**EE 521 | Fall 2022**

**Paper**

**Assignment 01**

**Submitted by**

**Sajjad Uddin Mahmud**

**WSU ID: 011789534**

**Logo, company name

Description automatically generated**

Summary:

The power flow of a 14-bus power system has been investigated in this report. The purpose of a power flow study is to solve the power flow equations and determine the amplitudes and angles of the network's bus voltages. The problem of power flow in a power system network can be solved in a variety of ways. For this power flow calculation, the Newton-Raphson method and the Fast Decoupled method are utilized.

After reading the bus and branch data of the IEEE 14-bus test system, the YBus matrix has been formed using the branch resistance, branch reactance, bus shunt conductance, bus shunt susceptance, line charge capacitance and transformer turns ratio (taps). Then the real and reactive powers of each bus are calculated using power flow equations given in (1).

……………….. (1)

The results that is the voltage magnitudes and angles, P and Q for every bus are generated in three different scenarios:

1. Without taps – Full Newton-Raphson
2. With taps – Full Newton-Raphson
3. With taps – Fast Decoupled

In my code, I have created options to choose between tap and no-tap as well as full NR and fast decoupled approach.

All the results have been given in Appendix 1 and the code has been given in Appendix 2.

The YBus matrix computation is the sole variation between the first two scenarios. The transformer turns ratio (taps) was not taken into account while calculating the YBus matrix in the first scenario. The components of the Jacobian matrix are generated for both scenarios 1 and 2 using the real and reactive power values that were calculated, the voltage magnitudes and angles that were assumed, and the bus admittance matrix's elementsSlack buses' real and reactive power equations are unknown, hence they are not included in the Jacobian matrix. The Jacobian matrix eliminates the relevant rows and columns for voltage control buses when voltage values are maintained constant and the equation for reactive power is not known i.e., PV buses. The network consists of nine load buses, four generator buses, and one slack bus. Nine load buses, four voltage-controlled buses, and one slack bus make up the network. As a result, after computing the Jacobian matrix, the appropriate rows and columns are eliminated for voltage-regulated buses. Hence, the size of the Jacobian matrix for IEEE 14 bus is 22 by 22.

The liner equation given below is solved to get the initial mismatch of voltage angles and magnitudes after computing the Jacobian matrix.

……………….. (2)

To solve this equation, I use Dolittle’s algorithm. The basis of Dolittle’s Algorithm is the factorization of the coefficient matrix into two separate matrices. These matrices come in upper-triangular and lower-triangular forms which can be formed by elementary row operation. Then the solution of the unknown variables may be quickly calculated using just the forward and backward substitution method if the coefficient matrix can be factored into an upper triangular matrix and a lower triangular matrix. After that the mismatches are obtained from the calculation of and . The entire process is repeated if the mismatches exceed the threshold of 0.01. Convergence is attained once and are both within tolerance at the end of this operation.

For the Fast Decoupled scenario, the change of real power with respect to voltage magnitude and change of reactive power with respect to voltage angles are neglected, therefore the submatrices and becomes zero.

The iteration that my code took is given below:

|  |  |
| --- | --- |
| Scenarios | Iteration |
| Without taps – Full Newton-Raphson | 3 |
| With taps – Full Newton-Raphson | 3 |
| With taps – Fast Decoupled | 14 |

Appendix 01

# Results: PV Curve for Each Load Bus

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generatedChart

Description automatically generatedChart

Description automatically generatedChart, line chart

Description automatically generatedChart, line chart

Description automatically generatedChart, line chart, scatter chart

Description automatically generatedChart

Description automatically generatedChart

Description automatically generated

Appendix 02

# Code: Platform MATLAB

## Main Code :

%% Assignment 03 - Continuation Power Flow - Sajjad Uddin Mahmud - Fall 2022 - WSU

%% Basic Initialization

clc;

clear all;

close all;

%% Reading From Bus Data

%% Bus Number

All\_Bus\_Number = xlsread('IEEE14\_Formatted.xlsx','A3:A16'); % Reading All Bus ID Data

Total\_Bus = length(All\_Bus\_Number); % Calculating Total Bus Number

%% Bus Type

All\_Bus\_Type = xlsread('IEEE14\_Formatted.xlsx','G3:G16'); % Reading All Bus Type Data

PQ\_Bus\_Type = 0;

PQ\_Bus\_Type\_1 = 1;

PV\_Bus\_Type = 2;

Slack\_Bus\_Type = 3;

Bus = All\_Bus\_Number(find(All\_Bus\_Type ~= Slack\_Bus\_Type)); % Bus Type Data Except the Slack Bus

Bus\_Type = All\_Bus\_Type(Bus); % Bus Type Data Except the Slack Bus

%% Bus Information

% Slack\_Bus\_Number = 1

Base\_MVA = 100;

