11

PV Curves for Steady-State Security Assessment with MATLAB

Ricardo Vargas, M.A Arjona and Manuel Carrillo
Instituto Tecnológico de la Laguna
División de Estudios de Posgrado e Investigación
México

1. Introduction

Most of the problem solutions oriented to the analysis of power systems require the implementation of sophisticated algorithms which need a considerable amount of calculations that must be carried out with a digital computer. Advances in software and hardware engineering have led to the development of specialized computing tools in the area of electrical power systems which allows its efficient analysis. Most of the computational programs, if not all of them, are developed under proprietary code, in other words, the users does not have access to the source code, which limits its usage scope. These programs are considered as black boxes that users only need to feed the required input data to obtain the results without knowing anything about the details of the inner program structure. In the academic or research areas this kind of programs does not fulfill all needs that are required hence it is common the usage of programming tools oriented to the scientific computing. These tools facilitate the development of solution algorithms for any engineering problem, by taking into account the mathematical formulations which define the solution of the proposed problem. Besides, it is also common that most of these programs are known as script or interpreted languages, such as MATLAB, Python and Perl. They all have the common feature of being high level programming languages that usually make use of available efficient libraries in a straightforward way. MATLAB is considered as a programming language that has become a good option for many researchers in different science and engineering areas because of it can allow the creation, manipulation and operation of sparse or full matrices; it also allows to the user the programming of any mathematical algorithm by means of an ordered sequence of commands (code) written into an ascii file known as script files. These files are portable, i.e. they can be executed in most of software versions in any processor under the operating systems Linux or windows.

The main objective of this chapter consists in presenting an efficient alternative of developing a script program in the MATLAB environment; the program can generate characteristic curves power vs. voltage (PV curves) of each node in a power system. The curves are used to analyze and evaluate the stability voltage limits in steady state, and they are calculated by employing an algorithm known as continuous load flows, which are a variation of the Newton-Raphson formulation for load flows but it avoids any possibility of singularity during the solution process under a scenario of continuous load variation. To illustrate the application of this analysis tool, the 14-node IEEE test system is used to

generate the PV curves. The code presented allows any modification throughout its script file and therefore it can be used for future power system studies and research.
The formulation of the load flow problem is firstly presented to obtain the PV curves, there are some issues that need to be taken into account in the algorithm oriented to the solution of load flows such as: mathematical formulation of the load flow, its adaptation to the Newton-Rhapson method and the implementation of the continuation theory to the analysis of load flows. It is also presented the necessary programming issues to the development of the script that plots the PV curves, the recommendations that are needed in the creation, manipulation and operation of sparse matrices, the use of vector operations, triangular decompositions techniques (that used in the solution of the set on linear equations) and finally the reading of ascii files and Graphical User Interface (GUI) development are also given.

1.1 Antecedents

Nowadays there are commercial programs which have been approved and used for the electric utilities in the analysis of electric power systems. Simulation programs as the Power World Simulator (PWS) (PowerWorld Corporation, 2010) and PSS (Siemens, 2005) are some of the most popular in the control and planning of a power system, and some of them are adopted by universities, e.g. PWS, because of its elegant interface and easy usage. Most of them have friendly user interfaces. On the other hand, a bachelor or graduate student, who want to reproduce or test new problem formulations to the solution of power system problems, need a simulation tool suitable for the generation of prototype programs. The code reutilization is important for integrating in a modular form, new functions required for the power system analysis (Milano, 2010). Commercial programs does not fulfill these requirements and therefore a search for alternatives is usually carry out, such as a new programming language or for the scientific language MATLAB. It is possible to find open source projects in several websites, which are usually named as "Toolbox" by their authors, and they cover a vast diversity of topics as: load flows, transient stability analysis, nodal analysis, and electromagnetic transients. Some of most relevant projects and its authors are: PSAT by Federico Milano (Milano, 2006), MatPower by Zimmerman, Carlos E. Murillo-Sánchez and Deqiang Gan (Zimmerman et al., 2011), PST by Graham Rogers, Joe H. Chow and Luigi Vanfretti (Graham et al., 2009) and MatEMTP by Mahseredjian, J. Alvarado and Fernando L. (Mahseredjian et al., 1997).
Similar projects have been developed at the Instituto Tecnológico de la Laguna (ITL) and they have been the basis for several MSc theses which have been integrated into the power system program PTL (figure 1). These projects have made possible the incorporation of new applications making a more flexible and robust program for the steady-state analysis of an electric power system.

2. Conceptual design of the PTL simulator

In spite of the foundation of the PTL program, i.e. being an integration of several graduate projects at the ITL, its design offers an interface which permits an intuitive user interaction and at the same time it has a dynamic performance which is able to solve load flow problems for any electric network regardless the node number. It offers the feature of showing the information graphically and numerically and besides it generates a report of the activities performed and exports files with data for making stability studies. The above PTL features make it a simulation program suitable for investigations because it allows

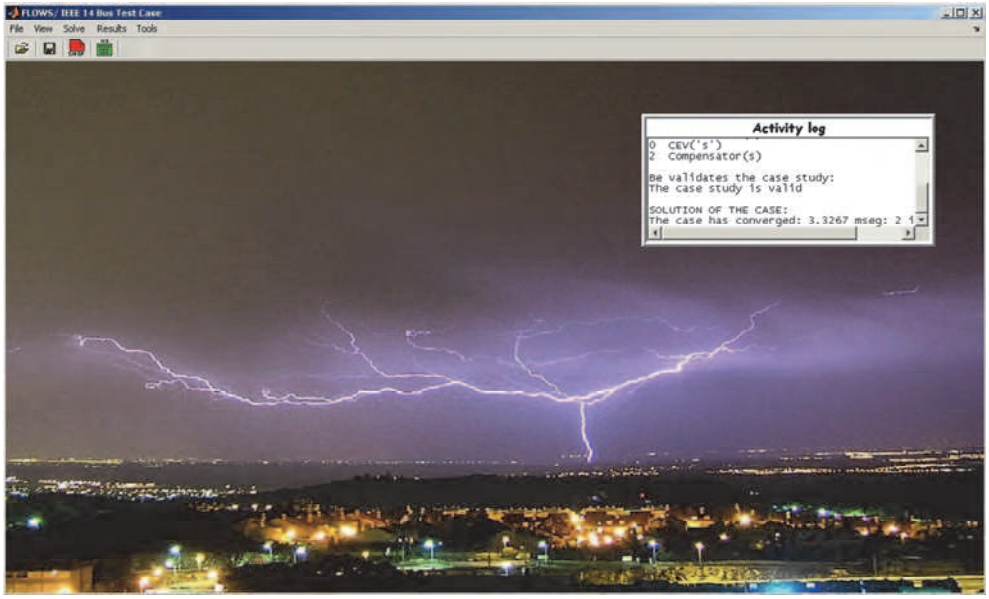

Fig. 1. GUI of the PTL program.

integrating solution algorithms for economic dispatch, calculation and plotting of PV curves, testing of methods with distributed slacks, static var compensator (SVC) models, transmission lines, generators, etc. It is also an important tool in power system research, making the PTL more complete for the analysis or studies oriented to the operation and control of electric power systems.

2.1 Data input details

As any other simulation program (commercial or free), the PTL requires of information data as input, it is needed for the analysis process. The information can be given as a data file that contains basic information to generate the base study case: the base power of the system, nodal information (number of nodes, voltages, load powers and generation power), machine limits, system branches, SVC information (if applies). The simulation program PTL can handle two file extensions: cdf (standardized IEEE format) and ptl (proposed PTL format).

2.2 Simulator description

The input information for carrying out the search of the solution process, as the related data to the problem results of the load flow problem (for generating the base case) are stored in defined data structures (e.g. Dat_Vn, Dat_Gen, Dat_Lin, Dat_Xtr). These structures allow easily the data extraction, by naming each field in such a way that the programming becomes intuitive and each variable can be easily identified with the corresponding physical variable of the problem. An example with two structures used in the PTL program is shown in Table 1.
The MATLAB structures are composed of non-primitive variable types that allow storing different data types in a hierarchical way with the same entity (García et al., 2005). They are formed by data containers called fields, which can be declared by defining the structure name and the desired field considering its value, e.g. Dat_Vn.Amp=1.02.

Structure	Field	Description
Dat_Vn	Amp	Voltage magnitude
	Ang	Phase angle of voltages
Dat_Gen	NumNG	Generation and load nodes
	Nslack	Slack node
	V_Rem_bus	Voltage information
	Pgen	Generated active power
	Qgen	Generated reactive power
	Pmin	Minimum limit of generator active power
	Pmax	Maximum limit of generator active power
	Qmin	Minimum limit of generator reactive power
	Qmax	Maximum limit of generator reactive power

Table 1. Example of the information handled in the PTL.

2.3 Output information

The PTL program displays the results obtained from the load flow execution in a boxlist (uicontrol MATLAB) with a defined format: nodal information, power flows in branches and generators. It gives the option of printing a report in Word format with the same information. In addition, it has the option of generating a file with the extension f2s which is oriented for stability studies and it contains all necessary information for the initialization of the state machine variables by using the results of the current power flow solution.

The definition of the conceptual simulation program PTL is presented as a recommendation by taking into account the three basic points that a simulation program must include: to be completely functional, to be general for any study case and to facilitate its maintenance; in other words it must allow the incorporation of new functions for the solution of new studies, such that it allows its free modification as easy as possible.

3. Load flows

In a practical problem, the knowledge of the operating conditions of an electric power system is always needed; that is, the knowledge of the nodal voltage levels in steady-state under loaded and generating conditions and the availability of its transmission elements are required to evaluate the system reliability. Many studies focused to the electric power systems start from the load flow solution which is known as "base case", and in some cases, these studies are used to initialize the state variables of dynamic elements of a network (generators, motors, SVC, etc) to carry out dynamic and transient stability studies. Another study of interest, that it also requires starting from a base case, is the analysis of the power system security that will be discussed in next sections.

The mathematical equations used to solve this problem are known as power flow equations, or network equations. In its more basic form, these equations are derived considering the transmission network with lumped parameters under lineal and balanced conditions, similarly as the known operating conditions in all nodes of the system (Arrillaga, 2001).

