# TASK REPORT

**Implementation of Standard Backward Forward Sweep(B/FS) method and Branch Current Based B/FS by MATLAB programming language.**

**Course: Electric Power Distribution Systems**
**Professor: Dr. Karimi**
**Student: Hamed Najafi**
University of Kashan

NOVEMBER 25, 2022

دانشگاه کاشان

# Introduction

The electric power distribution system is characterized by heavy loading conditions at some buses and a high R/X ratio, unbalanced load and mostly radial topology. Many power flow methods have been designed and proved to work efficiently for transmission systems (Newton–Raphson, Gauss–Seidel and improved methods). However, the design assumptions considered for power flow methods in transmission networks are not suitable for power flow analysis in radial distribution networks due to their convergence, memory requirements and computational efficiency.

Load flow solution that fits the requirements for a radial distribution network has been proposed. The backward/forward sweep (BFS) is among the most successful power flow methods for radial networks. The variants of BFS methods have been reported, such as the current summation method, power summation method and admittance summation method. The basic operation principle of BFS involves two computation processes at each iteration. The backward process involves the power or current flow solutions starting from the branch of the end nodes moving toward the branch connected to the reference node. The forward sweep calculates the voltage at each node starting from the reference node to the end nodes. During the backward sweep, the voltage is held constant, and during the forward sweep, the current or power value is held constant. After each iteration, the power flow convergence is tested.

# B/FS Implementation

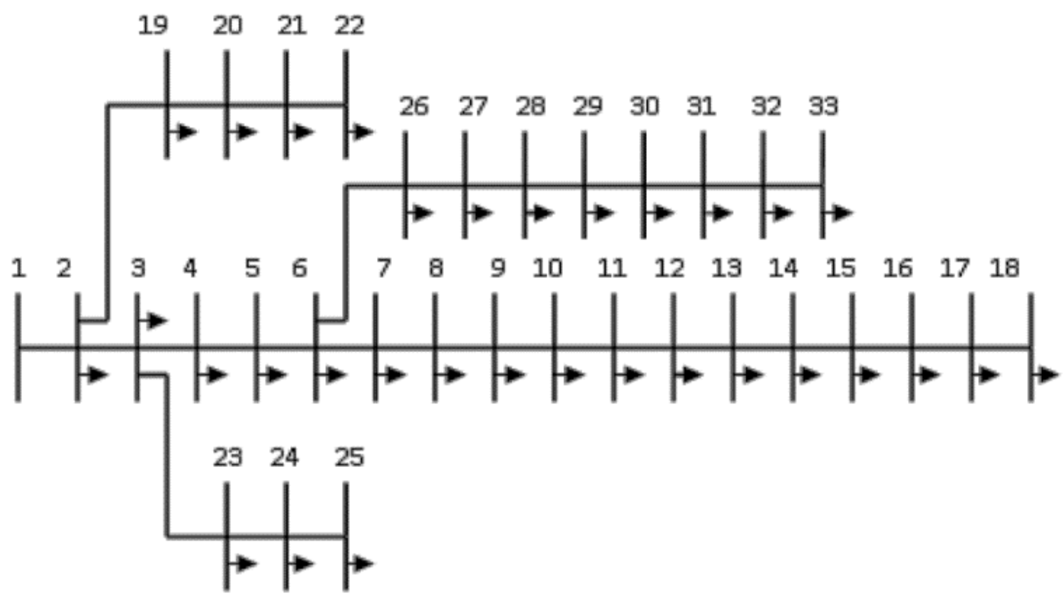
The program automatically read data that stored in two Excel files about branches and loads.

Details of codes are in appendix and commented in .m file as well.

Steps:

1. Reading Loads & Lines Data
2. Setting Initial Parameters (Base Values and…)
3. Forming Connection Matrix (C)
4. Determining the Radial Paths
5. Start loop
6. Backward Step
7. Forward Step
8. Convergence checking
9. Printing results

In this work we a 33-bus system with below topology and specifications (from [1]):
Network topology:




Bus 1 is our slack bus and its voltage remains at 1pu<0 º
Convergence criteria: max {$|\overline{V}_{new} - \overline{V}_{old}|$} < 0.01 $^{pu}$

