25.03.2019

MIDDLE EAST TECHNICAL UNIVERSITY
DEPARTMENT OF ELECTRICAL & ELECTRONICS ENGINEERING

EE472
Spring 2019

Term Project 2:
Power Flow Analysis Using Newton-Raphson Iterations in MATLAB Environment

Student: İven GÜZEL

Student Number: 2030831

Instructor: Murat GÖL

Table of Contents

1. INTRODUCTION 3
2. RESULTS 3
  2.1. Methods to Improve Computational Performance 3
  2.2. Test Results 4
  2.3. Convergence Threshold 7
3. CONCLUSION 10
APPENDIX 1: SOURCE CODE 10

1. INTRODUCTION

In a good power system operation, there should be no transmission line or transformer limit violation, all generating units should operate under their reactive power capacity, and bus voltages should be within the operating limit under the system's normal operation or any of its contingencies. Power flow analysis is an important tool to determine how a system should be modified or constructed to remove likely contingencies or to serve a new load. Therefore; solving a power flow problem using data from an input file in common data format (cdf) is an important step in power system operation planning.

In this project, implementing a function in MATLAB environment that can form the bus admittance matrix (Y_{bus}) of a system by Building Block Method and find complex bus voltages by solving the power balance equations using Newton Raphson iterations is aimed. The function must be able to find the voltage magnitudes and angles of all buses from different systems in common data format (cdf). Source code of this function can be found in Appendix I. Moreover, this report contains discussions on computational methods used to improve computational performance, test results, and convergence threshold of iterations in the Results section.

2. RESULTS

2.1. Methods to Improve Computational Performance

The methods that are used to improve the computational performance of the function are generally decided by following MATLAB's suggestions to have the operation faster.

First of all, as few numbers of vectors and matrices as possible are changing size in every loop iteration to make the function faster.

Secondly, to find Δx values in every iteration inverse of the Jacobian matrix should be taken as it can be seen in Equation (1). Computational cost of taking the inverse of an (nxn) matrix is n^3. Therefore; as the bus number of the system increases, iteration time increases significantly. Furthermore, with increasing bus number the required iteration number to reach a tolerable error increase as well. As an alternative, to decrease the computation time division comment is used in the function.

$$(x^{k+1} - x^k) = -J^{-1}(x^k).F(x^k) \qquad (1)$$

Table 1, provides the elapsed time between bus admittance matrix is formed using the cdf file and the IEEE bus systems converges using the two different methods mentioned above. These times are recorded before the reactive power limits of the generators are checked, therefore; they include only one set of Newton-Raphson iterations.

Table 1: Solution Durations of Division and Inversion Methods

Method	IEEE 14 Bus	IEEE 118 Bus	IEEE 300 Bus
Inverse of the Matrix	0.0559	0.1810	1.7800
'\' comment	0.0506	0.1542	1.2028

As a third method I wanted to use the sparsity property of the bus admittance matrix. However, I had a warning from MATLAB suggesting that sparse indexing expression of Ybus matrix will likely to be slow Also, storing the Ybus as a sparse matrix caused errors while I was implementing the solution of the power flow problem. As a solution, I decided to accumulate nonzero values after the computation of Ybus is done and then only calculate power balance equations and Jacobians that use the nonzero value indexes. Unfortunately, I did not have time to implement this method.

2.2. Test Results

From Figure 1 to 6 error between the found angle & voltages and the ones obtained from the cdf files of IEEE bus 14, 118 and 300 systems can be observed.

As it can be seen error in bus voltages are in the order of 10^{-3} and the error in theta values are slightly larger. Some bus voltage errors are considerable larger than the others in IEEE 188 and IEEE300 bus systems. They are due to a mistake in the lines where reactive power limits of the generators are checked. Unfortunately, I did not have the time to fix this error. However, I believe these errors are still tolerable values.