3.1 Power flow equations

An electric power system is formed with elements that can be represented for its equivalent circuit RLC, and with components as load and generating units which cannot be represented as basic elements of an electrical network, they are represented as nonlinear elements. However, the analysis of an electrical power system starts with the formulation of a referenced nodal system and it describes the relationship between the electrical variables (voltages and currents) as it is stated by the second Kirchhoff´s law or nodal law.

$$\mathbf{I}_{BUS} = \mathbf{Y}_{BUS} \cdot \mathbf{V}_{BUS} \tag{1}$$

where $\mathbf{I}_{\mathrm{BUS}}$ is a n×1 vector whose components are the electrical net current injections in the n network nodes, $\mathbf{V}_{\mathrm{BUS}}$ is a n×1 vector with the nodal voltages measured with respect to the referenced node and $\mathbf{Y}_{\mathrm{BUS}}$ is the n× n nodal admittance matrix of the electrical network; it has the properties of being symmetric and squared, and it describes the network topology.

In a real power system, the injected currents to the network nodes are unknown; what it is commonly known is the net injected power S_k. Conceptually, S_k is the net complex power injected to the k-th node of the electrical network, and it is determined by the product of voltage (V_k) and the current conjugate (I_k^*), where V_k and I_k are the voltages and nodal currents at the node k, that is, the k-th elements of vectors $\mathbf{V}_{BUS}$ y $\mathbf{I}_{BUS}$ in (1). Once the I_k is calculated using (1), the net complex power S_k can be expressed as:

$$S_k = V_k \cdot I_k^* = \mathrm{V}_k \cdot \left(\sum_{m=1}^{n} \mathrm{Y}_{k,m} \mathrm{V}_m \right)^*, \text{ for } k\text{= 1,2,…..n} \tag{2}$$

where $\mathrm{Y}_{k,m}$ is the element (k,m) of $\mathbf{Y}_{BUS}$ matrix in (1). S_k can also be represented for its real and imaginary components such as it is shown in the following expression:

$$S_k = P_k + jQ_k \text{ , for } k\text{= 1,2,…..n} \tag{3}$$

where P_k and Q_k are the net active and reactive power injected at node k of the system, respectively. They are defined as:

$$P_k = P_k^{Gen} - P_k^{Load} \tag{4}$$

$$Q_k = Q_k^{Gen} - Q_k^{Load} \tag{5}$$

where the variables P_k^{Gen} and Q_k^{Gen} represent the active and reactive powers respectively. They are injected at node k for a generator and the variables P_k^{Load} and Q_k^{Load} represent the active and reactive Powers, respectively of a load connected to the same node.

By representing the nodal voltages in polar form, we have:

$$V_k = V_k e^{j\theta_k} = V_k \left(\cos\theta_k + j \sin\theta_k \right) \tag{6}$$

and each element of the admittance matrix $\mathbf{Y}_{BUS}$ as,

$$Y_{km} = G_{km} + jB_{km} \tag{7}$$

Using the above expressions in (2), it results,

$$S_k = V_k e^{j\theta_k} \cdot \left(\sum_{m=1}^{n} (G_{km} + jB_{km}) V_m e^{j\theta_m} \right)^* = V_k \cdot \left(\sum_{m=1}^{n} V_m (G_{km} + jB_{km}) (cos\theta_{km} + j \sin\theta_{km}) \right)^* \quad \text{for } k\text{= 1,2,…..}n \tag{8}$$

where $\theta_{km} = \theta_k - \theta_m$. By separating the real and imaginary parts as it is suggested in (3), it is obtained the following,

$$P_k = V_k^2 G_{kk} + V_k \sum_{\substack{m=1 \\ m \neq n}}^{n} V_m \left(G_{km} Cos(\theta_k - \theta_m) + B_{km} Sin(\theta_k - \theta_m) \right) \text{ for } k\text{= 1,2,…..}n \tag{9}$$

$$Q_k = -V_k^2 B_{km} + V_k \sum_{\substack{m=1 \\ m \neq n}}^{n} V_m \left(G_{km} Cos(\theta_k - \theta_m) - B_{km} Cos(\theta_k - \theta_m) \right), \text{ for } k\text{= 1,2,…..}n \tag{10}$$

The equations (9) and (10) are commonly known as Power flow equations and they are needed for solving the load flow problem (Arrillaga, 2001). By analyzing these equations it can be clearly seen that each system node *k* is characterized for four variables: active power, reactive power, voltage magnitude and angle. Hence it is necessary to specify two of them and consider the remaining two as state variables to find with the solution of both equations.

3.2 Bus types in load flow studies

In an electrical power network, by considering its load flow equations, four variables are defined at each node, the active and reactive powers injected at node P_k and Q_k , and the magnitude and phase voltage at the node V_k y θ_k . The latter two variables determine the total electrical state of the network, then, the objective of the load flow problem consists in determining these variables at each node. The variables can be classified in controlled variables, that is, its values can be specified and state variables to be calculated with the solution of the load flow problem. The controlled or specified variables are determined by taking into account the node nature, i.e. in a generator node, the active power can be controlled by the turbine speed governor, and the voltage magnitude of the generator node can be controlled by the automatic voltage regulator (AVR). In a load node, the active and reactive power can be specified because its values can be obtained from load demand studies. Therefore, the system nodes can be classified as follows:

- *Generator node PV*: It is any node where a generator is connected; the magnitude voltage and generated active power can be controlled or specified, while the voltage phase angle and the reactive power are the unknown state variables (Arrillaga, 2001).
- *Load node PQ*: It is any node where a system load is connected; the active and reactive consumed powers are known or specified, while the voltage magnitude and its phase angle are the unknown state variables to be calculated (Arrillaga, 2001).
- *Slack node* (Compensator): In a power system at least one of the nodes has to be selected and labeled with this node type. It is a generating node where it cannot be specified the generated active power as in the PV node, because the transmission losses are not known beforehand and thus it cannot be established the balance of active power of the loads and

generators. Therefore this node compensates the unbalance between the active power between loads and generating units as specified in the PQ and PV nodes (Arrillaga, 2001).

3.3 Solution of the nonlinear equations by the Newton-Rhapson method

The nature of the load flow problem formulation requires the simultaneous solution of a set of nonlinear equations; therefore it is necessary to apply a numerical method that guarantees a unique solution. There are methods as the Gauss-Seidel and Newton-Rhapson, in the work presented here, the load flow problem is solved with the Newton-Raphson (NR) (Arrillga, 2001).

The NR method is robust and has a fast convergence to the solution. The method has been applied to the solution of nonlinear equations that can be defined as,

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \tag{11}$$

where $\mathbf{f}(\mathbf{x})$ is a n×1 vector that contains the n equations to be solved $f_i(\mathbf{x}) = 0$, i=1,n,

$$\mathbf{f}(\mathbf{x}) = \left[f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_n(\mathbf{x}) \right]^T \tag{12}$$

and $\mathbf{x}$ is a n×1 vector that contains the state variables, x_i, i=1,n,

$$\mathbf{x} = \left[x_1, x_2, \ldots, x_n \right]^T \tag{13}$$

The numerical methods used to solve (11) are focused to determine a recursive formula $\mathbf{x}^{k+1} = \mathbf{x}^k + \Delta\mathbf{x}$. The solution algorithm is based in the iterative application of the above formula starting from an initial estimate $\mathbf{x}^0$, until a convergence criterion is achieved $\max(\Delta\mathbf{x}) < \varepsilon$, where ε is a small numerical value, and the vector $\mathbf{x}^k$ is an approximation to the k solution $\mathbf{x}^*$ of (11).

The Newton-Raphson method can be easily explained when it is applied to an equation of a single state variable (Arrillaga, 2001). A geometrical illustration of this problem is shown in figure 2.

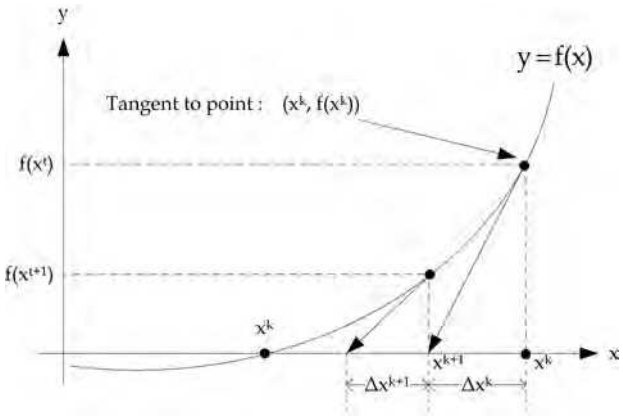

Fig. 2. Geometric interpretation of the NR algorithm.

From figure 2, it can be established that $\Delta x^k = x^{k+1} - x^k$

$$d\frac{f(x)}{dx}\bigg|_{x^k} = -\frac{f(x)}{\Delta x^k} \tag{14}$$

where,

$$\Delta x^k = -\left(f'(x_k)\right)^{-1} \cdot f(x_k) \tag{15}$$

As it can be seen in figure 2, the successive application of this correction $\Delta x^k, \Delta x^{k+1}, \Delta x^{k+2}$... leads to the solution $\mathbf{x}^*$ as nearer as desired.
To derive the recursive formula to be employed in the solution of the set of equations, and by expressing the equation (15) in matrix notation, it is obtained,

$$\Delta \mathbf{x}^k = -\left(\mathbf{f}'(\mathbf{x}_k)\right)^{-1} \cdot \mathbf{f}(\mathbf{x}_k) = -\left[\mathbf{J}(\mathbf{x}_k)\right]^{-1} \cdot \mathbf{f}(\mathbf{x}_k) \tag{16}$$

where

$$\mathbf{J}(\mathbf{x}_k) = \frac{\partial \mathbf{f}(\mathbf{x}_k)}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \frac{\partial f_1(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f_1(\mathbf{x})}{\partial x_n} \\ \frac{\partial f_2(\mathbf{x})}{\partial x_1} & \frac{\partial f_2(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f_2(\mathbf{x})}{\partial x_n} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial f_n(\mathbf{x})}{\partial x_1} & \frac{\partial f_n(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f_n(\mathbf{x})}{\partial x_n} \end{bmatrix} \tag{17}$$

and the recursive formula is given by,

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \Delta \mathbf{x} \tag{18}$$

3.4 Application of the NR to the power flow problem

The equations to be solved in the load flow problem, as it was explained in section 3.1, are here rewritten,

$$P_k^{esp} - \left[V_k^2 G_{kk} + V_k \sum_{\substack{m=1 \\ m \neq n}}^{n} V_m \left(G_{km} Cos(\theta_k - \theta_m) + B_{km} Sin(\theta_k - \theta_m) \right) \right] = 0 \tag{4}$$

$$Q_k^{esp} - \left[-V_k^2 B_{km} + V_k \sum_{\substack{m=1 \\ m \neq n}}^{n} V_m \left(G_{km} Cos(\theta_k - \theta_m) - B_{km} Cos(\theta_k - \theta_m) \right) \right] = 0 \tag{5}$$

for k= 1,2,…..n

The 2*n* equations to be solved are represented by (4) and (5). However, for all generator nodes, equation (5) can be omitted and for the slack node the equations (4) and (5) (Arrillaga, 2001). The resulting set of equations is consistent because for the neglected equations, its corresponding state variables P^{esp}, Q^{esp} are also omitted from them. All this operation gives as a result a set of equations to solve, where its state variables $\mathbf{x}$ only contain magnitudes of nodal voltages and its corresponding phase angles which are denoted by $\mathbf{V}$ y $\boldsymbol{\theta}$, which simplifies considerably a guaranteed convergence of the numerical method. Therefore, the set of equation can be represented in vector form as,

