Development of an Energy Management System for Loop Power Flow Controllers using heuristic methods

by

Elie Abi Habib

Submitted to the Department of Electrical Engineering, Electronics, Computers and Systems
in partial fulfillment of the requirements for the degree of
the Master Course in Electrical Energy Conversion and Power Systems
at the
UNIVERSIDAD DE OVIEDO
July 2015

Author . . . . . .

Certified by . . . . . .

Jose Manuel Cano Rodriguez
Associate Professor
Thesis Supervisor

Development of an Energy Management System for Loop Power Flow Controllers using heuristic methods

by

Elie Abi Habib

Submitted to the Department of Electrical Engineering, Electronics, Computers and Systems
on July 22, 2015, in partial fulfillment of the
requirements for the degree of
the Master Course in Electrical Energy Conversion and Power Systems

Abstract

The loops power flow control is an application that is used to a distribution radial feeders in order to improve the efficiency of the grid and to provide an adequate voltage support. This thesis will basically provide an energy management system in order to deliver an optimum reference values for active and reactive power of the LPC in order to reduce the losses to an optimum value. The developed algorithm can be implemented to any radial feeder, but in this case a basic six node feeder will be used just to have a lower simulation time.

The first part consist of the implementation of the energy management system, loop power flow control, in Matlab code with a generalized algorithm so it can work with any radial network. The code should display the reference values of the active and the two reactive power values and the optimum feeder losses.

The second part will consist of optimizing the constants involved in the particle swarm optimization so the solution of the first part will be as fast as possible without losing the efficiency. The third part will be the evaluation of the economical savings expected when applying a LPC to the specified grid using a daily or a weekly load profile for example. This will help to find out if the installation of a loop power control at the site has an advantage or not.

Keywords—Energy management system, Energy efficiency, Voltage Control, Loop power flow controllers, Particle swarm optimization, Active and Reactive power control, Random radial feeder.

Thesis Supervisor: Jose Manuel Cano Rodriguez
Title: Associate Professor

Acknowledgments

With the greatest honor, I reserve this paragraph of gratitude and appreciation to all those who contributed in any way to the success and implementation of the master thesis.

I would like to take this opportunity to express my deep gratitude to all my teachers of the electrical energy conversion Master.

Contents

List of Figures

List of Tables

Chapter 1
Introduction

The use of flexible AC transmission systems at the distribution grid is today a major interest. The decrease of power electronics cost and the increase of communications within the grid make anything possible to find a solution in order to adopt and optimize the feeders. One of the solution is the loop power flow control which allows the retrofit of radial grid into looped networks at an acceptable cost [1].

The renewable energy and the utilization of co-generation systems, both are expected to improve the efficiency of energy applications but a lot of distributed generation will be installed for feeder imbalance which can lead to some difficulties in order to maintain the proper voltage. A loop power flow controller (LPC) shall be implemented using a back to back converter (BTB) will control the loop of the radial feeder [2].

These devices provide a dynamic control of the voltages along the feeders and an optimization of the distribution losses through a balanced use of conductors without any increase in short circuit current [1].

An energy management system (EMS) is used to command the device in order to achieve the optimization of the network [1].

The basic concept of the loop power flow control is as follows [2]:

- Aims for free access to a distributed power supply.
- System responds flexibly to unbalanced load between feeders, and makes effective use of equipment.
- To enable this, the system is constructed in the shape of a loop from a radial.
- A loop distribution system is provided without altering existing systems such as the protection system, except for loop points.

The figure below shows the converter used to make the connection of the 2 radial feeder into a loop.

Figure 1: Structure of LPF control

Objectives of the MTh

This project is devoted to the development of an Energy Management System (EMS) for a Loop Power Flow Controller (LPC). The use of LPCs in the distribution grid turns traditional radial operated feeders in meshed networks allowing a noticeable reduction of distribution losses as well as providing dynamic voltage support (especially important in the increasing connection of Distributed Generation (DG) resources). However, the fast dynamics of the converters used in these configurations do not significantly affect the protection system, as their contribution to short-circuit currents can be practically neglected.

The EMS will use heuristic techniques to solve the optimization problem. The algorithm will be suitable for the application in real-time controllers, as well as to assess the benefits of the use of these devices at a particular location.

State of the Art

In japan, a 6.6 kV overhead distribution system were constructed as radial feeder. In these system each feeder voltage is controlled between a value of 6V for each 100V and 20V for the standard voltage of 200, where the voltage should remain for each customer. A line voltage compensator (LDC) maintained proper voltage for a bank of about six feeders. In Japan, generally one LDC control performs voltage regulation for about six feeders. An LDC is installed on the substation transformer in the distribution substation regulates line voltage. The LDC estimates the voltage drop of the lines from the line current and controls the voltage rate of the transformer, the line voltages remain within a proper range. The load imbalance of each feeder therefore had a negative impact on voltage stability [3].

In the near future, many distributed generators will be connected to the grid to provide power. The reverse power flow of the distributed generators will rise to the upper limit of the voltage range and on the other hand, a disconnection of the distributed generators with faults cause the line to be overload or under-voltage [3]. In this case, it has been proposed that loop or mesh distribution systems be designed to balance the power flow and regulate voltage. However, this will increase short-circuit current in the distribution system, and fault location detection methods for loop distribution systems have not yet been established. A loop power flow controller (LPC) using a pulse wide modulation (PWM) AC-DC-AC converter is able to control loop distribution systems without increasing short-circuit current. LPC loop distribution systems can be used in present protection method. The question here is whether the LPC is aware of the current status of the distribution system. A loop power flow control method using feeder power flow information has been proposed [3].

The equipment to achieve the basic concept to achieve loop distribution is called as a loop power flow controller (LPC), which replaces the sectionalizer at the loop points [3].

The electric energy industry has been regulated through national or municipal governors. Presently, open transmission access is a new legal requirement within the electric power industry since the deregulation is an inevitable outcome of a free market society that thrives on competition. In such a circumstance power system dynamic performances more and more occupy the interest of power system engineers. As a result, FACTS technology is being prompted as a means to extend the capacity of existing power transmission networks without adding new transmission lines. FACTS introduces new degree of freedom into the operation of power systems. This extra flexibility permits the independent adjustment of certain system variables such as power flows. This paper describes a loss minimizing control by a FACTS device which is capable of controlling the line power flow arbitrarily. In addition it is expected that the control simultaneously results in augmentation of voltage stability [9].

Concerning this topic, four papers has been published:

- Application of loop power flow controllers for power demand optimization at industrial customer sites, [1].
- Loop power flow control and voltage characteristics of distribution system for distributed generation including PV system, 2003, [2].
- Autonomous loop power flow control for distribution system, 2001, [3].
- Loop power flow control to minimize power losses and augment voltage stability, 1998, [9].

Chapter 2

Loop power flow control

Radial feeders can be looped using different converter technologies, in this case a back to back converter in used. As well, two or more feeders can be linked though the converter, the device is called a solid-state transformer. The following figure show the LPF control system [1]:

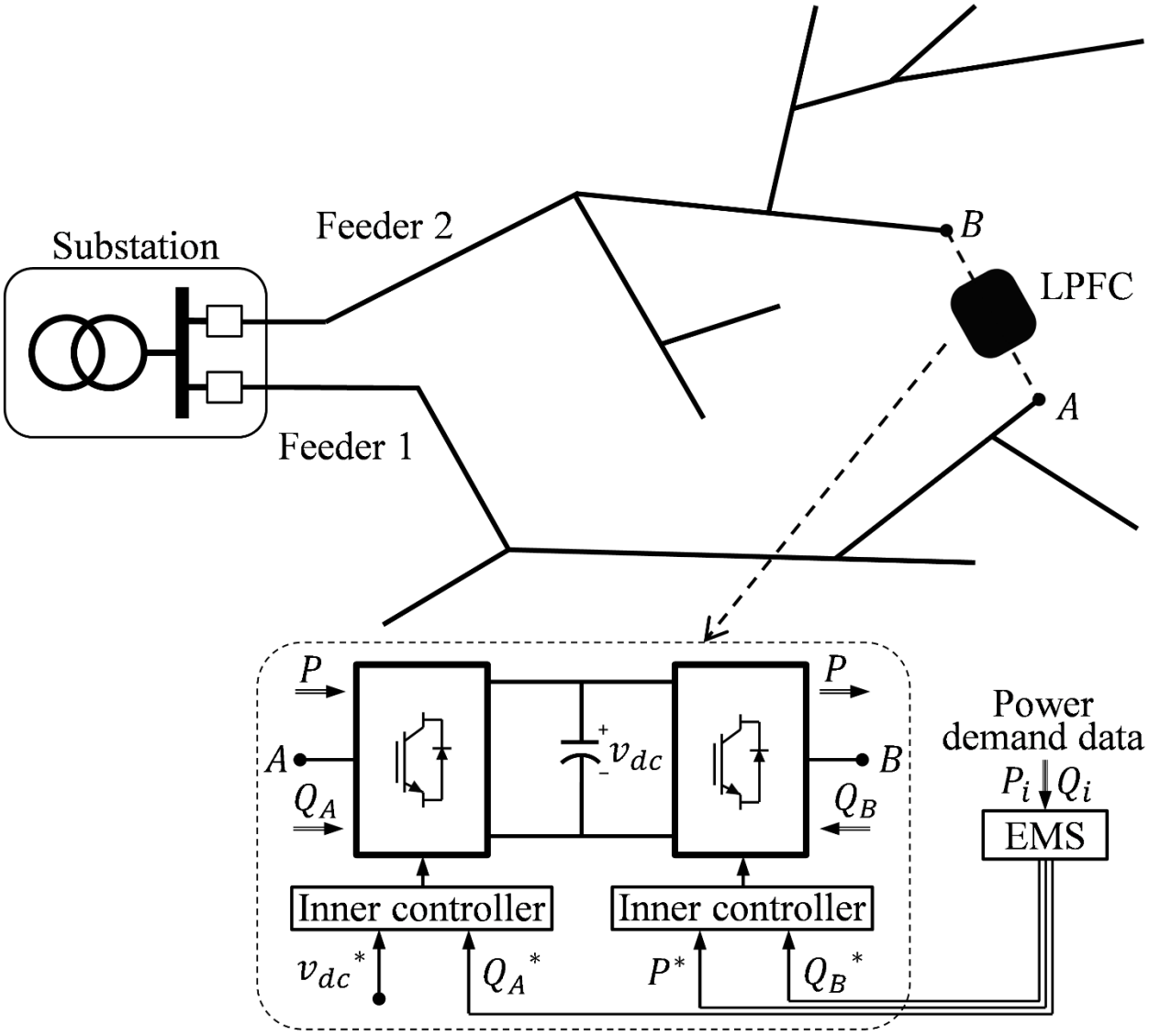

Figure 2: Looped network using a BTB converter as a LPF control

The back to back converter will regulate the active power flow between the two terminal node and the reactive power at each terminal. The three values will be the reference for the energy management system that should be considered in real time.

The EMS has to determine the reference values in order to be able to optimize the operation of the system. One these reference values are calculated, the power flow of the whole system can be

determined including voltage and current. The assessment of different candidate solution for the active and reactive power reference value is at the core of the heuristic used to achieve the optimization of the system. The grid internal variables can be used to define the appropriate optimization function (OF) [1].

The following chart represent the EMS algorithm which is the ladder technique that is used the build the Matlab code:

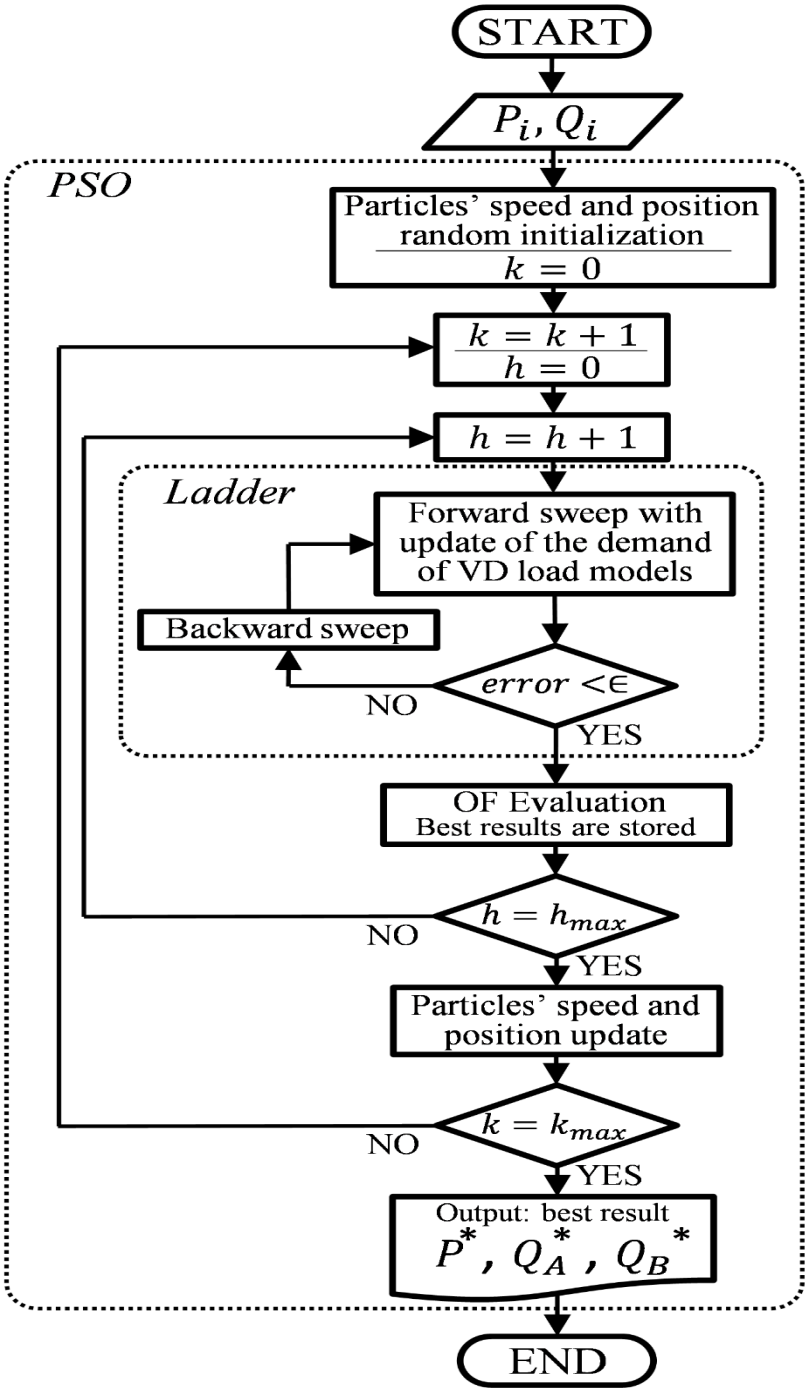

Figure 3: Flowchart of the EMS algorithm

The optimization of the system focus the minimization of the distribution losses among the grid using the OF with the function f which lead to a minimization of the electricity cost with a guarantee of an acceptable voltage level at any operating point of the feeder.

To assure what is needed or requested by any operator the following norms can be followed and added to the algorithm:

$$f(P^*, Q_1^*, Q_2^*) = c_p\left(\sum_{ij} R_{ij} I_{ij}^2\right) + c_q\left(\sum_{ij} X_{ij} I_{ij}^2\right)$$

$$V_{i,min} \leq V_i \leq V_{i,max}$$

$$S_1 \leq S_{1n}\ ;\ S_2 \leq S_{2n}$$

Where R_{ij} and X_{ij} refer to the resistance and the reactance of the feeders connected between the two node i and j, also the transformer impedances is included in the impedance calculation. These two coefficient c_p and c_q represent the unitary cost for both active and reactive power. The voltage limits $V_{i,min}$ and $V_{i,max}$ represent the minimum and maximum value of the voltage at each node. The rated power of the inverter are S_{1n} and S_{2n} which are the maximum power allowed.
The algorithm and the flowchart developed contain two important and interesting feature. The first one is the ladder technique, forward and backward sweep, which is used for the power flow analysis of the radial feeder so it's an easy way to calculate the voltage and current for example at each node of the radial feeder. The second one is the application of the heuristic technique such as the particle swarm optimization (PSO) to solve the optimization problem and it's well suited for practical real-time implementation due to the fact that it's has a fast and good convergence characteristic [1].

Particle swarm optimization

Particle swarm optimization (PSO) is a population based stochastic optimization technique developed by Dr. Eberhart and Dr. Kennedy in 1995, inspired by social behavior of bird flocking or fish schooling. PSO shares many similarities with evolutionary computation techniques such as Genetic Algorithms (GA). The system is initialized with a population of random solutions and searches for optima by updating generations. However, unlike GA, PSO has no evolution operators such as crossover and mutation. In PSO, the potential solutions, called particles, fly through the problem space by following the current optimum particles [4].