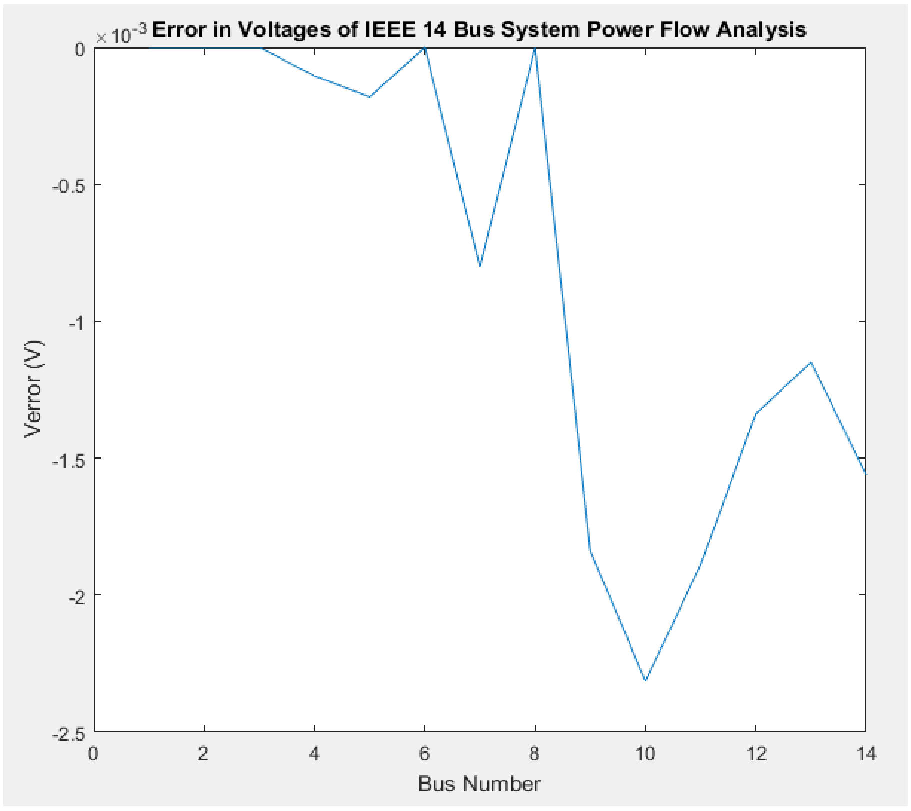

Figure 1: Error in Voltages of IEEE Bus 14 System

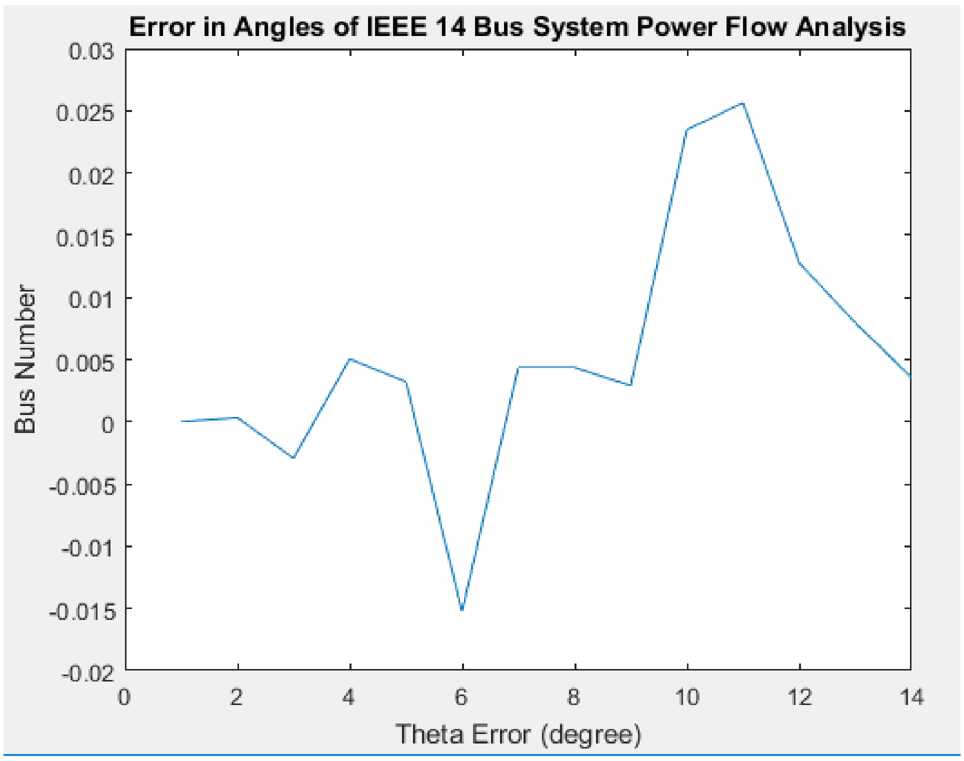

Figure 2: Error in Angles of IEEE Bus 14 System

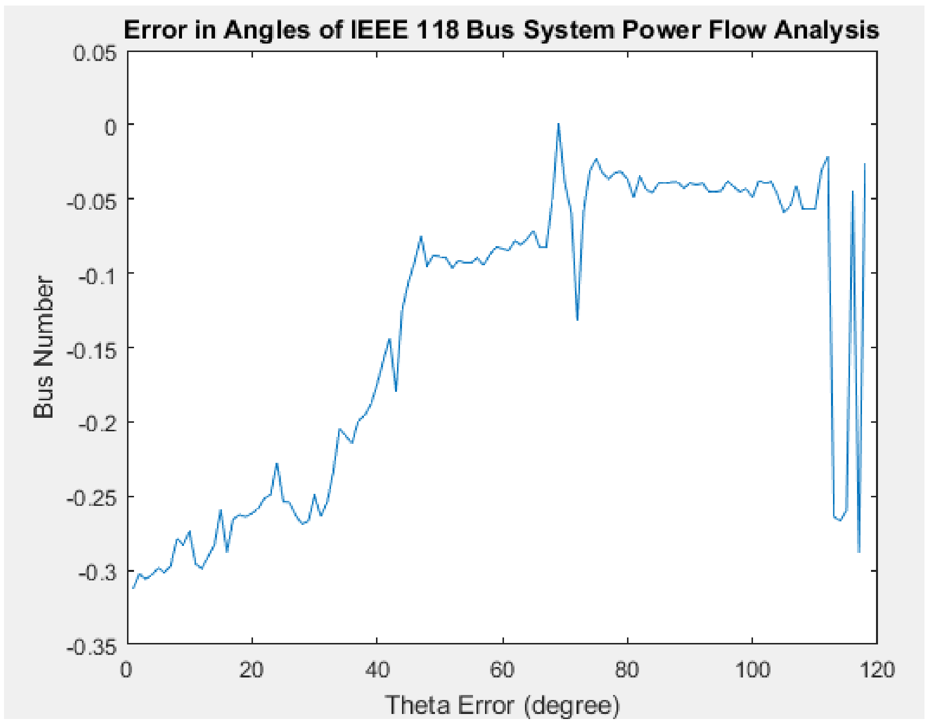

Figure 3: Error in Angles of IEEE Bus 118 System

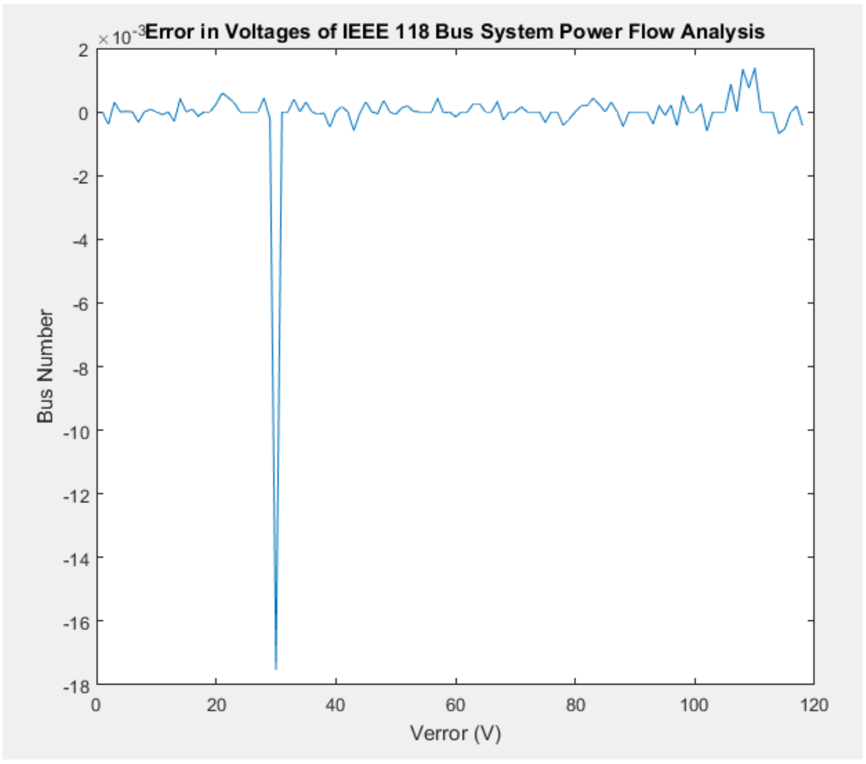

Figure 4: Error in Voltages of IEEE Bus 118 System

