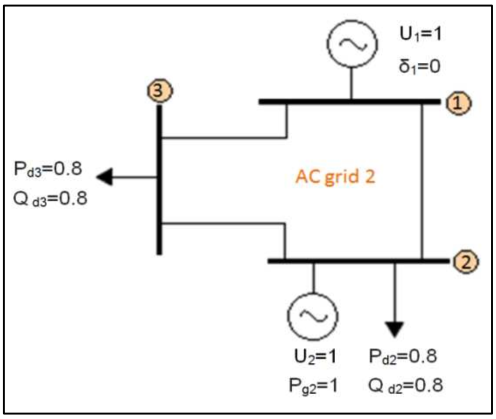

Figure 17. Configuration of AC system two

	Line	R_{ij} (p.u.)	X_{ij} (p.u.)	B (p.u.)
AC grids 1 and 3	Line 1-2	0.0108	0.0649	0.0660
	Line 1-3	0.0235	0.0941	0.0400
	Line 2-3	0.0147	0.0566	0.0800
AC grid 2	Line 1-2	0.0200	0.0600	0.0600
	Line 1-3	0.0200	0.0900	0.0400
	Line 2-3	0.0500	0.0500	0.0800

Table 6. Line input parameters of AC systems

```
Command Window

----  SOLUTION FOR SYSTEM 1  ----

  Admitance matrix:
   4.9931 -24.9433i  -2.4950 +14.9931i  -2.4981 +10.0031i
  -2.4950 +14.9931i   6.7937 -31.4715i  -4.2987 +16.5514i
  -2.4981 +10.0031i  -4.2987 +16.5514i   6.7968 -26.4945i

  Jacobian matrix:
   31.3639  -16.2225   -3.9410
  -16.3741   26.0251    6.0112
    4.5249   -7.1741   25.3722

  Number of iterations required: 3

  The solution after these iterations is the next repart of voltages:
   1.0000
   1.0000 - 0.0013i
   0.9836 - 0.0194i

  Absolutes values of voltages:
    1.0000
    1.0000
    0.9838

  Power=U.*(Y*U)
   0.2541 - 0.0591i
   0.3497 - 0.1240i
  -0.5857 + 0.3733i

  Current=Y*U
   0.2541 - 0.0591i
   0.3499 - 0.1235i
  -0.6027 + 0.3677i




----  SOLUTION FOR SYSTEM 2  ----

  Admitance matrix:
   7.3529 -25.5382i  -5.0000 +15.0000i  -2.3529 +10.5882i
  -5.0000 +15.0000i  15.0000 -24.9300i -10.0000 +10.0000i
  -2.3529 +10.5882i -10.0000 +10.0000i  12.3529 -20.5282i

   Jacobian matrix:
    24.5315   -9.5757   -9.5757
    -9.6288   19.6914   10.6716
     9.6288  -12.1081   18.1641

   Number of iterations required: 3

   The solution after these iterations is the next repart of voltages:
    1.0000
    0.9998 - 0.0188i
    0.9566 - 0.0228i
```

```
  Absolutes values of voltages:
    1.0000
    1.0000
    0.9568

  Power=U.*(Y*U)
   0.6259 - 0.2651i
   0.1846 - 0.4166i
  -0.7606 + 0.8370i

  Current=Y*U
   0.6259 - 0.2651i
   0.1923 - 0.4131i
  -0.8156 + 0.8556i




----  SOLUTION FOR SYSTEM 3  ----

  Admitance matrix:
   4.9931 -24.9433i  -2.4950 +14.9931i  -2.4981 +10.0031i
  -2.4950 +14.9931i   6.7937 -31.4715i  -4.2987 +16.5514i
  -2.4981 +10.0031i  -4.2987 +16.5514i   6.7968 -26.4945i

  Jacobian matrix:
   31.3639  -16.2225   -3.9410
  -16.3741   26.0251    6.0112
    4.5249   -7.1741   25.3722

  Number of iterations required: 3

  The solution after these iterations is the next repart of voltages:
   1.0000
   1.0000 - 0.0013i
   0.9836 - 0.0194i

  Absolutes values of voltages:
    1.0000
    1.0000
    0.9838

  Power=U.*(Y*U)
   0.2541 - 0.0591i
   0.3497 - 0.1240i
  -0.5857 + 0.3733i

  Current=Y*U
   0.2541 - 0.0591i
   0.3499 - 0.1235i
  -0.6027 + 0.3677i

fx >>
```

Figure 18. Final solution of the three power flow

4.3 Converter implementation

In this section, the interface block explained in *chapter 3* is implemented. With that goal, the previous explained codes are used, adding as well extra programming for the converters function.

Several codes has been created depending on these two criteria: the consideration or not of the losses of the converters, and the option chosen in case 3 of section 3.4, when none of the AC/DC nodes connected to the converter where slack, so a decision between which node sets the constant active power has to be taken. Therefore, four different codes have been created for solving all this possible cases:

- No losses consideration of the converter (*equation 25*).
  - The DC node is the one that sets the power.
  - The AC node is the one that sets the power.

- Losses of the converter considered (*equation 26*).
  - The DC node is the one that sets the power.
  - The AC node is the one that sets the power.

The configuration used for the demonstration of the correct running of the codes, regardless of the case considered, is sketched in *figure 19*. The information of the values at the nodes and the lines parameters of all the systems are indicated in *tables 7* and *8*. As the grid configuration, all these data are common to all the codes of the possible cases previously explained.

