**Assignment 3**

Submitted by Athul Jose P

WSU ID 011867566

**Executive Summary**

***State Estimation***

Steps in main.m function

1. Initializing 14 bus and importing data
2. Making Ybus by calling y\_bus\_calc.m (with taps or without taps should be mentioned)
3. Calculating the scheduled active power (P) and reactive power (Q)
4. Finding bus types and assigning to vectors
5. Initializing Voltage magnitude and angles
6. Calling Newton Raphson Function (NR.m)
7. Calculating P & Q after convergence by calling PQ\_calc.m
8. Calculating Pline and Qlines by calling PQline\_calc.m
9. Adding noise to the measurement
10. Initializing state estimation parameters with/without bad data
11. Starting iteration loop which will terminate when error greater than a tolerance
12. Calculating P & Q for SE
13. Calculating Pline and Qlines for SE
14. Generating Hx matrix by calling Hx\_calc.m
15. Calculating x hat vector
16. Updating voltage, angle and error
17. Calculating chi squared value
18. Calculating chi squared limit
19. Comparing them to show whether bad data is present or not

Steps in y\_bus\_calc.m function

1. Initializing Ybus with zeros
2. Calculating diagonal and off diagonal elements
3. Changing terms if tap is present

Steps in NR.m function

1. Initializing indexes and deltas
2. Starting iteration loop which will terminate either if converged or 100 iterations
3. Calling dpdq.m for calculating mismatch
4. Calling J\_calc.m for calculating Jacobian
5. Calling fwd\_bwd.m for calculating the ΔV and Δδ
6. Updating del\_V and del\_T (magnitude and angle) for next iteration
7. Updating the error. Here the error is taken as the maximum of absolute of deltas (ΔV and Δδ)

Steps in dpdq.m function

1. Initializing P & Q as zeros
2. Calculating P for PV bus and P & Q for PQ bus

Steps in J\_calc.m function

1. Calculating J1 with loops according to limits (n\_bus-1, n\_bus-1)
2. Calculating J2 with loops according to limits (n\_bus-1, n\_pq)
3. Calculating J3 with loops according to limits (n\_pq, n\_bus-1)
4. Calculating J4 with loops according to limits (n\_pq, n\_pq)
5. Combining all to make J

Steps in fwd\_bwd.m function

1. Calling LU.m for calculating Lower and Upper elements
2. Doing the backward substitution
3. Doing the forward substitution

Steps in LU.m function

1. Making the Q matrix
2. Dividing it into L & U matrices

Steps in PQ\_calc.m function

1. Calculates the P & Q of each bus

Steps in PQline\_calc.m function

1. Calculated Pij, Pji, Qij, Qji of each lines

Steps in Hx\_calc.m function

1. Calling J\_full\_calc.m to calculate all Jacobian values
2. Generating each and every Hx factor matrices
3. Combining all to form Hx matrix

Steps in J\_full\_calc.m function

1. Calculates every Jacobian terms corresponding no of bus
2. Combines 4 Jacobian components into one J

**Results**

1. Line power flows

|  |  |  |  |
| --- | --- | --- | --- |
| Pij | Pji | Qij | Qji |
| 1.568828865 | -1.525852864 | 0.759883338 | -0.685141284 |
| 0.755103798 | -0.727475074 | 0.384974317 | -0.31779948 |
| 0.732375781 | -0.709143088 | 0.358382674 | -0.309512589 |
| 0.561314946 | -0.544548368 | 0.317633637 | -0.310801239 |
| 0.415162135 | -0.406124603 | 0.251385597 | -0.268253643 |
| -0.232856906 | 0.236591356 | -0.137910268 | 0.107419203 |
| -0.611582311 | 0.616726507 | -0.240944492 | 0.221584108 |
| 0.280741765 | -0.280741765 | -0.210721247 | 0.215898419 |
| 0.160797583 | -0.160797583 | -0.065753225 | 0.062503234 |
| 0.440873188 | -0.440873188 | -0.204815889 | 0.231153771 |
| 0.073532769 | -0.072979036 | 0.079569127 | -0.08555867 |
| 0.077860669 | -0.077142576 | 0.076430528 | -0.082061884 |
| 0.177479767 | -0.175358913 | 0.185874842 | -0.18875935 |
| 3.32E-15 | -2.51E-15 | -0.171629675 | 0.176234481 |
| 0.280741786 | -0.280741786 | 0.057786902 | -0.049766213 |
| 0.052275506 | -0.052146757 | 0.066190336 | -0.065848329 |
| 0.094263803 | -0.093102262 | 0.095618849 | -0.093148096 |
| -0.037853243 | 0.037979056 | -0.038515516 | 0.03881003 |
| 0.016142577 | -0.016079595 | 0.015309987 | -0.015253003 |
| 0.056438518 | -0.055897738 | 0.055345662 | -0.054244615 |