The first practical application of PSO was in the field of neural network training and was reported together with the algorithm itself (Kennedy and Eberhart). Many more areas of application have been explored ever since, including telecommunications, control, data mining, design, combinatorial optimization, power systems, signal processing, and many others. To date, there are hundreds of publications reporting applications of particle swarm optimization algorithms. For a review. Although PSO has been used mainly to solve unconstrained, single-objective optimization problems, PSO algorithms have been developed to solve constrained problems, multi-objective

optimization problems, problems with dynamically changing landscapes, and to find multiple solutions [10].

A number of research directions are currently pursued, including [10]:

- Theoretical aspects
- Matching algorithms (or algorithmic components) to problems
- Application to more and/or different class of problems
- Parameter selection
- Comparisons between PSO variants and other algorithms
- New variants

The advantages of particle swarm optimization [11] [12]:

- PSO is easier to implement and there are fewer parameters to adjust.
- PSO based on the intelligence and it is applied on both scientific research and engineering.
- Every particle remembers its own previous best value as well as the neighborhood best; therefore, it has a more effective memory capability.
- PSO have no mutation and overlapping calculation. The search can be take place by the speed of the particle. Most optimist particle can able to transmit the information onto the other particles during the development of several generations, and the speed of researching is faster.
- PSO accepts the real number code, and that is decided directly by the solution. Calculation in PSO is simpler and efficient in global search
- More efficient in maintaining the diversity of the swarm (more similar to the ideal social interaction in a community), since all the particles use the information related to the most successful particle in order to improve themselves.

The disadvantages of particle swarm optimization [12]:

- It is slow convergence in refined search stage and weak local search ability.
- The method cannot work on the problems of non-coordinate systems like the solution of energy field and the moving rules for the particles in the energy field.

Each particle keeps track of its coordinates in the problem space which are associated with the best solution (fitness) it has achieved so far. (The fitness value is also stored.) This value is called pbest. Another "best" value that is tracked by the particle swarm optimizer is the best value, obtained so far by any particle in the neighbors of the particle. This location is called lbest. When a particle takes all the population as its topological neighbors, the best value is a global best and is called gbest [4].

The particle swarm optimization concept consists of, at each time step, changing the velocity of (accelerating) each particle toward its pbest and lbest locations (local version of PSO). Acceleration is weighted by a random term, with separate random numbers being generated for acceleration toward pbest and lbest locations.

In past several years, PSO has been successfully applied in many research and application areas. It is demonstrated that PSO gets better results in a faster, cheaper way compared with other methods [4].

Another reason that PSO is attractive is that there are few parameters to adjust. One version, with slight variations, works well in a wide variety of applications. Particle swarm optimization has been used for approaches that can be used across a wide range of applications, as well as for specific applications focused on a specific requirement [4].

The PSO is a meta-heuristic based on the behavior of natural systems that has been recently applied with a large success in different fields of technology, some of its features make the PSO interesting for the EMS. First thing is that no assumption have to be made on the characteristics of the problem, this will allow the engineers to adapt the controller to the special needs of each application without any concern of the solver. Second thing is the use of a fixed number of particles that opens the possibility of using FPGA to parallelize computation, which is an important feature to comply with the low latency requirements demanded by this application [1].

The equation of the PSO for each particle is the following:

$$v_h^{k+1} = w v_h^k + c_1 r_1 \left({x_{b\_h}}^k - {x_h}^k \right) + c_2 r_2 \left({x_{b\_g}}^k - {x_h}^k \right)$$

Where h represent a possible solution called a particle which is characterized by its position x_h, and its speed v_h within the space of solutions. For this case of the loops power flow control, the position will have three dimension which are the active and the two reactive power. The initial position and speed of each particle are randomly selected in order to be able to start the algorithm. At each iteration k, the performance of each particle is assessed according to the value of the OF for its position. During the iteration process, the best position of each particle ${x_{b_h}}^k$ and the best global position of the swarm ${x_{b_g}}^k$ are also stored so the speed of each particle will be updated according to the above equation. The two coefficient r_1 and r_2 are a random number selected between the range of 0 and 1 for each particle and iteration done. The constant w is the inertia weight and it determines the influence of the past speed in the calculation. The final two coefficient c_1 and c_2 are also constant called cognitive and social weights, they determine the contribution of the best past result of each particle and the best result of the whole swarm to the new speed. Finally, the position of each particle can be updated respectively to reach a new set of improved solution according the new above equation [1]:

$${x_h}^{k+1} = {x_h}^k + {v_h}^{k+1}$$

Ladder Iterative Technique

The power-flow analysis of a distribution feeder is similar to that of an interconnected transmission system. Typically, what will be known prior to the analysis will be the three-phase voltages at the substation and the complex power of all of the loads and the load model (constant complex power, constant impedance, constant current, or a combination). Sometimes the input complex power supplied to the feeder from the substation is also known [5].

Because a distribution feeder is radial, iterative techniques commonly used in transmission network power-flow studies are not used because of poor convergence characteristics. Instead, an iterative technique specifically designed for a radial system is used [5].

A power-flow analysis of a feeder can determine the following by phase and total three-phase [5]:
• Voltage magnitudes and angles at all nodes of the feeder
• Line flow in each line section specified in kW and kVAr, amps and degrees, or amps and power factor
• Power loss in each line section
• Total feeder input kW and kvar
• Total feeder power losses
• Load kW and kvar based upon the specified model for the load

The poor convergence characteristics of traditional power flow method in radial networks like the Newton-Raphson method make the application of iterative technique advisable. The power demand can be defined for the active and reactive power at each node in the complex form as following [1]:

$$S_i = P_i + jQ_i$$

For the case of the nodes which are connected with the LPF control, a initial virtual load should be added. The power of this load depend on the position of the considered particle, so $P + jQ_1$ for the first terminal and $-P + jQ_2$ for the second terminal respectively. The resulting problem is non-linear and should be resolved using iterative techniques. From the most downstream nodes and taking the rated voltage at these points as a first approximation to their values, the forward sweep analysis is started. During this process, the current at each node is estimated as the following [1]:

$$I_i = \left(\frac{S_i}{v_i}\right)$$

After that, using Kirchhoff current law (KCL), the current in the feeder connecting the node to the one upstream I_{ij} is calculated, and the subsequent application of the Kirchhoff voltage law (KVL) leads to the voltage at this upstream node V_j. The forward sweep analysis is finished when the initial node is reached with an approximation to its voltage that in a general case differs for the given data. At this point a backward sweep should be done. Taking the real value at the initial node, the voltage only at the downstream node is recalculated. This process in repeated iteratively until, at the end of a forward sweep, the error of the estimation voltage at the initial node is below a certain threshold value [1].

A typical distribution feeder will consist of the primary main, with laterals tapped off the primary main and sub-laterals tapped off the laterals, etc.

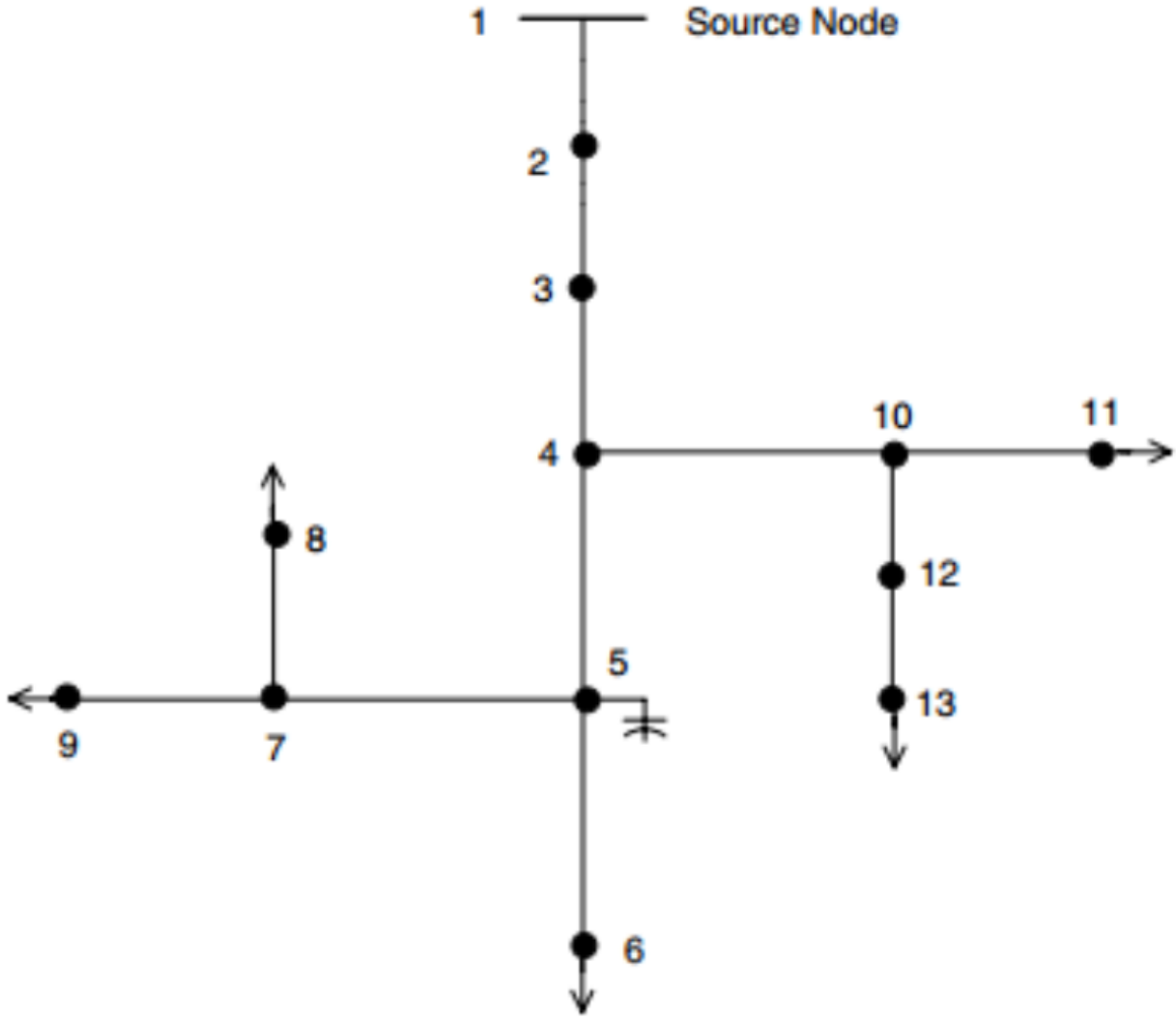

Figure 4: 13-node general network

The ladder iterative technique procedure for a general feeder [5]:

1. The forward sweep starts by assuming three-phase voltages at the end nodes. The usual assumption is to use the nominal or base voltages.
2. Starting from one of the terminal node, compute the node current (load current plus capacitor current if present).
3. With this current, apply Kirchhoff's voltage law (KVL) to calculate the node voltages at the node that are before the terminal node till a junction node connection is reached.
4. Since its impossible to calculate a junction node without all the data from the nodes that is connected too. Choosing another terminal node and starting the procedure again till a junction is reached. If all the nodes that are connected to the junction nodes are calculated than it will be possible to calculate the junction node.

5. Using the most recent value of the voltage at the junction node, the node current at the junction ode (if any) is computed.
6. Apply Kirchhoff's current law (KCL) to determine the current flowing from Node before the junction node toward the junction node. As well compute the voltage at current node.
7. All the above procedure are followed every time a junction node is reached.
8. The calculation will continue till the initial node is reached.
9. Calculate the voltage at the initial node.
10. Compare the calculated voltage at the initial to the specified source voltage.
11. If not within tolerance, use the specified source voltage and the forward sweep current flowing from the initial node to the next node, and compute the new voltage at the next node.
12. The backward sweep continues, using the new upstream voltage and line segment current from the forward sweep to compute the new downstream voltage.
13. The backward sweep is completed when new voltages at all end nodes have been completed.
14. This completes the first iteration.
15. Repeat the forward sweep, only now using the new end voltages rather than the assumed voltages as was done in the first iteration.
16. Continue the forward and backward sweeps until the calculated voltage at the source is within a specified tolerance of the source voltage.
17. At this point the voltages are known at all nodes, and the currents flowing in all line segments are known. An output report can be produced giving all desired results.

Example of ladder technique

Linear network

A modification of the ladder network theory of linear systems provides a robust iterative technique for power-flow analysis. A distribution feeder is nonlinear because most loads are assumed to be constant kW and kVAr. However, the approach taken for the linear system can be modified to take into account the nonlinear characteristics of the distribution feeder. Figure 10.1 shows a linear ladder network [5].

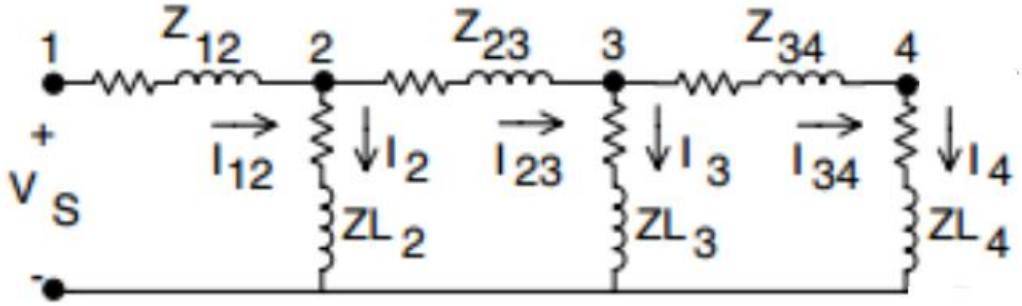

Figure 5: 4-node linear ladder network

For the ladder network it is assumed that all of the line impedances and load impedances are known along with the voltage at the source. The solution for this network is to assume a voltage at the most remote load. The load current I5 is then determined as [5]:

$$I_4 = \frac{V_4}{ZL_4}$$

For this end node case, the line current I_{34} is equal to the load current I_4, Applying Kirchhoff's voltage law (KVL), the voltage at Node 3 can be equal [5].:

$$V_3 = V_4 + Z_{34}.I_{34}$$

The load current I_3 can be determined, and then Kirchhoff's current law (KCL) applied to determine the line current I_{23}:

$$I_{23} = I_{34} + I_3$$

Kirchhoff's voltage law is applied to determine the node voltage V_2, this procedure is continued until a voltage V_1 has been computed at the source [5].

The computed voltage V_1 is compared to the specified voltage V_s, there will be a difference between these two voltages. The ratio of the specified voltage to the compute voltage can be determined as:

$$ratio = \frac{V_s}{V_1}$$

Since the network is linear, all of the line and load currents and node voltages in the network can be multiplied by the Ratio for the final solution to the network [5].

Non-Linear network

The linear network is modified to a nonlinear network by replacing all of the constant load impedances by constant complex power loads as shown in the below figure. The procedure outlined for the linear network is applied initially to the nonlinear network [5].

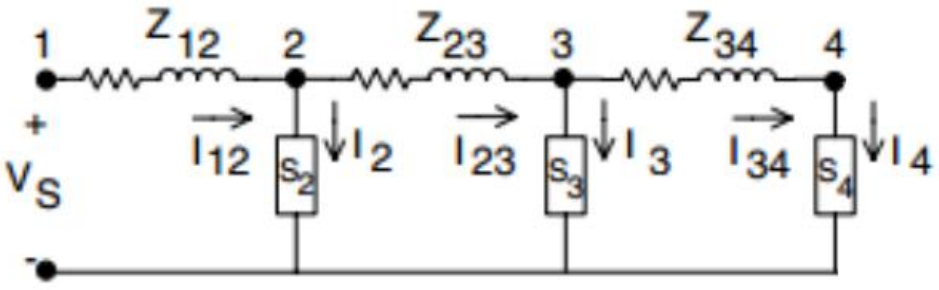

Figure 6: 4-node non-linear ladder network

The procedure outlined for the linear network is applied initially to the nonlinear network. The only difference is that the load current at each node is computed by [5]:

$$I_n = \frac{S_n}{V_n}$$

The forward sweep will determine a computed source voltage V_1 as in the linear case, this first iteration will produce a voltage that is not equal to the specified source voltage V_s because the network is nonlinear, multiplying currents and voltages by the ratio of the specified voltage to the computed voltage will not give the solution. The most direct modification to the ladder network theory is to perform a backward sweep. The backward sweep commences by using the specified source voltage and the line currents from the forward sweep. Kirchhoff's voltage law is used to compute the voltage at Node 2 by [5]:

$$V_2 = V_s - Z_{12}.I_{12}$$

This procedure is repeated for each line segment until a new voltage is determined at Node 4. Using the new voltage at Node 5, a second forward sweep is started that will lead to a new computed voltage at the source. The forward and backward sweep process is continued until the difference between the computed and specified voltage at the source is within a given tolerance [5].