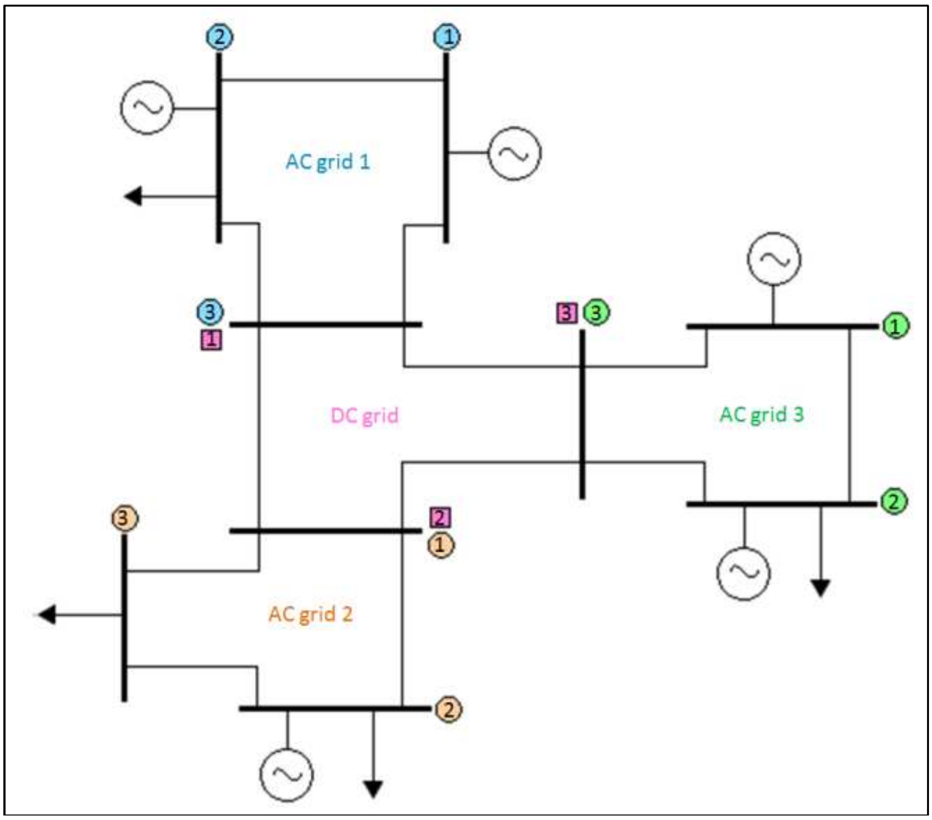

Figure 19. Configuration of the DC and the AC systems

	Line	R_{ij} (p.u.)	X_{ij} (p.u.)	B (p.u.)
AC grid 1	Line 1-2	0.0108	0.0649	0.0660
	Line 1-3	0.0235	0.0941	0.0400
	Line 2-3	0.0147	0.0566	0.0800
AC grid 2	Line 1-2	0.0200	0.0600	0.0600
	Line 1-3	0.0300	0.0900	0.0400
	Line 2-3	0.0250	0.0500	0.0800
AC grid 3	Line 1-2	0.0110	0.0649	0.066
	Line 1-3	0.0235	0.0942	0.0500
	Line 2-3	0.0150	0.0560	0.0800
DC grid	Line 1-2	0.0108	-	-
	Line 1-3	0.0235	-	-
	Line 2-3	0.0147	-	-

Table 7. Line parameters of the systems

	Node	Voltage (p.u.)	V. angle (rad.)	$P_{gen.}$ (p.u.)	$Q_{gen.}$ (p.u.)	$P_{dem.}$ (p.u.)	$Q_{dem.}$ (p.u.)
AC grid 1	1	1	0	-	-	-	-
	2	1	-	1	0	0.65	0.36
	3	1	-	0	0	0.6	0.35
AC grid 2	1	1	0	-	-	-	-
	2	1	-	1	0	0.8	0.71
	3	1	-	0	0	0.8	0.70
AC grid 3	1	1	0	-	-	-	-
	2	1	-	0.9	0	0.65	0.37
	3	1	-	0	0	0.6	0.36
DC grid	1	1	0	-	-	-	-
	2	1	-	1	-	0	-
	3	0.9	-	0	-	0.6	-

Table 8. Node parameters of the systems

With all these input values has to be introduced by the user at the beginning of the code, indicating in each system the values of the nodes (matrix *nodes*) and of the lines (matrix *lines*), and filling in the vector *connection*, to indicate which node of each system is connected to which DC node. As an example, in the configuration presented in *figure 19*, the vector connection will have the following elements according to the connections:

$$connection = \begin{bmatrix} 3 \\ 1 \\ 3 \end{bmatrix} \begin{array}{l} \longrightarrow \text{Node 3 of AC system 1 connected to DC grid node 1} \\ \longrightarrow \text{Node 1 of AC system 2 connected to DC grid node 2} \\ \longrightarrow \text{Node 3 of AC system 3 connected to DC grid node 3} \end{array}$$

When running, for example, the model converter taking in account it losses, and considering that in the case that none of the nodes connected to the converter are slack, then the AC node takes the active power of the DC node, the result obtained is the next one:

Command Window

```
----  SOLUTION FOR AC SYSTEM NUMBER 1  ----
  Admitance matrix:
  135.1458  -92.5926  -42.5532
  -92.5926  160.6198  -68.0272
  -42.5532  -68.0272  110.5804

  Jacobian matrix:
   13.1038    2.7167  -67.6371
   -2.7155    0.3418  109.1140
   67.6370 -109.8696   -0.3418

  Voltages:
   1.0000
   0.9856 + 0.1692i
   0.9926 + 0.1074i

  Absolute values voltages:
    1.0000
    1.0000
    0.9984

  Current=Y*U
    0.0677
    0.1083
   -0.1760

  Power=U.*(Y*U)
    0.0677
    0.1083
   -0.1757

----  SOLUTION FOR AC SYSTEM NUMBER 2  ----
  Admitance matrix:
   83.3333  -50.0000  -33.3333
  -50.0000   90.0000  -40.0000
  -33.3333  -40.0000   73.3333

  Jacobian matrix:
    1.6495    0.0960  -39.4169
   -0.1723    0.7627   70.1710
   39.4172  -72.2537   -0.7627

  Voltages:
   1.0000
   0.9933 - 0.1157i
   0.9878 - 0.0543i

  Absolute values voltages:
    1.0000
    1.0000
    0.9893

  Current=Y*U
    0.3556
    0.4267
   -0.7823

  Power=U.*(Y*U)
    0.3556
    0.4267
   -0.7739
```

```
----  SOLUTION FOR AC SYSTEM NUMBER 3  ----
  Admitance matrix:
  133.4623  -90.9091  -42.5532
  -90.9091  157.5758  -66.6667
  -42.5532  -66.6667  109.2199

  Jacobian matrix:
   -1.6510   -0.5444  -66.1964
    0.5456    0.3786  106.9329
   66.1964 -108.4505   -0.3786

  Voltages:
   1.0000
   0.9969 + 0.0786i
   0.9924 + 0.0504i

  Absolute values voltages:
    1.0000
    1.0000
    0.9937

  Current=Y*U
    0.2685
    0.4207
   -0.6892

  Power=U.*(Y*U)
    0.2685
    0.4207
   -0.6848

----  SOLUTION FOR DC SYSTEM ----
  Admitance matrix:
  135.1458  -92.5926  -42.5532
  -92.5926  160.6198  -68.0272
  -42.5532  -68.0272  110.5804

  Jacobian matrix:
  163.3470  -68.2587
  -68.2587  109.5699

  Voltages:
    1.0000
    1.0053
    0.9978

  Current=Y*U
   -0.3934
    0.9948
   -0.6013

  Power=U.*(Y*U)
   -0.3934
    1.0000
   -0.6000
```