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} \Delta\mathbf{P}(\mathbf{x}) \\ \Delta\mathbf{Q}(\mathbf{x}) \end{bmatrix} = \mathbf{0} \tag{19}$$

where $\Delta\mathbf{P}(\mathbf{x})$ represents the equation (4) for PV and PQ nodes, $\Delta\mathbf{Q}(\mathbf{x})$ represents the equation (5) for PQ nodes and $\mathbf{x}$ denotes the state variables $\mathbf{V}$ and $\boldsymbol{\theta}$, which are represented in vector notation as:

$$\mathbf{x} = \begin{bmatrix} \boldsymbol{\theta} \\ \mathbf{V} \end{bmatrix} \tag{20}$$

where:
$\mathbf{V}$ = *nc*x1 vector.
$\boldsymbol{\theta}$ = *nc*+*n*-1 vector.
nc = Number of PQ nodes.
n = Total number of nodes.
Considering equation (16), its matrix equation is obtained and it defines the solution of the load flow problem:

$$\underbrace{\begin{bmatrix} \Delta\boldsymbol{\theta} \\ \Delta\mathbf{V} \end{bmatrix}}_{\mathbf{x}^k} = -\underbrace{\begin{bmatrix} \frac{\partial\Delta\mathbf{P}}{\partial\boldsymbol{\theta}} & \frac{\partial\Delta\mathbf{P}}{\partial\mathbf{V}} \\ \frac{\partial\Delta\mathbf{Q}}{\partial\boldsymbol{\theta}} & \frac{\partial\Delta\mathbf{Q}}{\partial\mathbf{V}} \end{bmatrix}^{-1}}_{\mathbf{J}(\mathbf{x}^k)} \underbrace{\begin{bmatrix} \Delta\mathbf{P} \\ \Delta\mathbf{Q} \end{bmatrix}}_{\mathbf{f}(\mathbf{x}^k)} \tag{21}$$

By applying the recursive equation (18), the state variables ($\mathbf{V}$ y $\boldsymbol{\theta}$) are updated every iteration until the convergence criterion is achieved $\max(|\Delta\mathbf{P}(\mathbf{x})|) \le \varepsilon$ or $\max(|\Delta\mathbf{Q}(\mathbf{x})|) \le \varepsilon$ for a small ε or until the iteration number exceeds the maximum number previously defined which in this indicates convergence problems.
The Jacobian elements in (21) are,

$$\frac{\partial\Delta\mathbf{P}}{\partial\boldsymbol{\theta}} \tag{22}$$

They are calculated using (19), however, by taking into account that the specified powers P_k^{esp} are constants, we obtain,

$$\frac{\partial\Delta\mathbf{P}}{\partial\boldsymbol{\theta}} = -\frac{\partial\mathbf{P}}{\partial\boldsymbol{\theta}} \tag{23}$$

Therefore, equation (21) can be expressed as,

$$\begin{bmatrix} \Delta \mathbf{P} \\ \Delta \mathbf{Q} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial \mathbf{P}}{\partial \boldsymbol{\theta}} & \dfrac{\partial \mathbf{P}}{\partial \mathbf{V}} \\ \dfrac{\partial \mathbf{Q}}{\partial \boldsymbol{\theta}} & \dfrac{\partial \mathbf{Q}}{\partial \mathbf{V}} \end{bmatrix} \cdot \begin{matrix} \Delta \boldsymbol{\theta} \\ \Delta \mathbf{V} \end{matrix} \tag{24}$$

where the Jacobian elements are calculated using equations (9) and (10), provided equation (21) is normalized.
To simplify the calculation of the Jacobian elements of (24), it can be reformulated as:

$$\begin{bmatrix} \Delta \mathbf{P} \\ \Delta \mathbf{Q} \end{bmatrix} = \begin{bmatrix} \mathbf{H} & \mathbf{N} \\ \mathbf{J} & \mathbf{L} \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{\theta} \\ \dfrac{\Delta \mathbf{V}}{\mathbf{V}} \end{bmatrix} \tag{25}$$

where the quotient of each element with its corresponding element V allows that some elements of the Jacobian matrix can be expressed similarly (Arrillaga, 2001). The Jacobian can be formed by defining four submatrixes denoted as **H**, **N**, **J** and **L** which are defined as,
If $k \neq m$ (off diagonal elements):

$$H_{km} = \frac{\partial P_k}{\partial \theta_m} = V_k V_m \left(G_{km} \sin \theta_{km} - B_{km} \cos \theta_{km} \right) \tag{26}$$

$$N_{km} = V_k \frac{\partial P_k}{\partial V_m} = V_k V_m \left(G_{km} \cos \theta_{km} + B_{km} \sin \theta_{km} \right) \tag{27}$$

$$J_{km} = \frac{\partial Q_k}{\partial \theta_m} = -V_k V_m \left(G_{km} \cos \theta_{km} + B_{km} \sin \theta_{km} \right) \tag{28}$$

$$L_{km} = V_k \frac{\partial P_k}{\partial V_m} = V_k V_m \left(G_{km} \sin \theta_{km} - B_{km} \cos \theta_{km} \right) \tag{29}$$

If $k=m$ (main diagonal elements):

$$H_{kk} = \frac{\partial P_k}{\partial \theta_k} = -Q_k - B_{kk} V_k^2 \tag{30}$$

$$N_{kk} = V_k \frac{\partial P_k}{\partial V_k} = P_k + G_{kk} V_k^2 \tag{31}$$

$$J_{kk} = \frac{\partial Q_k}{\partial \theta_m} = P_k - G_{kk} V_k^2 \tag{32}$$

$$L_{kk} = V_k \frac{\partial P_k}{\partial V_m} = Q_k - B_{kk} V_k^2 \tag{33}$$

It is important to consider that subscripts *k* and *m* are different, $L_{km} = H_{km}$ and $J_{km} = -N_{km}$, on the contrary, to the principal diagonal elements. Nevertheless, there is a relationship when the following equation is solved:

$$\begin{bmatrix} V_1 & 0 & \cdots & 0 \\ 0 & V_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & V_n \end{bmatrix} \left(\left(\begin{bmatrix} Y_{11} & Y_{12} & \cdots & Y_{1n} \\ Y_{21} & Y_{22} & \cdots & Y_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ Y_{n1} & Y_{n1} & \cdots & Y_{nn} \end{bmatrix} \begin{bmatrix} V_1 & 0 & \cdots & 0 \\ 0 & V_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & V_n \end{bmatrix} \right)^* \right) \tag{34}$$

By simple inspection of (34), we can see that the operation for $-N_{km}$ and J_{km} results to be the real part of (34), while H_{km} and L_{km} are the imaginary component of it. The variant consists that all elements of the principal diagonal are calculated after (34) has been solved by subtracting or adding the corresponding P_k or Q_k as it can be seen in (30)-(33).
The Jacobian matrix has the characteristic of being sparse and squared with an order of the length of vector **X**, where its sub-matrixes have the following dimensions:
H: *n*-1 x *n*-1 matrix.
N: *n*-1 x *nc* matrix.
J: *nc* x *n*-1 matrix.
L: *nc* x *nc* matrix.

4. Voltage stability and collapse

The terms of voltage stability and collapse are closely related to each other in topics of operation and control of power systems.

4.1 Voltage stability

According to the IEEE, the voltage stability is defined as the capacity of a power system to maintain in all nodes acceptable voltage levels under normal conditions after a system disturbance for a given initial condition (Kundur, 1994). This definition gives us an idea of the robustness of a power system which is measured by its capability of keep the equilibrium between the demanded load and the generated power. The system can be in an unstable condition under a disturbance, increase of demanded load and changes in the topology of the network, causing an incontrollable voltage decrement (Kundur, 1994). The unstable condition can be originated for the operating limits of the power system components (Venkataramana, 2007), such as:
Generators: They represent the supply of reactive power enough to keep the power system in stable conditions by keeping the standardized voltage levels of normal operation. However, the generation of machines is limited by its capability curve that gives the constraints of the reactive power output due to the field winding current limitation.
Transmission lines: They are another important constraint to the voltage stability, and they also limit the maximum power that be transported and it is defined the thermal limits.
Loads: They represent the third elements that have influence on the stability voltage; they are classified in two categories: static and dynamic loads and they have an effect on the voltage profiles under excessive reactive power generation.

4.2 Voltage collapse

The voltage collapse is a phenomenon that might be present in a highly loaded electric power system. This can be present in the form of event sequence together with the voltage instability that may lead to a blackout or to voltage levels below the operating limits for a significant part of the power system (Kundur, 1994). Due to the nonlinear nature of the electrical network, as the phenomenon related to the power system, it is necessary to employ nonlinear techniques for the analysis of the voltage collapse (Venkataramana, 2007) and find out a solution to avoid it.

There many disturbances which contribute to the voltage collapse:

- Load increment.
- To reach the reactive power limits in generators, synchronous condensers or SVC.
- The operation of TAP changers in transformers.
- The tripping of transmission lines, transformers and generators.

Most of these changes have a significant effect in the production, consumption and transmission of reactive power. Because of this, it is suggested control actions by using compensator elements as capacitor banks, blocking of tap changers, new generation dispatch, secondary voltage regulation and load sectioning (Kundur, 1994).

4.3 Analysis methods for the voltage stability

Some of the tools used for the analysis of stability voltage are the methods based on dynamic analysis and those based in static analysis.

Dynamic Analysis. They consist in the numerical solution (simulation) of the set of differential and algebraic equations that model the power system (Kundur, 1994), this is similar as transients; however, this kind of simulations need considerable amount of computing resources and hence the solution time is large and they do not give information about the sensibility and stability degree.

Static analysis. They consist in the solution of the set of algebraic equations that represent the system in steady state (Kundur, 1994), with the aim of evaluating the feasibility of the equilibrium point represented by the operating conditions of the system and to find the critical voltage value. The advantage with respect to the dynamic analysis techniques is that it gives valuable information about the nature of the problem and helps to identify the key factors for the instability problem. The plotting of the PV curve helps to the analysis of the voltage stability limits of a power system under a scenario with load increments and with the presence of a disturbance such as the loss of generation or the loss of a transmission line.