2. Added error of 2% in voltage measurement and 10% in power measurements.

3. Done state estimation and results given below:-

*Case 1: Noisy measurements without any bad data*

**chi\_squared\_value =**

**99.982706322994**

**chi\_squared\_limit =**

**129.972678726799**

**No Bad Data**

*Case 2: Noisy measurements with one bad data*

**chi\_squared\_value =**

**16981.6928125453**

**chi\_squared\_limit =**

**129.972678726799**

**Measurement contains Bad Data**

*Case 3: Noisy measurements with three bad data*

**chi\_squared\_value =**

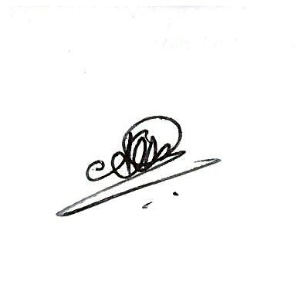
**566538.933408122**

**chi\_squared\_limit =**

**129.972678726799**

**Measurement contains Bad Data**

4. From the results, it is clear that state estimation is able to differentiate bad data from noisy measurement. In the first case with noisy measurement, chi squared value remained in the limit. In case 2 & 3, it shoot up to a very high value and state estimation could detect there is bad data in the measurement

Statement

I, Athul Jose P, states that all the code written and submitted here is completely done by me. I have not taken any help from others or any online resources.

