Final Report

Newton-Raphson Power Flow

University of Washington
EE 556 Winter 2016

Darrell Ross

March 11, 2016

Contents

1 Abstract

A Newton-Raphson Power Flow (NRPF) is implemented in MATLAB without using any symbolic functions. The NRPF is tested on two test circuits, a 4-bus circuit and a 14-bus circuit. Both test circuits converge within three iterations. Data about each system is presented. Limited analysis about some aspects of each circuit is done.

2 Introduction

Power Flow is the process of calculating an electrical power network's state variables, Voltage and Angle, at every bus on the system. Knowing the state of the buses on the system allows for insight into the operational capacity, status, and limits of the system. Every study that is run on a power system has Power Flow at its base. Contingency studies, fault current analysis, arc flash studies, and more all depend on Power Flow results.

This report provides insight into a NRPF algorithm written in MATLAB to provide basic power flow data about power systems while simultaneously providing a good learning experience for young power engineers.

3 Methods

All code for this project was written and executed in MATLAB R2015a. Newton-Raphson Power Flow design was taken from [1]. This section briefly covers how a Newton-Raphson Power Flow works and then covers some details about the code design in MATLAB.

3.1 Newton-Raphson Power Flow

Newton-Raphson Power Flow refers to using a Newton-Raphson method to solve a system of non-linear equations about a power system. The "root" in this case is the point of convergence where the calculated state variables produce the already known power values.

First, implicit power flow equations are formed using (3.1) and (3.2). Real power P equations are formed for PQ and PV buses while reactive power Q equations are formed for PQ buses only.

$$P_i^{inj} = \sum_{j=1}^{N} V_i V_j \left[G_{ij} \cos(\theta_i - \theta_j) + B_{ij} \sin(\theta_i - \theta_j) \right] \tag{3.1}$$

$$Q_i^{inj} = \sum_{j=1}^{N} V_i V_j \left[G_{ij} \sin(\theta_i - \theta_j) - B_{ij} \cos(\theta_i - \theta_j) \right] \tag{3.2}$$

3.1.1 Forming the Jacobian

A Jacobian matrix is formed for the P and Q implicit equations. Partials must be taken of the implicit equations to fill out the Jacobian as shown in (3.3).

$$J = \left[\begin{array}{ccc|ccc} \frac{\partial P_i}{\partial \theta_i} & \cdots & \frac{\partial P_i}{\partial \theta_n} & \frac{\partial P_i}{\partial V_i} & \cdots & \frac{\partial P_i}{\partial V_n} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial P_n}{\partial \theta_i} & \cdots & \frac{\partial P_n}{\partial \theta_n} & \frac{\partial P_n}{\partial V_i} & \cdots & \frac{\partial P_n}{\partial V_n} \\ \hline \frac{\partial Q_i}{\partial \theta_i} & \cdots & \frac{\partial Q_i}{\partial \theta_n} & \frac{\partial Q_i}{\partial V_i} & \cdots & \frac{\partial Q_i}{\partial V_n} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial Q_n}{\partial \theta_i} & \cdots & \frac{\partial Q_n}{\partial \theta_n} & \frac{\partial Q_n}{\partial V_i} & \cdots & \frac{\partial Q_n}{\partial V_n} \end{array}\right] = \left[\begin{array}{c|c} \boldsymbol{J_{11}} & \boldsymbol{J_{12}} \\ \hline \boldsymbol{J_{21}} & \boldsymbol{J_{22}} \end{array}\right] \tag{3.3}$$

Instead of using symbolic math to determine the Jacobian for Power Flow, it turns out that the partial derivatives can be reduced to the formulas shown in (3.4) through (3.11).

For J_{11} with $i \neq j$

$$J_{ij} = \frac{\partial P_i}{\partial \theta_j} = V_i V_j \left[G_{ij} \sin(\theta_i - \theta_j) - B_{ij} \cos(\theta_i - \theta_j)\right] \tag{3.4}$$

For J_{11} with $i = j$

$$\begin{aligned} J_{ii} = \frac{\partial P_i}{\partial \theta_i} &= \sum_{j=1}^{N} V_i V_j \left[-G_{ij} \sin(\theta_i - \theta_j) + B_{ij} \cos(\theta_i - \theta_j)\right] - B_{ii} V_i^2 \\ &= -Q_i - B_{ii} V_i^2 \end{aligned} \tag{3.5}$$

For J_{12} with $i \neq j$

$$J_{ij} = \frac{\partial P_i}{\partial V_j} = V_i \left[G_{ij} \cos(\theta_i - \theta_j) + B_{ij} \sin(\theta_i - \theta_j)\right] \tag{3.6}$$

For J_{12} with $i = j$

$$\begin{aligned} J_{ii} = \frac{\partial P_i}{\partial V_i} &= \sum_{j=1}^{N} V_j \left[G_{ij} \cos(\theta_i - \theta_j) + B_{ij} \sin(\theta_i - \theta_j)\right] + G_{ii} V_i \\ &= \frac{P_i}{V_i} + G_{ii} V_i \end{aligned} \tag{3.7}$$

For J_{21} with $i \neq j$

$$J_{ij} = \frac{\partial Q_i}{\partial \theta_j} = -V_i V_j \left[G_{ij} \cos(\theta_i - \theta_j) + B_{ij} \sin(\theta_i - \theta_j)\right] \tag{3.8}$$

For J_{21} with $i = j$

$$\begin{aligned} J_{ii} = \frac{\partial Q_i}{\partial \theta_i} &= \sum_{j=1}^{N} V_i V_j \left[G_{ij} \cos(\theta_i - \theta_j) + B_{ij} \sin(\theta_i - \theta_j)\right] - G_{ii} V_i^2 \\ &= P_i - G_{ii} V_i^2 \end{aligned} \tag{3.9}$$

For J_{12} with $i \neq j$

$$J_{ij} = \frac{\partial Q_i}{\partial V_j} = V_i \left[G_{ij} \sin(\theta_i - \theta_j) - B_{ij} \cos(\theta_i - \theta_j)\right] \tag{3.10}$$

For J_{12} with $i = j$

$$\begin{aligned} J_{ii} = \frac{\partial Q_i}{\partial V_i} &= \sum_{j=1}^{N} V_j \left[G_{ij} \sin(\theta_i - \theta_j) - B_{ij} \cos(\theta_i - \theta_j)\right] - B_{ii} V_i \\ &= \frac{Q_i}{V_i} - B_{ii} V_i \end{aligned} \tag{3.11}$$

3.1.2 Iterating

Once the jacobian is created, guesses at the value of the unknown state variables are provided, usually with a "flat start" of $1.0\angle 0^o$ pu for the voltage and angle. The power for each bus is calculated using (3.1) and (3.2) and the results compared to known values. If the mismatch between these is greater than a desired threshhold, then the program iterates.

An iteration uses the Jacobian to the error which is then added to the chosen state variables and the cycle repeats until the power mismatch is sufficiently small - called convergence. The algorithm will not converge for all possible input combinations.

A step-by-step guide follows:

1. Calculate power flow mismatches. If mismatch is sufficiently small, stop. Otherwise, continue. A common mismatch level is 0.001 pu although there is no requirement that it not be smaller. Mismatches are calculated using (3.1) and (3.2) and subtracting the provided P and Q values for each.

$$\begin{bmatrix} P_{\text{mismatch}} \\ Q_{\text{mismatch}} \end{bmatrix} = \begin{bmatrix} P(\theta^n, V^n) \\ Q(\theta^n, V^n) \end{bmatrix} - \begin{bmatrix} P^n \\ Q^n \end{bmatrix}$$

2. Calculate the Jacobian J with values using the functions derived in (3.4) through (3.11).

3. Calculate the deltas for θ and V:

$$\begin{bmatrix} \Delta\theta \\ \Delta V \end{bmatrix} = -J^{-1} \cdot \begin{bmatrix} P_{\text{mismatch}} \\ Q_{\text{mismatch}} \end{bmatrix}$$

4. Update state variables:

$$\begin{bmatrix} \theta^{n+1} \\ V^{n+1} \end{bmatrix} = \begin{bmatrix} \theta^n \\ V^n \end{bmatrix} + \begin{bmatrix} \Delta\theta \\ \Delta V \end{bmatrix}$$

3.1.3 Data Collection

Once the mismatch values have satisfied requirements of convergence, the power flow at each bus is calculated as well as the sending power for each branch.

3.2 Branch Power Flow

Branch power flow is the power sent down or received from each branch. The power will be different depending on where it is measured due to losses on the branches and sent versus received power. That is, there are four different possible values:

- branch flow sent from bus i to bus j
- branch flow received at bus i from bus j
- branch flow sent from bus j to bus i
- branch flow received at bus j from bus i

Power flow through line impedance:

$$S_{ij} = V_i I_{ij}^* = V_i \left(\frac{V_i - V_j}{R + jX} \right)^*$$

Power flow from the sending end i at this line:

$$S_{ij,\text{sending}} = V_i \left(\frac{V_i - V_j}{R + jX} \right)^* - jV_j^2 B \tag{3.12}$$

Power flow from the receiving end i at this line:

$$S_{ij,\text{receiving}} = V_i \left(\frac{V_i - V_j}{R + jX} \right)^* + jV_j^2 B \tag{3.13}$$

Note that in (3.12) *and* (3.13), $B = \frac{Imag(Y)}{2}$

The reverse power flows, $S_{ji,\text{sending}}$ and $S_{ji,\text{receiving}}$ can be calculated by reserving the the i and j indexes used.

3.3 MATLAB Algorithm

The MATLAB algorithm is capable of representing three different bus types:

- Slack Bus - only voltage and angle are "known", usually set at $1.0\angle 0^o$ pu
- PV Bus - only real power and voltage are known
- PQ Bus - only real power and reactive power injections are known

The program can represent buses as well as branches between buses modeled using a Π line model. Bus elements have data shown in Table 3.1 while branch elements have data shown in Table 3.2.

The program is written modularly putting different functionality in different files which allows for easy access to execute any one of them. In addition, all data is mapped into a large results structure which is returned. There are some small utilities designed to be used for printing out the data.

There is even a regression test which can be run to ensure that recent changes to the analysis has not changed the results.

A brief description of all project files along with instructions on how to run them is included in Appendix C.

4 Analysis: Four Bus Test System

A circuit diagram of the four-bus test system is shown in Figure 4.1 with the bus and branch data provided in Table A.1 and Table A.2, respectively. The system has given bases of $S_B = 100$ MVA and $V_B = 230$ kV.

The four-bus test system solves in just three iterations. The final solution is shown in Table 4.1.

Data	Description
Bus	Bus that data relates to
Type	Slack, PQ, or PV
P_G	Generated Real Power at Bus
Q_G	Generated Reactive Power at Bus
P_L	Real Power Load at Bus
Q_L	Reactive Power Load at Bus
V	Voltage at Bus
θ	Voltage Angle at Bus
G	Bus Shunt Inductance
B	Bus Shunt Capacitance

Table 3.1: Bus data which the power flow uses.

Data	Description
From	Bus number which branch starts at
To	Bus number which branch ends at
R(pu)	Resistance of branch
X(pu)	Reactance of branch
G(pu)	Branch Shunt Inductance
B(pu)	Branch Shunt Capacitance

Table 3.2: Branch data which the power flow uses.

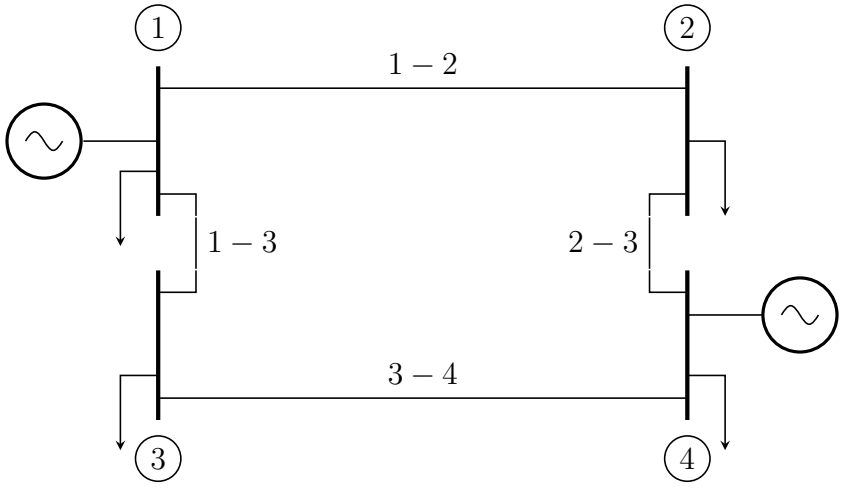


Figure 4.1: A diagram of the four-bus test circuit.

Bus	P_G(MW)	Q_G(MVAr)	P_L(MW)	Q_L(MVAr)	$V\angle\theta^o$(pu)
1	186.809	109.501	50.000	30.990	$1.00000\angle 0^o$
2	0	0	170.000	105.350	$0.98242\angle -0.97612^o$
3	0	0	200.000	123.940	$0.96900\angle -1.87218^o$
4	318.000	176.228	80.000	49.580	$1.02000\angle 1.52306^o$

Table 4.1: Bus power and voltage solution for the four-bus test circuit.

4.1 Jacobian Terms

$$\begin{bmatrix} 45.4429 & 0 & -26.3648 & 8.8818 & 0 \\ 0 & 41.2687 & -15.4209 & 0 & 8.1328 \\ -26.3648 & -15.4209 & 41.7857 & -5.2730 & -3.0842 \\ -9.0886 & 0 & 5.2730 & 44.2290 & 0 \\ 0 & -8.2537 & 3.0842 & 0 & 40.4590 \end{bmatrix}$$

Table 4.2: Iteration 1 Jacobian

$$\begin{bmatrix} 44.3749 & 0 & -25.6778 & 7.1397 & 0 \\ 0 & 39.7018 & -14.7736 & 0 & 5.9619 \\ -26.1256 & -15.1217 & 41.2473 & -4.1296 & -2.1828 \\ -10.3562 & 0 & 6.2998 & 43.0530 & 0 \\ 0 & -9.6597 & 3.8597 & 0 & 38.4642 \end{bmatrix}$$

Table 4.3: Iteration 2 Jacobian

$$\begin{bmatrix} 44.3270 & 0 & -25.6508 & 7.0969 & 0 \\ 0 & 39.6096 & -14.7398 & 0 & 5.8755 \\ -26.1026 & -15.0938 & 41.1963 & -4.1183 & -2.1655 \\ -10.3721 & 0 & 6.3048 & 42.9755 & 0 \\ 0 & -9.6932 & 3.8683 & 0 & 38.3186 \end{bmatrix}$$

Table 4.4: Iteration 3 Jacobian

4.2 Jacobian Freeze

When the Jacobian is frozen after the first iteration, the power flow still solves, but it takes four iterations to converge instead of three. This is likely because the Jacobian is essentially the "slope" of the line. That is, in the Taylor series expansion, we would stop at the first derivative term but the expansion could still work, just at a different pace, in this case a slower pace.

In simple terms, Taylor Series Expansion follows:

$$f(x) = f(x^0) + f'(x^0)(x - x^0) + \frac{1}{2}f''(x^0)(x - x^0)^2 + ...$$

With a frozen Jacobian, we instead get:

$$f(x) = f(x^0) + f'(x^0)(x - x^0) + \frac{1}{2}f'(x^0)(x - x^0)^2 + ...$$

Note that convergence is still possible with this scenario but that the frozen first derivative will mean no continuous improvement.

An interesting bit would be to determine if there is any speed benefit for larger systems to not redoing the Jacobian. Or more accurately, not re-inverting the Jacobian as the large matrix inversion is the most time consuming part of a Newton-Raphson based Power Flow. There is likely a point where the size of the system is so large that it takes less time to converge without re-inverting.

