# Sparse Power Flow

Assignment 5

Analysis of Power Systems EE521

Meshal Marzooq
11491503

## Introduction

This report outlines the steps required to solve a power flow problem, presented in the case study **IEEE 14-bus test system,** by utilizing the Newton-Raphson method and keeping into consideration the Sparsity and LU Factorization. Instead of employing Crouts' Algorithm in the NR method Sparse coding is used where all reordering, i.e. **tinny 0**, **tinny 1** and **tinny 2** have been implemented in the following code. To fully demonstrate the different characteristics of each reordering, this program has been used on **IEEE 30, 57** and **118 bus systems.**

In this problem the Q limits of PV buses are illustrated by setting the maximum Q of **Bus 2** as **35 MVArs** and the minimum Q of **Bus 8** as **20 MVArs** to ensure that these two buses hit their limitations

| ***Code Section*** | **Module Name** | **Description** |
|---|---|---|
| *1* | NR_withoutorder.m | • Main code that used to:<br>- to parse the data from a text file such that it can be used inside MATLAB.<br>- to calculates the Admittance Matrix to calculates the Jacobian Matrix by Crout's algorithm without ordering. |
| *2* | NR_Main_Ordering0.m | • Main code that used to:<br>- to parse the data from a text file such that it can be used inside MATLAB.<br>- to calculates the Admittance Matrix to calculates the Jacobian Matrix by Crout's algorithm with ordering 0. |
| *3* | NR_Main_Ordering1.m | • Main code that used to:<br>- to parse the data from a text file such that it can be used inside MATLAB.<br>- to calculates the Admittance Matrix to calculates the Jacobian Matrix by Crout's algorithm with ordering 1. |

| | | |
|---|---|---|
| 4 | NR_Main_Ordering2.m | • Main code that used to:<br>- to parse the data from a text file such that it can be used inside MATLAB.<br>- to calculates the Admittance Matrix to calculates the Jacobian Matrix by Crout’s algorithm with ordering 2. |
| 5 | LU_linked_list.m | • The main purpose of this function is to use searching and adding functions to create Q matrix, and then store the non-zeros in new Q linked list representation. |
| 6 | FB_linked_list.m | • The function is applied after getting the court’s linked list and calculate the solution vector x using LU factorization with forward and backward substitution. |
| 7 | store_in_linked_list.m | • This function transforms the Jacobian Matrix into a linked list representation. |
| 8 | search_in_linked_list.m | • The function is used to find the element a(i, j) in the liked list. The main logic to find the value by checking if ‘NRow’ equals to ‘i’ while ‘NCol’ equals to ‘j’. |

| | | |
|---|---|---|
| *9* | add_in_linked_list.m | • The function is used to find the element a(i, j) in the liked list. The main logic to add the new value aij at the position if ‘NRow’ equals to ‘i’ while ‘NCol’ equals to ‘j’. |
| *10* | ordering_scheme_tinney0.m | • This function performs the reordering by **tinny 0** scheme |
| *11* | ordering_scheme_tinney1.m | • This function performs the reordering by **tinny 1** scheme |
| *12* | ordering_scheme_tinney2.m | • This function performs the reordering by **tinny 2** scheme |

In addition to elaborate comments inside the MATLAB files, the code is described here with in great detail followed by the results obtained.

**General Idea:**

- NR_withoutorder.m
- NR_Main_Ordering0.m
- NR_Main_Ordering1.m
- NR_Main_Ordering2.m

These functions ((in different orderings)) are tasked with preparing the input in order to calculate the Admittance and the Jacobian Matrix by applying several steps outlined in the Newton-Raphson matrix to solve the power flow problem. The first step is to open and read **ieee14BusesData.txt and ieee14BranchesData.txt** which hold the data for buses and branches respectively which are subsequently displayed. In the next step, some important data is extracted from the aforementioned matrices and stored in separate variables to be used by **Yij,** which returns the **Admittance Matrix** for the network, which is then displayed. In the next steps, techniques outlined in the Newton-

Raphson Matrix algorithm are implemented. First, conversion of MW and MVAr to per unit counterparts by assuming base MVA = 100 is carried out followed by calculation of scheduled values of injected active and reactive power of each bus by using the formula:

$$Scheduled\ Value = \frac{Generated\ Power - Load\ Power}{base\ MVA}$$

Then a flat starting approach is adopted i.e. the voltage of PV buses are initialized with their desired values and of all slack and PQ buses are one and the angles of all buses are considered to be 0. The iteration is performed inside a while loop which is scheduled to stop when the error of the absolute value of the mismatch vector reaches its maximum at 0.01. Inside each loop, the injective active and reactive power of each are calculated using the updated voltage magnitude and angles. Since the scheduled injective active power is not available for the slack bus, it is omitted from the calculations. Furthermore, the scheduled injected reactive power is also unavailable for the slack bus and PV buses, hence it is only calculated for the PQ buses. In the end these calculated values are compared with the scheduled values to compute the mismatch vector which is used to check if the while loop should be stopped or not.

The next step is to calculate the Jacobian Matrix and those values are calculated that are to be added to the voltage magnitudes and the angles by using the formula:

$$\begin{bmatrix} \Delta\delta_i \\ \frac{\Delta V_i}{|V_i|} \end{bmatrix} = [J]^{-1} \begin{bmatrix} \Delta P_i \\ \Delta Q_i \end{bmatrix} \tag{1}$$

This equation is calculated by the **crout_algorithm.m** function and the voltage magnitudes and angles are adjusted after calculating the error vector in the above equation.

In the end the results are displayed. The next step requires the calculation of the injected reactive power of each PV bus. First it is assumed that the voltage magnitude of the PV buses is fixed at a set value. To maintain this value, it is assumed that the reactive power of the PV bus could be anything. But since each bus has a range for allowable generating and absorbing reactive power, if the required reactive power is not within this range of a PV bus, its voltage magnitude cannot be kept at the required value and it is switched to a PQ bus. Similarly, if the calculated generated reactive power of a PV bus which is calculated as the summation of injected reactive power and reactive load, is outside the range, then the generated reactive power is set minimum/maximum limit and the voltage magnitude value at the bus is relaxed, and the bus is then treated as a PQ bus for which the voltage is updated at each iteration.

**Explaining the code:**

- NR_withoutorder.m
- NR_Main_Ordering0.m
- NR_Main_Ordering1.m
- NR_Main_Ordering2.m

*Open and read the txt file- Codes*

*Please look at Appendix A*

*Please look at Appendix B*

*Please look at Appendix C*

*Please look at Appendix D*

To parse the data from a text file. The matrices are initialized with several vectors containing zero values. These matrices are then overwritten by the data read from the text file. The data is read one line at a time where each line has multiple columns, each corresponding to a vector.

*Calculating the Admittance Matrix -codes*

*Please look at Appendix A*

*Please look at Appendix B*

*Please look at Appendix C*

*Please look at Appendix D*

To calculate the elements of the admittance matrix. It takes the outputs of previous section as inputs and works on then to return the admittance matrix. In the first step, the code parses the input to retrieve required information such as number of lines, number of buses, and lines' information such as resistance, reactance, susceptance etc. In the second step, impedances and admittances are calculated for all lines.

The admittance matrix has two type of elements:

- Off-diagonal Elements: These are first replaced by the negative version of the admittance of each branch. If, however, a transformer exists on the branch then this value is divided by the conjugate of the transformer's ratio in primary side and by the transformer's ratio in the secondary side.
- Diagonal Elements: These are calculated by adding the admittances of all branches which are connected to the corresponding bus. If there is a tap changer of corresponding transformer, then the admittances connected to the transformer are divided by the square of the absolute value of the turns ratio of the said transformer. Since the $\pi\pi$ model is used, the line charging capacitance divided by 2 is added to the corresponding diagonal elements.

The admittance matrix depicts the actual admittance matrix of the network because effect of transformer final turns ratios is taken into account.

*Calculating the Jacobian Matrix – Codes*

*Please look at Appendix A*

*Please look at Appendix B*

*Please look at Appendix C*

*Please look at Appendix D*

The main task is to calculate the Jacobian matrix. It takes the output of the previous section i.e. admittance matrix, as input along with voltage magnitudes and angles, slack and PV buses. The voltage magnitudes of the PV buses i.e. 2, 3, 6, and 8 are fixed at their desired values but as the voltage magnitude and angle of the slack bus, i.e. bus 1, is fixed at the desired values, it is omitted from the calculation of injected active and reactive power. This assumption applies to all slack buses if multiple exist. Also, the scheduled reactive power of the PV buses is unknown so only injected active power can be calculated for the PV buses. Therefore:

- Each PV bus has one equation (active). A total of 4 PV buses yield 4 equations
- Each PQ bus has two equations (active and reactive). A total of 9 PQ buses yield 18 equations

This means that there are total 22 equations and hence the Jacobian Matrix has 22 rows.

Similarly, those columns from the Jacobian Matrix are excluded which:

- Are derivative of P and Q equations with these voltage magnitudes because voltage magnitudes of PV buses are known.
- Are derivative of P and Q equations with these voltage magnitudes and angle of the slack bus are removed because the voltage magnitude and angle of the slack bus is known.

This means that the dimensions of the Jacobian Matrix is 22x22.

The equation (9.68) in the book use to calculate the Jacobian matrix [1]. The Jacobian matrix consists of four sub-matrices named as **J11**, **J12**, **J21**, and **J22**. These four submatrices are derivatives of Ps with voltage angles, derivatives of Ps with voltage magnitudes multiplied by voltage magnitudes, derivatives of Qs with voltage angles, and derivatives of Qs with voltage magnitudes multiplied by voltage magnitudes, respectively. Equations (9.64) - (9-68) [1] are used to calculate the sub-matrices. Each function is responsible to different schemes.

- NR_withoutorder.m ((Normal Newton-Raphson without ordering))
- NR_Main_Ordering0.m ((Newton-Raphson – Tinny 0))
- NR_Main_Ordering1.m ((Newton-Raphson – Tinny 1))
- NR_Main_Ordering2.m ((Newton-Raphson – Tinny 2))

*Utilizing LU Factorization and Crout's Algorithm to Solve Equation (1)* - *Codes*

*Please look at Appendix E and F*

Earlier, to obtain the inverse of the Jacobian Matrix built-in function or Crout's Algorithm inside MATLAB was used but now the LU factorization of the sparse form of equations will be used. First, the one of four main functions **NR_withoutorder.m, NR_Main_Ordering0.m, NR_Main_Ordering1.m, or NR_Main_Ordering2.m** is called which calls all the subsequent functions. In the first step the Jacobain matrix is transformed into a table in **store_in_linked_list.m**. This table is then subjected to reordering by one of the three schemes: **tinny 0**, **tinny 1** and t**inny 2**. This choice is left up to the user and one of **ordering_scheme_tinney0.m, ordering_scheme_tinney0.m,** and **ordering_scheme_tinney0.m** is called respectively**.** If comparison is required, then all can be called consecutively. After the reordering step, **LU_linked_list.m** is called to perform LU Factorization. The inputs of the function are reordered Jacobian matrix and mismatch vector and the outputs are the results after solving the equation, Q matrix, Alpha, Beta, and number of fills.

*Converting Jacobian matrix to a table – Codes*

*Please look at Appendix G, H and I*

To convert the Jacobian matrix to table form, the function **store_in_linked_list.m** extracts the following information for each non-zero element of the matrix and outputs a structure:

- NROW: Row Number
- NCOL: Column Number
- NIR: Next in Row
- NIC: Next in Column
- FIR: First in Row
- FIC: First in column

*Reordering by Tinny 0 – Code*

*Please look at Appendix J*

The function used is **ordering_scheme_tinney0.m**. The inputs of this function are the Jacobian Table and the mismatch vector and the outputs are the ordered form of the inputs and the final order. The steps of the reordering are defined below:

Step 1: Calculate degree of nodes

Step 2: Switching the Rows and then Columns (As there is no matrix, this is done by switching their NROW and NCOL variable)

Step 3: The node with the lowest degree is to be at the first row, hence it is switched.