4.4 PV curves

The PV curves represent the voltage variation with respect to the variation of load reactive power. This curve is produced by a series of load flow solutions for different load levels uniformly distributed, by keeping constant the power factor. The generated active power is proportionally incremented to the generator rating or to the participating factors which are defined by the user. The P and Q components of each load can or cannot be dependant of the bus voltage accordingly to the load model selected. The determination of the critical point for a given load increment is very important because it can lead to the voltage collapse of the system. These characteristics are illustrated in figure 3.

Some authors (Yamura et al, 1998 & Ogrady et al, 1999) have proposed voltage stability indexes which are based in some kind of analysis of load flows with the aim to evaluate the

stability voltage limits. However, the Jacobian used in the load flows, when the Newton-Raphson is employed, becomes singular at the critical point, besides the load flow solutions at the points near to the critical region tend to diverge (Kundur, 1994). These disadvantages are avoided by using the method of continuation load flows (Venkataramana et al, 1992).

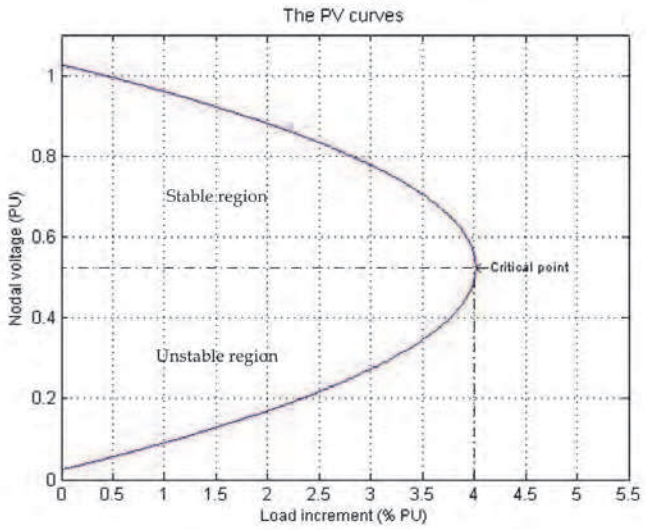

Fig. 3. PV curve

4.5 Application of the continuation method to the power flow problem

The continuous load flow procedure is based in a reformulation of the equations of the load flow problem and the application of the continuation technique with a local parameterization which has shown to be efficient in the trajectory plotting of PV curves.

The purpose of continuous load flows is to find a set of load flow solutions in a scenario where the load is continuously changing, starting from a base case until the critical point. Thereafter, the continuous load flows had been applied to understand and evaluate the problem of voltage stability and those areas that are likely to the voltage collapse. Besides, they have also been applied in other related problems like the evaluation of power transfer limits between regions.

The general principle of continuous load flows employs a predictor-corrector scheme to find a trajectory of solutions for the set of load flow equations (4) and (5) which are reformulated to include the load parameter λ.

$$\Delta P_i = \lambda \left(P_{Gi} - P_{Li} \right) - P_i = \lambda P_i^{esp} - P_i = 0 \tag{35}$$

$$\Delta Q_i = \lambda \left(Q_{Gi} - Q_{Li} \right) - Q_i = \lambda Q_i^{esp} - Q_i = 0 \tag{36}$$

$$1 \le \lambda \le \lambda_{crític} \tag{37}$$

The process is started from a known solution and a predictor vector which is tangent to the corrected solutions is used to estimate the future solutions with different values of the load parameter. The estimation is corrected using the same technique of the Newton-Rhapson employed in the conventional load flow with a new added parameter:

$$\mathbf{f}(\boldsymbol{\theta}, \mathbf{V}, \lambda) = \mathbf{0} \tag{38}$$

The parameterization plays an important role in the elimination of the Jacobian non-singularity.

4.5.1 Prediction of the new solution

Once the base solution has been found for λ=0, it is required to predict the next solution taking into consideration the appropriate step size and the direction of the tangent to the trajectory solution. The first task in this process consists in calculating the tangent vector, which is determined taking the first derivative of the reformulated flow equations (38).

$$d\mathbf{f}(\boldsymbol{\theta}, \mathbf{V}, \lambda) = \mathbf{f}_{\theta} d\boldsymbol{\theta} + \mathbf{f}_{V} d\mathbf{V} + \mathbf{f}_{\lambda} d\lambda = 0 \tag{39}$$

where **F** is the vector [**ΔP**, **ΔQ**, 0] that is augmented in one row; in a factorized form, the equation is expressed as,

$$\begin{bmatrix} \mathbf{f}_{\boldsymbol{\theta}} & \mathbf{f}_{\mathbf{F}} & \mathbf{f}_{\lambda} \end{bmatrix} \begin{bmatrix} d\boldsymbol{\theta} \\ d\mathbf{V} \\ d\lambda \end{bmatrix} = 0 \tag{40}$$

The left hand side of the equation is a matrix of partial derivatives that multiplies the tangent vector form with differential elements. The matrix of partial derivatives is known as the Jacobian of the conventional load flow problem that is augmented by the column $\mathbf{F}_{\lambda}$, which can be obtained by taking the partial derivatives with respect to λ (35) and (36), which gives:

$$\begin{bmatrix} \Delta P \\ \Delta Q \end{bmatrix} = \begin{bmatrix} \mathbf{H} & \mathbf{N} & -\mathbf{P}^{esp} \\ \mathbf{J} & \mathbf{L} & -\mathbf{Q}^{esp} \end{bmatrix} \begin{bmatrix} d\boldsymbol{\theta} \\ d\mathbf{V} \\ \hline \mathbf{V} \\ d\lambda \end{bmatrix} \tag{41}$$

Due to the nature of (41) which is a set of $nc+n-1$ equations with $nc+n$ unknowns and by adding λ to the load flow equations, it is not possible to find a unique and nontrivial solution of the tangent vector; consequently an additional equation is needed.

This problem is solved by selecting a magnitude different from zero for one of the components of the tangent vector. In other words, if the tangent vector is denoted by:

$$\mathbf{t} = \begin{bmatrix} d\boldsymbol{\theta} \\ d\mathbf{V} \\ d\lambda \end{bmatrix} = 0, \quad t_k = \pm 1 \tag{42}$$

which leads to:

$$\begin{bmatrix} \mathbf{H} & \mathbf{N} & -\mathbf{P}^{esp} \\ \mathbf{J} & \mathbf{L} & -\mathbf{Q}^{esp} \\ & \mathbf{e}_k & \end{bmatrix} \cdot \mathbf{t} = \begin{bmatrix} \mathbf{0} \\ \pm 1 \end{bmatrix} \tag{43}$$

where $\mathbf{e}_k$ is a vector of dimension m+1 with all elements equal to zero but the *k*-th one, which is equal to 1. If the index *k* is correctly selected, $t_k = \pm 1$ impose a none zero norm to the tangent vector and it guarantees that the augmented Jacobian will be nonsingular at the critical point (Eheinboldt et al, 1986). The usage of +1 or -1 depends on how the *k*-th state variable is changing during the trajectory solution which is being plotted. In a next section of this chapter, a method to select *k* will be presented. Once the tangent vector has found the solution of (43), the prediction is carried out as follows:

$$\begin{bmatrix} \boldsymbol{\theta}^* \\ \mathbf{V}^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} \boldsymbol{\theta} \\ \mathbf{V} \\ \lambda \end{bmatrix} + \sigma \begin{bmatrix} d\boldsymbol{\theta} \\ d\mathbf{V} \\ d\lambda \end{bmatrix} \tag{44}$$

where "*" denotes the prediction for a future value of λ (the load parameter) and σ is a scalar that defines the step size. One inconvenient in the control of the step size consists in its dependence of the normalized tangent vector (Alves et al, 2000),

$$\sigma = \frac{\sigma_0}{\|\mathbf{t}\|} \tag{45}$$

where $|t_k|$ is the Euclidian norm of the tangent vector and σ_0 is a predefined scalar. The process efficiency depends on making a good selection of σ_0, which its value is system dependant.

4.5.2 Parameterization and corrector

After the prediction has been made, it is necessary to correct the approximate solution. Every continuation technique has a particular parameterization that gives a way to identify the solution along the plotting trajectory. The scheme here presented is referred as a local parameterization. In this scheme, the original set of equations is augmented with one extra equation, which has a meaning of specifying the value of a single state variable. In the case of the reformulated equations, this has a meaning of giving a unity magnitude to each nodal voltage, the phase angle of the nodal voltage or the load parameter *λ*. The new set of equations involves the new definition of state variables as:

$$\mathbf{x} = \begin{bmatrix} \boldsymbol{\theta} \\ \mathbf{V} \\ \lambda \end{bmatrix} \tag{46}$$

where,

$$x_k = \eta \tag{47}$$

where $x_k \, \varepsilon \, \mathbf{x}$ and η represent the appropriate *k*-th element of **x**. Therefore, the new set of equations that substitute (38) is given by,

$$\begin{bmatrix} \mathbf{F}(\mathbf{x}) \\ x_k - \eta \end{bmatrix} = 0 \tag{48}$$

After an appropriate index *k* has been selected and its corresponding value of η, the load flow is solved with the slightly modified Newton Raphson method (48). In other words, the *k* index used in the corrector is the same as that used in the predictor and η is equal to the obtained x_k from the corrector (44), thus the variable x_k is the continuation parameter.
The application of the Newton-Raphson to (38) results in,

$$\begin{bmatrix} H & N & -\mathbf{P}^{esp} \\ M & L & -\mathbf{Q}^{esp} \\ & \mathbf{e}_k & \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{\theta} \\ \Delta\mathbf{V} \\ \Delta\lambda \end{bmatrix} = \begin{bmatrix} \Delta\mathbf{P} \\ \Delta\mathbf{Q} \\ 0 \end{bmatrix} \tag{49}$$

where $\mathbf{e}_k$ is the same vector used in (43), and the elements $\Delta\mathbf{P}$ and $\Delta\mathbf{Q}$ are calculated from (35) and (36). Once the x_k is specified in (48), the values of the other variables are dependant on it and they are solved by the iterative application of (49).

