Assignment 2

Submitted by Athul Jose P
WSU ID 011867566

Executive Summary

Continuation Power Flow

Steps in main.m function

1. Initializing 14 bus and importing data
2. Making Ybus by calling y_bus_calc.m with taps
3. Calculating the scheduled active power (P) and reactive power (Q)
4. Finding bus types and assigning to vectors
5. Doing Continuation Power Flow by calling CPF.m

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 CPF.m function

1. Initializing Voltage magnitude and angles
2. Initializing Continuation Parameters
3. Finding Initial Values by calling Newton Raphson Function (NR.m)
4. Phase 1: Continuation Parameter as Power (lambda)
    a. Predictor step with increasing power
    b. Corrector step with Newton Raphson method
    c. Looping until NR diverges
5. Phase 2: Continuation Parameter as Voltage
    a. Predictor step with decreasing voltage
    b. Corrector step with augmented Jacobian
    c. Looping until some percentage (75%) of lambda
6. Phase 3: Continuation Parameter as Power (lambda)
    a. Predictor step with decreasing power
    b. Corrector step with Newton Raphson method
    c. Looping until lambda = 0
7. Plotting the PV curve

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

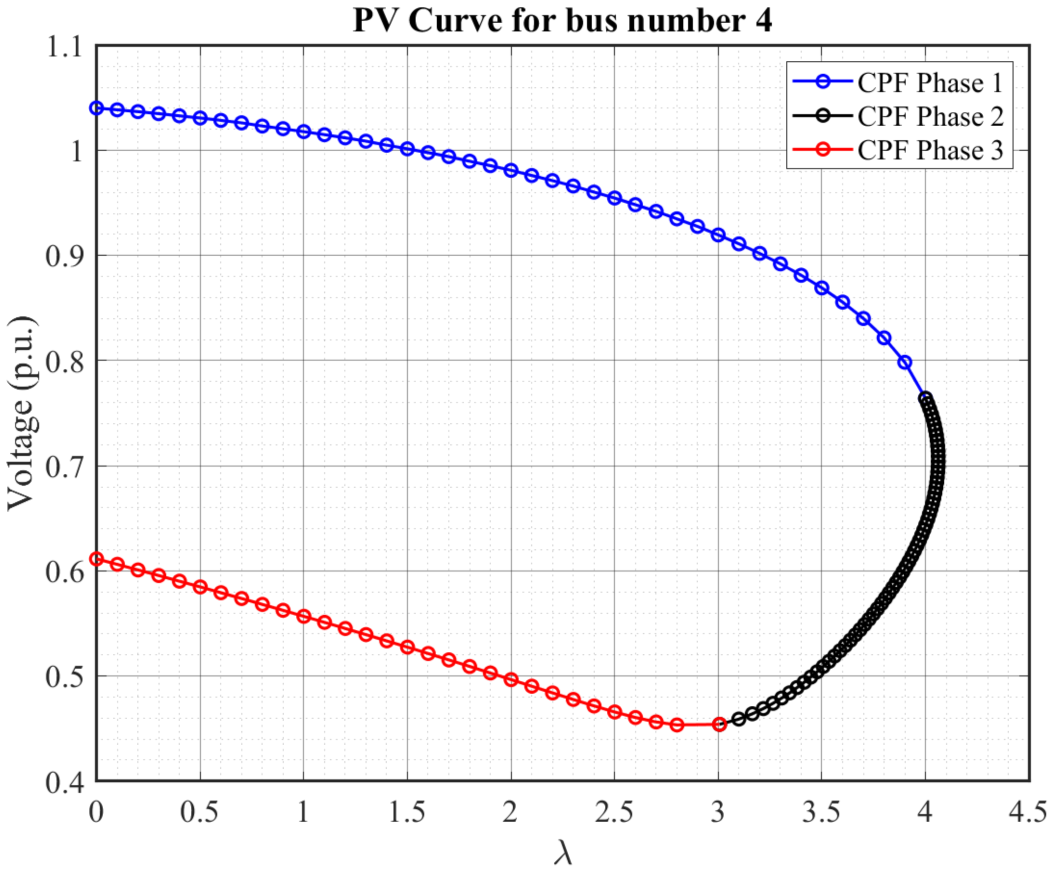
Steps in fwd_bwd.m function

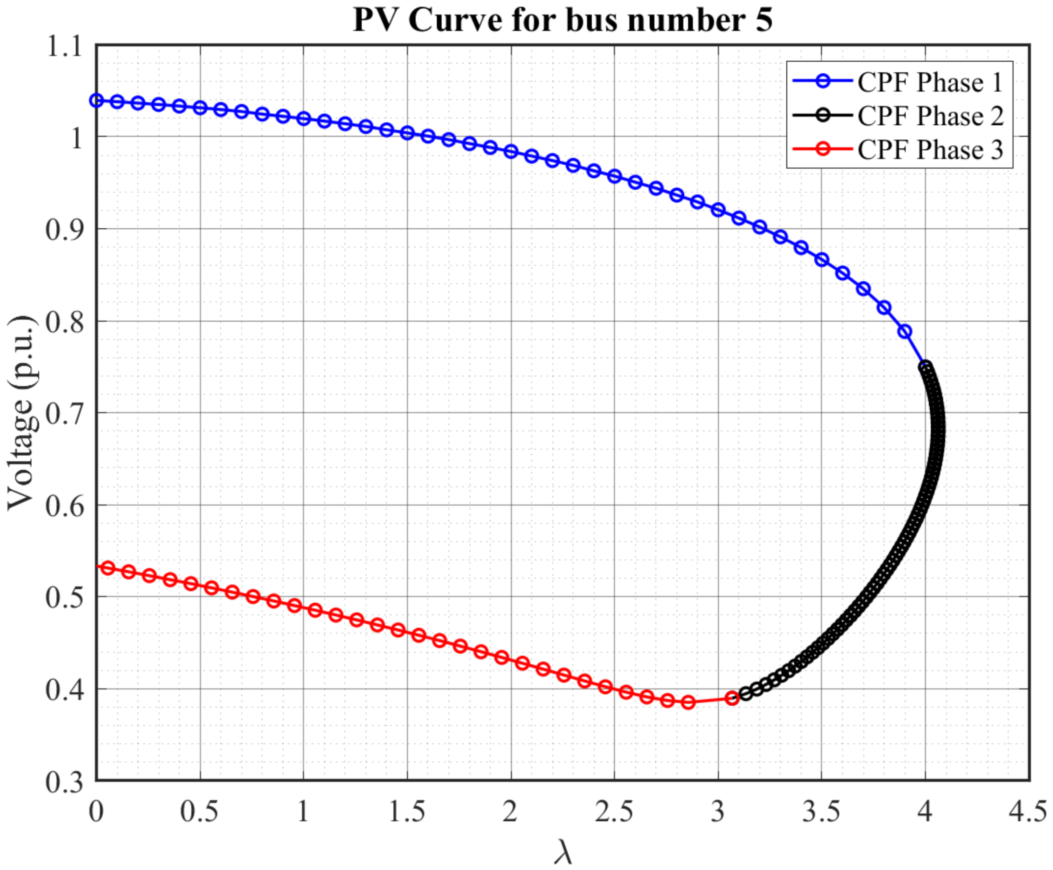
1. Calling LU.m for calculating Lower and Upper elements
2. Doing the backward substitution
3. Doing the forward substitution