V\_Desired = xlsread('IEEE14\_Formatted.xlsx','O3:O16'); % Given Initial Voltage

P\_Load\_All = xlsread('IEEE14\_Formatted.xlsx','J3:J16')/Base\_MVA; % Load MW pu

P\_Load = P\_Load\_All(find(All\_Bus\_Type~=Slack\_Bus\_Type ));

Q\_Load\_All = xlsread('IEEE14\_Formatted.xlsx','K3:K16')/Base\_MVA; % Load MVAR pu

Q\_Load = Q\_Load\_All(find(All\_Bus\_Type~=Slack\_Bus\_Type ));

P\_Gen\_All = xlsread('IEEE14\_Formatted.xlsx','L3:L16')/Base\_MVA; % Generator MW pu

P\_Gen = P\_Gen\_All(find(All\_Bus\_Type~=Slack\_Bus\_Type ));

Q\_Gen\_All = xlsread('IEEE14\_Formatted.xlsx','M3:M16')/Base\_MVA; % Generator MVAR pu

Q\_Gen = Q\_Gen\_All(find(All\_Bus\_Type~=Slack\_Bus\_Type ));

%% Reading from Branch Data

%% Branch Number

To\_Bus = xlsread('IEEE14\_Formatted.xlsx','A19:A38');

From\_Bus = xlsread('IEEE14\_Formatted.xlsx','B19:B38');

Total\_Bus = max(max(To\_Bus),max(From\_Bus));

%% Bus Shunt Conductance and Shunt Susceptance

G\_Shunt\_Bus = xlsread('IEEE14\_Formatted.xlsx','R3:R16');

B\_Shunt\_Bus = xlsread('IEEE14\_Formatted.xlsx','S3:S16');

%% Calculating Bus Shunt Admittance

Y\_Shunt\_Bus = G\_Shunt\_Bus + j.\*B\_Shunt\_Bus;

%% Branch Resistance Per Unit

R\_Branch = xlsread('IEEE14\_Formatted.xlsx','G19:G38');

%% Branch Reactance Per Unit

X\_Branch = xlsread('IEEE14\_Formatted.xlsx','H19:H38');

%% Line Charging B Per Unit

B\_Branch = xlsread('IEEE14\_Formatted.xlsx','I19:I38');

%% Transformer Turns Ratio

XFR\_TurnRatio = xlsread('IEEE14\_Formatted.xlsx','O19:O38');

%% Calculating Branch Impedence and Admittance

for i=1:length(From\_Bus)

Z\_Branch(i) = R\_Branch(i) + j\*X\_Branch(i); % Per Unit Impedance

Y\_Branch(i) = 1/Z\_Branch(i); % Per Unit Admittance

end

%% Tap Consideration

Tap\_Consideration = 1; % 0 = Without Taps, 1 = With Taps

if (Tap\_Consideration == 0)

for i = 1:length(XFR\_TurnRatio)

XFR\_TurnRatio(i) = 0; % If We Do Not Consider Tap, All the Turn Ratio of Transformer are 0

end

end

%% Calculating Y Bus Matrix:

% Initialization

Y\_Bus = zeros(Total\_Bus,Total\_Bus);

% LOOP: Computing Off-Diagonal Elements

for i=1:length(Y\_Branch)

if (XFR\_TurnRatio(i)==0)

Y\_Bus(To\_Bus(i),From\_Bus(i)) = - Y\_Branch(i);

Y\_Bus(From\_Bus(i),To\_Bus(i)) = - Y\_Branch(i);

Y\_Bus\_Diag(To\_Bus(i),From\_Bus(i)) = - Y\_Branch(i);

Y\_Bus\_Diag(From\_Bus(i),To\_Bus(i)) = - Y\_Branch(i);

else

T = (1/(XFR\_TurnRatio(i)));

Y\_Bus(To\_Bus(i),From\_Bus(i)) = - Y\_Branch(i) \* (T);

Y\_Bus(From\_Bus(i),To\_Bus(i)) = - Y\_Branch(i) \* (T);

Y\_Bus\_Diag(To\_Bus(i),From\_Bus(i)) = - Y\_Branch(i);

Y\_Bus\_Diag(From\_Bus(i),To\_Bus(i)) = - Y\_Branch(i) \* (T^2);

end

end

% LOOP: Computing Diagonal Elements

Y\_Bus\_Sum = sum(Y\_Bus\_Diag);

for i=1:Total\_Bus

Y\_Bus(i,i) = - Y\_Bus\_Sum(i) + Y\_Shunt\_Bus(i); % Adding Shunt Capacitance

end

% LOOP: Adding Line Charaging Capacitance

for i=1:length(To\_Bus)

Y\_Bus(To\_Bus(i),To\_Bus(i)) = Y\_Bus(To\_Bus(i),To\_Bus(i)) + j \* (B\_Branch(i) / 2);