4.3 Voltage Plots

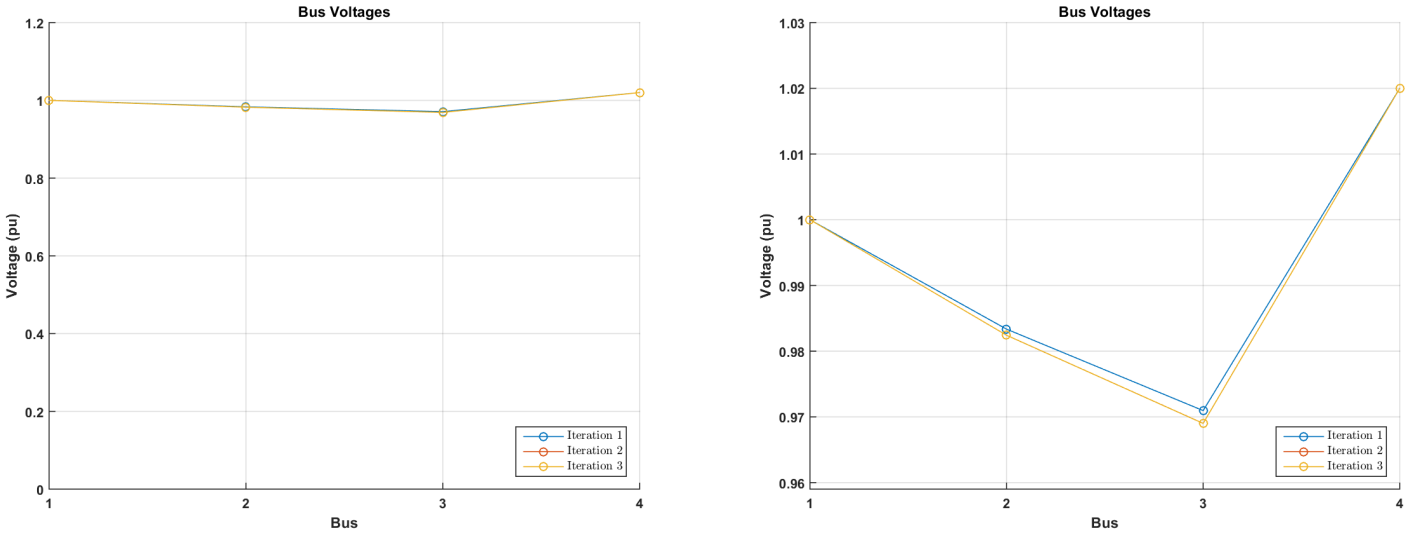


Figure 4.2: The voltages for each iteration over all buses are shown in per unit zoomed out (left) and zoomed in (right). The Slack bus (Bus 1) and the PV bus (Bus 4) both have constant voltage which is represented here by the voltage values not changing for those two buses.

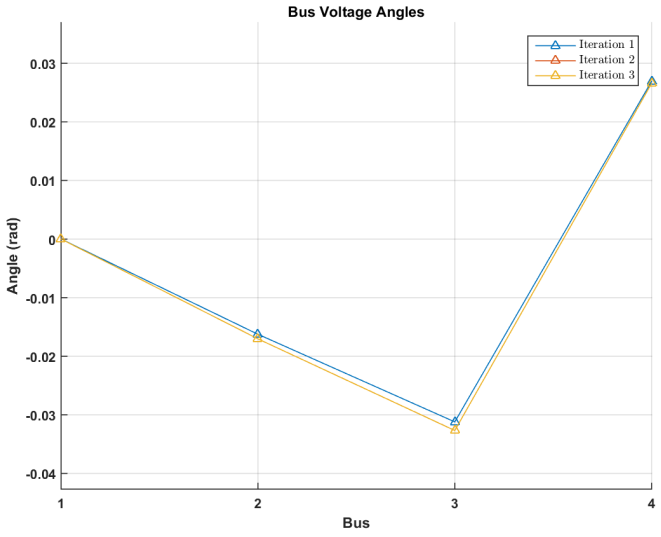


Figure 4.3: The voltages angles for each iteration over all buses are shown in radians.

4.4 Branch Flow

The power flowing from Bus 3 to Bus 4 (bold in Table 4.5)is $-103.3048 - j60.3585$ MVA.

From	To	S_{sending}(MVA)
1	2	38.6915 + j22.2984
1	3	98.1175 + j61.2123
2	4	-131.6625 - j74.1115
3	**4**	**-103.3048 - j60.3585**

Table 4.5: Real and reactive power flowing on each branch for the final solution to the four-bus test circuit.

From the load perspective, the negative values make sense because Bus 3 has a significant load at it and Bus 4 has a generator. Reviewing the final solution in Table 4.1, Bus 4 is generating 318 MW and clearly is not consuming it all.

From a voltage perspective:

- reactive power flows to lower voltage and Bus 3 has a lower voltage than Bus 4 so the negative reactive power flow fits
- the negative angle difference between Bus 3 and Bus 4 also supports the negative flow of real power

4.5 Low Bus Voltage

The lowest bus voltage in the system, as shown in Table 4.1, is Bus 3 with $0.96900\angle -1.87218^o$ pu.

One idea I had to fix this was to add a capacitor bank at Bus 3 to inject 100 MVAr. This effectively adds a Q_G of 0.1 pu to Bus 3. The solution to this is shown in Table 4.6.

It worked really well, bringing the voltage at Bus 3 up to $0.99350\angle -1.87218^o$ pu without adversely effecting the other bus voltages. Presumably, the next lowest voltage on Bus 2 could be fixed in a similar fashion. The reactive power loads on this system are pretty extreme.

Bus	P_G(MW)	Q_G(MVAr)	P_L(MW)	Q_L(MVAr)	$V\angle\theta^o$(pu)
1	186.063	44.170	50.000	30.990	$1.00000\angle 0^o$
2	0	0	170.000	105.350	$0.98242\angle -0.97612^o$
3	0	100.000	200.000	123.940	$0.99350\angle -1.87218^o$
4	318.000	137.337	80.000	49.580	$1.02000\angle 1.52306^o$

Table 4.6: Bus power and voltage solution for the four-bus test circuit with a 100 MVAr capacitor bank added to Bus 3.

5 Analysis: Fourteen Bus System

A circuit diagram of the fourteen-bus test system is shown in Figure 5.1 with the bus and branch data provided in Table B.1 and Table B.2, respectively.

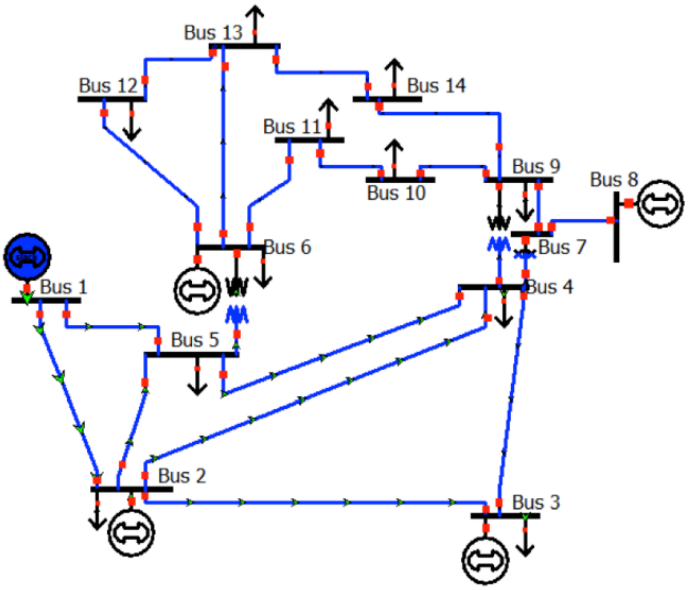


Figure 5.1: A diagram of the 14-bus test circuit.

Bus	P_G(MW)	Q_G(MVAr)	P_L(MW)	Q_L(MVAr)	$V\angle\theta^o$(pu)
1	232.375	-23.531	0	0	$1.06000\angle 0^o$
2	40.000	27.413	21.700	12.700	$1.04500\angle -4.9517^o$
3	0	18.014	94.200	19.000	$1.01000\angle -12.6157^o$
4	0	0	47.800	-3.900	$1.02949\angle -10.4201^o$
5	0	0	7.600	1.600	$1.03488\angle -8.9558^o$
6	0	40.364	11.200	7.500	$1.07000\angle -14.6718^o$
7	0	0	0	0	$1.05588\angle -13.5533^o$
8	0	13.839	0	0	$1.09000\angle -13.5533^o$
9	0	0	29.500	16.600	$1.04967\angle -15.1693^o$
10	0	0	9.000	5.800	$1.04582\angle -15.3657^o$
11	0	0	3.500	1.800	$1.05430\angle -15.1455^o$
12	0	0	6.100	1.600	$1.05469\angle -15.5160^o$
13	0	0	13.500	5.800	$1.04949\angle -15.5754^o$
14	0	0	14.900	5.000	$1.03154\angle -16.3530^o$

Table 5.1: Bus power and voltage solution for the fourteen-bus test circuit.

5.1 Jacobian Terms

The Jacobian of the first iteration is shown in Table 5.3 and Table 5.4. All remaining data is provided in Appendix B. Per request, presumably to verify my algorithm works, the Jacobian at index $(3, 3)$ for all iterations is provided in Table 5.2.

Iteration	$\text{Jacobian}_{(3,3)}$
1	38.6240
2	41.068017
3	40.578422

Table 5.2: Jacobian entry at index $(3, 3)$ for the fourteen-bus test system.

32.7276	-5.0470	-5.3461	-5.4277	0	0	0	0	0	0	0
-5.0470	10.1665	-5.1195	0	0	0	0	0	0	0	0
-5.3461	-5.1195	38.6240	-21.5786	0	-4.7819	0	-1.7980	0	0	0
-5.4277	0	-21.5786	35.7410	-4.2457	0	0	0	0	0	0
0	0	0	-4.2457	18.5546	0	0	0	0	-4.3807	-3.3983
0	0	-4.7819	0	0	20.0599	-6.1879	-9.0901	0	0	0
0	0	0	0	0	-6.1879	6.1879	0	0	0	0
0	0	-1.7980	0	0	-9.0901	0	24.2825	-10.3654	0	0
0	0	0	0	0	0	0	-10.3654	14.7683	-4.4029	0
0	0	0	0	-4.3807	0	0	0	-4.4029	8.7836	0
0	0	0	0	-3.3983	0	0	0	0	0	5.6503
0	0	0	0	-6.5299	0	0	0	0	0	-2.2520
0	0	0	0	0	0	0	-3.0291	0	0	0
1.7619	2.0058	-10.6087	6.8410	0	0	0	0	0	0	0
1.7777	0	6.8410	-9.7061	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	-5.3261	3.9020	0	0
0	0	0	0	0	0	0	3.9020	-5.7829	1.8809	0
0	0	0	0	2.0919	0	0	0	1.8809	-3.9728	0
0	0	0	0	1.6328	0	0	0	0	0	-4.1218
0	0	0	0	3.3159	0	0	0	0	0	2.4890
0	0	0	0	0	0	0	1.4240	0	0	0

Table 5.3: Iteration 1 Jacobian, Columns 1-11

0	0	-1.7619	-1.7777	0	0	0	0	0	0	0
0	0	-2.0058	0	0	0	0	0	0	0	0
0	0	10.4173	-6.8410	0	0	0	0	0	0	0
0	0	-6.8410	9.4299	0	0	0	0	0	0	0
-6.5299	0	0	0	0	0	0	-2.0919	-1.6328	-3.3159	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	-3.0291	0	0	0	5.3261	-3.9020	0	0	0	-1.4240
0	0	0	0	0	-3.9020	5.7829	-1.8809	0	0	0
0	0	0	0	0	0	-1.8809	3.6991	0	0	0
-2.2520	0	0	0	0	0	0	0	3.9082	-2.4890	0
11.0969	-2.3150	0	0	0	0	0	0	-2.4890	6.5080	-1.1370
-2.3150	5.3440	0	0	0	-1.4240	0	0	0	-1.1370	2.5610
0	0	38.0154	-21.5786	-4.7819	-1.7980	0	0	0	0	0
0	0	-21.5786	34.1260	0	0	0	0	0	0	0
0	0	-4.7819	0	19.0381	-9.0901	0	0	0	0	0
0	1.4240	-1.7980	0	-9.0901	23.9025	-10.3654	0	0	0	-3.0291
0	0	0	0	0	-10.3654	14.7683	-4.4029	0	0	0
0	0	0	0	0	0	-4.4029	8.2104	0	0	0
2.4890	0	0	0	0	0	0	0	5.2056	-2.2520	0
-6.9419	1.1370	0	0	0	0	0	0	-2.2520	10.2425	-2.3150
1.1370	-2.5610	0	0	0	-3.0291	0	0	0	-2.3150	5.3440

Table 5.4: Iteration 1 Jacobian, Columns 12-22

5.2 Jacobian Freeze

When the Jacobian is frozen after the first iteration, the power flow still solves but it takes five iterations to converge instead of three. Similar to the four-bus system, this makes sense. It also makes sense that it would take more iterations since the system is bigger and has to solve more variables.

5.3 Voltage Plots

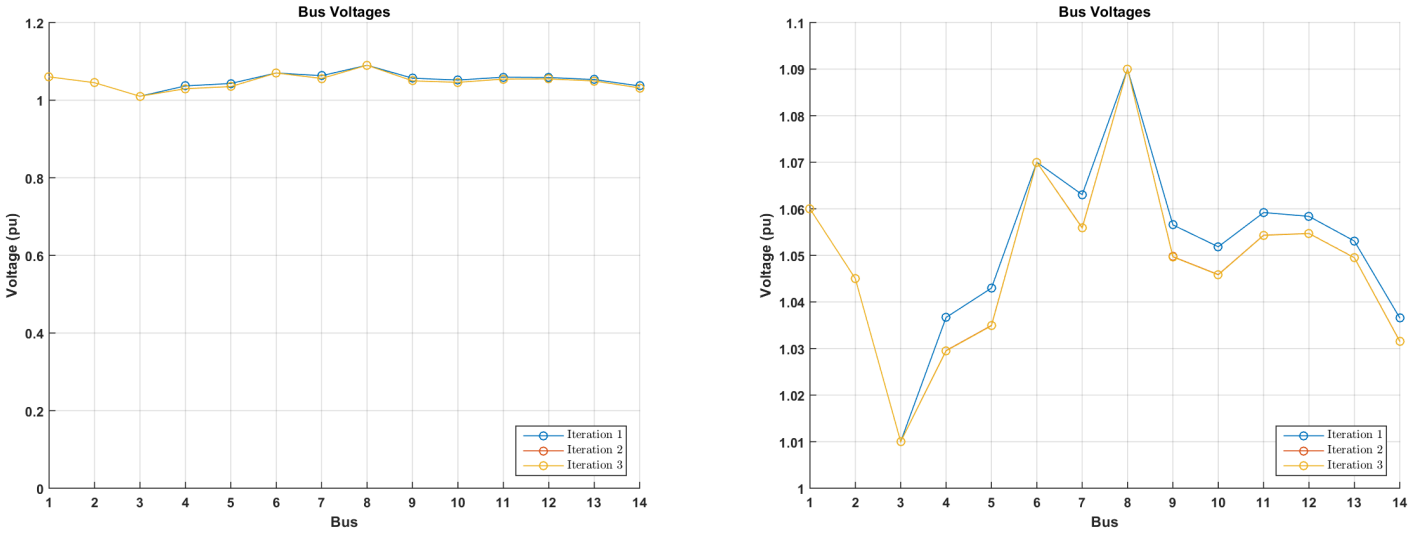


Figure 5.2: The voltages for each iteration over all buses are shown in per unit zoomed out (left) and zoomed in (right). The parts where the voltages do not appear to move between iterations are the PV buses with constant voltage.