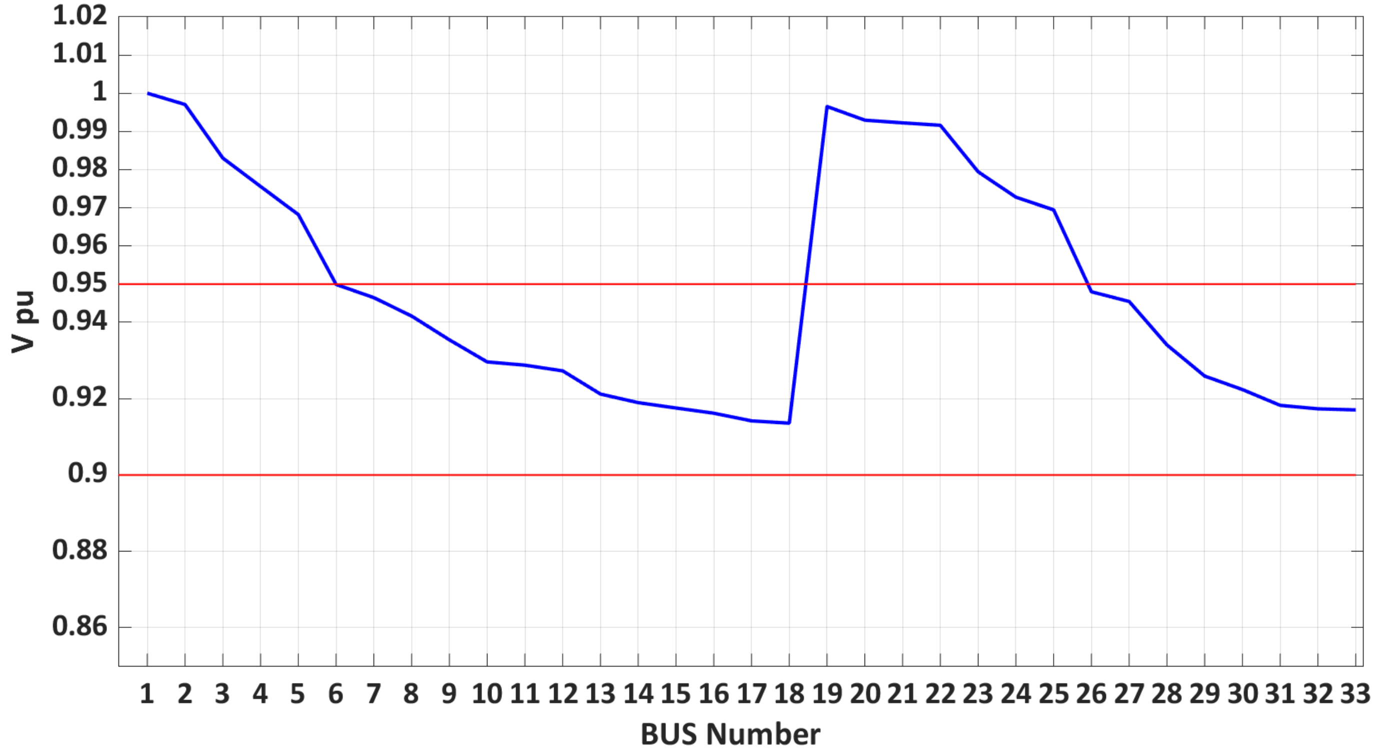
Network specifications**:**

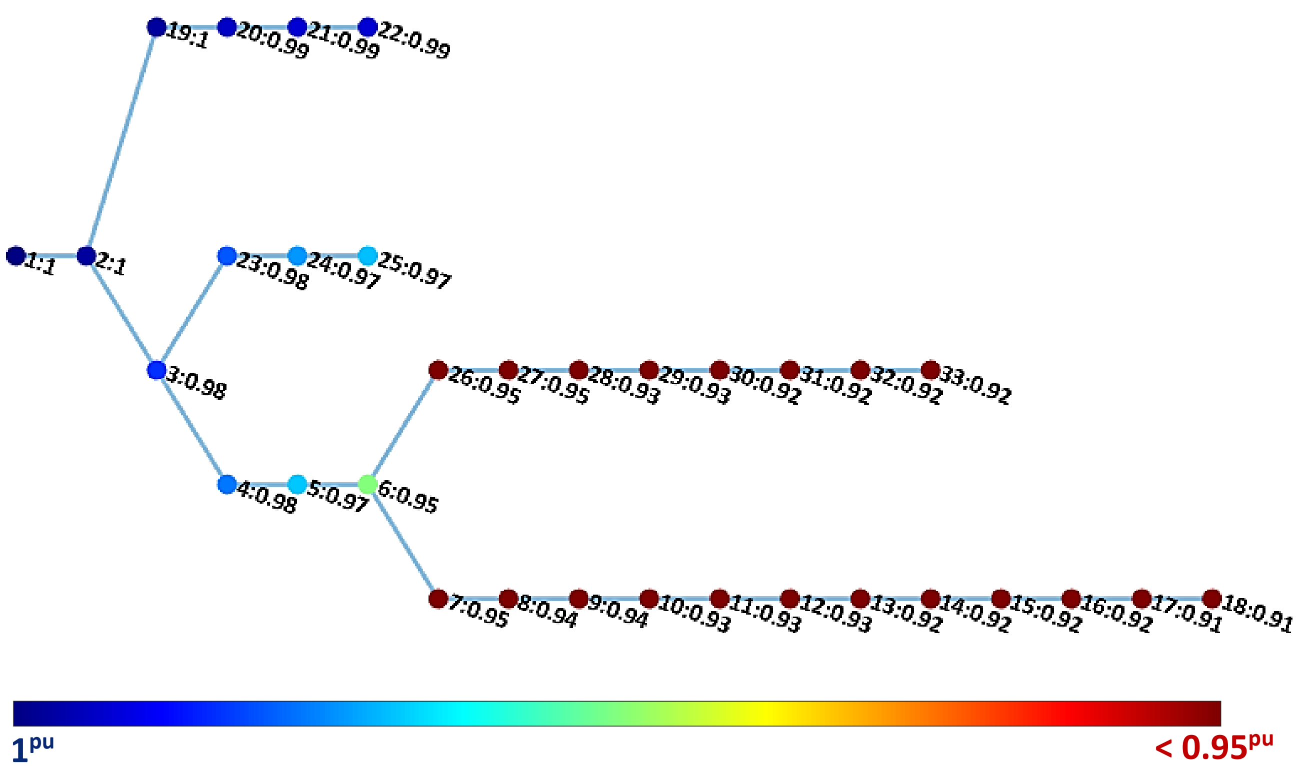
| Branch No. | From bus | To bus | R (Ω) | X (Ω) | Load at to bus | |
|---|---|---|---|---|---|---|
| | | | | | P (kW) | Q (kW) |
| 1 | 1 | 2 | 0.0922 | 0.0477 | 0 | 0 |
| 2 | 2 | 3 | 0.4930 | 0.2511 | 100 | 60 |
| 3 | 3 | 4 | 0.3660 | 0.1864 | 90 | 40 |
| 4 | 4 | 5 | 0.3811 | 0.1941 | 120 | 80 |
| 5 | 5 | 6 | 0.8190 | 0.7070 | 60 | 30 |
| 6 | 6 | 7 | 0.1872 | 0.6188 | 60 | 20 |
| 7 | 7 | 8 | 1.7114 | 1.2351 | 200 | 100 |
| 8 | 8 | 9 | 1.0300 | 0.7400 | 200 | 100 |
| 9 | 9 | 10 | 1.0400 | 0.7400 | 60 | 20 |
| 10 | 10 | 11 | 0.1966 | 0.0650 | 60 | 20 |
| 11 | 11 | 12 | 0.3744 | 0.1238 | 45 | 30 |
| 12 | 12 | 13 | 1.4680 | 1.1550 | 60 | 35 |
| 13 | 13 | 14 | 0.5416 | 0.7129 | 60 | 35 |
| 14 | 14 | 15 | 0.5910 | 0.5260 | 120 | 80 |
| 15 | 15 | 16 | 0.7463 | 0.5450 | 60 | 10 |
| 16 | 16 | 17 | 1.2890 | 1.7210 | 60 | 20 |
| 17 | 17 | 18 | 0.7320 | 0.5740 | 60 | 20 |
| 18 | 2 | 19 | 0.1640 | 0.1565 | 90 | 40 |
| 19 | 19 | 20 | 1.5042 | 1.3554 | 90 | 40 |
| 20 | 20 | 21 | 0.4095 | 0.4784 | 90 | 40 |
| 21 | 21 | 22 | 0.7089 | 0.9373 | 90 | 40 |
| 22 | 3 | 23 | 0.4512 | 0.3083 | 90 | 40 |
| 23 | 23 | 24 | 0.8980 | 0.7091 | 90 | 50 |
| 24 | 24 | 25 | 0.8960 | 0.7011 | 420 | 200 |
| 25 | 6 | 26 | 0.2030 | 0.1034 | 420 | 200 |
| 26 | 26 | 27 | 0.2842 | 0.1447 | 60 | 25 |
| 27 | 27 | 28 | 1.0590 | 0.9337 | 60 | 25 |
| 28 | 28 | 29 | 0.8042 | 0.7006 | 60 | 20 |
| 29 | 29 | 30 | 0.5075 | 0.2585 | 120 | 70 |
| 30 | 30 | 31 | 0.9744 | 0.9630 | 200 | 600 |
| 31 | 31 | 32 | 0.3105 | 0.3619 | 150 | 70 |
| 32 | 32 | 33 | 0.3410 | 0.5302 | 210 | 100 |

**Results:**

Algorithm converged after 2 iterations and the output results are:

Voltage profile:



**Final Voltages:**

| Bus NO | \|U\|(pu) | θ(°) |
|---|---|---|
| 1 | 1 | 0 |
| 2 | 1 | 0.01 |
| 3 | 0.98 | 0.1 |
| 4 | 0.98 | 0.16 |
| 5 | 0.97 | 0.23 |
| 6 | 0.95 | 0.13 |
| 7 | 0.95 | -0.1 |
| 8 | 0.94 | -0.06 |
| 9 | 0.94 | -0.13 |
| 10 | 0.93 | -0.2 |
| 11 | 0.93 | -0.19 |
| 12 | 0.93 | -0.18 |
| 13 | 0.92 | -0.27 |
| 14 | 0.92 | -0.35 |
| 15 | 0.92 | -0.38 |
| 16 | 0.92 | -0.41 |
| 17 | 0.91 | -0.48 |
| 18 | 0.91 | -0.49 |
| 19 | 1 | 0 |
| 20 | 0.99 | -0.06 |
| 21 | 0.99 | -0.08 |
| 22 | 0.99 | -0.1 |
| 23 | 0.98 | 0.06 |
| 24 | 0.97 | -0.02 |
| 25 | 0.97 | -0.07 |
| 26 | 0.95 | 0.17 |
| 27 | 0.95 | 0.23 |
| 28 | 0.93 | 0.31 |
| 29 | 0.93 | 0.39 |
| 30 | 0.92 | 0.49 |
| 31 | 0.92 | 0.41 |
| 32 | 0.92 | 0.39 |
| 33 | 0.92 | 0.38 |

**Total Power Loss in lines (kW) = 200.6229**

**Total Reactive Power Consumed by Lines (kVAr) = 133.7465**

-----------------------------------------------------------------------------------------------------------

# BCBB/FS Implementation

Again, the program automatically read data that stored in two Excel files about branches and loads and another excel file about DGs at buses and their specifications.

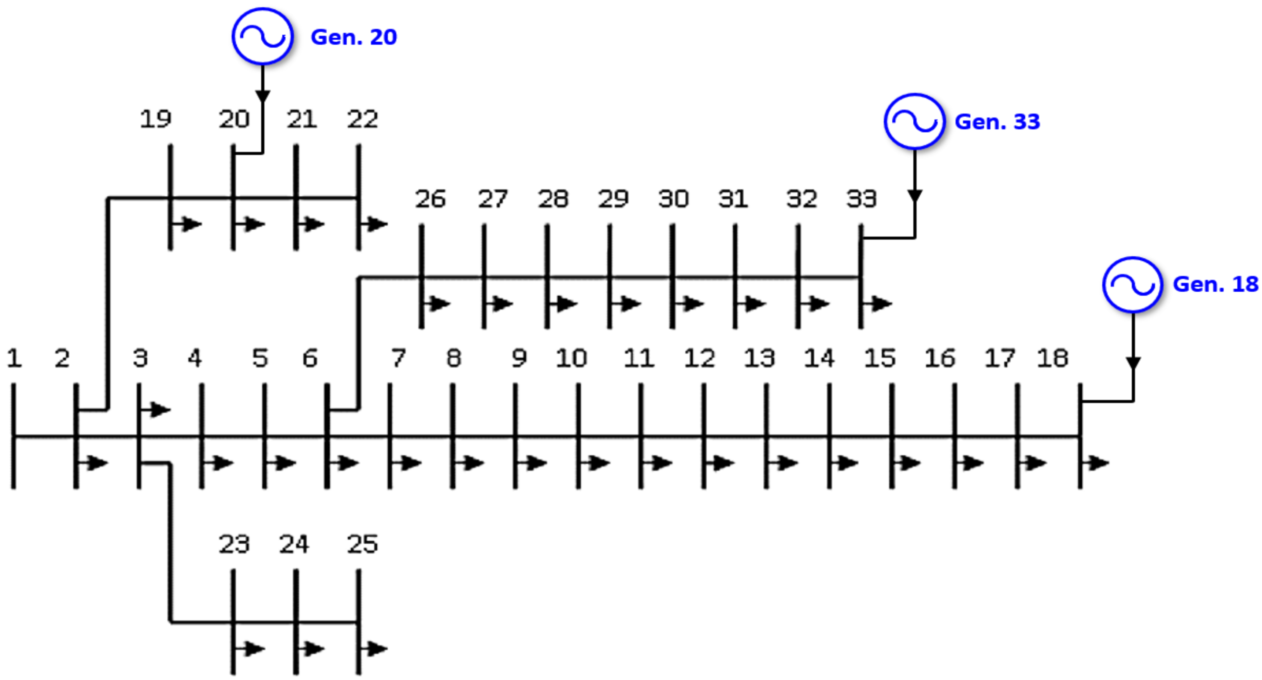
Details of codes are in appendix and commented in .m file as well.

Steps:

1. Reading Loads & Lines Data & DG Buses Data
2. Setting initial parameters
3. Extracting constant P&Q DGs information from DGs data
4. Extracting constant P&V DGs information from DGs data
5. Evaluating new Loads data
6. Forming Connection matrix
7. Forming reactance sensitivity matrix
8. Determining the radial paths
9. Start the outer loop
10. Start the inner loop
11. The inner loop convergence checking
12. End of the inner loop
13. Outer loop Convergence checking
14. If outer loop criteria not satisfied then Updating Q values and continue
15. End of the outer loop (after convergence or maximum iteration number)
16. Printing Results

We added three DGs to the system mentioned before that the specifications are below:

| BUS NUMBER | GENERATED P(KW) | GENERATED Q(KVAR) | **MODE (PQ:0,PV:1)** | SP. V (PU) | MAX. GEN. Q(KVAR) | MAX. CONS. Q(KVAR) |
|---|---|---|---|---|---|---|
| 20 | 60 | 40 | 0 | nan | nan | nan |
| 33 | 200 | nan | 1 | 1 | 100 | 75 |
| 18 | 250 | nan | 1 | 1 | 150 | 80 |



