Description of the code in Matlab

1. Description of the program's files and folders

The program is in the Load_Estimation_PS folder, which contains two subfolders: "01. Matlab" with the files and routines used and generated by Matlab; and "02. OpenDSS" with the OpenDSS files used for the simulation.

The "02. OpenDSS" folder has the following subfolders: "IEEE 13 Barras"; "IEEE 34 Barras", "IEEE 37 Barras" and "IEEE 123 Barras". Inside each of these folders are the .dss extension files and others needed for OpenDSS to compile each of these files. Any of these circuits can be opened with the OpenDSS graphical interface and simulate them. These files were copied from the OpenDSS installation folder, specifically from the "IEEETest Cases" subfolder.

Inside the "01. Matlab" folder, there are main files responsible for the program:

- main_program.m: main routine, file to be executed by the user to perform load estimation;
- gera_dados.m: - generate_data.m: routine that simulates the chosen system using the COM interface with OpenDSS and returns system-related data;
- divideSystems: routine that divides the data generated by the "gera_dados.m" routine by measurement area, if chosen by the user;
- opti_ybus.m: function to be optimized for load estimation;
- userData.m: file with the user's input options. It is recommended that only this file be edited by the user;
- matrizes.mat: file with the main data of the simulated system;
- matrizesN.mat: file with the main data of the simulated system related to measurement area N.

In addition to these files, there are subfolders:

- “common”: auxiliary functions that can be used by any routine or function:
    - defineYLoad.m – defineYLoad(Yest, Ynet, Yposition): A function that adds the load admittances present in the Yest matrix to the Ynet network admittance matrix, returning the result of the sum. The real matrix Yest has dimensions $n_l \times 2$, where n_l is the number of loads. The two columns refer to the real and imaginary parts of the admittances. Each row of the matrix contains the real and imaginary parts of one of the load admittances. By converting the admittances to complex form, each admittance is added to the element whose position within the Ynet matrix is defined by the Yposition vector.
    - inverseZY.m – inverseZY(vector): receives a matrix vector $[a_{ij}]$ of dimensions $n_l \times 2$ and returns a matrix $[b_{ij}]$ of the same dimensions, such that:

$$b_{i1} + i.b_{i2} = \frac{1}{(a_{i1} + i.a_{i2})}$$

    - modifyVector.m – modifyVector(vector): receives a matrix vector $[a_{ij}]$ of dimensions $n_l \times 2$ and returns a matrix $[b_{ij}]$ of dimensions $2n_l \times 1$, such that:

$$\begin{cases} [b_{i1}] = [a_{i1}], if\ i \leq n_l \\ [b_{i1}] = [a_{(i-n_l)2}], if\ i > n_l \end{cases}$$

- unModifyVector.m – unModifyVector(vector): receives a matrix vector $[a_{ij}]$ of dimensions $2n_l \times 1$ and returns a matrix $[b_{ij}]$ of dimensions $n_l \times 2$, such that:

$$\begin{cases} [b_{i1}] = [a_{i1}] \\ [b_{i2}] = [a_{(i+n_l)1}] \end{cases}, \text{ where } i \leq n_l$$

The other folders, whose functions will be discussed later, are:

- “gera_dados”: contains the functions used by the “gera_dados.m” routine, described in section 3.3;
- “log”: contains the result files of simulations generated by the "main_program.m" routine;
- “main_program”: contains the functions used by the "main_program.m" routine, described in section 3;
- “opti_ybus”: contains auxiliary functions used to optimize the “opti_ybus.m” function, described in section 3.5.

The entire process is carried out from the main routine main_program.m. It is what calls the other routines related to the process.

2. Division of the algorithm into steps

The description of this routine will be used to comment on the other routines and functions. The tasks performed by this routine, in sequence, are:

1. User selection of the circuit to be simulated;
2. User-defined simulation options through the "userData.m" routine;
3. Executes the “gera_dados.m” routine to obtain data from the chosen system;
4. Processes the data for estimation, which consists of the following substeps:
    a. Processes the system data to obtain the input data for the "opti_ybus.m" function;
    b. Processes the system data to obtain the input data for the "patternsearch" function;
    c. Creates lists for evaluating the estimation, defines the nomenclature and location for saving log files and result record files and carries out the initial filling of these lists and files;
5. Executes the Pattern Search routine in order to optimize the "opti_ybus.m" function;
6. Processes the results and saves them for processing and logging;
7. Finalization.

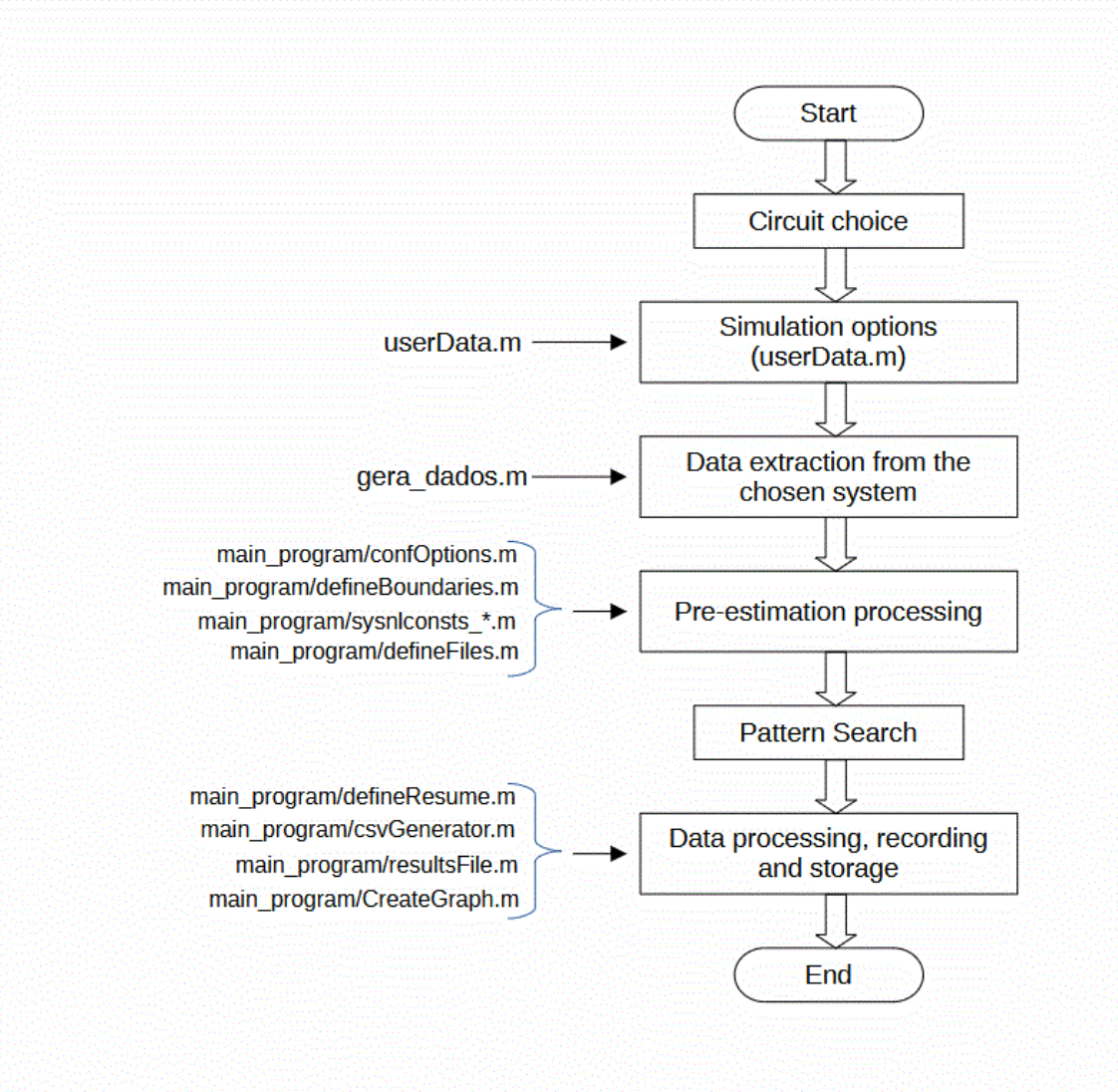

Figure 1 - Flowchart of the load estimation process.