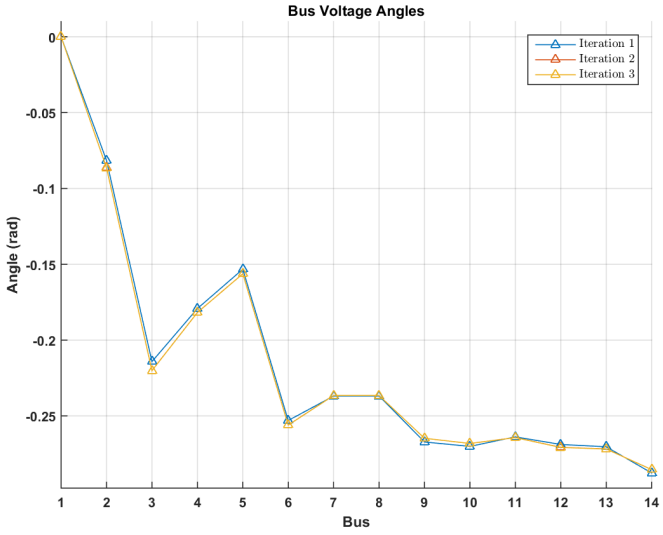


Figure 5.3: The voltage angles for each iteration over all buses are shown in radians. They barely moved.

5.4 Branch Flow

The branch power flow sending as seen from the sending end is shown in Table 5.5.

To examine one flow from a load perspective, the flow from Bus 6 to Bus 11 (in bold) is sending 7.11 + 5.07 MVA. Reviewing the bus data in Table B.1, it is clear that Bus 11 had a load of 3.5 + 1.8

From	To	S_{sending}(MVA)	From	To	S_{sending}(MVA)
1	2	155.951548-j20.186395	**6**	**11**	**7.110914+j5.072284**
1	5	76.423769-j3.344173	6	12	7.798562+j2.703653
2	3	72.118603+j3.665842	6	13	17.652295+j7.993629
2	4	55.687283-j8.317336	7	8	0.000000-j20.454673
2	5	41.144081-j7.844704	7	9	28.411396+j6.354714
3	4	-24.227618-j2.270042	9	10	5.479761+j2.727774
4	5	-62.310861+j7.394812	9	14	9.518168+j2.648791
4	7	28.411396-j12.212948	10	11	-3.531057-j3.100961
4	9	16.086537-j3.068813	12	13	1.625424+j0.951433
5	6	43.761770-j12.235647	13	14	5.553718+j2.711432

Table 5.5: Real and reactive power sending as seen from the "From" bus on each branch for the final solution to the fourteen-bus test circuit.

MVA. Some of the power flows to that load and the remaining power flows from Bus 11 to Bus 10 where there is an additional load of 9 + 5.8 MVA. It is clear that the load at Bus 10 will not be met by the residual especially after losses on the 10-11 branch, so the flow from Bus 9 to Bus 10 supplies the remainder. The branch power flows in Table 5.5 support this explanation.

From a voltage perspective, evaluating the flow from Bus 6 to Bus 11 also makes sense:

- reactive power flows to lower voltage and Bus 11 has a lower voltage than Bus 6 so the positive reactive power flow from Bus 6 to Bus 11 fits
- the voltage angle difference from Bus 6 to Bus 11 ends up positive (-14.67^o-(-15.14^o))=+0.47 so the real power flow from Bus 6 to Bus 11 is positive

5.5 Low Bus Voltage

The lowest voltage in the fourteen-bus test system is on Bus 3 at 1.01 pu. This is a PV bus which means it has constant voltage. Since it is PV, this means that we cannot change the voltage through the model. Since Q is unknown, the Q-equation is explicit and is not used. I could change it to a PQ bus.

On another note, why would we want to raise the voltage on a node that is at 1.01 pu? It seems to me that we should be looking to lower the voltage on the highest one which is at 1.09 pu which is, incidentally, also a PV bus. Really kind of a weird system. The voltage is far less consistent than the four-bus test system.

Interestingly, if we change Bus 3 to be a PQ bus, the voltage drops to 0.990$\angle$-12.424^o.

6 Conclusions

The Newton-Raphson Power Flow is an effective way to solve load flows with three-phase balanced power systems. There are numerous ways in which this algorithm could be enhanced. One of the very first would be to take into account the ratings of various elements.

For example, the textbook describes a way of limiting the reactive power on a PQ element by changing it to be a PV bus once it has reached it's maximum rated Q. Setting it's P_Q to the maximum rating at that point before continuing iterating allows taking into account reactive power maximums.

A Four Bus Test System

Bus	Type	P_G(MW)	Q_G(MW)	P_L(MW)	Q_L(MW)	V(pu)	θ(rad)	B(pu)
1	Slack	—	—	50	30.99	1.0	?	0.05
2	PQ	0	0	170	105.35	?	?	?
3	PQ	0	0	200	123.94	?	?	?
4	PV	318	?	80	49.58	1.02	?	?

Table A.1: Bus data for the four-bus test circuit shown in Figure 4.1.

From	To	r(pu)	x(pu)	$y/2$(pu)	b(pu)
1	2	0.01008	0.05040	j0.05125	0.1025
1	3	0.00744	0.03720	j0.03875	0.0775
2	4	0.00744	0.03720	j0.03875	0.0775
3	4	0.01272	0.06360	j0.06375	0.1275

Table A.2: Branch data for the four-bus test circuit shown in Figure 4.1. The $y/2$ value was given but since my algorithm expects the b-value, I have indicated it above in the last column, $b = \text{Imag}(2 \cdot y/2)$.

B Fourteen Bus Test System

Bus	Type	P_G(MW)	Q_G(MW)	P_L(MW)	Q_L(MW)	V(pu)	θ(rad)	G(pu)	B(pu)
1	Slack	?	?	0	0	1.06	0	0	0
2	PV	40	?	21.7	12.7	1.045	?	0	0
3	PV	0	?	94.2	19	1.01	?	0	0
4	PQ	0	0	47.8	-3.9	?	?	0	0
5	PQ	0	0	7.6	1.6	?	?	0	0
6	PV	0	?	11.2	7.5	1.07	?	0	0
7	PQ	0	0	0	0	?	?	0	0
8	PV	0	?	0	0	1.09	?	0	0.061250
9	PQ	0	0	29.5	16.6	?	?	0	0.190000
10	PQ	0	0	9	5.8	?	?	0	0
11	PQ	0	0	3.5	1.8	?	?	0	0
12	PQ	0	0	6.1	1.6	?	?	0	0
13	PQ	0	0	13.5	5.8	?	?	0	0
14	PQ	0	0	14.9	5	?	?	0	0

Table B.1: Bus data for the fourteen-bus test circuit shown in Figure 5.1.

From	To	R(pu)	X(pu)	G(pu)	B(pu)	From	To	R(pu)	X(pu)	G(pu)	B(pu)
1	2	0.01938	0.05917	0	0.05280	6	11	0.09498	0.19890	0	0
1	5	0.05403	0.22304	0	0.04920	6	12	0.12291	0.25581	0	0
2	3	0.04699	0.19797	0	0.04380	6	13	0.06615	0.13027	0	0
2	4	0.05811	0.17632	0	0.03400	7	8	0	0.17615	0	0
2	5	0.05695	0.17388	0	0.03460	7	9	0	0.11001	0	0
3	4	0.06701	0.17103	0	0.01280	9	10	0.03181	0.08450	0	0
4	5	0.01335	0.04211	0	0	9	14	0.12711	0.27038	0	0
4	7	0	0.20912	0	0	10	11	0.08205	0.19207	0	0
4	9	0	0.55618	0	0	12	13	0.22092	0.19988	0	0
5	6	0	0.25202	0	0	13	14	0.17093	0.34802	0	0

Table B.2: Branch data for the fourteen-bus test circuit shown in Figure 5.1.

C Attachments

C.1 Matlab Full Program

The full program was run which outputs all possible data and that is attached directly to the end of this report. Hopefully all data you need is present in the primary sections because parsing the entire data dump is arduous.

C.2 Matlab Code

The MATLAB code will be included in a zipped file along with the digital version of this report. The files included in that archive are briefly described below.

- **Master files**
  - runall.m - used to run all tests and generate all figures
  - master.m - master function for running multiple tests
  - tsa.m - loads all data for Test System A
  - tsb.m - loads all data for Test System B
- **Newton-Raphson Implementation**
  - nrpf.m - Full Newton-Raphson Power Flow
  - nrpf_jac.m - Single interation calculation of Newton-Raphson
  - pfunc.m - Implementation of real power estimate function (3.1)
  - pbranch.m - runs through a single branch of the summation performed within pfunc.m
  - qfunc.m - Implementation of reactive power estimate function (3.2)
  - qbranch.m - runs through a single branch of the summation performed within qfunc.m
  - jptheta.m - Implementation of $\frac{\partial P_i}{\partial \theta_j}$ in (3.4) and (3.5)
  - jpv.m - Implementation of $\frac{\partial P_i}{\partial V_j}$ in (3.6) and (3.7)
  - jqtheta.m - Implementation of $\frac{\partial Q_i}{\partial \theta_j}$ in (3.8) and (3.9)
  - jqv.m - Implementation of $\frac{\partial Q_i}{\partial V_j}$ in (3.10) and (3.11)
- **Newton-Raphson Helper Functions**
  - ybus.m - Calculates the ybus from bus and branch data
  - jacobianCount.m - Returns number of implicit P and Q functions
  - mismatch.m - Calculates mismatch during NR iterations
  - figureplot.m - Plots figures and makes them look good (written by me last term during a different course)
  - parse_branch_data.m - parses branch data into desired variables
- **Utilities**
  - latexprint.m - used to generate matrices for LaTeX.
  - nearzero.m - forces a value to be zero if the input is within a set threshold which defaults to 10^{-6}. This helps readability.
  - nrpf_test.m - runs a regression test on the algorithm which is used to ensure changes to the algorithm do not break it.
  - compare_maps.m - used to compare two maps — primarily useful when comparing regression test results.

C.3 Matlab Instructions

I left extensive comments in the MATLAB code so that readers can learn how each function works. To further facilitate its use, a few instructions are provided here.

- **How to run a Newton-Raphson Power Flow**

  1. Create your branch and bus data matrices similar to those shown in Table B.1 and Table B.2. To acquire the Test System A or Test System B data, use the following commands:

     ```
     [busdata,branchdata]=tsa();
     [busdata,branchdata]=tsb();
     ```

  2. Use the command "nrpf" to run the Newton-Raphson Power Flow.

     ```
     [datamap,err]=nrpf(busdata,branchdata);
     ```

  3. Inspect your data by reviewing the output "datamap". There is a print function which will print the entire contents of the datamap to the screen:

     ```
     print_nrpf(datamap);
     ```

  4. Inspect specific data by reviewing the keys of the datamap and accessing each piece individually. Use "datamap.keys" to list the keys available and then access the data by using the key in quotes.

     ```
     datamap.keys
     iter1map=datamap('iter1')
     iter1map.keys
     iter1map('jacobian')
     ```

- **How to create voltage plots**

  1. Run the NRPF.
  2. Plot the voltages by passing the entire datamap to the "plot_voltages" function. The second input is the name that will be used when saving images of the plotted figures.

     ```
     plot_voltages(datamap,'imagename')
     ```

- **How to freeze the first Jacobian**

  1. The nrpf() function has several optional inputs and freezing the jacobian is the last one so all the optional ones must be supplied in order to use it.

     ```
     [datamap,err]=nrpf(busdata,branchdata,0,0.001,10,1);
     ```

     The inputs beyond the bus and branch data here are:

     - 0 = tells it not to print every single iteration of data
     - 0.001 = tells it to aim for a mismatch threshold of 0.001
     - 10 = tells it to run for a maximum of 10 iterations
     - 1 = tells it to freeze after the first jacobian

References

[1] J. Duncan Glover, Mulukutla S. Sarma, and Thomas Overbye. *Power System Analysis and Design, Fifth Edition*. 5th. Cengage Learning, Jan. 2011. ISBN: 9781111425777. URL: http://amazon.com/o/ASIN/1111425779/.

EE 556 Project

Runs entire project Dumps all data to the screen Creates all images for the report

Contents

Run NRPF

```
doTestA=1;
doTestB=1;
printIterations=0;
[abranch,abus,amap,bbranch,bbus,bmap]=master(printIterations,doTestA,doTestB);
```

```
------------------------------------------------------
Test System A: 4 Bus
, mismatch target of S=0.001000 met
Final Results: 3 iterations
name =
                      PG           QG           PL           QL           V
        Bus_1      1.86809      1.09501      0.50000      0.30990      1.00000
        Bus_2            0            0      1.70000      1.05350      0.98242
        Bus_3            0            0      2.00000      1.23940      0.96900
        Bus_4      3.18000      1.76228      0.80000      0.49580      1.02000

                  Th(deg)
        Bus_1           0
        Bus_2     -0.97612
        Bus_3     -1.87218
        Bus_4      1.52306

------------------------------------------------------
Test System B: 14 Bus
, mismatch target of S=0.001000 met
Final Results: 3 iterations
name =
                      PG           QG           PL           QL           V
        Bus_1      2.32375     -0.23531            0            0      1.06000
        Bus_2      0.40000      0.27413      0.21700      0.12700      1.04500
        Bus_3            0      0.18014      0.94200      0.19000      1.01000
        Bus_4            0            0      0.47800     -0.03900      1.02949
        Bus_5            0            0      0.07600      0.01600      1.03488
        Bus_6            0      0.40364      0.11200      0.07500      1.07000
        Bus_7            0            0            0            0      1.05588
        Bus_8            0      0.13839            0            0      1.09000
        Bus_9            0            0      0.29500      0.16600      1.04967
       Bus_10            0            0      0.09000      0.05800      1.04582
```

```
       Bus_11            0            0      0.03500      0.01800      1.05430
       Bus_12            0            0      0.06100      0.01600      1.05469
       Bus_13            0            0      0.13500      0.05800      1.04949
       Bus_14            0            0      0.14900      0.05000      1.03154

                  Th(deg)
        Bus_1           0
        Bus_2     -4.95178
        Bus_3    -12.61575
        Bus_4    -10.42016
        Bus_5     -8.95580
        Bus_6    -14.67188
        Bus_7    -13.55339
        Bus_8    -13.55339
        Bus_9    -15.16938
       Bus_10    -15.36576
       Bus_11    -15.14555
       Bus_12    -15.51605
       Bus_13    -15.57542
       Bus_14    -16.35309
```

