Assignment 4

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. Case1: Doing economic dispatch by calling ED.m
9. Calculating Ptotal for case 2. Then does ED
10. Case3: Doing OPF without line limit by calling OPF.m
11. Case4: Doing OPF with line limit

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 ED.m function

1. Creating cost functions
2. Creating functions in sym environment
3. Finding the solution

Steps in OPF.m function

1. Initializing OPF parameters and syms variables
2. Starting loop for OPF with error tolerance as 0.1
3. Calculating different P&Q injections
4. Calculating different differential terms
5. Correcting the error values

Results

1. Economic Dispatch with P total = 259MW

```
Economic Dispatch with total P = 259MW

Share of Generator-1 is 50.3636MW

Share of Generator-2 is 208.6364MW

Cost of Generation is 8.4029/MWhr

Total Cost is $1957.300000/hr
```

2. Economic Dispatch with P total = P1 + P2

```
Economic Dispatch with total P = P1 + P2

Share of Generator-1 is 57.6691MW

Share of Generator-2 is 214.7242MW

Cost of Generation is 8.4614/MWhr

Total Cost is $2070.200000/hr
```

3. OPF without line limit

```
Optimal Power Flow without line limits

Share of Generator-1 is 57.6691MW
```

```
Share of Generator-2 is 214.6958MW

Total Cost is $2070.000000/hr
```

4. OPF with line limit

```
Optimal Power Flow with line limits

Share of Generator-1 is 50.7659MW

Share of Generator-2 is 217.0886MW

Total Cost is $2032.000000/hr
```

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);

% Case 1: Total P = 259 MW
P_tot = 259;
P_eq = [8.0 0.004; 6.4 0.0048];
[P_1, P_2, cost, x] = ED(P_eq,P_tot);
disp('Economic Dispatch with total P = 259MW')
disp(['Share of Generator-1 is ' num2str(P_1) 'MW'])
disp(['Share of Generator-2 is ' num2str(P_2) 'MW'])
disp(['Cost of Generation is ' num2str(x) '/MWhr'])
fprintf('Total Cost is $%f/hr\n', cost)
disp('-----------------------------------------------------------')

% Case 2: Total P = P1 + P2
P_tot = (P(1) + P(2) + bus_data(2,6)/base_MVA) * base_MVA;
[P_1, P_2, cost, x] = ED(P_eq,P_tot);
disp('Economic Dispatch with total P = P1 + P2')
disp(['Share of Generator-1 is ' num2str(P_1) 'MW'])
disp(['Share of Generator-2 is ' num2str(P_2) 'MW'])
disp(['Cost of Generation is ' num2str(x) '/MWhr'])
```

```
fprintf('Total Cost is $%f/hr\n', cost)
disp('------------------------------------------------------------')

syms T_2
PG0 = [P_1; P_2];
QG0 = [Q(1); Q(2)]*base_MVA;
bus_no = [1 2];
eq = P_tot == bus_data(2,6) + base_MVA*(real(Y(2,2))*V(2)^2 +
abs(Y(2,1)*V(1)*V(2))*cos(angle(Y(2,1)) + T_2));
T_2_soln= solve(eq, T_2);
T_2_Val = T_2_soln(1);

% Case 3: OPF without line limits
constraint = 0;
[P_1,P_2, cost] =
OPF(V,T,Y,PG0,QG0,bus_no,T_2_Val,base_MVA,constraint,bus_data,n_bus,n_pq,pq
_i,P_inj,Q_inj,P_eq);
disp('Optimal Power Flow without line limits')
disp(['Share of Generator-1 is ' num2str(P_1) 'MW'])
disp(['Share of Generator-2 is ' num2str(P_2) 'MW'])
fprintf('Total Cost is $%f/hr\n', cost)
disp('------------------------------------------------------------')

% Case 4: OPF with line limits
constraint = 1;
[P_1,P_2, cost] =
OPF(V,T,Y,PG0,QG0,bus_no,T_2_Val,base_MVA,constraint,bus_data,n_bus,n_pq,pq
_i,P_inj,Q_inj,P_eq);
disp('Optimal Power Flow with line limits')
disp(['Share of Generator-1 is ' num2str(P_1) 'MW'])
disp(['Share of Generator-2 is ' num2str(P_2) 'MW'])
fprintf('Total Cost is $%f/hr\n', cost)
disp('------------------------------------------------------------')
```

```
% 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
```

```
% ED.m
function [P_1, P_2, cost, x] = ED(P_eq,P_total)
y = sym('y', [2 1]);
syms P_tot x

y_x = num2cell(y);
C_1(y) = P_eq(1, 2)*y(1)^2 + P_eq(1, 1)*y(1);
C_2(y) = P_eq(2, 2)*y(2)^2 + P_eq(2, 1)*y(2);

yfull = [y; x];
f(y) = C_1(y_x{:}) + C_2(y_x{:});
c(y) = sum([y_x{:}].*[1 1]) - P_tot;
c(y) = subs(c(y_x{:}), P_tot, P_total);

y_x_all = num2cell(yfull);
C(yfull) = C_1(y_x{:}) + C_2(y_x{:}) - x*c(y_x{:});
Delta_C(yfull) = jacobian(subs(C(y_x_all{:}), P_tot, P_total), yfull).';

y_x_soln = solve(Delta_C(y_x_all{:}), y_x_all{:});
P_1 = double(y_x_soln.y1);
P_2 = double(y_x_soln.y2);
x = double(y_x_soln.x);
cost = f(P_1, P_2);
end
```