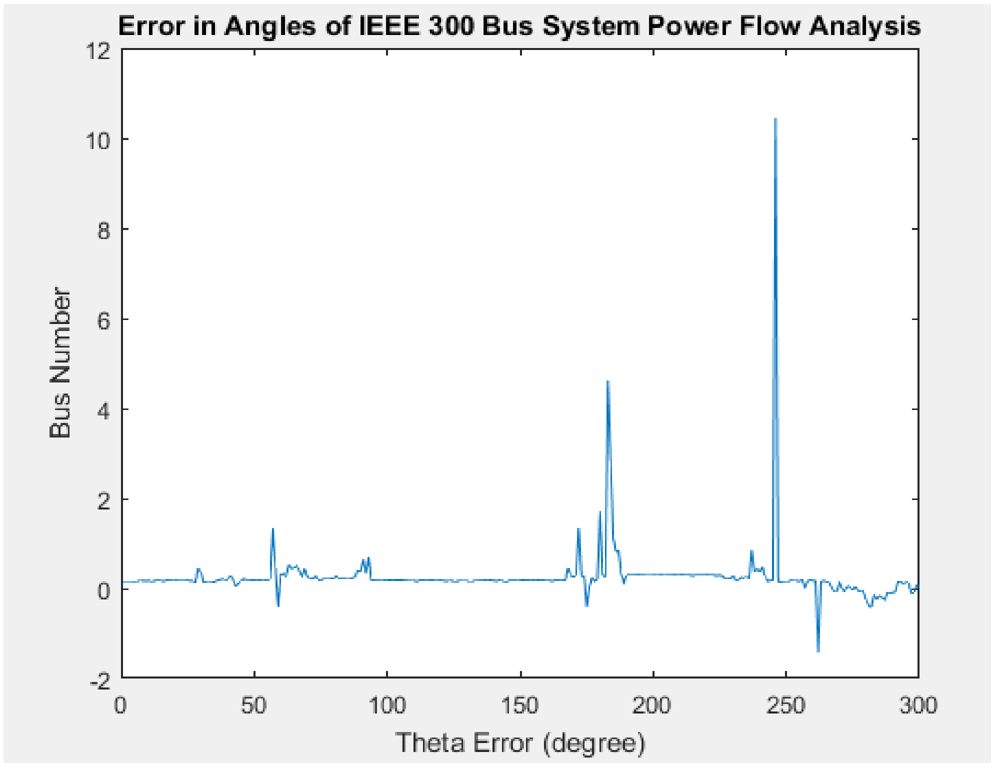

Figure 5: Error in Voltages of IEEE Bus 300 System

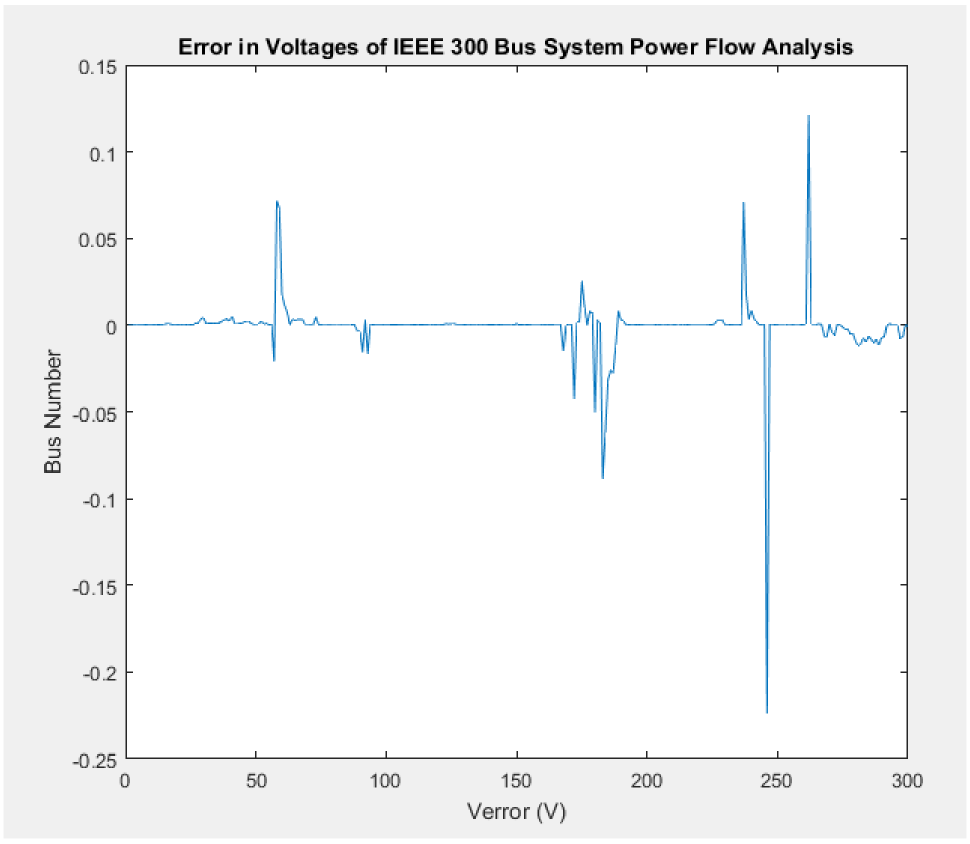

Figure 6: Error in Voltages of IEEE Bus 300 System

2.3. Convergence Threshold

In Figure 7, 8, and 8 tolerance versus iteration of different bus systems can be seen. From these plots it can be seen that Newton-Raphson method has quadratic converges thanks to this convergence is fast. In fact, I limited the tolerance to 10^{-5} for convergence. Convergence is so fast that decreasing this value to 10^{-4} did not affect the iteration numbers for different bus systems as it can also be observed in Table 2, which shows the tolerance in each iteration.