Y\_Bus(From\_Bus(i),From\_Bus(i)) = Y\_Bus(From\_Bus(i),From\_Bus(i)) + j \* (B\_Branch(i) / 2);

end

% Converting Y Bus Data into Polar Form

Rho = abs(Y\_Bus); % Magnitude of Y Bus Entries

Theta = angle(Y\_Bus); % Angle of Y Bus Entries in radian

B = imag(Y\_Bus); % Imaginary Part of Y Bus Entries

G = real(Y\_Bus); % Real Part of Y Bus Entries

% End of Y Bus Formation. Y Bus is Ready

%% Getting K Vector

PQ\_Bus\_Number = All\_Bus\_Number(find((All\_Bus\_Type ~= Slack\_Bus\_Type) & (All\_Bus\_Type ~= PV\_Bus\_Type)));

PQ\_Bus\_Title = All\_Bus\_Number(find((All\_Bus\_Type ~= Slack\_Bus\_Type) & (All\_Bus\_Type ~= PV\_Bus\_Type)));

P\_Scheduled = P\_Gen\_All - P\_Load\_All;

Q\_Scheduled = Q\_Gen\_All-Q\_Load\_All;

P\_K = P\_Scheduled(find(All\_Bus\_Type ~= Slack\_Bus\_Type));

Q\_K = Q\_Scheduled(find((All\_Bus\_Type~=Slack\_Bus\_Type) & (All\_Bus\_Type~=PV\_Bus\_Type)));

K\_Vector = [P\_K ; Q\_K];

for Parameter = 1:length(PQ\_Bus\_Title)

%% Lambda and Sigma Initialization

Lambda = 0;

Sigma = 0.1;

Bus\_Nose = PQ\_Bus\_Title(Parameter);

% Initial Voltage Magnitude and Angle (Flat Start)

V = ones(1,length(All\_Bus\_Number));

Delta\_in\_Rad = zeros(1,length(All\_Bus\_Number));

%% Newton-Raphson Power Flow Solution

Tolerance = 0.01;

% Function Calling: Power Flow Using Newton-Raphson Method

[V, Delta\_in\_Rad, Iteration] = Newton\_Raphson\_Function(Y\_Bus, V, Delta\_in\_Rad, Lambda, P\_Gen, P\_Load, Q\_Gen, Q\_Load, All\_Bus\_Number, All\_Bus\_Type, Tolerance);

Nose = 0;

while 1

Nose = Nose + 1;

V\_X = V;

Delta\_X = Delta\_in\_Rad;

Lambda\_X = Lambda;

%% Upper Part

% Predictor

[J] = Jacobian\_Function(V, Delta\_in\_Rad, Y\_Bus, All\_Bus\_Type);

Delta\_Base = Delta\_in\_Rad(find(All\_Bus\_Type ~= Slack\_Bus\_Type ));

V\_Base = V(find((All\_Bus\_Type ~= Slack\_Bus\_Type) & (All\_Bus\_Type ~= PV\_Bus\_Type)));

% Initializing Extra Equation for Lambda

E\_K\_Vector = zeros(1, length(K\_Vector)+1);

E\_K\_Vector(end) = 1; % Setting Last Value of Extra Vector as 1

% Initializing Predictor B Vector

Predictor\_B\_Vector = zeros(length(E\_K\_Vector),1);

Predictor\_B\_Vector(end) = 1;

% Calculating Tangent Vector Using LU Factorization Dolittle's Method

Delta\_V\_Lamda\_Vector = [transpose(Delta\_Base); transpose(V\_Base); Lambda];

J\_Predictor = [J K\_Vector; E\_K\_Vector];

[Tangent\_Vector] = LU\_Factorization\_Dolittle\_Function(J\_Predictor, Predictor\_B\_Vector);

% Computing Predicted Values

Delta\_V\_Lamda\_Vector\_Predicted = Delta\_V\_Lamda\_Vector + (Sigma \* Tangent\_Vector);

% Predicted Lambda

Lambda = Delta\_V\_Lamda\_Vector\_Predicted(end);

% LOOP: Predicted Voltages and Angles

for i=1:(length(Delta\_V\_Lamda\_Vector\_Predicted) - 1)

if (i<=length(Delta\_Base))

Delta\_Predicted(i) = Delta\_V\_Lamda\_Vector\_Predicted(i);

else

V\_Predicted(i-(length(Delta\_Base))) = Delta\_V\_Lamda\_Vector\_Predicted(i);

end

end

% Preparing for Correction Stage

V\_Index = (find((All\_Bus\_Type == PQ\_Bus\_Type) | (All\_Bus\_Type == PQ\_Bus\_Type\_1)));