4.5.3 Selection of the continuation parameter

The most appropriate selection corresponds to that state variable with the component of the tangent vector with the largest rate of change relative to the given solution. Typically, the load parameter is the best starting selection, i.e. λ=1. This is true if the starting base case is characterized for a light or normal load; in such conditions, the magnitudes of the nodal voltages and angles keep almost constants with load changes. On the other hand, when the load parameter has been increased for a given number of continuation steps, the solution trajectory is approximated to the critical point, and the voltage magnitudes and angles probably will have more significant changes. At this point, λ has had a poor selection as compared with other state variables. Then, once the first step selection has been made, the following verification must be made:

$$x_k \leftarrow \max\left\{ \frac{t_1}{x_1} \quad \frac{t_2}{x_2} \quad \cdots \quad \frac{t_{m-1}}{xm-1} \quad \frac{t_m}{x_m} \right\} \tag{50}$$

where *m* is equal to the state variables; including the load parameter and *k* corresponding to the maximum t/x component. When the continuation parameter is selected, the sign of the corresponding component of the tangent vector must be taken into account to assign +1 or -1 to t_k in (42) for the subsequent calculation of the tangent vector.

5. MATLAB resources for electrical networks

The reason why MATLAB is frequently chosen for the development of academic or research tools is because its huge amount of mathematical operations as those related to vectors and matrixes. In addition, it also has several specialized libraries (toolboxes) for more specific areas as control, optimization, symbolic mathematics, etc. In the area of power systems, it is

possible to point it out several advantages considered as key points for the development of a script program, which are discussed below.

5.1 Sparse matrix manipulation

The electrical networks are studied using nodal analysis, as it was presented in section 3.1. In this frame of reference, the network matrixes as Ybus or Jacobian (1) and (21) have a sparse structure, considering that a node in an electrical network is connected to about 2.4 nodes in average. To illustrate this issue, if we consider the creation of a squared matrix with order 20, i.e. the matrix has 200 elements which 256 are zeros and the remaining are different from zero that are denoted as nz, as it is shown in figure 4. The sparsity pattern is displayed by using the MATLAB function *spy*.

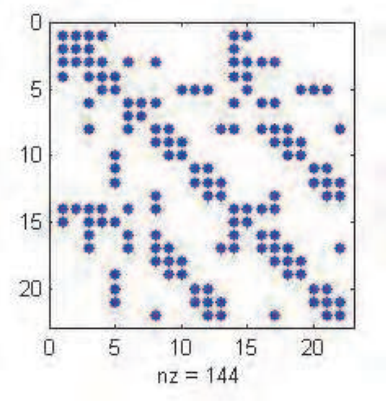

Fig. 4. An example of a sparse matrix.

The operation on this type of matrices with conventional computational methods leads to obtain prohibited calculation times (Gracia et al, 2005). Due to this reason, there have been adopted special techniques for deal with this type of matrixes to avoid the unnecessary usage of memory and to execute the calculation processes on the nonzero elements. It is important to point it out that this kind of matrixes is more related to a computation technique than to a mathematic concept.

MATLAB offers mechanisms and methods to create, manipulate and operate on this kind of matrixes. The sparse matrixes are created with the sparse function, which requires the specification of 5 arguments in the following order: 3 arrays to specify the position *i*, the position *j* – row and column- and the element values x that correspond to each position (i,j), and the two integer variables to determine the dimensions *mxn* of the matrix, for example:

>> A = sparse(i, j, x, m, n)

In MATLAB language, the indexing of full matrixes is equal to the sparse matrixes. This mechanism consists in pointing to a set of matrix elements through the use of two arrays that makes reference to each row and column of the matrix, e.g. **B = A[I, J]** or by simply pointing to each element o elements of the matrix to be modified, e.g. A**[I, J] = X.**

where **A** is a *mxn* spare matrix, I and J are the arrays that point to the rows and columns and **X** is the array that contains each element corresponding to each ordered pair (I_k, J_k). All mentioned arrays are composed for a number of *k* elements, such as $k < n$ and $k < m$.

MATLAB has a function called spdiags that is intended for the direct operation on any diagonal of the matrix, e.g. to make uniform changes on the elements of the diagonal "d" of matrix A:

>> A = spdiags(B, d, A)

A mechanism frequently used to form matrixes from other defined ones is called the *concatenation*. Even more, to form sparse matrixes by using other arrays of the same type but with specified dimensions in such a way that there is a consistency to gather into a single one, i.e. the concatenating is horizontal, the matrix rows must be equal to those of B matrix, e.g. C = [A B]. On the other hand, if the concatenating is vertical, then the columns of both matrices must be equal, e.g. C = [A; B].

5.2 The LU factorization for solving a set of equations

In electrical network applications, to find a solution of the algebraic system $\mathbf{Ax} = \mathbf{b}$ in an efficient way, one option is to use the triangular decomposition as the LU factorization technique.

The triangular factorization LU consists in decomposing a matrix (A) such that it can be represented as the product of two matrixes, one of them is a lower triangular (L) while the other is an upper triangular (U), $\mathbf{A} = \mathbf{L} \cdot \mathbf{U}$. This representation is commonly named explicit factorization LU; even though it is very related to the Gaussian elimination, the elements of L and U are directly calculated from the **A** elements, the principal advantage with respect to the Gaussian elimination (Gill et al., 1991) consists in obtaining the solution of an algebraic system for any **b** vector, if and only if the "**A**" matrix is not modified.

5.3 Sparse matrix ordering using AMD

The Approximate minimum degree permutation (AMD) is a set of subroutines for row and column permutation of a sparse matrix before executing the Cholesky factorization or for the LU factorization with a diagonal pivoting (Tim, 2004). The employment of this subroutine is made by using the function "amd" to the matrix to be permuted, e.g. P = amd(**A**).

5.4 Sparse matrix operations

Care must be taken with a several rules in MATLAB when operations are carried out that include full and sparse matrixes, for example, the operation eye(22) + speye(22) gives a full matrix.

5.5 Vector operations

An approach to develop the script programs in MATLAB to be executed faster consists in coding the algorithms with the use of vectors within the programs and avoiding the use of loops such as *for, while and do-while*. The vector operations are made by writing the symbol "." before the operation to be made, e.g. .+, .-, .*, ./. This discussion is illustrated by comparing two MATLAB script programs to solve the following operation:

$$\begin{bmatrix} C_1 \\ C_2 \\ \vdots \\ C_n \end{bmatrix} = \begin{bmatrix} A_1 \cdot B_1 \\ A_1 \cdot B_2 \\ \vdots \\ A_n \cdot B_n \end{bmatrix}$$

Script program with the loop command	Script program with vector operations.
for k = 1:1000 C(k,1) = k*k; end	A = 1:1000; B = 1:1000; C = A.*B;
Execution time: 0.002086 s	*Execution time*: 0.000045 s

It is evident that the use of the *for-loop* command to carry out the described operation gives a larger computation time than by using the vector operation **A**.***B**.

6. MATLAB application for plotting the PV curves

The program for plotting the PV curves is integrated with four specific tasks:

- Reading of input data
- Generation of the flow base case
- Calculation of points for the PV curves at each node.
- Graphical interface for editing the display of PV curves.