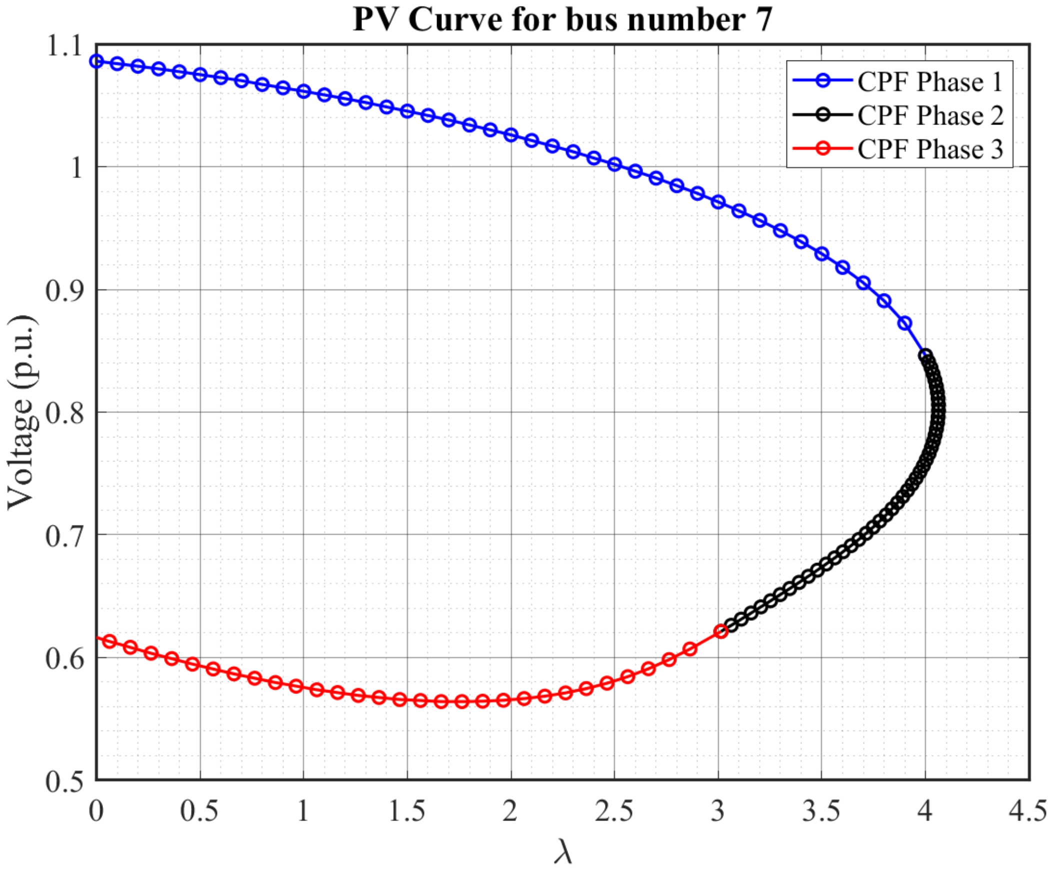
Steps in LU.m function

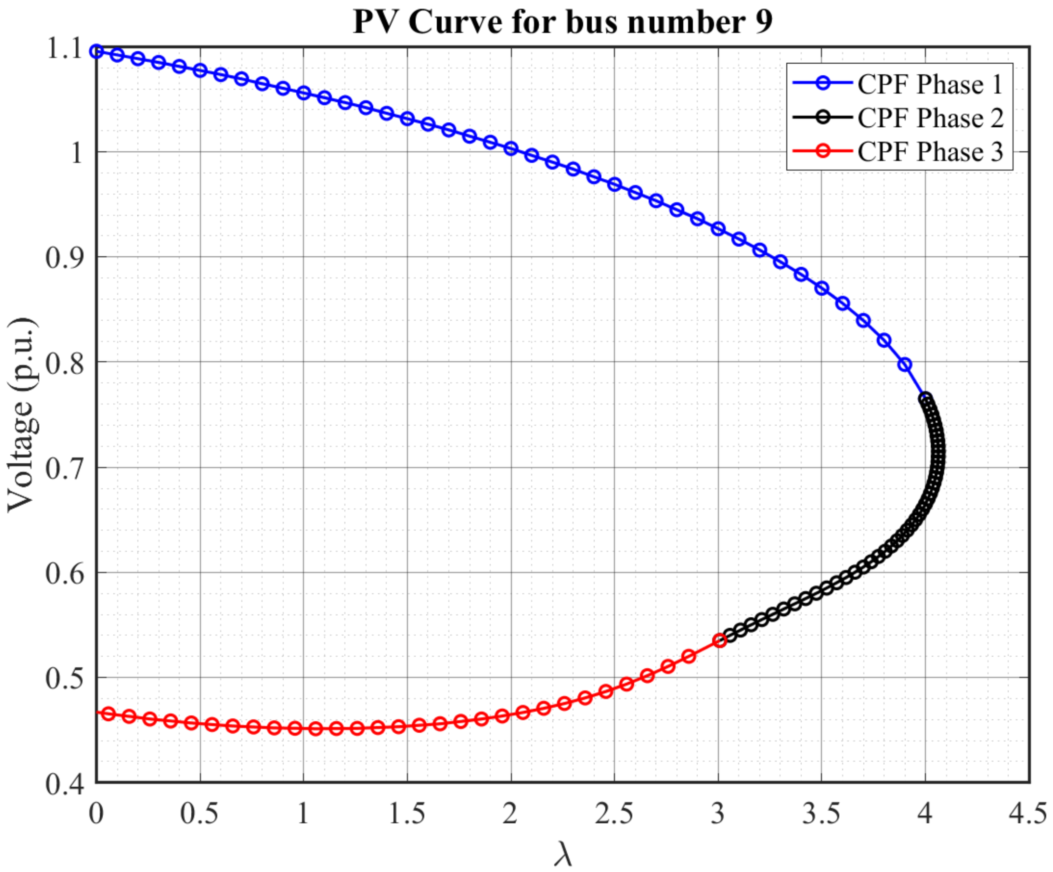
1. Making the Q matrix
2. Dividing it into L & U matrices