Figure 20. Results of the running of the code for converter with losses and set of DC active power

Chapter 5

Conclusions

The elaboration of this project, consisted of a first documentation and assimilation on the topic, and a later programming, deals with the following codes:

- Matlab code for the power flow solution of a single DC grid (*appendix 1*). The size of this system has no constrains and the nodes and lines parameters can be on a freeway selected by the user.
- Code with the algorithm for solving a single AC power flow problem (*appendix 2*). As in the previous case, the configuration and the parameters of this system can be chosen by the user.
- By modifications on the previous code, a new one was created for solving several AC power flows, with just one program running (*appendix 3*).
- Code for solving the power flow of a DC grid with three terminals, each of them connected to a AC grid, not taking into account the losses of the converter, and considering that in case that none of the buses connected to the converter corresponds to the slack one, the DC bus will assign the power of the AC grid.
- Code for solving the same previous case, with the difference that in this case, the AC grid sets up the value of the power of the DC system.

- Matlab code for the solution of the power flow of one DC grid connected to three AC grids, using a converter with a constant of losses of 2%, and assuming that the DC grid sets the power of the AC grid in case of conflict.

- Code for the same previous case, but changing the last constrain, so that in case of conflict in the converter because none of the nodes are slack, then the AC grid will make constant the DC power with the value of the AC active power of that node.

The study of the last four codes deals with an interesting discussion about the influence of these two parameters: the model of the converter (with losses or without them), and the election of assigning the DC or the AC power value in case that none of them are fixed.

After running and studying these last four possibilities, the conclusion obtained about the consideration of a converter model with or without losses, is that there are differences observables. For example, when studying the results obtained in AC system 3 (case where the DC grid sets the AC power), the power at the nodes changes in the following magnitude:

AC GRID 3	Converter with no losses	Converter with losses (k=2%)
P_1 (pu)	0.1603	0.2685
P_2 (pu)	0.2512	0.4207
P_3 (pu)	-0.41	-0.6848
Total sum (pu)	0.0042	0.0044

Table 9. Comparison of the results with a converter with and without losses

In addition, at the election of fixing the AC or the DC power in the converter code implementation, the difference is also considerable. An example of this can be seen in the case that the converter is considered with losses. The configuration chosen makes a desirable conflict in the connection between the DC grid and AC system number three, because on that bus, the DC node is power type, and the AC is load node type. As none of them are slack buses, an election between fixing the AC or the DC voltage was made. On the one hand, the admittance matrix obtained is the same for both of them, since the lines parameters are identical. However, there are small differences in the voltage values. At the calculation of the current, when multiplying these two variables, the difference at the result increases. Consequently, at the final calculation of the power, the difference is increased again (because of the multiplication of the voltage), so at the end the total difference is considerable.

This study deals with the final conclusion that a previous detailed study about the constrains to implement, has to be made in advance of the calculation of any power flow solution in order to obtain reliable results.

Bibliography

Matias Ebbe Theisen. Offshore Grid. Technical report, NTNU, July 2011.

Temesgen M. Haileselassie and Kjetil Uhlen. A Fast Method for Load Flow Analysis of DC Power Networks.

Jef Beerten, Stijn Cole and Roniie Belmans. A Sequential AC/DC Power Flow Algorithm for Networks Containing Multi-terminal VSC HVDC Systems.

Hans-Peter Nee and Lennart Angquis. Perspectives on Power Electronics and Grid Solutions for Offshore wind farms. Elforsk rapport 10:96. November, 2010.

Temesgen M.Haileselassie and Kjetil Uhlen. Control of Multi-terminal HVDC and its Impact on Multi-national Power Market. Norwegian University on Science and Technology, NTNU.

Temesgen M.Haileselassie and Kjetil Uhlen. Power Flow analysis of Multi-terminal HVDC Networks. Department of Electric Power Engineering. Norwegian University on Science and Technology, NTNU.

J. Beerten, S. Cole, R. Belmans. Implementation aspects of a Sequential AC/DC Power Flow Computation Algorithm for Multi-Terminal VSC HVDC Systems. Department of Electrical Engineering (ESAT), Division ELECTRA, Katholieke Universiteit Leuven, Belgium.

Temesgen M.Haileselassie and Kjetil Uhlen. Impact of DC Line Voltage Drops on Power Flow of Multi-Terminal VSC-HVDCs using Droop Control. Department of Electric Power Engineering. Norwegian University on Science and Technology, NTNU. July, 2011.

Jonas Persson, STRI AB. Kundur´s Two-Area System. July 1996

Class notes course TET 4115 “Power system analysis”, Norwegian University on Science and Technology, NTNU. Fall 2011