Athul Jose P

*Appendix*

**% main.m**

**clc**

**clear all; close all;**

**% Initializing 14 bus and importing data**

**n\_bus = 14;**

**bus\_data = importdata('ieee14bus.txt').data;**

**branch\_data = importdata('ieee14branch.txt').data;**

**% Ybus formation**

**t = 1; % 0 for without tap, 1 for with tap**

**Y = y\_bus\_calc(n\_bus,bus\_data,branch\_data,t);**

**% Scheduled power calculation**

**base\_MVA = 100;**

**P\_inj = (bus\_data(:,8) - bus\_data(:,6)) / base\_MVA;**

**Q\_inj = (bus\_data(:,9) - bus\_data(:,7)) / base\_MVA;**

**% Creating indexes for functions**

**pv\_i = find(bus\_data(:,3) == 2);**

**pq\_i = find(bus\_data(:,3) == 0);**

**n\_pv = length(pv\_i);**

**n\_pq = length(pq\_i);**

**fr = branch\_data(:,1);**

**to = branch\_data(:,2);**

**n\_br = length(fr);**

**B = branch\_data(:,9);**

**% Initializing Voltage magnitude and angles**

**V = bus\_data(:,11);**

**V(find(V(:)==0)) = 1;**

**T = zeros(n\_bus,1);**

**% Executing NR for Power Flow results**

**[V\_data,T\_data,T1] = NR(bus\_data,V,T,P\_inj,Q\_inj,n\_bus,Y,n\_pq,pq\_i);**

**V = V\_data(:,size(V\_data,2));**

**T = T\_data(:,size(T\_data,2));**

**% P,Q calculation after convergence**

**[P,Q] = PQ\_calc(V,T,Y);**

**% calculating line power flows**

**[Pij Pji Qij Qji] = PQline\_calc(V,T,Y,n\_br,fr,to,B)**

**% for\_graph = [[Pij Pji Qij Qji]]**

**z = [V;P;Q;Pij;Pji;Qij;Qji];**

**% Voltage & Power measurement error**

**V\_tol = 2/100;**

**P\_tol = 10/100;**

**% Introducing noise into measurement**

**n = [V\_tol\*randn(length([V]), 1);P\_tol\*randn(length([P;Q;Pij;Pji;Qij;Qji]), 1)];**

**var = n.^2;**

**z\_m = z + n;**

**W = zeros(length(var),length(var));**

**for i=1:length(var)**

**for j=1:length(var)**

**if i == j**

**W(i,j) = 1/var(i);**

**end**

**end**

**end**

**% Initializing SE parameters**

**error = 1;**

**iter = 0;**

**SE\_tol = 0.001;**

**% Adding Bad Data**

**z\_m(2) = 3;**

**z\_m(26) = 10;**

**z\_m(54) = 10;**

**while error >= SE\_tol**

**% calculating bus powers**

**[P,Q] = PQ\_calc(V,T,Y);**

**% calculating line power flows**

**[Pij Pji Qij Qji] = PQline\_calc(V,T,Y,n\_br,fr,to,B);**

**% Generating Hx matric**

**Hx = Hx\_calc(V, T, Y,B,P, Q, n\_bus,n\_br,fr,to);**

**x = [T(2:end); V];**

**x\_prev = x;**

**h = [V;P;Q;Pij;Pji;Qij;Qji];**

**x = x + inv(Hx' \* W \* Hx) \* Hx' \* W \* (z\_m-h);**

**T(2:end) = x(1:n\_bus-1);**

**V = x(n\_bus:end);**

**iter = iter + 1;**

**error = max(abs(x - x\_prev));**

**end**

**% calculating chi squared value**

**chi\_squared\_value = 0;**

**for i=1:length(var)**

**chi\_squared\_value = chi\_squared\_value + (z\_m(i) - h(i))^2 / var(i);**

**end**

**chi\_squared\_value**

**% Finding chi squared limit**

**chi\_squared\_limit = chi2inv(0.99, length(z\_m) - length(x))**

**% Checking bad data present or not**

**if chi\_squared\_value >= chi\_squared\_limit**

**disp('Measurement contains Bad Data')**

**else**