Algorithm development

The Matlab program algorithm consists of two files, the first file "dist_grid_data.m" used is about filling all the data of the specified network in order to construct the feeder using only the m.file. This File should include the following characteristics of the feeder:

- Number of nodes of the network.
- Nodes number used in the LFP control.
- Sons of the different nodes.
- Parents of the different nodes (build automatically form the previous one).
- Load apparent power.
- Voltage base value for each node.
- Transformers characteristics if exist:
    - Rated power of the transformer between two specified nodes.
    - Taps value of the transformers.
    - Transformers impedances.
    - Transformer location between which 2 nodes.
    - Node reference base voltage used to change all the voltages to the same base voltage.
    - Transformation needed on each node to node to change the base voltage.
- Line characteristics:
    - Line length.
    - Line impedance in pu.
    - Automatic calculation of line impedance.

As well defining all the characteristics related to PSO and LPF control:

- Number of particles.
- Number of iteration.
- Dimension of the particle swarm optimization (number of references).
- The maximum and minimum value of the position and speed.
- The value of the inertia, cognitive and social coefficient (optimum value used).
- Tolerance value, epsilon.

The data of this file can be changed accordingly to suite any network that is required to study the LPF control for it. No changes will be needed on the second file which is the main program that contain all the algorithm as the program is universal for all type of constructed network.
The second file contain all the algorithm used during the process, it start by calculating the transformers impedances and transformation ratio. After that it will calculate to total impedance on each line including line and transformers impedance and choosing a specified base ratio. After finishing all these line and transformers calculation.
The LPF control algorithm will start by an initial allocation of the value used in the particle swarm optimization. The ladder method is followed, by a forward sweep and backward sweep in order to calculate network voltage and current and other required characteristics like line losses at each node. After the performance of the forward and backward sweep the particle swarm optimization algorithm is used to rectify the obtained values in order to reach the optimum values at the end. All this process is repeated for each particle and as well for the maximum number of iteration using the function 'for'.

In this part, the building of the code of the algorithm will be explained. Before beginning the forward sweep analysis, it has to start by considering the terminal node has a predefined value which is the theoretical base voltage value so it's possible to calculate the voltage on all terminal nodes. Followed by defining the initial node of the network.

Based on this consideration and knowing the apparent power on each node including the terminal, it will be possible to calculate the current that flows from the node before to the terminal, as was explained in the above part. The forward sweep analysis start by choosing one of the terminal node to start the calculation. Each node calculated is added to a table, in this case it's called 'treated' that allow to check later if the node has been calculated or not. The matlab function 'ismember' is used to check if an element exist between the sons of a node and the 'treated' table, this function give true (1) or false (0) if the element exist in these two tables. If all the elements are treated, it will be possible to treat the node, if not the program will look to another terminal node to repeat the procedure. The procedure is repeated through the 'while' function until all the nodes are added to the 'treated' table.

After finishing the forward sweep analysis, the base voltage of the initial node must be changed to node reference base voltage, using the transformation ratio of the transformers. After that, the voltage obtained at the initial node from the forward sweep calculation is compared to the base voltage of the initial node too with a difference no more than the tolerance value. Using the function 'if', if the two compared value are high than the tolerance the backward sweep analysis is executed.

In the backward sweep, the initial node will be defined with a voltage value equal to the base voltage. After that, also using a new table 'v_treated' which every calculated voltage of the node will be added to it and also using the function 'ismember'. In this part, the starting point will be the initial node and the calculation will be performed till a terminal node is reached using a 'while' function. For every nodes, when all the sons of it are treated, this node is added the new table named 'treated_bs, this table will directly include the initial node. Another 'while' function is also used with two condition, the first check if every element of the table 'v_treated' is included in the table 'treated_bs' and the other condition to check the length of the table treated_bs which should not pass the limits. Once an element of the table 'v_treated' is not an element of the table 'treated_bs' the procedure will be repeated from the beginning till all the element are included in the 'treated_bs' table.

After finishing the forward and backward sweep analysis, line losses calculation is performed including all the lines that exist on the network.
After that, this part is optional and is used to calculate the voltage at each node and at each particles for every iteration, and it's used to display the voltage changed to the reference base voltage and the real voltage.

This part compare the results obtained, if the result of the line losses is lower than the one from the previous iteration, the value is stored in a different variable alongside the ones of active and reactive power, this is performed for each particles. Also another variable is used to store the best result from all the particles for the line losses and the active and reactive power which will be the optimum values obtained at the end of the algorithm after all the iteration are done.
Finally, PSO equation is used to correct the values of active and reactive power before the performance of the next iteration followed by checking if the values obtained doesn't exceed the maximum and the minimum values.

Using the function 'display' the optimum values for losses and active and reactive power are displayed on the command screen at the end of the program.

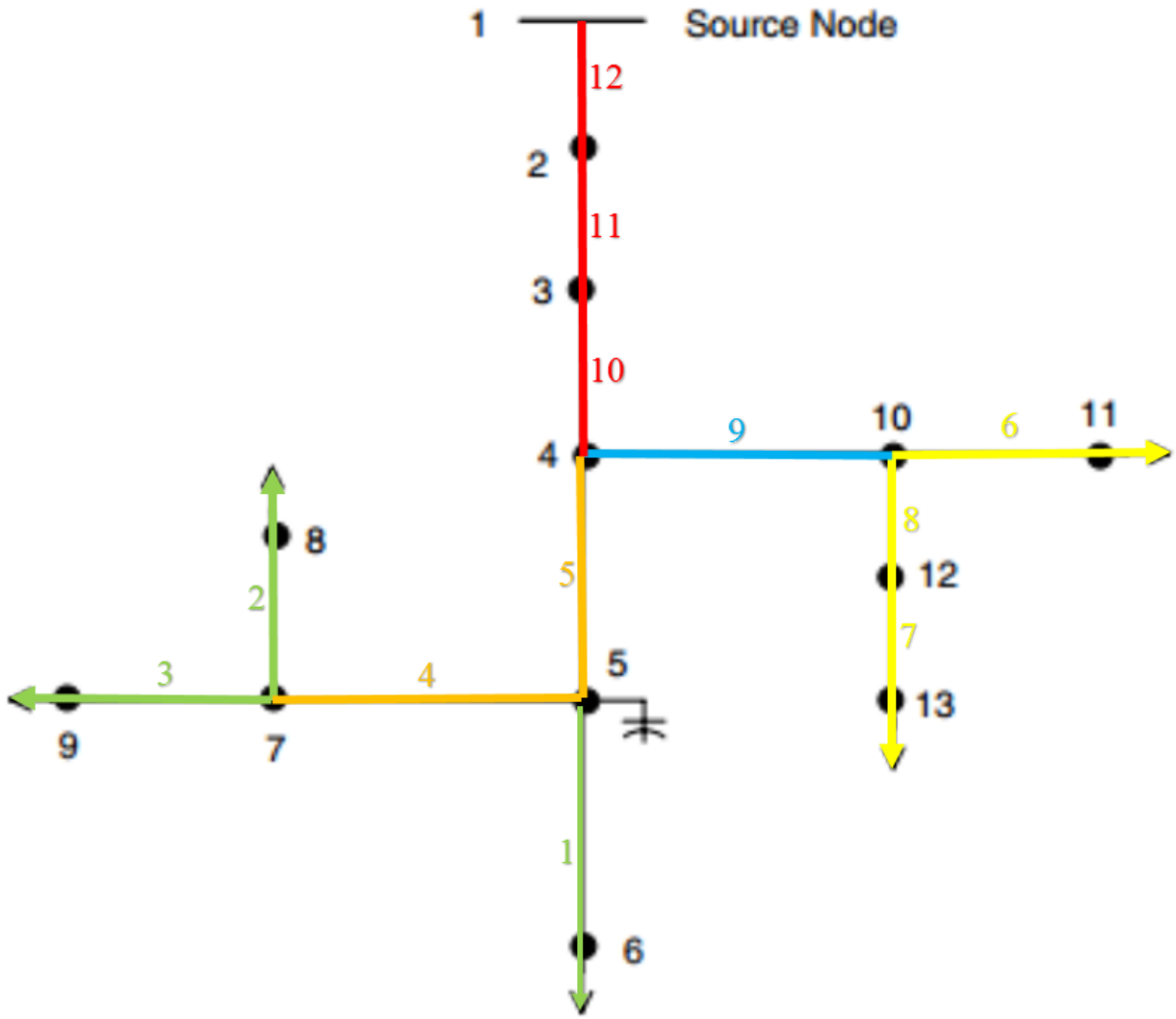

Figure 7: shows how the code work for a forward sweep on a generalized feeder

The figure above shows an example of how the algorithm works for a forward sweep analysis step by step on a generalized feeder to calculate the voltage, current, etc. on each node starting from a terminal node till it reach the initial node.

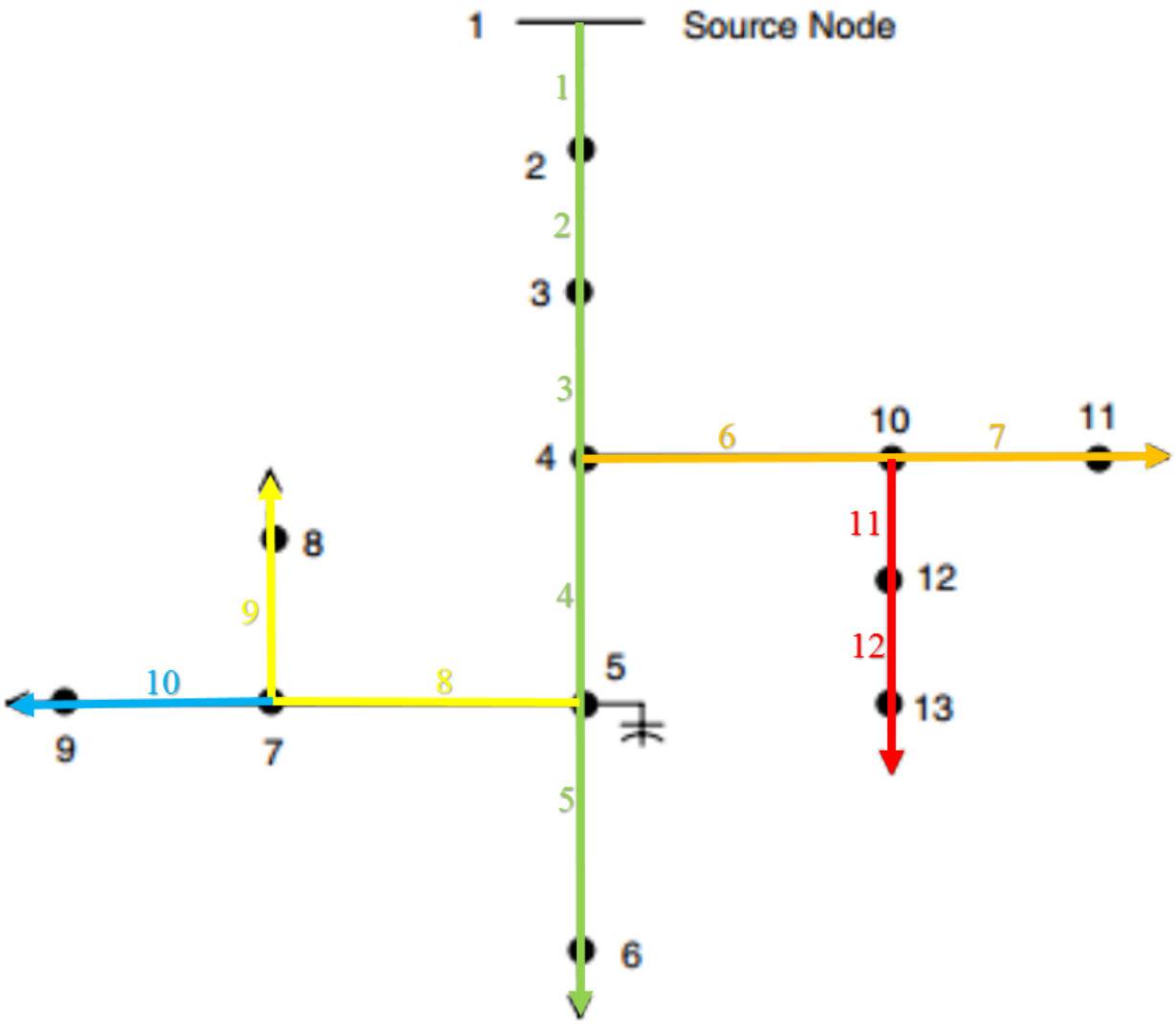

Figure 8: shows how the code work for a backward sweep on a generalized feeder

The figure above shows an example of how the algorithm works for a backward sweep analysis step by step on a generalized feeder in order to correct the voltage for an accurate value on each node starting from the initial node.

Simulation

The figure below shows the electrical diagram of a 6 node feeder system with its parameters that is used to test the proper performance of the EMS constructed algorithm using Matlab.

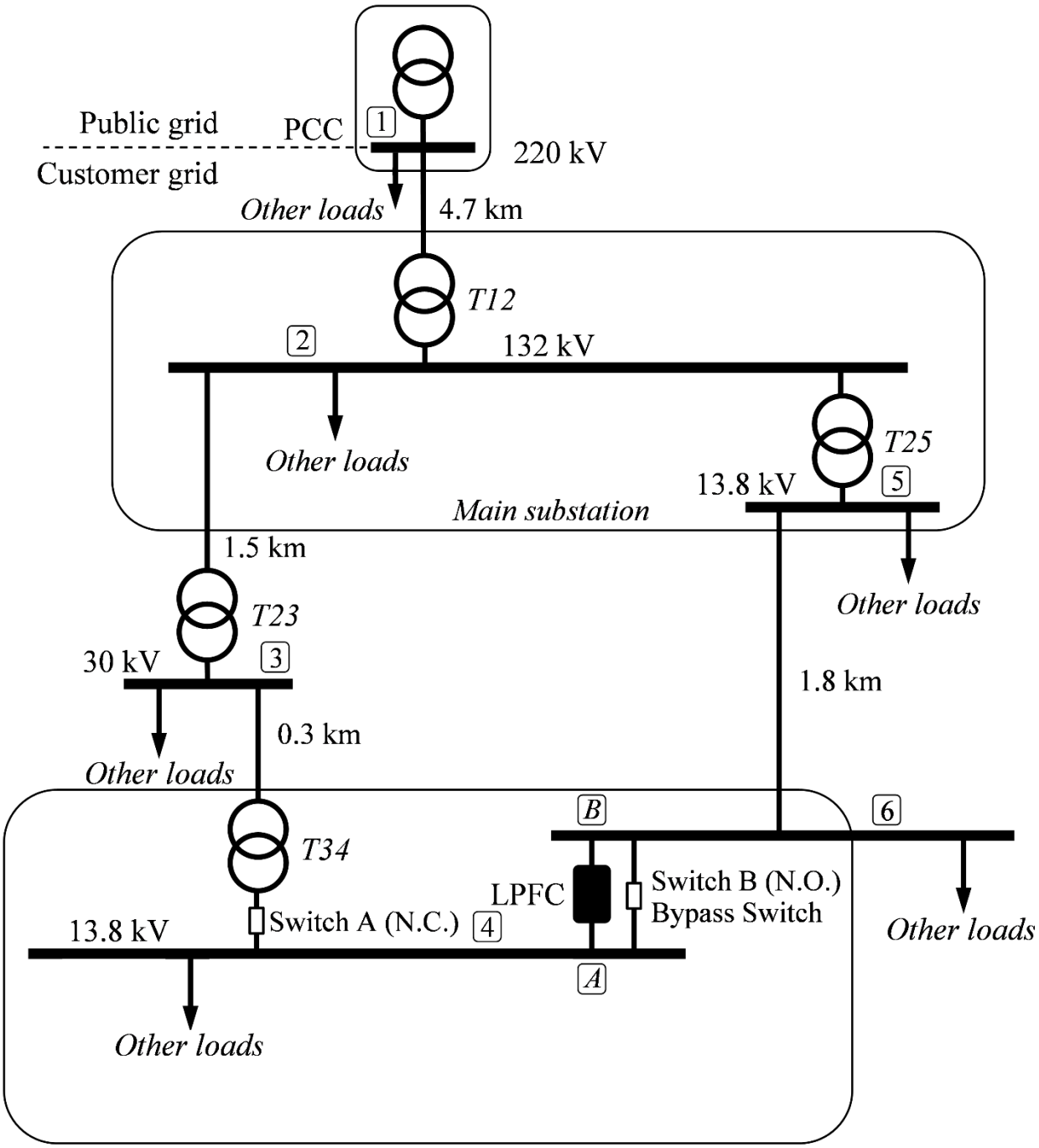

Figure 9: Simplified electrical system diagram of the 6-node grid

Network and LPF control parameters:

- Lines (z) and transformers (S_n, R_{sc}, X_{sc}, tap_0, tap_{LPC})
  - Line 12: $0.025 + 0.024i\ \Omega/km$
  - Line 23: $0.161 + 0.151i\ \Omega/km$
  - Line 23: $0.568 + 0.133i\ \Omega/km$
  - Line 25: $0.062 + 0.165i\ \Omega/km$
  - Line 56: $0.161 + 0.112i\ \Omega/km$
  - T12: $2x70$ MVA, $0.90\%, 12.9\%, 2\%, 2\%$
  - T23: $3x37.5MVA, 0.90\%, 9\%, 1\%, 1\%$
  - T34: $10MVA, 0.95\%, 4.8\%, 0\%, -1.5\%$
  - T25: $3x50MVA, 0.92\%, 48.5\%, 2\%, -1.5\%$
- Non-voltage dependent loads (P, Q)
  - Bus 1: $20\ MW, 25\ MVAr$
  - Bus 2: $84\ MW, 26\ MVAr$
  - Bus 3: $34\ MW, 12\ MVAr$
  - Bus 4: $7.5\ MW, 5\ MVAr$
  - Bus 5: $52\ MW, 39\ MVAr$
  - Bus 6: $1.3\ MW, 2\ MVAr$
- PSO algorithm and ladder iterative technique
  - Number if iterations: $k_{max} = 70$
  - Number of particles: $h_{max} = 20$
  - Inertia weight: $w = 0.729$
  - Cognitive weight: $c_1 = 1.49445$
  - Social weight: $c_2 = 1.49445$
  - Threshold error: $\epsilon = 4.10^8\ p.u$

After performing the simulation, with a maximum number of iteration of 70 and a number of particles of 20, the following reference results are obtained:

- Optimum losses: 1293.2 kW
- Reference active power: -2322.9 kW
- Reference reactive power 1: -7646.3 kW
- Reference reactive power 2: -4564.9 kVAr

Another simulation should be done without the use of LPC control to be able to compare the different values. In order to do so, the maximum number of iteration should have a value of 1, the same applies for the number of particles and the initial values of the active and the two reactive power is equal to 0, the following results are obtained:

- Optimum losses:1441 kW

From these two results, the power saving on the 6-node grid is the following,

$$Power\ Saving = 1441 - 1293.2 = 147.8\ kW$$

Chapter 3
Optimization of Number of particles and number of iteration

The optimization of the number of particles (N) and the number of iterations (itmax) lead to a reduction of the time needed to perform the activity in the algorithm. The best set of values is the one that will assure a specific guaranty of convergence characteristic with the shortest time of execution. In order to evaluate the execution time it is possible to change this parameter with the total number of execution of the function to be optimized (EX=N.itmax) because it's not reliable to measure the execution time on a computer. In order to find the optimum set, each pair to be tested cannot be evaluated just once, as with statistical basis. The random selection of the initial position, the initial speed of particles and the random values or r_1 and r_2 at each step can bring the same pair to meet specific convergence criteria or not.

The probability of finding convergence for a pair (N, itmax) in a set of n trials follows a binomial distribution.

In probability theory and statistics, the binomial distribution with parameters n and p is the discrete probability distribution of the number of successes in a sequence of n independent yes/no experiments, each of which yields success with probability p. A success/failure experiment is also called a Bernoulli experiment or Bernoulli trial; when n = 1, the binomial distribution is a Bernoulli distribution. The binomial distribution is the basis for the popular binomial test of statistical significance.

The binomial distribution is frequently used to model the number of successes in a sample of size n drawn with replacement from a population of size N. If the sampling is carried out without replacement, the draws are not independent and so the resulting distribution is a hypergeometric distribution, not a binomial one. However, for N much larger than n, the binomial distribution is a good approximation, and widely used.

The binomial distribution gives the discrete probability distribution $P_p(n|N)$ of obtaining exactly n successes out of N Bernoulli trials (where the result of each Bernoulli trial is true with probability p and false with probability q=1-p). The binomial distribution is therefore given by

$$P_p(n|N) = \binom{N}{n} p^n q^{N-n} = \frac{N!}{n!\,(N-n)!} p^n (1-p)^{N-n}$$

Where $\binom{N}{n}$ is a binomial coefficient.

The figure below shows the binomial distribution of a set of N=100 trials with a probability of convergence or success of p=0.7.

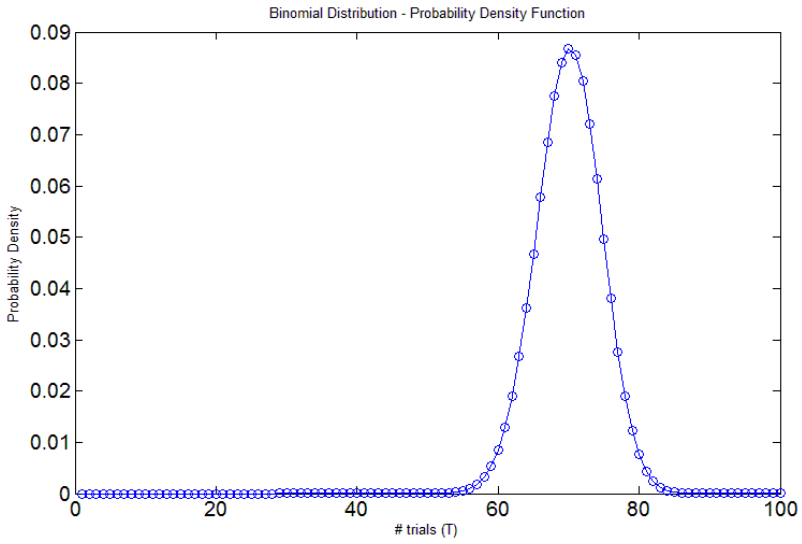

Figure 10: Binomial distribution

However, the probability of convergence of the binomial distribution is not known and it could only be determined for sure by performing an infinite number of trials. As this is not feasible, we have to specify a confidence interval (1-α) in which the media of the binomial distribution lies between two values (p1 and p2). In our case we can specify the value of p1, according to the rate of success expected.

Given a fix number of trials, the values for p1 and p2 can be calculated by

$$p_1 = \frac{X}{(n - X + 1)F_{\alpha|2,2(n-X+1),2X} + X}$$

$$p_2 = \frac{(X + 1)F_{\alpha|2,2(X+1),2(n-X)}}{(n - X) + (X + 1)F_{\alpha|2,2(X+1),2(n-X)}}$$

Where X is the number of successes and $F_{\alpha|2,a,b}$ is the Fisher-Snedecor distribution with two degrees of freedom, *a*, *b*, that leaves at the right hand a probability of α/2 in a confidence interval of (1-α)*100%.

In probability theory and statistics, the Fisher-Snedecor (F-distribuition) distribution is a continuous probability distribution. The F-distribution provides a basis for comparing the ratios of subsets of these variances associated with different factors. Many experimental scientists make use of the technique called analysis of variance. This method identifies the relative effects of the "main" variables and interactions between these variables. The F distribution represents the ratios of the variances due to these various sources [7]. The main use of F-distribution is to test whether two independent samples have been drawn for the normal populations with the same variance, or if two independent estimates of the population variance are homogeneous or not, since it is often desirable to compare two variances rather than two averages. For instance, college administrators would prefer two college professors grading exams to have the same variation in their grading [8].

Given a minimum value of the mean of the binomial distribution of p1>0.9 and α=0.01, we will assure a minimum mean convergence ratio of 90% with a confidence interval of 99%. By

inspection of the above equation, these figures can be assured with different sets of n, X, as seen in the table below:

n	X	P1	P2	
51	51	0.9013	1	1st attempt
71	70	0.9000	0.9999	2nd attempt
89	87	0.9000	0.9988	Too time-consuming

Table 1: Results obtained using binomial distribution

In consequence, to test the validity of a give pair (*N*,*IT*), 51 trials will be run. If 51 successes are obtained the pair is considered VALID. If more than two failures are obtained the pair is disregarded, as it does not meet the convergence criteria or proving it will need too much time. If just one failure is obtained, more trials are added until 71. If 70 successes are obtained, then the pair is also considered VALID. If new failures appear in these additional trials, the pair is discarded.

Once we have a solid criterion to decide if a specific pair is considered VALID, now map of the N, IT plain has to be explored in order to find the better solution. For each value of N, a minimum value of IT can be found meeting the converge criterion (i.e. N,It being VALID). Any value of IT higher than this will be also valid but more time-consuming. The application of this rule leads to the blue line in Fig. 6.

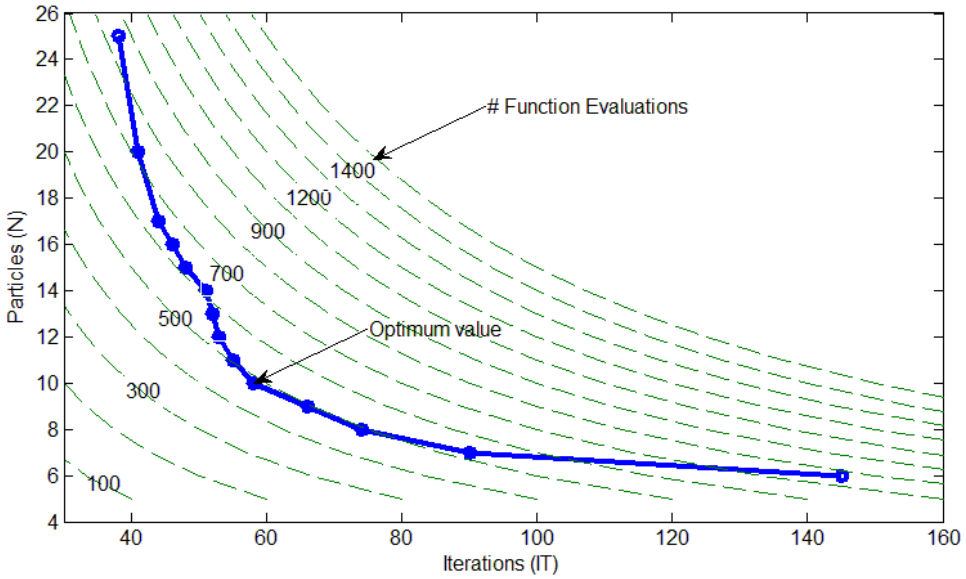

Figure 11: Set of curves for different values of N and it

The above figure includes a set of curves that states combinations of particles and iterations with the same computational effort (i.e. EX= N · IT is constant).

The pair contained in the blue line, with the lower value of EX is the optimum solution to the problem. For this example, it corresponds to the pair N=10, IT=58, that requires a computation effort of EF=580 evaluations of the function to be optimize.

From figure 6, we can see that using more than 20 particles does not add any benefit to the convergence of the algorithm. On the other side very low values of the number of particles (under 7) makes convergence very difficult demanding a huge amount of iterations.

Chapter 4
Coefficient optimization of inertia, social and cognitive weight

To optimize the three coefficient inertia weight, social weight and cognitive weight, the particle swarm optimization is used to obtain the best values of these parameters with the lowest time of execution. In order to do so, a large fix number of particles is used for example N=15, which should be enough to obtain convergence. For the number of iteration, no fix number shall be used, instead of that it will varies till the convergence is reached to its required values with a high precision of 99% for example.

Optimizing the number of iteration lead to an optimization of the time needed to perform the required task which is the calculation of the active and reactive power as the number of particles is a fixed number.

The algorithm will contain two iteration loop, or two particle swarm optimization. Both PSO have the same principle with little bit of change to be able to be used for the optimization of the three coefficient.

The first will be used to optimize the losses on the network and the second is used to optimize the three coefficient. By adding another PSO to the algorithm, the execution time of the algorithm will increase due to the fact that for every iteration and particle obtained for the coefficient, the PSO control of the network will be executed, so the time needed for the algorithm will be more than ten time higher than before.

Algorithm development

The algorithm used here is similar to the one developed in chapter 2, the main difference is the use of two particle swarm optimization as mentioned above. For the iteration related to calculating the active and reactive power reference, the function ‘while’ is used instead of ‘for’ in order to find the minimum iteration needed to get to convergence. The objective of this chapter is to optimize to coefficient used for the optimization of the active and reactive power reference, the values for the coefficient used for the first PSO that is used to optimize the coefficient are the following

The parameters for the first PSO related to optimize the coefficient of the PSO related to the optimization of the active and reactive power reference are as following:

- Number of particles.
- Number of iteration.
- Dimension of the particle swarm optimization (number of references).
- The maximum and minimum value of the position and speed for the inertia weight.

- The maximum and minimum value of the position and speed for the social and cognitive weight.
- The value of the inertia, cognitive and social coefficient (optimum value used).

The parameters for the second PSO related to the optimization of the active and reactive power reference are the same of chapter 2, but excluding the number of iteration because in this case it varies and the inertia, cognitive and social coefficient because they will be defined from the previous PSO. The simulation is performed five times for a better precision and the results for the number of iteration and precision is the average of the five results obtained for each iteration and number of particles.
Also, another part is added related for checking the accuracy of the results obtained in this chapter with the optimum one from chapter 2, so if the values are accurate and with a lower execution time or lower number of iteration, the coefficient values are accepted.
Finally the obtained results will be the number of iteration and the three optimum values of the coefficient inertia weight, social weight and cognitive weight.

Simulation

The simulation of the program responsible for the optimization of the inertia weight, social weight and cognitive weight coefficients also with the objective of reducing the number of iteration which lead to a reduction of time needed to perform the operation. One problem occur during the simulation which is the validation of the results, the new values can lead to a loss in accuracy of the results so the active and reactive power reference value diverge from the optimal one which were calculated in chapter 2. In order to get correct results, for every iteration and every particles an accuracy test with 1% of error should be done to verify if the obtained results match the correct one, if not the obtained value should be dropped. The final answer will be the one who have the lowest number of iteration and thus the lowest time and of course with an error of less than 1%.
The following equation calculate the error or the accuracy between the optimum value calculated in chapter 2 and the values obtained for each set of coefficient:

$$Acc_P = \frac{\left|P_{opt} - P\_calc\right|}{P_{opt}}$$

$$Acc_{Q1} = \frac{\left|Q_{1,opt} - Q_{1,calc}\right|}{Q_{1,opt}}$$

$$Acc_{Q2} = \frac{\left|Q_{2,opt} - Q_{2,calc}\right|}{Q_{2,opt}}$$

$$Acc_{losses} = \frac{\left|losses_{opt} - losses_{calc}\right|}{losses_{opt}}$$

Than the above results should have an error less than 1% to accept the values of the coefficient and the less execution time or less iteration needed the better. So the final result will have the

fastest execution time or the minimum iteration needed between all results obtained with an error or accuracy less than 1%.

$$objctiv_{function} = it_{convergence} + precision$$

Where is the number of iteration needed to reach the convergence and the precision has the following equation:

$$precision = 100 * max(Acc_P, Acc_{Q1}, Acc_{Q2})$$

For this case, after performing the simulation, these results are obtained for inertia weight, social weight and cognitive weight and can be adopted:

- Optimum number of iteration: : $itmax = 18$
- inertia weight: $w = 0.62629$
- social weight: $c_1 = 1.4437$
- cognitive weight: $c_2 = 1.4112$

The results obtained are close to the one used in chapter 2 due to the fact that the one used are the optimum values so in this chapter the results that should be obtained should be close to them which was the case.

Chapter 5

Evaluation of the economical savings

In electrical engineering, a load profile is a graph of the variation in the electrical load versus time. A load profile will vary according to customer type (residential, commercial and industrial), temperature and holiday seasons. Power producers use this information to plan how much electricity they will need to make available at any given time. In a power system, a load curve or load profile is a chart illustrating the variation in demand/electrical load over a specific time.