Appendix 1

DC Power flow Matlab code

```
%% Esther Gil Colmenero   %%
%% Specialization Project %%
%%      Fall 2010         %%

%Matlab program that solves DC Power Flow by Newton-Raphson method.
%Cleaning parameters values and screen
    clear all
    clc
%Input data enter by the user refering the nodes and lines of each system.
    %"nodes" matrix contains the data of the nodes.
      %ROWS=>node
%COLUMNS=>node,module_voltage(pu),(1=slack,2=P),Pgenerated(pu),Pdeman(pu)
       nodes=[1 1   1 0 0;
              2 1   2 1 0;
              3 0.9 2 0 0.6];
      [row_nodes,column_nodes]=size(nodes);
    %"lines" matrix contains the data of the lines.
       %ROWS=>line
       %COLUMNS=>origin node, destiny node, Rxij(pu)
       lines=[1 2 0.0108;
              1 3 0.0235;
              2 3 0.0147];
       [row_lines,column_lines]=size(lines);
%Generalization of the types of nodes
    nodesOSC=0;
    nodesP=0;
    for num_rows_nodes=1:row_nodes
        switch(nodes(num_rows_nodes,3))
            case(1)%Slack node
                nodesOSC=nodesOSC+1;
            case(2)%P node
                nodesP=nodesP+1;
        end
    end
```

```
%Calculation of Admitance Matrix Y
    Y=zeros(row_nodes);
    for num_lines_rows=1:row_lines
        node_origin=lines(num_lines_rows,1);
        node_destiny=lines(num_lines_rows,2);

Y(node_origin,node_origin)=Y(node_origin,node_origin)+(1/lines(num_lines_ro
ws,3));%Principal diagonal
        Y(node_origin,node_destiny)=Y(node_origin,node_destiny)-
1/(lines(num_lines_rows,3));

Y(node_destiny,node_destiny)=Y(node_destiny,node_destiny)+(1/(lines(num_lin
es_rows,3)));%Principal diagonal
        Y(node_destiny,node_origin)=Y(node_destiny,node_origin)-
1/(lines(num_lines_rows,3));
    end
% Calculation of power injection calculated
    size_Y=max(size(Y));% Because the Admitance Matrix Y is a square matrix
I calculate its size in a single variable
    Pcalculated=zeros(size_Y,1);
    for row_Y=2:size_Y
        for colum_Y =1:size_Y
Pcalculated(row_Y)=Pcalculated(row_Y)+nodes(row_Y,2)*nodes(colum_Y,2)*Y(row
_Y,colum_Y);%Pcalculated=sum(ui*uj*Yij)
        end
    end
%Calculation of active power injected nodes matrix for nodes type P
        for num_rows_nodes=1:row_nodes
            P(num_rows_nodes)=nodes(num_rows_nodes,4)-
nodes(num_rows_nodes,5);  %Pgi-Pdi
        end
 %Calculation of error vector
     size1=1;
     for num_rows_nodes=2:row_nodes
           if nodes(num_rows_nodes,3)==2  %nodes type P
                error(size1,1)=P(num_rows_nodes)-
Pcalculated(num_rows_nodes);
                size1=max(size(error));
                size1=size1+1;
           end
      end
%Comprobation that the error is smaller than the tolerance
    tolerance=10e-5;% I define a tolerance on the error
    num_iterations=0;
    while max(error)>tolerance % If the condition is true, the program
continues. If it is false the values obtained will be shown
        num_iterations= num_iterations+1;   % Increase the number of
iterations
  %Calculation of Jacobian Matrix (J)
   for i=2:row_nodes
       J(i-1,i-1)=Pcalculated(i)+Y(i,i)*(nodes(i,2))^2; %Jii=Pi+Yii*Ui^2
           for k=2:row_nodes
               if k~=i
                  J(k-1,i-1)=nodes(i,2)*nodes(k,2)*Y(k,i); %Jij= Ui*Uj*Yij
               end
```

```
           end
   end
   %Calculation of the vector of corrections: solution J*incX=error
          incX=0;
          incX=J\error;
   % Actualization of the module of the voltage, from the increases
obtained
           for i=2:row_nodes
                nodes(i,2)=nodes(i,2)+incX(i-1)*nodes(i,2);
           end
     %Calculation of the power injected calculated
        size_Y=max(size(Y));% As the Admitance Matrix is square, I
calculate its size at only one variable
        Pcalculated=zeros(size_Y,1);
        for row_Y=2:size_Y
            for colum_Y =1:size_Y
Pcalculated(row_Y)=Pcalculated(row_Y)+nodes(row_Y,2)*nodes(colum_Y,2)*Y(row
_Y,colum_Y);
            end
        end
    % Calculation of the powers injected matrix nodes
        %Active power, for nodes type P
            for num_rows_nodes=1:row_nodes
                P(num_rows_nodes)=nodes(num_rows_nodes,4)-
nodes(num_rows_nodes,5);  %Pgi-Pdi
            end
     %Calculation of the error after the modification of the data
           error=0;
           size1=1;
             for num_rows_nodes=2:row_nodes
                   if nodes(num_rows_nodes,3)==2 %nodes type P
                        error(size1,1)=P(num_rows_nodes)-
Pcalculated(num_rows_nodes);
                        size1=max(size(error));
                        size1=size1+1;
                   end
             end
    end  %end of the principal loop
%I show the result
     fprintf ('\n----  SOLUTION FOR DC SYSTEM ----\n');
     fprintf('\n  Number of iterations required: %d\n\n ', num_iterations);
     disp ('  Admitance matrix:'); disp (Y);
     disp ('  Jacobian matrix:'); disp (J);
     U=nodes(:,2); P2=U.*(Y*U); disp ('  Power=U.*(Y*U)'); disp (P2);
     I2=Y*U; disp ('  Current=Y*U'); disp (I2);
```

