RAO PAHLAD SINGH COLLEGE OF ENGG. & TECH.
Approved by AICTE / Govt. of India & Affiliated to I.G.U., Meerpur
Balana, Mohindergarh, Haryana - 123029
Telephone: 01285-241431 Fax: 241434
E-mail: rpsbalana@gmail.com, Website: www.rpsinstitutions.org

POWER SYSTEM-II LABORATORY

[CODE: LC-EE-304]

Made by:

Er. Vishal Kumar Mittal
Asst. Prof.
(EE Department)

RAO PAHLAD SINGH COLLEGE OF ENGG. & TECH.
Approved by AICTE / Govt. of India & Affiliated to I.G.U., Meerpur
Balana, Mohindergarh, Haryana - 123029
Telephone: 01285-241431 Fax: 241434
E-mail: rpsbalana@gmail.com, Website: www.rpsinstitutions.org

Table of Contents

EXPERIMENT 1

Aim: To develop algorithm and to develop the computer program for the formation of the Y_{Bus} of a generalized network.

Requirements: Computer, MATLAB

Theory: Y_{Bus} is the bus admittance matrix, it relates the current injections into the buses with the bus voltages. Y_{Bus} is a symmetrical matrix of order $n \times n$, where n is the number of buses in the network. Y_{Bus} can be obtained using singular transformation as related in eq. 1.

$$Y_{Bus} = A^T Y_{prim} A \qquad \ldots\ldots (1)$$

Where Y_{prim} is the primitive admittance matrix which is a square matrix of order $b \times b$, where b is the number of branches in the network, and A is the element to bus incidence matrix of order $b \times n$. Bus admittance matrix is used to carry out the load flow study.

Case Study:

Table 1.1: *Line data for the given network*

Line No.	From Bus	To Bus	R (pu)	X (pu)	Mutually coupled to line	Y (pu) mutual	Charging admittance (pu)
1	3	0	0	1.250	0	0	0
2	2	3	0	0.160	5	3.75j	0
3	1	2	0	0.125	0	0	0
4	1	4	0	0.400	0	0	0
5	1	3	0	0.160	2	3.75j	-0.16j
6	2	4	0	0.200	0	0	0
7	4	0	0	1.250	0	0	0

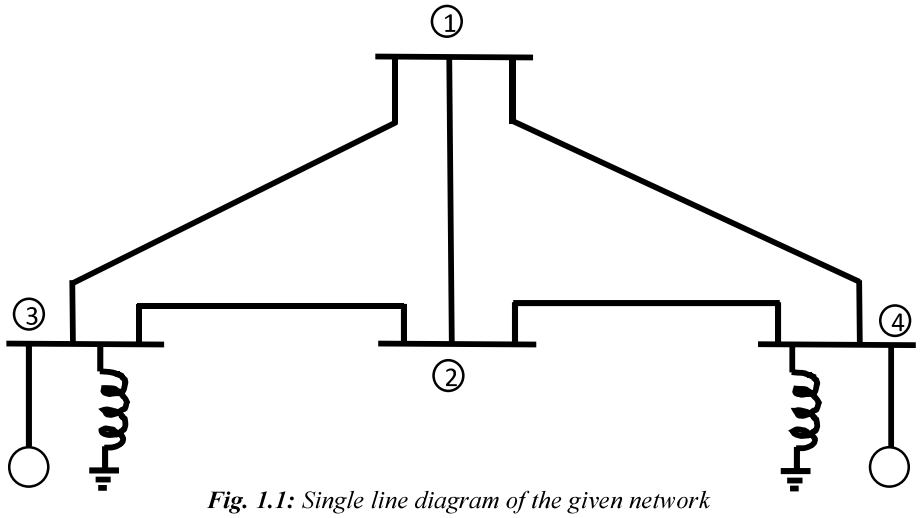


Fig. 1.1: *Single line diagram of the given network*

Algorithm:

1. Load the line data (branch impedances, mutual admittances and charging admittances).
2. Find no. of buses in the given system.
3. Find no. of buses in the given system.
4. Obtain primitive admittance matrix for the system.
5. Let the count for charging admittance be *n*.
6. For k^{th} line –
   - $Y_{primitive}^{new} = \begin{bmatrix} Y_{primitive}^{old} & colomn\ matrix\ of\ '0'\ of\ order\ 'k+n-1' \\ row\ matrix\ of\ '0'\ of\ order\ 'k+n-1' & 1/(k^{th}\ line\ impedance) \end{bmatrix}$
   - If charging admittance is present in k^{th} line (using π model of line) –
   - Increment charging admittance count *n* by 1.
   - Adding the half of the charging admittance of k^{th} line to i^{th} bus and updating $Y_{primitive}$ with it.
   - $Y_{primitive}^{new} = \begin{bmatrix} Y_{primitive}^{old} & colomn\ matrix\ of\ '0'\ of\ order\ 'k+n-1 \\ row\ matrix\ of\ '0'\ of\ order\ 'k+n-1' & (charging\ admittance\ of\ k^{th}\ line)/2 \end{bmatrix}$
   - Increment charging admittance count *n* by 1.
   - Adding other half of the charging admittance of k^{th} line to j^{th} bus and updating $Y_{primitive}$ with it.
   - $Y_{primitive}^{new} = \begin{bmatrix} Y_{primitive}^{old} & colomn\ matrix\ of\ '0'\ of\ order\ 'k+n-1 \\ row\ matrix\ of\ '0'\ of\ order\ 'k+n-1' & (charging\ admittance\ of\ k^{th}\ line)/2 \end{bmatrix}$
7. Repeat the above step for the given number of lines.
8. Update $Y_{primitive}$ with mutually coupled admittances –
   - Let charging admittance rows count in $Y_{primitive}$ is *n* up to k^{th} line.
     For k^{th} line if mutually coupling is there–
   - $[Y_{primitive}^{old}]_{(k+n)th\ line,coupling\ line} = given\ mutually\ coupled\ admittance$
   - If charging admittance is associated with the present line, then increment the charging admittance count *n* with 2.
9. Repeat above step for all the given lines.
10. Obtaining bus incidence matrix –
    - Finding no. of elements which is equal to the size of the primitive admittance matrix.
    - Initialize bus incidence matrix *A* with zeros of size (no. of elements, no. of buses).
    - Let charging admittance rows count in $Y_{primitive}$ is *n* up to k^{th} line.
      For k^{th} line –
    - If *from bus* is not equal to zero, then update matrix *A* at location (k+n, from bus) with '1'.
    - If *to bus* is not equal to zero, then update matrix *A* at location (k+n, to bus) with '-1'.
    - If charging admittance is associated with the present line, then increment the charging admittance count *n* with 1, and update matrix *A* at location (k+n, from bus) with '1'.
    - Again, increment the charging admittance count *n* with 1, and update matrix *A* at location (k+n, to bus) with '1'.
11. Repeat above step for all the given lines.
12. Compute $Y_{bus} = A^T Y_{primitive} A$.

```
%--------------YBus formulation using singular transformation--------------

%Data:
%-------------------------------------------------------------------------
%                        |   Line     |Mutually|
%       Line |From| To   | R      X   |coupled | y pu |  Charging    |
%       no.  |Bus | Bus  | pu     pu  |to line |mutual| admittance pu|
%-------------------------------------------------------------------------

ydata=[ 1      3     0    0    1.250      0       0          0j;
        2      2     3    0    0.160      5    1j*3.75       0j;
        3      1     2    0    0.125      0       0          0j;
        4      1     4    0    0.400      0       0          0j;
        5      1     3    0    0.160      2    1j*3.75       -1j*0.16;
        6      2     4    0    0.200      0       0          0j;
        7      4     0    0    1.250      0       0          0j];

%-------------------------------------------------------------------------
[Ybus,A,Yprim]=Y_BUS(ydata);

%-------------------------------------------------------------------------
%---------------------Saving the Ybus in a text file----------------------
folder_path='C:\Users\VISHAL MITTAL\Desktop\PS_II Lab\';
fid=fopen(strcat(folder_path,'Exp_1.txt'),'w+');
fprintf(fid,'%s \n\n','The Ybus solution obtained by Vishal Mittal:');

Yprim_str=num2str(Yprim,5);
A_str=num2str(A,3);
Ybus_str=num2str(Ybus,5);
fprintf(fid,'%s \n\n','Primitive Admittance Matrix');
for k=1:1:size(Yprim_str,1)
    fprintf(fid,'%s',Yprim_str(k,:));
    fprintf(fid,'\n');
end
fprintf(fid,'\n\n\n');
fprintf(fid,'%s \n\n','Bus Incidence Matrix');
for k=1:1:size(A_str,1)
    fprintf(fid,'%s',A_str(k,:));
    fprintf(fid,'\n');
end
fprintf(fid,'\n\n\n');
fprintf(fid,'%s \n\n','Bus Admittance Matrix');
for k=1:1:size(Ybus_str,1)
    fprintf(fid,'%s',Ybus_str(k,:));
    fprintf(fid,'\n');
end
fclose(fid);
```

```
 1
 2 %-------------------------Function to obtain-------------------------------
 3 %--------------YBus formulation using singular transformation--------------
 4
 5 %Line Data Format for 'ydata' variable:
 6 %Note: While writing Ymutual and Charging admittances: write 'j' explicitly
 7 %-----------------------------------------------------------------------
 8 %                        |   Line     |Mutually|
 9 %       Line |From| To   | R      X   |coupled | y pu |  Charging    |
10 %       no.  |Bus | Bus | pu     pu  |to line |mutual| admittance pu|
11 %-----------------------------------------------------------------------
12
13 function [Ybus,A,Yprim]=Y_BUS(ydata)
14     Nbus=max(max(ydata(:,2)),max(ydata(:,3)));
15
16     Nline=size(ydata,1);
17     Yprim=[];
18
19     %updating Yprim with line admittances and self admittances
20     n=0;
21     for k=1:1:Nline
22         Yprim=[Yprim              zeros(k+n-1,1);
23                zeros(1,k+n-1)     1/(ydata(k,4)+1j*ydata(k,5))];
24
25         if(ydata(k,8)~=0)
26             n=n+1;
27             Yprim=[Yprim            zeros(k+n-1,1);
28                    zeros(1,k+n-1)   ydata(k,8)/2   ];
29             n=n+1;
30             Yprim=[Yprim            zeros(k+n-1,1);
31                    zeros(1,k+n-1)   ydata(k,8)/2   ];
32         end
33     end
34
35     %updating Yprim with mutually coupled admittances
36     n=0;
37     for k=1:1:Nline
38         if(ydata(k,6)~=0)
39             Yprim(k+n,ydata(k,6))=ydata(k,7);
40         end
41         if(ydata(k,8)~=0)
42             n=n+2;
43         end
44     end
45
46     %forming bus incidence matrix
47     Nelmnts=size(Yprim,1);
48     A=zeros(Nelmnts,Nbus);
49     n=0;
50     for k=1:1:Nline
51         if(ydata(k,2)~=0)
```

```
52              A(k+n,ydata(k,2))=1;
53          end
54          if(ydata(k,3)~=0)
55              A(k+n,ydata(k,3))=-1;
56          end
57          if(ydata(k,8)~=0)
58              n=n+1;
59              A(k+n,ydata(k,2))=1;
60              n=n+1;
61              A(k+n,ydata(k,3))=1;
62          end
63      end
64
65      Ybus=(A'*Yprim)*A;
```

The Ybus solution obtained by Vishal Mittal:

Primitive Admittance Matrix

0-0.8i	0+0i	0+0i	0+0i	0+0i	0+0i	0+0i	0+0i	0+0i
0+0i	0-6.25i	0+0i	0+0i	0+3.75i	0+0i	0+0i	0+0i	0+0i
0+0i	0+0i	0-8i	0+0i	0+0i	0+0i	0+0i	0+0i	0+0i
0+0i	0+0i	0+0i	0-2.5i	0+0i	0+0i	0+0i	0+0i	0+0i
0+0i	0+3.75i	0+0i	0+0i	0-6.25i	0+0i	0+0i	0+0i	0+0i
0+0i	0+0i	0+0i	0+0i	0+0i	0-0.08i	0+0i	0+0i	0+0i
0+0i	0+0i	0+0i	0+0i	0+0i	0+0i	0-0.08i	0+0i	0+0i
0+0i	0+0i	0+0i	0+0i	0+0i	0+0i	0+0i	0-5i	0+0i
0+0i	0+0i	0+0i	0+0i	0+0i	0+0i	0+0i	0+0i	0-0.8i

Bus Incidence Matrix

0	0	1	0
0	1	-1	0
1	-1	0	0
1	0	0	-1
1	0	-1	0
1	0	0	0
0	0	1	0
0	1	0	-1
0	0	0	1

Bus Admittance Matrix

0-16.83i	0+11.75i	0+2.5i	0+2.5i
0+11.75i	0-19.25i	0+2.5i	0+5i
0+2.5i	0+2.5i	0-5.88i	0+0i
0+2.5i	0+5i	0+0i	0-8.3i

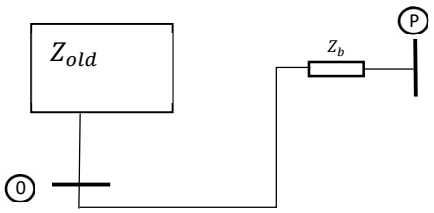
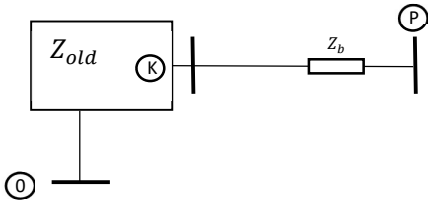
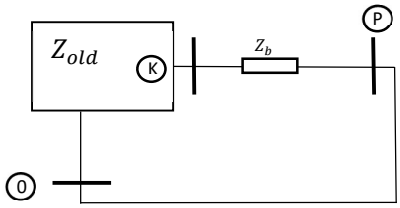
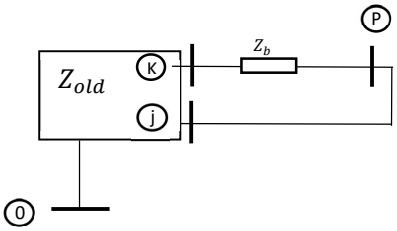
EXPERIMENT 2

Aim: To develop an algorithm and to develop the computer program for the formation of the Z_{Bus} of a generalized network.