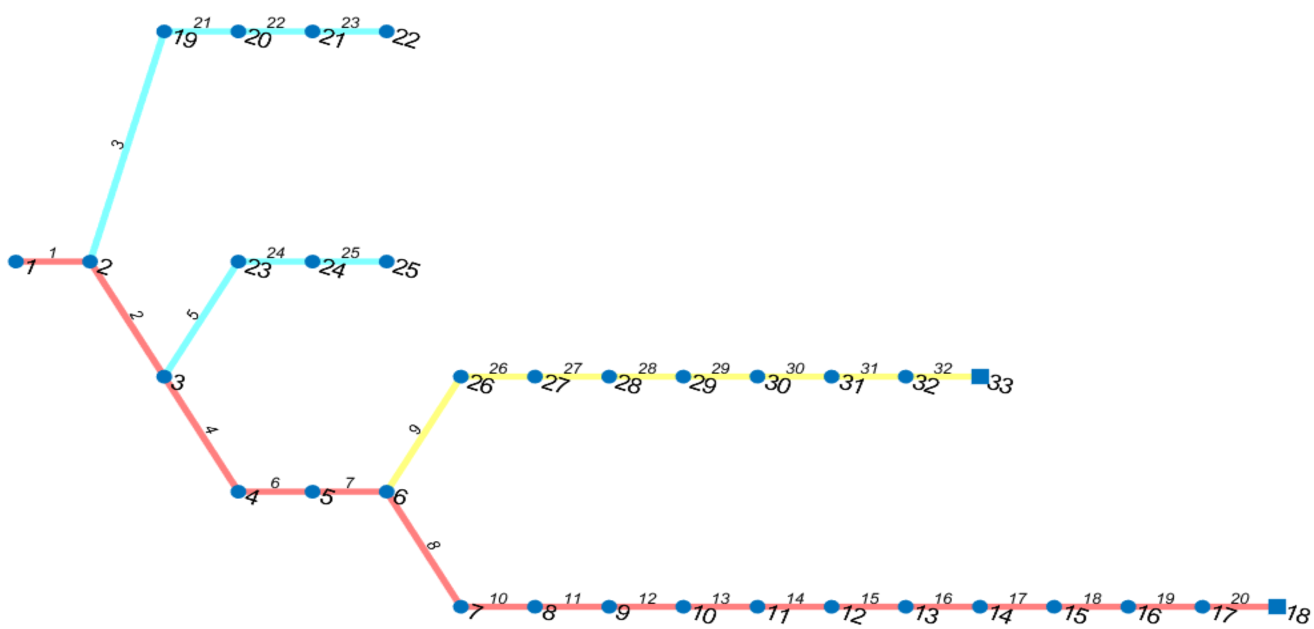

Other conditions such as loads and line impedances and … are like before.

Convergence criteria: $\max_{\text{PV Voltages}} \{ \left| |\overline{\boldsymbol{V}}_{sp}| - |\overline{\boldsymbol{V}}_{calc}| \right| \} < 0.01$ pu