Appendix 2

Single AC grid Power flow Matlab code

```
%% Esther Gil Colmenero   %%
%% Specialization Project %%
%%      Fall 2010         %%
%Matlab program that solves Power Flow by Newton-Raphson method.
%I first clean parameters values and screen
    clear all
    clc
%Input data for a system of 3 nodes, conecting nodes 1-2,1-3 and 2-3
   %"nodes" matrix contains the data of the nodes.
   %ROWS=>node.
   %COLUMNS=>node,module_voltage(pu),angle_voltage(rad),(1=slack,2=PV,3=PQ)
   %Pgenerated(pu),Qgenerated(pu),Pdeman(pu),Qdemand(pu),B(pu)
    nodes=[1 1 0 1 0   0 0    0    0;
           2 1 0 2 1.0 0 0.65 0.36 0;
           3 1 0 3 0   0 0.6  0.35 0];
   [row_nodes,column_nodes]=size(nodes);
   %"lines" matrix contains the data of the lines.
   %ROWS=>line.
   %COLUMNS=>origin node, destiny node, Rxij(pu), Xij(pu), Bij(pu)
    lines=[1 2 0.0108 0.0649 0.066;
           1 3 0.0235 0.0941 0.04;
           2 3 0.0147 0.0566 0.08];
    [row_lines,column_lines]=size(lines);
%I generalize the type of node
    nodesOSC=0;
    nodesPV=0;
    nodesPQ=0;
    for num_rows_nodes=1:row_nodes
        switch(nodes(num_rows_nodes,4))
            case(1)%Node OSC
                nodesOSC=nodesOSC+1;
            case(2)%Nude PV
                nodesPV=nodesPV+1;
            case(3)%Nude PQ
                nodesPQ=nodesPQ+1;
```

```
        end
    end
%Calculation of Admitance Matrix Y
    Y=zeros(row_nodes);
    for num_lines_rows=1:row_lines
        node_origin=lines(num_lines_rows,1);
        node_destiny=lines(num_lines_rows,2);
Y(node_origin,node_origin)=Y(node_origin,node_origin)+(1/(lines(num_lines_r
ows,3)+j*lines(num_lines_rows,4))+(j*lines(num_lines_rows,5)/2));%Principal
diagonal
        Y(node_origin,node_destiny)=Y(node_origin,node_destiny)-
1/(lines(num_lines_rows,3)+j*lines(num_lines_rows,4));

Y(node_destiny,node_destiny)=Y(node_destiny,node_destiny)+(1/(lines(num_lin
es_rows,3)+j*lines(num_lines_rows,4))+(j*lines(num_lines_rows,5)/2));%Princ
ipal diagonal
        Y(node_destiny,node_origin)=Y(node_destiny,node_origin)-
1/(lines(num_lines_rows,3)+j*lines(num_lines_rows,4));
    end
    for num_rows_nodes=1:row_nodes
Y(num_rows_nodes,num_rows_nodes)=Y(num_rows_nodes,num_rows_nodes)+(j*nodes(
num_rows_nodes,9));%At the Principal diagonal of the Admitance Matrix Y
    end
% Calculation of power injection calculated
    size_Y=max(size(Y));% Because the Admitance Matrix Y is a square matrix
I calculate its size in a single variable
    Qcalculated=zeros(size_Y,1);
    Pcalculated=zeros(size_Y,1);
    for row_Y=2:size_Y
        for colum_Y =1:size_Y
Pcalculated(row_Y)=Pcalculated(row_Y)+nodes(row_Y,2)*nodes(colum_Y,2)*norm(
Y(row_Y,colum_Y))*cos(-
nodes(row_Y,3)+nodes(colum_Y,3)+angle(Y(row_Y,colum_Y)));%Pcalculated=
?(ui*uj*Yij*cos(?i-?j-?ij))
            Qcalculated(row_Y)=Qcalculated(row_Y)-
nodes(row_Y,2)*nodes(colum_Y,2)*norm(Y(row_Y,colum_Y))*sin(-
nodes(row_Y,3)+nodes(colum_Y,3)+angle(Y(row_Y,colum_Y)));%Qcalculated=
?(ui*uj*Yij*sin(?i-?j-?ij))
        end
    end
%Calculation of powers injected nodes matrix
    %Active power, for nodes PQ y PV
        for num_rows_nodes=1:row_nodes
            P(num_rows_nodes)=nodes(num_rows_nodes,5)-
nodes(num_rows_nodes,7);  %Pgi-Pdi
        end
   %Reactive power, for nodes PQ
       for num_rows_nodes=1:row_nodes
           if nodes(num_rows_nodes,4)==3 % filter for only calculate on
nodes PQ
            Q(num_rows_nodes)=nodes(num_rows_nodes,6)-
nodes(num_rows_nodes,8); %Qgi-Qdi
           end
       end

 %Calculation of error vector
```

```
     size1=1;
     for num_rows_nodes=2:row_nodes
           if nodes(num_rows_nodes,4)==2|3   %nodes PV y PQ
                error(size1,1)=-
P(num_rows_nodes)+Pcalculated(num_rows_nodes);
                size1=max(size(error));
                size1=size1+1;
           end
      end
       for num_rows_nodes=1:row_nodes
           if nodes(num_rows_nodes,4)==3  %nodes PQ
                error(size1,1)=Q(num_rows_nodes)-
Qcalculated(num_rows_nodes);
                size1=size1+1;
           end
       end
%Comprobation that the error is smaller than the tolerance
    tolerance=10e-5;% I define a tolerance on the error
    num_iterations=0;
   while max(error)>tolerance%If the condition is true, the program
continues. If it is false the values obtained will be shown
        num_iterations= num_iterations+1;%Increase the number of iterations
  %Calculation of Jacobian Matrix (J)
     %I start defining the matrix H,for the cases:'only PQ','only PV' or
'PQ+PV'
for i=2:row_nodes%I start in row 2 because the row 1 corresponds to OSC
node
   H(i-1,i-1)=-imag(Y(i,i))*(nodes(i,2)^2)-Qcalculated(i);%Hii=-Bij*Ui^2-Qi
             for k=2:row_nodes
                 if i~=k
                     H(i-1,k-1)=-
nodes(i,2)*nodes(k,2)*norm(Y(i,k))*sin(angle(Y(i,k))+nodes(i,3)-
nodes(k,3));%Hij=-Ui*Uj*Yij*sen(Tij+Dj-Di))
                 end
             end
         end
         %Ajustment of dimensions for obtaining the Jacobian Matrix
         H1=zeros(nodesPV+nodesPQ);
         H1=H(1:nodesPQ+nodesPV);
    %Definition of matrix N
         for i=row_nodes+1:row_nodes+nodesPQ
             N(i-row_nodes+nodesPV,i-1)=Pcalculated(i-nodesPQ)+real(Y(i-