Print all Test System A Data

```
fprintf('----------------------------------------\n');
fprintf('TEST SYSTEM A\n');
print_nrpf(amap);
```

```
----------------------------------------
TEST SYSTEM A
P
    1.3681
   -1.7000
   -2.0000
    2.3800
PG
    1.8681
         0
         0
    3.1800
PL
    0.5000
    1.7000
    2.0000
    0.8000
Q
    0.7851
   -1.0535
   -1.2394
    1.2665
QG
    1.0950
         0
         0
```

```
    1.7623
QL
    0.3099
    1.0535
    1.2394
    0.4958
T
         0
   -0.0170
   -0.0327
    0.0266
V
    1.0000
    0.9824
    0.9690
    1.0200
bffr
   1.0000 + 0.0000i   2.0000 + 0.0000i   0.3869 + 0.3767i
   1.0000 + 0.0000i   3.0000 + 0.0000i   0.9812 + 0.7284i
   2.0000 + 0.0000i   4.0000 + 0.0000i  -1.3128 - 0.6290i
   3.0000 + 0.0000i   4.0000 + 0.0000i  -1.0213 - 0.4244i
bffs
   1.0000 + 0.0000i   2.0000 + 0.0000i   0.3869 + 0.1717i
   1.0000 + 0.0000i   3.0000 + 0.0000i   0.9812 + 0.5734i
   2.0000 + 0.0000i   4.0000 + 0.0000i  -1.3179 - 0.7785i
   3.0000 + 0.0000i   4.0000 + 0.0000i  -1.0370 - 0.6633i
bfrr
   2.0000 + 0.0000i   1.0000 + 0.0000i  -0.3813 - 0.1640i
   3.0000 + 0.0000i   1.0000 + 0.0000i  -0.9661 - 0.5267i
   4.0000 + 0.0000i   2.0000 + 0.0000i   1.3282 + 0.8700i
   4.0000 + 0.0000i   3.0000 + 0.0000i   1.0404 + 0.7681i
bfrs
   2.0000 + 0.0000i   1.0000 + 0.0000i  -0.3880 - 0.3618i
   3.0000 + 0.0000i   1.0000 + 0.0000i  -0.9756 - 0.6719i
   4.0000 + 0.0000i   2.0000 + 0.0000i   1.3368 + 0.7090i
   4.0000 + 0.0000i   3.0000 + 0.0000i   1.0545 + 0.5032i
iter1
Pmm
    1.5966
    1.9395
   -2.2129
Qmm
    0.4465
    0.8345
T
         0
   -0.0162
   -0.0312
    0.0269
V
    1.0000
    0.9834
    0.9710
    1.0200
jacobian
   45.4429         0  -26.3648    8.8818         0
         0   41.2687  -15.4209         0    8.1328
```

```
  -26.3648  -15.4209   41.7857   -5.2730   -3.0842
   -9.0886         0    5.2730   44.2290         0
         0   -8.2537    3.0842         0   40.4590
iter2
Pmm
    0.0323
    0.0645
   -0.0359
Qmm
    0.0342
    0.0620
T
         0
   -0.0170
   -0.0327
    0.0266
V
    1.0000
    0.9824
    0.9690
    1.0200
jacobian
   44.3749         0  -25.6778    7.1397         0
         0   39.7018  -14.7736         0    5.9619
  -26.1256  -15.1217   41.2473   -4.1296   -2.1828
  -10.3562         0    6.2998   43.0530         0
         0   -9.6597    3.8597         0   38.4642
iter3
Pmm
   1.0e-03 *
    0.0339
    0.1455
   -0.0424
Qmm
   1.0e-03 *
    0.0414
    0.1646
T
         0
   -0.0170
   -0.0327
    0.0266
V
    1.0000
    0.9824
    0.9690
    1.0200
jacobian
   44.3270         0  -25.6508    7.0969         0
         0   39.6096  -14.7398         0    5.8755
  -26.1026  -15.0938   41.1963   -4.1183   -2.1655
  -10.3721         0    6.3048   42.9755         0
         0   -9.6932    3.8683         0   38.3186
ybus
   8.9852 -44.7860i  -3.8156 +19.0781i  -5.1696 +25.8478i   0.0000 + 0.0000i
  -3.8156 +19.0781i   8.9852 -44.8360i   0.0000 + 0.0000i  -5.1696 +25.8478i
  -5.1696 +25.8478i   0.0000 + 0.0000i   8.1933 -40.8638i  -3.0237 +15.1185i
```

```
   0.0000 + 0.0000i  -5.1696 +25.8478i  -3.0237 +15.1185i   8.1933 -40.8138i
```

Print all Test System B Data

```
fprintf('----------------------------------------\n');
fprintf('TEST SYSTEM B\n');
print_nrpf(bmap);
```

```
----------------------------------------
TEST SYSTEM B
P
    2.3238
    0.1830
   -0.9420
   -0.4780
   -0.0760
   -0.1120
         0
         0
   -0.2950
   -0.0900
   -0.0350
   -0.0610
   -0.1350
   -0.1490
PG
    2.3238
    0.4000
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
PL
         0
    0.2170
    0.9420
    0.4780
    0.0760
    0.1120
         0
         0
    0.2950
    0.0900
    0.0350
    0.0610
    0.1350
```

    0.1490
Q
   -0.2353
    0.1471
   -0.0099
    0.0390
   -0.0160
    0.3286
         0
    0.1384
   -0.1660
   -0.0580
   -0.0180
   -0.0160
   -0.0580
   -0.0500
QG
   -0.2353
    0.2741
    0.1801
         0
         0
    0.4036
         0
    0.1384
         0
         0
         0
         0
         0
         0
QL
         0
    0.1270
    0.1900
   -0.0390
    0.0160
    0.0750
         0
         0
    0.1660
    0.0580
    0.0180
    0.0160
    0.0580
    0.0500
T
         0
   -0.0864
   -0.2202
   -0.1819
   -0.1563
   -0.2561
   -0.2366
   -0.2366
   -0.2648
   -0.2682

```
   -0.2643
   -0.2708
   -0.2718
   -0.2854
V
    1.0600
    1.0450
    1.0100
    1.0295
    1.0349
    1.0700
    1.0559
    1.0900
    1.0497
    1.0458
    1.0543
    1.0547
    1.0495
    1.0315
bffr
   1.0000 + 0.0000i   2.0000 + 0.0000i   1.5595 - 0.1129i
   1.0000 + 0.0000i   5.0000 + 0.0000i   0.7642 + 0.0495i
   2.0000 + 0.0000i   3.0000 + 0.0000i   0.7335 + 0.1073i
   2.0000 + 0.0000i   4.0000 + 0.0000i   0.5665 - 0.0283i
   2.0000 + 0.0000i   5.0000 + 0.0000i   0.4212 - 0.0226i
   3.0000 + 0.0000i   4.0000 + 0.0000i  -0.2339 - 0.0050i
   4.0000 + 0.0000i   5.0000 + 0.0000i  -0.6231 + 0.0739i
   4.0000 + 0.0000i   7.0000 + 0.0000i   0.2841 - 0.1221i
   4.0000 + 0.0000i   9.0000 + 0.0000i   0.1609 - 0.0307i
   5.0000 + 0.0000i   6.0000 + 0.0000i   0.4376 - 0.1224i
   6.0000 + 0.0000i  11.0000 + 0.0000i   0.0711 + 0.0507i
   6.0000 + 0.0000i  12.0000 + 0.0000i   0.0780 + 0.0270i
   6.0000 + 0.0000i  13.0000 + 0.0000i   0.1765 + 0.0799i
   7.0000 + 0.0000i   8.0000 + 0.0000i   0.0000 - 0.2045i
   7.0000 + 0.0000i   9.0000 + 0.0000i   0.2841 + 0.0635i
   9.0000 + 0.0000i  10.0000 + 0.0000i   0.0548 + 0.0273i
   9.0000 + 0.0000i  14.0000 + 0.0000i   0.0952 + 0.0265i
  10.0000 + 0.0000i  11.0000 + 0.0000i  -0.0353 - 0.0310i
  12.0000 + 0.0000i  13.0000 + 0.0000i   0.0163 + 0.0095i
  13.0000 + 0.0000i  14.0000 + 0.0000i   0.0555 + 0.0271i
bffs
   1.0000 + 0.0000i   2.0000 + 0.0000i   1.5595 - 0.2315i
   1.0000 + 0.0000i   5.0000 + 0.0000i   0.7642 - 0.0611i
   2.0000 + 0.0000i   3.0000 + 0.0000i   0.7171 + 0.0131i
   2.0000 + 0.0000i   4.0000 + 0.0000i   0.5537 - 0.1015i
   2.0000 + 0.0000i   5.0000 + 0.0000i   0.4082 - 0.0971i
   3.0000 + 0.0000i   4.0000 + 0.0000i  -0.2451 - 0.0286i
   4.0000 + 0.0000i   5.0000 + 0.0000i  -0.6231 + 0.0739i
   4.0000 + 0.0000i   7.0000 + 0.0000i   0.2841 - 0.1221i
   4.0000 + 0.0000i   9.0000 + 0.0000i   0.1609 - 0.0307i
   5.0000 + 0.0000i   6.0000 + 0.0000i   0.4376 - 0.1224i
   6.0000 + 0.0000i  11.0000 + 0.0000i   0.0711 + 0.0507i
   6.0000 + 0.0000i  12.0000 + 0.0000i   0.0780 + 0.0270i
   6.0000 + 0.0000i  13.0000 + 0.0000i   0.1765 + 0.0799i
   7.0000 + 0.0000i   8.0000 + 0.0000i   0.0000 - 0.2045i
   7.0000 + 0.0000i   9.0000 + 0.0000i   0.2841 + 0.0635i
   9.0000 + 0.0000i  10.0000 + 0.0000i   0.0548 + 0.0273i
```

```
   9.0000 + 0.0000i  14.0000 + 0.0000i   0.0952 + 0.0265i
  10.0000 + 0.0000i  11.0000 + 0.0000i  -0.0353 - 0.0310i
  12.0000 + 0.0000i  13.0000 + 0.0000i   0.0163 + 0.0095i
  13.0000 + 0.0000i  14.0000 + 0.0000i   0.0555 + 0.0271i
bfrr
   2.0000 + 0.0000i   1.0000 + 0.0000i  -1.5071 + 0.3586i
   5.0000 + 0.0000i   1.0000 + 0.0000i  -0.7199 + 0.1719i
   3.0000 + 0.0000i   2.0000 + 0.0000i  -0.6835 + 0.0762i
   4.0000 + 0.0000i   2.0000 + 0.0000i  -0.5303 + 0.1499i
   5.0000 + 0.0000i   2.0000 + 0.0000i  -0.3941 + 0.1230i
   4.0000 + 0.0000i   3.0000 + 0.0000i   0.2481 + 0.0391i
   5.0000 + 0.0000i   4.0000 + 0.0000i   0.6281 - 0.0583i
   7.0000 + 0.0000i   4.0000 + 0.0000i  -0.2841 + 0.1410i
   9.0000 + 0.0000i   4.0000 + 0.0000i  -0.1609 + 0.0448i
   6.0000 + 0.0000i   5.0000 + 0.0000i  -0.4376 + 0.1709i
  11.0000 + 0.0000i   6.0000 + 0.0000i  -0.0705 - 0.0494i
  12.0000 + 0.0000i   6.0000 + 0.0000i  -0.0773 - 0.0255i
  13.0000 + 0.0000i   6.0000 + 0.0000i  -0.1744 - 0.0757i
   8.0000 + 0.0000i   7.0000 + 0.0000i  -0.0000 + 0.2112i
   9.0000 + 0.0000i   7.0000 + 0.0000i  -0.2841 - 0.0552i
  10.0000 + 0.0000i   9.0000 + 0.0000i  -0.0547 - 0.0270i
  14.0000 + 0.0000i   9.0000 + 0.0000i  -0.0941 - 0.0241i
  11.0000 + 0.0000i  10.0000 + 0.0000i   0.0355 + 0.0314i
  13.0000 + 0.0000i  12.0000 + 0.0000i  -0.0162 - 0.0095i
  14.0000 + 0.0000i  13.0000 + 0.0000i  -0.0549 - 0.0259i
bfrs
   2.0000 + 0.0000i   1.0000 + 0.0000i  -1.5270 + 0.2450i
   5.0000 + 0.0000i   1.0000 + 0.0000i  -0.7524 + 0.0716i
   3.0000 + 0.0000i   2.0000 + 0.0000i  -0.7216 - 0.0046i
   4.0000 + 0.0000i   2.0000 + 0.0000i  -0.5560 + 0.0825i
   5.0000 + 0.0000i   2.0000 + 0.0000i  -0.4169 + 0.0525i
   4.0000 + 0.0000i   3.0000 + 0.0000i   0.2385 + 0.0138i
   5.0000 + 0.0000i   4.0000 + 0.0000i   0.6281 - 0.0583i
   7.0000 + 0.0000i   4.0000 + 0.0000i  -0.2841 + 0.1410i
   9.0000 + 0.0000i   4.0000 + 0.0000i  -0.1609 + 0.0448i
   6.0000 + 0.0000i   5.0000 + 0.0000i  -0.4376 + 0.1709i
  11.0000 + 0.0000i   6.0000 + 0.0000i  -0.0705 - 0.0494i
  12.0000 + 0.0000i   6.0000 + 0.0000i  -0.0773 - 0.0255i
  13.0000 + 0.0000i   6.0000 + 0.0000i  -0.1744 - 0.0757i
   8.0000 + 0.0000i   7.0000 + 0.0000i  -0.0000 + 0.2112i
   9.0000 + 0.0000i   7.0000 + 0.0000i  -0.2841 - 0.0552i
  10.0000 + 0.0000i   9.0000 + 0.0000i  -0.0547 - 0.0270i
  14.0000 + 0.0000i   9.0000 + 0.0000i  -0.0941 - 0.0241i
  11.0000 + 0.0000i  10.0000 + 0.0000i   0.0355 + 0.0314i
  13.0000 + 0.0000i  12.0000 + 0.0000i  -0.0162 - 0.0095i
  14.0000 + 0.0000i  13.0000 + 0.0000i  -0.0549 - 0.0259i
iter1
Pmm
   -0.0606
    0.9219
    0.3823
   -0.0621
    0.6048
         0
         0
    0.2950
    0.0900
```

```
   -0.1019
   -0.0458
   -0.0819
    0.1490
Qmm
   -0.3433
   -0.7915
   -0.5109
   -0.0240
    0.0580
   -0.2686
   -0.2063
   -0.3692
    0.0500
T
         0
   -0.0816
   -0.2142
   -0.1792
   -0.1532
   -0.2530
   -0.2369
   -0.2369
   -0.2672
   -0.2701
   -0.2638
   -0.2688
   -0.2704
   -0.2876
V
    1.0600
    1.0450
    1.0100
    1.0367
    1.0430
    1.0700
    1.0630
    1.0900
    1.0565
    1.0518
    1.0592
    1.0584
    1.0531
    1.0366
jacobian
  Columns 1 through 7
   32.7276   -5.0470   -5.3461   -5.4277         0         0         0
   -5.0470   10.1665   -5.1195         0         0         0         0
   -5.3461   -5.1195   38.6240  -21.5786         0   -4.7819         0
   -5.4277         0  -21.5786   35.7410   -4.2457         0         0
         0         0         0   -4.2457   18.5546         0         0
         0         0   -4.7819         0         0   20.0599   -6.1879
         0         0         0         0         0   -6.1879    6.1879
         0         0   -1.7980         0         0   -9.0901         0
         0         0         0         0         0         0         0
         0         0         0         0   -4.3807         0         0
         0         0         0         0   -3.3983         0         0
```

```
       0         0         0         0   -6.5299         0         0
       0         0         0         0         0         0         0
  1.7619    2.0058  -10.6087    6.8410         0         0         0
  1.7777         0    6.8410   -9.7061         0         0         0
       0         0         0         0         0         0         0
       0         0         0         0         0         0         0
       0         0         0         0         0         0         0
       0         0         0         0    2.0919         0         0
       0         0         0         0    1.6328         0         0
       0         0         0         0    3.3159         0         0
       0         0         0         0         0         0         0
Columns 8 through 14
       0         0         0         0         0         0   -1.7619
       0         0         0         0         0         0   -2.0058
 -1.7980         0         0         0         0         0   10.4173
       0         0         0         0         0         0   -6.8410
       0         0   -4.3807   -3.3983   -6.5299         0         0
 -9.0901         0         0         0         0         0         0
       0         0         0         0         0         0         0
 24.2825  -10.3654         0         0         0   -3.0291         0
-10.3654   14.7683   -4.4029         0         0         0         0
       0   -4.4029    8.7836         0         0         0         0
       0         0         0    5.6503   -2.2520         0         0
       0         0         0   -2.2520   11.0969   -2.3150         0
 -3.0291         0         0         0   -2.3150    5.3440         0
       0         0         0         0         0         0   38.0154
       0         0         0         0         0         0  -21.5786
       0         0         0         0         0         0   -4.7819
 -5.3261    3.9020         0         0         0    1.4240   -1.7980
  3.9020   -5.7829    1.8809         0         0         0         0
       0    1.8809   -3.9728         0         0         0         0
       0         0         0   -4.1218    2.4890         0         0
       0         0         0    2.4890   -6.9419    1.1370         0
  1.4240         0         0         0    1.1370   -2.5610         0
Columns 15 through 21
 -1.7777         0         0         0         0         0         0
       0         0         0         0         0         0         0
 -6.8410         0         0         0         0         0         0
  9.4299         0         0         0         0         0         0
       0         0         0         0   -2.0919   -1.6328   -3.3159
       0         0         0         0         0         0         0
       0         0         0         0         0         0         0
       0         0    5.3261   -3.9020         0         0         0
       0         0   -3.9020    5.7829   -1.8809         0         0
       0         0         0   -1.8809    3.6991         0         0
       0         0         0         0         0    3.9082   -2.4890
       0         0         0         0         0   -2.4890    6.5080
       0         0   -1.4240         0         0         0   -1.1370
-21.5786   -4.7819   -1.7980         0         0         0         0
 34.1260         0         0         0         0         0         0
       0   19.0381   -9.0901         0         0         0         0
       0   -9.0901   23.9025  -10.3654         0         0         0
       0         0  -10.3654   14.7683   -4.4029         0         0
       0         0         0   -4.4029    8.2104         0         0
       0         0         0         0         0    5.2056   -2.2520
       0         0         0         0         0   -2.2520   10.2425
       0         0   -3.0291         0         0         0   -2.3150
```

```
  Column 22
         0
         0
         0
         0
         0
         0
         0
   -1.4240
         0
         0
         0
   -1.1370
    2.5610
         0
         0
         0
   -3.0291
         0
         0
         0
   -2.3150
    5.3440
iter2
Pmm
    0.0766
    0.0072
    0.0014
    0.0473
   -0.0047
    0.0059
         0
   -0.0365
   -0.0081
    0.0082
    0.0035
    0.0070
   -0.0108
Qmm
    0.0745
    0.1272
    0.0477
    0.0133
   -0.0015
    0.0166
    0.0130
    0.0217
    0.0020
T
         0
   -0.0864
   -0.2201
   -0.1818
   -0.1563
   -0.2560
   -0.2365
   -0.2365
```

```
   -0.2648
   -0.2682
   -0.2643
   -0.2707
   -0.2718
   -0.2854
V
    1.0600
    1.0450
    1.0100
    1.0296
    1.0350
    1.0700
    1.0559
    1.0900
    1.0497
    1.0459
    1.0543
    1.0547
    1.0495
    1.0316
jacobian
  Columns 1 through 7
   33.0336   -5.1611   -5.6938   -5.7791         0         0         0
   -4.8443   10.0756   -5.2313         0         0         0         0
   -5.3376   -5.3767   41.0680  -23.1310         0   -5.2611         0
   -5.5136         0  -23.5155   37.8892   -4.4061         0         0
         0         0         0   -4.4061   19.6293         0         0
         0         0   -5.2611         0         0   22.0441   -6.5780
         0         0         0         0         0   -6.5780    6.5780
         0         0   -1.9617         0         0  -10.2049         0
         0         0         0         0         0         0         0
         0         0         0         0   -4.6158         0         0
         0         0         0         0   -3.5689         0         0
         0         0         0         0   -6.8150         0         0
         0         0         0         0         0         0         0
    2.3582    1.8926  -11.7747    8.0005         0   -0.3036         0
    2.2547         0    6.7877  -10.4368   -0.4412         0         0
         0         0    0.3036         0         0    0.0059         0
         0         0    0.1730         0         0    0.3095         0
         0         0         0         0         0         0         0
         0         0         0         0    2.2657         0         0
         0         0         0         0    1.7847         0         0
         0         0         0         0    3.6105         0         0
         0         0         0         0         0         0         0
  Columns 8 through 14
         0         0         0         0         0         0   -1.2322
         0         0         0         0         0         0   -2.1836
   -1.9617         0         0         0         0         0   10.4387
         0         0         0         0         0         0   -6.5476
         0         0   -4.6637   -3.6235   -6.9360         0         0
  -10.2049         0         0         0         0         0   -0.2929
         0         0         0         0         0         0         0
   27.0471  -11.5319         0         0         0   -3.3486   -0.1669
  -11.5067   16.3988   -4.8921         0         0         0         0
         0   -4.9185    9.5343         0         0         0         0
         0         0         0    6.0831   -2.5142         0         0
```

```
       0         0         0   -2.5057   11.8688   -2.5481         0
 -3.2849         0         0         0   -2.5052    5.7901         0
 -0.1730         0         0         0         0         0   39.8344
       0         0         0         0         0         0  -22.6837
 -0.3095         0         0         0         0         0   -5.0750
 -6.2770    4.3029         0         0         0    1.4915   -1.8923
  4.3699   -6.4962    2.1263         0         0         0         0
       0    2.0647   -4.3304         0         0         0         0
       0         0         0   -4.5550    2.7703         0         0
       0         0         0    2.7780   -7.5858    1.1974         0
  1.6270         0         0         0    1.2846   -2.9115         0
Columns 15 through 21
 -1.3844         0         0         0         0         0         0
       0         0         0         0         0         0         0
 -7.6709    0.2856    0.1638         0         0         0         0
  9.9517         0         0         0         0         0         0
 -0.4230         0         0         0   -2.0445   -1.5789   -3.2022
       0    0.0056    0.2930         0         0         0         0
       0         0         0         0         0         0         0
       0   -0.2912    5.3135   -4.0908         0         0         0
       0         0   -4.1361    5.9895   -2.0074         0         0
       0         0         0   -1.9629    4.0377         0         0
       0         0         0         0         0    4.1950   -2.6307
       0         0         0         0         0   -2.6248    6.9604
       0         0   -1.5399         0         0         0   -1.2198
-22.1779   -4.9491   -1.8567         0         0         0         0
 36.5414         0         0         0         0         0         0
       0   20.8264   -9.6588         0         0         0         0
       0   -9.5997   25.3104  -10.9635         0         0         0
       0         0  -10.8908   15.4775   -4.6187         0         0
       0         0         0   -4.6760    8.9988         0         0
       0         0         0         0         0    5.7420   -2.3875
       0         0         0         0         0   -2.3675   11.2016
       0         0   -3.1091         0         0         0   -2.3789
Column 22
       0
       0
       0
       0
       0
       0
       0
 -1.4389
       0
       0
       0
 -1.1551
  2.5006
       0
       0
       0
 -3.2304
       0
       0
       0
 -2.4581
  5.4932
```

```
iter3
Pmm
   1.0e-03 *
    0.2533
    0.1333
    0.1348
    0.4473
    0.1127
    0.0722
         0
   -0.4647
   -0.0526
    0.1178
    0.0182
    0.0779
   -0.0657
Qmm
   1.0e-03 *
    0.6361
    0.9833
    0.3717
    0.1644
    0.0182
    0.0768
    0.0429
    0.0830
    0.0408
T
         0
   -0.0864
   -0.2202
   -0.1819
   -0.1563
   -0.2561
   -0.2366
   -0.2366
   -0.2648
   -0.2682
   -0.2643
   -0.2708
   -0.2718
   -0.2854
V
    1.0600
    1.0450
    1.0100
    1.0295
    1.0349
    1.0700
    1.0559
    1.0900
    1.0497
    1.0458
    1.0543
    1.0547
    1.0495
    1.0315
```