Delta\_Index = find(All\_Bus\_Type ~= Slack\_Bus\_Type );

for i=1:length(Delta\_Index)

Delta\_in\_Rad(Delta\_Index(i)) = Delta\_Predicted(i);

end

for i=1:length(V\_Index)

V(V\_Index(i)) = V\_Predicted(i);

end

% Corrector

% Function Calling: Power Flow Using Newton-Raphson Method

[V, Delta\_in\_Rad, Iteration] = Newton\_Raphson\_Function(Y\_Bus, V, Delta\_in\_Rad, Lambda, P\_Gen, P\_Load, Q\_Gen, Q\_Load, All\_Bus\_Number, All\_Bus\_Type, Tolerance);

if Iteration >= 25

V = V\_X;

Delta\_in\_Rad = Delta\_X;

Lambda = Lambda\_X;

break;

end

V\_PV\_Upper(Nose) = V(Bus\_Nose);

Delta\_PV\_Upper(Nose) = Delta\_in\_Rad(Bus\_Nose);

Lamda\_PV\_Upper(Nose) = Lambda;

end

%% Critial Point

Sigma = 0.025; % Changing Sigma to 0.025 Because Voltage Changes at a Slower Rate than Power

Critical = 0;

while 1

Critical = Critical + 1;

V\_X = V;

Delta\_X = Delta\_in\_Rad;

Lambda\_X = Lambda;

% Predictor Stage

% Function Calling: Jacobian

[J] = Jacobian(V, Delta\_in\_Rad, Y\_Bus, All\_Bus\_Type);

Delta\_Base = Delta\_in\_Rad(find(All\_Bus\_Type ~= Slack\_Bus\_Type ));

V\_Base = V(find((All\_Bus\_Type ~= Slack\_Bus\_Type) & (All\_Bus\_Type ~= PV\_Bus\_Type)));

% Initializing Extra Equation for Lambda

E\_K\_Vector = zeros(1, length(K\_Vector) + 1);

E\_K\_Vector(Total\_Bus - 1 + find(PQ\_Bus\_Number == Bus\_Nose)) = -1;

% Initializing Predictor B Vector

Predictor\_B\_Vector = zeros(length(E\_K\_Vector),1);

Predictor\_B\_Vector(end) = 1;

% Calculating Tangent Vector Using LU Factorization Dolittle's Method

Delta\_V\_Lamda\_Vector = [transpose(Delta\_Base); transpose(V\_Base); Lambda];

J\_Predictor = [-J K\_Vector; E\_K\_Vector];

[Tangent\_Vector] = LU\_Factorization\_Dolittle\_Function(J\_Predictor, Predictor\_B\_Vector);

% Computing Predicted Values

Delta\_V\_Lamda\_Vector\_Predicted = Delta\_V\_Lamda\_Vector + (Sigma \* Tangent\_Vector);

% Predicted Lambda

Lambda = Delta\_V\_Lamda\_Vector\_Predicted(end);

% LOOP: Predicted Voltages and Angles

for i=1:(length(Delta\_V\_Lamda\_Vector\_Predicted) - 1)

if (i <= length(Delta\_Base))

Delta\_Predicted(i) = Delta\_V\_Lamda\_Vector\_Predicted(i);

else

V\_Predicted(i-(length(Delta\_Base))) = Delta\_V\_Lamda\_Vector\_Predicted(i);

end

end

% Preparing for Correction Stage

V\_Index = (find((All\_Bus\_Type == PQ\_Bus\_Type) | (All\_Bus\_Type == PQ\_Bus\_Type\_1)));

Delta\_Index = find(All\_Bus\_Type ~= Slack\_Bus\_Type );

for i=1:length(Delta\_Index)

Delta\_in\_Rad(Delta\_Index(i)) = Delta\_Predicted(i);

end

for i=1:length(V\_Index)

V(V\_Index(i)) = V\_Predicted(i);

end

% Corrector Stage

% Initialization

Iteration = 0;

while 1

Iteration = Iteration + 1;

PV = find(Bus\_Type == PV\_Bus\_Type);

% Schedule Real and Reactive Power

P\_Scheduled = transpose(P\_Gen - (Lambda \* P\_Load));

Q\_Scheduled = transpose(Q\_Gen - (Lambda \* Q\_Load));

%% Calculating Real Power

% Initialization

P\_Calculated = zeros(1,length(Bus));

% LOOP: Computing Real Power

for i=1:length(Bus)

for n=1:Total\_Bus

P\_Calculated(i) = P\_Calculated(i) + (abs(abs(Y\_Bus(Bus(i),n)) \* V(Bus(i)) \* V(n))) \* (cos(angle(Y\_Bus(Bus(i),n)) + Delta\_in\_Rad(n) - Delta\_in\_Rad(Bus(i))));

end

end

%% Calculating Reactive Power

% Initialization

Q\_Calculated=zeros(1,length(Bus));