Requirements: Computer, MATLAB

Theory: Z_{Bus} is the bus impedance matrix, it relates the bus voltages with the current injections at the buses. Z_{Bus} is a symmetrical matrix of order $n \times n$, where n is the number of buses in the network. To construct Z_{Bus} , four-modification algorithms are to be used. Single-single branch is to be taken to modify the old Z_{Bus}. This branch of impedance Z_b can be connected between:

a. New bus to reference node
b. New bus to existing bus
c. Existing bus to reference node
d. Existing bus to other existing bus

a		$\begin{bmatrix} \boldsymbol{Z_{old}} & \boldsymbol{0} \\ \boldsymbol{0} \quad \ldots & \boldsymbol{Z_b} \end{bmatrix}$
b		$\begin{bmatrix} \boldsymbol{Z_{old}} & col\ \boldsymbol{k}\ of\ Z_{old} \\ row\ \boldsymbol{k}\ of\ Z_{old} & \boldsymbol{Z_{kk} + Z_b} \end{bmatrix}$
c		$\begin{bmatrix} \boldsymbol{Z_{old}} & col.\boldsymbol{k}\ of\ Z_{old} \\ row\ \boldsymbol{k}\ of\ Z_{old} & \boldsymbol{Z_{kk} + Z_b} \end{bmatrix}$ Then remove P^{th} row and column applying Kron reduction.
d		$\begin{bmatrix} \boldsymbol{Z_{old}} & col.\boldsymbol{j} - col.\ \boldsymbol{k} \\ row\ \boldsymbol{j} - row\ \boldsymbol{k} & \boldsymbol{Z_{jj} + Z_{kk} - 2Z_{jk} + Z_b} \end{bmatrix}$ Then remove P^{th} row and column applying Kron reduction.

Case Study:

Table 2.1: *Line data for the given network*

Line No.	From Bus	To Bus	R (pu)	X (pu)
1	1	0	0	1.250
2	2	1	0	0.250
3	3	0	0	1.250
4	3	2	0	0.400
5	4	3	0	0.200
6	4	2	0	0.125

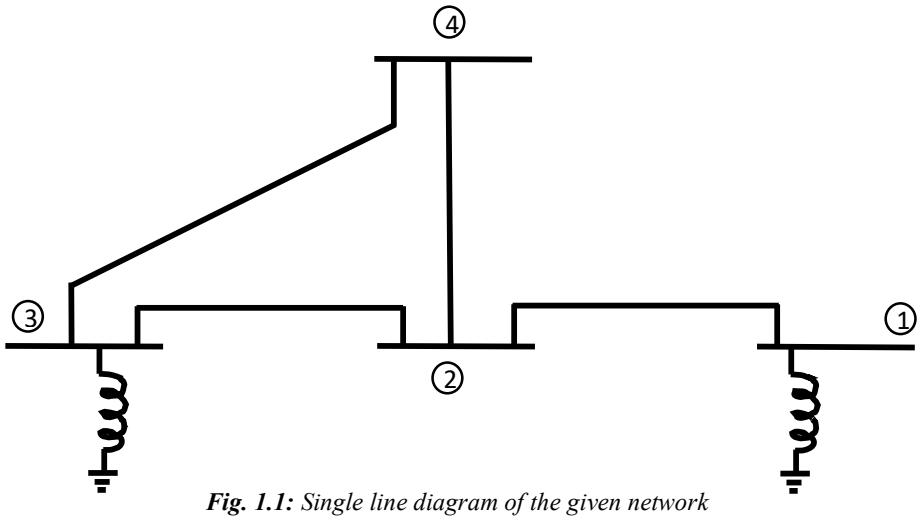


Fig. 1.1: *Single line diagram of the given network*

Algorithm:

1. Load Z_{bus} with a null matrix.
2. Defining a variable to hold the buses already considered, say currentbus.
3. For k^{th} line –
   - Find from bus.
   - Find to bus.
   - Say newbus the max(from bus, to bus).
   - Say refbus the min(from bus, to bus).
   - Say k^{th} branch impedance is Z_b.
     - Check if newbus > currentbus and refbus = 0: i.e., Type 'a' modification.
       - $Z_{bus}^{new} = \begin{bmatrix} Z_{bus}^{old} & col.matrix\ of\ '0'\ of\ order\ Z_{bus}^{old} \\ row\ matrix\ of\ '0'\ of\ order\ Z_{bus}^{old} & Z_b \end{bmatrix}$
     - Otherwise, check if newbus > currentbus and refbus ≠ 0: i.e., Type 'b' modification.
       - $Z_{bus}^{new} = \begin{bmatrix} Z_{bus}^{old} & col.\ \ of\ Z_{bus}^{old}\ at\ refbus\ location \\ row\ \ of\ Z_{bus}^{old}\ at\ refbus\ location & Z_b + Z_{bus}^{old}\ element\ at\ location\ (refbus, refbus) \end{bmatrix}$

- Otherwise, check if newbus <= currentbus and refbus = 0: i.e., Type ‘c’ modification.
  - $Z_{bus}^{new} = Z_{bus}^{old} - (Z_{bus}^{old}(newbus\ col.) * Z_{bus}^{old}(newbus\ row)\ )/(Z_b + Z_{bus}^{old}(newbus, newbus)$
- Otherwise, check if newbus <= currentbus and refbus ≠ 0: i.e., Type ‘d’ modification.
  - $Z_{bus}^{new} = Z_{bus}^{old} - ((Z_{bus}^{old}(from\ bus\ col.) - Z_{bus}^{old}(to\ bus\ col.)) * (Z_{bus}^{old}(from\ bus\ row) - Z_{bus}^{old}(to\ bus\ row))\ )/(Z_b + Z_{bus}^{old}(from\ bus, from\ bus) + Z_{bus}^{old}(to\ bus, to\ bus) - 2 * Z_{bus}^{old}(from\ bus, to\ bus))$
- Update currentbus to newbus.

- Repeat above step for each branch.

```
%------------------------------Zbus formulation--------------------------

%NOTE: Enter the Zdata in a way that first element should be connected to
%reference bus 0 and start with bus no. 1 followed by buses connected to
%bus 1 and so on

%--------------------------------------------------------------------------
%       Element no. |   From bus   |  To bus |     R (pu)   |     X (pu)
%--------------------------------------------------------------------------
Zdata=[     1               1           0           0           1.25    ;
            2               2           1           0           0.25    ;
            3               3           0           0           1.25    ;
            4               3           2           0           0.4     ;
            5               4           3           0           0.2     ;
            6               4           2           0           0.125  ];

Zbus=[];
currentbus=0;               %variable to hold the no. of buses taken already
for i=1:1:size(Zdata,1)
    from_bus=Zdata(i,2);
    to_bus=Zdata(i,3);
    newbus=max(from_bus,to_bus);
    refbus=min(from_bus,to_bus);
    Zb=Zdata(i,4)+1j*Zdata(i,5);
    [rows,cols]=size(Zbus);

    %----------------------Type 1 Modification---------------------------
    if((newbus>currentbus)&&(refbus==0))
        Zbus=[ Zbus              zeros(rows,1) ;
               zeros(1,cols)         Zb       ];
        currentbus=newbus;
        continue;
    end

    %----------------------Type 2 Modification---------------------------
    if((newbus>currentbus)&&(refbus~=0))
        Zbus=[ Zbus             Zbus(:,refbus)          ;
               Zbus(refbus,:)   Zbus(refbus,refbus)+Zb] ;
        currentbus=newbus;
        continue;
    end

    %----------------------Type 3 Modification---------------------------
    if((newbus<=currentbus)&&(refbus==0))
        Zbus=Zbus-(Zbus(:,newbus)*Zbus(newbus,:))./(Zbus(newbus,newbus)+Zb);
        continue;
    end

    %----------------------Type 4 Modification---------------------------
    if((newbus<=currentbus)&&(refbus~=0))
        Zbus=Zbus-((Zbus(:,from_bus)-Zbus(:,to_bus))*(Zbus(from_bus,:)-Zbus↙
```

```
(to_bus,:))./(Zb+Zbus(from_bus,from_bus)+Zbus(to_bus,to_bus)-2*Zbus(from_bus,to_bus));
52          continue;
53      end
54  end
55  %-------------------------------------------------------------------------
56  %----------------------Saving the Zbus in a text file---------------------
57  folder_path='C:\Users\VISHAL MITTAL\Desktop\PS_II Lab\';
58  fid=fopen(strcat(folder_path,'Exp_2.txt'),'w+');
59  fprintf(fid,'%s \n\n','The Zbus solution obtained by Vishal Mittal:');
60
61  Zbus_str=num2str(Zbus,4);
62  fprintf(fid,'%s \n','Network Data:');
63  fprintf(fid,'\n');
64  fprintf(fid,'Element\t From Bus\t  To Bus\t  R (pu)\t     X (pu)');
65  fprintf(fid,'\n');
66  fprintf(fid,'%s↙
\n','-----------------------------------------------------------------------------');
67
68  for k=1:1:size(Zdata,1)
69      for j=1:1:size(Zdata,2)
70          fprintf(fid,'      ');
71          fprintf(fid,'%s\t',num2str(Zdata(k,j)));
72      end
73      fprintf(fid,'\n');
74  end
75  fprintf(fid,'\n\n\n');
76  fprintf(fid,'%s \n\n','Bus Impedance Matrix');
77  for k=1:1:size(Zbus_str,1)
78      fprintf(fid,'%s',Zbus_str(k,:));
79      fprintf(fid,'\n');
80  end
81  fclose(fid);
```

The Zbus solution obtained by Vishal Mittal:

Network Data:

Element	From Bus	To Bus	R (pu)	X (pu)
1	1	0	0	1.25
2	2	1	0	0.25
3	3	0	0	1.25
4	3	2	0	0.4
5	4	3	0	0.2
6	4	2	0	0.125

Bus Impedance Matrix

0+0.7166i	0+0.6099i	0+0.5334i	0+0.5805i
0+0.6099i	0+0.7319i	0+0.6401i	0+0.6966i
0+0.5334i	0+0.6401i	0+0.7166i	0+0.6695i
0+0.5805i	0+0.6966i	0+0.6695i	0+0.7631i

EXPERIMENT 3

Aim: To develop algorithm and to develop the computer program to carry out Gauss-Seidel load flow analysis.

Requirements: Computer, MATLAB

Theory: Power flow studies are important for planning and designing the future expansion of power systems and in determining the best operation of existing systems. The load flow study provides the information regarding the magnitude and phase angle of the voltage at each bus, and the real and reactive power flowing in each line. The power-flow equations are non-linear simultaneous equations, hence iterative solution is feasible. In Gauss-Seidel method, the corrected voltages are found using the equation 1 at each iteration, and done so till amount of correction in each voltage at every bus is less than the predefined tolerance. The method is simple, but convergence is slow.

$$\bar{V}_i^{r+1} = \frac{1}{Y_{ii}}\left[\frac{P_i - jQ_i}{(V_i^r)^*} - \sum_{k=1}^{i-1} Y_{ik}\bar{V}_k^{r+1} - \sum_{j=i+1}^{n} Y_{ij}\bar{V}_j^r\right] \qquad \ldots\ldots\ldots (eq1)$$

Case Study:

Table 3.1: *Line data for the given network*

Line No.	From Bus	To Bus	R (pu)	X (pu)	Mutually coupled to line	Y (pu) mutual	Charging admittance (pu)
1	1	2	0.0108	0.0649	0	0j	0.066j
2	1	4	0.0235	0.0941	0	0j	0.040j
3	2	5	0.0118	0.0471	0	0j	0.070j
4	3	5	0.0147	0.0588	0	0j	0.080j
5	4	5	0.0118	0.0529	0	0j	0.060j
6	2	3	0.0000	0.0400	0	0j	0.000j

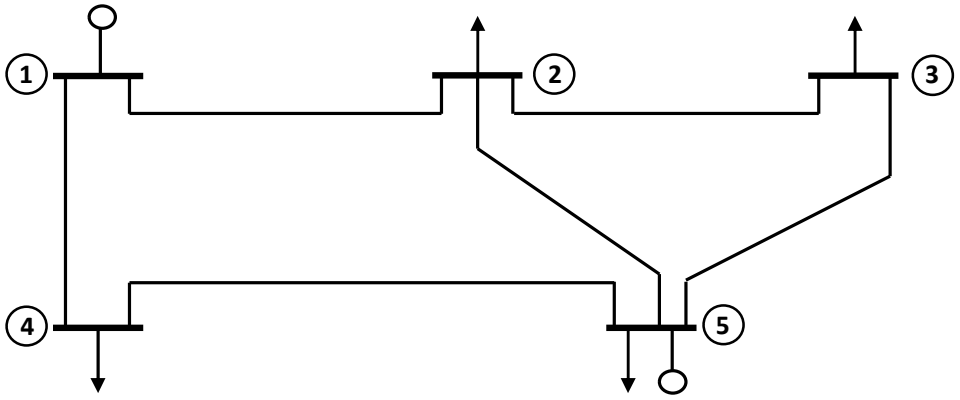


Fig. 3.1: *Single line diagram of the given network*

Table 3.2: *Bus data for the given network*

Bus	P_G (pu)	Q_G (pu)	P_L (pu)	Q_L (pu)	$V\angle\delta$	Q_G Limit (pu)	Remark
1	-	-	0	0	$1.01\angle 0^o$	-	Slack
2	0	0	0.60	0.35	-	-	PQ
3	0	0	0.70	0.42	-	-	PQ
4	0	0	0.80	0.50	-	-	PQ
5	1.9	-	0.65	0.36	1.00	$0 \leq Q_G \leq 1.5$	PV

Algorithm:

1. Obtain the Y_{Bus}, bus-admittance matrix for the given network.
2. Find the number of buses in the network.
3. Assume flat voltages at the PQ and PV buses., defining bus voltage vector V_{bus} containing initial guesses of the bus voltages.
4. Compute the bus real powers P_i and reactive powers Q_i, as

$$P_i = P_G - P_{L,} \qquad Q_i = Q_G - Q_L$$

5. Starting the iterations from r=1:
6. Check whether the iteration number is less than the max. iteration number, if not then stop.
7. If error $|V_{bus}^r - V_{bus}^{r-1}| < 0.00001$, then stop.
8. For bus no. *i*:

   Compute

$$\sum_{k \neq i} V_k Y_{ik} = [V_{bus}]^T [Y_{bus}\ i^{th}\ col] - [V_{bus}]_{i,i}[Y_{bus}]_{i,i}$$

9. Check whether the i^{th} bus is PV bus or not, if yes then follow steps number 10 to 15.
10. Compute $Q_i = -Im\{([{V_{bus}}^r]^T [Y_{bus}\ i^{th}\ col])\bar{V}_i^{r^*}\}$
11. If $Q_i + Q_L\ for\ i^{th}\ bus\ is\ less\ than\ its\ lower\ limit\ of\ Q, then\ set\ Q_i = Q_{Gmin} - Q_L$.
12. Consider the bus as PQ bus by assuming initial voltage guess as 1 pu at this bus.

$$\bar{V}_i^{\,r} = \frac{1}{Y_{ii}}\left[\frac{P_i - jQ_i}{(\overline{V}_i^{r-1})^*} - \sum_{k \neq i} V_k Y_{ik}\right]$$

13. If $Q_i + Q_L\ for\ i^{th}\ bus\ is\ greater\ than\ its\ upper\ limit\ of\ Q, then\ set\ Q_i = Q_{Gmax} - Q_L$.
14. Consider the bus as PQ bus by assuming initial voltage guess as 1 pu at this bus.

$$\bar{V}_i^{\,r} = \frac{1}{Y_{ii}}\left[\frac{P_i - jQ_i}{(\overline{V}_i^{r-1})^*} - \sum_{k \neq i} V_k Y_{ik}\right]$$

15. If $Q_i + Q_L$ *for* i^{th} *bus is within the limits of* Q *specified, then compute angle of* $\bar{V}_i$

$$\angle \bar{V}_i^{\,r} = \angle \frac{1}{Y_{ii}} \left[\frac{P_i - jQ_i}{(\bar{V}_i^{\,r-1})^*} - \sum_{k \neq i} V_k Y_{ik} \right]$$

$$\bar{V}_i^{\,r} = |\bar{V}_i| \left(\cos\left(\angle \bar{V}_i^{\,r} \right) + jsin(\angle \bar{V}_i^{\,r}) \right)$$

16. If the i^{th} bus is PQ bus, then follow step 17.
17. Compute bus voltage as,

$$\bar{V}_i^{\,r} = \frac{1}{Y_{ii}} \left[\frac{P_i - jQ_i}{(\bar{V}_i^{\,r-1})^*} - \sum_{k \neq i} V_k Y_{ik} \right]$$

18. Complete the present iteration for each bus by following the above steps.
19. For the next iteration, reset the PV bus voltage magnitude to the prespecified voltage and angle as the computed during this iteration if the limits were not met.

```
%------------Load Flow Analysis using Gauss-Seidel Method-----------------

%Case Study: Consider a 5-bus system. Bus 1 is slack bus, buses 2,3 and 4
%are PQ buses, and bus 5 is PV bus.


%Line Data:
%-------------------------------------------------------------------------
%                       |     Line      | Mutually|                       
%      Line|From | To  |  R       X    | coupled | y pu |  Charging    |
%      no. |Bus  | Bus |  pu      pu   | to line |mutual| admittance pu|
%-------------------------------------------------------------------------
ydata=[ 1   1       2   0.0108  0.0649      0       0j       0.066j   ;
        2   1       4   0.0235  0.0941      0       0j       0.040j    ;
        3   2       5   0.0118  0.0471      0       0j       0.070j    ;
        4   3       5   0.0147  0.0588      0       0j       0.080j    ;
        5   4       5   0.0118  0.0529      0       0j       0.060j    ;
        6   2       3   0.0000  0.0400      0       0j       0.000j      ];
%-------------------------------------------------------------------------

%Bus Data:
%NOTE: Always make bus 1 as slack bus while entering the bus data
%Bus Type Assignment
%       1 for Slack Bus                 Assuming Flat      Controllable
%       2 for PV Bus                   Voltage Profile    Reactive Power
%       3 for PQ Bus                    for PQ Buses          Source
%-------------------------------------------------------------------------
%        |Bus | Generation|   Load    | Vmag |delta |Bus |Qg_min|Qg_max|
%        |    |P(pu)|Q(pu)|P(pu)|Q(pu)| (pu) |(deg) |Type| (pu) | (pu) |
%-------------------------------------------------------------------------
busdata=[  1    0     0     0     0      1.01   0     1     0      0    ;
           2    0     0     0.6   0.35   1      0     3     0      0    ;
           3    0     0     0.7   0.42   1      0     3     0      0    ;
           4    0     0     0.8   0.5    1      0     3     0      0    ;
           5    1.9   0     0.65  0.36   1      0     2     0      1.5 ];
%-------------------------------------------------------------------------


%------------------------Ybus of the given system-------------------------
[Ybus,A,Yprim]=Y_BUS(ydata);
%-------------------------------------------------------------------------

%---------------------Opening a Text File to store results----------------
folder_path='C:\Users\VISHAL MITTAL\Desktop\PS_II Lab\';
fid=fopen(strcat(folder_path,'Exp_3.txt'),'w+');
fprintf(fid,'%s \n\n','The Load Flow solution using Gauss-Seidel Method obtained by↙
Vishal Mittal:');
Nbus=max(busdata(:,1));
fprintf(fid,'%s \t','Itr no.');

for i=2:1:Nbus
```

```
51      if(busdata(i,8)==2)
52          bus_param=strcat('Qg',num2str(i));
53          fprintf(fid,'%s\t\t',bus_param);
54          bus_param=strcat('V',num2str(i));
55          fprintf(fid,'%s\t\t',bus_param);
56      else
57          bus_param=strcat('V',num2str(i));
58          fprintf(fid,'%s\t\t',bus_param);
59      end
60  end
61  fprintf(fid,'\n');
62  fprintf(fid,'%s \n','-------------------------------------------------');
63  %-------------------------------------------------------------------------
64
65  %------------------------Gauss-Seidel Algorithm--------------------------
66  Nbus=max(busdata(:,1));
67  P_Bus=busdata(:,2)-busdata(:,4);        %Pbus = Pgen - Pload
68  Q_Bus=busdata(:,3)-busdata(:,5);        %Qbus = Qgen - Qload
69
70  V_Bus_r=zeros(Nbus,1);
71  V_Bus_r1=busdata(:,6).*(cosd(busdata(:,7))+1j.*sind(busdata(:,7)));        %↙
converting polar form to rectangular form
72
73  itr=1;
74  while(itr<=30)
75      if(abs(real(V_Bus_r1-V_Bus_r))<=0.00001 & abs(imag(V_Bus_r1-V_Bus_r))<0.00001)
76          break;                                                          %if↙
tolerance level is achieved, stops further iterating
77      end
78      V_Bus_r=V_Bus_r1;                                                   %making↙
computed iteration results as old
79      fprintf(fid,'%d\t',itr);
80      for i=2:1:Nbus                                                      %finding↙
Vbus at each bus other than slack bus (bus 1)
81          sum_Vk_Yik=conj(V_Bus_r1')*Ybus(:,i)-V_Bus_r1(i,1)*Ybus(i,i);
82          if(busdata(i,8)==2)                                             %if the↙
current bus is a PV bus
83              Q_Bus(i,1)=-imag((conj(V_Bus_r1')*Ybus(:,i))*conj(V_Bus_r1(i,1))); %↙
computing Qbus for current PV bus
84              fprintf(fid,'%4.4f\t',Q_Bus(i,1)+busdata(i,5));
85              if((Q_Bus(i,1)+busdata(i,5))<busdata(i,9))                  %if Qgmin↙
limits fail on PV bus
86                  Q_Bus(i,1)=busdata(i,9)-busdata(i,5);                   %take PV↙
bus as PQ bus by setting Qg=Qmin for current itr
87                  V_Bus_r1(i,1)=1;                                        %taking↙
flat voltage profile for this bus
88                  V_Bus_r1(i,1)=((P_Bus(i,1)-1j*Q_Bus(i,1))/conj(V_Bus_r1(i,1))-↙
sum_Vk_Yik)/Ybus(i,i);
89              elseif((Q_Bus(i,1)+busdata(i,5))>busdata(i,10))
90                  Q_Bus(i,1)=busdata(i,10)-busdata(i,5);                  %take PV↙
bus as PQ bus by setting Qg=Qmax for current itr
```

```
 91                 V_Bus_r1(i,1)=1;                                                %taking↙
flat voltage profile for this bus
 92                 V_Bus_r1(i,1)=((P_Bus(i,1)-1j*Q_Bus(i,1))/conj(V_Bus_r1(i,1))-↙
sum_Vk_Yik)/Ybus(i,i);
 93             else                                                                %if all↙
limits meet, compute angle of bus voltage
 94                 Vi_angle=angle(((P_Bus(i,1)-1j*Q_Bus(i,1))/conj(V_Bus_r1(i,1))-↙
sum_Vk_Yik)/Ybus(i,i));
 95                 V_Bus_r1(i,1)=abs(V_Bus_r1(i,1))*(cos(Vi_angle)+1j*sin(Vi_angle)); %↙
PV bus voltage=given Vmag(cos(new_delta)+jsin(new_delta))
 96             end
 97             fprintf(fid,'%s\t',num2str(V_Bus_r1(i,1)));
 98         else                                                                    %if the↙
current bus is PQ bus
 99             V_Bus_r1(i,1)=((P_Bus(i,1)-1j*Q_Bus(i,1))/conj(V_Bus_r1(i,1))-sum_Vk_Yik)↙
/Ybus(i,i);
100             fprintf(fid,'%s\t',num2str(V_Bus_r1(i,1)));
101         end
102     end
103     %Restting the Voltage magnitudes of PV buses for next iteration to as
104     %given in the busdata those considered as PQ in present iteration
105     for i=2:1:Nbus
106         if(busdata(i,8)==3)                                                     %continue↙
if PQ bus
107             continue;
108         end
109         if(((Q_Bus(i,1)+busdata(i,5))==busdata(i,9))||((Q_Bus(i,1)+busdata(i,5))↙
==busdata(i,10))) %if Qi=Qgmin or Qgmax i.e. assumed PQ bus
110             V_Bus_r1(i,1)=busdata(i,6)*(cos(angle(V_Bus_r1(i,1)))+1j*sin(angle↙
(V_Bus_r1(i,1))));  %making magnitude equal to given in busdata
111↙
%and angle as per of computed voltage value
112         end
113     end
114
115     disp(itr);
116     disp([abs(V_Bus_r1) angle(V_Bus_r1).*(180/pi)]);
117     disp(Q_Bus);
118     itr=itr+1;
119     fprintf(fid,'\n\n');
120
121 end
122 %-------------------------------------------------------------------------
123
124 %----------Finding Pgen and Qgen at Slack bus, and Qgen at PV bus----------
125 %-------------------------and updating in busdata--------------------------
126 for i=1:1:Nbus
127     if(busdata(i,8)==1)                                                         %if↙
current bus is slack bus
128         busdata(i,2)=busdata(i,4)+real((conj(V_Bus_r1')*Ybus(:,i))*conj(V_Bus_r1(i,↙
1)));  %Pgen=Pload+Pbus
```

```
129         busdata(i,3)=busdata(i,5)-imag((conj(V_Bus_r1')*Ybus(:,i))*conj(V_Bus_r1(i,↙
1)));  %Qgen=Qload+Qbus
130     elseif(busdata(i,8)==2)                                                 %if↙
current bus is PV bus
131         busdata(i,3)=busdata(i,5)-imag((conj(V_Bus_r1')*Ybus(:,i))*conj(V_Bus_r1(i,↙
1)));  %Qgen=Qload+Qbus
132     else
133         continue;                                                           %continue↙
if current bus is PQ bus
134     end
135 end
136 %-------------------------------------------------------------------------
137 %-----------Updating Vmag and Vangle at the buses with V_Bus_r1------------
138 busdata(:,6)=abs(V_Bus_r1);
139 busdata(:,7)=angle(V_Bus_r1).*(180/pi);
140 %-------------------------------------------------------------------------
141 %---------------------Printing Results in text file-----------------------
142 fprintf(fid,'\n\n');
143 fprintf(fid,'%s\n','---------------------------------------BUS↙
INFORMATION-----------------------------------');
144 fprintf(fid,'%s\t','Bus no.');
145 fprintf(fid,'%s\t\t','  Volts');
146 fprintf(fid,'%s\t\t',' angle');
147 fprintf(fid,'%s\t\t','  Generation');
148 fprintf(fid,'%s\t\t','      Load');
149 fprintf(fid,'%s\n','Bus');
150 fprintf(fid,'\t');
151 fprintf(fid,'%s\t\t','  (pu)');
152 fprintf(fid,'%s\t\t',' (deg.)');
153 fprintf(fid,'%s','  P (pu)');
154 fprintf(fid,'%s\t\t',' Q (pu)');
155 fprintf(fid,'%s',' P (pu)');
156 fprintf(fid,'%s\t\t','  Q (pu)');
157 fprintf(fid,'%s\n','Type');
158 fprintf(fid,'%↙
s\n','-------------------------------------------------------------------------------↙
---------');
159
160 for i=1:1:Nbus
161     fprintf(fid,'%d \t',busdata(i,1));
162     fprintf(fid,'% 4.4f\t\t',busdata(i,6));
163     fprintf(fid,'% 4.4f\t\t',busdata(i,7));
164     fprintf(fid,'% 4.4f\t',busdata(i,2));
165     fprintf(fid,'% 4.4f\t\t',busdata(i,3));
166     fprintf(fid,'% 4.4f\t',busdata(i,4));
167     fprintf(fid,'% 4.4f\t\t',busdata(i,5));
168     if(busdata(i,8)==1)
169         fprintf(fid,'%s\n\n','Slack');
170     elseif(busdata(i,8)==2)
171         fprintf(fid,'%s\n\n','PV');
172     else
```

```
173         fprintf(fid,'%s\n\n','PQ');
174     end
175 end
176
177 %-------------------------Finding Line flows-----------------------------
178 Nline=size(ydata,1);
179 lineflow=[];
180 for L=1:1:Nline                                                        %↙
S_ij=Vi*conj((Vi-Vj)/Z_ij)-Vimag^2*Yi (charging VAR at ith bus)
181     S_ij=V_Bus_r1(ydata(L,2))*conj((V_Bus_r1(ydata(L,2),1)-V_Bus_r1(ydata(L,3),1))/↙
(ydata(L,4)+1j*ydata(L,5)))-(abs(V_Bus_r1(ydata(L,2)))^2)*ydata(L,8)/2;
182     lineflow=[lineflow; [ydata(L,2) ydata(L,3) real(S_ij) imag(S_ij)]];
183     S_ji=V_Bus_r1(ydata(L,3))*conj((V_Bus_r1(ydata(L,3),1)-V_Bus_r1(ydata(L,2),1))/↙
(ydata(L,4)+1j*ydata(L,5)))-(abs(V_Bus_r1(ydata(L,3)))^2)*ydata(L,8)/2;
184     lineflow=[lineflow; [ydata(L,3) ydata(L,2) real(S_ji) imag(S_ji)]];
185 end
186 %-------------------------------------------------------------------------
187 %-------------------------------Losses------------------------------------
188 Ploss=sum(busdata(:,2))-sum(busdata(:,4));                             %↙
Ploss=Pgen-Pload
189 Qloss=sum(busdata(:,3))-sum(busdata(:,5));                             %↙
Qloss=Qgen-Qload
190 %-------------------------------------------------------------------------
191
192 %-------------------------Printing Lineflows------------------------------
193 fprintf(fid,'\n\n');
194 fprintf(fid,'%s\n','----------------Line Flows---------------');
195 fprintf(fid,'%s\t','From');
196 fprintf(fid,'%s\t','To');
197 fprintf(fid,'%s\t\t',' P (pu)');
198 fprintf(fid,'%s\n',' Q (pu)');
199 fprintf(fid,'%s\t','Bus');
200 fprintf(fid,'%s\n','Bus');
201 fprintf(fid,'%s\n','---------------------------------------');
202 for L=1:1:size(lineflow,1)
203     fprintf(fid,'%d\t',lineflow(L,1));
204     fprintf(fid,'%d\t',lineflow(L,2));
205     fprintf(fid,'% 4.4f\t\t',lineflow(L,3));
206     fprintf(fid,'% 4.4f\n\n',lineflow(L,4));
207 end
208 fprintf(fid,'%s\n\n\n','-------------------------------------');
209 fprintf(fid,'%s\t','Total Real Power Generation (pu) :');
210 fprintf(fid,'%4.4f\n\n',sum(busdata(:,2)));
211 fprintf(fid,'%s\t','Total Reactive Power Generation (pu) :');
212 fprintf(fid,'%4.4f\n\n',sum(busdata(:,3)));
213 fprintf(fid,'%s\t\t','Total Real Power Demand (pu) :');
214 fprintf(fid,'%4.4f\n\n',sum(busdata(:,4)));
215 fprintf(fid,'%s\t','Total Reactive Power Demand (pu) :');
216 fprintf(fid,'%4.4f\n\n',sum(busdata(:,5)));
217 fprintf(fid,'%s\t\t','Total Real Power Loss (pu) :');
218 fprintf(fid,'% 4.4f\n\n',Ploss);
```

```
219 fprintf(fid,'%s\t','Total Reactive Power Loss (pu) :');
220 fprintf(fid,'% 4.4f\n',Qloss);
221
222 fclose('all');
```