```
% OPF.m

function [P1,P2, cost] =
OPF(V,T,Y,PG0,QG0,bus_no,T_2_Val,base_MVA,limit,bus_data,n_bus,n_pq,pq_i,P_
inj,Q_inj,P_eq)
syms PG_i QG_i P12
u = sym('u', [1 1]);
u_r = num2cell(u);
x0 = sym('x', [n_bus+n_pq 1]);
x = x0(2:end);
```

```
% initializing OPF parameters
r = 0.06;
p_coef = 1;
corr = 100;
tol = 0.1;
P12_x = 100.0;
P12_Limit = 5;
iter = 1;

i = bus_no(1);
j = bus_no(2);
u_x = PG0(j);
xVals = [T_2_Val; T(3:end); V(pq_i)];
P1 = PG0(i);
Q1_Val = QG0(i)/base_MVA;
J = J_calc(bus_data,V,T,Y,n_bus,n_pq,pq_i);
V = V(2:end);
C_i = P_eq(1, 2)*PG_i^2 + P_eq(1, 1)*PG_i;
C_j(u) = P_eq(2, 2)*u^2 + P_eq(2, 1)*u;
f(u_r{:}) = C_i + C_j(u_r{:});

if limit == 1
    f(u_r{:}) = f(u_r{:}) + p_coef*(P12 - P12_Limit)^2;
end
display(f(u_r{:}));

% loop for OPF
while corr > tol || P12_x*limit > P12_Limit
    if iter > 1
        bus_data(2,8) = u_x;
        P_inj = (bus_data(:,8) - bus_data(:,6)) / base_MVA;
        V = bus_data(:,11);
        V(find(V(:)==0)) = 1;
        T = zeros(n_bus,1);

        [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] = PQ_calc(V,T,Y);

        xVals = [T(2:n_bus); V(pq_i)];
        P1 = P(1)*base_MVA;
        Q1_Val = Q(1)*base_MVA;

        J = J_calc(bus_data,V,T,Y,n_bus,n_pq,pq_i);
        P12_x = abs(abs(V(2)*V(1)*Y(1, 2))*cos(-T(2) - angle(Y(1, 2))) -
real(Y(1, 2))*(V(1))^2);
        P12_x = P12_x*base_MVA;
        dfdxP12 = p_coef*[2*(P12_x-5)*abs(V(1)*V(2)*Y(1,2))*sin(T(1)-T(2)-
angle(Y(1,2))); zeros(n_bus+n_pq-2, 1)];
    end
    dgdu = [1; zeros(n_bus + n_pq - 2, 1)];
    dgdx = - J;

    df_du_temp = jacobian(f(u_r{:}), u);
    dfdu = subs(df_du_temp, u, u_x);

    P_inj_1_temp = Pinj_calc(x, V, Y);
    P1_temp = P_inj_1_temp + bus_data(2,6)/base_MVA;
```

```
    Q_inj_1_temp = Qinj_calc(x, V, Y);
    Q1_temp = Q_inj_1_temp + bus_data(1,7)/base_MVA;

    df_dP1_temp = transpose(jacobian(f(u_r{:}), PG_i));
    dfdP1 = subs(df_dP1_temp, PG_i, P1);
    dP1_dx_temp = jacobian(P1_temp, x);
    dP1dx = subs(dP1_dx_temp, x, xVals);

    df_dQ1_temp = transpose(jacobian(f(u_r{:}), QG_i));
    dfdQ1 = subs(df_dQ1_temp, QG_i, Q1_Val);
    dQ1_dx_temp = jacobian(Q1_temp, x(14:end));
    dQ1dx = subs(dQ1_dx_temp, x, xVals);

    dfdx = dfdP1*dP1dx;

    if limit == 1
        if iter > 1
            dfdx = dfdx + transpose(dfdxP12);
        end
    end

    del_C = dfdu - transpose(dgdu)*inv(transpose(dgdx))*transpose(dfdx);
    corr = r*del_C;
    u_x = u_x - corr;
    cost = subs(f(u_x), PG_i, P1);

    iter = iter + 1
    if iter > 50
        break;
    end
end

P2 = double(u_x);
P12_x = abs(abs(V(2)*V(1)*Y(1, 2))*cos(-T(2) - angle(Y(1, 2))) - real(Y(1,
2))*(V(1))^2);
P12_x = P12_x*base_MVA
if limit == 1
    cost = subs(f(P2), [PG_i P12], [P1 P12_x]);
else
    cost = subs(f(P2), PG_i, P1);
end
end
```

```
% Pinj_calc.m

function P_inj_temp = Pinj_calc(x, V, Y)
P_inj_temp = real(Y(1, 1))*abs(V(1)^2);
listOfNeighbours = [2,5];
for k = listOfNeighbours
    if k == 5
        Vk = x(15);
    else
        Vk = V(k);
    end
    P_inj_temp = P_inj_temp + abs( Y(1, k)*Vk* V(1) )*cos( angle(Y(1, k)) +
x(k-1));
end
P_inj_temp;
end
```