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 $\Delta\delta$
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 $\Delta\delta$)

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
```