The Load Flow solution using Gauss-Seidel Method obtained by Vishal Mittal:

Itr no.	V2	V3	V4	Qg5	V5
1	0.99665-0.0091798i	0.98659-0.021379i	0.982-0.023525i	0.6348	1+0.0027691i
2	0.98981-0.016302i	0.98137-0.024133i	0.98104-0.021589i	0.8011	1-0.00074747i
3	0.98756-0.018366i	0.97998-0.026689i	0.98104-0.02392i	0.8793	0.99999-0.0034405i
4	0.98693-0.020262i	0.97959-0.028855i	0.98097-0.025591i	0.9058	0.99998-0.005501i
5	0.98672-0.021821i	0.97944-0.03058i	0.98091-0.026875i	0.9172	0.99997-0.0071016i
6	0.98661-0.023056i	0.97937-0.031934i	0.98086-0.027872i	0.9239	0.99997-0.0083486i
7	0.98655-0.024023i	0.97932-0.032993i	0.98082-0.028649i	0.9286	0.99996-0.0093211i
8	0.9865-0.024778i	0.97929-0.03382i	0.98079-0.029255i	0.9322	0.99995-0.01008i
9	0.98647-0.025368i	0.97926-0.034465i	0.98077-0.029727i	0.9350	0.99994-0.010672i
10	0.98644-0.025828i	0.97924-0.034968i	0.98075-0.030096i	0.9371	0.99994-0.011134i
11	0.98642-0.026187i	0.97922-0.035361i	0.98074-0.030384i	0.9388	0.99993-0.011495i
12	0.9864-0.026468i	0.97921-0.035668i	0.98072-0.030609i	0.9401	0.99993-0.011776i
13	0.98639-0.026686i	0.97919-0.035907i	0.98071-0.030784i	0.9411	0.99993-0.011996i
14	0.98638-0.026857i	0.97919-0.036094i	0.98071-0.030921i	0.9420	0.99993-0.012167i
15	0.98637-0.02699i	0.97918-0.03624i	0.9807-0.031028i	0.9426	0.99992-0.012301i
16	0.98636-0.027094i	0.97917-0.036353i	0.9807-0.031111i	0.9431	0.99992-0.012405i
17	0.98636-0.027175i	0.97917-0.036442i	0.98069-0.031176i	0.9435	0.99992-0.012487i
18	0.98635-0.027239i	0.97917-0.036511i	0.98069-0.031227i	0.9438	0.99992-0.01255i
19	0.98635-0.027288i	0.97916-0.036565i	0.98069-0.031267i	0.9440	0.99992-0.0126i
20	0.98635-0.027327i	0.97916-0.036608i	0.98069-0.031297i	0.9442	0.99992-0.012639i
21	0.98635-0.027357i	0.97916-0.036641i	0.98069-0.031322i	0.9443	0.99992-0.012669i
22	0.98634-0.02738i	0.97916-0.036666i	0.98069-0.03134i	0.9444	0.99992-0.012693i
23	0.98634-0.027399i	0.97916-0.036686i	0.98068-0.031355i	0.9445	0.99992-0.012711i
24	0.98634-0.027413i	0.97916-0.036702i	0.98068-0.031367i	0.9446	0.99992-0.012725i
25	0.98634-0.027424i	0.97916-0.036714i	0.98068-0.031376i	0.9446	0.99992-0.012737i
26	0.98634-0.027433i	0.97916-0.036724i	0.98068-0.031383i	0.9447	0.99992-0.012745i

----------BUS INFORMATION----------

Bus no.	Volts (pu)	angle (deg.)	Generation P (pu)	Generation Q (pu)	Load P (pu)	Load Q (pu)	Bus Type
1	1.0100	0.0000	0.8661	0.4521	0.0000	0.0000	Slack
2	0.9867	-1.5931	0.0000	0.0000	0.6000	0.3500	PQ
3	0.9798	-2.1479	0.0000	0.0000	0.7000	0.4200	PQ
4	0.9812	-1.8329	0.0000	0.0000	0.8000	0.5000	PQ
5	1.0000	-0.7303	1.9000	0.9448	0.6500	0.3600	PV

----------Line Flows----------

From Bus	To Bus	P (pu)	Q (pu)
1	2	0.4750	0.2555
2	1	-0.4718	-0.3016
1	4	0.3910	0.1966
4	1	-0.3864	-0.2178
2	5	-0.3619	-0.2192
5	2	0.3639	0.1581
3	5	-0.4658	-0.2527
5	3	0.4699	0.1904
4	5	-0.4134	-0.2822
5	4	0.4163	0.2362
2	3	0.2340	0.1708
3	2	-0.2340	-0.1673

Total Real Power Generation (pu) : 2.7661

Total Reactive Power Generation (pu) : 1.3969

Total Real Power Demand (pu) : 2.7500

Total Reactive Power Demand (pu) : 1.6300

Total Real Power Loss (pu) : 0.0161

Total Reactive Power Loss (pu) : -0.2331

EXPERIMENT 4

Aim: To develop a computer program to carry out load flow analysis using Newton-Raphson's Method.

Requirements: Computer, MATLAB

Theory: Power flow studies are important for planning and designing the future expansion of power systems and in determining the best operation of existing systems. The load flow study provides the information regarding the magnitude and phase angle of the voltage at each bus, and the real and reactive power flowing in each line. The power-flow equations are non-linear simultaneous equations, hence iterative solution is feasible. Newton-Raphson's load flow method is fast but requires more memory space because it involves finding of inverse of Jacobian matrix. The new updated values of independent variables is obtained by the equation given below.

$$[X]^{r+1} = [X]^r - [J^{-1}\Delta F]_{at\ X^r}$$

Case Study:

Table 4.1: *Line data for the given network*

Line No.	From Bus	To Bus	R (pu)	X (pu)	Mutually coupled to line	Y (pu) mutual	Charging admittance (pu)
1	1	2	0.0108	0.0649	0	0j	0.066j
2	1	4	0.0235	0.0941	0	0j	0.040j
3	2	5	0.0118	0.0471	0	0j	0.070j
4	3	5	0.0147	0.0588	0	0j	0.080j
5	4	5	0.0118	0.0529	0	0j	0.060j
6	2	3	0.0000	0.0400	0	0j	0.000j