For this case, considering a daily load profile with an interval of two hours between each load value taken, it will have the following standard shape

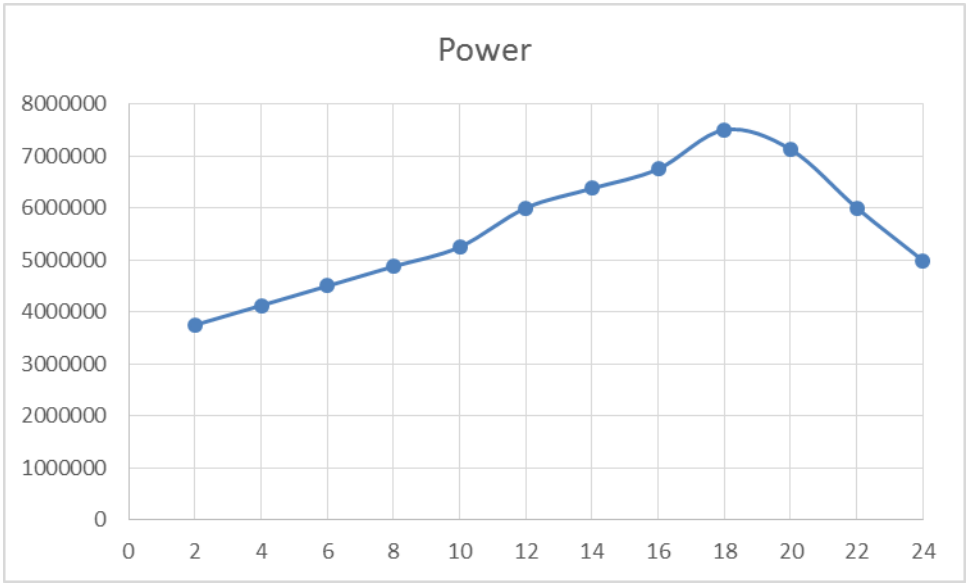

Figure 12: Daily load profile curve used

In order to evaluate the expected savings on a grid when applying the loop power flow control. For example, A daily load profile can be built using the excel file, it's what is used in the report. The input table should include at first column the time followed by the active and reactive power at each node. The LPC will calculate to optimum losses at each time interval than will generate the final optimum energy and power losses.

The following chart represent the EMS algorithm which is the ladder technique that is used the build the Matlab code, it's more or less the same as the previous one but with some adjustment to meet the requirements needed for this chapter:

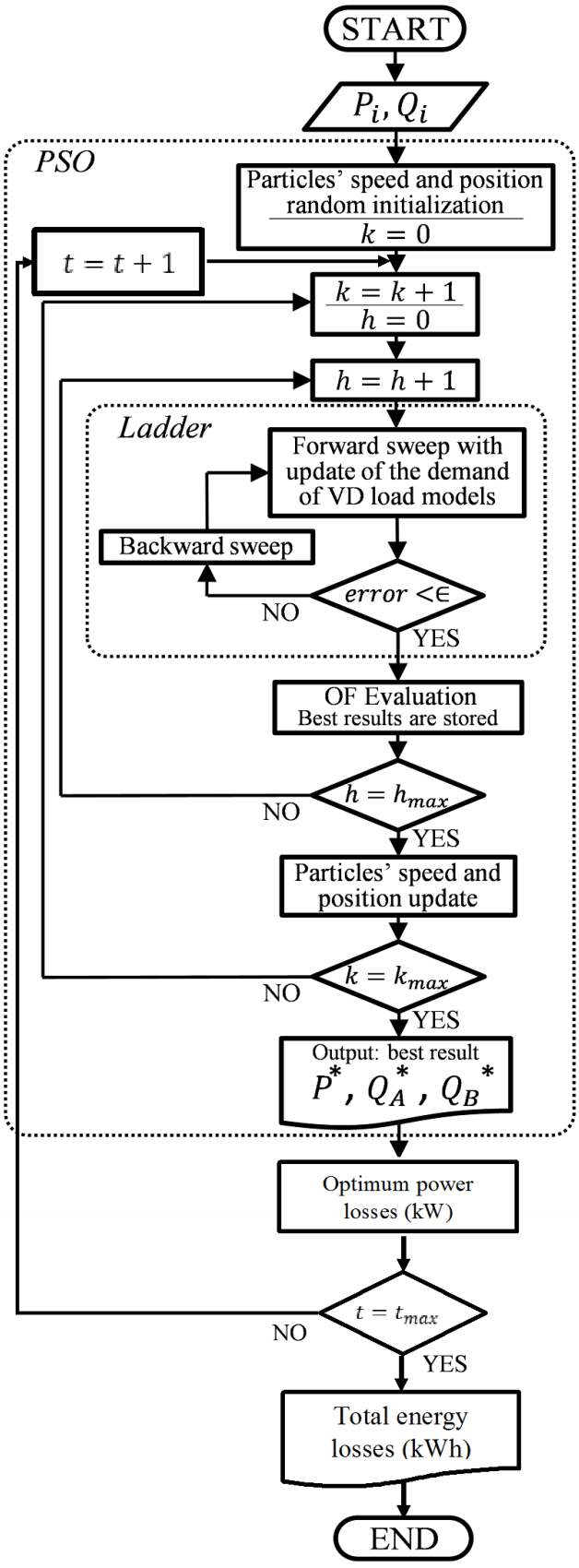

Figure 13: Flowchart of the EMS algorithm for energy saving

Maximum Size of the inverters

After calculating the optimum active and reactive power needed to optimize the feeder from the daily load profile. The maximum active and reactive optimum value can be also taken in order to be able to calculate the apparent power through the following equation

$$S_{1,max} = \sqrt{{P_{max}}^2 + {Q_{1,max}}^2}$$

$$S_{2,max} = \sqrt{{P_{max}}^2 + {Q_{2,max}}^2}$$

Considering the financial aspect related to the cost of the inverter where the cost to be paid must not exceed a certain amount. So a limitation can be added to the maximum rated power S_k allowed as in general, increasing the rated power of the inverter lead to an increase of its cost:

$$S_{1,max} \leq S_k$$
$$S_{2,max} \leq S_k$$

Algorithm development

The algorithm used in this chapter in more or less the same as the one in chapter two, the only differences is that the algorithm has to import the data from the excel file and has to be repeated for inch interval of time the values were taken and then calculate what is required, for example energy losses and average power losses each day. The algorithm will display as well the power losses and the active and reactive power reference for each interval of time.

Simulation

Using the following daily load profile for the 6-node grid where the load varies from a maximum values to a minimum value equal half of the maximum as was seen in the above graph. The only difference in the algorithm between this simulation and the one done in chapter is that in this case the PSO algorithm is executed more than one time, it depend on the number of intervals that exist in the daily load, in this case the PSO is executed 12 times. Including to that, the calculation of energy losses is also added for every execution time so the total energy losses will be the sum of each part. The average power losses in a day with be the total energy losses over the total number of intervals.

$$total\ energy\ losses = \sum_{interval} P * \Delta t$$

$$P_{avg} = \frac{total\ energy\ losses}{total\ number\ of\ intervals}$$

The table below shows the daily load for 24h with an interval time between each value recorded of 2 hours. All the nodes in this case have the load curve variation only for simplification purpose.

The daily load data are recorded in an excel file where Matlab import the data values in order to be able to calculate the optimum reference value for active and reactive power.

time	P_1	Q_1	P_2	Q_2	P_3	Q_3	P_4	Q_4	P_5	Q_5	P_6	Q_6
2	10000000	12500000	42000000	13000000	17000000	6000000	3750000	2500000	26000000	19500000	650000	1000000
4	11000000	13750000	46200000	14300000	18700000	6600000	4125000	2750000	28600000	21450000	715000	1100000
6	12000000	15000000	50400000	15600000	20400000	7200000	4500000	3000000	31200000	23400000	780000	1200000
8	13000000	16250000	54600000	16900000	22100000	7800000	4875000	3250000	33800000	25350000	845000	1300000
10	14000000	17500000	58800000	18200000	23800000	8400000	5250000	3500000	36400000	27300000	910000	1400000
12	16000000	20000000	67200000	20800000	27200000	9600000	6000000	4000000	41600000	31200000	1040000	1600000
14	17000000	21250000	71400000	22100000	28900000	10200000	6375000	4250000	44200000	33150000	1105000	1700000
16	18000000	22500000	75600000	23400000	30600000	10800000	6750000	4500000	46800000	35100000	1170000	1800000
18	20000000	25000000	84000000	26000000	34000000	12000000	7500000	5000000	52000000	39000000	1300000	2000000
20	19000000	23750000	79800000	24700000	32300000	11400000	7125000	4750000	49400000	37050000	1235000	1900000
22	16000000	20000000	67200000	20800000	27200000	9600000	6000000	4000000	41600000	31200000	1040000	1600000
24	13300000	16625000	55860000	17290000	22610000	7980000	4987500	3325000	34580000	25935000	864500	1330000

Table 2: daily load profile for the 6-node network

Two types of simulation are performed, the first one include the LPF control and the following results are obtained in the table below:

Time (h)	Optimum losses (kW)	P_ref (kW)	Q1_ref (kVAr)	Q2_ref (kVAr)	Optimum energy losses (kWh)	Max S1 (kVA)	Max S2 (kVA)
2	383.6	-2373.9	-6299.0	-3222.8	-	-	-
4	447.1	-2370.7	-6423.4	-3352.4	-	-	-
6	516.5	-2366.6	-6550.0	-3478.2	-	-	-
8	591.9	-2361.4	-6678.5	-3608.9	-	-	-
10	673.3	-2356.8	-6809.9	-3739.5	-	-	-
12	854.6	-2346.2	-7079.5	-4008.8	-	-	-
14	954.6	-2342.3	-7220.1	-4145.6	-	-	-
16	1061.0	-2339.8	-7363.6	-4286.7	-	-	-
18	1293.2	-2323.0	-7646.0	4565.8	-	-	-
20	1173.8	-2329.3	-7501.1	-4423.1	-	-	-
22	854.6	2346.4	7079.1	-4008.0	-	-	-
24	615.7	-2360.0	-6719.0	-3647.3	-	-	-
Average	785.0				18839.8	7991.2	5122.8

Table 3: Results obtained using LPF control

The second simulation is done without the use of the LPF control, so same procedure is followed as in task one, maximum number of iteration set to 1, number of particles also, and the initial values of active and reactive power are set to 0. The following results are obtained:

Time (h)	Optimum losses (kW)	Optimum energy losses (kWh)
2	478.9	-
4	546.7	-
6	620.5	-
8	700.5	-
10	786.8	-
12	978.4	-
14	1084.0	-
16	1196.2	-
18	1441.0	-
20	1315.2	-
22	978.4	-
24	725.7	-
Average	904.4	21705.3

Table 4: Result obtained without LPF control

The average power saving with and without LPF control is:

$$Average\ power\ saving = 904.4 - 785.0 = 119.4\ kW$$

The energy saving per day is:

$$Energy\ saving\ per\ day = 21705.3 - 18839.8 = 2865.5\ kWh$$

For the above two table results, it can be concluded that the LPF control helps by reducing losses and thus saving energy. In this case, around 2865 kWh are saved each day. It can be concluded that the LPF control method is efficient and help reduce the losses that occur on the network.
The results obtained with LPF control give also the maximum two values of apparent power that needed for the active and reactive reference power, these 2 values will be used to determine the rated power needed for the inverter that is used so it will not be undersized or even oversized.

Conclusion

It’s concluded that the energy management system using loop power flow control achieved a reduction in the losses on the network grid. As well, the algorithm is suitable for any random feeder to be tested, it’s only need to adjust the input data in order to execute the program without forgetting the optimization tests that can be performed. It can be added that the algorithm used can also be optimized to have a fast response which mean lower time to process and to give the optimum active and reactive power reference values. This will make the algorithm suitable for a real-time applications. Furthermore, the use of particle swarm optimization shows a good potential and it’s possible to be implemented in parallel computing devices if an even faster execution time is required.

Future developments

The use of this method requires a lot of materials which have a high cost and the cost for example will also depend on the maximum reference apparent power of the inverter. So another financial study should be made to find out if the method has a financial advantage between installation cost and reduction of the losses on the network, as well for example calculating the payback year of this system. In general, a financial study should be also done to check if applying this method is worth it or not for a specified network.

In the technical part, the algorithm could be developed to locate the best location to install the inverter in order to increase the efficiency at its maximum value. In order to do so, this algorithm should be able to test all node connections possibilities and at the end to find the optimum location in technical and financial part.

Reference

1. Application of loop power flow controllers for power demand optimization at industrial customer sites, José m. Cano, Joaquin g. Norniella, Carlos h. Rojas and Gonzalo a. Orcajo, electrical engineering department university of Oviedo Gijón, spain
2. Loop power flow control and voltage characteristics of distribution system for distributed generation including PV system, Naotaka Okada, Hiromu Kobayashi, Kiyoshi Takigawa, Masahide Ichikawa and Kosuke Kurokawa, central research institute of electric power industry, Tokyo university of agriculture and technology, Tokyo, japan, 2003.
3. Autonomous loop power flow control for distribution system, N. Okada, central research institute of electric power industry, japan, 2001.
4. http://www.swarmintelligence.org/
5. Distribution system modeling and analysis, William h. Kersting, CRC press.
6. http://mathworld.wolfram.com/binomialdistribution.html
7. http://onlinelibrary.wiley.com/doi/10.1002/9780470627242.ch20/summary
8. https://explorable.com/f-distribution
9. Loop power flow control to minimize power losses and augment voltage stability, y. Mitani, g. Matushiro, k. Tsuji, department of electrical engineering, faculty of engineering, graduate school of Osaka university, 1998.
10. http://www.scholarpedia.org/article/Particle_swarm_optimization#Applications_of_PSO_and_Current_Trends
11. Particle swarm optimization: basic concepts, variants and applications in power systems, Yamille del Valle, Salman Mohagheghi, Ronald g. Harley, IEEE transactions on evolutionary computation, vol. 12, no. 2, april 2008.
12. Survey on K-mean Clustering and Particle Swarm Optimization, Pritesh Vora, Bhavesh Oza, International Journal of Science and Modern Engineering (IJISME) ISSN: 2319-6386, Volume-1, Issue-3, February 2013

Annex

6-node data m.file (chapter 2-5)

```
clear all
% Node list
N=6;
Node=6;

%% Choose the 2 nodes for the LFP control
node1=4;
node2=6;

%% PSO and Ladder caracteristics
N_data=20; % particle number
%N=1;
itmax_data=70;
%itmax=1; % no. de iteraciones
Dim_data=3;
minX_data=-10e6;
maxX_data=10e6;
minV_data=-maxX_data;
maxV_data=maxX_data;

w_data = 0.729; % inertia weight
c1_data = 1.49445; % cognitive weight
c2_data = 1.49445; % social weight

epsilon_data=0.0001; % Tolerance for convergence (in per unit)
%epsilon_data=0.000000000001; % Tolerance for convergence (in per unit)

%% Sons of the different nodes. [0] means it is a terminal node.
s{1}=[2];
s{2}=[3,5];
s{3}=[4];
s{4}=[0];
s{5}=[6];
s{6}=[0];

%% Parents (this should be built automatically)
for n=1:Node
p{n}=[0];
end
for n=1:Node
    len=length(s{n});
    for no=1:len
        if s{n}(no)~=0
        p{s{n}(no)}=[n];
        end
    end
end

%% Loads
```

```
S(1)=(20+25i)*1e6;
S(2)=(84+26i)*1e6;
S(3)=(34+12i)*1e6;
S(4)=(7.5+5i)*1e6;
S(5)=(52+39i)*1e6;
S(6)=(1.3+2i)*1e6;

%% Base Values
Ubase=13.8e3; % Tomado como general del sistema
Ub(1)=220e3;    % Nodo 1
Ub(2)=132e3;    % Nodo 2
Ub(3)=30e3;     % Nodo 3
Ub(4)=Ubase;    % Nodo 4
Ub(5)=Ubase;    % Nodo 5
Ub(6)=Ubase;    % Nodo 6

%% Transformers
% Datos de los transformadores (potencia nominal, número y ecc)
Sn(1:N,1:N)=zeros;
Sn(1,2)=270e6;
Nt(1,2)=2;
Sn(2,3)=37.5e6;
Nt(2,3)=3;
Sn(2,5)=50e6;
Nt(2,5)=3;
Sn(3,4)=10e6;
Nt(3,4)=1;
% Taps de los trafos
tap(1,2)=0.01;
tap(2,3)=0;
tap(3,4)=-0.01;
tap(2,5)=-0.035;
% Impedancias de los trafos lado aguas arriba
eRcc(1,2)=0.0090;
eXcc(1,2)=0.1297;
eRcc(2,3)=0.0090;
eXcc(2,3)=0.0895;
eRcc(2,5)=0.0092;
eXcc(2,5)=0.0795;
eRcc(3,4)=0.0095;
eXcc(3,4)=0.0476;

% Reference base node
tr{1}=[1,2];
tr{2}=[2,3];
tr{3}=[2,5];
tr{4}=[3,4];

nref=4; % node 4
% transformers from the node to the node reference
% trasnformer number and +(to primary of transformer 1/rt) - (to secondary
% rt) and 0 if no ratio transformation needed
d{1,2}=[1,2,4];
d{2,3}=[2,4];
d{2,5}=[2 4];
d{3,4}=[4];
```

```
d{5,6}=[2,-3,4];

nod{1}=[1,2,4];
nod{2}=[2,4];
nod{3}=[4];
nod{4}=[0];
nod{5}=[2,-3,4];
nod{6}=[2,-3,4];

%% Lines
% (lengths)
l_(1,2)=4700e-3;
l_(2,3)=1500e-3;
l_(2,5)=0e-3;
l_(3,4)=300e-3;
l_(5,6)=1800e-3;

% (pu impedance)
zc_(1,2)=0.025+0.240i;
zc_(2,3)=0.161+0.151i;
zc_(2,5)=0.062+0.165i;
zc_(3,4)=0.568+0.133i;
zc_(5,6)=0.161+0.112i;
% Lines (impedances)
zl(1,2)=(l_(1,2)*zc_(1,2));
zl(2,3)=(l_(2,3)*zc_(2,3));
zl(2,5)=(l_(2,5)*zc_(2,5));
zl(3,4)=(l_(3,4)*zc_(3,4));
zl(5,6)=(l_(5,6)*zc_(5,6));
```