3. Description of algorithm steps and functions

3.1. Choice of simulated circuit

It is defined through a prompt in which the user chooses the value of the “escolha” variable. The program offers 4 options:

- 1 corresponds to the IEEE 13-bus circuit;
- 2 corresponds to the IEEE 34-bus circuit;
- 3 corresponds to the IEEE 37-bus circuit;
- 4 corresponds to the IEEE 123-bus circuit.

Any other choice not listed prompts the user again.

3.2. 3.1. Definition of user simulation options - userData.m routine

It occurs when the “userData.m” routine is executed. This routine is the second user interface with the program, in addition to the prompt. It defines the “mainset” structured variable that contains the user's simulation settings. The attributes that the user can modify are:

mainset.version: a natural number between 1 and 16 that defines how the objective function is calculated. The relationship between each number and the calculation of the objective function is shown below. In the notation used below, zj is considered to be the j^{th} coordinate of the z vector.

mainset.version=1

$$\begin{cases} f(y) = max_{1\le j\le 3}\left(\left|z_j - h_j(y)\right|\right) \\ [z] = \begin{bmatrix} \dot{I_{1A}} \\ \dot{I_{1B}} \\ \dot{I_{1C}} \end{bmatrix}, \quad h(y) = Y(y) \times \begin{bmatrix} \dot{V_{1A}} \\ \dot{V_{1B}} \\ \dot{V_{1C}} \end{bmatrix} \end{cases}$$

mainset.version=2

$$\begin{cases} f(y) = max_{1\le j\le 3}\left(\left|z_j - h_j(y)\right|\right) \\ [z] = \begin{bmatrix} \dot{V_{1A}} \\ \dot{V_{1B}} \\ \dot{V_{1C}} \end{bmatrix}, \quad h(y) = Y^{-1}(y). \begin{bmatrix} \dot{I_{1A}} \\ \dot{I_{1B}} \\ \dot{I_{1C}} \end{bmatrix} \end{cases}$$

mainset.version=3

$$\begin{cases} f(y) = max_{1\le j\le 6}\left(\left|z_j - h_j(y)\right|\right) \\ [z] = \begin{bmatrix} Re\left(\dot{I_{1A}}\right) \\ Re\left(\dot{I_{1B}}\right) \\ Re\left(\dot{I_{1C}}\right) \\ Im\left(\dot{I_{1A}}\right) \\ Im\left(\dot{I_{1B}}\right) \\ Im\left(\dot{I_{1C}}\right) \end{bmatrix}, \quad h(y) = \begin{bmatrix} Re\left(\dot{h_A}\right) \\ Re\left(\dot{h_B}\right) \\ Re\left(\dot{h_C}\right) \\ Im\left(\dot{h_A}\right) \\ Im\left(\dot{h_B}\right) \\ Im\left(\dot{h_C}\right) \end{bmatrix}, \quad \begin{bmatrix} \dot{h_A} \\ \dot{h_A} \\ \dot{h_A} \end{bmatrix} = Y(y) \times \begin{bmatrix} \dot{V_{1A}} \\ \dot{V_{1B}} \\ \dot{V_{1C}} \end{bmatrix} \end{cases}$$

mainset.version=4

$$\begin{cases} f(y) = max_{1\le j\le 6}\left(\left|z_j - h_j(y)\right|\right) \\ [z] = \begin{bmatrix} Re\left(\dot{V_{1A}}\right) \\ Re\left(\dot{V_{1B}}\right) \\ Re\left(\dot{V_{1C}}\right) \\ Im\left(\dot{V_{1A}}\right) \\ Im\left(\dot{V_{1B}}\right) \\ Im\left(\dot{V_{1C}}\right) \end{bmatrix}, \quad h(y) = \begin{bmatrix} Re\left(\dot{h_A}\right) \\ Re\left(\dot{h_B}\right) \\ Re\left(\dot{h_C}\right) \\ Im\left(\dot{h_A}\right) \\ Im\left(\dot{h_B}\right) \\ Im\left(\dot{h_C}\right) \end{bmatrix}, \quad \begin{bmatrix} \dot{h_A} \\ \dot{h_A} \\ \dot{h_A} \end{bmatrix} = Y(y) \times \begin{bmatrix} \dot{I_{1A}} \\ \dot{I_{1B}} \\ \dot{I_{1C}} \end{bmatrix} \end{cases}$$

Where Y(y) is the matrix relationship between the three-phase voltages and currents on the feeder obtained from nodal analysis, as a function of the vector y of estimated loads. Any other value for the mainset.version variable works on an experimental basis and does not guarantee the consistency of the results.

mainset.restriction: Enables restrictions to be applied to the objective function.

- mainset.restriction=1 enables power factor restriction;
- mainset.restriction=2 enables voltage level factor restriction;
- mainset.restriction=3 enables restriction on transformer loading;
- mainset.restriction=4 enables restriction on voltage measurements taken on busbars;

- mainset.restriction=5 enables a restriction on current measurements made on network elements;
- any other value for this variable runs the optimization function without restriction.

Other experimental fields within the mainset structure, which are not incorporated into the userData.m file, but which can be modified by the user are:

- mainset.const: multiplier constant to increase all the dimensions of the search space (programmed with a constant value equal to 1 so as not to interfere with the operation of the algorithm);
- mainset.dom: modifies the search domain of the optimization function between the impedance (mainset.dom=1) and admittance (mainset.dom=2) options. It was programmed with a value of 1 to maintain the search domain between admittances. The operation of the routine for the impedance option is not guaranteed.
- mainset.method: defines the search method. The program was originally designed to optimize with several different search methods. The methods that have been incorporated so far are optimization using pattern search (mainset.method=0) and genetic algorithms (mainset.method=1). The variable was programmed to use only the pattern search function. The use of GA was incorporated on an experimental basis.

All the repository results were generated without modifying the default value of these last 3 mainset attributes. Modifying them leads to results not described in the repository or unexpected results. The operation of the program is not guaranteed by modifying these variables.

Furthermore, in the userData.m file, the user can modify the value of the variables relating to the busbars that contain voltage and current measurements. These variables are:

- barraVmedx: list of the number of bars in the circuit IEEEx bars that have voltage measurements, where x is the number of bars in the circuit. The name of the bars must be identical to the name of the bar in the native OpenDSS file for the circuit.
- elemImedx: list of network elements in the IEEEx busbar circuit in which current measurement is carried out. The nomenclature of the elements must be the same as the corresponding element modeled in the native OpenDSS file.