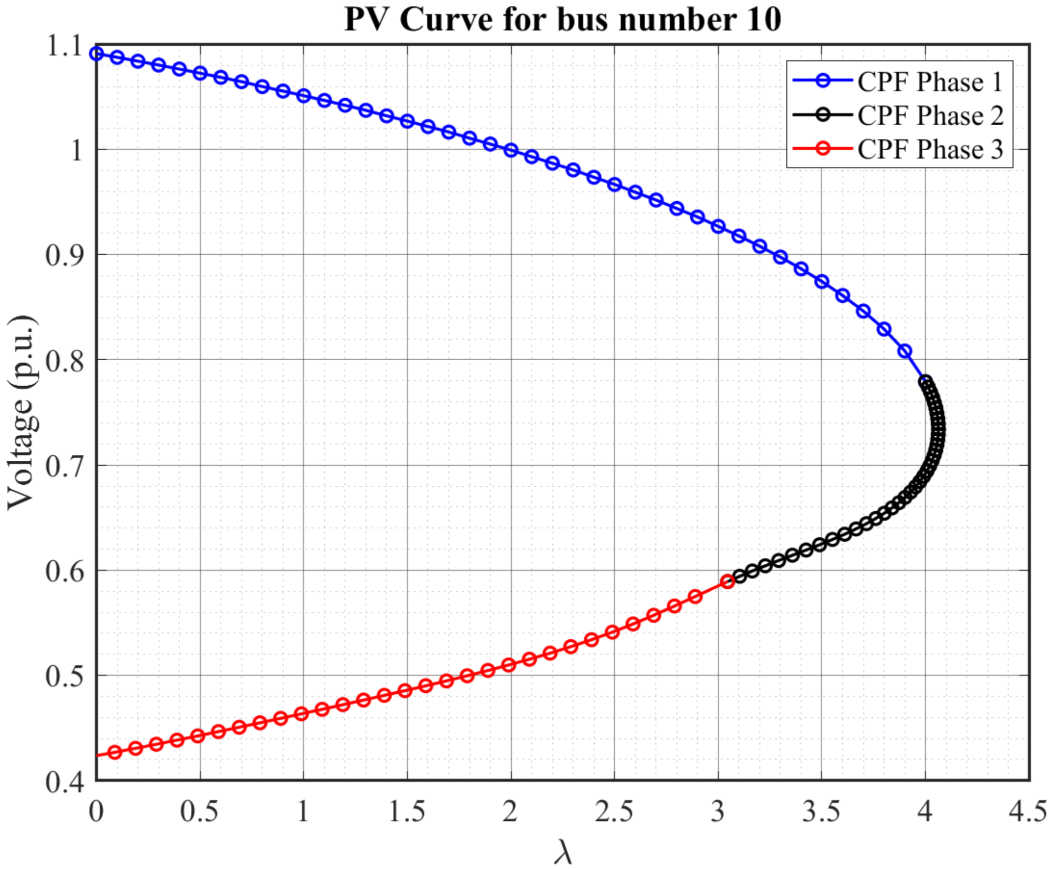
Results

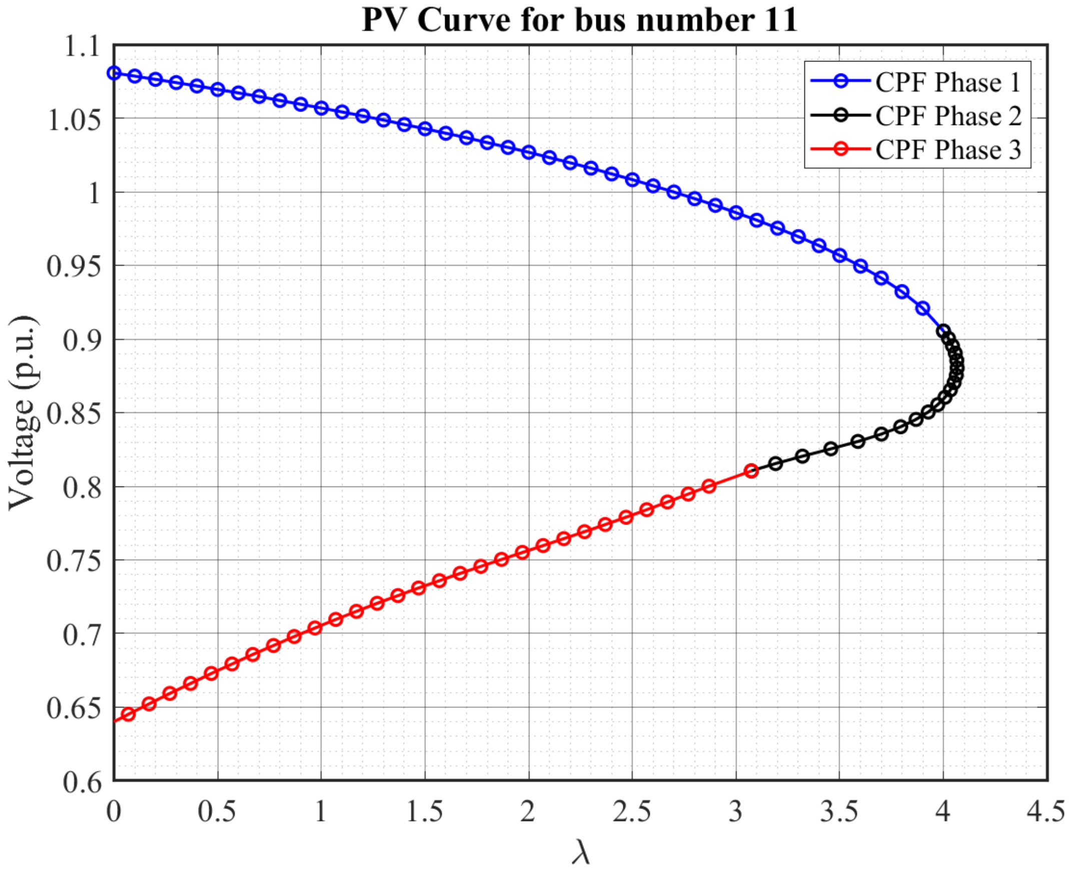


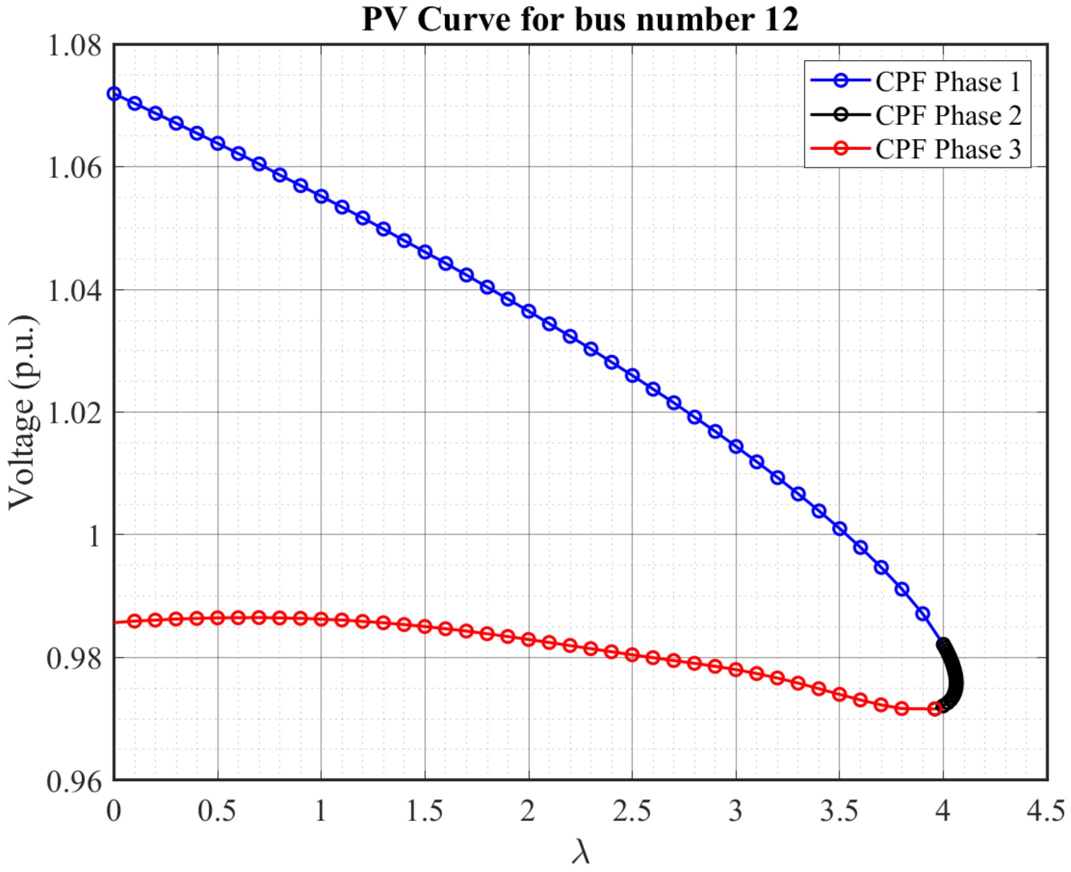

PV Curve for bus number 5
CPF Phase 1
CPF Phase 2
CPF Phase 3
Voltage (p.u.)
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
λ