nodesPQ,i-nodesPQ))*(nodes(i-nodesPQ,2)^2); %Nii=Pi+Gii*Ui^2
           for k=2:row_nodes
               if k~=i-nodesPQ
                 N(k-1,i-1)=nodes(i-nodesPQ,2)*nodes(k,2)*norm(Y(k,i-
nodesPQ))*cos(angle(Y(k,i-nodesPQ))+nodes(i-nodesPQ,3)-nodes(k,3)); %Nij=
Ui*Uj*Yij*cos(Di-Dj+Tij)
               end
           end
         end
          %Ajustment of dimensions for obtaining the Jacobian Matrix
                N1=zeros(nodesPQ+nodesPV,nodesPQ);
               N1=N(1:nodesPQ+nodesPV,nodesPV+nodesPQ+1:nodesPV+nodesPQ*2);
    %Definition of matrix M
        for i=row_nodes+1:row_nodes+nodesPQ
```

```
             M(i-1,i-1-nodesPQ)=Pcalculated(i-nodesPQ)-real(Y(i-nodesPQ,i-
nodesPQ))*(nodes(i-nodesPQ,2)^2); %Mii=Pi-Gii*Ui^2
         for k=2:row_nodes
           if k~=i-nodesPQ
              M(i-1,k-1)=-nodes(i-nodesPQ,2)*nodes(k,2)*norm(Y(i-
nodesPQ,k))*cos(angle(Y(i-nodesPQ,k))-nodes(i-nodesPQ,3)+nodes(k,3));%Mij=-
Nij
               end
           end
        end
        %Ajustment of dimensions for obtaining the Jacobian Matrix
        M1=zeros(nodesPQ,nodesPV+nodesPQ);
        M1=M(nodesPQ+nodesPV+1:nodesPQ*2+nodesPV,:);
    %Definition of matrix L
        for i=row_nodes+1:row_nodes+nodesPQ
            L(i-1,i-1)=Qcalculated(i-nodesPQ)-imag(Y(i-nodesPQ,i-
nodesPQ))*(nodes(i-nodesPQ,2)^2); %Lii=Qii-Bii*Ui^2
            for k=row_nodes+1:row_nodes+nodesPQ
                   if k~=i
                      L(k-1,i-1)=-nodes(i-nodesPQ,2)*nodes(k-
nodesPQ,2)*norm(Y(i-nodesPQ,k-nodesPQ))*sin(angle(Y(i-nodesPQ,k-
nodesPQ))+nodes(k-nodesPQ,3)-nodes(i-nodesPQ,3)); %Lij=Hij
                   end
               end
        end
           %Ajustment of dimensions for obtaining the Jacobian Matrix
            L1=zeros(nodesPQ);
L1=L(nodesPQ+nodesPV+1:nodesPQ*2+nodesPV,nodesPV+nodesPQ+1:nodesPV+nodesPQ*
2);
    %Definition of Jacobian Matrix
      J=[H N1;M1 L1];
    %Calculation of the vector of corrections: solution J*incX=error
          incX=0;
          incX=J\error;
    %Actualization of the voltage of the nodes
        %First I modify the angle of the voltages
              num_incX=1;
              for i=2:row_nodes
                      nodes(i,3)=nodes(i,3)+incX(num_incX);
                      num_incX=num_incX+1;
              end
    %Actualization of the module of the voltage,from the increases obtained
           for i=row_nodes-nodesPQ+1:row_nodes
                nodes(i,2)=nodes(i,2)+incX(i+nodesPQ-1)*nodes(i,2);
            end
     %Calculation of the powers injected calculated
        size_Y=max(size(Y));% As the Admitance Matrix is square, I
calculate its size at only one variable
        Qcalculated=zeros(size_Y,1);
        Pcalculated=zeros(size_Y,1);
        for row_Y=2:size_Y
            for colum_Y =1:size_Y
Pcalculated(row_Y)=Pcalculated(row_Y)+nodes(row_Y,2)*nodes(colum_Y,2)*norm(
Y(row_Y,colum_Y))*cos(-
nodes(row_Y,3)+nodes(colum_Y,3)+angle(Y(row_Y,colum_Y)));
```

```
                Qcalculated(row_Y)=Qcalculated(row_Y)-
nodes(row_Y,2)*nodes(colum_Y,2)*norm(Y(row_Y,colum_Y))*sin(-
nodes(row_Y,3)+nodes(colum_Y,3)+angle(Y(row_Y,colum_Y)));
            end
        end
    % Calculation of the powers injected matrix nodes
        %Active power, for nodes PQ and PV
            for num_rows_nodes=1:row_nodes
                P(num_rows_nodes)=nodes(num_rows_nodes,5)-
nodes(num_rows_nodes,7);  %Pgi-Pdi
            end
       %Reactive power, for nodes PQ
           for num_rows_nodes=1:row_nodes
               if nodes(num_rows_nodes,4)==3 % Filter for only calculation
on nodes PQ
                Q(num_rows_nodes)=nodes(num_rows_nodes,6)-
nodes(num_rows_nodes,8); %Qgi-Qdi
               end
           end
       %Calculation of the error after the modification of the data
           error=0;
           size1=1;
             for num_rows_nodes=2:row_nodes
                   if nodes(num_rows_nodes,4)==2|3   %nodes PV
                        error(size1,1)=P(num_rows_nodes)-
Pcalculated(num_rows_nodes);
                        size1=max(size(error));
                        size1=size1+1;
                   end
              end
               for num_rows_nodes=1:row_nodes
                   if nodes(num_rows_nodes,4)==3  %nodes PQ
                        error(size1,1)=Q(num_rows_nodes)-
Qcalculated(num_rows_nodes);
                        size1=size1+1;
                   end;
               end
       %Transformation of the voltage
           Umod=nodes(:,2);
           Uang=nodes(:,3);
           for i=1:row_nodes
            U(i,1)=complex(Umod(i)*cos(Uang(i)),Umod(i)*sin(Uang(i)));
           end
end %end of the principal loop
%Ones the the condition error<tolerance is true, I show the result
    fprintf('\n----  SOLUTION FOR AC SYSTEM  ----\n\n');
    disp ('  Admitance matrix:'); disp (Y);
    disp ('  Jacobian matrix:'); disp (J);
    disp ('  Number of iterations required:'); disp (num_iterations);
    disp ('  The solution after these iterations is the next repart of
voltages:'); disp (U);
    disp ('  Absolutes values of voltages:'); disp (abs(U));
    P2=U.*(Y*U); disp ('  Power=U.*(Y*U)'); disp (P2);
    I2=Y*U; disp ('  Current=Y*U'); disp (I2);
```