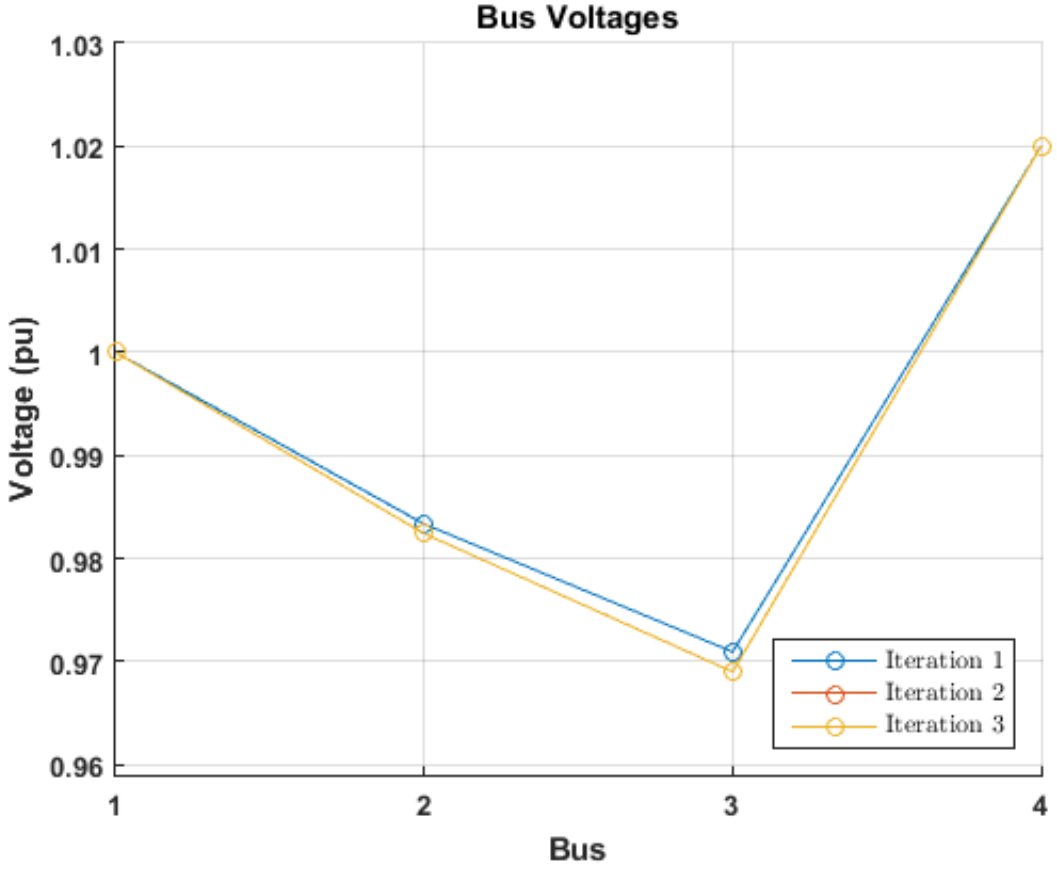
```
jacobian
  Columns 1 through 7
   32.9117   -5.1617   -5.6519   -5.7321         0         0         0
   -4.8422   10.0301   -5.1879         0         0         0         0
   -5.3061   -5.3460   40.5784  -22.7989         0   -5.1909         0
   -5.4752         0  -23.1716   37.4332   -4.3723         0         0
         0         0         0   -4.3723   19.5255         0         0
         0         0   -5.1909         0         0   21.7968   -6.5340
         0         0         0         0         0   -6.5340    6.5340
         0         0   -1.9365         0         0  -10.0719         0
         0         0         0         0         0         0         0
         0         0         0         0   -4.6002         0         0
         0         0         0         0   -3.5584         0         0
         0         0         0         0   -6.7975         0         0
         0         0         0         0         0         0         0
    2.3302    1.8618  -11.6214    7.8746         0   -0.2843         0
    2.2276         0    6.6993  -10.3241   -0.4375         0         0
         0         0    0.2843         0         0    0.0001         0
         0         0    0.1610         0         0    0.2843         0
         0         0         0         0         0         0         0
         0         0         0         0    2.2438         0         0
         0         0         0         0    1.7748         0         0
         0         0         0         0    3.5877         0         0
         0         0         0         0         0         0         0
  Columns 8 through 14
         0         0         0         0         0         0   -1.2444
         0         0         0         0         0         0   -2.2004
   -1.9365         0         0         0         0         0   10.3595
         0         0         0         0         0         0   -6.5070
         0         0   -4.6368   -3.6092   -6.9073         0         0
  -10.0719         0         0         0         0         0   -0.2761
         0         0         0         0         0         0         0
   26.7142  -11.3946         0         0         0   -3.3113   -0.1563
  -11.3653   16.2124   -4.8470         0         0         0         0
         0   -4.8631    9.4634         0         0         0         0
         0         0         0    6.0541   -2.4956         0         0
         0         0         0   -2.4899   11.8102   -2.5228         0
   -3.2476         0         0         0   -2.4893    5.7369         0
   -0.1610         0         0         0         0         0   39.4907
         0         0         0         0         0         0  -22.5064
   -0.2843         0         0         0         0         0   -5.0419
   -6.1644    4.2451         0         0         0    1.4740   -1.8809
    4.3229   -6.4157    2.0929         0         0         0         0
         0    2.0552   -4.2990         0         0         0         0
         0         0         0   -4.5273    2.7526         0         0
         0         0         0    2.7578   -7.5422    1.1967         0
    1.6094         0         0         0    1.2650   -2.8744         0
  Columns 15 through 21
   -1.3944         0         0         0         0         0         0
         0         0         0         0         0         0         0
   -7.6087    0.2692    0.1533         0         0         0         0
    9.8294         0         0         0         0         0         0
   -0.4228         0         0         0   -2.0555   -1.5825   -3.2124
         0    0.0001    0.2709         0         0         0         0
         0         0         0         0         0         0         0
         0   -0.2693    5.3095   -4.0589         0         0         0
         0         0   -4.1181    5.9621   -1.9850         0         0
```

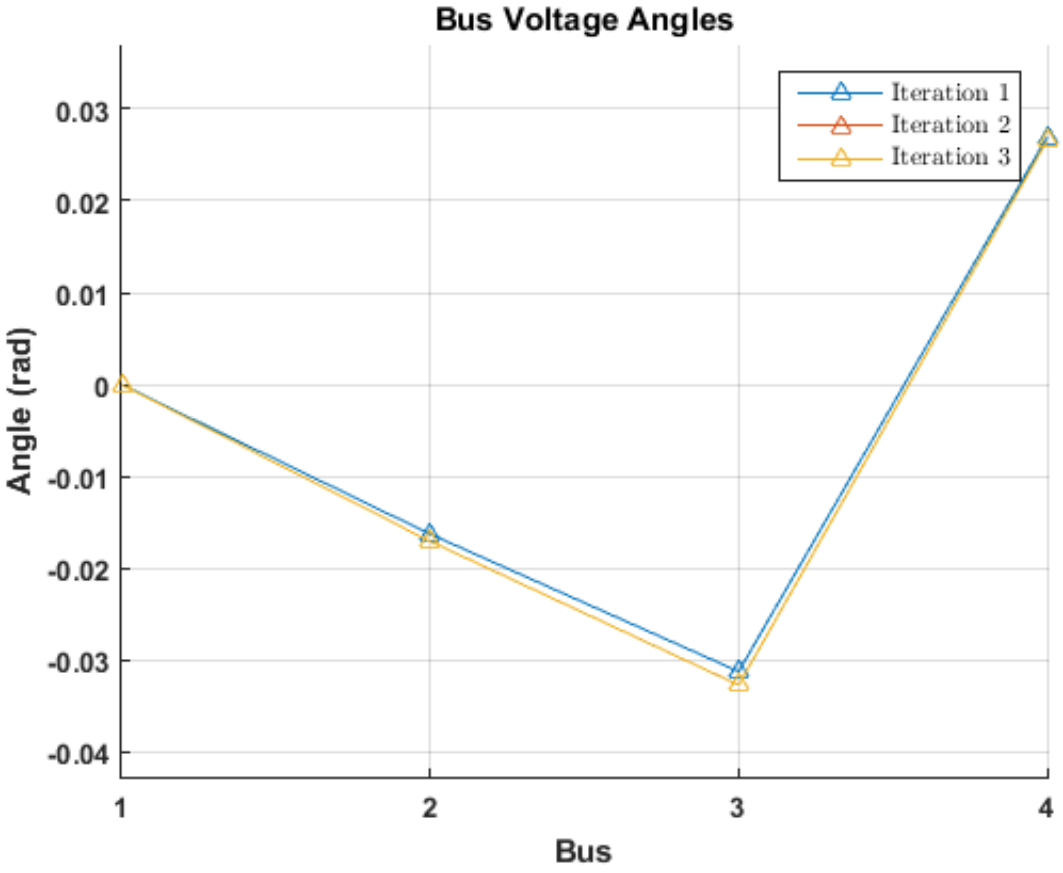
```
         0         0         0   -1.9651    4.0112         0         0
         0         0         0         0         0    4.1768   -2.6227
         0         0         0         0         0   -2.6147    6.9293
         0         0   -1.5332         0         0         0   -1.2053
  -22.0290   -4.9159   -1.8448         0         0         0         0
   36.1400         0         0         0         0         0         0
         0   20.6428   -9.5947         0         0         0         0
         0   -9.5383   25.1327  -10.8948         0         0         0
         0         0  -10.8269   15.3904   -4.5973         0         0
         0         0         0   -4.6498    8.9417         0         0
         0         0         0         0         0    5.7098   -2.3779
         0         0         0         0         0   -2.3607   11.1427
         0         0   -3.0937         0         0         0   -2.3719
  Column 22
         0
         0
         0
         0
         0
         0
         0
   -1.4289
         0
         0
         0
   -1.1601
    2.4974
         0
         0
         0
   -3.2099
         0
         0
         0
   -2.4456
    5.4644
ybus
  Columns 1 through 4
   6.0250 -19.4471i  -4.9991 +15.2631i   0.0000 + 0.0000i   0.0000 + 0.0000i
  -4.9991 +15.2631i   9.5213 -30.2721i  -1.1350 + 4.7819i  -1.6860 + 5.1158i
   0.0000 + 0.0000i  -1.1350 + 4.7819i   3.1210 - 9.8224i  -1.9860 + 5.0688i
   0.0000 + 0.0000i  -1.6860 + 5.1158i  -1.9860 + 5.0688i  10.5130 -38.3197i
  -1.0259 + 4.2350i  -1.7011 + 5.1939i   0.0000 + 0.0000i  -6.8410 +21.5786i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 4.7819i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 1.7980i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
  Columns 5 through 8
  -1.0259 + 4.2350i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
  -1.7011 + 5.1939i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
  -6.8410 +21.5786i   0.0000 + 0.0000i   0.0000 + 4.7819i   0.0000 + 0.0000i
```