% LOOP: Computing Reactive Power

for i=1:length(Bus)

for n=1:Total\_Bus

Q\_Calculated(i) = Q\_Calculated(i) + (abs(abs(Y\_Bus(Bus(i),n)) \* V(Bus(i)) \* V(n))) \* (sin(angle(Y\_Bus(Bus(i),n)) + Delta\_in\_Rad(n) - Delta\_in\_Rad(Bus(i))));

end

Q\_Calculated(i) = - Q\_Calculated(i);

end

%% Calculating Mismatch

Delta\_P = P\_Scheduled - P\_Calculated;

Delta\_Q = Q\_Scheduled - Q\_Calculated;

Delta\_Q(:,PV)=[];

Delta\_P\_Q\_0 = [transpose(Delta\_P);transpose(Delta\_Q); 0];

% Initializing Extra Equation for Lambda

E\_K\_Vector = zeros(1, length(K\_Vector)+1);

E\_K\_Vector(Total\_Bus - 1 + find(PQ\_Bus\_Number == Bus\_Nose)) = -1;

% Function Calling: Jacobian

[J] = Jacobian\_Function(V,Delta\_in\_Rad, Y\_Bus, All\_Bus\_Type);

J\_Corrected = [-J K\_Vector; E\_K\_Vector];

Mismatch\_V\_Delta\_Lambda = LU\_Factorization\_Dolittle\_Function(J\_Corrected, -Delta\_P\_Q\_0);

Delta\_Lamda = Mismatch\_V\_Delta\_Lambda(end);

Lambda = Lambda + Delta\_Lamda;

for i=1:(length(Mismatch\_V\_Delta\_Lambda) - 1)

if (i <= length(Delta\_Index))

Mismatch\_Delta(i) = Mismatch\_V\_Delta\_Lambda(i);

else

Mismatch\_V(i-(length(Delta\_Index))) = Mismatch\_V\_Delta\_Lambda(i);

end

end

for i=1:length(Delta\_Index)

Delta\_in\_Rad(Delta\_Index(i)) = Mismatch\_Delta(i) + Delta\_in\_Rad(Delta\_Index(i));

end

for i=1:length(V\_Index)

V(V\_Index(i)) = Mismatch\_V(i) + V(V\_Index(i));

end

if (max(abs(Mismatch\_V\_Delta\_Lambda)) < 0.1)

break;

end

if Iteration >= 5

break;

end

end

if Iteration >= 5

V = V\_X;

Delta\_in\_Rad = Delta\_X;

Lambda = Lambda\_X;

break;

end

V\_PV\_Critical(Critical) = V(Bus\_Nose);

Delta\_PV\_Critical(Critical) = Delta\_in\_Rad(Bus\_Nose);

Lamda\_PV\_Critical(Critical) = Lambda;

end

%% Lower Part

Sigma = 0.1;

Lower = 0;

while 1

Lower = Lower + 1;

V\_X = V;

Delta\_X = Delta\_in\_Rad;

Lambda\_X = Lambda;

% Predictor Stage

[J] = Jacobian(V,Delta\_in\_Rad, Y\_Bus, All\_Bus\_Type);

Delta\_Base = Delta\_in\_Rad(find(All\_Bus\_Type ~= Slack\_Bus\_Type ));

V\_Base = V(find((All\_Bus\_Type ~= Slack\_Bus\_Type) & (All\_Bus\_Type ~= PV\_Bus\_Type)));

% Initializing Extra Equation for Lambda

E\_K\_Vector = zeros(1, length(K\_Vector) + 1);

E\_K\_Vector(length(E\_K\_Vector)) = -1;

% Initializing Predictor B Vector

Predictor\_B\_Vector = zeros(length(E\_K\_Vector),1);

Predictor\_B\_Vector(length(E\_K\_Vector)) = 1;

% Calculating Tangent Vector Using LU Factorization Dolittle's Method

Delta\_V\_Lamda\_Vector = [transpose(Delta\_Base); transpose(V\_Base); Lambda];

J\_Predictor = [J K\_Vector; E\_K\_Vector];

[Tangent\_Vector] = LU\_Factorization\_Dolittle\_Function(J\_Predictor, Predictor\_B\_Vector);

Delta\_V\_Lamda\_Vector\_Predicted = Delta\_V\_Lamda\_Vector + (Sigma \* Tangent\_Vector);

% Predicted Lambda

Lambda = Delta\_V\_Lamda\_Vector\_Predicted(end);

% LOOP: Predicted Voltages and Angles

for i=1:(length(Delta\_V\_Lamda\_Vector\_Predicted)-1)

if (i<=length(Delta\_Base))