3.3. Obtaining system data – “gera_dados.m” routine

The "generate_data.m" routine, based on the selected circuit for simulation, locates the file in the OpenDSS folder, and uses the OpenDSS COM interface to execute the file corresponding to the circuit. Finally, the routine stores the main variables generated in the "matrizes.mat" file, in list form. The purpose of generating data in list form is to make it easier for the user to interpret and debug. The "matrizes.mat" file contains the following variables:

- escolhadiv – variable that indicates whether the user would like to divide the circuit into measurement areas. The division will only be possible if the program identifies in the list of points chosen by the user to contain measurement, any bar with voltage and current measurement other than the feeder bar;
- barraVmed – list of voltage measurement points, imported from the userData.m routine;
- elemImed – elements with current measurement, imported from the userData.m routine;

- barras – variable that indicates the number of bars in the simulated circuit;
- name – string in the form “IEEEx” that indicates the name of the circuit, in which x is the number of busbars;
- divpoints – when the same busbar has voltage and current measurements, it can be used to divide the circuit into two subsystems, as if it were a feeder. The divpoints routine identifies these busbars and adds them to this list;
- subredes – a structure which, if the system is divided into measurement areas, contains lists of the busbars present per subrede;
- node_order – list of circuit busbars with clear identification of feeder busbars, busbars with voltage measurement and busbars with current measurement;
- ptos_medV – number of (single-phase) busbars on which voltage is measured;
- ptos_medSource – number of single-phase busbars on the feeder (usually 3);
- Ysistema_lista – list that contains the admittance matrix of the complete circuit, considering the loads;
- Vorder – list that contains the nodal voltage vector, rows in the same order as the rows in the node_order list;
- Iorder – list that contains the vector of current injections, rows in the same order as the rows in the node_order list;
- Load – list of circuit loads and their electrical attributes;
- Yraiz – vector that contains the true admittance of the loads in the circuit;
- Zraiz – vector that contains the true impedance of the loads in the circuit;
- Sraiz – vector that contains the true power consumed by the loads in the circuit;
- Ypos – list extracted from the 'Load' list with the position of the loads within the Yrede_list matrix, so that adding the admittances of the Yraiz vector in the Yrede_list positions indicated by this list results in the admittance matrix of the circuit with the Ysistema_list loads;
- medi_list: list of elements where current and attributes are measured;
- ptos_medI – number of elements with current measurement;
- trafoList – list of transformers in the system and attributes;
- Yrede_list – list that contains the admittance matrix of the system without the loads;
- Yload_list – list that contains the difference matrix between the matrices contained in Ysistema_list and Yrede_list, i.e., it only considers the system loads;
- YLoadError – real number that symbolizes the error involved in assembling the admittance matrix.

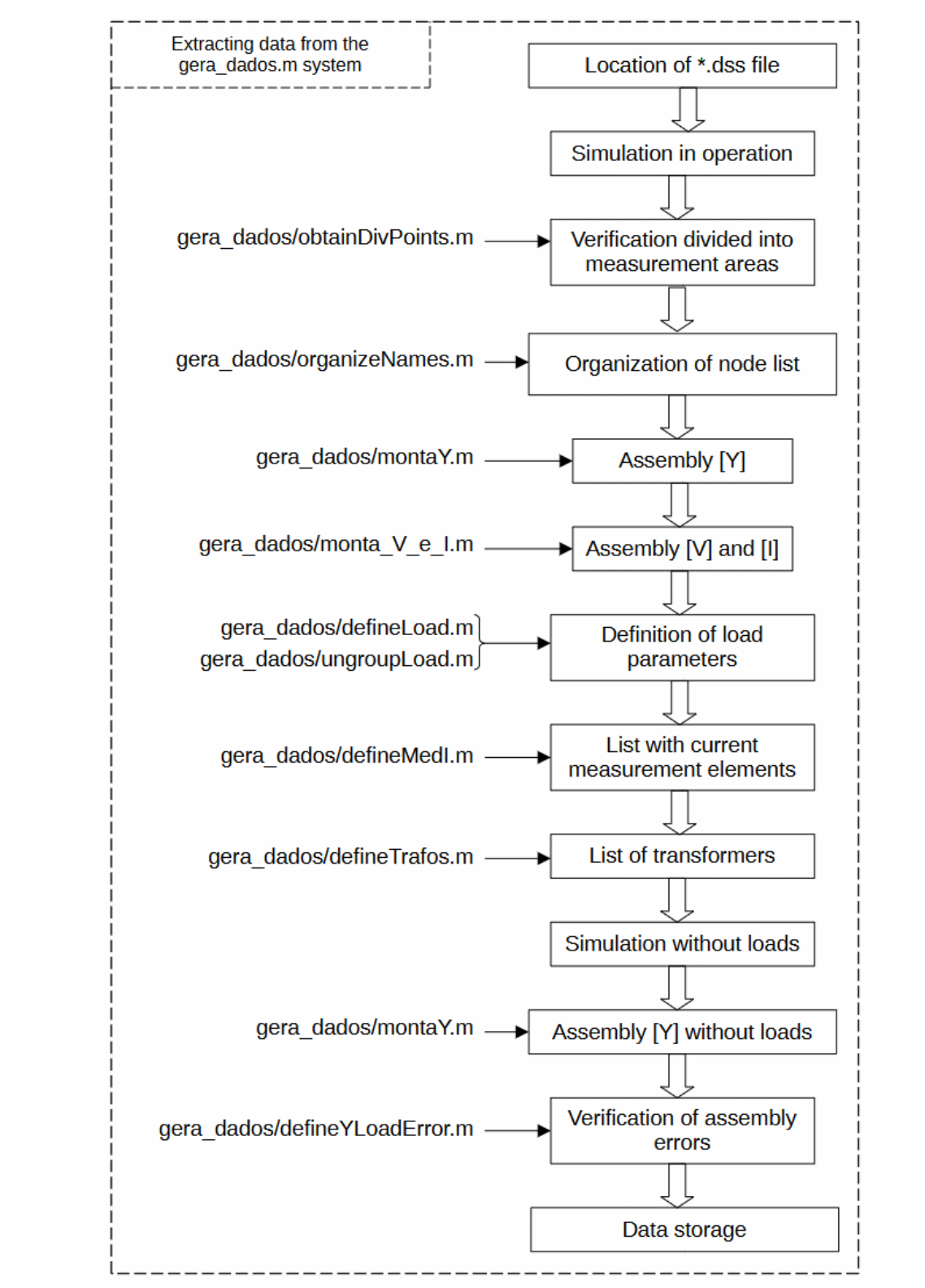

Figure 2 – Flowchart of the process carried out by the gera_dados.m routine

a) Location of the file to be simulated

The routine uses the variable choice, defined in the main routine, to find out the user's choice. A window opens for the user to indicate where the main_program.m file is located. With this indication, the path to the Load_Estimation folder is stored in the string "pasta". This way, files

related to OpenDSS and auxiliary routines can be located on any computer[1], and the functions used by the routine and auxiliary functions can be loaded into the Matlab path.

b) Simulating the circuit with loads from the COM interface

For each possible choice the user makes, the “caminho1” and “caminho2” variables store the paths to each file in the "02. OpenDSS" folder relating to the simulated circuit. The lists of voltage and current measurement bars are loaded and the number of bars in the circuit is set. The COM interface is used to simulate the circuit without opening the OpenDSS interface in the loaded condition. This interface consists of structures and functions that can be used by Matlab. For example, the DSSText structure is used to give commands involving OpenDSS. Thus, the row