Afterwards, the complete application code is listed. In case the code is copied and pasted into a new m-file, this will have the complete application, i.e. there is no necessity to add new code lines.

```
function main()
%% **************    READING OF INPUT DATA    **************

% -> The filename and its path are obtained:
   [file, trayectoria] = uigetfile( ...
      { '*.cdf;','Type file (*.cdf)'; '*.cf', ...
      'IEEE Common Format (*.cf)'}, 'Select any load flow file');

   if file == 0 & trayectoria == 0
      return;
   end
   archivo = [trayectoria file];

% -> the code lines are stored in the following variable:
   DATA = textread(archivo,'%s','delimiter','\n','whitespace','');

% -> Reading of nodal data:
   X_b = DATA{1};
   Sbase = str2double(X_b(32:37)); % Base power (MVA): Sbase

   X_b = DATA{2};
   N_Bus = findstr(X_b,'ITEMS');
   N_Bus = str2double(fliplr(strtok(fliplr(X_b(1:N_Bus-1)))));
   Nnod  = N_Bus;  % Total number of buses
   N_Bus = 2+N_Bus;
   DN    = strvcat(DATA{3:N_Bus});
```

```
   No_B = str2num(DN(:,1:4));      % Number of buses: No_B
   Tipo = str2num(DN(:,25:26));    % Bus type: Tipo

   Nslack = find(Tipo == 3);   % Slack bus
   NumNG  = find(Tipo >= 2);   % Index of PV nodes
   NumNC  = find(Tipo == 0);   % Index of PQ nodes
   NumCG  = find(Tipo <= 2);   % Index of all nodes except slack

   Vabs = str2num(DN(:,28:33));    % Magnitude of nodal voltages.
   Tetan = str2num(DN(:,34:40));   % Phase angle of nodal voltages (degrees)
   Tetan = Tetan*pi/180;          %  Degrees to radians conversion

   Pload = str2num(DN(:,41:49));   % Demanded active power (MW)
   Qload = str2num(DN(:,50:59));   % Demanded reactive power (MVAr)

   Pgen = str2num(DN(:,60:67));    % Generated active power (MW)
   Qgen = str2num(DN(:,68:75));    % Generated reactive power (MVAr)
   Pgen = Pgen(NumNG);     % PV nodes are selected
   Qgen = Qgen(NumNG);     %PV nodes are selected
   Gc    = str2num(DN(:,107:114)); % Compensator conductance
   Bc    = str2num(DN(:,115:122)); % Compensator susceptance
   NumND = find(Bc~=0);    % Indexes of all compensating nodes.
   Bcomp = Bc(NumND);      % Susceptance of all compensating nodes  NumND

% -> Branch data is read:
   X_b = DATA{N_Bus+2};
   N_Bra = findstr(X_b,'ITEMS');
   N_Bra = str2double(fliplr(strtok(fliplr(X_b(1:N_Bra-1)))));
   N_Bra1 = N_Bus+3;
   N_Bra2 = N_Bus+N_Bra+2;
   XLN = strvcat(DATA{N_Bra1:N_Bra2});

   X_b    = str2num(XLN(:,19));
   NramL  = find(X_b == 0);    % Indexes of transmission lines (LT)
   NramT  = find(X_b >= 1);    % Indexes of transformers (Trafos)
   PLi    = str2num(XLN(NramL,1:4));   % Initial bus for the transmission lines
   QLi    = str2num(XLN(NramL,6:9));   % Final bus for the transmission lines
   PTr    = str2num(XLN(NramT,1:4));   % Initial bus for the transformers
   QTr    = str2num(XLN(NramT,6:9));   % Final bus for the transformers

   RbrL =  str2num(XLN(NramL,20:29));  % Resistance of transmission lines (pu)
   XbrL =  str2num(XLN(NramL,30:40));  % Reactance of transmission lines (pu)
   Blin = str2num(XLN(NramL,41:50))/2; % Susceptance of transmission lines (pu)

   RbrX =  str2num(XLN(NramT,20:29));  % Resistance of transmission lines (pu)
   XbrX =  str2num(XLN(NramT,30:40));  % Reactance of transmission lines (pu)
```

```
    TAP  = str2num(XLN(NramT,77:82));   % TAP changers of transformers

% -> Admittance matrix:

   Ylin =1./(RbrL+1j*XbrL);       % Series admítanse of transmission lines
   Yxtr =(1./(RbrX+1j*XbrX))./TAP; % Series admítanse of transformers
%   Mutual admittances:
   Ynodo = sparse([  PLi;   PTr;   QLi;   QTr ],...
                  [  QLi;   QTr;   PLi;   PTr ],...
                  [-Ylin; -Yxtr; -Ylin; -Yxtr ], Nnod, Nnod);

 %  Self admittances:
   Ynodo = Ynodo + ...
   sparse([       PLi;        QLi;      PTr;      QTr;    NumND],...
          [       PLi;        QLi;      PTr;      QTr;    NumND],...
          [Ylin+1j*Blin; Ylin+1j*Blin; Yxtr./TAP; Yxtr.*TAP; 1j*Bcomp],...
           Nnod, Nnod);

%%  ******************   CASE BASE GENERATION    ********************

   Tol_NR   = 1e-3;    % Tolerance error                    .
   Max_Iter = 60;      % Maximum number of possible iterations.

%  -> Initialization:
   Vabs(NumNC) = 1.0;  % Voltage magnitude of PQ nodes
   Tetan       = zeros(Nnod,1);    % Phase angles in all nodes

 %  -> Calculation of specified powers:
 %  Active power:
   PLg = Pload(NumNG);
   QLg = Qload(NumNG);
   Pesp(NumNG,1)  = (Pgen-PLg)/Sbase;      % for PV nodes
   Pesp(NumNC,1)  = (-Pload(NumNC))/Sbase; % for PQ nodes
%  Reactive power:
   Qesp  = -Qload/Sbase;   % for PQ nodes

   Ndim = Nnod + length(NumNC) - 1;    % Jacobian dimension.
   iter = 0;   % Initialization of iteration counter.
   tic

   while iter <= Max_Iter  % Limit of the number of iterations
%   -> Calculation of net active and reactive powers injected to each node:
      [Pnodo Qnodo] = scmplx();

%   -> Calculation of  misadjustment of reactive and active powers:
```

```
      DelP  = Pesp(NumCG) - Pnodo(NumCG);
      DelQ  = Qesp(NumNC) - Qnodo(NumNC);
      Pmismatch = [DelP; DelQ];

%  -> Check the convergence method.
      if max(abs(Pmismatch)) < Tol_NR
         tsol = toc;
         fprintf( 'The case has converged: %d iterations \n', iter);
         fprintf( 'in: %f seg. \n', tsol);
         break;
      end

%   -> Jacobian calculation:
      JB = Jacob();

%   -> Solution of the set of equations:
      DeltaX = linear_solver(JB, Pmismatch);

%   -> Update of magnitudes and angles of nodal voltages:
      Vabs(NumNC)  = Vabs(NumNC).*(1 + DeltaX(Nnod:Ndim));
      Tetan(NumCG) = Tetan(NumCG) + DeltaX(1:Nnod-1);

      iter = iter + 1;
   end

%% ******   CALCULATION OF POINTS OF PV CURVES AT EACH NODE    ******

   Ndim = Nnod + length(NumNC); % Augmented Jacobian dimension.

   sigma0    = 0.3966;     % Predictor step.
   Vpredict  = [Vabs];     % Predicted voltage values.
   Vexact    = [Vabs];     % Corrected voltage values.
   Carga     = [0];        % Load increment.
   lambda    = 0;          % Initial value of lambda.
   b(Ndim,1) = 1;          % Column vector of equation (43).
   Ix = Ndim;              % Index that indicates the continuation parameter.

 % Flag to indicate state before the critical point:
   Superior  = true;

 % Counter used to calculate exactly the number of points alter the critical point:
   Inferior  = 0;

   while(Superior == true) || (Inferior > 0)
%   -> After the critical point has been reached, only more 14 points are obtained.
      if (Superior == false) Inferior = Inferior - 1; end
```

```
%  -> Calculation of the augmented Jacobian as in equation (43):
    JB = Jacob();
    JB(Ndim,Ix) = 1;
    JB(1:Ndim-1,Ndim) = [-Pesp(NumCG); -Qesp(NumNC)];

%  -> Calculation of the tangent vector as in equation (43)
    t = linear_solver(JB, b);

%  -> The critical point has been reached:
    if b(end) < 0
      if Superior == true;
        sigma0   = 0.25;   % Decrement of the predictor step
        Inferior = 14;     %  Calculation of additional 14 points.
        Superior = false;

        Carga(end-1:end)      = [];
        Vpredict(:, end-1:end) = [];
        Vexact(:, end)        = [];
      end
    end

%  -> Selection of the continuation parameter:
    x  = [Tetan(NumCG); Vabs(NumNC); lambda];
    [xk Ix] = max(abs(t)./x);   % As in equation (50)

%  -> Calculation of the predictor step as in equation (45):
    sigma = sigma0/norm(t);

%  -> Calculation of predictor:
    Vabs(NumNC)  = Vabs(NumNC).*(1 + t(Nnod:Ndim-1)*sigma);
    Tetan(NumCG) = Tetan(NumCG) + t(1:Nnod-1)*sigma;
    lambda = lambda + sigma*t(end);

    Vpredict = [Vpredict Vabs]; % Calculated values with the predictor
    Carga  = [Carga lambda];    % Load increment

%  -> Calculation with the corrector:
    iter = 0;
    while iter <= Max_Iter

      [Pnodo Qnodo] = scmplx();   % Computation of the complex power
%     -> Calculation of misadjustment  in active and reactive powers as in:
      DelP = Pesp(NumCG)*(1 + lambda) - Pnodo(NumCG); % Equation (35)
      DelQ = Qesp(NumNC)*(1 + lambda) - Qnodo(NumNC); % Equation (36)
      Pmismatch = [DelP; DelQ; 0];
```

```
        if max(abs(Pmismatch)) < Tol_NR     % Convergence criterion
          break;
        end
%       -> Calculation of the augmented Jacobian.
        JB = Jacob();
        JB(1:Ndim-1,Ndim) = [-Pesp(NumCG); -Qesp(NumNC)];
        JB(Ndim,Ix) = 1;

%       -> Calculation of the set of equations as in (49).
        DeltaX = linear_solver(JB, Pmismatch);

%       -> Checking if the critical point has been found.
        if DeltaX(end) < 0 && Superior == true
          b(Ndim,1) = -1;
          break;
        end

 %      -> Update state variables in the corrector:
        Vabs(NumNC)  = Vabs(NumNC).*(1 + DeltaX(Nnod:Ndim-1));
        Tetan(NumCG) = Tetan(NumCG) + DeltaX(1:Nnod-1);
        lambda       = lambda + DeltaX(end);

        iter = iter + 1;
      end

      Vexact   = [Vexact Vabs];
      Vpredict = [Vpredict Vabs];
      Carga    = [Carga lambda];

    end

%% **   GRAPHICAL INTERFACE FOR DISPLAYING THE PV CURVES   **
    PV_PRINT( Carga, Vpredict, Vexact );

%% **********************   Nested functions   **********************
    function [P Q] = scmplx()

      Vfasor = Vabs.*exp(1j*Tetan);      % Complex form of nodal voltage using Euler.
      Scal =   Vfasor.*conj(Ynodo*Vfasor);   % Calculation of complex power as in (8).
      P = real(Scal); % Net active power injected to each node as in (9).
      Q = imag(Scal); % Net reactive power injected to each node as in (10).

  end

    function Jac = Jacob()
```

```
    Jac = sparse(Ndim, Ndim);

    Vdiag = sparse(1:Nnod, 1:Nnod, Vabs.*exp(1j*Tetan));
    J2    = Vdiag*conj(Ynodo*Vdiag);   % Computation of equation (34)
    P = sparse(1:Nnod, 1:Nnod, Pnodo);
    Q = sparse(1:Nnod, 1:Nnod, Qnodo);

    H = ( imag(J2(NumCG, NumCG)) - Q(NumCG, NumCG));   % Equations  (26) and (30)
    N = ( real(J2(NumCG, NumNC)) + P(NumCG, NumNC));   % Equations (27) and (31)
    J = (-real(J2(NumNC, NumCG)) + P(NumNC, NumCG));   % Equations (28) and (32)
    L = ( imag(J2(NumNC, NumNC)) + Q(NumNC, NumNC));   % Equations (29) and (33)

    Jac = horzcat( vertcat(H, J), vertcat(N, L) );  % Submatrix concatenation as indicated in
(25)

  end

  function x = linear_solver(A, b)

    P = amd(A);           % Ordering of matrix A (Jacobian).
    [L, U]  = lu(A(P, P));  % Application of LU factorization.
  % The set of equations (b=A*x) are solved using the LU factorization:
    y      = L\b(P);
    x(P, 1) = U\y;

  end

end
```

As it was shown above, the four specific tasks of this application are integrated around the principal function: main(), which function is the sequential integration of these tasks. Even though the first three tasks are described in a set of continuous code lines, each task represents one independent process from each other; hence it is possible to separate them as external functions, such as the function PV_PRINT, which executes the four tasks. Additionally, it can be seen that the calculation tasks, flow case base and the points of PV curves use nested functions within the main function that are common to both tasks. The following nested functions are:
Scmplx: This function computes the net complex power injected to each node of the system and it returns its real and imaginary components.
Jacob: This function computes the Jacobian of the conventional load flow problem.
Linear_solver: This function solves the set of linear equations $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$ by using the LU factorization with previous reordering of **A** matrix.

6.1 Reading of input data

A program based on the solution of load flows requires of some input data that describes the network to be analyzed. The information can be included in a text file with a

standardized format with the extension cdf (Common Data Format) (University of Washington electrical engineering, 1999). The reading of the complete information is achieved by using the MATLAB function textread. The reading includes the use of a standard uicontrol of MATLAB to locate the file path. This information is stored in a data matrix of string type, and it can be interpreted by using functions that operate on the variables of string type. There is basic information that is used in the developed program; the variables can be divided into two categories:

Nodal information:

Sbase : Base power of the system.

Nnod : Number of system nodes.

No_B : *n*x1 vector with the numbering of each system bus.

Name_B : *n*x1 vector with the name of each bus system.

Tipo : *n*x1 vector with the node type of the system.

Nslack : The number of the slack node.

Vn : *n*x1 vector with voltage magnitudes at each system node.