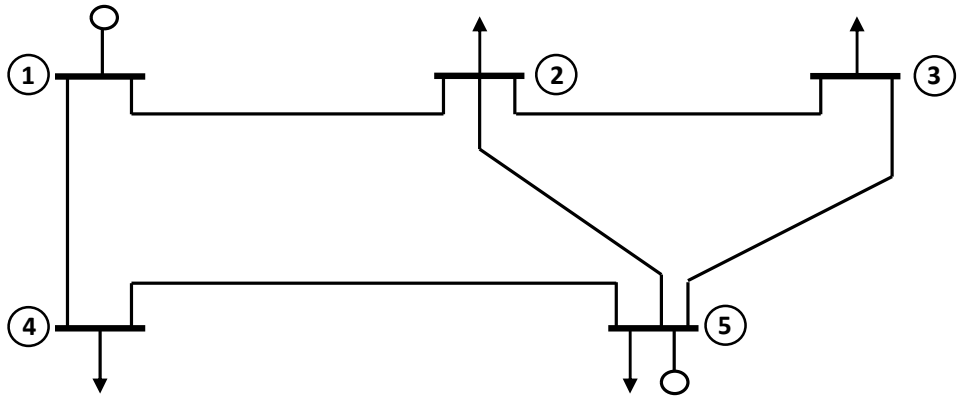


Fig. 4.1: *Single line diagram of the given network*

Table 4.2: *Bus data for the given network*

Bus	**P_G (pu)**	**Q_G (pu)**	**P_L (pu)**	**Q_L (pu)**	$V\angle\delta$	**Q_G Limit (pu)**	**Remark**
1	-	-	0	0	$1.01\angle 0^o$	-	Slack
2	0	0	0.60	0.35	-	-	PQ
3	0	0	0.70	0.42	-	-	PQ
4	0	0	0.80	0.50	-	-	PQ
5	1.9	-	0.65	0.36	1.00	$0 \leq Q_G \leq 1.5$	PV

```
%------------Load Flow Analysis using Newton-Raphson Method-----------------

%Case Study: Consider a 5-bus system. Bus 1 is slack bus, buses 2,3 and 4
%are PQ buses, and bus 5 is PV bus.

%Line Data:
%------------------------------------------------------------------------
%                        |     Line      | Mutually|                      
%      Line|From | To  |  R       X     | coupled | y pu |  Charging    |
%      no. |Bus  | Bus |  pu      pu    | to line |mutual| admittance pu|
%------------------------------------------------------------------------
ydata=[ 1   1       2   0.0108  0.0649      0       0j       0.066j   ;
        2   1       4   0.0235  0.0941      0       0j       0.040j    ;
        3   2       5   0.0118  0.0471      0       0j       0.070j    ;
        4   3       5   0.0147  0.0588      0       0j       0.080j    ;
        5   4       5   0.0118  0.0529      0       0j       0.060j    ;
        6   2       3   0.0000  0.0400      0       0j       0.000j      ];
%-------------------------------------------------------------------------

%Bus Data:
%NOTE: Always make bus 1 as slack bus while entering the bus data
%Bus Type Assignment
%       1 for Slack Bus                Assuming Flat      Controllable
%       2 for PV Bus                  Voltage Profile    Reactive Power
%       3 for PQ Bus                   for PQ Buses          Source
%-------------------------------------------------------------------------
%        |Bus | Generation|   Load    | Vmag |delta |Bus |Qg_min|Qg_max|
%        |    |P(pu)|Q(pu)|P(pu)|Q(pu)| (pu) |(deg) |Type| (pu) | (pu) |
%-------------------------------------------------------------------------
busdata=[  1    0     0     0     0      1.01   0     1     0      0    ;
           2    0     0     0.6   0.35   1      0     3     0      0    ;
           3    0     0     0.7   0.42   1      0     3     0      0    ;
           4    0     0     0.8   0.5    1      0     3     0      0    ;
           5    1.9   0     0.65  0.36   1      0     2     0      1.5 ];
%-------------------------------------------------------------------------
%------------------------Ybus of the given system-------------------------
[Ybus,A,Yprim]=Y_BUS(ydata);
theta_ik=angle(Ybus).*(180/pi);
Yik=abs(Ybus);
%-------------------------------------------------------------------------
%---------------------Opening a Text File to store results----------------
folder_path='C:\Users\VISHAL MITTAL\Desktop\PS_II Lab\';
fid=fopen(strcat(folder_path,'Exp_4.txt'),'w+');
fprintf(fid,'%s \n\n','The Load Flow solution using Newton-Raphson Method obtained by↙
Vishal Mittal:');
Nbus=max(busdata(:,1));
fprintf(fid,'%s \t','Itr no.');

for i=2:1:Nbus
    bus_param=strcat('d',num2str(i));
    fprintf(fid,'%4s\t\t',bus_param);
```

```
 51     bus_param=strcat('V',num2str(i));
 52     fprintf(fid,'%4s\t\t',bus_param);
 53 end
 54 fprintf(fid,'\n');
 55 fprintf(fid,'%s \n','-------------------------------------------------');
 56 %-------------------------------------------------------------------------
 57
 58 %-------------------------Newton-Raphson Algorithm------------------------
 59 V0=zeros(length(busdata(:,6)),1);
 60 delta0=zeros(length(V0),1);
 61
 62 V1=busdata(:,6);              %initial guess of bus voltage magnitudes
 63 delta1=busdata(:,7);          %initial guess of bus voltage angles in deg.
 64
 65 itr=0;
 66 while(itr<10)
 67     if((abs(V1-V0)<0.00001) & (abs(delta1-delta0)<0.00001))
 68         break;
 69     end
 70     itr=itr+1;
 71     %making matrix F_aprox = [P2 ; Q2 ; P3 ; Q3 ; ....]
 72     %and F_actul = [P2 ; Q2 ; P3 ; Q3 ; ....]
 73     %and a matrix Bus_type to hold bus type info 2: PV and 3: PQ
 74     F_aprox=[];
 75     F_actul=[];
 76     Bus_type=[1];
 77
 78     for i=2:1:length(busdata(:,1))
 79         if(busdata(i,8)==3)             %if PQ bus
 80             Pi=0;
 81             Qi=0;
 82             for k=1:1:length(busdata(:,1))
 83                 Pi=Pi+V1(k)*V1(i)*Yik(i,k)*cosd(theta_ik(i,k)+delta1(k)-delta1(i));
 84                 Qi=Qi-V1(k)*V1(i)*Yik(i,k)*sind(theta_ik(i,k)+delta1(k)-delta1(i));
 85             end
 86                 F_aprox=[F_aprox ; Pi ; Qi];
 87                 F_actul=[F_actul ; busdata(i,2)-busdata(i,4) ; busdata(i,3)-busdata↙
(i,5)];
 88                 Bus_type=[Bus_type ; 3];
 89         else                             %if PV bus
 90             Pi=0;
 91             Qi=0;
 92             for k=1:1:length(busdata(:,1))
 93                 Pi=Pi+V1(k)*V1(i)*Yik(i,k)*cosd(theta_ik(i,k)+delta1(k)-delta1(i));
 94                 Qi=Qi-V1(k)*V1(i)*Yik(i,k)*sind(theta_ik(i,k)+delta1(k)-delta1(i));
 95             end
 96             %if Qgi is out of bounds the min max limits then consider
 97             %it as PQ bus otherwise PV bus
 98             if((Qi+busdata(i,5))<busdata(i,9))
 99                 F_aprox=[F_aprox ; Pi ; Qi];
100                 F_actul=[F_actul ; busdata(i,2)-busdata(i,4) ; busdata(i,9)-busdata↙
```

```
(i,5)];
                Bus_type=[Bus_type ; 3];
            elseif((Qi+busdata(i,5))>busdata(i,10))
                F_aprox=[F_aprox ; Pi ; Qi];
                F_actul=[F_actul ; busdata(i,2)-busdata(i,4) ; busdata(i,10)-busdata
(i,5)];
                Bus_type=[Bus_type ; 3];
            else
                F_aprox=[F_aprox ; Pi];
                F_actul=[F_actul ; busdata(i,2)-busdata(i,4)];
                Bus_type=[Bus_type ; 2];
            end
        end
    end
    %-------------------Storing the iteration results----------------------
    fprintf(fid,'%4d \t\t',itr);
    for i=2:1:Nbus
        fprintf(fid,'% 5.4f \t',delta1(i));
        fprintf(fid,'%5.4f \t',V1(i));
    end
    fprintf(fid,'\n');
   %-----------------------making Jacobian matrix--------------------------
   J=zeros(length(F_aprox),length(F_aprox));
   row_indx=1;
    for i=2:1:length(busdata(:,1))        %for P2, Q2, P3, Q3 ,... i.e. F
        col_indx=1;
        for k=2:1:length(busdata(:,1))    %for x to compute dP_i/dx
            if(k~=i)
                %dP_i/d_delta_k
                J(row_indx,col_indx)=-V1(k)*V1(i)*Yik(i,k)*sind(theta_ik(i,k)+delta1
(k)-delta1(i));
            else
                %dP_i/d_delta_i
                for j=1:1:length(busdata(:,1))
                    J(row_indx,col_indx)=J(row_indx,col_indx)+V1(j)*V1(i)*Yik(i,j)
*sind(theta_ik(i,j)+delta1(j)-delta1(i));
                end
                J(row_indx,col_indx)=J(row_indx,col_indx)-V1(i)*V1(i)*Yik(i,i)*sind
(theta_ik(i,i));
            end
            col_indx=col_indx+1;
            if(Bus_type(k)==3)        %if PQ bus then x: delta and Vmag
                if(k~=i)
                    %dP_i/dV_k
                    J(row_indx,col_indx)=V1(i)*Yik(i,k)*cosd(theta_ik(i,k)+delta1(k)-
delta1(i));
                else
                    %dP_i/dV_i
                    for j=1:1:length(busdata(:,1))
                        J(row_indx,col_indx)=J(row_indx,col_indx)+V1(j)*Yik(i,j)*cosd
(theta_ik(i,j)+delta1(j)-delta1(i));
```

```
145                     end
146                     J(row_indx,col_indx)=J(row_indx,col_indx)+V1(i)*Yik(i,i)*cosd↙
(theta_ik(i,i));
147                 end
148                 col_indx=col_indx+1;
149             end
150         end
151         if(Bus_type(i)==3)              %if ith bus is PQ, find dQ_i/dx also
152             row_indx=row_indx+1;
153             col_indx=1;
154             for k=2:1:length(busdata(:,1))    %for x to compute dQ_i/dx
155                 if(k~=i)
156                     %dQ_i/d_delta_k
157                     J(row_indx,col_indx)=-V1(k)*V1(i)*Yik(i,k)*cosd(theta_ik(i,k)↙
+delta1(k)-delta1(i));
158                 else
159                     %dQ_i/d_delta_i
160                     for j=1:1:length(busdata(:,1))
161                         J(row_indx,col_indx)=J(row_indx,col_indx)+V1(j)*V1(i)*Yik(i,↙
j)*cosd(theta_ik(i,j)+delta1(j)-delta1(i));
162                     end
163                     J(row_indx,col_indx)=J(row_indx,col_indx)-V1(i)*V1(i)*Yik(i,i)↙
*cosd(theta_ik(i,i));
164                 end
165                 col_indx=col_indx+1;
166                 if(Bus_type(k)==3)            %if PQ bus then x: delta and Vmag
167                     if(k~=i)
168                         %dQ_i/dV_k
169                         J(row_indx,col_indx)=-V1(i)*Yik(i,k)*sind(theta_ik(i,k)↙
+delta1(k)-delta1(i));
170                     else
171                         %dQ_i/dV_i
172                         for j=1:1:length(busdata(:,1))
173                             J(row_indx,col_indx)=J(row_indx,col_indx)-V1(j)*Yik(i,j)↙
*sind(theta_ik(i,j)+delta1(j)-delta1(i));
174                         end
175                         J(row_indx,col_indx)=J(row_indx,col_indx)-V1(i)*Yik(i,i)*sind↙
(theta_ik(i,i));
176                     end
177                     col_indx=col_indx+1;
178                 end
179             end
180         end
181         row_indx=row_indx+1;
182     end
183     %-------------------------------------------------------------------------
184     dF=F_actul-F_aprox;
185     dX=inv(J)*dF;
186     V0=V1;
187     delta0=delta1;
188     %X=Xold+dX
```

```
    %note: dX is of the form: [d_delta2; dV2; d_delta3; dV3; ....]
    for i=2:1:length(Bus_type)
        if(Bus_type(i)==3)                              %if PQ bus
            delta1(i)=delta1(i)+dX(2*(i-1)-1)*180/pi; %separating d_delta from dX
            V1(i)=V1(i)+dX(2*(i-1));                    %separating dV from dX
        else                                            %if PV bus
            delta1(i)=delta1(i)+dX(2*(i-1)-1)*180/pi;
        end
    end
end
%----storing the last iteration result in text file and busdata matrix-----
fprintf(fid,'%4d \t\t',itr+1);
for i=2:1:Nbus
    fprintf(fid,'% 5.4f \t',delta1(i));
    fprintf(fid,'%5.4f \t',V1(i));
end
fprintf(fid,'\n');
busdata(:,6)=V1;
busdata(:,7)=delta1;
%-------------------------------------------------------------------------
V_Bus=V1.*(cosd(delta1)+1j.*sind(delta1));   %finding V_Bus in complex form

%----------Finding Pgen and Qgen at Slack bus, and Qgen at PV bus----------
%-------------------------and updating in busdata-------------------------
for i=1:1:Nbus
    if(busdata(i,8)==1)                                                    %if↙
current bus is slack bus
        busdata(i,2)=busdata(i,4)+real((conj(V_Bus')*Ybus(:,i))*conj(V_Bus(i,1)));  %↙
Pgen=Pload+Pbus
        busdata(i,3)=busdata(i,5)-imag((conj(V_Bus')*Ybus(:,i))*conj(V_Bus(i,1)));  %↙
Qgen=Qload+Qbus
    elseif(busdata(i,8)==2)                                                %if↙
current bus is PV bus
        busdata(i,3)=busdata(i,5)-imag((conj(V_Bus')*Ybus(:,i))*conj(V_Bus(i,1)));  %↙
Qgen=Qload+Qbus
    else
        continue;                                                          %continue↙
if current bus is PQ bus
    end
end
%------------------Printing the Bus information results--------------------
fprintf(fid,'\n\n');
fprintf(fid,'%s\n','---------------------------------------BUS↙
INFORMATION-----------------------------------');
fprintf(fid,'%s\t','Bus no.');
fprintf(fid,'%s\t\t','  Volts');
fprintf(fid,'%s\t\t',' angle');
fprintf(fid,'%s\t\t','  Generation');
fprintf(fid,'%s\t\t','      Load');
fprintf(fid,'%s\n','Bus');
fprintf(fid,'\t');
```

```
233 fprintf(fid,'%s\t\t','  (pu)');
234 fprintf(fid,'%s\t\t',' (deg.)');
235 fprintf(fid,'%s','  P (pu)');
236 fprintf(fid,'%s\t\t',' Q (pu)');
237 fprintf(fid,'%s',' P (pu)');
238 fprintf(fid,'%s\t\t','  Q (pu)');
239 fprintf(fid,'%s\n','Type');
240 fprintf(fid,'%↙
s\n','-------------------------------------------------------------------------------↙
---------');
241
242 for i=1:1:Nbus
243     fprintf(fid,'%d \t',busdata(i,1));
244     fprintf(fid,'% 4.4f\t\t',busdata(i,6));
245     fprintf(fid,'% 4.4f\t\t',busdata(i,7));
246     fprintf(fid,'% 4.4f\t',busdata(i,2));
247     fprintf(fid,'% 4.4f\t\t',busdata(i,3));
248     fprintf(fid,'% 4.4f\t',busdata(i,4));
249     fprintf(fid,'% 4.4f\t\t',busdata(i,5));
250     if(busdata(i,8)==1)
251         fprintf(fid,'%s\n','Slack');
252     elseif(busdata(i,8)==2)
253         fprintf(fid,'%s\n','PV');
254     else
255         fprintf(fid,'%s\n','PQ');
256     end
257 end
258 %--------------------------Finding Line flows-----------------------------
259 Nline=size(ydata,1);
260 lineflow=[];
261 for L=1:1:Nline                                                    %↙
S_ij=Vi*conj((Vi-Vj)/Z_ij)-Vimag^2*Yi (charging VAR at ith bus)
262     S_ij=V_Bus(ydata(L,2))*conj((V_Bus(ydata(L,2),1)-V_Bus(ydata(L,3),1))/(ydata(L,4)↙
+1j*ydata(L,5)))-(abs(V_Bus(ydata(L,2)))^2)*ydata(L,8)/2;
263     lineflow=[lineflow; [ydata(L,2) ydata(L,3) real(S_ij) imag(S_ij)]];
264     S_ji=V_Bus(ydata(L,3))*conj((V_Bus(ydata(L,3),1)-V_Bus(ydata(L,2),1))/(ydata(L,4)↙
+1j*ydata(L,5)))-(abs(V_Bus(ydata(L,3)))^2)*ydata(L,8)/2;
265     lineflow=[lineflow; [ydata(L,3) ydata(L,2) real(S_ji) imag(S_ji)]];
266 end
267 %-------------------------------------------------------------------------
268 %---------------------------------Losses----------------------------------
269 Ploss=sum(busdata(:,2))-sum(busdata(:,4));                         %↙
Ploss=Pgen-Pload
270 Qloss=sum(busdata(:,3))-sum(busdata(:,5));                         %↙
Qloss=Qgen-Qload
271 %-------------------------------------------------------------------------
272 %-------------------------Printing Lineflows------------------------------
273 fprintf(fid,'\n\n');
274 fprintf(fid,'%s\n','----------------Line Flows---------------');
275 fprintf(fid,'%s\t','From');
276 fprintf(fid,'%s\t','To');
```

```
277 fprintf(fid,'%s\t\t',' P (pu)');
278 fprintf(fid,'%s\n',' Q (pu)');
279 fprintf(fid,'%s\t','Bus');
280 fprintf(fid,'%s\n','Bus');
281 fprintf(fid,'%s\n','----------------------------------------');
282 for L=1:1:size(lineflow,1)
283     fprintf(fid,'%d\t',lineflow(L,1));
284     fprintf(fid,'%d\t',lineflow(L,2));
285     fprintf(fid,'% 4.4f\t\t',lineflow(L,3));
286     fprintf(fid,'% 4.4f\n',lineflow(L,4));
287 end
288 fprintf(fid,'%s\n\n\n','----------------------------------------');
289 fprintf(fid,'%s\t','Total Real Power Generation (pu) :');
290 fprintf(fid,'%4.4f\n',sum(busdata(:,2)));
291 fprintf(fid,'%s\t','Total Reactive Power Generation (pu) :');
292 fprintf(fid,'%4.4f\n',sum(busdata(:,3)));
293 fprintf(fid,'%s\t\t','Total Real Power Demand (pu) :');
294 fprintf(fid,'%4.4f\n',sum(busdata(:,4)));
295 fprintf(fid,'%s\t','Total Reactive Power Demand (pu) :');
296 fprintf(fid,'%4.4f\n',sum(busdata(:,5)));
297 fprintf(fid,'%s\t\t','Total Real Power Loss (pu) :');
298 fprintf(fid,'% 4.4f\n',Ploss);
299 fprintf(fid,'%s\t','Total Reactive Power Loss (pu) :');
300 fprintf(fid,'% 4.4f\n',Qloss);
301 fclose('all');
```