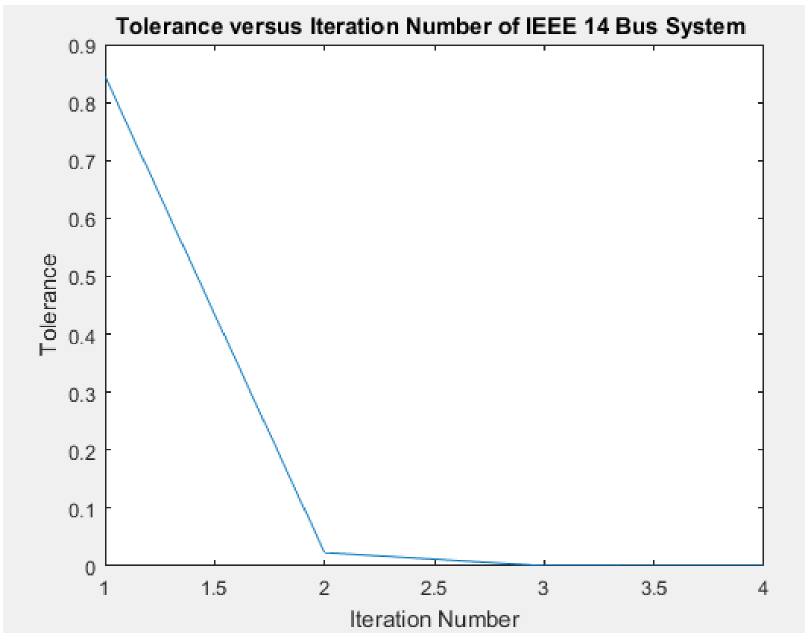

Figure 7: Tolerance of IEEE 14 Bus system

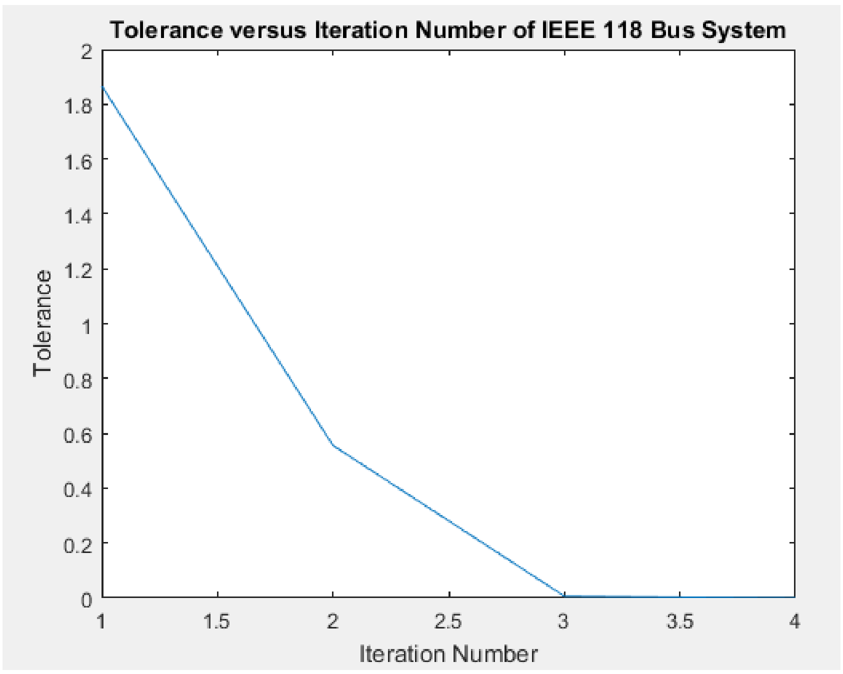

Figure 8: Tolerance of IEEE 118 Bus system

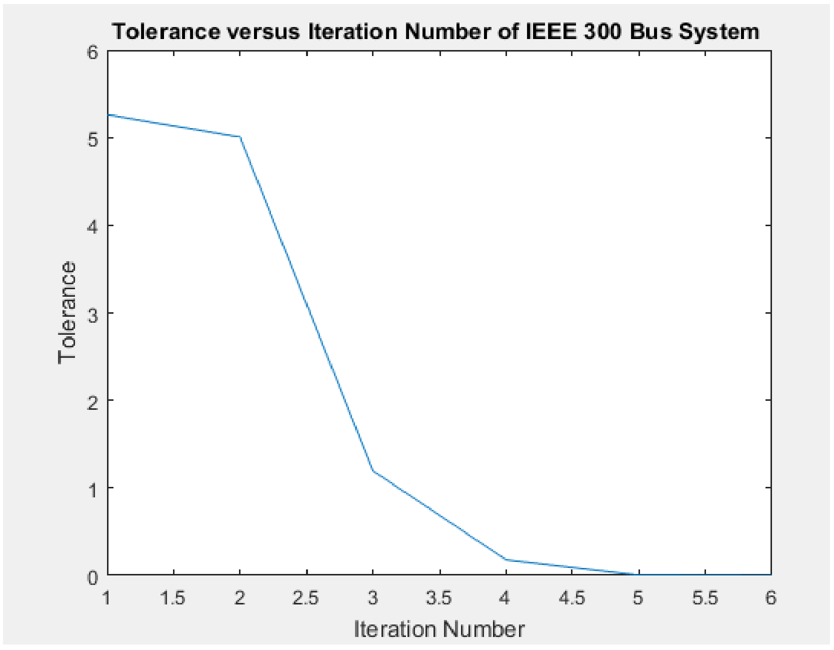

Figure 9: Tolerance of IEEE 300 Bus system

Table 2: Tolerance in each iteration

Iteration Number	IEEE 14 Bus	IEEE 118 Bus	IEEE 300 Bus
1	0,8461	5.3	5.263
2	0,0232	1.287	5.003
3	0,00024	0.0252	1.193
4	$2.6x10^{-8}$	$2.1x10^{-5}$	0.181
5		$2.39x10^{-11}$	0.0047
6			$3.06x10^{-6}$

3. CONCLUSION

To sum up, in this project a function that solves power flow analysis has been implemented. Due to a mistake in implementing Q limits of generating buses I have larger errors compared to other buses in some buses but these errors are still small.