Step 4: NIC of the elements of the rows and the previous rows are modified

Step 5: Then the corresponding columns are switched

Step 6: NIR, NCOL, FIR and FIC are then adjusted.

*Reordering by Tinny 1 – Code*

*Please look at Appendix K*

The function used is **ordering_scheme_tinney1.m**. The steps of the reordering are defined below:

Step 1: Calculate degree of nodes

Step 2: Iterative algorithm to remove nodes with the lowest degree by locating the nodes connected to nodes that are to be removed. The degrees of these connected nodes are decremented. If no connection existed before the degrees are incremented and the row and column eliminated for those nodes which still have a connection.

Step 3: After this, the mismatch vector and vector and the Jacobian Table is reordered like in **tinny 0.**

*Reordering by Tinny 2 – Code*

*Please look at Appendix L*

The function used is **ordering_scheme_tinney2.m**. Similar to **tinny 1**, the degrees of nodes are calculated in the first step and an iterative algorithm is used to determine the final orders. The most important factor here is the number of fill at each step which are used along with the degree of nodes to determine the final order.

## Results

In this section, the results of the above code i.e. **Admittance Matrix**, **Jacobian Matrix**, **Voltage Magnitudes & Angles** and **Injected Active & Reactive Powers** of each bus at each iteration is represented. Moreover, the reordering results by each scheme and non-zero elements of Q matrix of the Jacobian matrix of different IEEE systems are illustrated.

- **IEEE 14 Bus System**

| | Without Reordering | Tinny 0 | Tinny 1 | Tinny 2 |
|---|---|---|---|---|
| Alpha | **1726** | **338** | **320** | **320** |
| Beta | **348** | **166** | **162** | **162** |
| Total Additional Calculation | **2074** | **504** | **482** | **482** |
| Number of Fills | **202** | **20** | **16** | **16** |

- **IEEE 30 Bus System**

| | Without Reordering | Tinny 0 | Tinny 1 | Tinny 2 |
|---|---|---|---|---|
| Alpha | **17208** | **1500** | **1166** | **1166** |
| Beta | **1721** | **531** | **479** | **479** |
| Total Additional Calculation | **18929** | **2031** | **1645** | **1645** |
| Number of Fills | **1342** | **152** | **100** | **100** |

- **IEEE 57 Bus System**

| | Without Reordering | Tinny 0 | Tinny 1 | Tinny 2 |
|---|---|---|---|---|
| Alpha | **142984** | **4972** | **3028** | **2984** |
| Beta | **6906** | **1330** | **1088** | **1080** |
| Total Additional Calculation | **149890** | **6302** | **4116** | **4064** |
| Number of Fills | **6188** | **612** | **370** | **362** |

- **IEEE 118 Bus System**

| | Without Reordering | Tinny 0 | Tinny 1 | Tinny 2 |
|---|---|---|---|---|
| Alpha | **388637** | **5324** | **2602** | **2516** |
| Beta | **14848** | **1705** | **1330** | **1309** |
| Total Additional Calculation | **403485** | **7029** | **3932** | **3825** |
| Number of Fills | **13797** | **654** | **279** | **258** |

The admittance matrix:

| Bus | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| **1** | 6.025 – 19447i | -4.9991 + 15.263i | 0 | 0 | -1.0259 + 4.235i |
| **2** | -4.9991 + 15.263i | 9.5213 – 30.272i | -1.135 + 4.7819i | -1.686 + 5.1158i | -1.7011 + 5.1939i |
| **3** | 0 | -1.135 + 4.7819i | 3.121 – 9.8224i | -1.986 + 5.0688i | 0 |
| **4** | 0 | -1.686 + 5.1158i | -1.986 + 5.0688i | 10.513 – 38.654i | -6.841 + 21.579i |
| **5** | -1.0259 + 4.235i | -1.7011 + 5.1939i | 0 | -6.841 + 21.579i | 9.568 – 35.534i |
| **6** | 0 | 0 | 0 | 0 | 4.2574i |
| **7** | 0 | 0 | 0 | 4.8895i | 0 |
| **8** | 0 | 0 | 0 | 0 | 0 |
| **9** | 0 | 0 | 0 | 1.8555i | 0 |
| **10** | 0 | 0 | 0 | 0 | 0 |
| **11** | 0 | 0 | 0 | 0 | 0 |
| **12** | 0 | 0 | 0 | 0 | 0 |
| **13** | 0 | 0 | 0 | 0 | 0 |
| **14** | 0 | 0 | 0 | 0 | 0 |

| Bus | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|
| **1** | 0 | 0 | 0 | 0 | 0 |
| **2** | 0 | 0 | 0 | 0 | 0 |
| **3** | 0 | 0 | 0 | 0 | 0 |
| **4** | 0 | 4.8895i | 0 | 1.8555i | 0 |
| **5** | 4.2574i | 0 | 0 | 0 | 0 |
| **6** | 6.5799 – 17.341i | 0 | 0 | 0 | 0 |
| **7** | 0 | -19.549i | 5.677i | 9.0901i | 0 |
| **8** | 0 | 5.677i | -5.677i | 0 | 0 |
| **9** | 0 | 9.0901i | 0 | 5.3261 – 24.093i | -3.902 + 10.365i |
| **10** | 0 | 0 | 0 | -3.902 + 10.365i | 5.7829 – 14.768i |
| **11** | -1.955 + 4.0941i | 0 | 0 | 0 | -1.8809 + 4.4029i |
| **12** | -1.526 + 3.176i | 0 | 0 | 0 | 0 |
| **13** | -3.0989 + 6.1028i | 0 | 0 | 0 | 0 |
| **14** | 0 | 0 | 0 | -1.424 + 3.0291i | 0 |

| Bus | 11 | 12 | 13 | 14 |
|---|---|---|---|---|
| **1** | 0 | 0 | 0 | 0 |
| **2** | 0 | 0 | 0 | 0 |
| **3** | 0 | 0 | 0 | 0 |
| **4** | 0 | 0 | 0 | 0 |
| **5** | 0 | 0 | 0 | 0 |
| **6** | -1.955 + 4.0941i | -1.526 + 3.176i | -3.0989 + 6.1028i | 0 |
| **7** | 0 | 0 | 0 | 0 |
| **8** | 0 | 0 | 0 | 0 |
| **9** | 0 | 0 | 0 | -1.424 + 3.0291i |
| **10** | -1.8809 + 4.4029i | 0 | 0 | 0 |
| **11** | 3.8359 – 8.497i | 0 | 0 | 0 |
| **12** | 0 | 4.015 – 5.4279i | -2.489 + 2.252i | 0 |
| **13** | 0 | -2.489 + 2.252i | 6.7249 – 10.67i | -1.137 + 2.315i |
| **14** | 0 | 0 | -1.137 + 2.315i | 2.561 – 5.344i |

Jacobian matrix for 1st iteration:

| Bus | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| **1** | 32.728 | -5.047 | -5.3461 | -5.4277 | 0 | 0 | 0 | 0 | 0 |
| **2** | -5.047 | 10.167 | -5.1195 | 0 | 0 | 0 | 0 | 0 | 0 |
| **3** | -5.3461 | -5.1195 | 38.789 | -21.579 | 0 | -4.8895 | 0 | -1.8555 | 0 |
| **4** | -5.4277 | 0 | -21.579 | 36.051 | -4.5555 | 0 | 0 | 0 | 0 |
| **5** | 0 | 0 | 0 | -4.5555 | 18.864 | 0 | 0 | 0 | 0 |
| **6** | 0 | 0 | -4.8895 | 0 | 0 | 20.168 | -6.1879 | -9.0901 | 0 |
| **7** | 0 | 0 | 0 | 0 | 0 | -6.1879 | 6.1879 | 0 | 0 |
| **8** | 0 | 0 | -1.8555 | 0 | 0 | -9.0901 | 0 | 24.34 | -10.365 |
| **9** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -10.365 | 14.768 |
| **10** | 0 | 0 | 0 | 0 | -4.3807 | 0 | 0 | 0 | -4.4029 |
| **11** | 0 | 0 | 0 | 0 | -3.3983 | 0 | 0 | 0 | 0 |
| **12** | 0 | 0 | 0 | 0 | -6.5299 | 0 | 0 | 0 | 0 |
| **13** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -3.0291 | 0 |
| **14** | 1.7619 | 2.0058 | -10.609 | 6.841 | 0 | 0 | 0 | 0 | 0 |
| **15** | 1.7777 | 0 | 6.841 | -9.7061 | 0 | 0 | 0 | 0 | 0 |
| **16** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **17** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -5.3261 | 3.902 |
| **18** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.902 | -5.7829 |
| **19** | 0 | 0 | 0 | 0 | 2.0919 | 0 | 0 | 0 | 1.8809 |
| **20** | 0 | 0 | 0 | 0 | 1.6328 | 0 | 0 | 0 | 0 |
| **21** | 0 | 0 | 0 | 0 | 3.3159 | 0 | 0 | 0 | 0 |
| **22** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.424 | 0 |

| Bus | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
|---|---|---|---|---|---|---|---|---|---|
| **1** | 0 | 0 | 0 | 0 | 0 | -1.7619 | -1.7777 | 0 | 0 |
| **2** | 0 | 0 | 0 | 0 | 0 | -2.0058 | 0 | 0 | 0 |
| **3** | 0 | 0 | 0 | 0 | 0 | 10.417 | -6.841 | 0 | 0 |
| **4** | 0 | 0 | 0 | 0 | 0 | -6.841 | 9.4299 | 0 | 0 |
| **5** | 0 | -4.3807 | -3.3983 | -6.5299 | 0 | 0 | 0 | 0 | 0 |
| **6** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **7** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **8** | -10.365 | 0 | 0 | 0 | -3.0291 | 0 | 0 | 0 | 5.3261 |
| **9** | 14.768 | -4.4029 | 0 | 0 | 0 | 0 | 0 | 0 | -3.902 |
| **10** | -4.4029 | 8.7836 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **11** | 0 | 0 | 5.6503 | -2.252 | 0 | 0 | 0 | 0 | 0 |
| **12** | 0 | 0 | -2.252 | 11.097 | -2.315 | 0 | 0 | 0 | 0 |
| **13** | 0 | 0 | 0 | -2.315 | 5.344 | 0 | 0 | 0 | -1.424 |
| **14** | 0 | 0 | 0 | 0 | 0 | 38.519 | -21.579 | -4.8895 | -1.8555 |
| **15** | 0 | 0 | 0 | 0 | 0 | -21.579 | 35.017 | 0 | 0 |
| **16** | 0 | 0 | 0 | 0 | 0 | -4.8895 | 0 | 18.931 | -9.0901 |
| **17** | 3.902 | 0 | 0 | 0 | 1.424 | -1.8555 | 0 | -9.0901 | 23.845 |
| **18** | -5.7829 | 1.8809 | 0 | 0 | 0 | 0 | 0 | 0 | -10.365 |
| **19** | 1.8809 | -3.9728 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **20** | 0 | 0 | -4.1218 | 2.489 | 0 | 0 | 0 | 0 | 0 |
| **21** | 0 | 0 | 2.489 | -6.9419 | 1.137 | 0 | 0 | 0 | 0 |
| **22** | 0 | 0 | 0 | 1.137 | -2.561 | 0 | 0 | 0 | -3.0291 |

| Bus | 18 | 19 | 20 | 21 | 22 |
|---|---|---|---|---|---|
| **1** | 0 | 0 | 0 | 0 | 0 |
| **2** | 0 | 0 | 0 | 0 | 0 |
| **3** | 0 | 0 | 0 | 0 | 0 |
| **4** | 0 | 0 | 0 | 0 | 0 |
| **5** | 0 | -2.0919 | -1.6328 | -3.3159 | 0 |
| **6** | 0 | 0 | 0 | 0 | 0 |
| **7** | 0 | 0 | 0 | 0 | 0 |
| **8** | -3.902 | 0 | 0 | 0 | -1.424 |
| **9** | 5.7829 | -1.8809 | 0 | 0 | 0 |
| **10** | -1.8809 | 3.6991 | 0 | 0 | 0 |
| **11** | 0 | 0 | 3.9082 | -2.489 | 0 |
| **12** | 0 | 0 | -2.489 | 6.508 | -1.137 |
| **13** | 0 | 0 | 0 | -1.137 | 2.561 |
| **14** | 0 | 0 | 0 | 0 | 0 |
| **15** | 0 | 0 | 0 | 0 | 0 |
| **16** | 0 | 0 | 0 | 0 | 0 |
| **17** | -10.365 | 0 | 0 | 0 | -3.0291 |
| **18** | 14.768 | -4.4029 | 0 | 0 | 0 |
| **19** | -4.4029 | 8.2104 | 0 | 0 | 0 |
| **20** | 0 | 0 | 5.2056 | -2.252 | 0 |
| **21** | 0 | 0 | -2.252 | 10.243 | -2.315 |
| **22** | 0 | 0 | 0 | -2.315 | 5.344 |