**disp('No Bad Data')**

**end**

**% y\_bus\_calc.m**

**function Y = y\_bus\_calc(N\_bs,D\_bs,D\_br,t)**

**Y = zeros(N\_bs);**

**% Calculating elements of Ybus**

**for k = 1:size(D\_br,1)**

**Y(D\_br(k,1),D\_br(k,1)) = Y(D\_br(k,1),D\_br(k,1)) + 1/(D\_br(k,7) + i\*D\_br(k,8)) + i\*D\_br(k,9)/2;**

**Y(D\_br(k,2),D\_br(k,2)) = Y(D\_br(k,2),D\_br(k,2)) + 1/(D\_br(k,7) + i\*D\_br(k,8)) + i\*D\_br(k,9)/2;**

**Y(D\_br(k,1),D\_br(k,2)) = -1/(D\_br(k,7) + i\*D\_br(k,8));**

**Y(D\_br(k,2),D\_br(k,1)) = Y(D\_br(k,1),D\_br(k,2));**

**end**

**for k = 1:N\_bs**

**Y(k,k) = Y(k,k) + D\_bs(k,14) + i\*D\_bs(k,15);**

**end**

**% adjusting for taps**

**if(t == 1)**

**for k = 1:size(D\_br,1)**

**if(D\_br(k,15) ~= 0)**

**t = D\_br(k,15);**

**((t^2) / i\*D\_br(k,8));**

**Y(D\_br(k,1),D\_br(k,1)) = Y(D\_br(k,1),D\_br(k,1)) + Y(D\_br(k,1),D\_br(k,2)) - (Y(D\_br(k,1),D\_br(k,2)))/(t^2);**

**Y(D\_br(k,1),D\_br(k,1));**

**Y(D\_br(k,1),D\_br(k,2)) = Y(D\_br(k,1),D\_br(k,2))/t;**

**Y(D\_br(k,2),D\_br(k,1)) = Y(D\_br(k,1),D\_br(k,2));**

**end**

**end**

**end**

**end**

**% NR.m**

**function [V\_data,T\_data] = NR(bus\_data,V,T,P\_inj,Q\_inj,n\_bus,Y,n\_pq,pq\_i)**

**% Initializing index**

**i = 0;**

**Tol = 1;**

**del\_T = zeros(n\_bus,1);**

**del\_V = zeros(n\_bus,1);**

**% Iteration loop**

**while(Tol > 1e-5 & i < 100)**

**i = i+1**

**V = V+del\_V;**

**T = T+del\_T;**

**T\_data(:,i) = T;**

**V\_data(:,i) = V;**

**[del\_P, del\_Q] = dpdq\_calc(bus\_data,V,T,P\_inj,Q\_inj,n\_bus,Y);**

**dpdq = [del\_P, del\_Q]; % mismatch calculation**

**J = J\_calc(bus\_data,V,T,Y,n\_bus,n\_pq,pq\_i); % Jacobian calculation**

**delta = fwd\_bwd(J,dpdq); % finding errors**

**del\_T = [0 delta(1:n\_bus-1)]';**

**for j = 1:n\_pq**

**del\_V(pq\_i(j)) = delta(n\_bus+j-1);**

**end**

**Tol = max(abs(delta)) % updating error for convergence**

**end**

**end**

**% dpdq.m**

**function [del\_P, del\_Q] = dpdq\_calc(bus\_data,V,T,P\_inj,Q\_inj,n\_bus,Y)**

**P = zeros(n\_bus,1);**

**Q = zeros(n\_bus,1);**

**Pi = 1;**

**Qi = 1;**

**for i = 1:n\_bus**

**if(bus\_data(i,3) ~= 3)**

**for j = 1:n\_bus**

**P(i) = P(i) + V(i)\*V(j)\*abs(Y(i,j))\*cos(T(i)-T(j)-angle(Y(i,j)));**

**Q(i) = Q(i) + V(i)\*V(j)\*abs(Y(i,j))\*sin(T(i)-T(j)-angle(Y(i,j)));**

**end**

**del\_P(Pi) = P\_inj(i) - P(i);**

**Pi = Pi+1;**

**if(bus\_data(i,3) == 0)**

**del\_Q(Qi) = Q\_inj(i) - Q(i);**

**Qi = Qi+1;**

**end**

**end**

**end**

**end**

**% J\_calc.m**

**function J = J\_calc(bus\_data,V,T,Y,n\_bus,n\_pq,pq\_i)**

**% J1 calculation**

**J1 = zeros(n\_bus-1);**

**for i = 1:n\_bus**

**for j = 1:n\_bus**

**if(bus\_data(i,3) ~=3 & bus\_data(j,3) ~=3)**

**if(i==j)**

**for k = 1:n\_bus**

**J1(i-1,j-1) = J1(i-1,j-1)+(V(i)\*V(k)\*abs(Y(i,k))\*sin(angle(Y(i,k))-T(i)+T(k)));**

**end**

**J1(i-1,j-1) = J1(i-1,j-1) - ((V(i)^2) \* (imag(Y(i,i))));**

**else**

**J1(i-1,j-1) = -V(i)\*V(j)\*abs(Y(i,j))\*sin(angle(Y(i,j))-T(i)+T(j));**