The Load Flow solution using Newton-Raphson Method obtained by Vishal Mittal:

Itr no.	d2	V2	d3	V3	d4	V4	d5	V5
1	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000
2	-1.5510	0.9872	-2.0825	0.9805	-1.7790	0.9818	-0.6861	1.0000
3	-1.5949	0.9867	-2.1498	0.9798	-1.8343	0.9812	-0.7320	1.0000
4	-1.5949	0.9867	-2.1499	0.9798	-1.8343	0.9812	-0.7321	1.0000
5	-1.5949	0.9867	-2.1499	0.9798	-1.8343	0.9812	-0.7321	1.0000

BUS INFORMATION

Bus no.	Volts (pu)	angle (deg.)	Generation P (pu)	Generation Q (pu)	Load P (pu)	Load Q (pu)	Bus Type
1	1.0100	0.0000	0.8668	0.4520	0.0000	0.0000	Slack
2	0.9867	-1.5949	0.0000	0.0000	0.6000	0.3500	PQ
3	0.9798	-2.1499	0.0000	0.0000	0.7000	0.4200	PQ
4	0.9812	-1.8343	0.0000	0.0000	0.8000	0.5000	PQ
5	1.0000	-0.7321	1.9000	0.9448	0.6500	0.3600	PV

Line Flows

From Bus	To Bus	P (pu)	Q (pu)
1	2	0.4755	0.2554
2	1	-0.4722	-0.3015
1	4	0.3913	0.1966
4	1	-0.3867	-0.2178
2	5	-0.3619	-0.2192
5	2	0.3639	0.1581
3	5	-0.4659	-0.2527
5	3	0.4699	0.1904
4	5	-0.4133	-0.2822
5	4	0.4162	0.2363
2	3	0.2341	0.1708
3	2	-0.2341	-0.1673

Total Real Power Generation (pu) : 2.7668
Total Reactive Power Generation (pu) : 1.3968
Total Real Power Demand (pu) : 2.7500
Total Reactive Power Demand (pu) : 1.6300
Total Real Power Loss (pu) : 0.0168
Total Reactive Power Loss (pu) : -0.2332

EXPERIMENT 5

Aim: To develop a computer program for economic load dispatching among units using reduced gradient search method.

Requirements: Computer, MATLAB

Theory: The economic load dispatching is the problem to find the optimum solution of units scheduling such that the total cost of generation is minimized. The gradient search method or steepest descent method works on the principle that the minimum of a function can be found by a series of steps that always take us in a downward direction. If ∇f is the gradient of function *f(*x*)* at the initial guess x^0, then the updated x is given as,

$$x^1 = x^0 - \alpha \nabla f$$

where, α is the learning rate.

The learning rate is a scalar which allow us to guarantee that the process converges. Larger the learning rate will result into faster convergence, but sometimes may lead to divergence also. Smaller the learning rate will guarantee the convergence, but the process becomes too slow. Thus, the learning rate is set through experimenting with different learning rates for a problem.

Case Study:

Table 5.1: *Generator Data*

Unit	A (MBtu/MW²h)	B (MBtu/MWh)	C (MBtu/h)	Fuel Cost (Rs/MBtu)	Min Generation (MW)	Max Generation (MW)
1	0.0060	9	510	1.0	50	200
2	0.0048	6	310	1.0	100	400
3	0.0040	8	78	1.0	80	250
4	0.0034	10	100	1.0	110	300

Total Demand is of 800 MW

```
%----------Economic Load scheduling among units of a plant using----------
%-------------------------Gradient Search Method-------------------------


%Heat Rate Hi in MBtu/h for ith thermal unit is given as: Hi=A*Pi^2+B*Pi+C
%fuel cost is in Rs/MBtu

%--------------------------------------------------------------------------
%            Unit      A         B       C    fuel cost   min out  max out
%--------------------------------------------------------------------------
plant_data=[   1    0.00600     9.00    510     1.0          50       200 ;
               2    0.00480     6.00    310     1.0         100       400 ;
               3    0.00400     8.00    78      1.0          80       250 ;
               4    0.00340     10.0    100     1.0         110       300];

dmnd=800;                           %in MW
%------------------------opening file to write result----------------------
folder_path='C:\Users\VISHAL MITTAL\Desktop\PS_II Lab\';
fid=fopen(strcat(folder_path,'Exp5.txt'),'w');
fprintf(fid,'The Gradient Descent for the Economic Loading of Generators in a↙
Plant');
fprintf(fid,'\n \n');
fprintf(fid,'%4s \t','Itr');
str_dash='--------';
for unit=1:1:max(plant_data(:,1))
    str=strcat('P',num2str(unit));
    fprintf(fid,'%7s \t',str);
    str_dash=strcat(str_dash,'--------------');
end

fprintf(fid,'%7s \n','Gen. Cost');
str_dash=strcat(str_dash,'-------------');
fprintf(fid,str_dash);
fprintf(fid,'\n');
%--------------------------------------------------------------------------

%making cost function: Fi = Hi * fuel cost
plant_data(:,2)=plant_data(:,2).*plant_data(:,5);
plant_data(:,3)=plant_data(:,3).*plant_data(:,5);
plant_data(:,4)=plant_data(:,4).*plant_data(:,5);

x0=[20;20;20];                      %initial values: [Pg1; Pg2; Pg3]

a=10;                                %learning rate
dL0=zeros(length(x0),1);            %initially assumed gradient

%-----------------------running gradient descent--------------------------
for itr=1:1:100

    min_max_set_flag=0;  %0 if min_max limit on each Gen not set, else 1
    set_count=0;         %varbl count no. of times min-max limits setting
```

```
51                          %done over last gen which is reduced from gradient
52      for i=1:1:length(x0)
53          if(x0(i)<plant_data(i,6))      %if Pgi < Pgi_min
54              x0(i)=plant_data(i,6);     %set Pgi = Pgi_min
55          end
56          if(x0(i)>plant_data(i,7))      %if Pgi > Pgi_max
57              x0(i)=plant_data(i,7);     %set Pgi = Pgi_max
58          end
59      end
60
61      %note: the last generator that is reduced using the demand constraint :
62      % Pn = dmnd - Pg1 - Pg2 - Pg3, must satisfy its min and max limits of
63      %generation. If it is not satisfying then choose other generator for
64      %reduction process. For this shuffling in plant_data is done
65      while(min_max_set_flag==0)
66          if set_count>max(plant_data(:,1))
67              fprintf(fid,'Generation Limits are not meeting');
68              break;
69          end
70          Pn=dmnd-ones(1,length(x0))*x0;     %last generator output Pn= dmnd-Pg1-Pg2-Pg3
71          if Pn<plant_data(i+1,6)            %checking if Pn < Pmin limit
72              Pn=plant_data(i+1,6);          %set Pn = Pmin
73              %shuffling the last gen at 1st position in plant_data matrix
74              %and rest generators shifted down by one
75              plant_data=[zeros(1,7); plant_data];
76              plant_data(1,:)=plant_data(length(plant_data(:,1)),:);
77              plant_data(length(plant_data(:,1)),:)=[];
78              %similarly shuffling x0 vector also
79              x0(2:length(x0))=x0(1:length(x0)-1);
80              x0(1)=Pn;
81
82              set_count=set_count+1; %set_count incremented by 1
83              continue;              %continue to compute Pn for new gen
84                                     %which took place for gradient reduction
85          end
86
87          if Pn>plant_data(i+1,7)    %similarly if Pmax limit not satisfied
88              Pn=plant_data(i+1,7);
89              plant_data=[zeros(1,7); plant_data];
90              plant_data(1,:)=plant_data(length(plant_data(:,1)),:);
91              plant_data(length(plant_data(:,1)),:)=[];
92              x0(2:length(x0))=x0(1:length(x0)-1);
93              x0(1)=Pn;
94              set_count=set_count+1;
95              continue;
96          end
97          min_max_set_flag=1;
98      end
99      if min_max_set_flag==0         %if generator limits are not getting set
100         break;                     %break the loop
101     end
```

```
102     X=[x0;Pn];                      %set of Pout vector [P1; P2; P3;...]
103     cost=plant_data(:,2)'*(X.*X)+plant_data(:,3)'*X+plant_data(:,4)'*ones(length(X),↙
1);
104
105     %-------------------------------------------------------------------
106     %---------------------------display section--------------------------
107     Pgen=zeros(length(X),1);
108     for k=1:1:length(X)             %sorting generator no.in ascending order
109         Pgen(plant_data(k,1))=X(k);
110     end
111
112     fprintf(fid,'%4d \t %7.2f \t %7.2f \t %7.2f \t %7.2f \t %7.2f ',[itr Pgen'↙
cost]);
113     fprintf(fid,'\n');
114     %-------------------------------------------------------------------
115
116     %computing gradient for the current iteration
117     dL1=2.*(diag(plant_data(1:length(x0),2))+plant_data(length(X),2).*ones(length↙
(x0),length(x0)))*x0...
118     +plant_data(1:length(x0),3)-(plant_data(length(X),3)+2*dmnd*plant_data(length(X),↙
2)).*ones(length(x0),1);
119
120     x0=x0-a.*dL1;
121     %if current gradient and previous itr gradient are same - stop loop
122     if abs(dL1-dL0)<0.001
123         break;
124     end
125     dL0=dL1;
126 end
127 fclose('all');
```

The Gradient Descent for the Economic Loading of Generators in a Plant

Itr	P1	P2	P3	P4	Gen. Cost
1	50.00	200.00	250.00	300.00	8411.00
2	50.00	262.00	229.20	258.80	8223.95
3	50.00	307.63	216.02	226.35	8115.56
4	50.00	341.24	208.27	200.49	8050.61
5	50.00	366.02	204.36	179.62	8010.01
6	50.00	384.30	203.15	162.54	7983.30
7	50.89	396.93	203.79	148.38	7964.58
8	58.01	400.00	205.60	136.40	7949.32
9	71.25	398.56	206.11	124.08	7935.51
10	78.79	398.85	208.17	114.20	7925.99
11	79.03	400.00	210.97	110.00	7921.68
12	76.42	400.00	213.58	110.00	7921.07
13	74.34	400.00	215.66	110.00	7920.68
14	72.67	400.00	217.33	110.00	7920.43
15	71.34	400.00	218.66	110.00	7920.26
16	70.27	400.00	219.73	110.00	7920.16
17	69.42	400.00	220.58	110.00	7920.10
18	68.80	399.93	221.27	110.00	7920.05
19	68.38	399.79	221.82	110.00	7920.02
20	68.10	399.62	222.28	110.00	7920.00
21	67.90	399.43	222.67	110.00	7919.98
22	67.76	399.23	223.01	110.00	7919.97
23	67.67	399.04	223.30	110.00	7919.96
24	67.60	398.85	223.55	110.00	7919.95
25	67.55	398.67	223.78	110.00	7919.94
26	67.51	398.50	223.98	110.00	7919.93
27	67.48	398.35	224.17	110.00	7919.93
28	67.46	398.21	224.33	110.00	7919.92
29	67.45	398.07	224.48	110.00	7919.92
30	67.43	397.95	224.62	110.00	7919.92
31	67.42	397.84	224.74	110.00	7919.91
32	67.41	397.74	224.85	110.00	7919.91
33	67.40	397.64	224.95	110.00	7919.91

EXPERIMENT 6

Aim: To develop a computer program for economic load dispatching among plants using Newton's method.

Requirements: Computer, MATLAB

Theory: The economic load dispatching is the problem to find the optimum solution of units scheduling such that the total cost of generation is minimized. The Newton's method is based on gradient of the cost function in which the aim is to drive gradient to zero. If Δx is an effort to make the gradient $g(x)$ to zero at initial guess x^0, then

$$\Delta x = -[g'(x)]^{-1} g(x)$$

$$x^1 = x^0 + \Delta x$$

This method is good in optimizing the non-linear equations involving the coordination equations of economic loading of plants, which include the transmission losses which depend upon the plants generations.