Jacobian matrix for 2$^{nd}$ iteration:

| Bus | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| **1** | 32.9953 | -5.1612 | -5.6715 | -5.7646 | 0 | 0 | 0 | 0 | 0 |
| **2** | -5.047 | 10.0532 | -5.2092 | 0 | 0 | 0 | 0 | 0 | 0 |
| **3** | -5.3209 | -5.3596 | 40.7727 | -22.9975 | 0 | -5.1829 | 0 | -1.9119 | 0 |
| **4** | -5.5008 | 0 | -23.3666 | 37.7044 | -4.3937 | 0 | 0 | 0 | 0 |
| **5** | 0 | 0 | 0 | -4.3937 | 19.5320 | 0 | 0 | 0 | 0 |
| **6** | 0 | 0 | -5.1829 | 0 | 0 | 21.5534 | -6.5034 | -9.8671 | 0 |
| **7** | 0 | 0 | 0 | 0 | 0 | -6.5034 | 6.5034 | 0 | 0 |
| **8** | 0 | 0 | -1.9119 | 0 | 0 | -9.8671 | 0 | 26.0859 | -11.0751 |
| **9** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -11.0427 | 15.8044 |
| **10** | 0 | 0 | 0 | 0 | -4.5781 | 0 | 0 | 0 | -4.7764 |
| **11** | 0 | 0 | 0 | 0 | -3.5629 | 0 | 0 | 0 | 0 |
| **12** | 0 | 0 | 0 | 0 | -6.7961 | 0 | 0 | 0 | 0 |
| **13** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -2.4953 | 0 |
| **14** | 2.3433 | 1.8785 | -11.6969 | 7.9214 | 0 | 0 | 0 | 0 | 0 |
| **15** | 2.2475 | 0 | 6.7672 | -10.3894 | -0.4560 | 0 | 0 | 0 | 0 |
| **16** | 0 | 0 | 0.2918 | 0 | 0 | 0.0002 | 0 | -0.2920 | 0 |
| **17** | 0 | 0 | 0.1645 | 0 | 0 | 0.2920 | 0 | -6.0071 | 4.1201 |
| **18** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4.2061 | -6.2606 |
| **19** | 0 | 0 | 0 | 0 | 2.2302 | 0 | 0 | 0 | 2.0200 |
| **20** | 0 | 0 | 0 | 0 | 1.7808 | 0 | 0 | 0 | 0 |
| **21** | 0 | 0 | 0 | 0 | 3.5911 | 0 | 0 | 0 | 0 |
| **22** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.5760 | 0 |

| Bus | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
|---|---|---|---|---|---|---|---|---|
| **1** | 0 | 0 | 0 | 0 | -1.2388 | -1.3862 | 0 | 0 |
| **2** | 0 | 0 | 0 | 0 | -2.1903 | 0 | 0 | 0 |
| **3** | 0 | 0 | 0 | 0 | 10.3931 | -7.6232 | 0.2776 | 0.1592 |
| **4** | 0 | 0 | 0 | 0 | -6.5517 | 9.9239 | 0 | 0 |
| **5** | -4.6123 | -3.6167 | -6.9092 | 0 | 0 | -0.4382 | 0.0002 | 0 |
| **6** | 0 | 0 | 0 | 0 | -0.2825 | 0 | 0 | 0.2826 |
| **7** | 0 | 0 | 0 | 0 | 0 | 0 | -0.2778 | 0 |
| **8** | 0 | 0 | 0 | -3.2319 | -0.1592 | 0 | 0 | 5.1930 |
| **9** | -4.7617 | 0 | 0 | 0 | 0 | 0 | 0 | -4.0707 |
| **10** | 9.3545 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **11** | 0 | 6.0617 | -2.4989 | 0 | 0 | 0 | 0 | 0 |
| **12** | 0 | -2.4953 | 11.7913 | -2.4998 | 0 | 0 | 0 | 0 |
| **13** | 0 | 0 | -2.4656 | 5.6290 | 0 | 0 | 0 | -1.5253 |
| **14** | 0 | 0 | 0 | 0 | 39.6859 | -22.1039 | -4.9314 | -1.8503 |
| **15** | 0 | 0 | 0 | 0 | -22.6225 | 36.4520 | 0 | 0 |
| **16** | 0 | 0 | 0 | 0 | -5.0178 | 0 | 20.5837 | -9.5494 |
| **17** | 0 | 0 | 0 | 1.4305 | -1.8510 | 0 | -9.3885 | 24.9352 |
| **18** | 2.0545 | 0 | 0 | 0 | 0 | 0 | 0 | -10.6871 |
| **19** | -4.2502 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **20** | 0 | -4.5392 | 2.7583 | 0 | 0 | 0 | 0 | 0 |
| **21** | 0 | 2.7616 | -7.5372 | 1.1846 | 0 | 0 | 0 | 0 |
| **22** | 0 | 0 | 1.2542 | -2.8303 | 0 | 0 | 0 | -3.0615 |

| Bus | 18 | 19 | 20 | 21 | 22 |
|---|---|---|---|---|---|
| **1** | 0 | 0 | 0 | 0 | 0 |
| **2** | 0 | 0 | 0 | 0 | 0 |
| **3** | 0 | 0 | 0 | 0 | 0 |
| **4** | 0 | 0 | 0 | 0 | 0 |
| **5** | 0 | -2.05777 | -1.5796 | -3.2093 | 0 |
| **6** | 0 | 0 | 0 | 0 | 0 |
| **7** | 0 | 0 | 0 | 0 | 0 |
| **8** | -3.9902 | 0 | 0 | 0 | -1.3998 |
| **9** | 5.8792 | -1.9585 | 0 | 0 | 0 |
| **10** | -1.9564 | 3.9962 | 0 | 0 | 0 |
| **11** | 0 | 0 | 4.1871 | -2.6281 | 0 |
| **12** | 0 | 0 | -2.6139 | 6.9352 | -1.1591 |
| **13** | 0 | 0 | 0 | -1.1950 | 2.4647 |
| **14** | 0 | 0 | 0 | 0 | 0 |
| **15** | 0 | 0 | 0 | 0 | 0 |
| **16** | 0 | 0 | 0 | 0 | 0 |
| **17** | -10.7259 | 0 | 0 | 0 | -3.1625 |
| **18** | 15.1920 | -4.5392 | 0 | 0 | 0 |
| **19** | -4.6258 | 8.9095 | 0 | 0 | 0 |
| **20** | 0 | 0 | 5.7314 | -2.3809 | 0 |
| **21** | 0 | 0 | -2.3619 | 11.1625 | -2.4462 |
| **22** | 0 | 0 | 0 | -2.3492 | 5.4141 |

Jacobian matrix for 3rd iteration:

| Bus | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| **1** | 32.8782 | -5.1618 | -5.6318 | -5.7191 | 0 | 0 | 0 | 0 | 0 |
| **2** | -4.8418 | 10.0097 | -5.1679 | 0 | 0 | 0 | 0 | 0 | 0 |
| **3** | -5.2910 | -5.3306 | 40.3103 | -22.6794 | 0 | -5.1197 | 0 | -1.8897 | 0 |
| **4** | -5.4636 | 0 | -23.0385 | 37.2678 | -4.3609 | 0 | 0 | 0 | 0 |
| **5** | 0 | 0 | 0 | -4.3609 | 19.4378 | 0 | 0 | 0 | 0 |
| **6** | 0 | 0 | -5.1197 | 0 | 0 | 21.3419 | -6.4655 | -9.7568 | 0 |
| **7** | 0 | 0 | 0 | 0 | 0 | -6.4655 | 6.4655 | 0 | 0 |
| **8** | 0 | 0 | -1.8897 | 0 | 0 | -9.7568 | 0 | 25.8109 | -10.9637 |
| **9** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -10.9281 | 15.6522 |
| **10** | 0 | 0 | 0 | 0 | -4.5648 | 0 | 0 | 0 | -4.7316 |
| **11** | 0 | 0 | 0 | 0 | -3.5529 | 0 | 0 | 0 | 0 |
| **12** | 0 | 0 | 0 | 0 | -6.7792 | 0 | 0 | 0 | 0 |
| **13** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -3.1320 | 0 |
| **14** | 2.4820 | 1.8490 | -11.5479 | 7.8133 | 0 | -0.2760 | 0 | -0.01555 | 0 |
| **15** | 2.2213 | 0 | 6.6805 | -10.2788 | -0.4531 | 0 | 0 | 0 | 0 |
| **16** | 0 | 0 | 0.2760 | 0 | 0 | 0 | 0 | -0.2760 | 0 |
| **17** | 0 | 0 | 0.1555 | 0 | 0 | 0.2760 | 0 | -5.9202 | 4.0733 |
| **18** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4.1679 | -6.1963 |
| **19** | 0 | 0 | 0 | 0 | 2.2127 | 0 | 0 | 0 | 2.0200 |
| **20** | 0 | 0 | 0 | 0 | 1.7717 | 0 | 0 | 0 | 0 |
| **21** | 0 | 0 | 0 | 0 | 3.5713 | 0 | 0 | 0 | 0 |
| **22** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.5617 | 0 |