APPENDIX 1: SOURCE CODE

```
function [V,theta] = e203083_guzel_Power_Flow(path_ieee_cdf)

%READING DATA FROM THE TEXT FILE


fileID = fopen(path_ieee_cdf); %opens the file for binary read access by
giving file identifier
tline = fgetl(fileID); %returns the next line of fileID (removes nexline
char)
file = tline;
while ischar(tline)
    tline = fgetl(fileID);
    file = char(file,tline);
end   %cdf file is written and ready to be worked on

fclose(fileID);

%% CREATE Ybus
[Nbus,PQ,PV,BusIndex,G,B,Slack,S,Qmin,Qmax,VSlack,Nbus_first,V_real,Theta_r
eal,Theta_Slack] = createYbus(file) ;

%% RUN NEWTON RAPHSON ITERATIONS (without Q limits of the generators)
tic;
[V,theta,Q,tol_vec] =
NewtonRap(Nbus,PQ,PV,BusIndex,G,B,Slack,S,VSlack,Theta_Slack) ;
Tdiv =toc;
%% CHECK Q LIMITS

i = 1 ;

 while(1)

 ind=find(BusIndex == PV(i,1));
 if Qmin(ind,1)>Q(ind,1) ||  Q(ind,1)>Qmax(ind,1)

file(ind+Nbus_first-1,(26)) = num2str(0) ;
[Nbus,PQ,PV,BusIndex,G,B,Slack,S,Qmin,Qmax,VSlack,Nbus_first,V_real,Theta_r
eal,Theta_Slack] = createYbus(file) ;
[V,theta,Q] = NewtonRap(Nbus,PQ,PV,BusIndex,G,B,Slack,S,VSlack,Theta_Slack)
;
file(ind+Nbus_first-1,(26)) = num2str(2) ;

 end
 i = i+1 ;
 if  i-1==length(PV)
     break;
 end

end

theta = radtodeg(theta);

function
[Nbus,PQ,PV,BusIndex,G,B,Slack,S,Qmin,Qmax,VSlack,Nbus_first,V_real,Theta_r
eal,Theta_Slack]= createYbus(file)

%This local function takes cdf file as input and creates Ybus

BMva=100; %Sbase (MVA)

%% FIND the NUMBER of BUSES
first_row = 1;

while(1)   %Find where BUS info starts
    if file(first_row,1:3) == 'BUS'
        break;
    else
        first_row = first_row + 1;
    end
end

first_row = first_row + 1; %Because bus data starts in the next line from
'BUS'
row_num = first_row;

while(1)  %Count the rows until you see '-999'
    if file(row_num,(1:4)) == '-999'
        break;
    end
row_num = row_num + 1;
end

Nbus_first = first_row;
Nbus_last = row_num -1 ;  %row_num is the line with -999
Nbus = Nbus_last - Nbus_first +1; %Number of buses in the system

%% FIND the NUMBER of BRANCHES
 Nbranch_start = row_num + 2;
 row_num = Nbranch_start;

 while(1)     %Look for -999
    if file(row_num,(1:4)) == '-999'
        break;
    end
    row_num=row_num+1;
 end

Nbranch_last = row_num - 1; %row_num is where -999 is.
Nbranch = Nbranch_last - Nbranch_start +1; %Number of branches

%% CREATING Ybus and FINDING INJECTED POWER & Q LIMITS & BUS TYPES
```

```
BusIndex = zeros(Nbus,1);
Ybus = zeros(Nbus,Nbus);
countpq = 1 ;
countpv = 1 ;
PQ = [] ;
PV= [] ;
Qmin = zeros(Nbus,1);
Qmax = zeros(Nbus,1);

for m=1:Nbus
    G = str2double(file(Nbus_first + (m-1) ,(107:114)));
    B = str2double(file(Nbus_first + (m-1),(115:122)));
    Ybus(m,m)= G + 1i*B ;
    Qmin(m,1) = str2double(file(Nbus_first + (m-1) ,(99:105)));
    Qmax(m,1) = str2double(file(Nbus_first + (m-1) ,(91:97)));
    P_cdf(m,1) = str2double(file(Nbus_first + (m-1),(60:66))) -
str2double(file(Nbus_first + (m-1),(41:48))) ;
    V_real(m,1) = str2double(file(Nbus_first + (m-1) ,(28:33)));
    Theta_real(m,1) = str2double(file(Nbus_first + (m-1) ,(34:40)));

    if (str2double(file(Nbus_first + (m-1),(25:26))) == 0 ||
(str2double(file(Nbus_first + (m-1),(25:26))) == 1)) %Type PQ buses
        PQ(countpq,1) = str2double(file(Nbus_first + (m-1),(1:4))) ;  %bus
name
        PQ(countpq,2) = str2double(file(Nbus_first + (m-1),(60:66))) -
str2double(file(Nbus_first + (m-1),(41:48))) ;  % Pg-Pl for PQ buses
        PQ(countpq,3) = str2double(file(Nbus_first + (m-1),(68:74))) -