Vbase : *n*x1 vector with base values of voltages at each system bus.

Tetan : *n*x1 vector with phase angles at each nodal voltage.

PLoad and **QLoad** : *n*x1 vector with the active and reactive load powers, respectively.

Pgen and **Qgen** : (*n-nc)*x1 vector with the active and reactive power of generators, respectively.

Gc and **Bc** : *n*x1 vector with conductance and susceptance of each compensating element, respectively.

Branch information (transmission lines and transformers), where *l* denotes the total number of transmission lines and *t* is the number of transformers in the electrical network:

PLi and **PTr** : Vectors with dimensions *l*x1 y *t*x1, that indicate one side of transmission lines and transformers connected to the P node, respectively.

QLr and **QTr** : They are similar as PL and PT with the difference that indicates the opposite side of the connected branch to the Q node.

RbrL and **RbrX** : They are similar as the above two variables with the difference that indicate the resistance of each transmission line and transformer.

XbrL and **XbrX** : They are similar as the above with the difference that indicate the reactance of each transmission line and transformer.

Blin : *l*x1 vector with the half susceptance of a transmission line.

TAP : *t*x1 vector with the current position of the TAP changer of the transformer.

After reading the input data, the admittance matrix **Ynode**, and a series of useful pointers are created to extract particular data from the nodal and branch data:

- **NumNG**: An array that points to the PV nodes.
- **NumNC**: An array that points to the PQ nodes.
- **NumCG**: An array that points to all nodes but the slack node.

Tol_NR – the maximum deviations in power are lower to this value in case of convergence - and Max_Iter –is the maximum number of iterations – these variables are used to control the convergence and a value is assigned to them by default after reading of input data.

The program does not consider the checking of limits of reactive power in generator nodes neither the effect of the automatic tap changer in transformers, nor the inter-area

power transfer, etc. Therefore the input data related to these controllers is not used in the program.

6.2 Graphical interface for editing the display of PV curves

In the last part of the program, a graphical user interface (GUI) is generated for the plotting of PV curves from the points previously calculated in the continuous load flows program. Even though the main objective of this interface is oriented to the plotting of PV curves, its design allows the illustration continuation method process, i.e. it has the option of displaying the calculated points with the predictor, corrector or both of them, for example, it can be seen in figure 5 the set of points obtained with the predictor-corrector for the nodes 10,12,13, and 14 of the 14-node IEEE test power system. This is possible by integrating the MATLAB controls called uicontrol´s into the GUI design, the resultant GUI makes more intuitive and flexible its use.

The numbers 1-12, that are indicated in red color in figure 5, make reference to each uicontrol or graphical object that integrates the GUI. The graphical objects with its uicontrol number, its handle and its description are given in table 1.

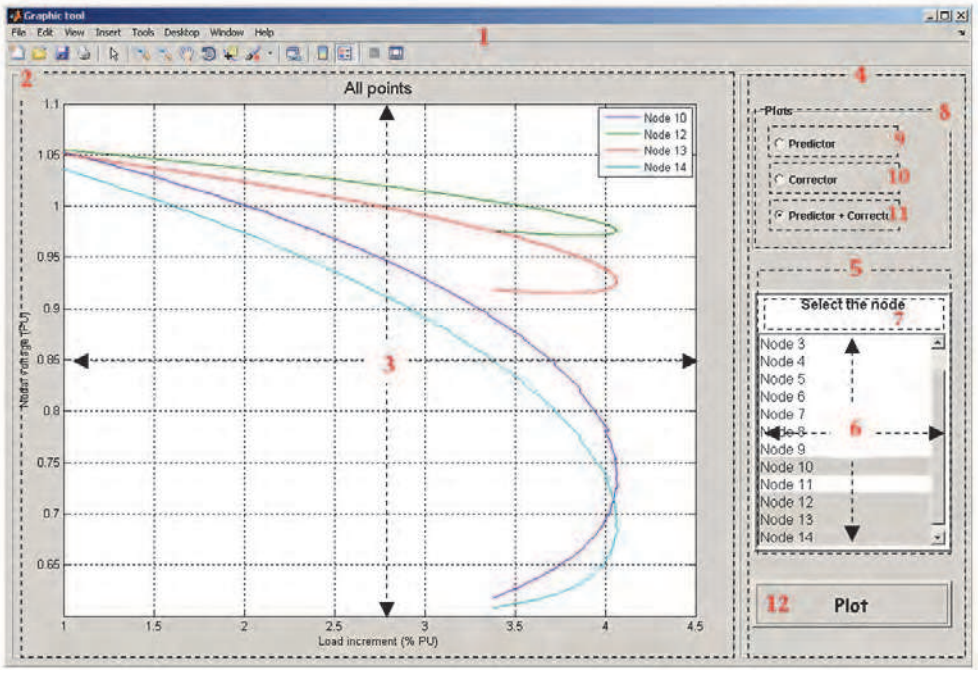

Fig. 5. GUI for plotting the PV curves for any node.

Uicontrol number	handle	Description
1	hPV	Graphical object known as figure
2	hPanel_print	Panel located to the left of hPV
3	hAxes_Curve	Object for 2D graphs known as axes
4	hPanel_control	Panel located to the right of hPV
5	hPanel	Panel located in the upper part of hPanel_control
6	hlb_log	Graphical object known as listbox
7	hlb_txt	Graphical object known as textbox
8	hChoice_PV	Set of objects known as radio buttons
9	opt0	Graphical objects known as radiobutton
10	opt1	Graphical objects known as radiobutton
11	opt2	Graphical objects known as radiobutton
12	- without handle-	Graphical objects known as pushbutton

Table 1. Description of each graphical object used in the GUI shown in figure 5.

The complete code of the function that generates the GUI is presented below, where the code lines for creating each graphical object mentioned in table 1 are fully described.

```
function PV_PRINT(lambda, predictor, corrector )

   [Nodos columnas] = size(predictor);
   clear columnas;
   etiqueta = {};

% Variable for registering the elements of listbox that relates the system nodes, which can
have plotted its PV curve
   for ii = 1:Nodos
      etiqueta{ii, 1} = ['Node ' num2str(ii)];
   end

 % Variables for configuring the graphical object, color and screen size.
   BckgrClr = get(0, 'defaultUicontrolbackgroundColor');
   Sc_Sz = get(0, 'ScreenSize');
   wF = 0.85;
   hF = 0.85;
   wF_Pix = wF*Sc_Sz(3);
   hF_Pix = hF*Sc_Sz(4);

 % Creation of the object type figure: main window.
   hPV = figure(...
         'Name', 'Graphic tool', ...
         'units', 'normalized', 'Color', BckgrClr,...
```

```
        'MenuBar', 'figure', 'numbertitle', 'off',...
        'visible', 'on', 'Position', [0.05 0.05 wF hF]);
    delete(gca);

% Generation of the left panel included in hPV.
    hPanel_print = uipanel('Parent', hPV, ...
        'Units', 'normalized', 'BackgroundColor', BckgrClr, ...
        'BorderType', 'etchedout', 'BorderWidth', 1,...
        'visible', 'on', 'Position', [0.005 0.01 0.75 0.98]);

% Declaration of axes object where the curves are to be plotted and it is located in
hPanel_Print.
    hAxes_Curve = axes('Parent', hPanel_print, ...
        'units', 'normalized', 'Position',[0.07 0.07 0.88 0.88], ...
        'Box', 'on', 'Tag', 'hAxes_Curve', 'Visible', 'off');

% Generation of the right panel in hPV.
    hPanel_control = uipanel('Parent',hPV, ...
        'Units', 'normalized', 'BackgroundColor', BckgrClr, ...
        'BorderType', 'etchedout', 'BorderWidth', 1,...
        'visible', 'on', 'Position', [0.76 0.01 0.235 0.98]);

% Declaration of a panel inside the hPanel_control  and it contains the listbox and textbox.
    hPanel = uipanel('Parent', hPanel_control, ...
        'Units', 'normalized', 'BackgroundColor', BckgrClr, ...
        'BorderType', 'etchedout', 'BorderWidth', 3,...
        'visible', 'on', 'Position', [0.05 0.18 0.9 0.45]);

% Listbox located in hPanel and it is used for selecting the node or desired nodes for
displaying its curves.
    hlb_log = uicontrol('Parent', hPanel, ...
        'Units', 'normalized', 'BackgroundColor', [1 1 1], ...
        'FontName', 'Lucida Console', 'HorizontalAlignment', 'left',...
        'Tag', 'hlb_log', 'Position', [0 0 1 0.85], 'fontsize', 12,...
        'Enable', 'on', 'Style', 'listbox', 'String', etiqueta, ...
        'Max', Nodos, 'Min', 0, 'Clipping', 'off', 'Visible', 'on');

% Textbox located in hPanel and it is used for displaying any string.
    hlb_txt = uicontrol('Parent', hPanel,...
        'Units', 'normalized', 'HorizontalAlignment', 'center',...
        'style', 'text', 'fontname', 'Comic Sans MS',...
        'BackgroundColor', 'w', 'string', 'Select the node',...
        'fontsize', 12, 'fontweight', 'bold', 'visible', 'on',...
        'Position', [0.0 0.85 1 0.15]);

% Uicontrol of type optionbutton, and it is located in hPanel_control.
   hChoice_PV = uibuttongroup('Parent', hPanel_control, ...
```

```
        'Units', 'normalized', 'Position', [0.05 0.7 0.9 0.25], ...
        'BackgroundColor', BckgrClr, 'FontName', 'Lucida Console', ...
        'FontSize', 10, 'FontWeight', 'bold', 'Tag','hChoice_PV',...
        'Title', 'Plots', 'visible', 'off');

 % Optionbutton located in hChoice_PV which is used for ploting the points obtained with
the corrector.
   opt0 = uicontrol('Parent', hChoice_PV, ...
        'Units', 'normalized', 'pos', [0.1 0.68 0.8 0.25], ...
        'BackgroundColor', BckgrClr, 'FontName', 'Lucida Console', ...
        'FontSize', 9, 'FontWeight', 'bold', 'Style', 'Radio',...
        'String', 'Predictor', 'HandleVisibility', 'off');

 % Optionbutton located in hChoice_PV and it is used for plotting the points calculated with
the corrector.
   opt1 = uicontrol('Parent', hChoice_PV, ...
        'Units', 'normalized', 'pos', [0.1 0.4 0.8 0.25], ...
        'BackgroundColor', BckgrClr, 'FontName', 'Lucida Console', ...
        'FontSize', 9, 'FontWeight', 'bold', 'Style', 'Radio',...
        'String', 'Corrector', 'HandleVisibility', 'off');
 % Optionbutton located in hChoice_PV  and it is used for plotting the points obtained with
the predictor + corrector.
   opt2 = uicontrol('Parent', hChoice_PV, ...
        'Units', 'normalized', 'pos', [0.1 0.125 0.8 0.25], ...
        'BackgroundColor', BckgrClr, 'FontName', 'Lucida Console', ...
        'FontSize', 9, 'FontWeight', 'bold', 'Style', 'Radio',...
        'String', 'Predictor + Corrector', 'HandleVisibility', 'off');

 % Configuration of the graphical object hChoice_PV.
   set(hChoice_PV, 'SelectionChangeFcn', @selcbk);
   set(hChoice_PV, 'SelectedObject', opt0);
   set(hChoice_PV, 'UserData', 0);
   set(hChoice_PV, 'Visible', 'on');

% Generation of the command button which calls the function that generates the selected
graphs for the set of optionbutton located in hChoice_PV.
   uicontrol('Parent', hPanel_control, ...
        'Units', 'normalized', 'style', 'pushbutton',...
        'FontName', 'Comic Sans MS', 'visible', 'on',...
        'fontsize', 9, ...
        'callback', {@RefreshAxes, hChoice_PV, hAxes_Curve, hlb_log, lambda, predictor,
corrector},...
        'FontWeight', 'bold', 'string', 'Plot', ...
        'Position',[0.05 0.05 0.9 0.08]);

end
```

```
function selcbk(source, eventdata)

   switch get(eventdata.NewValue, 'String')
      case 'Predictor'
         set(source, 'UserData', 0);
      case 'Corrector'
         set(source, 'UserData', 1);
      otherwise
         set(source, 'UserData', 2);

   end

end

function RefreshAxes(source, eventdata, hChoice_PV, hAxes_Curve, hlb_log, lambda,
predictor, corrector)

  Nodos = get(hlb_log, 'Value');      % It is obtained the node(s) whose PV curves will be
plotted
   switch get(hChoice_PV, 'UserData') % Option selection for the object uibuttongroup

      case 0  % The points obtained with the predictor are plotted for the selected nodes.
         plot(hAxes_Curve, 1 + [lambda(1), lambda(2:2:end)], [predictor(Nodos, 1),
predictor(Nodos, 2:2:end)]);
         title(hAxes_Curve, 'Plot the points of predictor', 'FontSize', 14);

      case 1  % The points obtained with the corrector are plotted for the selected nodes.
         plot(hAxes_Curve, 1 + lambda(1:2:end), corrector(Nodos,:));
         title(hAxes_Curve, 'Plot the points of corrector ', 'FontSize', 14);
      otherwise
      % The points stored in the variable predictor are plotted for the selected nodes.
         plot(hAxes_Curve, 1 + lambda, predictor(Nodos, :));
         title(hAxes_Curve, ' All points ', 'FontSize', 14);

   end

    if length(Nodos) > 1
    % If there are more than one node, they are labeled for marking the shown curves
      etiqN = get(hlb_log, 'String');
      legend(hAxes_Curve, etiqN{Nodos});
   end

% The grid are shown in the plot.
   grid(hAxes_Curve, 'on');

% Automatic numbering of axes (selection of maximum and minimum limits for x-y axes)
```

```
    axis(hAxes_Curve, 'auto');

% Labeling of X and Y axes.
    xlabel(hAxes_Curve, 'Load increment (% PU)', 'FontSize', 10);
    ylabel(hAxes_Curve, 'Nodal voltage (PU)', 'FontSize', 10);

% The object becomes visible with the handle hAxes_Curve.
    set(hAxes_Curve, 'Visible', 'on');

end
```