| Bus | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
|---|---|---|---|---|---|---|---|---|
| **1** | 0 | 0 | 0 | 0 | -1.2501 | -1.3957 | 0 | 0 |
| **2** | 0 | 0 | 0 | 0 | -2.2066 | 0 | 0 | 0 |
| **3** | 0 | 0 | 0 | 0 | 10.3221 | -7.5662 | 0.2641 | 0.1514 |
| **4** | 0 | 0 | 0 | 0 | -6.5103 | 9.8073 | 0 | 0 |
| **5** | -4.5904 | -3.6033 | -6.8832 | 0 | 0 | -0.4387 | 0 | 0 |
| **6** | 0 | 0 | 0 | 0 | -0.2689 | 0 | 0 | 0.2684 |
| **7** | 0 | 0 | 0 | 0 | 0 | 0 | -0.2641 | 0 |
| **8** | 0 | 0 | 0 | -3.2008 | -0.1516 | 0 | 0 | 5.1861 |
| **9** | -4.7241 | 0 | 0 | 0 | 0 | 0 | 0 | -4.0557 |
| **10** | 9.2964 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **11** | 0 | 6.0349 | -2.4820 | 0 | 0 | 0 | 0 | 0 |
| **12** | 0 | -2.4802 | 11.7374 | -2.4781 | 0 | 0 | 0 | 0 |
| **13** | 0 | 0 | -2.4503 | 5.5823 | 0 | 0 | 0 | -1.5197 |
| **14** | 0 | 0 | 0 | 0 | 39.3602 | -21.9621 | -4.8999 | -1.8388 |
| **15** | 0 | 0 | 0 | 0 | -22.4515 | 36.0599 | 0 | 0 |
| **16** | 0 | 0 | 0 | 0 | -4.9892 | 0 | 20.4262 | -9.4940 |
| **17** | 0 | 0 | 0 | 1.4154 | -1.8415 | 0 | -9.3379 | 24.7930 |
| **18** | 2.0284 | 0 | 0 | 0 | 0 | 0 | 0 | -10.6338 |
| **19** | -4.2236 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **20** | 0 | -4.5131 | 2.7414 | 0 | 0 | 0 | 0 | 0 |
| **21** | 0 | 2.7431 | -7.4963 | 1.1820 | 0 | 0 | 0 | 0 |
| **22** | 0 | 0 | 1.2386 | -2.8303 | 0 | 0 | 0 | -3.0477 |

| Bus | 18 | 19 | 20 | 21 | 22 |
|---|---|---|---|---|---|
| **1** | 0 | 0 | 0 | 0 | 0 |
| **2** | 0 | 0 | 0 | 0 | 0 |
| **3** | 0 | 0 | 0 | 0 | 0 |
| **4** | 0 | 0 | 0 | 0 | 0 |
| **5** | 0 | -2.0662 | -1.5828 | -3.2175 | 0 |
| **6** | 0 | 0 | 0 | 0 | 0 |
| **7** | 0 | 0 | 0 | 0 | 0 |
| **8** | -3.9639 | 0 | 0 | 0 | -1.3911 |
| **9** | 5.8548 | -1.9411 | 0 | 0 | 0 |
| **10** | -1.9569 | 3.9750 | 0 | 0 | 0 |
| **11** | 0 | 0 | 4.1700 | -2.6202 | 0 |
| **12** | 0 | 0 | -2.6049 | 6.9070 | -1.1617 |
| **13** | 0 | 0 | 0 | -1.1838 | 2.4593 |
| **14** | 0 | 0 | 0 | 0 | 0 |
| **15** | 0 | 0 | 0 | 0 | 0 |
| **16** | 0 | 0 | 0 | 0 | 0 |
| **17** | -10.6695 | 0 | 0 | 0 | -3.1458 |
| **18** | 15.1192 | -4.5208 | 0 | 0 | 0 |
| **19** | -4.6046 | 8.8619 | 0 | 0 | 0 |
| **20** | 0 | 0 | 5.7006 | -2.3723 | 0 |
| **21** | 0 | 0 | -2.3553 | 11.1078 | -2.4355 |
| **22** | 0 | 0 | 0 | -2.3420 | 5.3882 |

Final Results

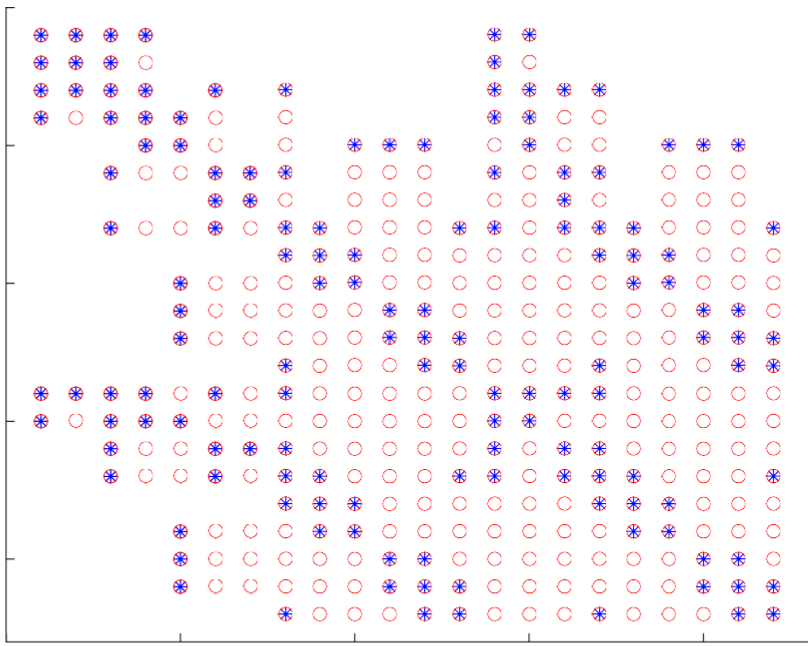
| Bus | V | Angle | Injection | | Generation | | Load | |
|---|---|---|---|---|---|---|---|---|
| | **(P.U.)** | **(Degrees)** | **(MW)** | **(MVAr)** | **(MW)** | **(MVAr)** | **(MW)** | (MVAr) |
| 1 | 1.0600 | 0.00 | 232.393 | -16.549 | 232.393 | -16.549 | 0.000 | 0.000 |
| 2 | 1.0450 | -4.98 | 18.300 | 30.857 | 40.000 | 43.557 | 21.700 | 12.700 |
| 3 | 1.0100 | -12.73 | -94.200 | 6.075 | 0.000 | 25.075 | 94.200 | 19.000 |
| 4 | 1.0261 | -10.31 | -47.800 | 3.900 | 0.000 | 0.000 | 47.800 | -3.900 |
| 5 | 1.0326 | -8.77 | -7.600 | -1.600 | 0.000 | 0.000 | 7.600 | 7.600 |
| 6 | 1.0700 | -14.22 | -11.200 | 5.231 | 0.000 | 12.731 | 11.200 | 7.500 |
| 7 | 1.0448 | -13.36 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 8 | 1.0900 | -13.36 | 0.000 | 17.623 | 0.000 | 17.623 | 0.000 | 0.000 |
| 9 | 1.0276 | -14.94 | -29.500 | -16.600 | 0.000 | 0.000 | 29.500 | 16.600 |
| 10 | 1.0275 | -15.10 | -9.000 | -5.800 | 0.000 | 0.000 | 9.000 | 5.800 |
| 11 | 1.0449 | -14.79 | -3.500 | -1.800 | 0.000 | 0.000 | 3.500 | 1.800 |
| 12 | 1.0530 | -15.08 | -6.100 | -1.600 | 0.000 | 0.000 | 6.100 | 1.600 |
| 13 | 1.0462 | -15.16 | -13.500 | -5.800 | 0.000 | 0.000 | 13.500 | 5.800 |
| 14 | 1.0174 | -16.03 | -14.900 | -5.000 | 0.000 | 0.000 | 14.900 | 5.000 |
| Total | | | 13.393 | 8.938 | 272.393 | 82.438 | 259.000 | 73.500 |

## Reference

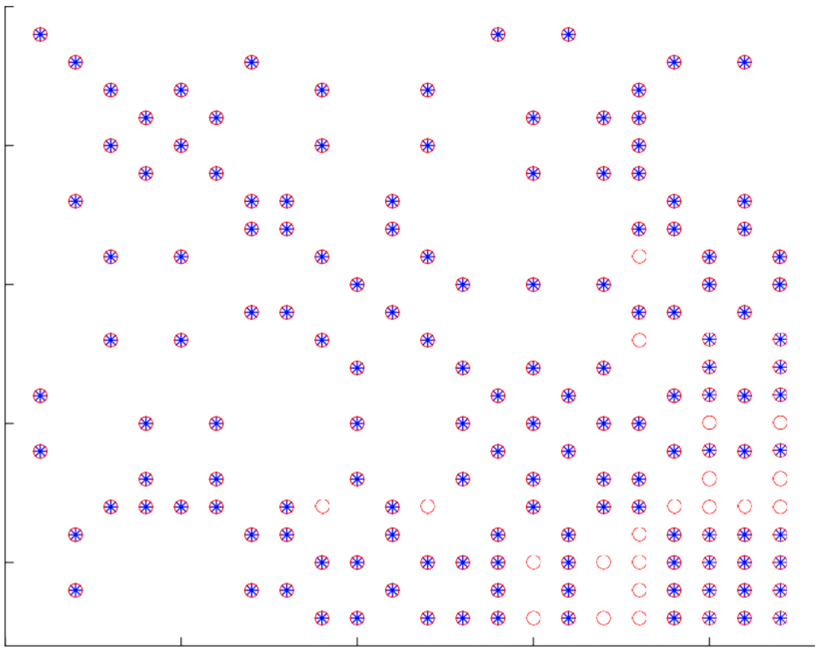

[1] J. J. Grainger, and W. D. Stevenson, *"Power System Analysis"*, Vol. 31, New York, McGraw-Hill, 1994.

[2] M. L. Crow, *"Computational Methods for Electric Power Systems"*, CRC Press, 2015.