Chapter 2 m.file

```
clear all
dist_grid_data_6node;

%% RECEPCIÓN DE DATOS
Sp(node1)=S(node1);
Sp(node2)=S(node2);

%% Transformer caracteristic
% Transformer relation and impedences
rt(1:Node,1:Node)=zeros;
zt(1:Node,1:Node)=zeros;
for n=1:1:Node
 len=length(s{n});
 for no=1:1:len
    ns=s{n}(no);
  if  s{n}~=(0)
    if Sn(n,ns)~=0
    rt(n,ns)=Ub(n)/(Ub(ns)*(1+tap(n,ns)));
    zt(n,ns)=Ub(n)^2/Sn(n,ns)*(eRcc(n,ns)+eXcc(n,ns)*1i)*1/Nt(n,ns);
    end
  end
 end
```

```
end

% Total impedances
% Reference base voltage Ubase 13.8 KV
z(1:Node,1:Node)=zeros;
for n=1:Node
  len=length(s{n});
  for no=1:len
      ns=s{n}(no);
      if  s{n}~=(0)
        z(n,ns)=zl(n,ns)+zt(n,ns);
      end
  end
end

for n=1:Node
  len=length(s{n});
  for no=1:len
      ns=s{n}(no);
       if  s{n}~=(0)
           dlen=length(d{n,ns});
           for u=1:dlen
           if d{n,ns}(u)<0
               tran=abs(d{n,ns}(u));
               ratio=rt(tr{tran}(1),tr{tran}(2));
               z(n,ns)=z(n,ns)*ratio^2;
           elseif d{n,ns}(u)>0
               tran=abs(d{n,ns}(u));
               ratio=rt(tr{tran}(1),tr{tran}(2));
               z(n,ns)=z(n,ns)*(1/ratio)^2;
           else
               z(n,ns)=z(n,ns);
           end

           end

       end
  end
end

%% SE CALCULA LA TENSIÓN MEDIANTE LADDER

% Ladder iterative method
epsilon=epsilon_data; % Tolerance for convergence (in per unit)

%--------------------------------------------------------------------------
%% PARTICLE SWARM OPTIMIZATION
N=N_data; % particle number
itmax=itmax_data; % no. de iteraciones
Dim=Dim_data;
minX=minX_data;
maxX=maxX_data;
minV=minV_data;
maxV=maxV_data;
```

```
w = w_data; % inertia weight
c1 = c1_data; % cognitive weight
c2 = c2_data; % social weight

% Allocation
Xdat(1:N)={zeros(1,3)};
Vdat(1:N)={zeros(1,3)};
Swarm=struct('X',Xdat,'V',Vdat,'Best_Fitness',inf);
% End allocation

%Swarm(1).X=[0,0,0];
for h=2:N

Swarm(h).X=[random('Uniform',minX,maxX),random('Uniform',minX,maxX),random('U
niform',minX,maxX)];
end
for h=1:N

Swarm(h).V=[random('Uniform',minV,maxV),random('Uniform',minV,maxV),random('U
niform',minV,maxV)];
end
for h=1:N
    Swarm(h).Best_Fitness=inf; % Inicialización del best_fitness a un valor
infinito
end
Swarm_Global.Best_Fitness=inf; % Inicialización del Global_best_fitness a un
valor infinito
for it=1:itmax
    %it
    % Allocation
    Current(1:N)=zeros(1,N);
    Sconv(1:N)=zeros(1,N);
    voltage_nodes(1:N,1:Node)=zeros(N,Node)+1i*zeros(N,Node);
    voltage_nodes_real(1:N,1:Node)=zeros(N,Node)+1i*zeros(N,Node);
    % End allocation
for h=1:N
    %h
    P=Swarm(h).X(1);
    Q1=Swarm(h).X(2);
    Q2=Swarm(h).X(3);
    SL(node1)=P+Q1*1i;
    SL(node2)=-P+Q2*1i;

%% Ladder iterative method
% Variables preallocation
v(1:Node)=zeros;
I(1:Node,1:Node)=zeros;
v_treated(1:Node+1)=zeros;
% Initial start for voltage at terminal nodes
for nu=1:Node
    if s{nu}==[0]
        v(nu)=Ubase/sqrt(3);
    end
end
% Looking for initial node
nu=1;
```

```
while p{nu}~=[0]
    nu=nu+1;
end
init_node=nu;

iter=0;
stop=0;
while stop==0
iter=iter+1;

S(node1)=Sp(node1)+SL(node1);
S(node2)=Sp(node2)+SL(node2);

% Forward Sweep
    clear treated
    treated=0;
% Looking for a terminal node to start
nu=1;
while s{nu}~=[0]
    nu=nu+1;
end
ns=nu;
I(ns,ns)=(S(ns)/(3*v(ns)))';
np=p{ns};
I(np,ns)=I(ns,ns);
v(np)=v(ns)+z(np,ns)*I(np,ns);
treated(length(treated)+1)=ns;
ns=np;
while length(treated)<Node+1
    if sum(ismember(s{ns},treated))==length(s{ns}) % if the sons of this node
are treated we treat the node
        I(ns,ns)=(S(ns)/(3*v(ns)))';
        np=p{ns};
        if np==[0] % For the case we reach the initial node
            I(ns)=sum(I(ns,:));
        else % We does not still reach the initial node
            I(np,ns)=sum(I(ns,:));
            v(np)=v(ns)+z(np,ns)*I(np,ns);
        end
        treated(length(treated)+1)=ns;
        ns=np;
    else % if any of the sons of the node are not treated we look for a
terminal node
        while s{ns}~=[0]
            b=1;
            while ismember(s{ns}(b),treated)==1
                b=b+1;
            end
            ns=s{ns}(b);
        end
    end
end

%
 dlen=length(nod{init_node});
 vo=abs(Ub(init_node)/sqrt(3));
```

```
 for u=1:dlen
            if nod{init_node}(u)<0
               tran=abs(nod{init_node}(u));
               ratio=rt(tr{tran}(1),tr{tran}(2));
               vo=vo*ratio;
            elseif nod{init_node}(u)>0
               tran=abs(nod{init_node}(u));
               ratio=rt(tr{tran}(1),tr{tran}(2));
               vo=vo*(1/ratio);
            else
               vo=vo;
            end

end

if abs(abs(v(init_node))*sqrt(3)-vo*sqrt(3))>Ubase*epsilon
    % Start backward sweep
     clear treated_bs
        clear v_treated
        v(init_node)=vo;
        v_treated(1)=0; % The nodes downstream from terminal are considered
already treated
        treated_bs(1)=0; % The nodes downstream from terminal are considered
already treated
        np=init_node;
        treated_bs(2)=np;
        n=2;
        v_treated(n)=np;
        m=2;

        while length(treated_bs)~=Node+1
            % Trip towards terminal node
            while s{np}~=0
                b=1;
                while ismember(s{np}(b),v_treated)==1
                    b=b+1;
                end
                ns=s{np}(b);
                v(ns)=v(np)-z(np,ns)*I(np,ns);
                n=n+1;
                v_treated(n)=ns;
                if sum(ismember(s{np},v_treated))==length(s{np})
                    treated_bs(m)=np;
                    m=m+1;
                end
                np=ns;
            end
            treated_bs(m)=np;
            m=m+1;
            % Looking for the first node in v_treated still not in treated_bs
            q=1;
            while (ismember(v_treated(q),treated_bs)==1) &&
(length(treated_bs)~=Node+1)
                q=q+1;
            end
            np=v_treated(q);
```

```
        end

else
    stop=1;
end
end

% Pérdidas por efecto Joule en los conductores
Losses=0;
for n=1:1:Node
 len=length(s{n});
 for no=1:1:len
    ns=s{n}(no);
  if  s{n}~=(0)
    Losses=Losses+3*real(z(n,ns))*abs(I(n,ns))^2;
  end
 end
end

% Constraint node voltage
% Vlim6=0.95; %i added
% Obj_c1=Vlim6; % Tensión en el nudo 6 no puede ser menor de Vlim6 (expresada
en pu)
% if abs(v(6)*sqrt(3))*1/rt25*rt23*rt34/Ub6<Obj_c1
%     Constrain1=1e6+(abs(v(6)*sqrt(3))*1/rt25*rt23*rt34/Ub6-Obj_c1); %
Corriente por el conductor que une nudos 5 y 6 no puede ser mayor de 250A
%     Losses=Losses+Constrain1;
% end

Swarm(h).Fitness=Losses;
LossessJh(h)=Losses;
Lossessh(h)=Losses;

for n=1:Node
dlen=length(nod{n});
voltage_nodes(h,n)=abs(v(n)*sqrt(3))/Ub(n);
voltage_nodes_real(h,n)=abs(v(n)*sqrt(3));
           for u=1:dlen
           if nod{n}(u)<0
               tran=abs(nod{n}(u));
               ratio=rt(tr{tran}(1),tr{tran}(2));
               voltage_nodes(h,n)=voltage_nodes(h,n)*(1/ratio);
               voltage_nodes_real(h,n)=voltage_nodes_real(h,n)*(1/ratio);
           elseif nod{n}(u)>0
               tran=abs(nod{n}(u));
               ratio=rt(tr{tran}(1),tr{tran}(2));
               voltage_nodes(h,n)=voltage_nodes(h,n)*ratio;
               voltage_nodes_real(h,n)=voltage_nodes_real(h,n)*ratio;
           else
               voltage_nodes(h,n)=voltage_nodes(h,n);
               voltage_nodes_real(h,n)=voltage_nodes_real(h,n);
           end

           end
end
```

```
end

for h=1:N
    if Swarm(h).Fitness < Swarm(h).Best_Fitness
        Swarm(h).Best_Fitness=Swarm(h).Fitness;
        Swarm(h).Best_Fitness_X=Swarm(h).X;
        if Swarm(h).Best_Fitness < Swarm_Global.Best_Fitness
            Swarm_Global.Best_Fitness=Swarm(h).Best_Fitness;
            Swarm_Global.Best_Fitness_X=Swarm(h).Best_Fitness_X;
            LossesJbest=LossessJh(h);
            Lossesbest=Lossessh(h); % Es lo mismo que
Swarm_Global.Best_Fitness
            Voltagebest=voltage_nodes(h,node1);
            Voltagebest6=voltage_nodes(h,node2);
        end
    end
end

for h=1:N
    r1=random('Uniform',0,1); % randomization
    r2=random('Uniform',0,1); % randomization
    Swarm(h).V=w*Swarm(h).V+c1*r1*(Swarm(h).Best_Fitness_X-
Swarm(h).X)+c2*r2*(Swarm_Global.Best_Fitness_X-Swarm(h).X);
    Swarm(h).X=Swarm(h).X+Swarm(h).V;
    for t=1:Dim
        if Swarm(h).X(t)>maxX
            Swarm(h).X(t)=maxX;
        else
            if Swarm(h).X(t)<minX
                Swarm(h).X(t)=minX;
            end
        end
    end
end
end

display(['Optimum Losses ',num2str(round(Swarm_Global.Best_Fitness/100)/10),'
kW'])
display(['References: P=',
num2str(round(Swarm_Global.Best_Fitness_X(1)/100)/10), ' kW; Q1=',
num2str(round(Swarm_Global.Best_Fitness_X(2)/100)/10), 'kvar; Q2=',
num2str(round(Swarm_Global.Best_Fitness_X(3)/100)/10), 'kvar'])
```

6-node data m.file (chapter 4)

```
clear all
% Node list
N=6;
Node=6;

%% Choose the 2 nodes for the LFP control
node1=4;
node2=6;
```

```
%% PSO caracteristics for coefficient
N1_data=15; % particle number
%N1_data=1;
itmax1_data=90 ;
%itmax1_data=1; % no. de iteraciones
Dim1_data=3;
minX1_data=1.3;
maxX1_data=1.6;
minV1_data=-minX1_data/10;
maxV1_data=minX1_data/10;

minX2_data=0.6;
maxX2_data=1.8;
minV2_data=-minX2_data/10;
maxV2_data=minX2_data/10;

w_1_data = 0.729/10; % inertia weight
c_1_data = 1.49445/10; % cognitive weight
c_2_data = 1.49445/10; % social weight
%% PSO and Ladder caracteristics
N_data=20; % particle number
%N=1;
%itmax_data=70;
%itmax=1; % no. de iteraciones
Dim_data=3;
minX_data=-10e6;
maxX_data=10e6;
minV_data=-maxX_data;
maxV_data=maxX_data;

w_data = 0.729; % inertia weight
c1_data = 1.49445; % cognitive weight
c2_data = 1.49445; % social weight

epsilon_data=0.0001; % Tolerance for convergence (in per unit)
%epsilon_data=0.000000000001; % Tolerance for convergence (in per unit)

%% accuracy
Pacc_data=-2.322373547904614e+06;
Q1acc_data=-7.646484135096628e+06;
Q2acc_data=-4.565359650387196e+06;
lossesacc_data=1.292945219010391e+06;

acc_data=0.01; % 1%
acc1_data=0.01;

%% Sons of the different nodes. [0] means it is a terminal node.
s{1}=[2];
s{2}=[3,5];
s{3}=[4];
s{4}=[0];
s{5}=[6];
s{6}=[0];
```

```
%% Parents (this should be built automatically)
for n=1:Node
p{n}=[0];
end
for n=1:Node
    len=length(s{n});
    for no=1:len
        if s{n}(no)~=0
        p{s{n}(no)}=[n];
        end
    end
end

%% Loads
S(1)=(20+25i)*1e6;
S(2)=(84+26i)*1e6;
S(3)=(34+12i)*1e6;
S(4)=(7.5+5i)*1e6;
S(5)=(52+39i)*1e6;
S(6)=(1.3+2i)*1e6;

%% Base Values
Ubase=13.8e3; % Tomado como general del sistema
Ub(1)=220e3;    % Nodo 1
Ub(2)=132e3;    % Nodo 2
Ub(3)=30e3;     % Nodo 3
Ub(4)=Ubase;    % Nodo 4
Ub(5)=Ubase;    % Nodo 5
Ub(6)=Ubase;    % Nodo 6

%% Transformers
% Datos de los transformadores (potencia nominal, número y ecc)
Sn(1:N,1:N)=zeros;
Sn(1,2)=270e6;
Nt(1,2)=2;
Sn(2,3)=37.5e6;
Nt(2,3)=3;
Sn(2,5)=50e6;
Nt(2,5)=3;
Sn(3,4)=10e6;
Nt(3,4)=1;
% Taps de los trafos
tap(1,2)=0.01;
tap(2,3)=0;
tap(3,4)=-0.01;
tap(2,5)=-0.035;
% Impedancias de los trafos lado aguas arriba
eRcc(1,2)=0.0090;
eXcc(1,2)=0.1297;
eRcc(2,3)=0.0090;
eXcc(2,3)=0.0895;
eRcc(2,5)=0.0092;
eXcc(2,5)=0.0795;
eRcc(3,4)=0.0095;
eXcc(3,4)=0.0476;
```

```
% Reference base node
tr{1}=[1,2];
tr{2}=[2,3];
tr{3}=[2,5];
tr{4}=[3,4];

nref=4; % node 4
% transformers from the node to the node reference
% trasnformer number and +(to primary of transformer 1/rt) - (to secondary
% rt) and 0 if no ratio transformation needed
d{1,2}=[1,2,4];
d{2,3}=[2,4];
d{2,5}=[2 4];
d{3,4}=[4];
d{5,6}=[2,-3,4];

nod{1}=[1,2,4];
nod{2}=[2,4];
nod{3}=[4];
nod{4}=[0];
nod{5}=[2,-3,4];
nod{6}=[2,-3,4];

%% Lines
% (lengths)
l_(1,2)=4700e-3;
l_(2,3)=1500e-3;
l_(2,5)=0e-3;
l_(3,4)=300e-3;
l_(5,6)=1800e-3;

% (pu impedance)
zc_(1,2)=0.025+0.240i;
zc_(2,3)=0.161+0.151i;
zc_(2,5)=0.062+0.165i;
zc_(3,4)=0.568+0.133i;
zc_(5,6)=0.161+0.112i;
% Lines (impedances)
zl(1,2)=(l_(1,2)*zc_(1,2));
zl(2,3)=(l_(2,3)*zc_(2,3));
zl(2,5)=(l_(2,5)*zc_(2,5));
zl(3,4)=(l_(3,4)*zc_(3,4));
zl(5,6)=(l_(5,6)*zc_(5,6));
```