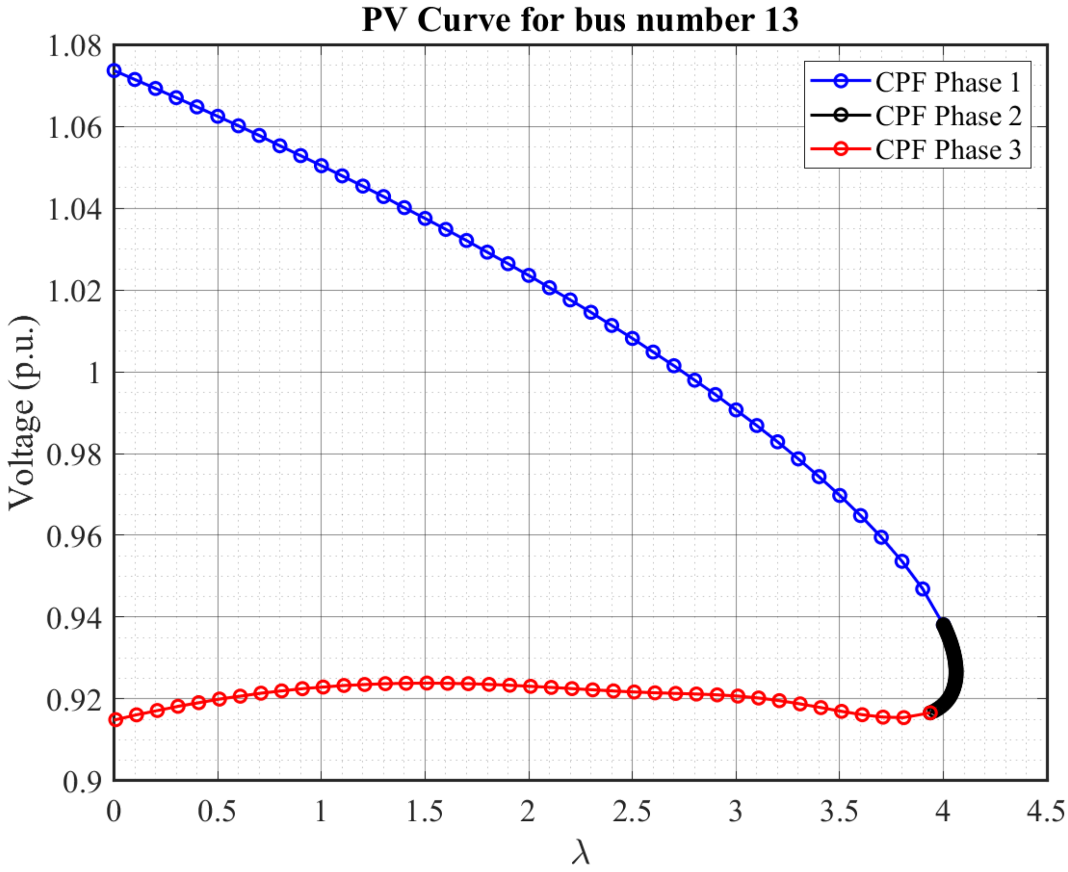

PV Curve for bus number 7
CPF Phase 1
CPF Phase 2
CPF Phase 3
Voltage (p.u.)
1.1
1
0.9
0.8
0.7
0.6
0.5
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
λ