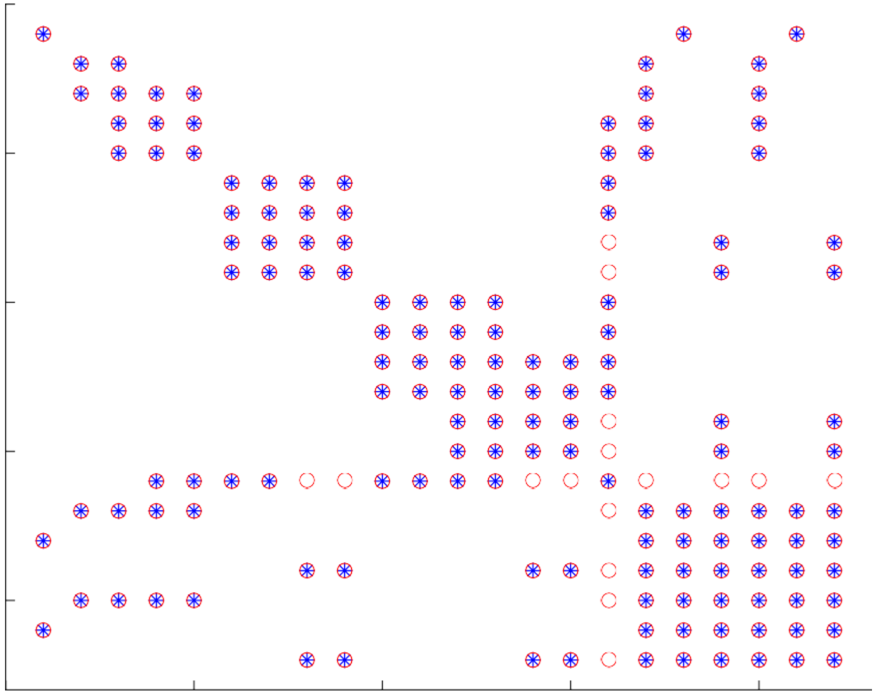
**IEEE 14 Bus System**



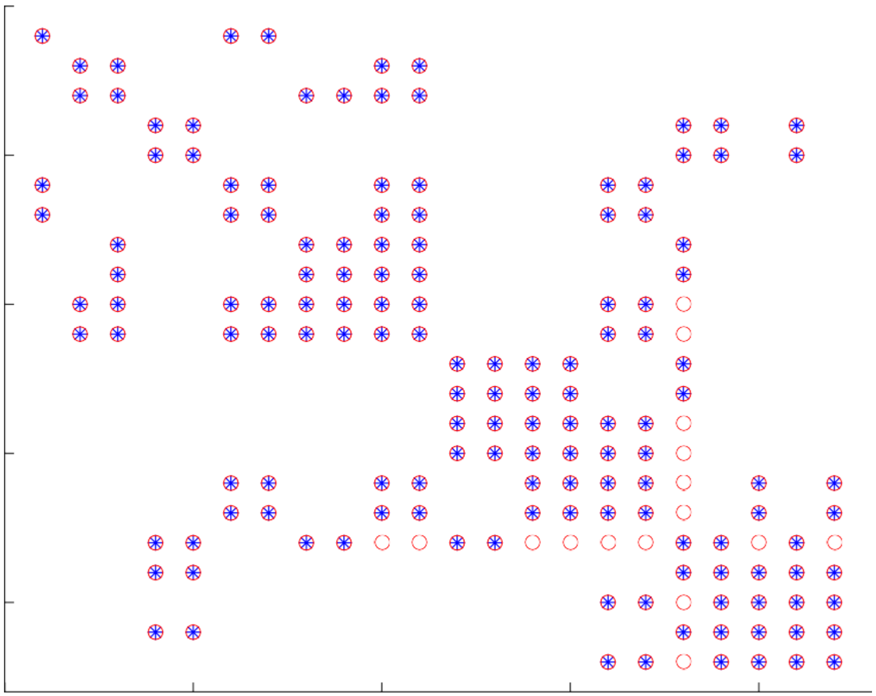
Without Ordering



Tinny 0



Tinny 1



Tinny  2

Appendix A

```
% This code use to read the bus data
fid = fopen('ieee14BusesData.txt','r');
Bus = textscan(fid,'%u8 %s %u8 %s %u8 %u8 %u8 %f32 %f32 %f32 %f32 %f32 %f32
%f32 %f32 %f32 %f32 %f32 %f32 %f32');
fclose(fid);
% Total_buses = length([Bus{1,1}]);
% This code use to read the branch data
fid = fopen('ieee14BranchesData.txt','r');
Branch = textscan(fid,'%f32 %f32 %u8 %u8 %u8 %u8 %f32 %f32 %f32 %u8 %u8 %u8
%u8 %u8 %f32 %f32 %f32 %f32 %f32 %f32 %f32');
fclose(fid);
Branches = [Branch{1,1}, Branch{1,2}, Branch{1,7}, Branch{1,8}, Branch{1,9},
Branch{1,15}];
BranchY = 1./(Branches(:,3)+Branches(:,4)*1i);
BranchB = Branch{1,9};
Total_buses = length([Bus{1,1}]); %to give the total number of the buses
% To find the Y bus matrix
Yij = zeros(Total_buses);
for i = 1:size(Branches,1)
    BUS1 = Branches(i,1);
    BUS2 = Branches(i,2);
    Yij(BUS1,BUS1) = Yij(BUS1,BUS1) + BranchY(i) + (Branches(i,5)-
0.5*BranchB(i))*1i;
    Yij(BUS2,BUS2) = Yij(BUS2,BUS2) + BranchY(i) + (Branches(i,5)-
0.5*BranchB(i))*1i;
    Yij(BUS1,BUS2) = Yij(BUS1,BUS2) - BranchY(i);
    Yij(BUS2,BUS1) = Yij(BUS2,BUS1) - BranchY(i);
end
Yij_Magnitude = abs(Yij); % magnitude of Yij
Yij_Angle = angle(Yij); % phase angle of Yij in radian
%************** Buses Type *************
BusType = [Bus{1,7}];
PQ = find(BusType == 0 | BusType == 1);
PV = find(BusType == 2);
Swing = find(BusType == 3);
BUS_no = [Bus{1,1}]; Bus_no_PVPQ = [Bus{1,1}]; Bus_no_PVPQ(Swing) = [];
Voltage = [Bus{1,8}]; Voltage(PQ',1) = 1 ; % Set PQ buses Voltage = 1
Phase_Angle = [Bus{1,9}]; Phase_Angle([PQ',PV'],1) = 0 ; % Set PV&PQ buses
Phase angle = 0
Pi = [Bus{1,12}]; Pi(Swing) = 0 ; % Set swing bus P = 0
Qi = [Bus{1,13}]; Qi(Swing) = 0; % Set swing bus Q = 0
P_injection = (Pi - [Bus{1,10}])/100;
Q_injection = (Qi - [Bus{1,11}])/100;
count = 0; Err = 1; Stp_Err = 0.01; % Err= error.. Stp_Err= stopping error
while Err > Stp_Err
    DELTA_P = [];
    DELTA_Q = [];
    J11 = []; J12 = []; J21 = []; J22 = [];
    %J11 i/i
    for i = Bus_no_PVPQ'
        S = 0; Jacobian = 0;
        for j = BUS_no'
            S = S + Voltage(j)*Yij_Magnitude(i,j)*cos(Phase_Angle(i)-
Phase_Angle(j)-Yij_Angle(i,j));
```

```
            Jacobian = Jacobian +
Voltage(j)*Yij_Magnitude(i,j)*sin(Phase_Angle(i)-Phase_Angle(j)-
Yij_Angle(i,j));
        end
        DELTA_P = [DELTA_P; P_injection(i) - Voltage(i)*S];
        J11 = [J11, -Voltage(i)*Jacobian -
(Voltage(i)^2)*Yij_Magnitude(i,i)*sin(Yij_Angle(i,i))];
    end
    J11 = diag(J11);
    % J11 i/j
    J11_nondiag = [];
    for i = Bus_no_PVPQ'
        for j = Bus_no_PVPQ'
            if i ~= j
                J11_nondiag(i,j) =
Voltage(i)*Voltage(j)*Yij_Magnitude(i,j)*sin(Phase_Angle(i)-Phase_Angle(j)-
Yij_Angle(i,j));
            end
        end
    end
    J11_nondiag(Swing,:) = [];
    J11_nondiag(:,Swing) = [];
    J11 = J11 + J11_nondiag;
    %J12 i/i and J21 i/i
    for i = PQ'
        Jacobian = 0;
        for j = BUS_no'
            Jacobian = Jacobian +
Voltage(j)*Yij_Magnitude(i,j)*cos(Phase_Angle(i)-Phase_Angle(j)-
Yij_Angle(i,j));
        end
        J12(find(Bus_no_PVPQ==i),find(PQ==i)) = Jacobian +
Voltage(i)*Yij_Magnitude(i,i)*cos(Yij_Angle(i,i));
        J21(find(PQ==i),find(Bus_no_PVPQ==i)) = Voltage(i)*Jacobian -
(Voltage(i)^2)*Yij_Magnitude(i,i)*cos(Yij_Angle(i,i));
    end
    % J12 i/j
    for i = Bus_no_PVPQ'
        for j = PQ'
            if i ~= j
                J12(find(Bus_no_PVPQ==i),find(PQ==j)) =
Voltage(i)*Yij_Magnitude(i,j)*cos(Phase_Angle(i)-Phase_Angle(j)-
Yij_Angle(i,j));
            end
        end
    end
    % J21 i/j
    for i = PQ'
        for j = Bus_no_PVPQ'
            if i ~= j
                J21(find(PQ==i),find(Bus_no_PVPQ==j)) = -
Voltage(i)*Voltage(j)*Yij_Magnitude(i,j)*cos(Phase_Angle(i)-Phase_Angle(j)-
Yij_Angle(i,j));
            end
        end
    end
    % J22 i/i
```

```
    for i = PQ'
        S = 0;
        for j = BUS_no'
            S = S + Voltage(j)*Yij_Magnitude(i,j)*sin(Phase_Angle(i)-
Phase_Angle(j)-Yij_Angle(i,j));
        end
        DELTA_Q = [DELTA_Q; Q_injection(i)-Voltage(i)*S];
        J22(find(PQ==i),find(PQ==i)) = S -
Voltage(i)*Yij_Magnitude(i,i)*sin(Yij_Angle(i,i));
    end
    %J22 i/j
    for i = PQ'
        for j = PQ'
            if i ~= j
                J22(find(PQ==i),find(PQ==j)) =
Voltage(i)*Yij_Magnitude(i,j)*sin(Phase_Angle(i)-Phase_Angle(j)-
Yij_Angle(i,j));
            end
        end
    end
    Delta_PhaseangleVoltage = [DELTA_P; DELTA_Q];
    % Jacobian
    Jacobian = [J11, J12; J21, J22];
    Jacobian(find(abs(Jacobian)<0.5)) = 0;

    [TJ, TJF] = store_in_linked_list(Jacobian);
    ordered_index = 1:22;
    [TQ, TQF] = LU_linked_list(TJ, TJF, ordered_index);

    dv_x = FB_linked_list(TQ, TQF, Delta_PhaseangleVoltage(ordered_index));

    dv_x = ordering_scheme_reversion(dv_x, ordered_index);

    Phase_Angle(Bus_no_PVPQ',1) = Phase_Angle(Bus_no_PVPQ',1) +
dv_x(1:length(Bus_no_PVPQ));
    Voltage(PQ',1) = Voltage(PQ',1) + dv_x(length(Bus_no_PVPQ)+1:end);

    Err = max(abs(Delta_PhaseangleVoltage));
end
% compute the P and Q
for i = BUS_no'
    sigma1 = 0; sigma2 = 0;
    for j = BUS_no'
        sigma1 = sigma1 + Voltage(j)*Yij_Magnitude(i,j)*cos(Phase_Angle(i)-
Phase_Angle(j)-Yij_Angle(i,j));
        sigma2 = sigma2 + Voltage(j)*Yij_Magnitude(i,j)*sin(Phase_Angle(i)-
Phase_Angle(j)-Yij_Angle(i,j));
    end
    P_injection(i) = Voltage(i)*sigma1;
    Q_injection(i) = Voltage(i)*sigma2;
end
Pgen = P_injection + [Bus{1,10}]/100;
Qgen = Q_injection + [Bus{1,11}]/100;
```