DSSText.command = ‘Compile < path and name of the .dss file to be compiled>’

makes OpenDSS compile the specified file. The strings are manipulated to compile the file corresponding to the system the user has chosen. Other commands are used to change the model of all the loads to constant impedance, disable voltage regulators and transformer tap controllers before compilation. After this simulation, other COM interface structures provide all the information of interest about the system, such as the list of busbars, voltage, current and admittance matrices, etc. The structures most commonly used throughout the text are DSSCircuit, which contains general variables relating to the circuit (admittance matrices, nodal voltage vectors, etc.); DSSElement, which contains data relating to the circuit elements and DSSLoads, with data specific to the system loads.

c) Checking whether the circuit can be divided

After the simulation, the *obtainDivPoints(DSSElement, barraVmed ,elemImed)* function compares the lists of voltage measurement points and current measurement elements, identifies the bars with voltage and current measurements, makes a list of the points where the circuit can be divided into smaller subnetworks (measurement areas) and asks the user at the prompt if they prefer to subdivide the system (option 1). The user's choice is stored in the variable escolhadiv. When the user chooses to divide the system, the gera_dados routine proceeds normally, but at the end of the routine, the divideSystems.m program saves the data for each measurement area in a matrizesX.mat file, analogous to the "matrizes.mat" file, where X is the number of the measurement area. All the results presented were designed without using this feature.

gera_dados/obtainDivPoints.m - obtainDivPoints(DSSElem, barraV ,elemI): function which, from the network elements with current measurement specified in the "elemI" list, obtains the model of these elements in OpenDSS from the DSSElem object, checks whether the terminals of these elements are present in the list of points with voltage measurement "barraV". If so, it is a busbar with both voltage and current measurement, and therefore able to divide the circuit into subnetworks. Returns a list of busbars in this condition.

d) Organization of the list of circuit nodes

The list of nodes in the simulated circuit is obtained from the DSSCircuit.YNodeOrder and DSSCircit.AllNodeNames variables in the DSSCircuit structure. The list contains the identification of the circuit nodes, usually numeric format with some alphanumeric characters, with no apparent standardization. From these lists, at this stage, the "node_order" list is generated, which is a list of bars with standardized identification, in which the feeder bars and

[1] As long as the internal structure of the Load_Estimation folder is kept the way it was presented.

the bars that contain voltage measurement are marked with the asterisk character, for easy identification, and occupy the first positions in the list.

Thus, the node_order list is organized to follow this order: feeder busbars in the first positions, followed by busbars with voltage measurement and busbars with current measurement. In the last positions are the unmeasured busbars. This order is achieved using matlab's sortrows function, which organizes a list in alphabetical order. The organizeBus(barraVmed, DSSCircuit, DSSElements, DSSSolution, divpoints) function processes the list of bars so that the native "sortrows" command has the expected effect.

gera_dados/organizeBus.m – organizeBus (barraV, DSSCirc, DSSElem, DSSSol, divpoint): receives as arguments DSSCirc, DSSEleme and DSSSol objects from the COM interface, the list of division points of the divpoint circuit and the list of bars with voltage measurement barraV and has the purpose of modifying the naming of OpenDSS native bars in order to:

a) allow simple identification of the feeder, the current and voltage measurement points and the subsystem to which the busbar belongs (in the case of measurement areas);
b) allow the busbars to be sorted, using the sortrows command, so that the generator busbars are in the first positions and the voltage measurement busbars come next;
c) when there is more than one measurement area, the busbars must be sorted, from the sortrows command, so that the busbars of measurement area 1 are in the first positions, followed by the busbars of measurement area 2 and so on;
d) within each measurement area, the members must be sorted according to item 2, with the generator being replaced by the main measurement, which gives rise to the measurement area in question For this purpose, the general naming structure of the lists will be adopted as follows:

(SS)_(MA/MT)_(Barra).(Fase) or (SS)_(Barra).(Fase)

where:

- SS = Subsystem, indicates the measurement area in which the busbar is located. If the system is not divided into measurement areas, SS will be equal to 01 for all busbars. The correspondence between each busbar and its measurement area is based on the network's incidence matrix and the fact that it is radial;
- MA/MT = Feeder marking/Voltage measurement marking, indicates whether the busbar belongs to the system feeder/measuring area or whether the busbar has voltage measurement.
  - If it is a feeder/generator busbar, MA/MT = '***';
  - If it is a busbar with voltage measurement only, MA/MT = '*_(tag_ordem)'.

    Ps: (tag_ordem) is a two-digit number to sort the busbars according to the medVODSS list;

  - If it is a busbar with voltage measurement and there is a current measuring element such that the current is incident on that busbar, this could be a point of division of the circuit into measurement areas. In this case, MA/MT = '**_(tag_ordem)'. Ps: (tag_ordem) is a two-digit

number to sort the busbars according to their order in the medVODSS list;

- If the busbar is neither a feeder busbar nor has voltage measurement, this marker will be absent in the nomenclature, MA/MT = '', the busbar nomenclature will be of the type (SS)_(Barra).(Fase).
  - ✓ (Barra) - busbar identifier, as provided by OpenDSS, corrected by a naming system described in organizeNames.m;
  - ✓ (Fase) - 1 and/or 2 and/or 3, depending on which phase of the busbar will be used. When more than one phase is used, the phases are separated by dots (e.g. 00670.1.2 refers to phases 1 and 2 of busbar 00670).

Since the identification of the buses is numeric, the organizeNames.m function is used within the organizebus.m routine to standardize the naming of the buses.

gera_dados/organizeNames.m – organizeNames(name): receives a list of busbar names and modifies it so that each busbar is associated with a 5-character identifier, with missing characters being filled in with zeros. When the busbars are two-phase or single-phase, the busbar names discriminate the phases using dots (e.g. "23.1.2" means busbar 23 is connected to the circuit on phases A and B only). These phase indicators are maintained by the function and do not count as bar nomenclature.

In addition to the modified YNodeOrder list of busbars, the organizebus.m function returns the divpoint corrected list of circuit division points, if there are points that cannot be in different subnets; it also returns a "subredes" list of busbars belonging to each subnet, if applicable. The YNodeOrder list is submitted to matlab's sortrows function, resulting in the node_order variable.

e) Assembly of the Nodal Admittance Matrix

The node_order variable is used to count how many single-phase nodes refer to feeders and how many refer to busbars with voltage measurements. After that, the montaY(DSSCircuit, YNodeOrder, Ysistema_list, escolhadiv) function retrieves the elements of the admittance matrix of the COM interface system from the DSSCircuit.SystemY attribute and processes them to return the Ysistema_list list.

gera_dados/montaY.m – montaY(DSSCirc, nos, nos_ordem): a function that, from the OpenDSS DSSCirc (DSSCircuit) object, retrieves the admittance matrix of the network elements, whose rows and columns follow the same order as the "nos" bar list. The function reorders the matrix to follow the same order as the list of bars “nos_ordem” and generates a list containing the admittance matrix, in which the rows and columns are identified with the names of the respective nodes.

f) Assembly of Nodal Voltage and Injected Current Vectors

The attributes DSSCircuit.YNodeVarray and DSSCirc.YCurrents contain the elements of the vectors of nodal voltages and injected currents, respectively. The function monta_V_e_I(DSSCircuit, YNodeOrder, Ysistema_list, escolhadiv) retrieves these elements and processes them to obtain the Vorder and Iorder lists.

gera_dados/monta_V_e_I.m – monta_V_e_I(DSSCirc, nos, Ysis_list,choice): obtains the nodal voltage and injected current vectors from the DSSCirc object of the COM interface, whose ordering is the same as the “nós” list and processes them to generate the Vorder and Iorder

lists with bus voltage values and injected currents. In the Vorder list, the columns are identified as follows:

- Column 1: identification of the nodes, according to node_order;
- Column 2: value of the complex injected voltage at each node;
- Column 3: value of the voltage modulus in pu at each node;
- Column 4: vector corresponding to the following operation between matrices contained in the Ysistema_list and Iorder lists: $\left[Y_{sistema_{list}}\right]^{-1} * [I_{order}]$
- Modulus of the difference between the values in column 4 and column 2, in order to check for possible processing errors.