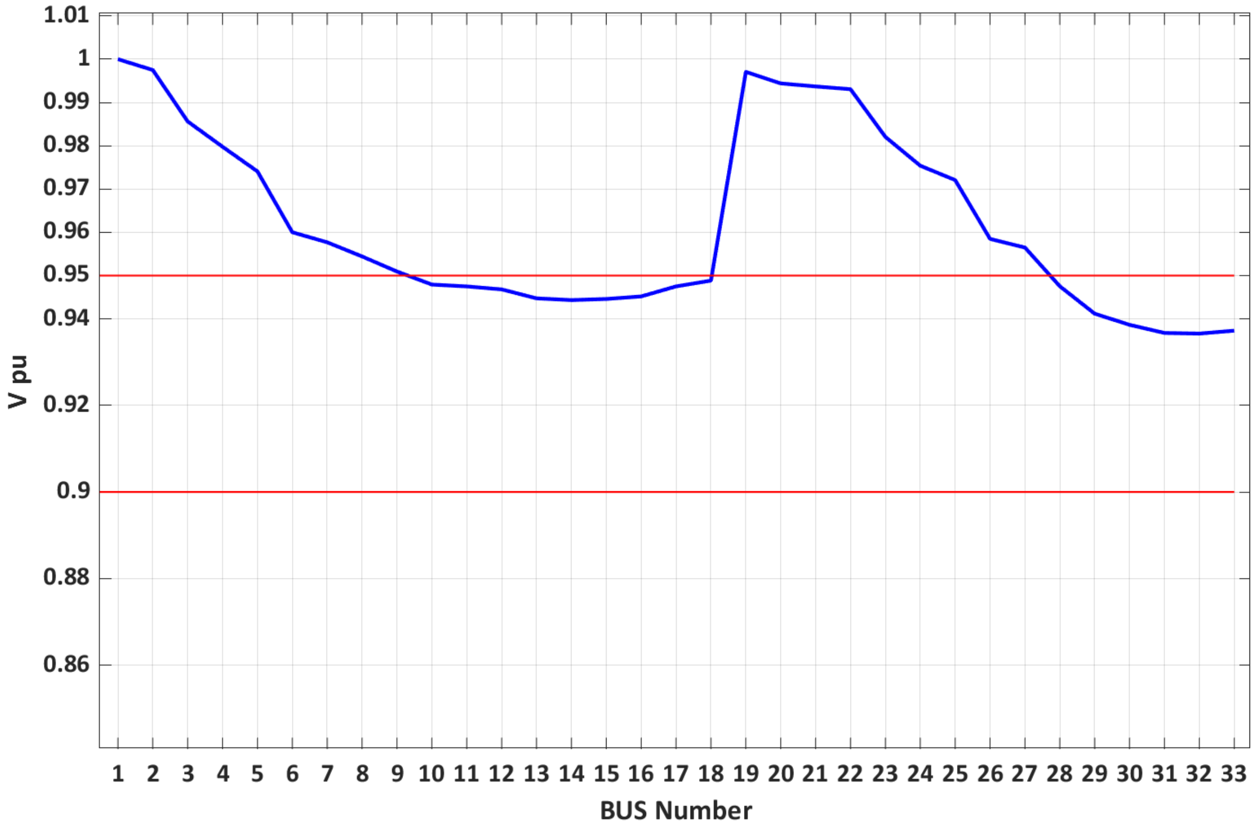
**Results:**

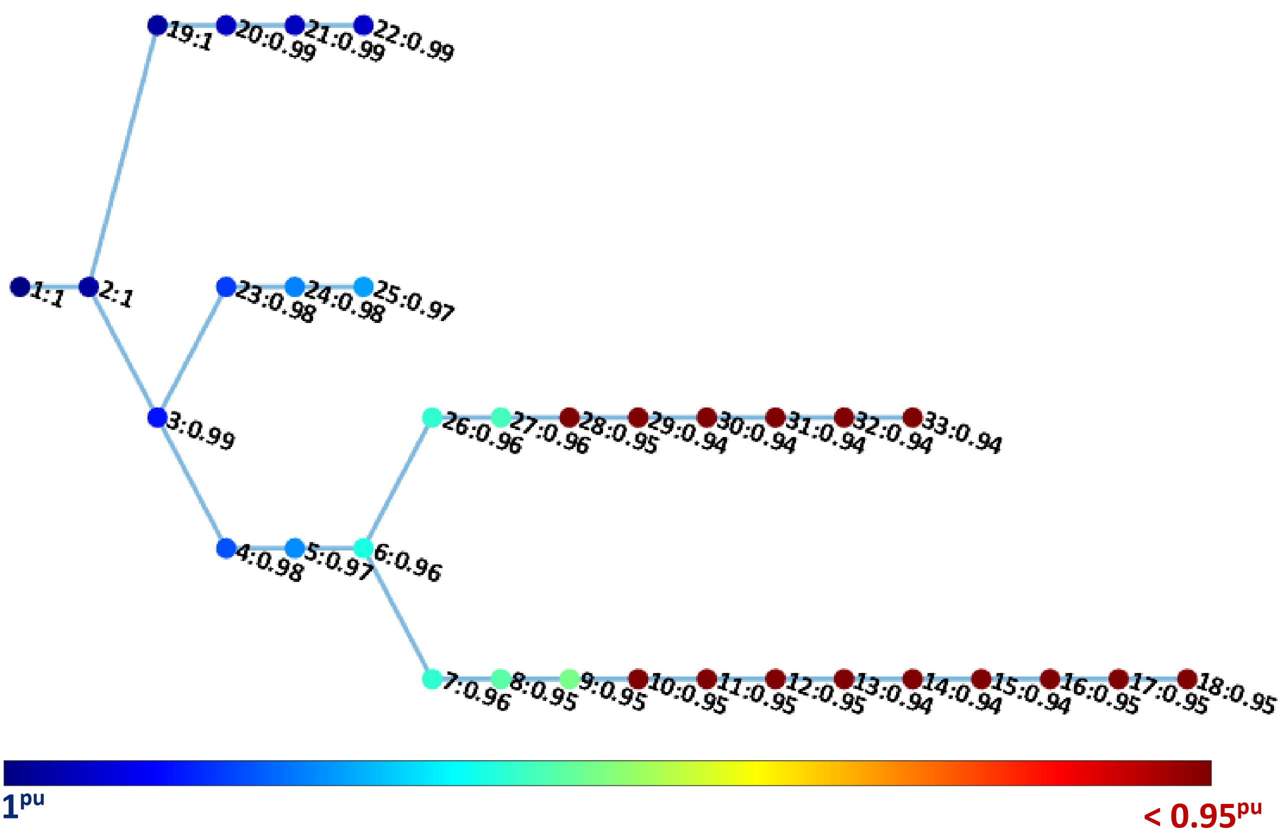
Algorithm converged after 3 iterations (of outer loop) and the output results are:

Network graph (PV buses are shown by square and their paths to the slack node are highlighted)**:**



## Voltage Profile:



**Final Voltages:**

| Bus NO | \|U\|(pu) | θ(°) | Q(pu) |
|---|---|---|---|
| 1 | 1 | 0 | 0 |
| 2 | 1 | 0.01 | 0.0600 |
| 3 | 0.99 | 0.07 | 0.0400 |
| 4 | 0.98 | 0.12 | 0.0800 |
| 5 | 0.97 | 0.17 | 0.0300 |
| 6 | 0.96 | 0.1 | 0.0200 |
| 7 | 0.96 | -0.08 | 0.1000 |
| 8 | 0.95 | -0.08 | 0.1000 |
| 9 | 0.95 | -0.15 | 0.0200 |
| 10 | 0.95 | -0.22 | 0.0200 |
| 11 | 0.95 | -0.22 | 0.0300 |
| 12 | 0.95 | -0.22 | 0.0350 |
| 13 | 0.94 | -0.31 | 0.0350 |
| 14 | 0.94 | -0.35 | 0.0800 |
| 15 | 0.94 | -0.38 | 0.0100 |
| 16 | 0.95 | -0.4 | 0.0200 |
| 17 | 0.95 | -0.4 | 0.0200 |
| 18 | 0.95 | -0.41 | -0.1500 |
| 19 | 1 | 0 | 0.0400 |
| 20 | 0.99 | -0.06 | 0 |
| 21 | 0.99 | -0.08 | 0.0400 |
| 22 | 0.99 | -0.1 | 0.0400 |
| 23 | 0.98 | 0.04 | 0.0500 |
| 24 | 0.98 | -0.05 | 0.2000 |
| 25 | 0.97 | -0.09 | 0.2000 |
| 26 | 0.96 | 0.13 | 0.0250 |
| 27 | 0.96 | 0.18 | 0.0250 |
| 28 | 0.95 | 0.28 | 0.0200 |
| 29 | 0.94 | 0.36 | 0.0700 |
| 30 | 0.94 | 0.46 | 0.6000 |
| 31 | 0.94 | 0.4 | 0.0700 |
| 32 | 0.94 | 0.39 | 0.1000 |
| 33 | 0.94 | 0.41 | -0.1000 |

-------------------------------------------------------------------

*Total Power Loss in lines (kW) = 126.6627*

*Total Reactive Power Consumed by Lines (kVAr) =83.0662*

# Conclusion

**It is obvious and was expected that in presence of DGs the voltage profile improves significantly however the voltage magnitude at nodes with voltage controlled DGs couldn’t reach to the set point of one per unit due to the limitations of their capability to provide sufficient reactive power.**

# References (files are in the Report folder)


[1]. Baran ME, Wu FF (1989) Network reconfiguration in distribution systems for loss reduction and load balancing. IEEE Power Eng Rev 9(4):101–102 (FILE)ref1.pdf

[۲]. دکتر کریمی، علی . (۱۴۰۱) . جزوه توزیع انرژی الکتریکی . دانشگاه کاشان.

[3]. Shrivastava, C., Gupta, M., Koshti, A., & Scholar, P. G. (2015). Review of forward & backward sweep method for load flow analysis of radial distribution system. International journal of advanced research in electrical, electronics and instrumentation engineering, 4(6), 5595-5599. (FILE)ref3.pdf

[4]. Kawambwa, S., Mwifunyi, R., Mnyanghwalo, D. et al. An improved backward/forward sweep power flow method based on network tree depth for radial distribution systems. Journal of Electrical Systems and Inf Technol 8, 7 (2021). https://doi.org/10.1186/s43067-021-00031-0 (FILE)ref4.pdf

[۵]. عسکری ، صدیقه . (۱۳۹۵) . پخش بار در شبکه‌های توزیع در حضور منابع تولید پراکنده با در نظرگرفتن مدل بار . پایان‌نامه کارشناسی ارشد . دانشگاه صنعتی شاهرود . (FILE)**ref5.pdf**

[۶]. ایزانلو، علی،۱۳۹۳،پخش بار در شبکه توزیع به روش جاروب رفت و برگشت در حضور منابع تولید پراکنده،اولین کنفرانس سراسری توسعه محوری مهندسی عمران، معماری،برق و مکانیک ایران،گرگان،https://civilica.com/doc/325619 (FILE)ref6(PAYED)(C).pdf