Delta\_Predicted(i)=Delta\_V\_Lamda\_Vector\_Predicted(i);

else

V\_Predicted(i-(length(Delta\_Base)))=Delta\_V\_Lamda\_Vector\_Predicted(i);

end

end

% Preparing for Correction Stage

V\_Index=(find((All\_Bus\_Type == PQ\_Bus\_Type) | (All\_Bus\_Type == PQ\_Bus\_Type\_1)));

Delta\_Index=find(All\_Bus\_Type ~= Slack\_Bus\_Type );

for i=1:length(Delta\_Index)

Delta\_in\_Rad(Delta\_Index(i)) = Delta\_Predicted(i);

end

for i=1:length(V\_Index)

V(V\_Index(i))=V\_Predicted(i);

end

% Corrector Stage

[V, Delta\_in\_Rad, Iteration] = Newton\_Raphson\_Function(Y\_Bus, V, Delta\_in\_Rad, Lambda, P\_Gen, P\_Load, Q\_Gen, Q\_Load, All\_Bus\_Number, All\_Bus\_Type, Tolerance);

if Lambda <= 0.1

V = V\_X;

Delta\_in\_Rad = Delta\_X;

Lambda = Lambda\_X;

break;

end

V\_PV\_Lower(Lower) = V(Bus\_Nose);

Delta\_PV\_Lower(Lower) = Delta\_in\_Rad(Bus\_Nose);

Lamda\_PV\_Lower(Lower) = Lambda;

end

V\_PV = [V\_PV\_Upper V\_PV\_Critical V\_PV\_Lower];

Lamda\_PV = [Lamda\_PV\_Upper Lamda\_PV\_Critical Lamda\_PV\_Lower];

%% Output

figure (Parameter)

plot (Lamda\_PV, V\_PV,'--o')

hold on;

plot(Lamda\_PV\_Upper, V\_PV\_Upper,'g')

hold on;

plot(Lamda\_PV\_Critical, V\_PV\_Critical,'r')

hold on

plot(Lamda\_PV\_Lower, V\_PV\_Lower,'b')

title(['PV Curve of Bus ', num2str(Bus\_Nose)])

xlabel('Lamda');

ylabel('Voltage Magnitude');

V\_PV = [];

Lamda\_PV = [];

V\_PV\_Upper = [];

V\_PV\_Critical = [];

V\_PV\_Lower = [];

Lamda\_PV\_Upper = [];

Lamda\_PV\_Critical = [];

Lamda\_PV\_Lower = [];

Lambda = 0;

end

%% Power Flow Function: Newton-Raphson

function [V, Delta\_in\_Rad, Iteration] = Newton\_Raphson\_Function(Y\_Bus, V, Delta\_in\_Rad, Lambda, P\_Gen, P\_Load, Q\_Gen, Q\_Load, All\_Bus\_Number, All\_Bus\_Type, Tolerance)

%% Bus Number

Total\_Bus = length(All\_Bus\_Number);

%% Bus Type

PQ\_Bus\_Type = 0;

PQ\_Bus\_Type\_1 = 1;

PV\_Bus\_Type = 2;

Slack\_Bus\_Type = 3;

Bus = All\_Bus\_Number(find(All\_Bus\_Type ~= Slack\_Bus\_Type)); % Bus Type Data Except the Slack Bus

Bus\_Type = All\_Bus\_Type(Bus); % Bus Type Data Except the Slack Bus

PV\_Bus = find(Bus\_Type == PV\_Bus\_Type);

%% Initialization

Iteration = 0;

Tolerance = Tolerance;

while 1

Iteration = Iteration+1;

%% Schedule Real and Reactive Power

P\_Scheduled = transpose(P\_Gen - (Lambda \* P\_Load));

Q\_Scheduled = transpose(Q\_Gen - (Lambda \* Q\_Load));

%% Calculating Real Power

% Initialization

P\_Calculated = zeros(1,length(Bus));

% LOOP: Computing Real Power

for i=1:length(Bus)

for n=1:Total\_Bus

P\_Calculated(i) = P\_Calculated(i) + (abs(abs(Y\_Bus(Bus(i),n)) \* V(Bus(i))\*V(n))) \* (cos(angle(Y\_Bus(Bus(i),n)) + Delta\_in\_Rad(n) - Delta\_in\_Rad(Bus(i))));

end

end

%% Calculating Reactive Power

% Initialization

Q\_Calculated = zeros(1,length(Bus));

% LOOP: Computing Reactive Power

for i=1:length(Bus)

for n=1:Total\_Bus

Q\_Calculated(i) = Q\_Calculated(i) + (abs(abs(Y\_Bus(Bus(i),n)) \* V(Bus(i)) \* V(n))) \* (sin(angle(Y\_Bus(Bus(i),n)) + Delta\_in\_Rad(n) - Delta\_in\_Rad(Bus(i))));

end

Q\_Calculated(i) = - Q\_Calculated(i);

end

%% Calculating Mismatch

Delta\_P = P\_Scheduled - P\_Calculated;