The Iorder list is similar, except that it does not contain the column for the modulus of the quantity in pu.

g) Definition of the List of Circuit Loads and Positions in the Admittance Matrix

The defineLoad(DSSCircuit, DSSElement, DSSCktElement, DSSLoads, node_order, barraVmed, barras, escolhadiv) function extracts the load list from the DSSLoads structure and returns the "Load" load list and the "verify" variable, which is an error indicator when assembling this list.

To make it easier for the optimization algorithm to work, the three-phase loads in the load list are broken down into single-phase loads using the ungroupLoad(Load,Vorder) function.

gera_dados/defineLoad.m – defineLoad(DSSCirc, DSSElem, DSSCktElem, DSSCarga, nos_ordem, busmedV, buses, choice): from the objects DSSCirc, DSSElem, DSSCktElem, DSSCarga of the COM interface, the list of nodes nos_ordem, the list of busbars with voltage measurement busVmed, the number of busbars of the circuit buses and the choice escolha if the circuit is subdivided into measurement areas, it returns a list of loads consisting of a table in which each row contains the attributes of one of the loads in the following order:

1. Subsystem: indicates the measurement area in which the load is located;
2. Name: name of the busbar and phases to which the load connects, obtained from the DSSElem.BusNames object and the DSSCarga.Name element;
3. Tensão (V): nominal voltage of the load, obtained from the variable DSSCarga.kV;
4. W: nominal active power of the load, obtained from DSSLoad.kW;
5. Var: nominal reactive power of the load, obtained from DSSCarga.kvar;
6. Conn: load connection to the system, which can be Phase-Earth, Phase-Phase, 3Ph or Delta. This data is obtained by processing the information from the nodes where each load connects.
7. Bus: nodes to which the load connects, obtained from the node_order list, with standardized nomenclature;
8. Ypos: row and column indices that demarcate the position within the nodal admittance matrix where the admittance of that load is located;
9. Yprim: admittance matrix of the load as a network element;
10. Ycarga: main admittance representing the load;
11. P1(kW): real active power consumed by the load, obtained from the DSSElem.Powers object;
12. Q1(kW): real reactive power consumed by the load, obtained from the DSSElem.Powers object;

The real number verify is obtained from the Yprim column by summing the rows of the matrix. It is known that this number must be close to zero, which is an indication of errors in the algorithm for assembling and processing this matrix.

gera_dados/ungroupLoad.m - ungroupLoad(Load, Vorder): transforms the Load load list so that all loads have only a phase-to-phase or phase-to-ground connection. Thus, star loads are broken down into 3 phase-to-earth loads while delta loads are broken down into 3 phase-to-phase loads. The Vorder nodal voltage list is used for this and the decomposed load list is returned.

h) Generation of a list with the elements in which current is measured

This list will be used to build constraints or to improve the objective function. This is done using the defineMedI(DSSCircuit, DSSElement, Vorder, barraVmed, elemImed, node_order, barras, escolhadiv) function. It should be noted that the system feeder is always in this list, and is always the first element.

\gera_dados\defineMedI.m – defineMedI(DSSCirc, DSSElem, Vordem, barraV_med, elem_Imed, no_ordem, buses, choice): uses the OpenDSS objects DSSCirc and DSSElem, the list of current measurement elements elem_Imed, the list of voltages in the Vordem nodes, the list of voltage measurement bars barV_med, the list of nodes no_ordem, the number of buses and the user's choice of dividing the circuit into measurement areas choice. It returns a list in table form of current measuring elements, the columns of which are in the following order:

1. 'Elemento': name of the network element, obtained from the DSSCirc object;
2. Yprim: admittance matrix of the element's network elements, obtained from the DSSElem.Yprim variable;
3. Node1: busbar 1 to which the network element is connected;
4. Subsystem1: measurement area in which busbar 1 is located;
5. Indice Node 1: position in the no_ordem list where busbar 1 is located. If the busbar contains more than one phase, this field will contain a column matrix with more than one number, each indicating the position of a phase;
6. V1: nodal voltage in each phase of busbar 1, obtained from the Vorder list;
7. I1: The current injected into the network element from busbar 1, obtained from the DSSElem.Currents variable;
8. S1: Power flow through the network element from bar 1, obtained from the DSSElem.Powers variable;
9. Node2: busbar 2 to which the network element is connected;
10. Subsystem2: measurement area in which busbar 2 is located;
11. Indice Node 2: position in the no_ordem list where busbar 2 is located, analogous to the Index Node 1 field.
12. V2: nodal voltage in each phase of busbar 2, obtained from the Vorder list;
13. I2: The current injected into the network element from busbar 2, obtained from the DSSElem.Currents variable;
14. S2: Power flow through the network element from busbar 2, obtained from the DSSElem.Powers variable.
15. Imed: current symbolizing the value obtained by the current meter, numerically equivalent to column 13.

i) Generation of a list of transformers in the circuit

This list will be used to build constraints based on the loading of the transformers. The list is built by the defineTrafos(DSSCircuit, DSSElement, node_order, barras, escolhadiv) function.

gera_dados/defineTrafos.m – defineTrafos(DSSCirc, DSSElem, no_ordem, buses, choice): extracts the information on the system's transformers from the DSSCirc.Transformers structure and information from the DSSElem structure and builds a list of the circuit's transformers, with the following attributes:

1. Nome: name of the transformer obtained from the DSSCirc.Transformers.Name variable.
2. Kva: apparent nominal power of the transformer, obtained from the variable DSSCircuit.Transformers.kva.
3. Yprim: admittance matrix of the transformer, obtained from DSSElem.Yprim.
4. Bus1: name of bus 1 to which the transformer is connected;
5. Indice Bus1: index of the positions of the nodes defined in the "Bus1" field in the no_ordem list;
6. Subsystem: área de medição em que a barra 1 está localizada;
7. Bus2: nome da barra 2 na qual o transformador está documentado;
8. Indice Bus2: índice das posições dos nós definidos no campo “Bus2” na lista no_ordem;
9. Subsystem: measurement area in which busbar 1 is located;

j) Simulating the circuit without loads from the COM interface

Similarly to item b, the circuit is simulated again, but now using the command line "DSSText.command = 'batchedit Load..* enabled=no'” ”, which disables the loads. Then, the list Yrede_list that contains the matrix of admittance without the charges is obtained from the montaY function in the same way as in item e.

k) Checking assembly errors in the matrix with loads

The defineYLoadError(Ysistema_list, Yrede_list, Load, node_order, verify) function assembles the load admittance matrix in two different ways and returns the system admittance matrix considering only the Yload_list loads and the YLoadError error index.