Appendix 3

Three AC systems power flow Matlab code

```
%% Esther Gil Colmenero   %%
%% Specialization Project %%
%%      Fall 2010         %%

%Matlab program that solves AC Power Flow by Newton-Raphson method.

%Cleaning of the parameters values and screen
    clear all
    clc

%Number of total systems. Input enter by the user refering the number of
power flows to solve
systems=3;

for num_systems=1:1:systems
        %I delete the variables in each system
        clear nodes lines Y P Q error J N M L H U Pcalculated Qcalculated

%Input data enter by the user refering the nodes and lines of each system.
        %"nodes" matrix contains the data of the nodes.
           %ROWS=>node.

%COLUMNS=>node,module_voltage(pu),angle_voltage(rad),(1=slack,2=PV,3=PQ),Pg
enerated(pu),Qgenerated(pu),Pdeman(pu),Qdemand(pu),B(pu)

        %"lines" matrix contains the data of the lines.
           %ROWS=>line.
           %COLUMNS=>origin node, destiny node, Rxij(pu), Xij(pu), Bij(pu)
    switch num_systems
        case 1, %Input data for system number 1
            nodes=[1 1 0 1 0   0 0    0    0;
```

```
                2 1 0 2 1.0 0 0.65 0.36 0;
                3 1 0 3 0   0 0.6  0.35 0];
         lines=[1 2 0.0108 0.0649 0.066;
                1 3 0.0235 0.0941 0.04;
                2 3 0.0147 0.0566 0.08];

        case 2, %Input data for system number 2
             nodes=[1 1 0 1 0   0 0   0   0;
                    2 1 0 2 1.0 0 0.8 0.8 0;
                    3 1 0 3 0   0 0.8  0.8 0];
             lines=[1 2 0.02 0.06 0.06;
                    1 3 0.02 0.09 0.04;
                    2 3 0.05 0.05 0.08];

        case 3,%Input data for system number 3
            nodes=[1 1 0 1 0   0 0    0    0;
                   2 1 0 2 1.0 0 0.65 0.36 0;
                   3 1 0 3 0   0 0.6  0.35 0];
            lines=[1 2 0.0108 0.0649 0.066;
                   1 3 0.0235 0.0941 0.04;
                   2 3 0.0147 0.0566 0.08];
    end %end case for selecting AC system

   [row_nodes,column_nodes]=size(nodes);
   [row_lines,column_lines]=size(lines);

%Generalization of the type of node
    nodesOSC=0;
    nodesPV=0;
    nodesPQ=0;
    for num_rows_nodes=1:row_nodes
        switch(nodes(num_rows_nodes,4))
            case(1)%Slack node
                nodesOSC=nodesOSC+1;
            case(2)%Node PV
                nodesPV=nodesPV+1;
            case(3)%Node PQ
                nodesPQ=nodesPQ+1;
        end
    end

%Calculation of Admitance Matrix Y
    Y=zeros(row_nodes);
    for num_lines_rows=1:row_lines
        node_origin=lines(num_lines_rows,1);
        node_destiny=lines(num_lines_rows,2);

Y(node_origin,node_origin)=Y(node_origin,node_origin)+(1/(lines(num_lines_r
ows,3)+j*lines(num_lines_rows,4))+(j*lines(num_lines_rows,5)/2));%Principal
diagonal
        Y(node_origin,node_destiny)=Y(node_origin,node_destiny)-
1/(lines(num_lines_rows,3)+j*lines(num_lines_rows,4));

Y(node_destiny,node_destiny)=Y(node_destiny,node_destiny)+(1/(lines(num_lin
```

```
es_rows,3)+j*lines(num_lines_rows,4))+(j*lines(num_lines_rows,5)/2));%Princ
ipal diagonal
        Y(node_destiny,node_origin)=Y(node_destiny,node_origin)-
1/(lines(num_lines_rows,3)+j*lines(num_lines_rows,4));
    end
    for num_rows_nodes=1:row_nodes

Y(num_rows_nodes,num_rows_nodes)=Y(num_rows_nodes,num_rows_nodes)+(j*nodes(
num_rows_nodes,9));%At the Principal diagonal of the Admitance Matrix Y
    end
% Calculation of power injection calculated
    size_Y=max(size(Y));% Because the Admitance Matrix Y is a square matrix
I calculate its size in a single variable
    Qcalculated=zeros(size_Y,1);
    Pcalculated=zeros(size_Y,1);
    for row_Y=2:size_Y
        for colum_Y =1:size_Y

Pcalculated(row_Y)=Pcalculated(row_Y)+nodes(row_Y,2)*nodes(colum_Y,2)*norm(
Y(row_Y,colum_Y))*cos(-
nodes(row_Y,3)+nodes(colum_Y,3)+angle(Y(row_Y,colum_Y)));%Pcalculated=sum(u
i*uj*Yij*cos(angi-angj-angij))
            Qcalculated(row_Y)=Qcalculated(row_Y)-
nodes(row_Y,2)*nodes(colum_Y,2)*norm(Y(row_Y,colum_Y))*sin(-
nodes(row_Y,3)+nodes(colum_Y,3)+angle(Y(row_Y,colum_Y)));%Qcalculated=sum(u
i*uj*Yij*sin(angi-angj-angij))
        end
    end
%Calculation of powers injected nodes matrix
    %Active power, for nodes PQ y PV
        for num_rows_nodes=1:row_nodes
            P(num_rows_nodes)=nodes(num_rows_nodes,5)-
nodes(num_rows_nodes,7);  %Pgi-Pdi
        end
   %Reactive power, for nodes PQ
       for num_rows_nodes=1:row_nodes
           if nodes(num_rows_nodes,4)==3 % filter for only calculating
nodes PQ
            Q(num_rows_nodes)=nodes(num_rows_nodes,6)-
nodes(num_rows_nodes,8); %Qgi-Qdi
           end
       end
 %Calculation of error vector
     size1=1;
     for num_rows_nodes=2:row_nodes
           if nodes(num_rows_nodes,4)==2|3   %nodes PV y PQ
               error(size1,1)=-
P(num_rows_nodes)+Pcalculated(num_rows_nodes);
               size1=max(size(error));
               size1=size1+1;
           end
      end
       for num_rows_nodes=1:row_nodes
           if nodes(num_rows_nodes,4)==3  %nodes PQ
               error(size1,1)=Q(num_rows_nodes)-
Qcalculated(num_rows_nodes);
```

```
                size1=size1+1;
            end
        end
%Comprobation that the error is smaller than the tolerance
    tolerance=10e-5;% Definition of a tolerance on the error
    num_iterations=0;
    while max(error)>tolerance % If the condition is true, the program
continues. If it is false the values obtained will be shown
        num_iterations= num_iterations+1;%Increase the number of iterations
  %Calculation of Jacobian Matrix (J)
     %I start defining the matrix H,for the cases:'only PQ','only PV' or
'PQ+PV'
         for i=2:row_nodes % I start in row 2 because the row 1 corresponds
to slack node
             H(i-1,i-1)=-imag(Y(i,i))*(nodes(i,2)^2)-Qcalculated(i);%Hii=-