**end**

**end**

**end**

**end**

**J1;**

**% J2 calculation**

**J2 = zeros(n\_bus-1,n\_pq);**

**for i = 2:n\_bus**

**for j = 1:n\_pq**

**n = pq\_i(j);**

**if(n == i)**

**for k = 1:n\_bus**

**J2(i-1,j) = J2(i-1,j)+(V(i)\*V(k)\*abs(Y(i,k))\*cos(angle(Y(i,k))-T(i)+T(k)));**

**end**

**J2(i-1,j) = J2(i-1,j) + ((V(i)^2) \* (real(Y(i,i))));**

**else**

**J2(i-1,j) = V(i)\*V(n)\*abs(Y(i,n))\*cos(angle(Y(i,n))-T(i)+T(n));**

**end**

**end**

**end**

**J2;**

**% J3 calculation**

**J3 = zeros(n\_pq,n\_bus-1);**

**for i = 1:n\_pq**

**n = pq\_i(i);**

**for j = 2:n\_bus**

**if(n==j)**

**for k = 1:n\_bus**

**J3(i,j-1) = J3(i,j-1)+(V(n)\*V(k)\*abs(Y(n,k))\*cos(angle(Y(n,k))-T(n)+T(k)));**

**end**

**J3(i,j-1) = J3(i,j-1) - ((V(n)^2) \* (real(Y(n,n))));**

**else**

**J3(i,j-1) = -V(n)\*V(j)\*abs(Y(n,j))\*cos(angle(Y(n,j))-T(n)+T(j));**

**end**

**end**

**end**

**J3;**

**% J4 calculation**

**J4 = zeros(n\_pq);**

**for i = 1:n\_pq**

**n1 = pq\_i(i);**

**for j = 1:n\_pq**

**n2 = pq\_i(j);**

**if(n1==n2)**

**for k = 1:n\_bus**

**J4(i,j) = J4(i,j)+(V(n1)\*V(k)\*abs(Y(n1,k))\*sin(angle(Y(n1,k))-T(n1)+T(k)));**

**end**

**J4(i,j) = - J4(i,j) - ((V(n1)^2) \* (imag(Y(n1,n1))));**

**else**

**J4(i,j) = -V(n1)\*V(n2)\*abs(Y(n1,n2))\*sin(angle(Y(n1,n2))-T(n1)+T(n2));**

**end**

**end**

**end**

**J4;**

**J = [J1, J2; J3, J4];**

**end**

**% fwd\_bwd.m**

**function x = fwd\_bwd(A,b)**

**[L, U] = LU(A);**

**% Forward Substitution**

**for k = 1:length(A)**

**s = 0;**

**for j = 1:k-1**

**s = s + (L(k,j)\*y(j));**

**end**

**y(k) = (b(k) - s) / L(k,k);**

**end**

**% Backward Substitution**

**for k = length(A):-1:1**

**s = 0;**

**for j = k+1:length(A)**

**s = s + (U(k,j)\*x(j));**

**end**

**x(k) = y(k) - s;**

**end**

**end**

**% LU.m**

**function [L, U] = LU(a)**

**Q = zeros(length(a));**

**for j = 1:length(a)**

**for k = j:length(a)**

**s = 0;**

**for m = 1:j-1**

**s = s + (Q(k,m)\*Q(m,j));**

**end**

**Q(k,j) = a(k,j) - s;**

**end**

**if j < length(a)**

**for k = j+1:length(a)**

**s = 0;**

**for m = 1:j-1**

**s = s + (Q(j,m)\*Q(m,k));**

**end**

**Q(j,k) = (a(j,k) - s) / Q(j,j);**

**end**

**end**

**end**

**L = tril(Q);**

**U = Q - L + eye(length(a));**

**end**

**% PQ\_calc.m**

**function [P,Q] = PQ\_calc(V,T,Y)**

**n\_bus = size(V,1);**

**P = zeros(n\_bus,1);**

**Q = zeros(n\_bus,1);**

**for i = 1:n\_bus**

**for j = 1:n\_bus**

**P(i) = P(i) + V(i)\*V(j)\*abs(Y(i,j))\*cos(T(i)-T(j)-angle(Y(i,j)));**

**Q(i) = Q(i) + V(i)\*V(j)\*abs(Y(i,j))\*sin(T(i)-T(j)-angle(Y(i,j)));**

**end**

**end**

**end**

**% PQline\_calc.m**

**function [Pij Pji Qij Qji] = PQline\_calc(V,T,Y,n\_br,fr,to,B)**

**for i = 1:n\_br**

**a = fr(i);**

**b = to(i);**

**Pij(i,1) = -(V(a)^2)\*real(Y(a,b)) + abs(V(a)\*V(b)\*Y(a,b))\*cos(angle(Y(a,b)) + T(b) - T(a));**

**Pji(i,1) = -(V(b)^2)\*real(Y(b,a)) + abs(V(b)\*V(a)\*Y(b,a))\*cos(angle(Y(b,a)) + T(a) - T(b));**