str2double(file(Nbus_first + (m-1),(50:57))) ;  % Qg-Ql for PQ buses
        countpq = countpq +1 ;
    end

    if (str2double(file(Nbus_first + (m-1),(25:26)))) == 2  %Type PV buses
        PV(countpv,1)= str2double(file(Nbus_first + (m-1),(1:4))) ;
%Stores bus name
        PV(countpv,2)= str2double(file(Nbus_first + (m-1),(28:32))) ; %
Stores final voltage values (pu)
        PV(countpv,3) = str2double(file(Nbus_first + (m-1),(60:66))) -
str2double(file(Nbus_first + (m-1),(41:48))) ;  % Pg-Pl for PV buses
        countpv = countpv +1 ;
    end

    if (str2double(file(Nbus_first + (m-1),(25:26)))) == 3  %Type Slack
        Slack(1,1) = str2double(file(Nbus_first + (m-1),(1:4))) ; %bus name
        VSlack = str2double(file(Nbus_first + (m-1),(28:32))) ;  %theta
slack is assumed to be zero
        Theta_Slack = str2double(file(Nbus_first + (m-1),(34:40))) ;
    end

    BusIndex(m,1) =str2double(file(Nbus_first+ (m-1) ,(1:4))) ; %Bus names
might be different than our index, so store these for branch relations
end


for m=1:Nbranch     %This loop constructs Ybus
    send_bus = str2double(file(Nbranch_start+m-1,(1:4)));
    rec_bus = str2double(file(Nbranch_start+m-1,(6:9)));
    R = str2double(file(Nbranch_start+m-1,(20:29)));
    X = str2double(file(Nbranch_start+m-1,(30:40)));
    B = str2double(file(Nbranch_start+m-1,(41:50)));
    Turn_ratio = str2double(file(Nbranch_start+m-1,(77:82)));

    TR_type = str2double(file(Nbranch_start+m-1,(19)));

    if TR_type == 4
        Phase_angle = str2double(file(Nbranch_start+m-1,(84:90)));
        Turn_ratio = (Turn_ratio*cos(Phase_angle*0.0174532925)+
Turn_ratio*sin(Phase_angle*0.0174532925) ); %For Phase shifters 1
deg=0.0174532925 radians
    end

    if Turn_ratio == 0 %they put zero sometimes to indicate no transformers
       Turn_ratio = 1;
    end

    Y1 = find(BusIndex == send_bus);           % To see which index sending
bus name corresponds
    Y2 = find(BusIndex == rec_bus);        % To see which index receiving
bus name corresponds

    Ybus(Y1,Y1) = Ybus(Y1,Y1) + (1/(R + 1i* X)) / (Turn_ratio^2) + 1i*B/2;
% When the type is TL it does not hange anything since n=1
    Ybus(Y2,Y2) = Ybus(Y2,Y2)+ (1/(R + 1i* X)) + 1i*B/2;
    Ybus(Y1,Y2) = Ybus(Y1,Y2) - ((1/(R + 1i* X)) / (conj(Turn_ratio)));
    Ybus(Y2,Y1) = Ybus(Y2,Y1) - ((1/(R + 1i* X)) /Turn_ratio) ;

end


%% PROJECT 2:

G = real(Ybus) ; %These values were in Pu
B = imag(Ybus) ;
Slack = find (Slack == BusIndex) ;
S = [P_cdf([1:Slack-1,Slack+1:end]);PQ(:,3)] ; % [Injected power of all
buses except slack's , Injected react power of Q buses] ==> True values
S = S / BMva ; % Find pu values

Qmin = Qmin / BMva ; % Find pu values
Qmax = Qmax / BMva ;

end


function [V,theta,Q,tol_vec] =
NewtonRap(Nbus,PQ,PV,BusIndex,G,B,Slack,S,VSlack,Theta_Slack)

theta = zeros(Nbus,1) ;     % flat start
theta(Slack) = degtorad(Theta_Slack);
V = ones(Nbus,1) ;  % flat start
V(Slack) = VSlack ;


for i=1:length(PV)
 ind=find(BusIndex == PV(i,1)); % PV(:,1) stores bus names , bus index
might be different than bus name
 V(ind,1) = PV(i,2) ; %We know V of PV buses so add this info
end
```

```
tol = 100 ;   % We will check if the tolerance converges

it_number = 0;
while(tol>0.00001)   %SET THE ACCEPTABLE ERROR in the APPROXIMATION HERE !!

%% POWER BALANCE EQUATIONS

P = zeros(Nbus,1);
Q = zeros(Nbus,1);
for r=1:Nbus
    for j=1:Nbus
 P(r,1)=P(r,1)+V(r)*V(j)*(G(r,j)*cos(theta(r)-
theta(j))+B(r,j)*sin(theta(r)-theta(j)));