Chapter 4 m.file

```
clear all
dist_grid_data_6node_task2;

%% accuracy
Pacc=Pacc_data;
Q1acc=Q1acc_data;
Q2acc=Q2acc_data;
lossesacc=lossesacc_data;
```

```
acc=acc_data; % 1%
acc1=acc1_data;

%% RECEPCIÓN DE DATOS
Sp(node1)=S(node1);
Sp(node2)=S(node2);

%% Transformer caracteristic
% Transformer relation and impedences
rt(1:Node,1:Node)=zeros;
zt(1:Node,1:Node)=zeros;
for n=1:1:Node
 len=length(s{n});
 for no=1:1:len
    ns=s{n}(no);
  if  s{n}~=(0)
    if Sn(n,ns)~=0
    rt(n,ns)=Ub(n)/(Ub(ns)*(1+tap(n,ns)));
    zt(n,ns)=Ub(n)^2/Sn(n,ns)*(eRcc(n,ns)+eXcc(n,ns)*1i)*1/Nt(n,ns);
    end
  end
 end
end

% Total impedances
% Reference base voltage Ubase 13.8 KV
z(1:Node,1:Node)=zeros;
for n=1:Node
  len=length(s{n});
  for no=1:len
      ns=s{n}(no);
      if  s{n}~=(0)
        z(n,ns)=zl(n,ns)+zt(n,ns);
      end
  end
end

for n=1:Node
  len=length(s{n});
  for no=1:len
      ns=s{n}(no);
       if  s{n}~=(0)
           dlen=length(d{n,ns});
           for u=1:dlen
           if d{n,ns}(u)<0
               tran=abs(d{n,ns}(u));
               ratio=rt(tr{tran}(1),tr{tran}(2));
               z(n,ns)=z(n,ns)*ratio^2;
           elseif d{n,ns}(u)>0
               tran=abs(d{n,ns}(u));
               ratio=rt(tr{tran}(1),tr{tran}(2));
               z(n,ns)=z(n,ns)*(1/ratio)^2;
           else
               z(n,ns)=z(n,ns);
           end
```

```
            end

        end
    end
end

%% SE CALCULA LA TENSIÓN MEDIANTE LADDER

% Ladder iterative method
epsilon=epsilon_data; % Tolerance for convergence (in per unit)

%% PARTICLE SWARM OPTIMIZATION for coeff opt
N1=N1_data; % particle number
itmax1=itmax1_data ; % no. de iteraciones
Dim1=Dim1_data;
minX1=minX1_data;
maxX1=maxX1_data;
minV1=minV1_data;
maxV1=maxV1_data;

minX2=minX2_data;
maxX2=maxX2_data;
minV2=minV2_data;
maxV2=maxV2_data;

w_1 = w_1_data; % inertia weight
c_1 = c_1_data; % cognitive weight
c_2 = c_2_data; % social weight

% Allocation
X1dat(1:N1)={zeros(1,3)};
V1dat(1:N1)={zeros(1,3)};
%X1dat(1:N1)={[1 1]};
%V1dat(1:N1)={[1 1]};

Swarm1=struct('X1',X1dat,'V1',V1dat,'Best_Fitness1',inf);
% End allocation
%Swarm(1).X=[0,0,0];
for h1=2:N1

Swarm1(h1).X1=[random('Uniform',minX2,maxX2),random('Uniform',minX1,maxX1),ra
ndom('Uniform',minX1,maxX1)];
end
for h1=1:N1

Swarm1(h1).V1=[random('Uniform',minV2,maxV2),random('Uniform',minV1,maxV1),ra
ndom('Uniform',minV1,maxV1)];
end
for h1=1:N1
    Swarm1(h1).Best_Fitness1=inf; % Inicialización del best_fitness a un
valor infinito
end
Swarm_Global1.Best_Fitness1=inf; % Inicialización del Global_best_fitness a
un valor infinito
```

```
for it1=1:itmax1
    %it1
for h1=1:N1
    %h
    if Swarm1(h1).X1(1)==0 && Swarm1(h1).X1(2)==0 && Swarm1(h1).X1(3)==0
    w = 0.729; % inertia weight
    c1 = 1.49445; % cognitive weight
    c2 = 1.49445; % social weight
    Swarm1(h1).X1(1)=w;
    Swarm1(h1).X1(2)=c1;
    Swarm1(h1).X1(3)=c2;
    else
    w=Swarm1(h1).X1(1);
    c1=Swarm1(h1).X1(2);
    c2=Swarm1(h1).X1(3);
    end

%avg=0;
%avg1=0;
%for c=1:5

%-------------------------------------------------------------------------
%% PARTICLE SWARM OPTIMIZATION for the feeder
N=N_data; % particle number
%itmax=itmax_data;% no. de iteraciones
Dim=Dim_data;
minX=minX_data;
maxX=maxX_data;
minV=minV_data;
maxV=maxV_data;

% Allocation
Xdat(1:N)={zeros(1,3)};
Vdat(1:N)={zeros(1,3)};
Swarm=struct('X',Xdat,'V',Vdat,'Best_Fitness',inf);
% End allocation

%Swarm(1).X=[0,0,0];
for h=2:N

Swarm(h).X=[random('Uniform',minX,maxX),random('Uniform',minX,maxX),random('U
niform',minX,maxX)];
end
for h=1:N

Swarm(h).V=[random('Uniform',minV,maxV),random('Uniform',minV,maxV),random('U
niform',minV,maxV)];
end
for h=1:N
    Swarm(h).Best_Fitness=inf; % Inicialización del best_fitness a un valor
infinito
end
Swarm_Global.Best_Fitness=inf; % Inicialización del Global_best_fitness a un
valor infinito
```

```
%for it=1:itmax
it=0;
convergence=0;
while convergence==0
    it=it+1;
%    it
    % Allocation
    Current(1:N)=zeros(1,N);
    Sconv(1:N)=zeros(1,N);
    voltage_nodes(1:N,1:Node)=zeros(N,Node)+1i*zeros(N,Node);
    voltage_nodes_real(1:N,1:Node)=zeros(N,Node)+1i*zeros(N,Node);
    % End allocation
for h=1:N
    %h
    P=Swarm(h).X(1);
    Q1=Swarm(h).X(2);
    Q2=Swarm(h).X(3);
    SL(node1)=P+Q1*1i;
    SL(node2)=-P+Q2*1i;

%% Ladder iterative method
% Variables preallocation
v(1:Node)=zeros;
I(1:Node,1:Node)=zeros;
v_treated(1:Node+1)=zeros;
% Initial start for voltage at terminal nodes
for nu=1:Node
    if s{nu}==[0]
        v(nu)=Ubase/sqrt(3);
    end
end
% Looking for initial node
nu=1;
while p{nu}~=[0]
    nu=nu+1;
end
init_node=nu;

iter=0;
stop=0;
while stop==0
iter=iter+1;

S(node1)=Sp(node1)+SL(node1);
S(node2)=Sp(node2)+SL(node2);

% Forward Sweep
    clear treated
    treated=0;
% Looking for a terminal node to start
nu=1;
while s{nu}~=[0]
    nu=nu+1;
end
ns=nu;
I(ns,ns)=(S(ns)/(3*v(ns)))';
```

```
np=p{ns};
I(np,ns)=I(ns,ns);
v(np)=v(ns)+z(np,ns)*I(np,ns);
treated(length(treated)+1)=ns;
ns=np;
while length(treated)<Node+1
    if sum(ismember(s{ns},treated))==length(s{ns}) % if the sons of this node
are treated we treat the node
        I(ns,ns)=(S(ns)/(3*v(ns)))';
        np=p{ns};
        if np==[0] % For the case we reach the initial node
            I(ns)=sum(I(ns,:));
        else % We does not still reach the initial node
            I(np,ns)=sum(I(ns,:));
            v(np)=v(ns)+z(np,ns)*I(np,ns);
        end
        treated(length(treated)+1)=ns;
        ns=np;
    else % if any of the sons of the node are not treated we look for a
terminal node
        while s{ns}~=[0]
            b=1;
            while ismember(s{ns}(b),treated)==1
                b=b+1;
            end
            ns=s{ns}(b);
        end
    end
end

%
 dlen=length(nod{init_node});
 vo=abs(Ub(init_node)/sqrt(3));
 for u=1:dlen
            if nod{init_node}(u)<0
               tran=abs(nod{init_node}(u));
               ratio=rt(tr{tran}(1),tr{tran}(2));
               vo=vo*ratio;
            elseif nod{init_node}(u)>0
               tran=abs(nod{init_node}(u));
               ratio=rt(tr{tran}(1),tr{tran}(2));
               vo=vo*(1/ratio);
            else
               vo=vo;
            end

end

if abs(abs(v(init_node))*sqrt(3)-vo*sqrt(3))>Ubase*epsilon
    % Start backward sweep
     clear treated_bs
        clear v_treated
        v(init_node)=vo;
        v_treated(1)=0; % The nodes downstream from terminal are considered
already treated
```

```
        treated_bs(1)=0; % The nodes downstream from terminal are considered
already treated
        np=init_node;
        treated_bs(2)=np;
        n=2;
        v_treated(n)=np;
        m=2;

        while length(treated_bs)~=Node+1
            % Trip towards terminal node
            while s{np}~=0
                b=1;
                while ismember(s{np}(b),v_treated)==1
                    b=b+1;
                end
                ns=s{np}(b);
                v(ns)=v(np)-z(np,ns)*I(np,ns);
                n=n+1;
                v_treated(n)=ns;
                if sum(ismember(s{np},v_treated))==length(s{np})
                    treated_bs(m)=np;
                    m=m+1;
                end
                np=ns;
            end
            treated_bs(m)=np;
            m=m+1;
            % Looking for the first node in v_treated still not in treated_bs
            q=1;
            while (ismember(v_treated(q),treated_bs)==1) &&
(length(treated_bs)~=Node+1)
                q=q+1;
            end
            np=v_treated(q);
        end

else
    stop=1;
end
end

% Pérdidas por efecto Joule en los conductores
Losses=0;
for n=1:1:Node
 len=length(s{n});
 for no=1:1:len
    ns=s{n}(no);
  if  s{n}~=(0)
    Losses=Losses+3*real(z(n,ns))*abs(I(n,ns))^2;
  end
 end
end

% Constraint node voltage
% Vlim6=0.95; %i added
```

```
% Obj_c1=Vlim6; % Tensión en el nudo 6 no puede ser menor de Vlim6 (expresada
en pu)
% if abs(v(6)*sqrt(3))*1/rt25*rt23*rt34/Ub6<Obj_c1
%     Constrain1=1e6+(abs(v(6)*sqrt(3))*1/rt25*rt23*rt34/Ub6-Obj_c1); %
Corriente por el conductor que une nudos 5 y 6 no puede ser mayor de 250A
%     Losses=Losses+Constrain1;
% end

Swarm(h).Fitness=Losses;
LossessJh(h)=Losses;
Lossessh(h)=Losses;

% for n=1:Node
% dlen=length(nod{n});
% voltage_nodes(h,n)=abs(v(n)*sqrt(3))/Ub(n);
% voltage_nodes_real(h,n)=abs(v(n)*sqrt(3));
%             for u=1:dlen
%             if nod{n}(u)<0
%                 tran=abs(nod{n}(u));
%                 ratio=rt(tr{tran}(1),tr{tran}(2));
%                 voltage_nodes(h,n)=voltage_nodes(h,n)*(1/ratio);
%                 voltage_nodes_real(h,n)=voltage_nodes_real(h,n)*(1/ratio);
%             elseif nod{n}(u)>0
%                 tran=abs(nod{n}(u));
%                 ratio=rt(tr{tran}(1),tr{tran}(2));
%                 voltage_nodes(h,n)=voltage_nodes(h,n)*ratio;
%                 voltage_nodes_real(h,n)=voltage_nodes_real(h,n)*ratio;
%             else
%                 voltage_nodes(h,n)=voltage_nodes(h,n);
%                 voltage_nodes_real(h,n)=voltage_nodes_real(h,n);
%             end
%
%             end
% end

end

for h=1:N
    if Swarm(h).Fitness < Swarm(h).Best_Fitness
        Swarm(h).Best_Fitness=Swarm(h).Fitness;
        Swarm(h).Best_Fitness_X=Swarm(h).X;
        if Swarm(h).Best_Fitness < Swarm_Global.Best_Fitness
            Swarm_Global.Best_Fitness=Swarm(h).Best_Fitness;
            Swarm_Global.Best_Fitness_X=Swarm(h).Best_Fitness_X;
            LossesJbest=LossessJh(h);
            Lossesbest=Lossessh(h); % Es lo mismo que
Swarm_Global.Best_Fitness
            Voltagebest=voltage_nodes(h,node1);
            Voltagebest6=voltage_nodes(h,node2);
        end
    end
end

for h=1:N
    r1=random('Uniform',0,1); % randomization
    r2=random('Uniform',0,1); % randomization
```

```
    Swarm(h).V=w*Swarm(h).V+c1*r1*(Swarm(h).Best_Fitness_X-
Swarm(h).X)+c2*r2*(Swarm_Global.Best_Fitness_X-Swarm(h).X);
    Swarm(h).X=Swarm(h).X+Swarm(h).V;
    for t=1:Dim
        if Swarm(h).X(t)>2*maxX
            Swarm(h).X(t)=2*maxX;
        else
            if Swarm(h).X(t)<2*minX
                Swarm(h).X(t)=2*minX;
            end
        end
    end
end


% end while
%accuracy
P_best=Swarm_Global.Best_Fitness_X(1);
Q1_best=Swarm_Global.Best_Fitness_X(2);
Q2_best=Swarm_Global.Best_Fitness_X(3);
Accu_P=abs((Pacc-P_best)/Pacc);
Accu_Q1=abs((Q1acc-Q1_best)/Q1acc);
Accu_Q2=abs((Q2acc-Q2_best)/Q2acc);

Acc_matrix=[Accu_P,Accu_Q1,Accu_Q2];
precision=100*max(Acc_matrix);
%     Accu_P
%     Accu_Q1
%     Accu_Q2

if Accu_P<acc && Accu_Q1<acc && Accu_Q2<acc
    convergence=1;
elseif it==201
    convergence=1;

else
    convergence=0;
end
end

display(['Optimum Losses ',num2str(round(Swarm_Global.Best_Fitness/100)/10),'
kW'])
display(['References: P=',
num2str(round(Swarm_Global.Best_Fitness_X(1)/100)/10), ' kW; Q1=',
num2str(round(Swarm_Global.Best_Fitness_X(2)/100)/10), 'kvar; Q2=',
num2str(round(Swarm_Global.Best_Fitness_X(3)/100)/10), 'kvar'])
it
%-------------------------------------------------------------------------
%% rest of PARTICLE SWARM OPTIMIZATION for coeff opt
telapsed=it+precision;
num_iteration=it;

%accuracy
Swarm1(h1).Fitness2=Swarm_Global.Best_Fitness_X(1); %P
Swarm1(h1).Fitness3=Swarm_Global.Best_Fitness_X(2); %Q1
```

```
Swarm1(h1).Fitness4=Swarm_Global.Best_Fitness_X(3); %Q2
Swarm1(h1).Fitness5=Swarm_Global.Best_Fitness; %losses

%accuracy calculation
Acc_P(h1)=abs((Pacc-Swarm1(h1).Fitness2)/Pacc);
Acc_Q1(h1)=abs((Q1acc-Swarm1(h1).Fitness3)/Q1acc);
Acc_Q2(h1)=abs((Q2acc-Swarm1(h1).Fitness4)/Q2acc);
Acc_losses(h1)=abs((lossesacc-Swarm1(h1).Fitness5)/lossesacc);

%time
Swarm1(h1).Fitness1=telapsed;
Swarm1(h1).Fitness6=num_iteration;

if Acc_P(h1)<acc1 && Acc_Q1(h1)<acc1 && Acc_Q2(h1)<acc1 &&
Acc_losses(h1)<acc1
%display('nothing'); % do nothing
else
   Swarm1(h1).Fitness1=1e8;
end

%avg=avg+Swarm1(h1).Fitness1;
%avg1=avg1+Swarm1(h1).Fitness6;
%end
%Swarm1(h1).Fitness1=avg/5;
%Swarm1(h1).Fitness6=avg1/5;
end

for h1=1:N1
    h1
    Part_iter=Swarm1(h1).X1

    if Acc_P(h1)<acc1 && Acc_Q1(h1)<acc1 && Acc_Q2(h1)<acc1 &&
Acc_losses(h1)<acc1
        display('Valid')
    else
%        Swarm1(h1).Fitness1=1e8;
        display('not valid')
    end

    Swarm1(h1).Fitness1=mean(Swarm1(h1).Fitness1);

    if Swarm1(h1).Fitness1 < Swarm1(h1).Best_Fitness1
        Swarm1(h1).Best_Fitness1=Swarm1(h1).Fitness1;
        Swarm1(h1).Best_Fitness2=Swarm1(h1).Fitness6;
        Swarm1(h1).Best_Fitness1_X=Swarm1(h1).X1;
        if Swarm1(h1).Best_Fitness1 < Swarm_Global1.Best_Fitness1
            Swarm_Global1.Best_Fitness1=Swarm1(h1).Best_Fitness1;
            Swarm_Global1.Best_Fitness2=Swarm1(h1).Best_Fitness2;
            Swarm_Global1.Best_Fitness1_X=Swarm1(h1).Best_Fitness1_X;
        end
    end

end

for h1=1:N1
```

```
    r1=random('Uniform',0,1); % randomization
    r2=random('Uniform',0,1); % randomization
    Swarm1(h1).V1=w_1*Swarm1(h1).V1+c_1*r1*(Swarm1(h1).Best_Fitness1_X-
Swarm1(h1).X1)+c_2*r2*(Swarm_Global1.Best_Fitness1_X-Swarm1(h1).X1);

%         t=1;
%         if Swarm(h).V(t)>2*maxV2
%             Swarm(h).V(t)=2*maxV2;
%         else
%             if Swarm(h).V(t)<2*minV2
%                 Swarm(h).V(t)=2*minV2;
%             end
%         end
%     for t=2:Dim1
%         if Swarm(h).V(t)>2*maxV1
%             Swarm(h).V(t)=2*maxV1;
%         else
%             if Swarm(h).V(t)<2*minV1
%                 Swarm(h).V(t)=2*minV1;
%             end
%         end
%     end

    Swarm1(h1).X1=Swarm1(h1).X1+Swarm1(h1).V1;

%         t=1;
%         if Swarm1(h1).X1(t)>2*maxX2
%             Swarm1(h1).X1(t)=2*maxX2;
%         else
%             if Swarm1(h1).X1(t)<2*minX2
%                 Swarm1(h1).X1(t)=2*minX2;
%             end
%         end
%     for t=2:Dim1
%         if Swarm1(h1).X1(t)>2*maxX1
%             Swarm1(h1).X1(t)=2*maxX1;
%         else
%             if Swarm1(h1).X1(t)<2*minX1
%                 Swarm1(h1).X1(t)=2*minX1;
%             end
%         end
%     end
end

end

display(['Optimum Time ',num2str(round(Swarm_Global1.Best_Fitness1)),'
secondes'])
display(['Optimum number of iteration
',num2str(round(Swarm_Global1.Best_Fitness2)),' number'])
display(['References: w_1=', num2str(Swarm_Global1.Best_Fitness1_X(1)), '
number; c_1=', num2str(Swarm_Global1.Best_Fitness1_X(2)), 'number; c_2=',
num2str(Swarm_Global1.Best_Fitness1_X(3)), 'number'])
```