Bij*Ui^2-Qi
             for k=2:row_nodes
                 if i~=k
                     H(i-1,k-1)=-
nodes(i,2)*nodes(k,2)*norm(Y(i,k))*sin(angle(Y(i,k))+nodes(i,3)-
nodes(k,3));%Hij=-Ui*Uj*Yij*sen(Tij+Dj-Di))
                 end
             end
         end
     %Ajustment of dimensions for obtaining the Jacobian Matrix definitive
         H1=zeros(nodesPV+nodesPQ);
         H1=H(1:nodesPQ+nodesPV);

    %Definition of matrix N
         for i=row_nodes+1:row_nodes+nodesPQ
             N(i-row_nodes+nodesPV,i-1)=Pcalculated(i-nodesPQ)+real(Y(i-
nodesPQ,i-nodesPQ))*(nodes(i-nodesPQ,2)^2); %Nii=Pi+Gii*Ui^2
           for k=2:row_nodes
               if k~=i-nodesPQ
                  N(k-1,i-1)=nodes(i-nodesPQ,2)*nodes(k,2)*norm(Y(k,i-
nodesPQ))*cos(angle(Y(k,i-nodesPQ))+nodes(i-nodesPQ,3)-nodes(k,3)); %Nij=
Ui*Uj*Yij*cos(Di-Dj+Tij)
               end
           end
         end
      %Ajustment of dimensions for obtaining the Jacobian Matrix definitive
                N1=zeros(nodesPQ+nodesPV,nodesPQ);
N1=N(1:nodesPQ+nodesPV,nodesPV+nodesPQ+1:nodesPV+nodesPQ*2);
    %Definition of matrix M
        for i=row_nodes+1:row_nodes+nodesPQ
            M(i-1,i-1-nodesPQ)=Pcalculated(i-nodesPQ)-real(Y(i-nodesPQ,i-
nodesPQ))*(nodes(i-nodesPQ,2)^2); %Mii=Pi-Gii*Ui^2
           for k=2:row_nodes
               if k~=i-nodesPQ
                  M(i-1,k-1)=-nodes(i-nodesPQ,2)*nodes(k,2)*norm(Y(i-
nodesPQ,k))*cos(angle(Y(i-nodesPQ,k))-nodes(i-nodesPQ,3)+nodes(k,3));
%Mij=-Nij
               end
           end
        end
```

```
        %Ajustment of dimensions for obtaining the Jacobian Matrix
definitive
        M1=zeros(nodesPQ,nodesPV+nodesPQ);
        M1=M(nodesPQ+nodesPV+1:nodesPQ*2+nodesPV,:);

    %Definition of matrix L
        for i=row_nodes+1:row_nodes+nodesPQ
            L(i-1,i-1)=Qcalculated(i-nodesPQ)-imag(Y(i-nodesPQ,i-
nodesPQ))*(nodes(i-nodesPQ,2)^2); %Lii=Qii-Bii*Ui^2
            for k=row_nodes+1:row_nodes+nodesPQ
                   if k~=i
                      L(k-1,i-1)=-nodes(i-nodesPQ,2)*nodes(k-
nodesPQ,2)*norm(Y(i-nodesPQ,k-nodesPQ))*sin(angle(Y(i-nodesPQ,k-
nodesPQ))+nodes(k-nodesPQ,3)-nodes(i-nodesPQ,3)); %Lij=Hij
                   end
               end
        end
           %Ajustment of dimensions for obtaining the Jacobian Matrix
definitive
            L1=zeros(nodesPQ);

L1=L(nodesPQ+nodesPV+1:nodesPQ*2+nodesPV,nodesPV+nodesPQ+1:nodesPV+nodesPQ*
2);
    %Definition of Jacobian Matrix
      J=[H N1;M1 L1];

    %Calculation of the vector of corrections: solution J*incX=error
          incX=0;
          incX=J\error;

    %Actualization of the voltage of the nodes
        %Modification of the angle of the voltages
              num_incX=1;
              for i=2:row_nodes
                      nodes(i,3)=nodes(i,3)+incX(num_incX);
                      num_incX=num_incX+1;
              end
  % Actualization of the module of the voltage, from the increases obtained
           for i=row_nodes-nodesPQ+1:row_nodes
                nodes(i,2)=nodes(i,2)+incX(i+nodesPQ-1)*nodes(i,2);
            end

     %Calculation of the powers injected calculated
        size_Y=max(size(Y));% As the Admitance Matrix is square, I
calculate its size at only one variable
        Qcalculated=zeros(size_Y,1);
        Pcalculated=zeros(size_Y,1);
        for row_Y=2:size_Y
            for colum_Y =1:size_Y
Pcalculated(row_Y)=Pcalculated(row_Y)+nodes(row_Y,2)*nodes(colum_Y,2)*norm(
Y(row_Y,colum_Y))*cos(-
nodes(row_Y,3)+nodes(colum_Y,3)+angle(Y(row_Y,colum_Y)));
                Qcalculated(row_Y)=Qcalculated(row_Y)-
nodes(row_Y,2)*nodes(colum_Y,2)*norm(Y(row_Y,colum_Y))*sin(-
nodes(row_Y,3)+nodes(colum_Y,3)+angle(Y(row_Y,colum_Y)));
```

```
            end
        end
    % Calculation of the powers injected matrix nodes
        %Active power, for nodes PQ and PV
            for num_rows_nodes=1:row_nodes
                P(num_rows_nodes)=nodes(num_rows_nodes,5)-
nodes(num_rows_nodes,7);  %Pgi-Pdi
            end
  %Reactive power, for nodes PQ
    for num_rows_nodes=1:row_nodes
      if nodes(num_rows_nodes,4)==3%Filter for only calculation on nodes PQ
                Q(num_rows_nodes)=nodes(num_rows_nodes,6)-
nodes(num_rows_nodes,8); %Qgi-Qdi
               end
           end

       %Calculation of the error after the modification of the data
           error=0;
           size1=1;
             for num_rows_nodes=2:row_nodes
                   if nodes(num_rows_nodes,4)==2|3  %nodes PV
                        error(size1,1)=P(num_rows_nodes)-
Pcalculated(num_rows_nodes);
                        size1=max(size(error));
                        size1=size1+1;
                   end
             end
               for num_rows_nodes=1:row_nodes
                   if nodes(num_rows_nodes,4)==3  %nodes PQ
                        error(size1,1)=Q(num_rows_nodes)-
Qcalculated(num_rows_nodes);
                        size1=size1+1;
                   end;
               end
       %Transformation of the voltage
           Umod=nodes(:,2);
           Uang=nodes(:,3);
           for i=1:row_nodes
            U(i,1)=complex(Umod(i)*cos(Uang(i)),Umod(i)*sin(Uang(i)));
           end
end %end of the principal loop

%Ones the the condition error<tolerance is true, I show the result
     fprintf('\n\n\n----  SOLUTION FOR SYSTEM %d  ----\n\n', num_systems);
     disp ('  Admitance matrix:'); disp (Y);
     disp ('  Jacobian matrix:'); disp (J);
     fprintf('  Number of iterations required: %d\n\n ', num_iterations);
     disp (' The solution after these iterations is the next repart of
voltages:'); disp (U);
     disp ('  Absolutes values of voltages:'); disp (abs(U));
     P2=U.*(Y*U); disp ('  Power=U.*(Y*U)'); disp (P2);
     I2=Y*U; disp ('  Current=Y*U'); disp (I2);
 end %end of all the AC systems
```