# Appendix

## MATLAB Code For B/FS:

```
clc
clear
close all
%% Reading Loads & Lines Data
Load_Data = readmatrix('bus33.xls');          %Reading Load Data (P & Q in kW & kVAr)
Line_Data = readmatrix('branch33.xls');       %Reading Line Data  (R & X in Ohms)
%% Setting Initial Parameters
N=10;                       % N: Maximum Number of Iterations
Sb=1;                       %  S_base (MVA)
Ub=12.66;                   % U_base (kV)   (line2line)
e=0.01;                     % Epsilon -> Convergence criteria : max(|vnew-vold|)<e
%% Evaluating Zbase
Zb=(Ub^2)/Sb;          % Z_base
%%
br_no=length(Line_Data);              % Number of Branches
bus_no=length(Load_Data);           % Number of Buses
%% Per unit Values
R = Line_Data(:,4)./Zb;
X = Line_Data(:,5)./Zb;
P = ((Load_Data(:,2))./(1000*Sb));      % P in kw & Sb in MVA
Q = ((Load_Data(:,3))./(1000*Sb));     % Q in kVAr & Sb in MVA
%% Forming Connection Matrix (C(branches,buses))
s_buses=Line_Data(:,2);                 %Sending buses
r_buses=Line_Data(:,3);                 %Recieving buses
C=zeros(br_no,bus_no);
for branch=1:br_no
    C(branch,s_buses(branch))=-1;     %Sending bus : -1
    C(branch,r_buses(branch))=+1;    %Recieving bus : +1
end
%% Print Values
% (1:no)
% [(1:br)',C]
%% Determining End Nodes
endnode=(find(sum(C)==1))';         %End Buses indexes
%% Print Values
%endnode
%% Determining the Path of each Radius(:Routes from first Bus to each Endnode)
h=length(endnode);                                           % h= Number of Radiuses
g=[0];                                                        % EndBuses Path to the first Bus
for route_no=1:h
    rnode=endnode(route_no);                            % Recieving node
    snode=s_buses(r_buses==rnode);                   %Sending Node
    g(route_no,1)=rnode;
    g(route_no,2)=snode;
    j=2;
    while(snode~=1)
        rnode=snode;
        g(route_no,j)=rnode;
        snode=s_buses(r_buses==rnode);
        g(route_no,j+1)=snode;
        j=j+1;
    end
end
%% Print Values
%% Sorting Radius Matrix Elements
```

```
gs=g;  %gs is sorted form of g
for i=1:length(endnode)
    rout=sort(nonzeros(g(i,:)));
    gs(i,1:length(rout))=rout;
end
%% print Values
%gs;
%% Forming Route Matrices for applications
gb=g;
mr=1;                                    %MainRoute_Row_index in g
for i=1:size(gb,1)
    if length(nonzeros(gb(i,:)))>length(nonzeros(gb(mr,:)))
        mr=i;
    end
end
temp=gb(1,:);          %Main Route placed in at the 1st row
gb(1,:)=gb(mr,:);
gb(mr,:)=temp;
for i=1:size(gb,1)
    for j=1:size(gb,2)
        n=gb(i,j);
        for ii=((i+1):size(gb,1))
            for jj=1:(size(gb,2)-1)
                if gb(ii,jj)==n
                    gb(ii,jj+1)=0;
                end
            end
        end
    end
end
gv=gb;                 %gv matrix will be used for KVL in Forward steps
for i=1:length(endnode)
    rout=sort(nonzeros(gb(i,:)));
    gv(i,1:length(rout))=rout;
end
sc=zeros(size(gb,1),1);
for j=1:size(gb,1)
    for i=1:size(gb,1)-1
        a=length(nonzeros(gb(i,:)));
        b=length(nonzeros(gb(i+1,:)));
        if a>b
            t=gb(i,:);
            gb(i,:)=gb(i+1,:);
            gb(i+1,:)=t;
        end
    end
end
g=gb;
%% initial guess
v = ones(bus_no,1);          %Bus Voltages vector Initialization (complex value) (flat initial guess)
I = zeros(br_no,1);              %Branches' Current vector Initialization %C:(br,bus)
%% Iteration Loop
for ni=1:N
    %% Backward Step
    vold=v;
    LC = conj(complex(P,Q)./v);     %Bus Load Currents  vector
    for r=1:route_no
        for i=1:size(g,2)-1
            b=g(r,i);
            if b==0
                break;
            end
            if sum(C(:,b))==1
                I(C(:,b)==1)=LC(b);
```

```
            LC(g(r,i+1))=LC(g(r,i+1))+LC(b);
          else
            if g(r,i+1)==1
              I(C(:,b)==1)=LC(b);
              break;
            else
              I(C(:,b)==1)=LC(b);
              if g(r,i+1)~=0
                LC(g(r,i+1))=LC(g(r,i+1))+LC(b);
              end
            end
          end
        end
      end
    %% Forward Step
    for r=1:route_no
      for i=1:size(gv,2)-1
        if gv(r,i+1)==0
          continue;
        end
        b= find(C(:,gv(r,i+1))==1);
        v(gv(r,i+1))=v(gv(r,i))-complex(R(b),X(b))*I(b);
        %fprintf("V("+num2str(gv(r,i+1))+")=V("+num2str(gv(r,i))+")-zI("+num2str(b)+")\n");
      end
    end
    vnew=v;
    if max(abs(vnew-vold))<e
      fprintf("Algorithm Converged!\nNumber of Iterations="+num2str(ni)+"\n-----------------------------------------\n")
      break;
    end
end
%% Print Calculated Values
vbp=[abs(v),angle(v).*(180/pi)];
vbp2=[((1:bus_no)'),abs(v),angle(v).*(180/pi)];
h={'Bus NO','|U|(pu)','?(°)'};
T = array2table(vbp2,'VariableNames',h);
T2 = array2table(round(vbp2,2),'VariableNames',h);
%% Print
fprintf("Final Voltages:\n")
disp(T2)
f=figure;
t=uitable(f,'data',vbp2,'columnname',h);
%% Plot Voltage Profile
f2=figure;
p=plot(vbp2(:,1),vbp2(:,2),'-b','LineWidth',2);
hold on
plot([0 33],[0.95 0.95],'-r','LineWidth',1);
plot([0 33],[0.9 0.9],'-r','LineWidth',1);
hold off
xlim([0 33])
ylim([0.85 1.02])
yaxes=[[0.86:0.02:0.95],[0.95:0.01:1.2]];
yticks(yaxes)
xticks(0:33)
xlabel("BUS Number")
ylabel("V_{Line} pu")
grid on
%%
Ibrpu=[abs(I) angle(I)*180/pi];     % Branches' Currents in pu magnitude and angle
%% Line Consumed Power Calculation
PL  = R.*(abs(I).^2);       % Active Power (pu) Consumed by each Line
QL = X.*(abs(I).^2);        %Reactive Power (pu) Consumed by each Line
PLkW=(PL)*Sb*1000;
QLkVAr=(QL)*Sb*1000;
```

```
PLt=sum(PL)*Sb*1000;    %Total kW consumed by lines (total power loss)
QLt=sum(QL)*Sb*1000;    %Total kVAr consumed by lines
%% Print
fprintf('--------------------------------------------------------------------\n')
fprintf("Total Power Loss in lines (kW) ="+num2str(PLt)+'\n');
fprintf("Total ReactivePower Consumed by Lines (kVAr) ="+num2str(QLt)+'\n');
%% Forming Network Graph
s=Line_Data(:,2);
t=Line_Data(:,3);
wp=round(Ibrpu,1);
w=wp(:,1);
for i=1:bus_no
    eq(i,1)=':';
end
names=string((1:bus_no)')+string(char(eq))+string(round(vbp(:,1),2));
NG=graph(s,t);
NGn=graph(s,t,w,names);
G = digraph(s,t,w);
figure
hold on
h8 = plot(NGn,'Layout','layered','Direction','right','MarkerSize',8,'LineStyle','-','LineWidth',2);
hold off
CC=(1-round(vbp(:,1),3)).*100;
CC(CC>5.001)=10;
NGn.Nodes.NodeColors=CC;
h8.NodeCData = NGn.Nodes.NodeColors;
colorbar('Ticks',[0 10],'TickLabels',{'1pu','<0.95 pu'});
colormap(jet)
```