\gera_dados\defineYLoadError.m – defineYLoadError(Ysis_list, Ynet_list, Carga, no_ordem, verifica): obtains the Ycarga_list matrix in two different ways:

- From the Load load list, summing the admittances of the loads in their respective positions (Ypos column) in a null matrix. At the end, the elements of this matrix are summed and the sum is stored in the variable verifica2;
- A partir da subtração $[Y_{sistema_{list}}] - [Y_{rede_{list}}]$ of the admittance matrix with the loads and the admittance matrix with the loads. At the end, the elements of the matrix are summed and the sum is stored in the variable verifica3.

At the end of the process, the differences between the variables verify, verify2 and verify3 are taken, of which the largest module is returned to the user in the form of an error index (YLoadError).

l) Storage of the main variables to be reused by other routines

The variables are stored in the matrizes.mat file and a warning “Dados principais gerados” (Main data generated) is issued to the user. If the user has chosen to subdivide the system into

measurement areas, the divideSystems.m routine is triggered. As the results obtained were not based on this routine, it will not be described.

3.4. Processing of system data according to the function opti_ybus.m

At the end of the gera_dados.m routine, if the user has chosen to divide the circuit into measurement areas, they must choose at a prompt which of these areas should be used for simulation. The measurement areas are stored in the structured variable subredes, and the ordering of the lists in this variable defines the number of each subrede and also defines the matrizesX.mat file in which the specific data for that subrede is saved. Thus, once the user has chosen the number of the subnet to be simulated, the matrizesX.mat file is loaded. The data from the file corresponding to the simulated system (matrizes.mat or matrizesX.mat) is loaded into matlab and stored in the structured variable "dados". Thus, all the variables described in section 3.3 become attributes of the "dados" structure.

Processing consists of accessing the lists in the "dados" structure and making them available in numerical format, to be used as arguments for the "opti_ybus.m" function and other processing functions. The post-processed data in this way is stored in the "circuit" structure. The fields in this structure are:

- circuit.Vmed: vector with actual voltage on the busbars;
- circuit.Vpu: vector with the modulus of the voltage on the busbars in pu;
- circuit.Iinj: vector of nodal current injections;
- circuit.Imedido: list of elements in which current is measured;
- circuit.Ssource: apparent power measured at the feeder;
- circuit.Yrede: admittance matrix of the chosen circuit without the loads;
- circuit.Ysistema: admittance matrix of the chosen circuit with the actual loads;
- circuit.Yload: difference between the circuit.Ysistema and circuit.Yrede matrices;
- circuit.Yprim: list with the admittance matrix of the elements in which current is measured and the index of the position of the busbars in which these elements are connected within the busbar voltage vector;
- circuit.Ypos: list with the position of each load admittance within the nodal admittance matrix;
- circuit.ptos_medSource: number of single-phase nodes on the feeder;
- circuit.ptos_medV: number of single-phase busbars where voltage is measured;

Except for the fields Vpu, Ssource and Yload, the other fields in this structure will be passed as arguments to the opti_ybus.m. function. These three fields will be arguments to other data processing functions.

3.5. Pre-estimation processing

a) Processing of system data according to the patternsearch function

At this stage, most of the fields in the search structure, which contains the parameters for the search, are filled in.

Initially, the vectors dados.Yraiz, dados.Zraiz and dados.Sraiz are processed, which contain, respectively, the admittances, impedances and real powers of the loads and store them in the search structure (variáveis search.Yraiz, search.Zraiz and search.Sraiz), which contains the

parameters for optimization. The other fields of the search variable filled in at this stage are those that will be used as arguments for the patternsearch function[2]:

- search.options: contains the search options characteristic of the patternsearch function. These options are available in the function's tutorial on the Internet[3];
- search.chute_inicial: vector with the initial load estimate, from which the algorithm is started;
- search.UB and search.LB: search boundaries. These are, respectively, the maximum and minimum values that each admittance component (real part and imaginary part) can take;
- search.nonlcon: pointer to the function representing the non-linear constraints[4]. If there is no non-linear constraint, this field is left empty.

The options variable is configured from the confOptions(mainset.method, search) function. It defines the search options such as the maximum number of iterations, stopping criteria, size of the search mesh, etc. The vector with the initial estimate is returned by the function defineChuteInicial(dados.Load, circuit.Ssource, mainset.dom) and the vector with the search limits is returned by the function defineBoundaries(search.chute_inicial, search.Raiz, mainset.dom). In the case of constraints, basically all constraints are non-linear. In this case, the patternsearch argument must be a pointer to a function that checks whether these constraints are met. This pointer, depending on the value of the mainset.restriction variable, is stored in the search.nonlcon variable. All the functions that represent non-linear constraints are in the "opti_ybus" folder. They all have the same set of arguments.

- mainset.restriction=1 corresponds to the pointer to the function sysnlconsts_fp(x, circuit.Vmed, dados.medI_list, circuit.Vpu, circuit.Iinj, circuit.Yrede, dados.Ypos, circuit.Y_prim, dados.barraVmed, dados.ptos_medI, dados.ptos_medV, dados.trafoList, mainset.dom);
- mainset.restriction=2 corresponds to the pointer to the function sysnlconsts_magV(x, circuit.Vmed, dados.medI_list, circuit.Vpu, circuit.Iinj, circuit.Yrede, dados.Ypos, circuit.Y_prim, dados.barraVmed, dados.ptos_medI, dados.ptos_medV, dados.trafoList, mainset.dom);
- mainset.restriction=3 corresponds to the pointer to the function sysnlconsts_trafo(x, circuit.Vmed, dados.medI_list, circuit.Vpu, circuit.Iinj, circuit.Yrede, dados.Ypos, circuit.Y_prim, dados.barraVmed, dados.ptos_medI, dados.ptos_medV, dados.trafoList, mainset.dom);
- mainset.restriction=4 corresponds to the pointer to the function sysnlconsts_V(x, circuit.Vmed, dados.medI_list, circuit.Vpu, circuit.Iinj, circuit.Yrede, dados.Ypos, circuit.Y_prim, dados.barraVmed, dados.ptos_medI, dados.ptos_medV, dados.trafoList, mainset.dom);
- mainset.restriction=5 corresponds to the pointer to the function sysnlconsts_currents(x, circuit.Vmed, dados.medI_list, circuit.Vpu, circuit.Iinj,

[2] Although the example used for the search method is pattern search, the same goes for any other search method, the option mainset.methd = 1, which uses GA (genetic algorithm) also requires these same parameters.