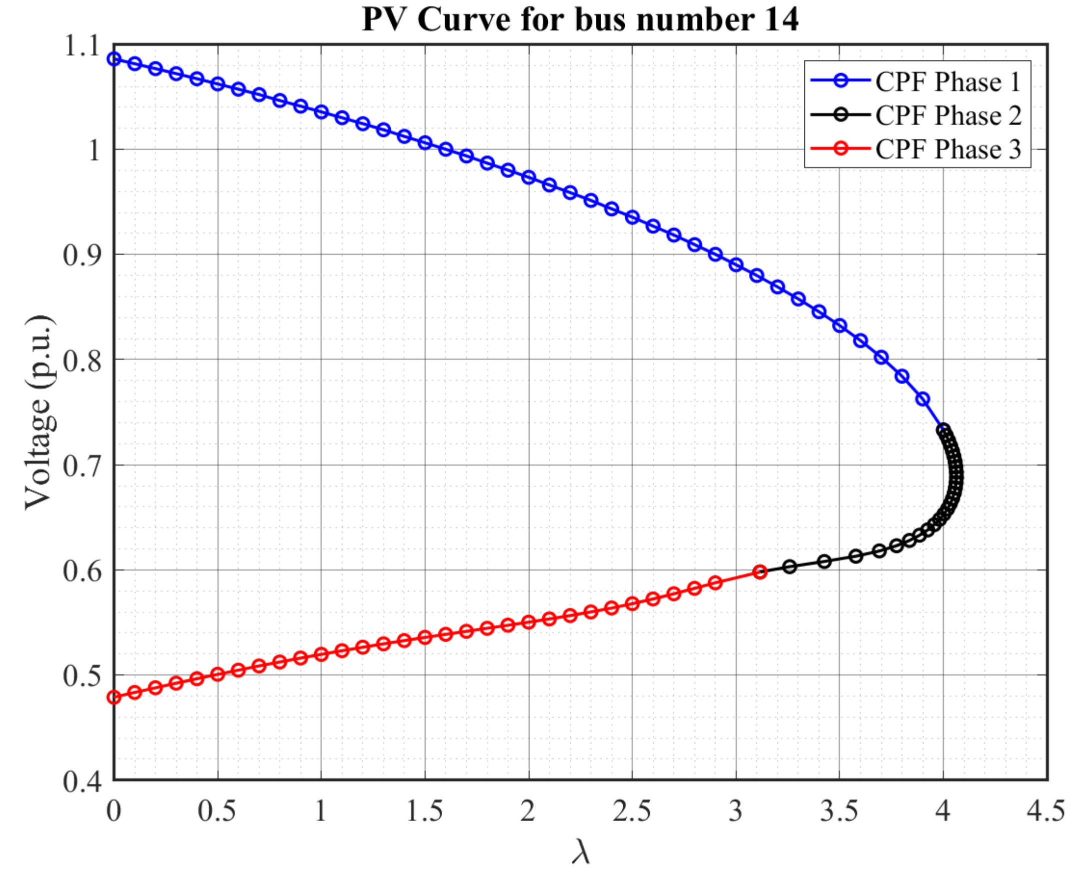

PV Curve for bus number 9
Voltage (p.u.)
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
λ
CPF Phase 1
CPF Phase 2
CPF Phase 3


PV Curve for bus number 10
Voltage (p.u.)
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
λ
CPF Phase 1
CPF Phase 2
CPF Phase 3


PV Curve for bus number 11
Voltage (p.u.)
λ
CPF Phase 1
CPF Phase 2
CPF Phase 3
1.1
1.05
1
0.95
0.9
0.85
0.8
0.75
0.7
0.65
0.6
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5


PV Curve for bus number 12
Voltage (p.u.)
λ
CPF Phase 1
CPF Phase 2
CPF Phase 3
1.08
1.06
1.04
1.02
1
0.98
0.96
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5


PV Curve for bus number 13
Voltage (p.u.)
1.08
1.06
1.04
1.02
1
0.98
0.96
0.94
0.92
0.9
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
λ
CPF Phase 1
CPF Phase 2
CPF Phase 3


PV Curve for bus number 14
Voltage (p.u.)
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
λ
CPF Phase 1
CPF Phase 2
CPF Phase 3

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;

% Finding bus types
sl_i = find(bus_data(:,3) == 3);
pv_i = find(bus_data(:,3) == 2);
pq_i = find(bus_data(:,3) == 0);
n_pv = length(pv_i);
n_pq = length(pq_i);

% Continuation Power Flow for each load bus
for i = 1:n_pq
    n = pq_i(i);
    CPF(n,n_bus,bus_data,P_inj,Q_inj,Y,sl_i,pq_i,n_pq);
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
```

```
% CPF.m

function CPF(n,n_bus,bus_data,P_inj,Q_inj,Y,sl_i,pq_i,n_pq)

% Initializing Voltage magnitude and angles
V = bus_data(:,11);
V(find(V(:)==0)) = 1;
T = zeros(n_bus,1);

% Initializing Continuation Parameters
P_k = P_inj;
P_k(sl_i) = [];
K = [P_k;Q_inj(pq_i)];
ek1 = [zeros(1,n_bus-1+n_pq) 1];
ek2 = zeros(1,n_bus+n_pq);
ek2_i = find(pq_i == n);
ek2(n_bus-1+ek2_i) = -1;
ek3 = -ek1;
sigma1 = 0.1;
sigma2 = 0.005;
lambda = 0;
i = 1;

% Phase 1: Power as Continuation Parameter
% Initial value from Newton Raphson Method
[V_data,T_data,T1,dvrg] =
NR(bus_data,V,T,P_inj*lambda,Q_inj*lambda,n_bus,Y,n_pq,pq_i);
V = V_data(:,size(V_data,2));
T = T_data(:,size(T_data,2));
y(i) = V(n);
x(i) = lambda;
i = i+1;
dvrg = 0;
% Iteration for corrector predictor until divergence
while(dvrg == 0 & i < 100)

    % Predictor
    theta = T(2:n_bus);
    voltage = V(pq_i);
    vec = [theta; voltage; lambda];
    J = J_calc(bus_data,V,T,Y,n_bus,n_pq,pq_i);
    pre = vec + sigma1*((fwd_bwd([J -K; ek1],ek1))');
    T = [0; pre(1:n_bus-1)];
    for j = 1:n_pq
        V(pq_i(j)) = pre(n_bus+j-1);
    end
    lambda = pre(end);

    % Corrector
    [V_data,T_data,T1,dvrg] =
NR(bus_data,V,T,P_inj*lambda,Q_inj*lambda,n_bus,Y,n_pq,pq_i);
    if dvrg == 1
        V = prev_V;
        T = prev_T;
        lambda = prev_lambda;
        ph1 = i-1;
    else
        V = V_data(:,size(V_data,2));
        T = T_data(:,size(T_data,2));
        prev_V = V;
        prev_T = T;
        prev_lambda = lambda;
        y(i) = V(n);
        x(i) = lambda;
        i = i+1;
    end
end