Appendix B

```
% This code use to read the bus data
fid = fopen('ieee14BusesData.txt','r');
Bus = textscan(fid,'%u8 %s %u8 %s %u8 %u8 %u8 %f32 %f32 %f32 %f32 %f32 %f32
%f32 %f32 %f32 %f32 %f32 %f32 %f32');
fclose(fid);
% Total_buses = length([Bus{1,1}]);
% This code use to read the branch data
fid = fopen('ieee14BranchesData.txt','r');
Branch = textscan(fid,'%f32 %f32 %u8 %u8 %u8 %u8 %f32 %f32 %f32 %u8 %u8 %u8
%u8 %u8 %f32 %f32 %f32 %f32 %f32 %f32 %f32');
fclose(fid);
Branches = [Branch{1,1}, Branch{1,2}, Branch{1,7}, Branch{1,8}, Branch{1,9},
Branch{1,15}];
BranchY = 1./(Branches(:,3)+Branches(:,4)*1i);
BranchB = Branch{1,9};
Total_buses = length([Bus{1,1}]); %to give the total number of the buses
% To find the Y bus matrix
Yij = zeros(Total_buses);
for i = 1:size(Branches,1)
    BUS1 = Branches(i,1);
    BUS2 = Branches(i,2);
    Yij(BUS1,BUS1) = Yij(BUS1,BUS1) + BranchY(i) + (Branches(i,5)-
0.5*BranchB(i))*1i;
    Yij(BUS2,BUS2) = Yij(BUS2,BUS2) + BranchY(i) + (Branches(i,5)-
0.5*BranchB(i))*1i;
    Yij(BUS1,BUS2) = Yij(BUS1,BUS2) - BranchY(i);
    Yij(BUS2,BUS1) = Yij(BUS2,BUS1) - BranchY(i);
end
Yij_Magnitude = abs(Yij); % magnitude of Yij
Yij_Angle = angle(Yij); % phase angle of Yij in radian
%************** Buses Type *************
BusType = [Bus{1,7}];
PQ = find(BusType == 0 | BusType == 1);
PV = find(BusType == 2);
Swing = find(BusType == 3);
BUS_no = [Bus{1,1}]; Bus_no_PVPQ = [Bus{1,1}]; Bus_no_PVPQ(Swing) = [];
Voltage = [Bus{1,8}]; Voltage(PQ',1) = 1 ; % Set PQ buses Voltage = 1
Phase_Angle = [Bus{1,9}]; Phase_Angle([PQ',PV'],1) = 0 ; % Set PV&PQ buses
Phase angle = 0
Pi = [Bus{1,12}]; Pi(Swing) = 0 ; % Set swing bus P = 0
Qi = [Bus{1,13}]; Qi(Swing) = 0; % Set swing bus Q = 0
P_injection = (Pi - [Bus{1,10}])/100;
Q_injection = (Qi - [Bus{1,11}])/100;
count = 0; Err = 1; Stp_Err = 0.01; % Err= error.. Stp_Err= stopping error
while Err > Stp_Err
    DELTA_P = [];
    DELTA_Q = [];
    J11 = []; J12 = []; J21 = []; J22 = [];
    %J11 i/i
    for i = Bus_no_PVPQ'
        S = 0; Jacobian = 0;
        for j = BUS_no'
            S = S + Voltage(j)*Yij_Magnitude(i,j)*cos(Phase_Angle(i)-
Phase_Angle(j)-Yij_Angle(i,j));
```

```
            Jacobian = Jacobian +
Voltage(j)*Yij_Magnitude(i,j)*sin(Phase_Angle(i)-Phase_Angle(j)-
Yij_Angle(i,j));
        end
        DELTA_P = [DELTA_P; P_injection(i) - Voltage(i)*S];
        J11 = [J11, -Voltage(i)*Jacobian -
(Voltage(i)^2)*Yij_Magnitude(i,i)*sin(Yij_Angle(i,i))];
    end
    J11 = diag(J11);
    % J11 i/j
    J11_nondiag = [];
    for i = Bus_no_PVPQ'
        for j = Bus_no_PVPQ'
            if i ~= j
                J11_nondiag(i,j) =
Voltage(i)*Voltage(j)*Yij_Magnitude(i,j)*sin(Phase_Angle(i)-Phase_Angle(j)-
Yij_Angle(i,j));
            end
        end
    end
    J11_nondiag(Swing,:) = [];
    J11_nondiag(:,Swing) = [];
    J11 = J11 + J11_nondiag;
    %J12 i/i and J21 i/i
    for i = PQ'
        Jacobian = 0;
        for j = BUS_no'
            Jacobian = Jacobian +
Voltage(j)*Yij_Magnitude(i,j)*cos(Phase_Angle(i)-Phase_Angle(j)-
Yij_Angle(i,j));
        end
        J12(find(Bus_no_PVPQ==i),find(PQ==i)) = Jacobian +
Voltage(i)*Yij_Magnitude(i,i)*cos(Yij_Angle(i,i));
        J21(find(PQ==i),find(Bus_no_PVPQ==i)) = Voltage(i)*Jacobian -
(Voltage(i)^2)*Yij_Magnitude(i,i)*cos(Yij_Angle(i,i));
    end
    % J12 i/j
    for i = Bus_no_PVPQ'
        for j = PQ'
            if i ~= j
                J12(find(Bus_no_PVPQ==i),find(PQ==j)) =
Voltage(i)*Yij_Magnitude(i,j)*cos(Phase_Angle(i)-Phase_Angle(j)-
Yij_Angle(i,j));
            end
        end
    end
    % J21 i/j
    for i = PQ'
        for j = Bus_no_PVPQ'
            if i ~= j
                J21(find(PQ==i),find(Bus_no_PVPQ==j)) = -
Voltage(i)*Voltage(j)*Yij_Magnitude(i,j)*cos(Phase_Angle(i)-Phase_Angle(j)-
Yij_Angle(i,j));
            end
        end
    end
    % J22 i/i
```

```
    for i = PQ'
        S = 0;
        for j = BUS_no'
            S = S + Voltage(j)*Yij_Magnitude(i,j)*sin(Phase_Angle(i)-
Phase_Angle(j)-Yij_Angle(i,j));
        end
        DELTA_Q = [DELTA_Q; Q_injection(i)-Voltage(i)*S];
        J22(find(PQ==i),find(PQ==i)) = S -
Voltage(i)*Yij_Magnitude(i,i)*sin(Yij_Angle(i,i));
    end
    %J22 i/j
    for i = PQ'
        for j = PQ'
            if i ~= j
                J22(find(PQ==i),find(PQ==j)) =
Voltage(i)*Yij_Magnitude(i,j)*sin(Phase_Angle(i)-Phase_Angle(j)-
Yij_Angle(i,j));
            end
        end
    end
    Delta_PhaseangleVoltage = [DELTA_P; DELTA_Q];
    % Jacobian
    Jacobian = [J11, J12; J21, J22];
    Jacobian(find(abs(Jacobian)<0.5)) = 0;

    [TJ, TJF] = store_in_linked_list(Jacobian);
    ordered_index = ordering_scheme_tinney0(TJ, TJF);

    [TQ, TQF] = LU_linked_list(TJ, TJF, ordered_index);

    dv_x = FB_linked_list(TQ, TQF, Delta_PhaseangleVoltage(ordered_index));

    dv_x = ordering_scheme_reversion(dv_x, ordered_index);

    Phase_Angle(Bus_no_PVPQ',1) = Phase_Angle(Bus_no_PVPQ',1) +
dv_x(1:length(Bus_no_PVPQ));
    Voltage(PQ',1) = Voltage(PQ',1) + dv_x(length(Bus_no_PVPQ)+1:end);

    Err = max(abs(Delta_PhaseangleVoltage));
end
% compute the P and Q
for i = BUS_no'
    sigma1 = 0; sigma2 = 0;
    for j = BUS_no'
        sigma1 = sigma1 + Voltage(j)*Yij_Magnitude(i,j)*cos(Phase_Angle(i)-
Phase_Angle(j)-Yij_Angle(i,j));
        sigma2 = sigma2 + Voltage(j)*Yij_Magnitude(i,j)*sin(Phase_Angle(i)-
Phase_Angle(j)-Yij_Angle(i,j));
    end
    P_injection(i) = Voltage(i)*sigma1;
    Q_injection(i) = Voltage(i)*sigma2;
end
Pgen = P_injection + [Bus{1,10}]/100;
Qgen = Q_injection + [Bus{1,11}]/100;
```

Appendix C

```
% This code use to read the bus data
fid = fopen('ieee14BusesData.txt','r');
Bus = textscan(fid,'%u8 %s %u8 %s %u8 %u8 %u8 %f32 %f32 %f32 %f32 %f32 %f32
%f32 %f32 %f32 %f32 %f32 %f32 %f32');
fclose(fid);
% Total_buses = length([Bus{1,1}]);
% This code use to read the branch data
fid = fopen('ieee14BranchesData.txt','r');
Branch = textscan(fid,'%f32 %f32 %u8 %u8 %u8 %u8 %f32 %f32 %f32 %u8 %u8 %u8
%u8 %u8 %f32 %f32 %f32 %f32 %f32 %f32 %f32');
fclose(fid);
Branches = [Branch{1,1}, Branch{1,2}, Branch{1,7}, Branch{1,8}, Branch{1,9},
Branch{1,15}];
BranchY = 1./(Branches(:,3)+Branches(:,4)*1i);
BranchB = Branch{1,9};
Total_buses = length([Bus{1,1}]); %to give the total number of the buses
% To find the Y bus matrix
Yij = zeros(Total_buses);
for i = 1:size(Branches,1)
    BUS1 = Branches(i,1);
    BUS2 = Branches(i,2);
    Yij(BUS1,BUS1) = Yij(BUS1,BUS1) + BranchY(i) + (Branches(i,5)-
0.5*BranchB(i))*1i;
    Yij(BUS2,BUS2) = Yij(BUS2,BUS2) + BranchY(i) + (Branches(i,5)-
0.5*BranchB(i))*1i;
    Yij(BUS1,BUS2) = Yij(BUS1,BUS2) - BranchY(i);
    Yij(BUS2,BUS1) = Yij(BUS2,BUS1) - BranchY(i);
end
Yij_Magnitude = abs(Yij); % magnitude of Yij
Yij_Angle = angle(Yij); % phase angle of Yij in radian
%************** Buses Type *************
BusType = [Bus{1,7}];
PQ = find(BusType == 0 | BusType == 1);
PV = find(BusType == 2);
Swing = find(BusType == 3);
BUS_no = [Bus{1,1}]; Bus_no_PVPQ = [Bus{1,1}]; Bus_no_PVPQ(Swing) = [];
Voltage = [Bus{1,8}]; Voltage(PQ',1) = 1 ; % Set PQ buses Voltage = 1
Phase_Angle = [Bus{1,9}]; Phase_Angle([PQ',PV'],1) = 0 ; % Set PV&PQ buses
Phase angle = 0
Pi = [Bus{1,12}]; Pi(Swing) = 0 ; % Set swing bus P = 0
Qi = [Bus{1,13}]; Qi(Swing) = 0; % Set swing bus Q = 0
P_injection = (Pi - [Bus{1,10}])/100;
Q_injection = (Qi - [Bus{1,11}])/100;
count = 0; Err = 1; Stp_Err = 0.01; % Err= error.. Stp_Err= stopping error
while Err >= Stp_Err
    DELTA_P = [];
    DELTA_Q = [];
    J11 = []; J12 = []; J21 = []; J22 = [];
    %J11 i/i
    for i = Bus_no_PVPQ'
        S = 0; Jacobian = 0;
        for j = BUS_no'
            S = S + Voltage(j)*Yij_Magnitude(i,j)*cos(Phase_Angle(i)-
Phase_Angle(j)-Yij_Angle(i,j));
```

```
            Jacobian = Jacobian + 
Voltage(j)*Yij_Magnitude(i,j)*sin(Phase_Angle(i)-Phase_Angle(j)-
Yij_Angle(i,j));
        end
        DELTA_P = [DELTA_P; P_injection(i) - Voltage(i)*S];
        J11 = [J11, -Voltage(i)*Jacobian - 
(Voltage(i)^2)*Yij_Magnitude(i,i)*sin(Yij_Angle(i,i))];
    end
    J11 = diag(J11);
    % J11 i/j
    J11_nondiag = [];
    for i = Bus_no_PVPQ'
        for j = Bus_no_PVPQ'
            if i ~= j
                J11_nondiag(i,j) = 
Voltage(i)*Voltage(j)*Yij_Magnitude(i,j)*sin(Phase_Angle(i)-Phase_Angle(j)-
Yij_Angle(i,j));
            end
        end
    end
    J11_nondiag(Swing,:) = [];
    J11_nondiag(:,Swing) = [];
    J11 = J11 + J11_nondiag;
    %J12 i/i and J21 i/i
    for i = PQ'
        Jacobian = 0;
        for j = BUS_no'
            Jacobian = Jacobian + 
Voltage(j)*Yij_Magnitude(i,j)*cos(Phase_Angle(i)-Phase_Angle(j)-
Yij_Angle(i,j));
        end
        J12(find(Bus_no_PVPQ==i),find(PQ==i)) = Jacobian + 
Voltage(i)*Yij_Magnitude(i,i)*cos(Yij_Angle(i,i));
        J21(find(PQ==i),find(Bus_no_PVPQ==i)) = Voltage(i)*Jacobian - 
(Voltage(i)^2)*Yij_Magnitude(i,i)*cos(Yij_Angle(i,i));
    end
    % J12 i/j
    for i = Bus_no_PVPQ'
        for j = PQ'
            if i ~= j
                J12(find(Bus_no_PVPQ==i),find(PQ==j)) = 
Voltage(i)*Yij_Magnitude(i,j)*cos(Phase_Angle(i)-Phase_Angle(j)-
Yij_Angle(i,j));
            end
        end
    end
    % J21 i/j
    for i = PQ'
        for j = Bus_no_PVPQ'
            if i ~= j
                J21(find(PQ==i),find(Bus_no_PVPQ==j)) = -
Voltage(i)*Voltage(j)*Yij_Magnitude(i,j)*cos(Phase_Angle(i)-Phase_Angle(j)-
Yij_Angle(i,j));
            end
        end
    end
    % J22 i/i
```

```
    for i = PQ'
        S = 0;
        for j = BUS_no'
            S = S + Voltage(j)*Yij_Magnitude(i,j)*sin(Phase_Angle(i)-
Phase_Angle(j)-Yij_Angle(i,j));
        end
        DELTA_Q = [DELTA_Q; Q_injection(i)-Voltage(i)*S];
        J22(find(PQ==i),find(PQ==i)) = S -
Voltage(i)*Yij_Magnitude(i,i)*sin(Yij_Angle(i,i));
    end
    %J22 i/j
    for i = PQ'
        for j = PQ'
            if i ~= j
                J22(find(PQ==i),find(PQ==j)) =
Voltage(i)*Yij_Magnitude(i,j)*sin(Phase_Angle(i)-Phase_Angle(j)-
Yij_Angle(i,j));
            end
        end
    end
    Delta_PhaseangleVoltage = [DELTA_P; DELTA_Q];
    % Jacobian
    Jacobian = [J11, J12; J21, J22];
    Jacobian(find(abs(Jacobian)<0.5)) = 0;

    [TA, TAF] = store_in_linked_list(Jacobian);
    ordered_index = ordering_scheme_tinney1(TA, TAF);

    [TQ, TQF] = LU_linked_list(TA, TAF, ordered_index);

    dv_x = FB_linked_list(TQ, TQF, Delta_PhaseangleVoltage(ordered_index));

    dv_x = ordering_scheme_reversion(dv_x, ordered_index);

    Phase_Angle(Bus_no_PVPQ',1) = Phase_Angle(Bus_no_PVPQ',1) +
dv_x(1:length(Bus_no_PVPQ));
    Voltage(PQ',1) = Voltage(PQ',1) + dv_x(length(Bus_no_PVPQ)+1:end);

    Err = max(abs(Delta_PhaseangleVoltage));
end
% compute the P and Q
for i = BUS_no'
    sigma1 = 0; sigma2 = 0;
    for j = BUS_no'
        sigma1 = sigma1 + Voltage(j)*Yij_Magnitude(i,j)*cos(Phase_Angle(i)-
Phase_Angle(j)-Yij_Angle(i,j));
        sigma2 = sigma2 + Voltage(j)*Yij_Magnitude(i,j)*sin(Phase_Angle(i)-
Phase_Angle(j)-Yij_Angle(i,j));
    end
    P_injection(i) = Voltage(i)*sigma1;
    Q_injection(i) = Voltage(i)*sigma2;
end
Pgen = P_injection + [Bus{1,10}]/100;
Qgen = Q_injection + [Bus{1,11}]/100;
```