[3] https://www.mathworks.com/help/gads/pattern-search-options.html

[4] To learn how to configure nonlinear constraints for the patternsearch function, visit: https://www.mathworks.com/help/gads/description-of-nonlinear-constraint-solver.html

circuit.Yrede, dados.Ypos, circuit.Y_prim, dados.barraVmed, dados.ptos_medI, dados.ptos_medV, dados.trafoList, mainset.dom);

main_program/confOptions.m – confOptions(metodo, busca): receives the search method and the structure with search parameters and configures these structures with the search parameters and options used.

main_program/defineBoundaries.m – defineBoundaries(initial_guess, Root, domain): receives the vector with the initial estimate initial_guess and the vector with the actual admittance values Root and, according to the chosen search domain, returns two vectors and a string. The two vectors are the maximum and minimum search limits and the string verifies that the region between the two vectors comprises the initial estimate.

opti_ybus/sysnlconsts_<restriction>.m - sysnlconsts_<restriction>(x, Vmed, Imedlist, Vp, Inodes, Ynet, Yposition, Yprimaria, nomeVmed, n1, n2, trafList, dominio): general form of the functions in the opti_ybus folder that indicate some linear restriction to be observed. All functions use the same arguments: vector x of admittance of the circuit loads, vector Vmed of nodal voltages, list Imedlist of circuit elements with current measurement, vector Vp of modulus of nodal voltages in pu, vector Inodes of currents injected into the nodes, matrix Ynet of nodal admittance of the circuit without the loads, list Yposition with the positions of all the loads in the matrix of nodal admittances, list Yprimaria with admittance matrices of the network elements in which there is current measurement; list nomeVmed with busbars where voltage is measured, number n1 of busbars with current measurement on an adjacent element and number n2 of busbars where voltage is measured, list trafList of circuit transformers and search domain. The <restriction> field can take on the values:

- fp: power factor. In this case, for each of the admittances in the x vector, the power factor is calculated and a vector with the difference between each power factor and 0.5 is returned;
- magV: voltage magnitude. In this case, for each of the admittances of x vector, the voltage modulus at each node is calculated in pu and a vector with the difference between 1.05 and each voltage modulus at each node in pu is returned, also returning in the same vector the difference between each voltage modulus in pu at each node and 0.95;
- trafo: transformer loading. In this case, from the list of admittances, the power of each transformer in the trafoList is calculated in pu, and a vector with the difference between 1.25 and this power factor is returned;
- V: voltage measurement. In this case, from the list of admittances, the voltage is calculated at each of the nodes where it is measured. A vector with the difference between the measured and calculated voltages is returned;
- current: current measurement. In this case, from the list of admittances, the current in each network element where current is measured is calculated using its network element admittance matrix. A vector with the difference between the measured and calculated currents is returned.

b) Definition and population of files to store results

- *Definition of the name and structure of the output files*

Furthermore, at this stage, the nomenclature and location of the files populated with the results are defined using the defineFiles(mainset, dados, search) The output files are stored in a subfolder of the "log" folder, whose name structure is given as follows:

IEEE<NOME>_<DOMINIO>_<HORARIO>_v<VERSION>_mv<MV>_mI<MI>_<NC>

Where:

- <NOME>: Identifier of the simulated circuit, usually the number of bars. When it is a subsystem, it will be indicated by the number of busbars in the larger circuit plus the subsystem number. For example, IEEE13_1 is subsystem 1 of the IEEE 13-bus circuit.
- <DOMINIO>: Domain in which the Pattern Search was executed. Y for admittances and Z for impedances.
- <HORÁRIO>: String with the time at which the simulation was carried out. The string format is <ANO><MÊS><DIA><HORA><MINUTO>.
- <VERSION>: Which objective function is being run.
- <MV>: Number of points with voltage measurement, counting the phases. A three-phase bar adds 3 points with voltage measurement.
- <MI>: number of points with current measurement, counting the phases.
- <NC>: indicates whether the simulation was carried out using non-linear constraints. The use of these constraints is marked with the string 'NC'. If the file name does not contain this string, non-linear constraints do not apply.

The results folder contains the subfolders "Results S", "Results Y" and "Results Z", along with the files "Log.txt" and "variables.mat", store the results for the same test, but in the formats of apparent power of the loads, admittance of the loads and impedance of the loads, respectively.

main_program/defineFiles.m - defineFiles(confprinc, data, busca): uses the main configuration structure confprinc, the structure with circuit data data and the structure with search data search to return structured variables whose attributes are the paths and names of the files in which the results will be stored. The outputs of this function consist of four structured variables, the fields of one of which, the files [5] variable, are described below:

- files.folder - path and name of the results folder, within the 'log' folder. The folder name follows the previously explained structure;
- files.nomearqlog - path and name of the 'Log.txt' file containing the main data and results of the simulation in text mode. This file is inside the results folder;
- files.nomearqmat - name of the '.mat' file that contains the matlab variables used and generated by the simulation, for further processing if necessary. This file is also in the results folder and is called "variables.mat".

The other 3 structured variables filesS (arqS), filesY (arqY) and filesZ (arqZ) contain analogous fields and are used to contain the name and path of the result recording files, respectively, in the form of power, admittance and impedance. The fields for these variables, indicated using the "files*" notation, are described below:

- files*.nomearqfig - name of the ‘.fig’ file containing the histogram generated by the simulation;
- files*.nomearqresume - name of the '.csv' file with the summary of the load estimation carried out, containing the results and individual errors per bar;

[5] The name "files" is changed to "arq" in the function definition. The names "filesS", "filesY" and "filesZ" are changed to "arqS", "arqY" and "arqZ" respectively.

- files*.nomearqtabFreq - name of the '.csv' file containing the frequency table for the errors of the load estimation performed;
- files*.compare1 - name of the '.csv' file containing the comparison between the errors of the estimate made and the errors relating to the initial guess;
- files*.compare2 - same as arq().compare1, but using statistical figures of merit, such as maximum error, average error and standard deviation. These figures of merit are discussed later.

- *Creation and initial population of the log file*

Once the nomenclature and path of the saved files have been defined, the log files are created and populated with initial data using the function initFile(files.arqlog, mainset.version, mainset.const, dados.ptos_medSource, dados.ptos_medV, dados.node_order, dados.barraVmed, dados.elemImed, options, search.LB, search.UB, search.chute_inicial, dados.barras, search.verifyBounds, mainset.method). This function records all these data in the log file in the form of a header.

main_program/initFile.m – initFile(arqlg, ver, con, ptosS, ptosV, nos, barraV, elemI, opt, lb, ub,

init_guess, bus, verificaB, method): creates a file from the path and name to the .txt log file of the arqlg algorithm and writes in it: the ver version of the objective function that to be evaluated; the con constant for expanding the domain of the objective function; the ptosS number of measurement points on the feeder, counting the phases; the ptosV list of network elements that contain current measurement, excluding the feeder; the nos list with busbars sorted according to organize_bus; the barraV list of busbars on which there is voltage measurement; the elemI list with elements in which there is current measurement; the opt structure with the Pattern Search configuration options; the lb lower limit for the estimation algorithm; the ub upper limit for the estimation algorithm; the init_guess initial guess for the estimation algorithm; the bus number of three-phase busbars in the circuit; the verificaB indicating whether the correct value is between the roots; and the method search method used.

3.6. Execution of the pattern search function

At this stage, the patternsearch method is executed, preceded by some preparations.

The sub-steps involved in this stage are:

1. Calculation of the function to be optimized at two points: the point of initial estimation and the point with the real admittances (which supposedly should lead to a null function value). The aim of this sub-step is to provide input for subsequent evaluation of the optimization. Once these two values have been calculated, they are stored in the search.finic and search.fmin variables and written to the log files.
2. Recording the system time before applying the function in the search.inicio variable.
3. Application of patternsearch aiming to optimize the opti_ybus.m function. Initially, the arguments of the opti_ybus function are an admittance vector (independent variable) and the following variables are assumed to be constant during the optimization process: the fields Vmed, Imedido, Iinj, Yrede, Ysistema, Ypos, and Y_prim of the circuit structure; the fields ptos_medSource and ptos_medV of the data structure; and the fields const, version, and dom of the mainset structure. Thus, for each independent variable (admittance vector), the opti_ybus function uses circuit parameters to calculate the nodal voltage of the feeder and evaluate its difference from the

measured voltage, generating a real number proportional to this difference. The patternsearch optimization function tests various admittance vectors following a predefined order so that the real number generated is zero, which occurs when the admittance vector is equal to the actual admittance vector of the circuit (search.Yraiz).