## MATLAB Code for BCBB/FS:

```
clc
clear
close all
%% Reading Loads & Lines Data
Load_Data = readmatrix('bus33.xls');          %Reading Load Data (P & Q in kW & kVAr)
Line_Data = readmatrix('branch33.xls');       %Reading Line Data  (R & X in Ohms)
DG_Data= readmatrix('DG_BUS.xlsx');
%% Setting Initial Parameters
N1=10;                    % N: Maximum Number of Iterations of Inner loop
N2=10;                    % N2: Maximum Number of Iterations
Sb=1;                     %  S_base (MVA)
Ub=12.66;                 % U_base (kV)   (line2line)
e=0.01;                   % Epsilon -> Convergence criteria : max(||vsp_pu|-|vcal_pu||)<e
e1=0.01;                  % Inner loop Convergence criteria: max(|vsp_old-vcal_new|)<e1
%% Processing const.P-const.Q DG Data in Load_Data Matrix
pq_dg=find(DG_Data(:,4)==0);
Load_Data([DG_Data(pq_dg,1)],[2 3])=Load_Data([DG_Data(pq_dg,1)],[2 3])-DG_Data(pq_dg,[2 3]);
%% Processing const.P-const.V DG Data in Load_Data Matrix and Extracting data
pv_dg=find(DG_Data(:,4)==1); % pv buses indices in DG_Data
Load_Data([DG_Data(pv_dg,1)],[2])=Load_Data([DG_Data(pv_dg,1)],[2])-DG_Data(pv_dg,2);
pv_bus_sp=DG_Data(pv_dg,[1 5]); %DG Voltage set points (pu)
pv_bus_no=DG_Data(pv_dg,1);
pv_bus_maxgen_q=DG_Data(pv_dg,[1 6]);  % Maximum Generated Q (pu)
pv_bus_maxcon_q=DG_Data(pv_dg,[1 7]);  % Maximum Consumed Q (pu)
pv_bus_maxgen_q(:,2)=pv_bus_maxgen_q(:,2)./(1000*Sb);
pv_bus_maxcon_q(:,2)=pv_bus_maxcon_q(:,2)./(1000*Sb);
pv_qm_gen=containers.Map(pv_bus_maxgen_q(:,1),pv_bus_maxgen_q(:,2));
pv_qm_con=containers.Map(pv_bus_maxcon_q(:,1),pv_bus_maxcon_q(:,2));
%% Evaluating Zbase
Zb=(Ub^2)/Sb;         % Z_base
%%
br_no=length(Line_Data);             % Number of Branches
bus_no=length(Load_Data);            % Number of Buses
%% Per unit Values
R = Line_Data(:,4)./Zb;
X = Line_Data(:,5)./Zb;
P = ((Load_Data(:,2))./(1000*Sb));      % P in kw & Sb in MVA
Q = ((Load_Data(:,3))./(1000*Sb));      % Q in kVAr & Sb in MVA
%% Forming Connection Matrix (C(branches,buses))
s_buses=Line_Data(:,2);              %Sending buses
r_buses=Line_Data(:,3);              %Recieving buses
C=zeros(br_no,bus_no);
for branch=1:br_no
   C(branch,s_buses(branch))=-1;     %Sending bus : -1
   C(branch,r_buses(branch))=+1;     %Recieving bus : +1
end
%% Formin Reactance Sesetivity Matrix

s=Line_Data(:,2);      % Sending Buses
r=Line_Data(:,3);      % Receiving Buses
wx=[Line_Data(:,1),X]; % line number   line X(pu)
NG=graph(s,r,wx(:,2));
figure
gp=plot(NG,'EdgeLabel',wx(:,1),'Layout','layered','Direction','right','LineWidth',4,'EdgeColor','c','MarkerSize',8,'Marker','o','NodeFontSiz
e',14,'EdgeFontSize',10);
highlight(gp,pv_bus_no,'Marker','s','MarkerSize',12)
X_s=zeros(length(pv_bus_no));
pv_path_bus=zeros(length(pv_dg),1);
pv_d_s=zeros(length(pv_dg),1);
pv_path_line=zeros(length(pv_dg),1);
for i=1:length(pv_dg)
   [a,b,c] = shortestpath(NG,1,pv_bus_no(i),'Method','unweighted');
```

```
    pv_path_bus(i,1:length(a))=a;
    pv_d_s(i)=b;
    pv_path_line(i,1:length(c))=c;
    singlepath=nonzeros(pv_path_line(i,:));
    X_s(i,i)=sum(wx(singlepath,2));
    colora=['c','y','r','b','g','m'];
    highlight(gp,a,'EdgeColor',colora(mode(i,4)+1))
end
for i=1:length(pv_bus_no)
    for j=i+1:length(pv_bus_no)
        %fprintf("i="+num2str(i)+','+'j='+num2str(j)+'\n');
        temp=pv_path_line(i,:)-pv_path_line(j,:);
        for ii=1:length(temp)
            if temp(ii)~=0
                break;
            end
        end
        ii=ii-1;
        commonpath=pv_path_line(i,1:ii);
        X_s(i,j)=sum(wx(commonpath,2));
        X_s(j,i)=X_s(i,j);
    end
end
%% Determining End Nodes
endnode=(find(sum(C)==1))';         %End Buses indexes
%% Print Values
%endnode
%% Determining the Path of each Radius(:Routes from first Bus to each Endnode)
h=length(endnode);                                   % h= Number of Radiuses
g=[0];                                                % EndBuses Path to the first Bus
for route_no=1:h
    rnode=endnode(route_no);                         % Recieving node
    snode=s_buses(r_buses==rnode);                   %Sending Node
    g(route_no,1)=rnode;
    g(route_no,2)=snode;
    j=2;
    while(snode~=1)
        rnode=snode;
        g(route_no,j)=rnode;
        snode=s_buses(r_buses==rnode);
        g(route_no,j+1)=snode;
        j=j+1;
    end
end
%% Print Values
%g
%% Sorting Radius Matrix Elements
gs=g;  %gs is sorted form of g
for i=1:length(endnode)
    rout=sort(nonzeros(g(i,:)));
    gs(i,1:length(rout))=rout;
end
%% print Values
%gs;
%% Forming Route Matrices for applications
gb=g;
mr=1;                                    %MainRoute_Row_index in g
for i=1:size(gb,1)
    if length(nonzeros(gb(i,:)))>length(nonzeros(gb(mr,:)))
        mr=i;
    end
end
temp=gb(1,:);          %Main Route placed in at the 1st row
gb(1,:)=gb(mr,:);
```

```
gb(mr,:)=temp;
for i=1:size(gb,1)
    for j=1:size(gb,2)
        n=gb(i,j);
        for ii=((i+1):size(gb,1))
            for jj=1:(size(gb,2)-1)
                if gb(ii,jj)==n
                    gb(ii,jj+1)=0;