Appendix D

```
% This code use to read the bus data
fid = fopen('ieee14BusesData.txt','r');
Bus = textscan(fid,'%u8 %s %u8 %s %u8 %u8 %u8 %f32 %f32 %f32 %f32 %f32 %f32
%f32 %f32 %f32 %f32 %f32 %f32 %f32');
fclose(fid);
% Total_buses = length([Bus{1,1}]);
% This code use to read the branch data
fid = fopen('ieee14BranchesData.txt','r');
Branch = textscan(fid,'%f32 %f32 %u8 %u8 %u8 %u8 %f32 %f32 %f32 %u8 %u8 %u8
%u8 %u8 %f32 %f32 %f32 %f32 %f32 %f32 %f32');
fclose(fid);
Branches = [Branch{1,1}, Branch{1,2}, Branch{1,7}, Branch{1,8}, Branch{1,9},
Branch{1,15}];
BranchY = 1./(Branches(:,3)+Branches(:,4)*1i);
BranchB = Branch{1,9};
Total_buses = length([Bus{1,1}]); %to give the total number of the buses
% To find the Y bus matrix
Yij = zeros(Total_buses);
for i = 1:size(Branches,1)
    BUS1 = Branches(i,1);
    BUS2 = Branches(i,2);
    Yij(BUS1,BUS1) = Yij(BUS1,BUS1) + BranchY(i) + (Branches(i,5)-
0.5*BranchB(i))*1i;
    Yij(BUS2,BUS2) = Yij(BUS2,BUS2) + BranchY(i) + (Branches(i,5)-
0.5*BranchB(i))*1i;
    Yij(BUS1,BUS2) = Yij(BUS1,BUS2) - BranchY(i);
    Yij(BUS2,BUS1) = Yij(BUS2,BUS1) - BranchY(i);
end
Yij_Magnitude = abs(Yij); % magnitude of Yij
Yij_Angle = angle(Yij); % phase angle of Yij in radian
%************** Buses Type *************
BusType = [Bus{1,7}];
PQ = find(BusType == 0 | BusType == 1);
PV = find(BusType == 2);
Swing = find(BusType == 3);
BUS_no = [Bus{1,1}]; Bus_no_PVPQ = [Bus{1,1}]; Bus_no_PVPQ(Swing) = [];
Voltage = [Bus{1,8}]; Voltage(PQ',1) = 1 ; % Set PQ buses Voltage = 1
Phase_Angle = [Bus{1,9}]; Phase_Angle([PQ',PV'],1) = 0 ; % Set PV&PQ buses
Phase angle = 0
Pi = [Bus{1,12}]; Pi(Swing) = 0 ; % Set swing bus P = 0
Qi = [Bus{1,13}]; Qi(Swing) = 0; % Set swing bus Q = 0
P_injection = (Pi - [Bus{1,10}])/100;
Q_injection = (Qi - [Bus{1,11}])/100;
count = 0; Err = 1; Stp_Err = 0.01; % Err= error.. Stp_Err= stopping error
while Err > Stp_Err
    DELTA_P = [];
    DELTA_Q = [];
    J11 = []; J12 = []; J21 = []; J22 = [];
    %J11 i/i
    for i = Bus_no_PVPQ'
        S = 0; Jacobian = 0;
        for j = BUS_no'
            S = S + Voltage(j)*Yij_Magnitude(i,j)*cos(Phase_Angle(i)-
Phase_Angle(j)-Yij_Angle(i,j));
```

```
            Jacobian = Jacobian +
Voltage(j)*Yij_Magnitude(i,j)*sin(Phase_Angle(i)-Phase_Angle(j)-
Yij_Angle(i,j));
        end
        DELTA_P = [DELTA_P; P_injection(i) - Voltage(i)*S];
        J11 = [J11, -Voltage(i)*Jacobian -
(Voltage(i)^2)*Yij_Magnitude(i,i)*sin(Yij_Angle(i,i))];
    end
    J11 = diag(J11);
    % J11 i/j
    J11_nondiag = [];
    for i = Bus_no_PVPQ'
        for j = Bus_no_PVPQ'
            if i ~= j
                J11_nondiag(i,j) =
Voltage(i)*Voltage(j)*Yij_Magnitude(i,j)*sin(Phase_Angle(i)-Phase_Angle(j)-
Yij_Angle(i,j));
            end
        end
    end
    J11_nondiag(Swing,:) = [];
    J11_nondiag(:,Swing) = [];
    J11 = J11 + J11_nondiag;
    %J12 i/i and J21 i/i
    for i = PQ'
        Jacobian = 0;
        for j = BUS_no'
            Jacobian = Jacobian +
Voltage(j)*Yij_Magnitude(i,j)*cos(Phase_Angle(i)-Phase_Angle(j)-
Yij_Angle(i,j));
        end
        J12(find(Bus_no_PVPQ==i),find(PQ==i)) = Jacobian +
Voltage(i)*Yij_Magnitude(i,i)*cos(Yij_Angle(i,i));
        J21(find(PQ==i),find(Bus_no_PVPQ==i)) = Voltage(i)*Jacobian -
(Voltage(i)^2)*Yij_Magnitude(i,i)*cos(Yij_Angle(i,i));
    end
    % J12 i/j
    for i = Bus_no_PVPQ'
        for j = PQ'
            if i ~= j
                J12(find(Bus_no_PVPQ==i),find(PQ==j)) =
Voltage(i)*Yij_Magnitude(i,j)*cos(Phase_Angle(i)-Phase_Angle(j)-
Yij_Angle(i,j));
            end
        end
    end
    % J21 i/j
    for i = PQ'
        for j = Bus_no_PVPQ'
            if i ~= j
                J21(find(PQ==i),find(Bus_no_PVPQ==j)) = -
Voltage(i)*Voltage(j)*Yij_Magnitude(i,j)*cos(Phase_Angle(i)-Phase_Angle(j)-
Yij_Angle(i,j));
            end
        end
    end
    % J22 i/i
```

```
    for i = PQ'
        S = 0;
        for j = BUS_no'
            S = S + Voltage(j)*Yij_Magnitude(i,j)*sin(Phase_Angle(i)-
Phase_Angle(j)-Yij_Angle(i,j));
        end
        DELTA_Q = [DELTA_Q; Q_injection(i)-Voltage(i)*S];
        J22(find(PQ==i),find(PQ==i)) = S -
Voltage(i)*Yij_Magnitude(i,i)*sin(Yij_Angle(i,i));
    end
    %J22 i/j
    for i = PQ'
        for j = PQ'
            if i ~= j
                J22(find(PQ==i),find(PQ==j)) =
Voltage(i)*Yij_Magnitude(i,j)*sin(Phase_Angle(i)-Phase_Angle(j)-
Yij_Angle(i,j));
            end
        end
    end
    Delta_PhaseangleVoltage = [DELTA_P; DELTA_Q];
    % Jacobian
    Jacobian = [J11, J12; J21, J22];
    Jacobian(find(abs(Jacobian)<0.5)) = 0;

    [TJ, TJF] = store_in_linked_list(Jacobian);
    ordered_index = ordering_scheme_tinney2(TJ, TJF);

    [TQ, TQF] = LU_linked_list(TJ, TJF, ordered_index);

    dv_x = FB_linked_list(TQ, TQF, Delta_PhaseangleVoltage(ordered_index));

    dv_x = ordering_scheme_reversion(dv_x, ordered_index);

    Phase_Angle(Bus_no_PVPQ',1) = Phase_Angle(Bus_no_PVPQ',1) +
dv_x(1:length(Bus_no_PVPQ));
    Voltage(PQ',1) = Voltage(PQ',1) + dv_x(length(Bus_no_PVPQ)+1:end);

    Err = max(abs(Delta_PhaseangleVoltage));
end
% compute the P and Q
for i = BUS_no'
    sigma1 = 0; sigma2 = 0;
    for j = BUS_no'
        sigma1 = sigma1 + Voltage(j)*Yij_Magnitude(i,j)*cos(Phase_Angle(i)-
Phase_Angle(j)-Yij_Angle(i,j));
        sigma2 = sigma2 + Voltage(j)*Yij_Magnitude(i,j)*sin(Phase_Angle(i)-
Phase_Angle(j)-Yij_Angle(i,j));
    end
    P_injection(i) = Voltage(i)*sigma1;
    Q_injection(i) = Voltage(i)*sigma2;
end
Pgen = P_injection + [Bus{1,10}]/100;
Qgen = Q_injection + [Bus{1,11}]/100;
```