Case Study:

Table 5.1: *Plant Data*

Unit	A (MBtu/MW²h)	B (MBtu/MWh)	C (MBtu/h)	Fuel Cost (Rs/MBtu)	Min Generation (MW)	Max Generation (MW)
1	0.00533	11.669	213.1	1.0	50	200
2	0.00889	10.333	200	1.0	37.5	150
3	0.00741	10.833	240	1.0	45	180

Total Demand is of 210 MW

Loss Coefficient matrix in pu,

$$\boldsymbol{B} = \begin{bmatrix} 0.0676 & 0.00953 & -0.00507 & -0.0383 \\ 0.00953 & 0.0521 & 0.00901 & -0.00171 \\ -0.00507 & 0.00901 & 0.0294 & 0.00945 \\ -0.0383 & -0.00171 & 0.00945 & 0.040357 \end{bmatrix} pu$$

```
%----------Economic Load scheduling among units of a plant using----------
%--------------------------Newton's Method-------------------------


%Heat Rate Hi in MBtu/h for ith thermal unit is given as: Hi=A*Pi^2+B*Pi+C
%fuel cost is in Rs/MBtu

%-------------------------------------------------------------------------
%            Unit      A         B       C    fuel cost   min out  max out
%-------------------------------------------------------------------------
plant_data=[   1    0.00533   11.669    213.1     1.0       50.0      200 ;
               2    0.00889   10.333    200.0     1.0       37.5      150 ;
               3    0.00741   10.833    240.0     1.0       45.0      180];

dmnd=210;                           %in MW
N=length(plant_data(:,1));  %no. of plants in the system
%Loss coefficient matrix in pu: B=[B11     B12    ... B10/2
%                                  B21     B22    ... B20/2
%                                   :       :          :
%                                  B10/2   B20/2  ... B00   ]
B=[ 0.0676   0.00953 -0.00507    -0.0383 ;
    0.00953  0.0521   0.00901    -0.00171;
   -0.00507  0.00901  0.0294      0.00945 ;
   -0.0383  -0.00171  0.00945     0.040357];
Sb=100;                               %base MVA

%Loss Coefficient matrix not in pu
B(1:N,1:N)=B(1:N,1:N)./Sb;
B(N+1,N+1)=B(N+1,N+1).*Sb;

%------------------------opening file to write result----------------------
folder_path='C:\Users\VISHAL MITTAL\Desktop\PS_II Lab\';
fid=fopen(strcat(folder_path,'Exp_6.txt'),'w');
fprintf(fid,'The Newton''s Method for the Economic Loading of Plants considering↙
Transmission Losses');
fprintf(fid,'\n \n');
fprintf(fid,'%4s \t','Itr');
str_dash='--------';
for unit=1:1:max(plant_data(:,1))
    str=strcat('P',num2str(unit));
    fprintf(fid,'%7s \t',str);
    str_dash=strcat(str_dash,'--------------');
end
fprintf(fid,'%7s \t','lambda');
str_dash=strcat(str_dash,'--------------');
fprintf(fid,'%7s \t','Gen. Cost');
str_dash=strcat(str_dash,'-------------');
fprintf(fid,'%7s \n','TL Loss');
str_dash=strcat(str_dash,'-------------');
fprintf(fid,str_dash);
fprintf(fid,'\n');
```

```
 51 %--------------------------------------------------------------------------
 52 %making cost function: Fi = Hi * fuel cost: updating the given plant data
 53 %matrix for the values of Ai, Bi and Ci
 54 plant_data(:,2)=plant_data(:,2).*plant_data(:,5);
 55 plant_data(:,3)=plant_data(:,3).*plant_data(:,5);
 56 plant_data(:,4)=plant_data(:,4).*plant_data(:,5);
 57
 58 x0=zeros(N+1,1);
 59 x1=[75.*ones(N,1);10];    %initial guess: [Pg1; Pg2; ...; lambda]
 60
 61 %--------------------Running Newton's Iterative Method---------------------
 62 itr=0;
 63 Ploss=[x1(1:N);1]'*(B*[x1(1:N);1]);
 64 cost=plant_data(:,2)'*(x1(1:N).^2)+plant_data(:,3)'*x1(1:N)+plant_data(:,4)'*ones(N,↙
1);
 65 %---------------------storing result for 0th iteration---------------------
 66 frmt='%7.2f \t ';
 67 res_frmt='%4d \t ';
 68 for j=1:1:N
 69     res_frmt=strcat(res_frmt,frmt);
 70 end
 71 res_frmt=strcat(res_frmt,'%7.3f \t ','%7.2f \t','%5.3f');
 72 fprintf(fid,res_frmt,[itr x1' cost  Ploss]);
 73 fprintf(fid,'\n');
 74
 75 while(itr<10)
 76     itr=itr+1;
 77     if(abs(x1-x0)<0.0001)      %if tolerance meets then break the loop
 78         break;
 79     end
 80     %computing trans. losses: [Pg1 Pg2 ... 1]*[B}*[Pg1; PG2; ... ;1]
 81     lambda=x1(length(x1));
 82     Ploss=[x1(1:N);1]'*(B*[x1(1:N);1]);
 83
 84     %computing partial derivative dPloss/dPgi for ith plant generation
 85     %dPloss/dPgi= Bii*Pgi + from k=1 to n [sum(Bik*Pgk)] + Bi0
 86     sum_Bik_Pgk=B(1:N,1:N)*x1(1:N);
 87     dPloss_dPgi=diag(B(1:N,1:N)).*x1(1:N)+sum_Bik_Pgk+2.*B(1:N,N+1);
 88
 89     %computing Gradient matrix g(x) at x=x1
 90     %g(x)=[2Ai*Pgi + Bi + lambda*(dPloss/dPgi -1); Pd + Ploss - sum(Pgk)]
 91     gx=2.*plant_data(:,2).*x1(1:N)+plant_data(:,3)+lambda.*(dPloss_dPgi-ones(1:N,1));
 92     gx=[gx; dmnd+Ploss-sum(x1(1:N))];
 93
 94     %computing g'(x):
 95     %g'(x)=[2A1+2B11*lambda   B12*lambda    ... B1n*lambda  dPloss/dPg1-1
 96     %       B21*lambda      2A2+2B22*lambda ... B2n*lambda  dPloss/dPg2-1
 97     %          :                 :               :            :
 98     %       Bn1*lambda        Bn2*lambda  ... 2An+2Bnn*lambda dPloss/dPgn-1
 99     %       dPloss/dPg1-1   dPloss/dPg2-1  ... dPloss/dPgn-1      0       ]
100
```

```
101     %g'(x)=[2[diag(A)]+lambda*{[B(1:N)]+[diag(B(1:N))]   [dPloss/dPg -1]
102     %          [dPloss/dPg -1]'                                  0         ]
103     dgx=[2.*diag(plant_data(:,2))+lambda.*(B(1:N,1:N)+diag(diag(B(1:N,1:N))))↙
dPloss_dPgi-ones(1:N,1);
104             (dPloss_dPgi-ones(1:N,1))'                                      0↙
];
105
106     %computing dX= -inv((g'(x)))*g(x) at x=x1
107     dX=-inv(dgx)*gx;
108     x0=x1;
109     x1=x0+dX;
110     cost=plant_data(:,2)'*(x1(1:N).^2)+plant_data(:,3)'*x1(1:N)+plant_data(:,4)'*ones↙
(N,1);
111
112     %-----------------storing results for current iteration-----------------
113     res_frmt=strcat(res_frmt,'%5.3f \t ','%7.2f \t','%5.3f');
114     fprintf(fid,res_frmt,[itr x1' cost  Ploss]);
115     fprintf(fid,'\n');
116 end
117 fclose('all');
```

The Newton's Method for the Economic Loading of Plants considering Transmission Losses

Itr	P1	P2	P3	lambda	Gen. Cost	TL Loss
0	75.00	75.00	75.00	10.000	3237.39	9.354
1	71.98	73.62	73.45	12.754	3165.14	9.354
2	72.14	73.59	73.28	12.747	3164.81	9.019
3	72.14	73.59	73.28	12.747	3164.81	9.009
4	72.14	73.59	73.28	12.747	3164.81	9.009

EXPERIMENT 7

Aim: To develop a computer program for stability analysis of a single machine infinite bus system using Runge-Kutta fourth order method.

Requirements: Computer, MATLAB

Theory: Transient stability is the ability of the system to regain synchronism after a large and sudden disturbance. The large disturbance can occur due to sudden changes in application or removal of large loads, line switching operations, faults on the system, sudden outage of a line, or loss of excitations. Transient stability studies are needed to ensure that the system can withstand the transient conditions following a major disturbance.

The swing curve indicates about the variation of torque angle with time following a disturbance. If torque angle keeps on increasing, the system is unstable. For the system to be stable followed by a disturbance such as fault, the fault must be cleared before the critical clearing time such that the torque angle decreases after achieving the maximum swing. The swing equation is given as,

$$M\frac{d^2\delta}{dt^2} = P_m - P_{emax}sin\delta$$

The state space model of swing equation is given as,

$$\dot{x}_1 = x_2 = \dot{\delta} \quad \text{and} \quad \dot{x}_2 = \frac{1}{M}(P_m - P_{emax}sinx_1)$$

To obtain the solution of above two first order differential equations via Runge-Kutta fourth order method:

$$k_1 = hx_2^0 \qquad l_1 = \frac{h}{M}(P_m - P_{emax}sinx_1^0)$$

$$k_2 = h\left(x_2^0 + \frac{l_1}{2}\right) \qquad l_2 = \frac{h}{M}\left(P_m - P_{emax}sin\left(x_1^0 + \frac{k_1}{2}\right)\right)$$

$$k_3 = h\left(x_2^0 + \frac{l_2}{2}\right) \qquad l_3 = \frac{h}{M}\left(P_m - P_{emax}sin\left(x_1^0 + \frac{k_2}{2}\right)\right)$$

$$k_4 = h(x_2^0 + l_3) \qquad l_4 = \frac{h}{M}\left(P_m - P_{emax}sin(x_1^0 + k_3)\right)$$

$$k = \frac{k_1 + 2k_2 + 2k_3 + k_4}{6} \qquad l = \frac{l_1 + 2l_2 + 2l_3 + l_4}{6}$$

$$x_1^1 = x_1^0 + k \qquad x_2^1 = x_2^0 + l \quad ; x_1^0 = \delta_0 \;\; and \;\; x_2^0 = 0 \; and \; h \; is \; the \; time \; step$$

Case Study:

A 20 MVA, 50 Hz generator delivers 18 MW over a double circuit line to an infinite bus. The generator has kinetic energy of 2.52 MJ/MVA at rated speed. The generator transient reactance is 0.35 pu. Each transmission circuit has a reactance of 0.2 pu on a 20 MVA base. The generator voltage behind the transient reactance is 1.1 pu and infinite bus voltage is 1 pu. A three-phase short circuit occurs at the mid-point of one of the transmission lines. Carry out the swing equation solution using Runge-Kutta fourth order method to plot swing curves for different fault clearing times.

```
%----------------------SMIB Transient Stability Study-----------------------
%-------Plotting Swing Curve using Runge-Kutta 4th Order Method------------

H=2.52;          %machine's inertia constant in MJ/MVA
G=20;            %machine's rating in MVA
f=50;            %freq. in Hz
%------------------computing inertia constant M in pu----------------------
M=H/(180*f);
%------------------------------------------------------------------------

Xpre=0.45;       %prefault reactance in pu
Xf=1.25;         %during fault reactance in pu
Xpost=0.55;      %postfault reactance in pu
Ef=1.1;          %voltage behind transient reactance of alternator in pu
V=1;             %infinite bus voltage in pu
%-------------computing Pemax pre/during/post fault conditions-------------
Pemax1=Ef*V/Xpre;
Pemax2=Ef*V/Xf;
Pemax3=Ef*V/Xpost;
%------------------------------------------------------------------------
Pm=0.9;          %mechanical power in pu

%---fault clearing times in sec, assuming time at which fault occurs=0s----
Tc=6.25*0.02;           %clears after 2.5 cycles
%------------------------------------------------------------------------
%--------------opening a text file to store the results--------------------
folder_path='C:\Users\VISHAL MITTAL\Desktop\PS_II Lab\';
fid=fopen(strcat(folder_path,'Exp_7.txt'),'w');
fprintf(fid,'%s \n\n','The solution of Swing Equation using Runge-Kutta fourth order↙
method as obtained');

fprintf(fid,'Fault clearing time in sec: ');
fprintf(fid,'%4.4f \n\n',Tc);

fprintf(fid,'%4s','time');
fprintf(fid,'%10s','Pemax');
fprintf(fid,'%12s','x1_old');
fprintf(fid,'%10s','Pa');
fprintf(fid,'%12s','x2_old');
fprintf(fid,'%12s','dx2');
fprintf(fid,'%12s','x2_new');
fprintf(fid,'%10s','dx1');
fprintf(fid,'%15s\n','delta(deg)');
dash_print='-';
for j=1:1:95
    dash_print=strcat(dash_print,'-');
end
fprintf(fid,'%s\n',dash_print);

%the swing equation is transformed into state-space model as
%dx1/dt=x2 ;   dx2/dt=[Pm-Pemax sin(x1)]/M
```

```
%x10=delta0 i.e. prefault operating torque angle i.e. asin(Pm/Pemax1)
%x20= avg of [Pm-Pemax1 sin(x10)]/M and [Pm-Pemax2 sin(x10)]/M
%Pm-Pemax sin(x1) is the accelerating power Pa

x10=asind(Pm/Pemax1);        %initial x10=delta0
x20=0;                       %initial x20 i.e. change in speed = 0

h=0.001;                %taking step size of 0.001 sec for the RK method
dx1=@(x2) x2;
dx2=@(Pemax,t0,x1) (Pm-Pemax*sind(x1))/M;

%--opening another text file to store only x1 - delta for plotting---------
plot_fid=fopen(strcat(folder_path,'Exp7_delta.txt'),'a');
fprintf(plot_fid,'%4.3f',Tc);
fprintf(plot_fid,'%10.4f',x10);
%at the time of fault, taking dx2 as avg of dx2 before fault i.e. 0 and dx2
%after fault i.e. (Pm-Pemax2*sind(x1))/M and x20=0+dx20
dx20=(x20+dx2(Pemax2,0,x10))/2;
x20=h*dx20;

fprintf(fid,'%4.3f',0);
fprintf(fid,'%10.3f',Pemax1);
fprintf(fid,'%12.4f',x10);
fprintf(fid,'%10.4f',dx20*M);
fprintf(fid,'%12.4f',0);
fprintf(fid,'%12.4f',dx20);
fprintf(fid,'%12.4f',x20);
fprintf(fid,'%10.4f',0);
fprintf(fid,'%13.4f\n',x10);

print_count=0;

for t=0:h:0.54           %obtaining swing curve for 0.6 sec
    print_count=print_count+1;
    if t<Tc              %fault is not cleared yet
        Pemax=Pemax2;
    elseif t==Tc         %at the time of fault clearing, taking Pa avg of Pa before↙
fault clearing and after fault clearing
        dx20=(dx2(Pemax2,t,x10)+dx2(Pemax3,t,x10))/2;
        x20=x20+h*dx20;
        Pemax=Pemax3;
    else
        Pemax=Pemax3;   %fault gets cleared
    end
        k1=h*dx1(x20);
        l1=h*dx2(Pemax,t,x10);

        k2=h*dx1(x20+l1/2);
        l2=h*dx2(Pemax,t+h/2,x10+k1/2);

        k3=h*dx1(x20+l2/2);
```