**Qij(i,1) = -((V(a)^2)\*(abs(B(b))/2 - imag(Y(a,b))) + abs(V(a)\*V(b)\*Y(a,b))\*sin(angle(Y(a,b)) + T(a) - T(b)));**

**Qji(i,1) = -((V(b)^2)\*(abs(B(a))/2 - imag(Y(b,a))) + abs(V(b)\*V(a)\*Y(b,a))\*sin(angle(Y(b,a)) + T(b) - T(a)));**

**end**

**end**

**% Hx\_calc.m**

**function H = Hx\_calc(V, T, Y,B,P, Q, n\_bus,n\_br,fr,to)**

**J = J\_full\_calc(V, T, P, Q, n\_bus, Y);**

**H1 = zeros(n\_bus, n\_bus-1);**

**H2 = eye(n\_bus, n\_bus);**

**H3 = J(1:n\_bus, 2:n\_bus);**

**H4 = J(1:n\_bus, n\_bus+1:end);**

**H5 = J(n\_bus+1:end, 2:n\_bus);**

**H6 = J(n\_bus+1:end, n\_bus+1:end);**

**for i = 1:n\_br**

**a = fr(i);**

**b = to(i);**

**if a~=1**

**H7(i, a-1) = abs(V(a) \* V(b) \* Y(a,b)) \* sin(angle(Y(a,b)) + T(b) - T(a));**

**H9(i, a-1) = -abs(V(b) \* V(a) \* Y(b,a)) \* sin(angle(Y(b,a)) + T(a) - T(b));**

**H11(i, a-1) = abs(V(a) \* V(b) \* Y(a,b)) \* cos(angle(Y(a,b)) + T(b) - T(a));**

**H13(i, a-1) = -abs(V(b) \* V(a) \* Y(b,a)) \* cos(angle(Y(b,a)) + T(a) - T(b));**

**end**

**if b~=1**

**H7(i, b-1) = -abs(V(a) \* V(b) \* Y(a,b)) \* sin(angle(Y(a,b)) + T(b) - T(a));**

**H9(i, b-1) = abs(V(b) \* V(a) \* Y(b,a)) \* sin(angle(Y(b,a)) + T(a) - T(b));**

**H11(i, b-1) = -abs(V(a) \* V(b) \* Y(a,b)) \* cos(angle(Y(a,b)) + T(b) - T(a));**

**H13(i, b-1) = abs(V(b) \* V(a) \* Y(b,a)) \* cos(angle(Y(b,a)) + T(a) - T(b));**

**end**

**H8(i, b) = abs(V(a) \* Y(a,b)) \* cos(angle(Y(a,b)) + T(b) - T(a));**

**H8(i, a) = -2\*V(a) \* real(Y(a,b)) + abs(V(b) \* Y(a,b)) \* cos(angle(Y(a,b)) + T(b) - T(a));**

**H10(i,a) = abs(V(b) \* Y(b,a)) \* cos(angle(Y(b,a)) + T(a) - T(b));**

**H10(i,b) = -2\*V(b) \* real(Y(b,a)) + abs(V(a) \* Y(b,a))\*cos(angle(Y(b,a)) + T(a) - T(b));**

**H12(i,b) = -abs(V(a) \* Y(a,b)) \* sin(angle(Y(a,b)) + T(b) - T(a));**

**H12(i,a) = -2 \* V(a) \* (0.5\*abs(B(i)) - imag(Y(a,b))) - abs(V(b) \* Y(a,b)) \* sin(angle(Y(a,b)) + T(b) - T(a));**

**H14(i,a) = -abs(V(b) \* Y(b,a)) \* sin(angle(Y(b,a)) + T(a) - T(b));**

**H14(i,b) = -2 \* V(b) \* (0.5 \* abs(B(i)) - imag(Y(b,a))) - abs(V(a) \* Y(b,a)) \* sin(angle(Y(b,a)) + T(a) - T(b));**

**end**

**H=[ H1, H2; H3, H4; H5, H6; H7, H8; H9, H10; H11, H12; H13, H14];**

**end**

**% J\_full\_calc.m**

**function J = J\_full\_calc(V, T, P, Q,n\_bus,Y)**

**for i = 1:n\_bus**

**for j = 1:n\_bus**

**if i ~= j**

**J1(i, j) = -abs(V(i) \* V(j) \* Y(i, j)) \* sin(angle(Y(i, j)) + T(j) - T(i));**

**J2(i, j) = abs(V(i) \* V(j) \* Y(i, j)) \* cos(angle(Y(i, j)) + T(j) - T(i));**

**J3(i, j) = -abs(V(i) \* V(j) \* Y(i, j)) \* cos(angle(Y(i, j)) + T(j) - T(i));**

**J4(i, j) = -abs(V(i) \* V(j) \* Y(i, j)) \* sin(angle(Y(i, j)) + T(j) - T(i));**

**else**

**J1(i, i) = -Q(i) - (abs(V(i))^2) \* imag(Y(i, i));**

**J2(i, i) = P(i) + (abs(V(i))^2) \* real(Y(i, i));**

**J3(i, i) = P(i) - (abs(V(i))^2) \* real(Y(i, i));**

**J4(i, i) = Q(i) - (abs(V(i))^2) \* imag(Y(i, i));**

**end**

**end**

**end**

**J=[J1 J2; J3 J4];**

**end**