## Appendix E

```
function [TQ, TQF] = LU_linked_list(TA, TAF, ordered_index)
Nmulti = 0;
Nnz = 0;
index = [];
value = [];
NRow = [];
NCol = [];
NIR = [];
NIC = [];
FIR = zeros(length(TAF.FIR), 1);
FIC = zeros(length(TAF.FIC), 1);

TQ = table(index, value, NRow, NCol, NIR, NIC);
TQF = table(FIR, FIC);

for j = 1: length(TAF.FIC)
    % calculate the Q column elements
    for k = j: length(TAF.FIR)
        col_temp = 0;
        for i = 1: j-1
            Qki = search_in_linked_list(TQ, TQF, k, i);
            Qij = search_in_linked_list(TQ, TQF, i, j);
            if Qki ~= 0 && Qij ~= 0
                col_temp = col_temp + Qki * Qij;
                Nmulti = Nmulti + 1;
            end
        end
        Akj = search_in_linked_list(TA, TAF, ordered_index(k),
ordered_index(j));
        Qkj = Akj - col_temp;
        if Qkj ~= 0 % store non-zeros
            [TQ, TQF] = add_in_linked_list(TQ, TQF, Qkj, k, j);
            Nnz = Nnz + 1;
        end
    end

    % calculate the Q row elements
    if search_in_linked_list(TQ, TQF, j, j) ~= 0 % Q(j,j) ~= 0
        for k = j+1: length(FIC)
            row_temp = 0;
            for i = 1: j-1
                Qji = search_in_linked_list(TQ, TQF, j, i);
                Qik = search_in_linked_list(TQ, TQF, i, k);
                if Qji ~= 0 && Qik ~= 0
                    row_temp = row_temp - Qji * Qik;
                end
            end
            Ajk = search_in_linked_list(TA, TAF, ordered_index(j),
ordered_index(k));
            Qjj = search_in_linked_list(TQ, TQF, j, j);
            Qjk = (Ajk + row_temp) / Qjj;
            if Qjk ~= 0 % store non-zeros
               [TQ, TQF] = add_in_linked_list(TQ, TQF, Qjk, j, k);
               Nnz = Nnz + 1;
```

```
            end
        end
    end
end
fprintf('Number of fills: %d\n', Nnz - length(TA.NIR));
fprintf('Number of non-zeros: %d\n', Nnz);
fprintf('Number of multiplications: %d\n', Nmulti * 2);
fprintf('Number of total processing steps: %d\n', Nnz + Nmulti * 2);
end
```

## Appendix F

```
function [x] = FB_linked_list(TQ, TQF, ordered_index)
% forward substitution
y = zeros(length(TQF.FIR), 1);
for k = 1:length(TQF.FIR)
    y_temp = 0;
    for j = 1:(k-1)
        Qkj = search_in_linked_list(TQ, TQF, k, j);
        y_temp = y_temp + Qkj * y(j);
    end
    Qkk = search_in_linked_list(TQ, TQF, k, k);
    y(k) = (ordered_index(k) - y_temp)/Qkk;
end

% backward substitution
x = zeros(length(TQF.FIC), 1);
for k = length(TQF.FIC):-1:1
    x_temp = 0;
    for j = (k+1):length(TQF.FIC)
        Qkj = search_in_linked_list(TQ, TQF, k, j);
        x_temp = x_temp + Qkj * x(j);
    end
    x(k) = y(k) - x_temp;
end
end
```

## Appendix G

```
function [TA, TAF] = store_in_linked_list(A)
%% Initialization parameters
% Initialzie parameters in linked list representation table of matrix A -
index, value, NRow, NCol, NIR, NIC
NIC = zeros(length(A), 1);
NIR = zeros(length(A), 1);
FIR = []; FIC = [];

[NCol, NRow, value] = find(A'); % record sparse matrix information - NCol,
NRow, value
index = find(value); % get index value

% Store the next nonzero in row / column
for i = 1:length(A)
    NIR_temp = find(NRow == i); % find index for element which has same row
number
```

```
    NIR(NIR_temp) = [NIR_temp(2: end); 0];  % next in row
    FIR = [FIR; NIR_temp(1)];  % first in row
    NIC_temp = find(NCol == i); % find index for element which has same
column number
    NIC(NIC_temp) = [NIC_temp(2: end); 0];  % next in column
    FIC = [FIC; NIC_temp(1)];  % first in column
end

% Get non-zero elements table of A
TA = table(index, value, NRow, NCol, NIR, NIC);
TAF = table(FIR, FIC);

end
```

## Appendix H

```
function [TA_new, TAF_new] = add_in_linked_list(TA, TAF, aij, i, j)
%% Add element to NRow, NCol, value; Initialize new NIR, NIC, FIR, FIC
if isempty(TA) == false
    index = [TA.index; TA.index(end)+1];
else
    index = [1];
end
value = [TA.value; aij];
NRow = [TA.NRow; i];
NCol = [TA.NCol; j];
NIR = TA.NIR;
NIC = TA.NIC;
FIR = TAF.FIR;
FIC = TAF.FIC;

%% Update element in FIR, NIR
row_index = FIR(i);

if row_index == false % update first NIRs, FIRs for Q
    FIR(i) = index(end);
    NIR(index(end), 1) = 0;
else % update NIC, FIC
    % update NIR, FIR (NRow keeps constant, NCol changes)
    while true
        if NCol(row_index) == j % update an existing value by aij
            value(row_index) = aij;
            value(end) = [];
            index(end) = [];
            NCol(end) = [];
            break;
        elseif NCol(row_index) > j % update aij as FIR in row i
            NIR(index(end)) = row_index;
            FIR(i) = index(end);
            break;
        elseif NIR(row_index) == 0 % update aij as the last element in row i
            NIR = [NIR; NIR(row_index)];
            NIR(row_index) = index(end);
            break;
```

```
        elseif NCol(row_index) < j && NCol(NIR(row_index)) > j % update aij
in between two elements in row i
            NIR = [NIR; NIR(row_index)];
            NIR(row_index) = index(end);
            break;
        end
            row_index = NIR(row_index);
    end
end

%% Update element in FIC, NIC
col_index = FIC(j);

if col_index == false % update first NICs, FICs for Q
    FIC(j) = index(end);
    NIC(index(end), 1) = 0;
else % update NIC, FIC
    while true
        if  NRow(col_index) == i % update an existing value by aij
            NRow(end) = [];
            break;
        elseif NRow(col_index) > i  % update aij as FIC in column j
            NIC(index(end)) = col_index;
            FIC(j) = length(NCol);
            break;
        elseif NIC(col_index) == 0 % update aij as the last element in column
j
            NIC = [NIC; NIC(col_index)];
            NIC(col_index) = index(end);
            break;
        elseif NRow(col_index) < i && NRow(NIC(col_index)) > i % update aij
in between two elements in column j
            NIC = [NIC; NIC(col_index)];
            NIC(col_index) = index(end);
            break;
        end
        col_index = NIC(col_index);
    end
end

%% Make table and table of firsts information
TA_new = table(index, value, NRow, NCol, NIR, NIC);
TAF_new = table(FIR, FIC);

% [A] = restore_sparse_matrix(TA_new, TAF_new)
end
```

## Appendix I

```
function [aij] = search_in_linked_list(TA, TAF, i, j)

row_index = TAF.FIR(i);

if  row_index == 0 % if value index 'row_index' is 0,the i th row are all
zeros and return aij=0;
```

```
    aij = 0;
    return
else
    while row_index ~= 0 % search all value in the row i
        if TA.NCol(row_index) == j % if NCOL(row_index) equals j, the target
value in jth column is returned
            aij = TA.value(row_index);
            return;
        end
        row_index = TA.NIR(row_index);
    end
    aij = 0; % if not found at the i th row, then return 0;
end

end
```

## Appendix J

```
function [ordered_index] = ordering_scheme_tinney0(TA, TAF)
% Initialization parameters
ordered_index = [];
degree = zeros(1, length(TAF.FIR));

% Tinney 0 ordering
% calculate degree at each node
for i = 1:length(TAF.FIR)
    degree(i) = sum(TA.NRow == i) - 1;
end
%degree = sum(A ~= 0) - 1; % get degree for each node
uni_degrees = unique(degree); % get unique degree number

% sort nodes in degree order (in case of tie, keep natural order)
for k = uni_degrees
    ordered_index = [ordered_index, find(degree == k)]; % order nodes from
the lowest degree
end
end
```

## Appendix K

```
function [ordered_index] = ordering_scheme_tinney1(TA, TAF)
% Initialization parameters
ordered_index = [];
NRow = TA.NRow;
NCol = TA.NCol;
FIR = TAF.FIR;

% Tinney 1 ordering
% calculated the order of A (order_index)
for i = 1:length(FIR)
    order_temp = []; % initialize temporary ordering index
    degree = zeros(1, length(FIR));
    % calculate degree at each node
    for j = 1:length(FIR)
```

```
        degree(j) = sum(NRow == j) - 1;
    end
    uni_degrees = unique(degree); % get unique degree number
    for k = uni_degrees
        order_temp = [order_temp, find(degree == k)];
    end

    % record current the lowest degree node (the node after reducing the
degrees will list at front)
    ordered_index(i) = order_temp(i);

    nz_index = NCol(find(NRow == ordered_index(i))); % get correlative node
index
    for j = nz_index'
        for k = nz_index'
            if ~ismember(k, NCol(find(NRow == j)))
                NRow(end + 1) = j;
                NCol(end + 1) = k;
            end
        end
    end

    % eliminate the current node and reduce the order
    del_index = find(NRow == ordered_index(i));
    NRow(del_index) = [];
    NCol(del_index) = [];
    del_index = find(NCol == ordered_index(i));
    NRow(del_index) = [];
    NCol(del_index) = [];
end

end
```

## Appendix L

```
function [ordered_index] = ordering_scheme_tinney2(TA, TAF)
% Initialization parameters
ordered_index = [];
NRow = TA.NRow;
NCol = TA.NCol;
FIR = TAF.FIR;

% Tinney 2 ordering
% calculated the order of A (order_index)
for i = 1:length(FIR)
    % Initialization parameters
    fills = zeros(1, length(FIR));
    order_temp = []; % initialize temporary ordering index

    % calculate degree at each node
    for j = 1:length(FIR)
        degree(j) = sum(NRow == j) - 1;
```

```
    end
    uni_degrees = unique(degree); % get unique degree number
    for k = uni_degrees
        order_temp = [order_temp, find(degree == k)];
    end

    for n = 1:length(FIR)
        NRow_temp = NRow;
        NCol_temp = NCol;

        nz_index = find(NRow_temp == n); % get correlative nodes
        nz_index(NCol_temp(nz_index) == n) = []; % remove the current node

        for j = NCol_temp(nz_index)'
            for k = NCol_temp(nz_index)'
                if ~ismember(k, NCol_temp(find(NRow_temp == j)))
%                      NRow_temp(end + 1) = j;
%                      NCol_temp(end + 1) = k;
                    fills(n) = fills(n) + 1;
                end
            end
        end
    end

    if isempty(ordered_index) == false
        fills(ordered_index) = -1;
    end

    fills_min = find(fills == min(fills(gt(fills, -1)))); % get the lowest
fills index

    % record current the lowest degree node (the node after reducing the
degrees will list at front)
    if length(fills_min) > 1
        fillsNdegree_min_index = find(degree(fills_min) ==
min(degree(fills_min)));
        ordered_index(i) = fills_min(fillsNdegree_min_index(1));
    else
        ordered_index(i) = fills_min;
    end

    index = find(NRow == ordered_index(i)); % get correlative nodes
    for j = NCol(index)'
        for k = NCol(index)'
            if ~ismember(k, NCol(find(NRow == j)))
                NRow(end + 1) = j;
                NCol(end + 1) = k;
            end
        end
    end

    % eliminate the current node and reduce the order
    del_index = find(NRow == ordered_index(i));
    NRow(del_index) = [];
    NCol(del_index) = [];
```

```
    del_index = find(NCol == ordered_index(i));
    NRow(del_index) = [];
    NCol(del_index) = [];
end

end
```