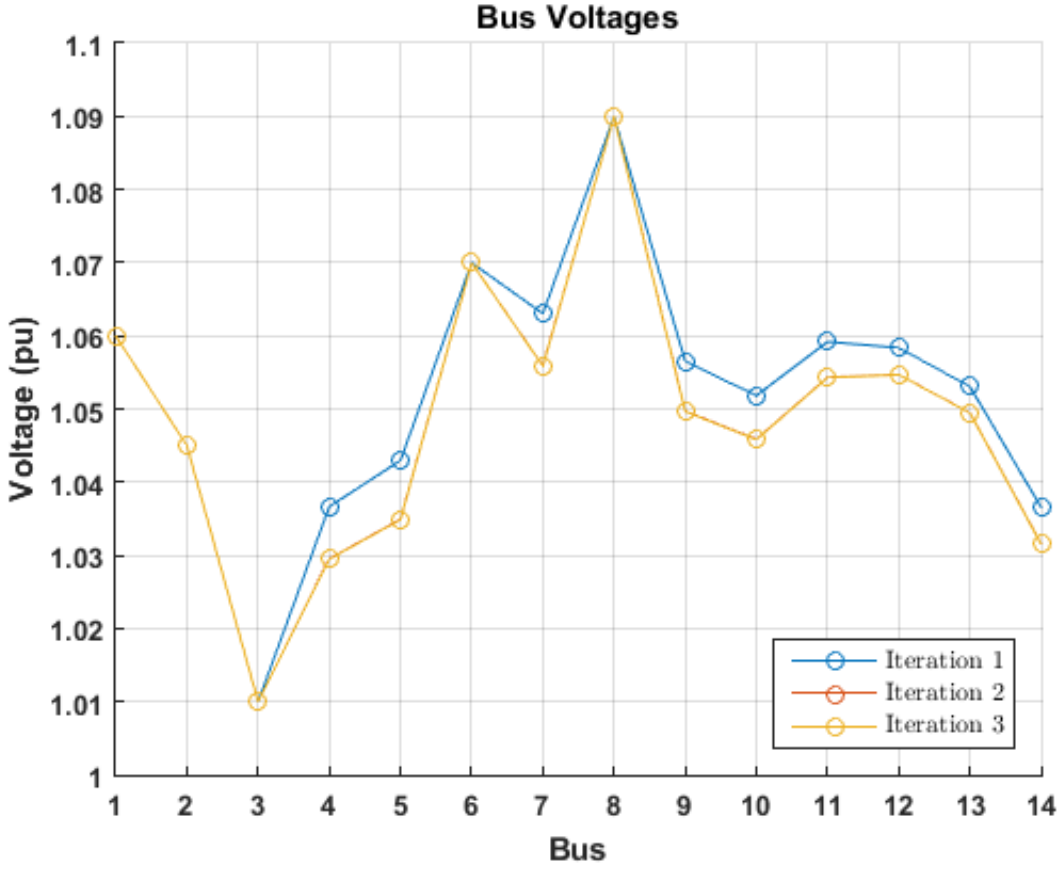
```
 9.5680 -34.9335i   0.0000 + 3.9679i   0.0000 + 0.0000i   0.0000 + 0.0000i
 0.0000 + 3.9679i   6.5799 -17.3407i   0.0000 + 0.0000i   0.0000 + 0.0000i
 0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 -19.5490i   0.0000 + 5.6770i
 0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 5.6770i   0.0000 - 5.6157i
 0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 9.0901i   0.0000 + 0.0000i
 0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
 0.0000 + 0.0000i  -1.9550 + 4.0941i   0.0000 + 0.0000i   0.0000 + 0.0000i
 0.0000 + 0.0000i  -1.5260 + 3.1760i   0.0000 + 0.0000i   0.0000 + 0.0000i
 0.0000 + 0.0000i  -3.0989 + 6.1028i   0.0000 + 0.0000i   0.0000 + 0.0000i
 0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
Columns 9 through 12
 0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
 0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
 0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
 0.0000 + 1.7980i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
 0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
 0.0000 + 0.0000i   0.0000 + 0.0000i  -1.9550 + 4.0941i  -1.5260 + 3.1760i
 0.0000 + 9.0901i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
 0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
 5.3261 -24.0925i  -3.9020 +10.3654i   0.0000 + 0.0000i   0.0000 + 0.0000i
-3.9020 +10.3654i   5.7829 -14.7683i  -1.8809 + 4.4029i   0.0000 + 0.0000i
 0.0000 + 0.0000i  -1.8809 + 4.4029i   3.8359 - 8.4970i   0.0000 + 0.0000i
 0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   4.0150 - 5.4279i
 0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i  -2.4890 + 2.2520i
-1.4240 + 3.0291i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
Columns 13 through 14
 0.0000 + 0.0000i   0.0000 + 0.0000i
 0.0000 + 0.0000i   0.0000 + 0.0000i
 0.0000 + 0.0000i   0.0000 + 0.0000i
 0.0000 + 0.0000i   0.0000 + 0.0000i
 0.0000 + 0.0000i   0.0000 + 0.0000i
-3.0989 + 6.1028i   0.0000 + 0.0000i
 0.0000 + 0.0000i   0.0000 + 0.0000i
 0.0000 + 0.0000i   0.0000 + 0.0000i
 0.0000 + 0.0000i  -1.4240 + 3.0291i
 0.0000 + 0.0000i   0.0000 + 0.0000i
 0.0000 + 0.0000i   0.0000 + 0.0000i
-2.4890 + 2.2520i   0.0000 + 0.0000i
 6.7249 -10.6697i  -1.1370 + 2.3150i
-1.1370 + 2.3150i   2.5610 - 5.3440i
```

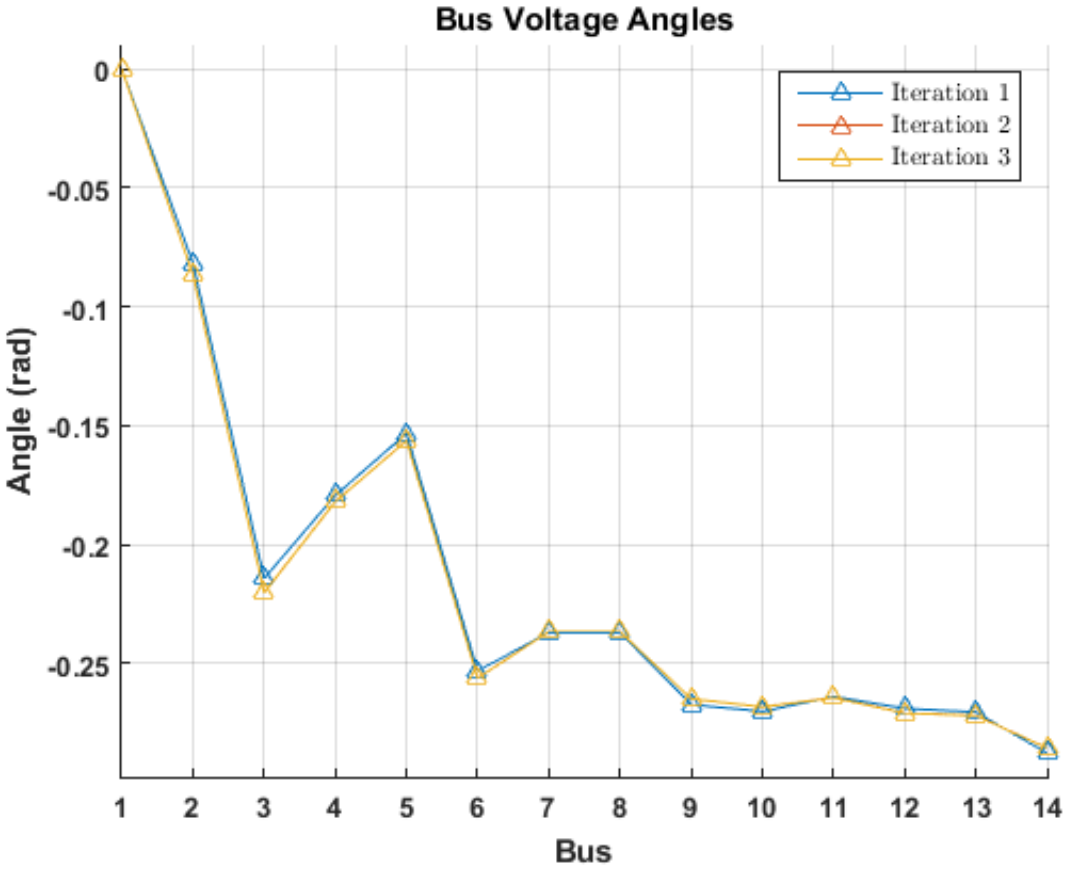
Generate Voltage Plots

```
plot_voltages(amap,'4busvolts')
plot_voltages(bmap,'14busvolts');
```


Bus Voltages
1.03
1.02
1.01
1
0.99
0.98
0.97
0.96
Voltage (pu)
1
2
3
4
Bus
Iteration 1
Iteration 2
Iteration 3


Bus Voltage Angles
0.03
0.02
0.01
0
-0.01
-0.02
-0.03
-0.04
Angle (rad)
1
2
3
4
Bus
Iteration 1
Iteration 2
Iteration 3


Bus Voltages
Voltage (pu)
1.1
1.09
1.08
1.07
1.06
1.05
1.04
1.03
1.02
1.01
1
1 2 3 4 5 6 7 8 9 10 11 12 13 14
Bus
Iteration 1
Iteration 2
Iteration 3


Bus Voltage Angles
Angle (rad)
0
-0.05
-0.1
-0.15
-0.2
-0.25
1 2 3 4 5 6 7 8 9 10 11 12 13 14
Bus
Iteration 1
Iteration 2
Iteration 3

Published with MATLAB® R2015a

EE 556 Project

Runs entire project Dumps all data to the screen Creates all images for the report

Contents

Run NRPF

```
doTestA=1;
doTestB=1;
printIterations=0;
[abranch,abus,amap,bbranch,bbus,bmap]=master(printIterations,doTestA,doTestB);
```

Print all Test System A Data

```
fprintf('----------------------------------------\n');
fprintf('TEST SYSTEM A\n');
print_nrpf(amap);
```

Print all Test System B Data

```
fprintf('----------------------------------------\n');
fprintf('TEST SYSTEM B\n');
print_nrpf(bmap);
```

Generate Voltage Plots

```
plot_voltages(amap,'4busvolts')
plot_voltages(bmap,'14busvolts');
```

Published with MATLAB® R2015a

Contents

EE 556, Project Master File

Runs all problems

```
function [abranch,abus,amap,bbranch,bbus,bmap]=master(printIters,doA,doB)
```

```
    if(nargin<1)
        doA=1;
        doB=1;
        printIters=0;
    end
```

Test System A

4-bus

```
    if(doA==1)
        fprintf('------------------------------------------------------\n');
        disp('Test System A: 4 Bus');
        [abus,abranch]=tsa(); % load data
        [amap,err]=nrpf(abus,abranch,printIters);
        if(isempty(err)==0)
            disp(err);
            return;
        end
    else
        abranch=0;abus=0;amap=0;
    end
```

Test System B

14-bus

```
    if(doB==1)
        fprintf('------------------------------------------------------\n');
        disp('Test System B: 14 Bus');
        [bbus,bbranch]=tsb;
        [bmap,err]=nrpf(bbus,bbranch,printIters);
        if(isempty(err)==0)
            disp(err);
            return;
        end
    else
```

```
        bbranch=0;bbus=0;bmap=0;
    end
```

```
end
```

Published with MATLAB® R2015a

Contents

tsa: Test System A

Provides bus and branch data for Test System A

```
function [busdata,branchdata]=tsa()
```

Bus Data 1 2 3 4 5 6 7 8 9 10

```
          Bus    Type     PG    QG   PL    QL      V     Theta   G   B
```

```
    busdata=[     1       1       0     0    50    30.99   1     0       0   0.05;
                  2       2       0     0    170   105.35  1     0       0   0;
                  3       2       0     0    200   123.94  1     0       0   0;
                  4       3       318   0    80    49.58   1.02  0       0   0.05;
    ];
    basemva=100;
    basekv=230;
    busdata(:,3:6)=busdata(:,3:6)/basemva;
```

Branch Data

```
             From   To   R(pu)     X(pu)     G(pu)   B(pu)    %% B/2 (given)
```

```
    branchdata=[      1      2    0.01008   0.05040   0       0.1025; %  0.05125
                      1      3    0.00744   0.03720   0       0.0775; %  0.03875
                      2      4    0.00744   0.03720   0       0.0775; %  0.03875
                      3      4    0.01272   0.06360   0       0.1275; %  0.06375
    ];
end
```

Published with MATLAB® R2015a

Contents

tsa: Test System B

Provides bus and branch data for Test System B

```
function [busdata,branchdata]=tsb()
```

Bus Data

Bus Data 1 2 3 4 5 6 7 8 9 10

```
          Bus    Type    PG    QG  PL     QL     V       Theta  G    B
```

```
    busdata=[     1      1       0     0   0      0      1.06     0      0    0;
                  2      3       40    0   21.7   12.7   1.045    0      0    0;
                  3      3       0     0   94.2   19     1.01     0      0    0;
                  4      2       0     0   47.8   -3.9   1.0      0      0    0;
                  5      2       0     0   7.6    1.6    1.0      0      0    0;
                  6      3       0     0   11.2   7.5    1.07     0      0    0;
                  7      2       0     0   0      0      1.0      0      0    0;
                  8      3       0     0   0      0      1.09     0      0    0.06125;
                  9      2       0     0   29.5   16.6   1.0      0      0    0.19;
                  10     2       0     0   9      5.8    1.0      0      0    0;
                  11     2       0     0   3.5    1.8    1.0      0      0    0;
                  12     2       0     0   6.1    1.6    1.0      0      0    0;
                  13     2       0     0   13.5   5.8    1.0      0      0    0;
                  14     2       0     0   14.9   5      1.0      0      0    0;
    ];
    basemva=100;
    basekv=230;
    busdata(:,3:6)=busdata(:,3:6)/basemva;
```

Branch Data

```
          From    To      R(pu)    X(pu)    G(pu)    B(pu)
```

```
    branchdata=[    1       2       0.01938 0.05917 0       0.05280 ;
                    1       5       0.05403 0.22304 0       0.04920 ;
                    2       3       0.04699 0.19797 0       0.04380 ;
                    2       4       0.05811 0.17632 0       0.03400 ;
                    2       5       0.05695 0.17388 0       0.03460 ;
                    3       4       0.06701 0.17103 0       0.01280 ;
                    4       5       0.01335 0.04211 0       0       ;
                    4       7       0       0.20912 0       0       ;
```

```
                    4       9       0       0.55618 0       0       ;
                    5       6       0       0.25202 0       0       ;
                    6       11      0.09498 0.19890 0       0       ;
                    6       12      0.12291 0.25581 0       0       ;
                    6       13      0.06615 0.13027 0       0       ;
                    7       8       0       0.17615 0       0       ;
                    7       9       0       0.11001 0       0       ;
                    9       10      0.03181 0.08450 0       0       ;
                    9       14      0.12711 0.27038 0       0       ;
                    10      11      0.08205 0.19207 0       0       ;
                    12      13      0.22092 0.19988 0       0       ;
                    13      14      0.17093 0.34802 0       0       ;
    ];
end
```

Published with MATLAB® R2015a

nrpf

Newton-Raphson Power Flow performs a classic power flow. Required inputs provide bus and branch data about the circuit.