4. Recording the system time at the end of the simulation in the search.fim variable;
5. Determination of the execution time from the time difference between the search.fim and search.inicio variables;
6. Recording the estimated admittance value in the resultsY.Estimado variables and converting the vector to impedance and power values, and recording them in the respective resultsZ.Estimado and resultsS.Estimado variables. The conversion to impedance values is done by taking the inverse of the estimated admittances (from the inverseZY.m function) and the conversion to power values is done using the defineS.m function, which takes as arguments the list of loads (dados.Load), the vector of nodal currents (circuit.Iinj), the admittance matrix of the network without the loads (circuit.Yrede) and the estimated admittance vector (resultsY.Estimado). The function of the resultsY, resultsZ and resultsS variables in storing the estimation results is discussed in more detail in the next section.
7. Checking whether the solution found for the optimization process violates any of the established restrictions: applying the search.nonlcon pointer to the solution found and checking, from the output, whether the value found violates any restrictions[6];

3.7. Processing, recording and storing results

a) Population of the results* structure (* = Y, Z or S)

The resume structure is populated from the defineResume function. The purpose of this structure is to store the lists that contain the test results. The fields/lists of this structure described for variable S (resultsS) are:

- id – identifier of the magnitude characterizing the result. The options are S, Y, and Z, referring to apparent power, admittance, and impedance, respectively;
- chute_inicial – identical to the search.chute_inicial variable but in matrix format with two columns, the first containing the real part and the second containing the imaginary part;
- Estimado – result of estimating loads in two columns (real part and imaginary part);
- Resume – intended to contain the estimation results and estimation errors for each load. It is generated as a table to make it easier for the user to consult. The resultsS.resume list contains the following fields (for the resultsY.resume and resultsZ.resume lists the fields are analogous):
    - Name : identification of the busbar to which the load connects;
    - Re(Sraiz): Real part of the load's true power;
    - Im(Sraiz): Imaginary part of the load's true power;
    - Re(Sest): Real part of the load's estimated power;
    - Im(Sest): Imaginary part of the load's estimated power;
    - Erro – Abs(Delta): Percentage of the modulus of the difference between the true and estimated powers in relation to the true power.

[6] To understand the type of response of a function used as a non-linear constraint, visit: https://www.mathworks.com/help/gads/description-of-nonlinear-constraint-solver.html

$$Erro\ Abs(Delta)(\%) = 100\ .\ \frac{|S_{raiz} - S_{est.}|}{|S_{raiz}|}$$

- Erro – Arg(Delta): Percentage of the difference between the arguments in relation to the true argument.

$$Erro\ Delta(Arg)(\%) = 100\ .\ \frac{\arg(S_{raiz}) - \arg(S_{est})}{\arg(S_{raiz})}$$

- Erro - Real: Percentage error in estimating the real part in relation to the true nominal active power.

$$Erro\ Real(\%) = 100\ .\ \frac{Re(S_{raiz}) - Re(S_{est})}{Re(S_{raiz})}$$

- Var9: Percentage of imaginary estimation error in relation to the true nominal reactive power.

$$Erro\ Imag(\%) = 100\ .\ \frac{Im(S_{raiz}) - Im(S_{est})}{Im(S_{raiz})}$$

- maxResume: contains statistical indices for each of the errors described above. Each of these errors is represented in a column of the list. The row indices of the list related to each of the estimation errors are:
  - MaxErro – maximum error among all circuit estimates in VA[7], considering only the numerator of the error formulas presented, without multiplying by 100;
  - MaxErroPercent – maximum percentage error among all estimates;
  - MinErro - minimum error among all circuit estimates in VA[8], considering only the numerator of the error formulas presented, without multiplying by 100;
  - MinErroPercent – minimum percentage error among all estimates;
  - MeanAvgError – average error, arithmetic mean between error modules;
  - MeanSysErrorPercent – systematic percentage error, percentage of the sum of the modules of the errors relative to the sum of the true dissipated powers;
  - stdAvgError – standard deviation associated with the MeanAvgError error;
  - cvErrorPercent – coefficient of variation associated with the average error;
  - stdSysError – standard deviation associated with the systemic percentage error;
  - mape – mean absolute percentage error, average of the modules of the percentage errors;
  - stdMape – standard deviation associated with the mape;
- tabFreq – frequency table associated with the estimation. The rows correspond to the estimation error bands, with 10 equally wide ranges between errors from 0 to 1%, 9 bands between errors from 1 to 10%, 9 bands between errors of 10 and 100%, and one band for errors greater than 100%. The columns correspond to the cumulative frequency of loads (percentage of loads in relation to the total) whose estimation error does not exceed the index determined by the row;
- hist. – stores histogram data built from the tabFreq list;

[7] Error in VA because the resultsS structure is being used as an example. If it were the resultsZ structure, the error would be in ohms.

[8] Error in VA because the resultsS structure is being used as an example. If it were the resultsZ structure, the error would be in ohms.

- resumeinic e maxResumeinic – analogous to the resume and maxResume lists, but does not use the estimation data for calculation but the initial estimate data, contained in resultsS.chute_inicial;
- compare1 – similar to the resume list, it contains, for each error field defined, the difference between the error modules obtained with the estimation performed and with the initial estimate, in order to show the evolution of the estimation;
- compare2 – similar to the maxResume list, it contains, for each defined statistical index, the difference between the modules of the index obtained with the performed estimation and the index obtained with the initial estimate, showing the evolution of the estimation.

The lists for the S variable are filled in using the defineResume(resultsS,dados) command and the other lists are filled in the same way.

main_program/defineResume.m – defineResume(resX, data): from the resX structure already containing the estimation results in terms of the quantity X, it fills in the other attributes of the resX variable as explained above.

b) Storage of lists from the results* structure in .csv files

Next, the csvGenerator function saves some of the internal tables of the results* structure in .csv files, as these are formats that can easily be imported by other applications. The function is used for each of the magnitudes used to represent the result: S, Y and Z. For magnitude S, the command is csvGenerator(resultsS, filesS) and for the other magnitudes, the command is analogous.

The log file is populated with the results. In addition to containing the results of the tables in the resume* structure, this file also contains simulation-specific results such as convergence time, number of iterations performed, and the optimized value of the function achieved by the algorithm. Much of the data is written by the function resultsFile(resultsY, resultsZ, resultsS, search.tempo, files.arqlog)

main_program\csvGenerator(resX, arqX): The function receives as arguments the resX structure and the arqX structure, which must refer to the same quantities used to express the estimation (e.g. resultsY can only be used together with filesY as arguments to the function). The function then generates the 4 .csv files explained in the definition of the "files*" structure.

main_program\resultsFile(resY, resZ, resS, time, arq): takes as arguments the three results structures in the form res*, the convergence time time and the name of the log file arq. It writes the test results to the log file and to the standard matlab output screen.

c) Histogram generation

The CreateGraph(resultsS, filesS) function is used to generate a histogram of the power results. For the other quantities, similar function calls are made.

main_program/CreateGraph.m – CreateGraph(resX, arqX): receives as arguments the resX structure and the corresponding nomenclature of the histogram file defined in arqX and generates a .png file with the histogram of the estimate, as well as displaying the histogram from matlab's standard output. The .png file is saved according to the path stored in the arqX structure.

d) Recording variables in .mat file

The main structures generated are stored in a file for further processing.