Delta\_Q = Q\_Scheduled - Q\_Calculated;

Delta\_Q(:,PV\_Bus)=[];

Delta\_P\_Q = [transpose(Delta\_P);transpose(Delta\_Q)];

[J] = Jacobian\_Function(V, Delta\_in\_Rad, Y\_Bus, All\_Bus\_Type);

Delta\_J = Delta\_in\_Rad(find(All\_Bus\_Type ~= Slack\_Bus\_Type ));

V\_J = V(find((All\_Bus\_Type ~= Slack\_Bus\_Type) & (All\_Bus\_Type ~= PV\_Bus\_Type)));

%% Updating V and Delta through LU Factorization

% Function Calling: LU Factorization Using Dolittle's Method

[V\_Delta\_Corrected] = LU\_Factorization\_Dolittle\_Function(J,Delta\_P\_Q);

V\_Delta\_Corrected = transpose(V\_Delta\_Corrected);

% LOOP: Sorting the Voltages and Angles after LU Factorization

for i=1:length(V\_Delta\_Corrected)

if (i <= length(Delta\_P))

Delta\_Corrected(i) = V\_Delta\_Corrected(i);

else

V\_Corrected(i-length(Delta\_P)) = V\_Delta\_Corrected(i);

end

end

% Updating Voltages and Angles

Delta\_Updated = Delta\_J + Delta\_Corrected;

V\_Updated = V\_J .\* (1 + V\_Corrected);

% Preparing for Next Iteration

V\_i = (find((All\_Bus\_Type == PQ\_Bus\_Type) | (All\_Bus\_Type == PQ\_Bus\_Type\_1)));

Delta\_i = find(All\_Bus\_Type ~= Slack\_Bus\_Type);

for i=1:length(Delta\_i)

Delta\_New(Delta\_i(i)) = Delta\_Updated(i);

end

for i=1:length(V\_i)

V(V\_i(i)) = V\_Updated(i);

end

Delta\_in\_Rad = Delta\_New;

Delta\_in\_Degree = (180 / pi) \* Delta\_in\_Rad;

if (max(abs(Delta\_P)) < Tolerance & max(abs(Delta\_Q)) < Tolerance)

break;

end

if Iteration >= 25

break;

end

end

end

%% Jacobian Function

function [J] = Jacobian\_Function(V, Delta\_in\_Rad, Y\_Bus, All\_Bus\_Type)

PQNum=0;

PVNum=0;

countPQ=0;

countPV=0;

nbus=length(All\_Bus\_Type);

for i=1:nbus

if All\_Bus\_Type(i) == 0

countPQ=countPQ+1;

PQNum=PQNum+1;

PQBus(countPQ)=i;

elseif All\_Bus\_Type(i) == 1

countPQ=countPQ+1;

PQNum=PQNum+1;

PQBus(countPQ)=i;

elseif All\_Bus\_Type(i) == 2

countPV=countPV+1;

PVNum=PVNum+1;

PVBus(countPV)=i;

end

end

% Calculating J11

J11 = zeros(nbus-1,nbus-1);

for i=1:nbus-1

n=i+1;

for jj=1:nbus-1

m=jj+1;

if n==m

for k=1:nbus

J11(i,jj) = J11(i,jj) + V(n)\*V(k)\*abs(Y\_Bus(n,k))\*sin(angle(Y\_Bus(n,k))+Delta\_in\_Rad(k)-Delta\_in\_Rad(n));

end

J11(i,jj) = J11(i,jj) - V(n)^2\*imag(Y\_Bus(n,n));

else

J11(i,jj) = -V(n)\*V(m)\*abs(Y\_Bus(n,m))\*sin(angle(Y\_Bus(n,m))+Delta\_in\_Rad(m)-Delta\_in\_Rad(n));

end

end

end

% Calculating J21

J21 = zeros(nbus-PVNum-1,nbus-1);

for i=1:nbus-PVNum-1

n=PQBus(i);

for jj=1:nbus-1

m=jj+1;

if n==m

for k=1:nbus

J21(i,jj) = J21(i,jj) + V(n)\*V(k)\*abs(Y\_Bus(n,k))\*cos(angle(Y\_Bus(n,k))+Delta\_in\_Rad(k)-Delta\_in\_Rad(n));

end

J21(i,jj) = J21(i,jj) - V(n)^2\*real(Y\_Bus(n,n));

else

J21(i,jj) = -V(n)\*V(m)\*abs(Y\_Bus(n,m))\*cos(angle(Y\_Bus(n,m))+Delta\_in\_Rad(m)-Delta\_in\_Rad(n));