Since the function PV_PRINT is external to the function main, the created variables within main cannot be read directly for the function PV_PRINT, hence it is required to declare arguments as inputs for making reference to any variable in main. The main advantage of using external functions consists in the ability to call them in any application just by giving the required arguments for its correct performance. In order to reproduce this application, it is necessary to copy the code in an m-file which is different from the main.

7. Conclusion

An electrical engineering problem involves the solution of a series of formulations and mathematical algorithm definitions that describe the problem physics. The problems related to the control, operation and diagnostic of power systems as the steady-state security evaluation for the example the PV curves, are formulated in matrix form, which involves manipulation techniques and matrix operations; however the necessity of operating on matrixes with large dimensions takes us to look for computational tools for handling efficiently these large matrixes. The use of script programming which is oriented to scientific computing is currently widely used in the academic and research areas. By taking advantage of its mathematical features which are normally found in many science or engineering problems allows us solving any numerical problem. It can be adapted for the development of simulation programs and for illustrating the whole process in finding a solution to a defined problem, and thus makes easier to grasp the solution method, such as the conventional load flow problem solved with the Newton-Raphson method.
MATLAB has demonstrated to be a good tool for the numerical experimentation and for the study of engineering problems; it provides a set of functions that make simple and straightforward the programming. It also offers mechanisms that allow dealing with mathematic abstractions such as matrixes in such a way that is possible to develop prototype programs which are oriented to the solution methods by matrix computations. The development of scripts or tools can be considered to be a priority in the academic area such that they allow achieving a valid solution. It is also advisable to take advantage of the MATLAB resources such as: vector operations, functions and mechanisms for operating on each matrix element without using any flow control for the program, i.e. for loop; this offers the advantage of decreasing the number of code lines of the script program. These recommendations reduce the computation time and allow its easy usage and modification by any user. Finally, MATLAB offers powerful graphical tools which are extremely useful for displaying the output information and to aid interpreting the simulation results. In this chapter, the plotting of the corrector points (PV curve) has been presented for a given load

that can be considered as critical point or voltage collapse of the power system. The technique for continuous load flow has been applied to determine the PV curves of the 14-node IEEE test system, and it has been shown that a 4.062% load increment can lead to the instability of the system, and it also has been determined that the node 14 is the weaker node of the system.

8. References

Arrillaga, J. & Walsion,N. (2001). *Computer Modeling of electrical Power Systems (2th),* Whyley, ISBN 0-471-87249-0.

D. A. Alves, L. C. P. Da Silva, C. A. Castro, V. F. da Costa, "Continuation Load Flor Method Parametrized by Transmissión Line Powers", Paper 0-7801-6338-8/00 IEEE 2000

Garcia, J., Rodriguez, J.& Vidal, J. (2005). Aprenda Matlab 7.0 como si estuviera en primero, Universidad Politécnica de Madrid, Retrieved from http://mat21.etsii.upm.es/ayudainf/aprendainf/Matlab70/ matlab70primero.pdf

Gill, P., Murray, W. & Wrigth M., (1991). *Nuemric linear Algebra and optimization vol 1* (ed), Addison-wesley, ISBN 0-201-12647-4, Redwood City, US

Graham, R., Chow, J., &, Vanfretti, L. (2009). Power System Toolbox, In: Power System Toolbox Webpage, 15.03.20011, Available from: http://www.ecse.rpi.edu/pst/

Kundur, P. (1994). Voltage Stability, *Power System stability and control*, pp. 959-1024, Mc Graw Hill, ISBN 0-07-035959-X, Palo Alto, California

Mahseredjian, J., & Alvarado, F. (1997). MatEMTP, In: Creating an electromagnetic transients program in MATLAB, Available: 15.013.2011, http://minds.wisconsin.edu/handle/1793/9012

M. G. Ogrady, M. A. Pai, "Analysis of Voltage Collapse in Power Systems", Proceedings of 21st Annual North American Power Symposium, Rolla, Missouiri, October 1999ric Manufacturers Association, Washington, D.C.

Milano, F. (2010). *Power system modelling and scripting* (*1th*), Springer, ISBN 9-783-642-13669-6, London, ___

Milano, F. (2006). PSAT, In: Dr. Federico Milano Website, 15.03.2011, http://www.power.uwaterloo.ca/~fmilano/psat.htm

PowerWorld Corporation (2010). http://www.powerworld.com

Siemens, PTI. (2005). http://www.energy.siemens.com/us/en/services/powertransmission-distribution/power-technologies-international/softwaresolutions/pss-e.htm

Tim, D. (2004), Research and software development in sparse matrix algorithms, In: AMD, 12.03.2011, Available from http://www.cise.ufl.edu/research/sparse/amd/

Venkataramana, A. (2007). *Power Computational Techniques for Voltage Stability Assessment and Control* (*1th*), Springer, ISBN 978-0-387-26080-8.

Venkataramana Ajjarapu, Colin Christy, "The Continuation Power flor: A Tool for Steady State Voltaje Stability Analysis", Transactions on Power Systems, Vol. 7, No. 1, Fabruary 1992

W. C. Eheinboldt, J. B. Burkhardt, "A Locally Parametrized Continuation Process", ACM Transactions on Mathematical Software, Vol. 9 No. 2, June 1986, pp. 215-235.

University of Washington electrical engineering. (1999). 14 Bus, In: Power Systems Test Case Archive, 25.01.2011, Available from http://www.ee.washington.edu/research/pstca/.

Y. Yamura, K. Sakamoto, Y. Tayama, "Voltaje Instability Proximity Index baesd on Multiple Load Flor Solutions in Ill-Conditioned Power Systems". Proceedings of the 27th IEEE Conference on Decision and Control, Austin, Texas, December 1988

Zimmerman, R., Murillo, C., & Gan, D. (2011). MATPOWER, In: A MATLAB Power System Simulation Package, 15.03.2011, Available from: http://www.pserc.cornell.edu/matpower/

MATLAB for Engineers - Applications in Control, Electrical Engineering, IT and Robotics

Edited by Dr. Karel Perutka

ISBN 978-953-307-914-1
Hard cover, 512 pages
Publisher InTech
Published online 13, October, 2011
Published in print edition October, 2011

The book presents several approaches in the key areas of practice for which the MATLAB software package was used. Topics covered include applications for: -Motors -Power systems -Robots -Vehicles The rapid development of technology impacts all areas. Authors of the book chapters, who are experts in their field, present interesting solutions of their work. The book will familiarize the readers with the solutions and enable the readers to enlarge them by their own research. It will be of great interest to control and electrical engineers and students in the fields of research the book covers.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Ricardo Vargas, M.A Arjona and Manuel Carrillo (2011). PV Curves for Steady-State Security Assessment with MATLAB, MATLAB for Engineers - Applications in Control, Electrical Engineering, IT and Robotics, Dr. Karel Perutka (Ed.), ISBN: 978-953-307-914-1, InTech, Available from: http://www.intechopen.com/books/matlab-for-engineers-applications-in-control-electrical-engineering-it-and-robotics/pv-curves-for-steady-state-security-assessment-with-matlab

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821