 Q(r,1)=Q(r,1)+V(r)*V(j)*(G(r,j)*sin(theta(r)-theta(j))-
B(r,j)*cos(theta(r)-theta(j)));
        end
end

% % % % % for i=1:length(PV)
% % % % %  ind=find(BusIndex == PV(i,1));
% % % % %  S_est(i,1) = P(ind,1) ;
% % % % % end


S_est = P([1:Slack-1,Slack+1:end]) ; %Write P&Q in column format

for i=1:length(PQ)
 ind=find(BusIndex == PQ(i,1));
 S_est(length(PV)+length(PQ)+i,1) = Q(ind,1) ;
end

%% JACOBIAN MATRICES

J_11=zeros(Nbus,Nbus); %J11 =delP/del(theta)

 for r=1:Nbus
        m=r;
        for j=1:Nbus
            n=j;
            if m==n
                for n=1:Nbus
                J_11(r,j)=J_11(r,j)+V(m)*V(n)*(-G(m,n)*sin(theta(m)-
theta(n))+B(m,n)*cos(theta(m)-theta(n)));
                end
                J_11(r,j)=J_11(r,j)-V(m)^2*B(m,m);
            else
                J_11(r,j)=V(m)*V(n)*(G(m,n)*sin(theta(m)-theta(n))-
B(m,n)*cos(theta(m)-theta(n)));
            end
        end
 end
%% Formation Of J_12
    J_12=zeros(Nbus,length(PQ));
    for r=1:Nbus
        m=r;
        for j=1:length(PQ)
            n=find(PQ(j,1)==BusIndex);
            if m==n
                for n=1:Nbus
                    J_12(r,j)=J_12(r,j)+V(n)*(G(m,n)*cos(theta(m)-
theta(n))+B(m,n)*sin(theta(m)-theta(n)));
                end
                J_12(r,j)=J_12(r,j)+V(m)*G(m,m);
            else
                J_12(r,j)=V(m)*(G(m,n)*cos(theta(m)-
theta(n))+B(m,n)*sin(theta(m)-theta(n)));
            end
        end
    end
    %% Formation Of J_21
    J_21=zeros(length(PQ),Nbus);
    for r=1:length(PQ)
        m=find(PQ(r,1)==BusIndex);
        for j=1:Nbus
            n=j;
            if m==n
                for n=1:Nbus
                    J_21(r,j)=J_21(r,j)+V(m)*V(n)*(G(m,n)*cos(theta(m)-
theta(n))+B(m,n)*sin(theta(m)-theta(n)));
                end
                J_21(r,j)=J_21(r,j)-V(m)^2*G(m,m);
            else
                J_21(r,j)=V(m)*V(n)*(-G(m,n)*cos(theta(m)-theta(n))-
B(m,n)*sin(theta(m)-theta(n)));
            end
        end
    end
    %% Formation Of J_22
    J_22=zeros(length(PQ),length(PQ));
    for r=1:length(PQ)
        m=find(PQ(r,1)==BusIndex);
        for j=1:length(PQ)
            n=find(PQ(j,1)==BusIndex);
            if m==n
                for n=1:Nbus
                J_22(r,j)=J_22(r,j)+V(n)*(G(m,n)*sin(theta(m)-theta(n))-
B(m,n)*cos(theta(m)-theta(n)));
                end
                J_22(r,j)=J_22(r,j)-V(m)*B(m,m);
            else
                J_22(r,j)=V(m)*(G(m,n)*sin(theta(m)-theta(n))-
B(m,n)*cos(theta(m)-theta(n)));
            end
        end
    end


J_11=J_11([1:Slack-1,Slack+1:end],:);
J_11=J_11(:,[1:Slack-1,Slack+1:end]);

J_12=J_12([1:Slack-1,Slack+1:end],:);


J_21=J_21(:,[1:Slack-1,Slack+1:end]);

J=[J_11 J_12 ; J_21 J_22]; % Jacobian Matrix
```

```
%% MISMATCH VECTOR F(x)

Fx = S_est - S ;

%X=inv(J)*Fx;

X = J\Fx    ;

theta([1:Slack-1,Slack+1:end],1) = theta([1:Slack-1,Slack+1:end],1)-
X(1:Nbus-1,1) ;   %Remove the slack bus info and update theta values

for i=1:length(PQ)  %Update unknown voltage values which are the voltage
values of PQ buses
 ind=find(BusIndex == PQ(i,1));
 V(ind,1) =  V(ind,1) - X(Nbus-1+i,1) ;
end

tol = norm(X) ; % Check the total error in Delta(X)
it_number = it_number +1;
tol_vec(it_number,1) = tol;
end

%%FINAL P&Q VALUES

P = zeros(Nbus,1);
Q = zeros(Nbus,1);

for r=1:Nbus
    for j=1:Nbus
 P(r,1)=P(r,1)+V(r)*V(j)*(G(r,j)*cos(theta(r)-
theta(j))+B(r,j)*sin(theta(r)-theta(j)));
 Q(r,1)=Q(r,1)+V(r)*V(j)*(G(r,j)*sin(theta(r)-theta(j))-
B(r,j)*cos(theta(r)-theta(j)));
    end
end
end




end
```