                end
            end
        end
    end
end
gv=gb;                 %gv matrix will be used for KVL in Forward steps
for i=1:length(endnode)
    rout=sort(nonzeros(gb(i,:)));
    gv(i,1:length(rout))=rout;
end
sc=zeros(size(gb,1),1);
for j=1:size(gb,1)
    for i=1:size(gb,1)-1
        a=length(nonzeros(gb(i,:)));
        b=length(nonzeros(gb(i+1,:)));
        if a>b
            t=gb(i,:);
            gb(i,:)=gb(i+1,:);
            gb(i+1,:)=t;
        end
    end
end
g=gb;
%% initial guess
v = ones(bus_no,1);                          %Bus Voltages vector Initialization (complex value) (flat initial guess)
v(pv_bus_sp(:,1))=pv_bus_sp(:,2);            % PV Buses initial guess according to their set points
I = zeros(br_no,1);                          %Branches' Current vector Initialization %C:(br,bus)
%% outer Iteration Loop
for oc=1:N2
    v_pv_old_abs=abs(v(pv_bus_no));
    %% Inner Iteration Loop (BFS)
    for ni=1:N1
    %% Backward Step
    vold=v;
    LC = conj(complex(P,Q)./v);     %Bus Load Currents  vector
    for r=1:route_no
        for i=1:size(g,2)-1
            b=g(r,i);
            if b==0
                break;
            end
            if sum(C(:,b))==1
                I(C(:,b)==1)=LC(b);
                LC(g(r,i+1))=LC(g(r,i+1))+LC(b);
            else
                if g(r,i+1)==1
                    I(C(:,b)==1)=LC(b);
                    break;
                else
                    I(C(:,b)==1)=LC(b);
                    if g(r,i+1)~=0
                        LC(g(r,i+1))=LC(g(r,i+1))+LC(b);
                    end
                end
            end
        end
```

```
    end
    %% Forward Step
    for r=1:route_no
        for i=1:size(gv,2)-1
            if gv(r,i+1)==0
                continue;
            end
            b= find(C(:,gv(r,i+1))==1);
            v(gv(r,i+1))=v(gv(r,i))-complex(R(b),X(b))*I(b);
            %fprintf("V("+num2str(gv(r,i+1))+")=V("+num2str(gv(r,i))+")-zI("+num2str(b)+")\n");
        end
    end
    vnew=v;
    if max(abs(vnew-vold))<e1
        %fprintf("Algorithm Converged!\nNumber of Iterations="+num2str(ni)+"\n-----------------------------------------\n")
        break;
    end
    end
    v_pv_new_abs=abs(v(pv_bus_no));
    if max(abs(v_pv_new_abs-v_pv_old_abs))<e
        fprintf("Algorithm Converged!\nNumber of Iterations(outer loop)="+num2str(oc)+"\n-----------------------------------------\n")
        break;
    end
    %% Updating Q Values
    DV=abs(pv_bus_sp(:,2))-abs(v(pv_bus_sp(:,1)));             %Calculating DeltaV Matrix
    DQ=X_s\DV;                                                 %Calculating DeltaV Matrix
    Q(pv_bus_no,1)=Q(pv_bus_no,1)-sign(DV).*DQ;                %updating Q values (pu)
    for qqi=1:length(pv_bus_no)                                %Checking Q limits (pu)
        qq=pv_bus_no(qqi);
        if(Q(qq,1)<-abs(pv_qm_gen(qq)))
            Q(qq,1)=-abs(pv_qm_gen(qq));
        elseif(Q(qq,1)>abs(pv_qm_con(qq)))
            Q(qq,1)=abs(pv_qm_con(qq));
        end
    end
end
%% Print Calculated Values
vbp=[abs(v),angle(v).*(180/pi)];
vbp2=[((1:bus_no)'),abs(v),angle(v).*(180/pi)];
h={'Bus NO','|U|(pu)','?(°)'};
T = array2table(vbp2,'VariableNames',h);
T2 = array2table(round(vbp2,2),'VariableNames',h);
%% Print
fprintf("Final Voltages:\n")
disp(T2)
f=figure;
t=uitable(f,'data',vbp2,'columnname',h);
%% Plot Voltage Profile
f2=figure;
p=plot(vbp2(:,1),vbp2(:,2),'-b','LineWidth',2);
hold on
plot([0 bus_no],[0.95 0.95],'-r','LineWidth',1);
plot([0 bus_no],[0.9 0.9],'-r','LineWidth',1);
hold off
xlim([0 bus_no])
ylim([0.85 1.02])
yaxes=[[0.86:0.02:0.95],[0.95:0.01:1.2]];
yticks(yaxes)
xticks(0:bus_no)
xlabel("BUS Number")
ylabel("V_{Line} pu")
grid on
%%
Ibrpu=[abs(I) angle(I)*180/pi];     % Branches' Currents in pu magnitude and angle
```

```
%% Line Consumed Power Calculation
PL  = R.*(abs(I).^2);       % Active Power (pu) Consumed by each Line
QL = X.*(abs(I).^2);        %Reactive Power (pu) Consumed by each Line
PLkW=(PL)*Sb*1000;
QLkVAr=(QL)*Sb*1000;
PLt=sum(PL)*Sb*1000;    %Total kW consumed by lines (total power loss)
QLt=sum(QL)*Sb*1000;    %Total kVAr consumed by lines
%% Print
fprintf('--------------------------------------------------------------------\n')
fprintf("Total Power Loss in lines (kW) ="+num2str(PLt)+'\n');
fprintf("Total ReactivePower Consumed by Lines (kVAr) ="+num2str(QLt)+'\n');
%% Forming Network Graph
s=Line_Data(:,2);
t=Line_Data(:,3);
wp=round(Ibrpu,1);
w=wp(:,1);
for i=1:bus_no
    eq(i,1)=':';
end
names=string((1:bus_no)')+string(char(eq))+string(round(vbp(:,1),2));
NG=graph(s,t);
NGn=graph(s,t,w,names);
G = digraph(s,t,w);
figure
hold on
h8 = plot(NGn,'Layout','layered','Direction','right','MarkerSize',8,'LineStyle','-','LineWidth',2,'NodeFontSize',12);
hold off
CC=(1-round(vbp(:,1),3)).*100;
CC(CC>5.001)=10;
NGn.Nodes.NodeColors=CC;
h8.NodeCData = NGn.Nodes.NodeColors;
colorbar('Ticks',[0 10],'TickLabels',{'1pu','<0.95 pu'});
colormap(jet)
figure
plot(NG,'Layout','layered','Direction','right')
```

**Thank you for your time.**