Contents

USAGE

- **[results]=nrpf(busdata,branchdata,PRINT_ITERS,THRESH,ITER_MAX,FREEZE_JAC)**
- **[results]=nrpf(busdata,branchdata,PRINT_ITERS,THRESH,ITER_MAX)**
- **[results]=nrpf(busdata,branchdata,PRINT_ITERS,THRESH)**
- **[results]=nrpf(busdata,branchdata,PRINT_ITERS)**
- **[results]=nrpf(busdata,branchdata)**

INPUTS

- **busdata**: bus data matrix in form: [Bus Number,Type,P,Q,V,Theta]
- **branchdata**: branch data matrix in form: [From Bus,To Bus,R(pu),X(pu),G(pu),B(pu)]
- **PRINT_ITERS**: 1 to print each iteration, defaults to 0
- **THRESH**: indicate mismatch threshold, defaults to 0.001
- **ITER_MAX**: indicate maximum number of iterations, defaults to 10
- **FREEZE_JAC**: a silly option for the project which requires us to freeze the jacobian after the first iteration

OUTPUTS

- **results**: a map conaintaing keys-value pairs: 'ybus' ybus of the system 'P' final real power for each bus 'Q' final reactive power for each bus 'V' final voltage for each bus 'T' final theta for each bus 'itermap' map of all maps returned from nrpf_jac
- **err**: empty string if all clear or an error string if there was a problem
- **Prints final result**

```
function [results,err]=nrpf(busdata,branchdata,PRINT_ITERS,THRESH,ITER_MAX,FREEZE_JAC)
```

```
    if(nargin<6)
```

```
    FREEZE_JAC=0;  % only calculate the first jacobian
end
if(nargin<5)
    ITER_MAX=10;   % maximum number of iterations
end
if(nargin<4)
    THRESH=0.001;  % power mismatch target
end
if(nargin<3)
    PRINT_ITERS=0; % debug tool: 1=print iterations, 0=do not print
end

% Maps of data to return
results=containers.Map;

% Input Preparation
BusNums=busdata(:,1);
BusTypes=busdata(:,2);
buscount=length(BusTypes);
PG=busdata(:,3);
QG=busdata(:,4);
PL=busdata(:,5);
QL=busdata(:,6);
V=busdata(:,7);
T=busdata(:,8);
BusG=busdata(:,9);
BusB=busdata(:,10);

% Calculate P injections
P=PG-PL;
Q=QG-QL;

% Admittance matrix
[ybus_matrix,err]=ybus(BusNums,BusG,BusB,branchdata);
if(isempty(err)==0)
    disp(err);
    return;
end
results('ybus')=ybus_matrix;

[Pmm,Qmm,err]=mismatch(P,Q,V,T,BusTypes,ybus_matrix);
if(isempty(err)==0)
    disp(err);
    return;
end
```

Newton-Raphson Iterations

```
iter=1;
if(FREEZE_JAC==1)
    [jfull,err]=nrpf_jac(BusTypes,V,T,ybus_matrix);
    if(isempty(err)==0)
        disp(err);
        return;
    end
end
```

```
while (max(abs(Qmm)) > THRESH || max(abs(Pmm)) > THRESH) && iter < ITER_MAX
    if(FREEZE_JAC==0)
        [jfull,err]=nrpf_jac(BusTypes,V,T,ybus_matrix);
        if(isempty(err)==0)
            disp(err);
            return;
        end
    end
    itermap=containers.Map;
    itermap('jacobian')=jfull;

    [Pmm,Qmm,err]=mismatch(P,Q,V,T,BusTypes,ybus_matrix);
    if(isempty(err)==0)
        disp(err);
        return;
    end
    itermap('Pmm')=Pmm;
    itermap('Qmm')=Qmm;

    % Invert Jacobian
    deltas=-1*jfull^-1*[Pmm;Qmm];

    % Update State Variables
    deltas_index=1;
    for n=1:buscount
        if BusTypes(n)==1 % slack
            continue;
        end
        T(n)=T(n)+deltas(deltas_index);
        deltas_index=deltas_index+1;
    end
    for n=1:buscount
        if BusTypes(n)==1 % slack
            continue;
        elseif BusTypes(n)==3 % PV
            continue;
        end
        V(n)=V(n)+deltas(deltas_index);
        deltas_index=deltas_index+1;
    end
    itermap('V')=V;
    itermap('T')=T;

    % Print Iteration
    if(PRINT_ITERS==1)
        pmm_index=1;
        qmm_index=1;
        Pmmp=zeros(buscount,1);
        Qmmp=zeros(buscount,1);
        for n=1:buscount
            if BusTypes(n)==1 % slack
                Pmmp(n,1)=0;
                Qmmp(n,1)=0;
            else
                Pmmp(n,1)=Pmm(pmm_index,1);
                pmm_index=pmm_index+1;
                if BusTypes(n)==3 % PV
```

```
                    Qmmp(n,1)=qfunc(n,V,T,ybus_matrix);
                else
                    Qmmp(n,1)=Qmm(qmm_index,1);
                    qmm_index=qmm_index+1;
                end
            end
        end
        iterstring=sprintf('Iter %d',iter);
        pmatrix=[V,T*180/pi,Pmmp,Qmmp];
        buslabels=sprintf('Bus_%d ', 1:buscount);
        printmat(pmatrix,iterstring,buslabels,'V Th(deg) MM_P MM_Q');
        results('pmatrix')=pmatrix;
    end
    iterkey=sprintf('iter%d',iter);
    results(iterkey)=itermap;
    iter=iter+1;
end
```

Calculate Final Results

Bus Power

```
for n=1:buscount
    Pinj=nearzero(pfunc(n,V,T,ybus_matrix));
    Qinj=nearzero(qfunc(n,V,T,ybus_matrix));
    PG(n)=nearzero(Pinj+PL(n));
    QG(n)=nearzero(Qinj+QL(n));
    PL(n)=nearzero(PL(n));
    QL(n)=nearzero(QL(n));
end

% Branch Flow Forward, Sending and Receiving
[From,To,R,X,~,B,err]=parse_branch_data(branchdata);
branchcount=length(From);
Ss=zeros(buscount,3);
Sr=zeros(buscount,3);
for n=1:branchcount
    Ss(n,1)=From(n);
    Ss(n,2)=To(n);
    Sr(n,1)=From(n);
    Sr(n,2)=To(n);
    Z=(R(n)+1i*X(n));
    vfrom=V(From(n))*(cos(T(From(n)))+1i*sin(T(From(n))));
    vto=V(To(n))*(cos(T(To(n)))+1i*sin(T(To(n))));
    I=(vfrom-vto)/Z;
    Ss(n,3)=vfrom*conj(I)-1i*((vfrom)^2)*B(n);
    Sr(n,3)=vfrom*conj(I)+1i*((vfrom)^2)*B(n);
end
results('bffs')=Ss;
results('bffr')=Sr;
% Branch Flow Reverse, Sending and Receiving
Ss=zeros(buscount,3);
Sr=zeros(buscount,3);
for n=1:branchcount
    Ss(n,1)=To(n);
```

```
        Ss(n,2)=From(n);
        Sr(n,1)=To(n);
        Sr(n,2)=From(n);
        Z=(R(n)+1i*X(n));
        vfrom=V(To(n))*(cos(T(To(n)))+1i*sin(T(To(n))));
        vto=V(From(n))*(cos(T(From(n)))+1i*sin(T(From(n))));
        I=(vfrom-vto)/Z;
        Ss(n,3)=vfrom*conj(I)-1i*((vfrom)^2)*B(n);
        Sr(n,3)=vfrom*conj(I)+1i*((vfrom)^2)*B(n);
    end
    results('bfrs')=Ss;
    results('bfrr')=Sr;
```

Note Success or Failure

```
    if iter<ITER_MAX
        fprintf(', mismatch target of S=%f met\n',THRESH);
    else
        fprintf(', mismatch target of S=%f likely not met as max iterations performed\n',THRESH);
    end
```

Print Final Results

```
    fprintf('Final Results: ');
    fprintf('%d iterations',iter-1);
    pmatrix=[PG,QG,PL,QL,V,T*180/pi];
    buslabels=sprintf('Bus_%d ', 1:buscount);
    printmat(pmatrix,'name',buslabels,'PG QG PL QL V Th(deg)');
```

Gather Results

```
    results('PG')=PG;
    results('QG')=QG;
    results('PL')=PL;
    results('QL')=QL;
    results('P')=PG-PL;
    results('Q')=QG-QL;
    results('V')=V;
    results('T')=T;
```

```
end
```

Published with MATLAB® R2015a

nrpfjacobian Newton-Raphson Power Flow Jacobian

Calculates the Jacobian for a provided power system

Contents

USAGE

- **[jfull,err]=nrpf_jac(BusTypes,V,T,ybus_matrix)**

INPUTS

- **BusTypes**: column vector of bustypes, 1=slack, 2=PQ, 3=PV
- **V**: column vector of voltage for each bus
- **T**: column vector of theta for each voltage of each bus
- **ybus_matrix**: admittance matrix for the system

OUTPUTS

- **jfull**: the full jacobian matrix
- **err**: empty string if no error, error string otherwise

```
function [jfull,err]=nrpf_jac(BusTypes,V,T,ybus_matrix)
    [pcount,qcount,err]=jacobianCount(BusTypes); % P and Q equation counts
    if(isempty(err)==0)
        disp(err);
        return;
    end

    % Jacobian Submatrixes
    j11=zeros(pcount,pcount);
    j12=zeros(pcount,qcount);
    j21=zeros(qcount,pcount);
    j22=zeros(qcount,qcount);
    % Jacobian Submatrix Indexes
    j11_i=1; j12_i=1; j21_i=1; j22_i=1;
    j11_j=1; j12_j=1; j21_j=1; j22_j=1;

    % Iteration
    buscount=length(BusTypes);
    for n=1:buscount
        if(BusTypes(n)==1) % Slack Bus
            continue;
        elseif(BusTypes(n)==2) % PQ Bus
            for m=1:buscount
                if(BusTypes(m)==1) % Slack
                    continue;
```

```
                end
                % Partial of P with respect to Theta
                j11(j11_i,j11_j)=jptheta(n,m,V,T,ybus_matrix);
                j11_j=j11_j+1;
                % Partial of Q with respect to Theta
                j21(j21_i,j21_j)=jqtheta(n,m,V,T,ybus_matrix);
                j21_j=j21_j+1;
                % Only do partials with respect to V for PQ buses
                if(BusTypes(m)==2)
                    % Partial of P with respect to V
                    j12(j12_i,j12_j)=jpv(n,m,V,T,ybus_matrix);
                    j12_j=j12_j+1;
                    % Partial of Q with respect to V
                    j22(j22_i,j22_j)=jqv(n,m,V,T,ybus_matrix);
                    j22_j=j22_j+1;
                end
            end
            j21_i=j21_i+1; j21_j=1;
            j22_i=j22_i+1; j22_j=1;
        elseif(BusTypes(n)==3) % PV Bus
            for m=1:buscount
                if(BusTypes(m)==1) % Slack
                    continue;
                end
                % Partial of P with respect to Theta
                j11(j11_i,j11_j)=jptheta(n,m,V,T,ybus_matrix);
                j11_j=j11_j+1;
                if(BusTypes(m)==2) % PQ
                    % Partial of P with respect to V
                    j12(j12_i,j12_j)=jpv(n,m,V,T,ybus_matrix);
                    j12_j=j12_j+1;
                end
            end
        end
        j11_i=j11_i+1; j11_j=1;
        j12_i=j12_i+1; j12_j=1;
    end
    jfull=[j11,j12;j21,j22];
end
```

Published with MATLAB® R2015a

pfunc

Calculates the real Power mismatch value

Contents

USAGE

- **[out]=pfunc(P_index,Vvect,Tvect)**

INPUTS

- **P_index**: index of the power vector Pvect that the jacobian is in terms of
- **Vvect**: vector of voltage data
- **Tvect**: vector of voltage angle data
- **Ybus**: full ybus matrix

OUTPUTS

- **out**: result of the real power provided given the input

```
function [out]=pfunc(P_index,Voltage,Theta,Ybus)
    % From Slide 37 in Notes
    out = 0;
    for n=1:length(Voltage)
        out=out + pbranch(P_index,n,Voltage,Theta,Ybus);
    end
end
```

Published with MATLAB® R2015a

pbranch()

calculate the power flow on a single branch

Contents

USAGE

- **[out]=pbranch(from,to,Voltage,Theta,Ybus)**

INPUTS

- **from_bus**: bus to start at
- **to_bus**: bus to end at
- **Voltage**: vector of voltage data
- **Theta**: vector of voltage angle data
- **Ybus**: full ybus matrix

OUTPUTS

- **out**: result of the real power on the target branch for given input

```
function [out]=pbranch(from,to,Voltage,Theta,Ybus)
    out=Voltage(from)*Voltage(to)*(real(Ybus(from,to))*cos(Theta(from)-Theta(to))...
                                  +imag(Ybus(from,to))*sin(Theta(from)-Theta(to)));
end
```

Published with MATLAB® R2015a

qfunc

Calculates the reactive Power mismatch value

Contents

USAGE

- **[out]=qfunc(Q_index,Vvect,Tvect)**

INPUTS

- **Q_index**: index of implicit reactive power equation we are working on
- **Voltage**: vector of voltage data
- **Theta**: vector of voltage angle data
- **Ybus**: full ybus matrix

OUTPUTS

- **out**: result of the reactive power provided given the input

```
function [out]=qfunc(Q_index,Voltage,Theta,Ybus)
    % From Slide 37 in Notes
    out = 0;
    for n=1:length(Voltage)
        out=out + qbranch(Q_index,n,Voltage,Theta,Ybus);
    end
end
```

Published with MATLAB® R2015a

qbranch()

calculate the reactive power flow on a single branch

Contents

USAGE

- **[out]=qbranch(from,to,Voltage,Theta,Ybus)**

INPUTS

- **from_bus**: bus to start at
- **to_bus**: bus to end at
- **Voltage**: vector of voltage data
- **Theta**: vector of voltage angle data
- **Ybus**: full ybus matrix

OUTPUTS

- **out**: result of the reactive power on the target branch for given input

```
function [out]=qbranch(from,to,Voltage,Theta,Ybus)
    out=Voltage(from)*Voltage(to)*(real(Ybus(from,to))*sin(Theta(from)-Theta(to))...
                                   -imag(Ybus(from,to))*cos(Theta(from)-Theta(to)));
end
```

Published with MATLAB® R2015a

jptheta

Calculates the Jacobian output for the partial of any P with respect to any Theta

Contents

USAGE

- **[out]=jptheta(P_index,T_index,Vvect,Tvect)**

INPUTS

- **P_index**: index of the real power this jacobian entry is calculated for
- **T_index**: index of the theta this jacobian entry is calculated for
- **Voltage**: vector of voltage data
- **Theta**: vector of voltage angle data
- **Ybus**: full ybus matrix