Chapter 5 m.file

```
clear all
dist_grid_data_6node;
%task3_loadalloc;

%% RECEPCIÓN DE DATOS
Sp(node1)=S(node1);
Sp(node2)=S(node2);

%% importing from excel
filename='dailyLoad';
sheet=1;

num=xlsread(filename,sheet);

%% Transformer caracteristic
% Transformer relation and impedences
rt(1:Node,1:Node)=zeros;
zt(1:Node,1:Node)=zeros;
for n=1:1:Node
 len=length(s{n});
 for no=1:1:len
    ns=s{n}(no);
  if  s{n}~=(0)
    if Sn(n,ns)~=0
    rt(n,ns)=Ub(n)/(Ub(ns)*(1+tap(n,ns)));
    zt(n,ns)=Ub(n)^2/Sn(n,ns)*(eRcc(n,ns)+eXcc(n,ns)*1i)*1/Nt(n,ns);
    end
  end
 end
end

% Total impedances
% Reference base voltage Ubase 13.8 KV
z(1:Node,1:Node)=zeros;
for n=1:Node
  len=length(s{n});
  for no=1:len
      ns=s{n}(no);
      if  s{n}~=(0)
        z(n,ns)=zl(n,ns)+zt(n,ns);
      end
  end
end

for n=1:Node
  len=length(s{n});
  for no=1:len
      ns=s{n}(no);
       if  s{n}~=(0)
           dlen=length(d{n,ns});
           for u=1:dlen
           if d{n,ns}(u)<0
               tran=abs(d{n,ns}(u));
```

```
                ratio=rt(tr{tran}(1),tr{tran}(2));
                z(n,ns)=z(n,ns)*ratio^2;
            elseif d{n,ns}(u)>0
                tran=abs(d{n,ns}(u));
                ratio=rt(tr{tran}(1),tr{tran}(2));
                z(n,ns)=z(n,ns)*(1/ratio)^2;
            else
                z(n,ns)=z(n,ns);
            end

            end

        end
  end
end

%% SE CALCULA LA TENSIÓN MEDIANTE LADDER

% Ladder iterative method
epsilon=epsilon_data; % Tolerance for convergence (in per unit)

%initial
eloss=0;
Total_elosses=0;
t_delta_previous=0;
total_delta=0;
S1max=0;
S2max=0;

% number of timetable
timetable=0;
sample_tt=1;
while num(sample_tt,1)~=0
 timetable=timetable+1;
 sample_tt=sample_tt+1;
end

for sample_t=1:timetable
% time difference
t_1=num(sample_t,1);
t_2=num(sample_t+1,1);
if t_2==0
    t_delta=t_delta_previous;
else
t_delta=t_2-t_1;
end

%% load allocation
for n=1:Node
pn(n)=num(sample_t,2*n);
qn(n)=num(sample_t,(2*n)+1);
S(n)=pn(n)+1i*qn(n);
end

%--------------------------------------------------------------------------
```

```
%% PARTICLE SWARM OPTIMIZATION
N=N_data; % particle number
itmax=itmax_data; % no. de iteraciones
Dim=Dim_data;
minX=minX_data;
maxX=maxX_data;
minV=minV_data;
maxV=maxV_data;

w = w_data; % inertia weight
c1 = c1_data; % cognitive weight
c2 = c2_data; % social weight

% Allocation
Xdat(1:N)={zeros(1,3)};
Vdat(1:N)={zeros(1,3)};
Swarm=struct('X',Xdat,'V',Vdat,'Best_Fitness',inf);
% End allocation

%Swarm(1).X=[0,0,0];
for h=2:N

Swarm(h).X=[random('Uniform',minX,maxX),random('Uniform',minX,maxX),random('U
niform',minX,maxX)];
end
for h=1:N

Swarm(h).V=[random('Uniform',minV,maxV),random('Uniform',minV,maxV),random('U
niform',minV,maxV)];
end
for h=1:N
    Swarm(h).Best_Fitness=inf; % Inicialización del best_fitness a un valor
infinito
end
Swarm_Global.Best_Fitness=inf; % Inicialización del Global_best_fitness a un
valor infinito
for it=1:itmax
    %it
    % Allocation
    Current(1:N)=zeros(1,N);
    Sconv(1:N)=zeros(1,N);
    voltage_nodes(1:N,1:Node)=zeros(N,Node)+1i*zeros(N,Node);
    voltage_nodes_real(1:N,1:Node)=zeros(N,Node)+1i*zeros(N,Node);
    % End allocation
for h=1:N
    %h
    P=Swarm(h).X(1);
    Q1=Swarm(h).X(2);
    Q2=Swarm(h).X(3);
    SL(node1)=P+Q1*1i;
    SL(node2)=-P+Q2*1i;

%% Ladder iterative method
% Variables preallocation
v(1:Node)=zeros;
I(1:Node,1:Node)=zeros;
```

```
v_treated(1:Node+1)=zeros;
% Initial start for voltage at terminal nodes
for nu=1:Node
    if s{nu}==[0]
        v(nu)=Ubase/sqrt(3);
    end
end
% Looking for initial node
nu=1;
while p{nu}~=[0]
    nu=nu+1;
end
init_node=nu;

iter=0;
stop=0;
while stop==0
iter=iter+1;

S(node1)=Sp(node1)+SL(node1);
S(node2)=Sp(node2)+SL(node2);

% Forward Sweep
    clear treated
    treated=0;
% Looking for a terminal node to start
nu=1;
while s{nu}~=[0]
    nu=nu+1;
end
ns=nu;
I(ns,ns)=(S(ns)/(3*v(ns)))';
np=p{ns};
I(np,ns)=I(ns,ns);
v(np)=v(ns)+z(np,ns)*I(np,ns);
treated(length(treated)+1)=ns;
ns=np;
while length(treated)<Node+1
    if sum(ismember(s{ns},treated))==length(s{ns}) % if the sons of this node
are treated we treat the node
        I(ns,ns)=(S(ns)/(3*v(ns)))';
        np=p{ns};
        if np==[0] % For the case we reach the initial node
            I(ns)=sum(I(ns,:));
        else % We does not still reach the initial node
            I(np,ns)=sum(I(ns,:));
            v(np)=v(ns)+z(np,ns)*I(np,ns);
        end
        treated(length(treated)+1)=ns;
        ns=np;
    else % if any of the sons of the node are not treated we look for a
terminal node
        while s{ns}~=[0]
            b=1;
            while ismember(s{ns}(b),treated)==1
                b=b+1;
```

```
            end
            ns=s{ns}(b);
        end
    end
end

%
 dlen=length(nod{init_node});
 vo=abs(Ub(init_node)/sqrt(3));
 for u=1:dlen
            if nod{init_node}(u)<0
                tran=abs(nod{init_node}(u));
                ratio=rt(tr{tran}(1),tr{tran}(2));
                vo=vo*ratio;
            elseif nod{init_node}(u)>0
                tran=abs(nod{init_node}(u));
                ratio=rt(tr{tran}(1),tr{tran}(2));
                vo=vo*(1/ratio);
            else
                vo=vo;
            end

end

if abs(abs(v(init_node))*sqrt(3)-vo*sqrt(3))>Ubase*epsilon
    % Start backward sweep
     clear treated_bs
        clear v_treated
        v(init_node)=vo;
        v_treated(1)=0; % The nodes downstream from terminal are considered
already treated
        treated_bs(1)=0; % The nodes downstream from terminal are considered
already treated
        np=init_node;
        treated_bs(2)=np;
        n=2;
        v_treated(n)=np;
        m=2;

        while length(treated_bs)~=Node+1
            % Trip towards terminal node
            while s{np}~=0
                b=1;
                while ismember(s{np}(b),v_treated)==1
                    b=b+1;
                end
                ns=s{np}(b);
                v(ns)=v(np)-z(np,ns)*I(np,ns);
                n=n+1;
                v_treated(n)=ns;
                if sum(ismember(s{np},v_treated))==length(s{np})
                    treated_bs(m)=np;
                    m=m+1;
                end
                np=ns;
            end
```

```
            treated_bs(m)=np;
            m=m+1;
            % Looking for the first node in v_treated still not in treated_bs
            q=1;
            while (ismember(v_treated(q),treated_bs)==1) &&
(length(treated_bs)~=Node+1)
                q=q+1;
            end
            np=v_treated(q);
        end

else
    stop=1;
end
end

% Pérdidas por efecto Joule en los conductores
Losses=0;
for n=1:1:Node
 len=length(s{n});
 for no=1:1:len
    ns=s{n}(no);
  if  s{n}~=(0)
    Losses=Losses+3*real(z(n,ns))*abs(I(n,ns))^2;
  end
 end
end

% Constraint node voltage
% Vlim6=0.95; %i added
% Obj_c1=Vlim6; % Tensión en el nudo 6 no puede ser menor de Vlim6 (expresada
en pu)
% if abs(v(6)*sqrt(3))*1/rt25*rt23*rt34/Ub6<Obj_c1
%     Constrain1=1e6+(abs(v(6)*sqrt(3))*1/rt25*rt23*rt34/Ub6-Obj_c1); %
Corriente por el conductor que une nudos 5 y 6 no puede ser mayor de 250A
%     Losses=Losses+Constrain1;
% end

Swarm(h).Fitness=Losses;
LossessJh(h)=Losses;
Lossessh(h)=Losses;

for n=1:Node
dlen=length(nod{n});
voltage_nodes(h,n)=abs(v(n)*sqrt(3))/Ub(n);
voltage_nodes_real(h,n)=abs(v(n)*sqrt(3));
           for u=1:dlen
           if nod{n}(u)<0
               tran=abs(nod{n}(u));
               ratio=rt(tr{tran}(1),tr{tran}(2));
               voltage_nodes(h,n)=voltage_nodes(h,n)*(1/ratio);
               voltage_nodes_real(h,n)=voltage_nodes_real(h,n)*(1/ratio);
           elseif nod{n}(u)>0
               tran=abs(nod{n}(u));
               ratio=rt(tr{tran}(1),tr{tran}(2));
               voltage_nodes(h,n)=voltage_nodes(h,n)*ratio;
```

```
                voltage_nodes_real(h,n)=voltage_nodes_real(h,n)*ratio;
            else
                voltage_nodes(h,n)=voltage_nodes(h,n);
                voltage_nodes_real(h,n)=voltage_nodes_real(h,n);
            end

            end
end

end

for h=1:N
    if Swarm(h).Fitness < Swarm(h).Best_Fitness
        Swarm(h).Best_Fitness=Swarm(h).Fitness;
        Swarm(h).Best_Fitness_X=Swarm(h).X;
        if Swarm(h).Best_Fitness < Swarm_Global.Best_Fitness
            Swarm_Global.Best_Fitness=Swarm(h).Best_Fitness;
            Swarm_Global.Best_Fitness_X=Swarm(h).Best_Fitness_X;
            LossesJbest=LossessJh(h);
            Lossesbest=Lossessh(h); % Es lo mismo que
Swarm_Global.Best_Fitness
            Voltagebest=voltage_nodes(h,node1);
            Voltagebest6=voltage_nodes(h,node2);
        end
    end
end

for h=1:N
    r1=random('Uniform',0,1); % randomization
    r2=random('Uniform',0,1); % randomization
    Swarm(h).V=w*Swarm(h).V+c1*r1*(Swarm(h).Best_Fitness_X-
Swarm(h).X)+c2*r2*(Swarm_Global.Best_Fitness_X-Swarm(h).X);
    Swarm(h).X=Swarm(h).X+Swarm(h).V;
    for t=1:Dim
        if Swarm(h).X(t)>maxX
            Swarm(h).X(t)=maxX;
        else
            if Swarm(h).X(t)<minX
                Swarm(h).X(t)=minX;
            end
        end
    end
end
end

display(['Optimum Losses ',num2str(round(Swarm_Global.Best_Fitness/100)/10),'
kW'])
display(['References: P=',
num2str(round(Swarm_Global.Best_Fitness_X(1)/100)/10), ' kW; Q1=',
num2str(round(Swarm_Global.Best_Fitness_X(2)/100)/10), 'kvar; Q2=',
num2str(round(Swarm_Global.Best_Fitness_X(3)/100)/10), 'kvar'])

loss=Swarm_Global.Best_Fitness;

%energy
```

```
eloss=loss*t_delta;
Total_elosses=Total_elosses+eloss;
t_delta_previous=t_delta;
total_delta=total_delta+t_delta;

%S1 S2 max
Pref=Swarm_Global.Best_Fitness_X(1);
Q1ref=Swarm_Global.Best_Fitness_X(2);
Q2ref=Swarm_Global.Best_Fitness_X(3);
S1=sqrt(Pref^2+Q1ref^2);
S2=sqrt(Pref^2+Q2ref^2);
if S1>S1max
    S1max=S1;
end
if S2>S2max
    S2max=S2;
end
end

Total_losses=Total_elosses/total_delta;
display(['Optimum energy Losses 24h ',num2str(round(Total_elosses/100)/10),'
kWh'])
display(['Optimum Losses 24h ',num2str(round(Total_losses/100)/10),' kW'])
display(['Max converter power: S1 ',num2str(round(S1max/100)/10),' kW;
S2=',num2str(round(S2max/100)/10)])
```