% Phase 2: Voltage as Continuation Parameter
f_stop = 0.75; % stop factor for second phase
% Increased sigma and stop factor for bus 12 & 13
if (n == 12 | n ==13)
    f_stop = 0.98
    sigma2 = 0.0005
end
% Iteration for corrector predictor until some percentage of lambda
while(lambda > f_stop*prev_lambda & i < 200)

    % Predictor
    theta = T(2:n_bus);
    voltage = V(pq_i);
    vec = [theta; voltage; lambda];
    J = J_calc(bus_data,V,T,Y,n_bus,n_pq,pq_i);
    pre = vec + sigma2*((fwd_bwd([J -K; ek2],ek1))');
    T = [0; pre(1:n_bus-1)];
    for j = 1:n_pq
        V(pq_i(j)) = pre(n_bus+j-1);
    end
    lambda = pre(end);

    % Corrector
    J = J_calc(bus_data,V,T,Y,n_bus,n_pq,pq_i);
    [del_P, del_Q] =
dpdq_calc(bus_data,V,T,P_inj*lambda,Q_inj*lambda,n_bus,Y);
    corr = (fwd_bwd([J -lambda*K; ek2],[del_P del_Q 0]'))';
    lambda = lambda + corr(end);
    if lambda <= f_stop*prev_lambda
        V = prev_V;
        T = prev_T;
        ph2 = i-1;
        break;
    else
        T = T + [0; corr(1:n_bus-1)];
        for j = 1:n_pq
            V(pq_i(j)) = V(pq_i(j)) + corr(n_bus+j-1);
        end
        prev_V = V;
        prev_T = T;
        y(i) = V(n);
        x(i) = lambda;
        i = i+1;
    end
end

% Phase 3: Power as Continuation Parameter
% Iteration for corrector predictor until negative lambda
while(lambda >= 0 & i < 250)

    % Predictor
    i;
    theta = T(2:n_bus);
    voltage = V(pq_i);
    vec = [theta; voltage; lambda];
    J = J_calc(bus_data,V,T,Y,n_bus,n_pq,pq_i);
    pre = vec + sigma1*((fwd_bwd([J -K; ek3],ek1))');
    T = [0; pre(1:n_bus-1)];
    for j = 1:n_pq
        V(pq_i(j)) = pre(n_bus+j-1);
    end
    lambda = pre(end);

    % Corrector
    [V_data,T_data,T1,dvrg] =
NR(bus_data,V,T,P_inj*lambda,Q_inj*lambda,n_bus,Y,n_pq,pq_i);
    V = V_data(:,size(V_data,2));
    T = T_data(:,size(T_data,2));
    y(i) = V(n);
    x(i) = lambda;
    i = i+1;
end

% plotting CPF curve
x_label = '\lambda'; % x axis label
y_label = 'Voltage (p.u.)'; % y axis label
legend_name = {'CPF Phase 1','CPF Phase 2','CPF Phase 3'}; % legend names
title_name = ['PV Curve for bus number ' num2str(n)]; % title name
figure('Renderer', 'painters', 'Position', [10 10 800 600])
plot(x(1:ph1),y(1:ph1),'-ob','LineWidth',1.5)
hold on
plot(x(ph1:ph2),y(ph1:ph2),'-ok','LineWidth',1.5)
plot(x(ph2:end),y(ph2:end),'-or','LineWidth',1.5)
xlabel(x_label,'FontSize',18,'FontName','Times New Roman')
ylabel(y_label,'FontSize',18,'FontName','Times New Roman')
legend (legend_name,'Location','northeast')
set(gca,'fontsize',16,'Fontname','Times New Roman','GridAlpha',0.5)
ax = gca
xlim([0 4.5])
title(title_name)
ax.XRuler.Axle.LineWidth = 1.5;
ax.YRuler.Axle.LineWidth = 1.5;
grid
grid minor
saveas(gca,[title_name '.png'])
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)))))/V(i);
        else
            J2(i-1,j) = V(i)*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)))))/V(n1);
        else
            J4(i,j) = -V(n1)*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
```

```
% FD.m
function [V_data,T_data] = FD(bus_data,V,T,P_inj,Q_inj,n_bus,Y,n_pq,pq_i)
% Initializing index
B = imag(Y);
B_T =  - B(2:n_bus,2:n_bus);
```

```
B_V = - B(pq_i,pq_i);
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);
    P_T = del_P'./V(2:n_bus);
    d_T = fwd_bwd(B_T,P_T);
    Q_V = del_Q'./V(pq_i);
    d_V = fwd_bwd(B_V,Q_V);
    del_T = [0 d_T]'; % angle calculation
    for j = 1:n_pq
        del_V(pq_i(j)) = d_V(j); % magnitude calculation
    end
    Tol = max(abs([del_T; del_V]));
end
end
```