OUTPUTS

- **out**: result of the Jacobian for (partial P(P_index))/(partial Theta(T_index))

```
function [out]=jptheta(P_index,T_index,Voltage,Theta,Ybus)
    % From Slide 55 in Notes
    if P_index == T_index
        Qii=qfunc(P_index,Voltage,Theta,Ybus);
        out=-1*Qii-imag(Ybus(P_index,P_index))*Voltage(P_index)^2;
    else
        out=Voltage(P_index)*Voltage(T_index)*(real(Ybus(P_index,T_index))*sin(Theta(P_index)-Theta(T_index))...
                                               -imag(Ybus(P_index,T_index))*cos(Theta(P_index)-Theta(T_index)));
    end
end
```

Published with MATLAB® R2015a

jpv

Calculates the Jacobian output for the partial of any P with respect to any V

Contents

USAGE

- **[out]=jptheta(Q_index,V_index,Pvect,Qvect,Vvect,Tvect)**

INPUTS

- **P_index**: index of the real power this jacobian entry is calculated for
- **V_index**: index of the voltage this jacobian entry is calculated for
- **Voltage**: vector of voltage data
- **Theta**: vector of voltage angle data
- **Ybus**: full ybus matrix

OUTPUTS

- **out**: result of the Jacobian for (partial Q(Q_index))/(partial V(V_index))

```
function [out]=jpv(P_index,V_index,Voltage,Theta,Ybus)
    % From Slide 57 in Notes
    if P_index == V_index
        Pi=pfunc(P_index,Voltage,Theta,Ybus);
        out=Pi/Voltage(P_index)+real(Ybus(P_index,V_index))*Voltage(P_index);
    else
        out=Voltage(P_index)*(real(Ybus(P_index,V_index))*cos(Theta(P_index)-Theta(V_index))...
                              +imag(Ybus(P_index,V_index))*sin(Theta(P_index)-Theta(V_index)));
    end
end
```

Published with MATLAB® R2015a

jqtheta

Calculates the Jacobian output for the partial of any Q with respect to any Theta

Contents

USAGE

- **[out]=jptheta(Q_index,T_index,Pvect,Qvect,Vvect,Tvect)**

INPUTS

- **Q_index**: index of the reactive power this jacobian entry is calculated for
- **T_index**: index of the theta this jacobian entry is calculated for
- **Voltage**: vector of voltage data
- **Theta**: vector of voltage angle data
- **Ybus**: full ybus matrix

OUTPUTS

- **out**: result of the Jacobian for (partial Q(Q_index))/(partial Theta(T_index))

```
function [out]=jqtheta(Q_index,T_index,Voltage,Theta,Ybus)
    % From Slide 56 in Notes
    if Q_index == T_index
        Pi=pfunc(Q_index,Voltage,Theta,Ybus);
        out=Pi-real(Ybus(Q_index,T_index))*Voltage(Q_index)^2;
    else
        out=-Voltage(Q_index)*Voltage(T_index)*(real(Ybus(Q_index,T_index))*cos(Theta(Q_index)-Theta(T_index))...
                                               +imag(Ybus(Q_index,T_index))*sin(Theta(Q_index)-Theta(T_index)));
    end
end
```

Published with MATLAB® R2015a

jqv

Calculates the Jacobian output for the partial of any Q with respect to any V

Contents

USAGE

- **[out]=jptheta(Q_index,V_index,Pvect,Qvect,Vvect,Tvect)**

INPUTS

- **Q_index**: index of the reactive power this jacobian entry is calculated for
- **V_index**: index of the voltage this jacobian entry is calculated for
- **Voltage**: vector of voltage data
- **Theta**: vector of voltage angle data
- **Ybus**: full ybus matrix

OUTPUTS

- **out**: result of the Jacobian for (partial Q(Q_index))/(partial V(V_index))

```
function [out]=jqv(Q_index,V_index,Voltage,Theta,Ybus)
    % From Slide 58 in Notes
    if Q_index == V_index
        Qii=qfunc(Q_index,Voltage,Theta,Ybus);
        out=Qii/Voltage(Q_index)-imag(Ybus(Q_index,V_index))*Voltage(Q_index);
    else
        out=Voltage(Q_index)*(real(Ybus(Q_index,V_index))*sin(Theta(Q_index)-Theta(V_index))...
                              -imag(Ybus(Q_index,V_index))*cos(Theta(Q_index)-Theta(V_index)));
    end
end
```

Published with MATLAB® R2015a

ybus

Calculates the ybus matrix from bus and branch data

Contents

USAGE

- **[yb]=ybus(bus,branch)**

INPUTS

- **bus**: bus G and B admittance data
- **branch**: branch R, X, G, and B data

OUTPUTS

- **yb**: ybus matrix
- **err**: blank if no problems, error string if problem

```
function [yb,err]=ybus(BusNums,BusG,BusB,branch)
    % Parse branch
    [From,To,R,X,~,B,err]=parse_branch_data(branch);
    if(isempty(err)==0)
        disp(err);
        return;
    end
    rowcount=length(From);

    buscount=length(BusNums);
    yb_offdiag=zeros(buscount); % defaults to square size
    yb_B=zeros(buscount);

    % Off-Diagonal
    for x=1:rowcount
        f=From(x);
        t=To(x);
        yval=1/(R(x)+X(x)*1i);
        yb_offdiag(f,t)=-yval;
        yb_offdiag(t,f)=yb_offdiag(f,t);
        yb_B(f,t)=B(x)*1i;
        yb_B(t,f)=yb_B(f,t);
    end

    % On-Diagonal
    yb=yb_offdiag;
    for x=1:buscount
        busnum=BusNums(x);
```

```
        offdiagsum = -1*(sum(yb_offdiag(x,:)));
        offdiag_Bs = sum(yb_B(x,:))/2;
        bus_g=BusG(x);
        bus_b=BusB(x);
        yb(busnum,busnum)=offdiagsum + bus_g + bus_b*1i + offdiag_Bs;
    end
end
```

Published with MATLAB® R2015a

jacobianCount

Counts the number of implicit P and implicit Q functions given a bus.

Contents

USAGE

- **[pcount,qcount]=jacobianCount(BusTypes)**

INPUTS

- **BusTypes**: column vector of bustypes, 1=slack, 2=PQ, 3=PV

OUTPUTS

- **pcount**: number of implicit P equations
- **qcount**: number of implicit Q equations
- **err**: error if any happened, emtpy string otherwise

```
function [pcount,qcount,err]=jacobianCount(BusTypes)
    buscount=length(BusTypes);
    err='';
    if max(max(BusTypes))>3 || min(min(BusTypes))<1
        err='ERROR: jacobianCount BusTypes exceed 1-3 range';
    end
    pcount=0;
    qcount=0;
    for n=1:buscount
        if BusTypes(n)==1     % Slack has no implicit equations
            continue;
        elseif BusTypes(n)==2 % PQ has two implicit equations
            pcount=pcount+1;
            qcount=qcount+1;
        elseif BusTypes(n)==3 % PV has one implicit equation
            pcount=pcount+1;
        end
    end
end
```

Published with MATLAB® R2015a

mismatch

Calculates the mismatch between power function estimate and provided values

Contents

USAGE

- **[Pmm,Qmm]=mismatch(P_index,T_index,Vvect,Tvect)**

INPUTS

- **P**: vector of real power values
- **Q**: vector of reactive power values
- **V**: vector of voltage values
- **T**: vector of theta values
- **BusTypes**: bus type vector (1=slack,2=pq,3=pv)
- **Ybus**: full ybus matrix

OUTPUTS

- **Pmm**: mismatch for P
- **Qmm**: mismatch for Q

```
function [Pmm,Qmm,err]=mismatch(P,Q,V,T,BusTypes,Ybus)
    [pcount,qcount,err]=jacobianCount(BusTypes);
    mismatch_index_P=1;
    mismatch_index_Q=1;
    Pmm=zeros(pcount,1);
    Qmm=zeros(qcount,1);
    buscount=length(BusTypes);
    for n=1:buscount
        if(BusTypes(n)==1) % Slack
            continue;
        end
        Pmm(mismatch_index_P,1)=pfunc(n,V,T,Ybus)-P(n);
        mismatch_index_P=mismatch_index_P+1;
        if BusTypes(n)==2 % PQ
            Qmm(mismatch_index_Q,1)=qfunc(n,V,T,Ybus)-Q(n);
            mismatch_index_Q=mismatch_index_Q+1;
        end
    end
end
```

Published with MATLAB® R2015a

figureplot

Produces a figure plot with all the fancy stuff I like - bolded titles, white background. Can be run with no inputs to pre-generate the figure which can then be passed in to reuse it.

Contents

USAGE

- **[subplot_out,fig_out]=figureplot(x,y,figtitle,figxlabel,figylabel,subplotId,fig,plotstyle,plotcolor)**

INPUTS

- **x (optional)**: x values (must be same size as y)
- **y (optional)**: y values (must be same size as y)
- **figtitle (optional)**: figure title
- **figxlabel (optional)**: figure xaxis label
- **figylabel (optional)**: figure yaxis label
- **subplotId (optional)**: subplot to add to
- **fig (optional)**: figure to add to
- **plotstyle (optional)**: the typeof line to use (eg: "-" or ".")
- **plotcolor (optional)**: the color to make the line

OUTPUTS

- **subplot_out**: subplot of the figure produced (so it can be reused)
- **fig_out**: figure of the figure produced (so it can be reused)

```
function [subplot_out,fig_out]=figureplot(x,y,figtitle,figxlabel,figylabel,subplotId,fig,plotstyle,plotcolor)
    if(nargin < 3)
        figtitle='Title';
    end
    if(nargin < 4)
        figxlabel='x-axis';
    end
    if(nargin < 5)
        figylabel='y-axis';
    end
    if(nargin < 7)
        fig=figure('Color',[1 1 1]);
    end
    if(nargin < 6)
        subplotId=subplot(1,1,1,'Parent',fig,'FontWeight','bold');
    end
    if(nargin < 8)
        plotstyle='-';
    end
    if(nargin < 9)
        plotcolor='';
    end
```

```
    hold(subplotId,'all');
    box(subplotId);
    if(nargin >= 2)
        if(size(plotcolor) > 0)
            plot(x,y,plotstyle,'Parent',subplotId,'MarkerEdgeColor',plotcolor);
        else
            plot(x,y,plotstyle,'Parent',subplotId);
        end
    end
    title(figtitle,'FontWeight','bold');
    xlabel(figxlabel,'FontWeight','bold');
    ylabel(figylabel,'FontWeight','bold');
    subplot_out = subplotId;
    fig_out=fig;
    grid on;
end
```

Published with MATLAB® R2015a

parse_branch_data(branchdata,datatype)

parse branch data into desired numbers @todo in the future, this should allow for different branchdata types Branch Data From To R(pu) X(pu) G(pu) B(pu)

```
function [From,To,R,X,G,B,err]=parse_branch_data(branchdata,datatype)
    if(nargin<2)
        datatype=1;
    end
    if(datatype==1)
        From=branchdata(:,1);
        To=branchdata(:,2);
        R=branchdata(:,3);
        X=branchdata(:,4);
        G=branchdata(:,5);
        B=branchdata(:,6);
        err='';
    else
        err='Unknown datatype.';
    end
end
```

Published with MATLAB® R2015a

mattex

prints matrix ready for LaTeX tabular environment

Contents

INPUTS

- **mat**: matrix to print
- **style**: defaults to matrix with & and //, 1 eliminates &, 2 eliminates & and //

```
function latexprint(mat,style)
    if(nargin<2)
        style=0;
    end
    [r,c]=size(mat);

    for(i=1:r)
        num=nearzero(mat(i,1));
        if(num==0)
            fprintf('%d',num);
        else
            fprintf('%f',num);
        end
        for(j=2:c)
            if(style==1 || style==2)
                fprintf(' ');
            else
                fprintf('   &   ');
            end
            num=nearzero(mat(i,j));
            if(num==0)
                fprintf('%d',num);
            else
                fprintf('%f',num);
            end
        end
        if(style==2)
            fprintf('\n');
        else
            fprintf('\\\\\n');
        end
    end
end
```


Published with MATLAB® R2015a

nearzero()

returns zero for anything close to zero

```
function [result]=nearzero(val,PRECISION)
    if(nargin<2)
        PRECISION=0.0000001;
    end
    if(abs(val)<PRECISION)
        result=0;
    else
        result=val;
    end
end
```

Published with MATLAB® R2015a

compare_maps(mapL,mapR)

Compares two maps to see if they are equivalent. Works at one nested level with maps (map of maps) Only works on numerical data (no strings) Works on arrays and matrixes

Contents

USAGE

- **[err]=compare_maps(mapL,mapR)**

INPUTS

- **mapL**: one map to compare
- **mapR**: second map to compare

OUTPUTS

- **err**: error if the maps are not equal or an empty string if they are

```
function [err]=compare_maps(mapL,mapR)
    keysL = mapL.keys;
    keysR = mapR.keys;

    % Since at this point it is known that keysL == keysR, keysL is used as a base
    results = zeros(size(keysL));
    for i = 1:length(keysL)
        value=mapL(keysL{i});
        if(mapR.isKey(keysL{i})==0)
            fprintf('Key "%s" in mapL is missing in mapR\n',keysL{i});
            results(i)=1;
            continue;
        end
        if(isa(value,'containers.Map'))
            itermapL=mapL(keysL{i});
            itermapR=mapR(keysL{i});
            iterKeys=itermapL.keys;
            results(i)=1;
            for j = 1:length(iterKeys)
                results(i)=results(i) && all(all(itermapL(iterKeys{j}) == itermapR(iterKeys{j})));
            end
        else
            results(i) = all(size(mapL(keysL{i})) == size(mapR(keysL{i})));
            if(results(i)==1)
                results(i) = all(all(mapL(keysL{i}) == mapR(keysL{i})));
            end
        end
    end
    for i = 1:length(keysR)
```

```
        if(mapL.isKey(keysR{i})==0)
            fprintf('Key "%s" in mapR is missing in mapL\n',keysL{i});
            continue;
        end
    end

    if all(results)
        err='';
    else
        err='Maps are not equal';
    end
end
```

Published with MATLAB® R2015a

nrpf_test

Runs a known system through nrpf to verify I haven't broken it

Contents

USAGE

- **[mapL,mapR,err]=nrpf_test()**

INPUTS

- **PRINT_ITERS**: passed into nrpf, prints out each iteration of = 1

OUTPUTS

- **mapL**: old version of the results
- **mapR**: new version of the results
- **err**: error result or empty string if there was none

```
function [mapL,mapR,err]=nrpf_test(PRINT_ITERS)
```

```
    if(nargin<1)
        PRINT_ITERS=0;
    end
```

Test Data from Homework 8, Problem 2

BusData Bustype: 1=slack, 2=PQ, 3=PV Bus Type PG QG PL QL V Theta G B

```
    busdata=[1      1      0    0   0    0      1.0  0      0   0;
             2      2      0    0   1    0.5    1.0  0      0   0;
             3      2      0    0   1.5  0.75   1.0  0      0   0;
    ];
    % Branch Data
    % Reverse Engineered to create provided ybus
    % ybus=1i*[-10,   5,   5;
    %            5, -10,   5;
    %            5,   5,  -10
    % ];
    %               From  To  R(pu)    X(pu)    G(pu)  B(pu)
```

```
    branchdata=[     1     2   0        0.2      0      0;
                     1     3   0        0.2      0      0;
                     2     3   0        0.2      0      0;
    ];
```

Run NRPF

```
    load('nrpf_test_results.mat');
    mapL=nrpf_test_results;
    [mapR,err]=nrpf(busdata,branchdata,PRINT_ITERS);
    if(isempty(err)==0)
        disp(err);
        return;
    end
```

Verify Results

```
    err=compare_maps(mapL,mapR);
    if(isempty(err)==0)
        disp('WARNING: Regression Test Failed');
    else
        disp('Regression Test Passed');
    end
```

```
end
```

Published with MATLAB® R2015a