end

end

end

% Calculating J12

J12 = zeros(nbus-1,nbus-PVNum-1);

for i=1:nbus-1

n=i+1;

for jj=1:nbus-PVNum-1

m=PQBus(jj);

if n==m

for k=1:nbus

J12(i,jj) = J12(i,jj) + V(n)\*V(k)\*abs(Y\_Bus(n,k))\*cos(angle(Y\_Bus(n,k))+Delta\_in\_Rad(k)-Delta\_in\_Rad(n));

end

J12(i,jj) = J12(i,jj) + V(n)^2\*real(Y\_Bus(n,n));

else

J12(i,jj) = V(n)\*V(m)\*abs(Y\_Bus(n,m))\*cos(angle(Y\_Bus(n,m))+Delta\_in\_Rad(m)-Delta\_in\_Rad(n));

end

end

end

% Calculating of J22

J22 = zeros(nbus-PVNum-1,nbus-PVNum-1);

for i=1:nbus-PVNum-1

n=PQBus(i);

for jj=1:nbus-PVNum-1

m=PQBus(jj);

if n==m

for k=1:nbus

J22(i,jj) = J22(i,jj) - V(n)\*V(k)\*abs(Y\_Bus(n,k))\*sin(angle(Y\_Bus(n,k))+Delta\_in\_Rad(k)-Delta\_in\_Rad(n));

end

J22(i,jj) = J22(i,jj) - V(n)^2\*imag(Y\_Bus(n,n));

else

J22(i,jj) = J11(n-1,m-1);

end

end

end

J=[J11,J12;J21,J22];

end

%% LU Factorization Function: Dolittle's Algorithm

function [ X\_Matrix ] = LU\_Factorization\_Dolittle\_Function(A\_Matrix,B\_Matrix)

% Getting the Size of Input Matrix

Length\_A = length(A\_Matrix);

% Initializing The Lower and Upper Triangular Matrices

Lower\_Triangular\_Matrix = zeros(Length\_A,Length\_A);

Upper\_Triangular\_Matrix = zeros(Length\_A,Length\_A);

% LOOP: Assigning 1 into All Diagonal Elements of Lower Traingular Matrix

for j = 1:Length\_A

Lower\_Triangular\_Matrix(j,j) = 1;

end

% Computing 1st Row of Upper Traingular Matrix

Upper\_Triangular\_Matrix(1,:) = A\_Matrix(1,:);

% Computing 1st Column of Lower Traingular Matrix

Lower\_Triangular\_Matrix(:,1) = A\_Matrix(:,1)/Upper\_Triangular\_Matrix(1,1);

% LOOP: Computing All Other Rows and Column of Upper and Lower Traingular Matrix

for j = 2:Length\_A

for k = j:Length\_A

Upper\_Triangular\_Matrix(j,k) = A\_Matrix(j,k) - Lower\_Triangular\_Matrix(j,1:j-1) \* Upper\_Triangular\_Matrix(1:j-1,k);

end

for l = j+1:Length\_A

Lower\_Triangular\_Matrix(l,j) = (A\_Matrix(l,j) - Lower\_Triangular\_Matrix(l,1:j-1) \* Upper\_Triangular\_Matrix(1:j-1,j)) / Upper\_Triangular\_Matrix(j,j);

end

end

% Output

% A\_Matrix

% Lower\_Triangular\_Matrix

% Upper\_Triangular\_Matrix

% Verification

% A\_Matrix - (Lower\_Triangular\_Matrix \* Upper\_Triangular\_Matrix)

%% Forward Substitution

% Initialization of Y Matrix

Y\_Matrix = zeros(Length\_A,1);

% Computing First Value of Y Matrix

Y\_Matrix(1) = B\_Matrix(1) / Lower\_Triangular\_Matrix(1,1);

% LOOP: Computing Rest of the Entries of Y Matrix

for j = 2:Length\_A

Y\_Matrix(j) = (B\_Matrix(j) - Lower\_Triangular\_Matrix(j,1:j-1) \* Y\_Matrix(1:j-1)) / Lower\_Triangular\_Matrix(j,j);

end

% Output

% Y\_Matrix

%% Backward Substitution

% Initialization of X Matrix

X\_Matrix = zeros(Length\_A,1);

% Computing Last Value of X Matrix

X\_Matrix(Length\_A) = Y\_Matrix(Length\_A) / Upper\_Triangular\_Matrix(Length\_A,Length\_A);

% LOOP: Computing Rest of the Entries of X Matrix

for j = Length\_A-1:-1:1

X\_Matrix(j) = (Y\_Matrix(j) - Upper\_Triangular\_Matrix(j,j+1:Length\_A) \* X\_Matrix(j+1:Length\_A)) / Upper\_Triangular\_Matrix(j,j);

end

% Output

% X\_Matrix

end