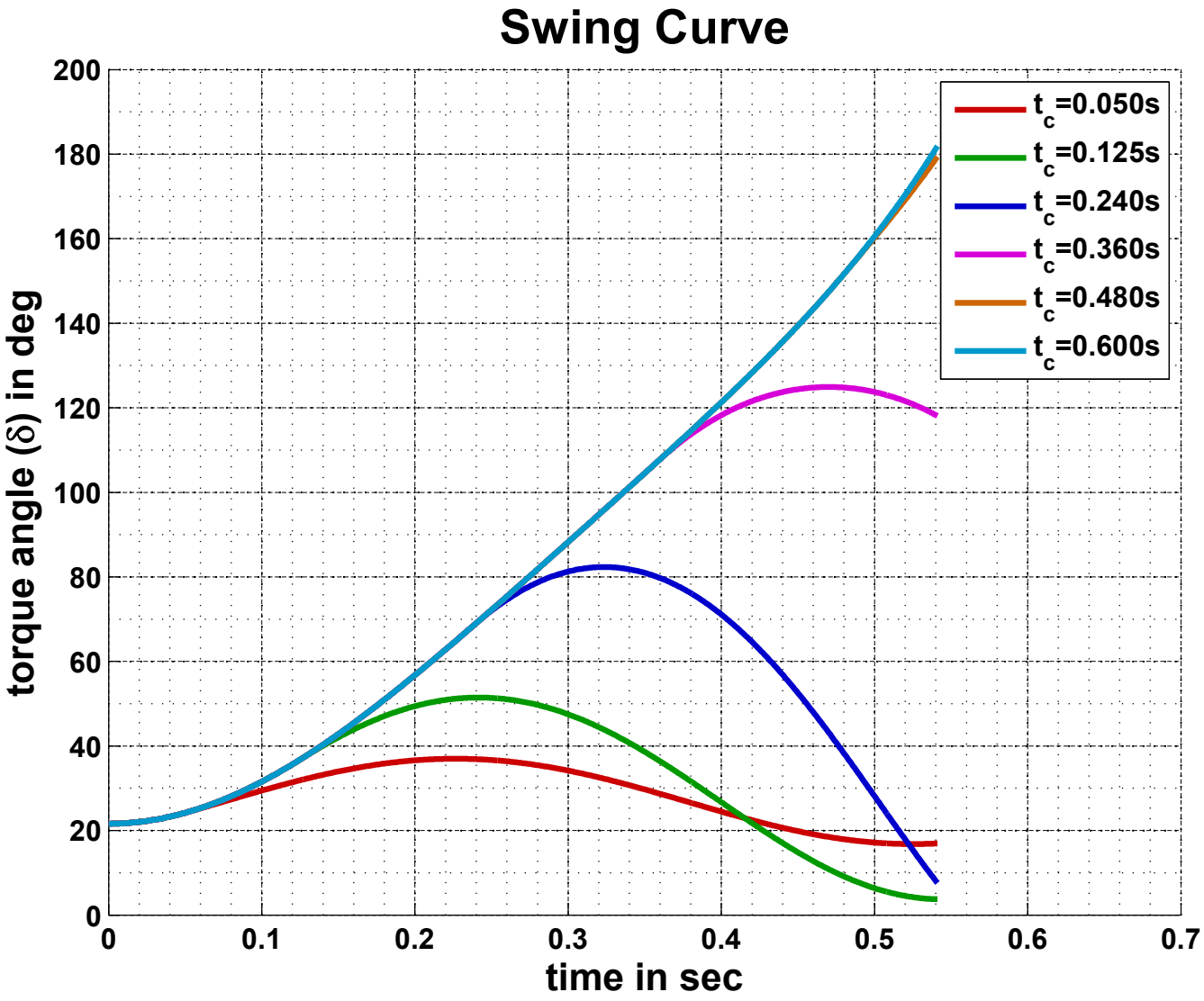
```
101         l3=h*dx2(Pemax,t+h/2,x10+k2/2);
102
103         k4=h*dx1(x20+l3);
104         l4=h*dx2(Pemax,t+h,x10+k3);
105
106         x11=x10+(k1+2*k2+2*k3+k4)/6;
107         x21=x20+(l1+2*l2+2*l3+l4)/6;
108
109         t1=t+h;
110         if print_count==10      %to print result after interval of 0.01 sec
111             print_count=0;
112             fprintf(fid,'%4.3f',t1);
113             fprintf(fid,'%10.3f',Pemax);
114             fprintf(fid,'%12.4f',x10);
115             if t~=Tc
116                 fprintf(fid,'%10.4f',dx2(Pemax,t,x10)*M);
117             else
118                 fprintf(fid,'%10.4f',dx20*M);   %Pa is avg of Pa before and after↙
fault clearing
119             end
120             fprintf(fid,'%12.4f',x20);
121             fprintf(fid,'%12.4f',x21-x20);   %it represents l=(l1+2l2+2l3+l4)/6
122             fprintf(fid,'%12.4f',x21);
123             fprintf(fid,'%10.4f',x11-x10);   %it represents k=(k1+2k2+2k3+k4)/6
124             fprintf(fid,'%13.4f\n',x11);
125         end
126
127         %updating x10 and x20 for next time t1
128         x10=x11;
129         x20=x21;
130         fprintf(plot_fid,'%10.4f',x10);
131 end
132 fprintf(plot_fid,'\n');
133 fclose('all');
134
135 %--opening the file Exp7_Delta.txt to plot the swing curves for different--
136 %---------------------------fualt clearing times----------------------------
137 plot_fid=fopen('C:\Users\VISHAL MITTAL\Desktop\PS_II Lab\Exp7_delta.txt','r');
138 m=(t/h+1)+2;        %no. of elements in a line of Exp7_delta.txt = no. of deltas + Tc↙
+ \n
139 delta=fscanf(plot_fid,'%f',[m,inf]);
140 tm=0:h:0.54+h;
141 flt_clr_tm=delta(1,:);
142 delta(1,:)=[];
143 tc_legnd=[];
144 for x=1:1:length(flt_clr_tm)
145     tc_legnd=[tc_legnd;cellstr(strcat('t_c=',num2str(flt_clr_tm(x),'%4.3f'),'s'))];
146 end
147
148 fig=figure(1);
149 set(fig,'color','white');
```

```
150 clr=[0.8   0     0    ;
151      0     0.6   0    ;
152      0     0     0.8  ;
153      0.85  0     0.85 ;
154      0.8   0.4   0    ;
155      0     0.6   0.8] ;
156 hold on;
157 for plt=1:1:length(flt_clr_tm)
158     plot(tm',delta(:,plt),'linewidth',2.5,'color',clr(plt,:));
159 end
160 title('Swing Curve','fontsize',20,'fontweight','bold');
161 xlabel('time in sec','fontsize',16,'fontweight','bold');
162 ylabel('torque angle (\delta) in deg','fontsize',16,'fontweight','bold');
163 set(gca,'fontsize',12,'fontweight','bold','gridlinestyle','--');
164 legend(tc_legnd);
165 grid on;
166 grid(gca,'minor');
167 fclose('all');
```

The solution of Swing Equation using Runge-Kutta fourth order method as obtained

Fault clearing time in sec: 0.1250

time	Pemax	x1_old	Pa	x2_old	dx2	x2_new	dx1	delta(deg)
0.000	2.444	21.6035	0.2880	0.0000	1028.5714	1.0286	0.0000	21.6035
0.010	0.880	21.6961	0.5747	19.5280	2.0519	21.5799	0.0206	21.7166
0.020	0.880	21.9938	0.5704	39.9851	2.0362	42.0213	0.0410	22.0348
0.030	0.880	22.4951	0.5633	60.2390	2.0103	62.2493	0.0612	22.5563
0.040	0.880	23.1975	0.5534	80.1879	1.9743	82.1622	0.0812	23.2787
0.050	0.880	24.0975	0.5407	99.7328	1.9286	101.6614	0.1007	24.1982
0.060	0.880	25.1905	0.5254	118.7787	1.8736	120.6523	0.1197	25.3102
0.070	0.880	26.4711	0.5077	137.2355	1.8100	139.0455	0.1381	26.6093
0.080	0.880	27.9330	0.4878	155.0191	1.7383	156.7574	0.1559	28.0889
0.090	0.880	29.5690	0.4657	172.0520	1.6593	173.7113	0.1729	29.7419
0.100	0.880	31.3713	0.4419	188.2648	1.5738	189.8386	0.1891	31.5603
0.110	0.880	33.3313	0.4165	203.5967	1.4827	205.0794	0.2043	33.5357
0.120	0.880	35.4401	0.3897	217.9964	1.3870	219.3834	0.2187	35.6588
0.130	2.000	37.6693	-0.3222	221.8808	-1.1617	220.7191	0.2213	37.8906
0.140	2.000	39.8270	-0.3809	209.3091	-1.3705	207.9386	0.2086	40.0356
0.150	2.000	41.8488	-0.4343	194.7343	-1.5602	193.1741	0.1940	42.0427
0.160	2.000	43.7156	-0.4822	178.3516	-1.7300	176.6216	0.1775	43.8931
0.170	2.000	45.4105	-0.5243	160.3619	-1.8795	158.4824	0.1594	45.5699
0.180	2.000	46.9182	-0.5608	140.9688	-2.0087	138.9601	0.1400	47.0582
0.190	2.000	48.2258	-0.5916	120.3751	-2.1177	118.2575	0.1193	48.3452
0.200	2.000	49.3224	-0.6168	98.7814	-2.2068	96.5746	0.0977	49.4200
0.210	2.000	50.1988	-0.6365	76.3846	-2.2764	74.1082	0.0752	50.2740
0.220	2.000	50.8480	-0.6509	53.3779	-2.3269	51.0511	0.0522	50.9002
0.230	2.000	51.2649	-0.6601	29.9509	-2.3586	27.5923	0.0288	51.2937
0.240	2.000	51.4463	-0.6640	6.2901	-2.3718	3.9183	0.0051	51.4514
0.250	2.000	51.3906	-0.6628	-17.4196	-2.3666	-19.7862	-0.0186	51.3720
0.260	2.000	51.0983	-0.6564	-40.9937	-2.3428	-43.3366	-0.0422	51.0562
0.270	2.000	50.5718	-0.6448	-64.2467	-2.3004	-66.5472	-0.0654	50.5064
0.280	2.000	49.8151	-0.6279	-86.9907	-2.2391	-89.2298	-0.0881	49.7270
0.290	2.000	48.8343	-0.6056	-109.0347	-2.1584	-111.1931	-0.1101	48.7242
0.300	2.000	47.6374	-0.5778	-130.1835	-2.0580	-132.2416	-0.1312	47.5062
0.310	2.000	46.2343	-0.5443	-150.2386	-1.9376	-152.1761	-0.1512	46.0831
0.320	2.000	44.6369	-0.5052	-168.9979	-1.7968	-170.7947	-0.1699	44.4670
0.330	2.000	42.8593	-0.4604	-186.2581	-1.6357	-187.8938	-0.1871	42.6722
0.340	2.000	40.9174	-0.4099	-201.8165	-1.4545	-203.2711	-0.2025	40.7149
0.350	2.000	38.8293	-0.3540	-215.4744	-1.2538	-216.7282	-0.2161	38.6132
0.360	2.000	36.6149	-0.2929	-227.0406	-1.0346	-228.0752	-0.2276	36.3874
0.370	2.000	34.2961	-0.2269	-236.3364	-0.7983	-237.1347	-0.2367	34.0593
0.380	2.000	31.8963	-0.1568	-243.2000	-0.5470	-243.7470	-0.2435	31.6528
0.390	2.000	29.4407	-0.0830	-247.4917	-0.2831	-247.7749	-0.2476	29.1930
0.400	2.000	26.9554	-0.0066	-249.0992	-0.0097	-249.1089	-0.2491	26.7063
0.410	2.000	24.4679	0.0716	-247.9415	0.2699	-247.6716	-0.2478	24.2201
0.420	2.000	22.0060	0.1506	-243.9737	0.5519	-243.4218	-0.2437	21.7623
0.430	2.000	19.5978	0.2292	-237.1894	0.8324	-236.3570	-0.2368	19.3610
0.440	2.000	17.2715	0.3062	-227.6229	1.1071	-226.5158	-0.2271	17.0444
0.450	2.000	15.0544	0.3805	-215.3501	1.3720	-213.9781	-0.2147	14.8397
0.460	2.000	12.9731	0.4510	-200.4880	1.6229	-198.8651	-0.1997	12.7734
0.470	2.000	11.0527	0.5166	-183.1935	1.8561	-181.3374	-0.1823	10.8705
0.480	2.000	9.3167	0.5762	-163.6603	2.0679	-161.5923	-0.1626	9.1541
0.490	2.000	7.7862	0.6290	-142.1162	2.2553	-139.8609	-0.1410	7.6452
0.500	2.000	6.4802	0.6743	-118.8189	2.4155	-116.4035	-0.1176	6.3626
0.510	2.000	5.4148	0.7113	-94.0516	2.5460	-91.5055	-0.0928	5.3220
0.520	2.000	4.6031	0.7395	-68.1183	2.6452	-65.4730	-0.0668	4.5363
0.530	2.000	4.0552	0.7586	-41.3395	2.7117	-38.6278	-0.0400	4.0152
0.540	2.000	3.7780	0.7682	-14.0472	2.7445	-11.3028	-0.0127	3.7653


Swing Curve
torque angle (δ) in deg
time in sec
0
20
40
60
80
100
120
140
160
180
200
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
t_c=0.050s
t_c=0.125s
t_c=0.240s
t_c=0.360s
t_c=0.480s
t_c=0.600s

EXPERIMENT 8

Aim: To study transient stability of a single machine infinite bus system using Simulink.

Requirements: Computer, MATLAB

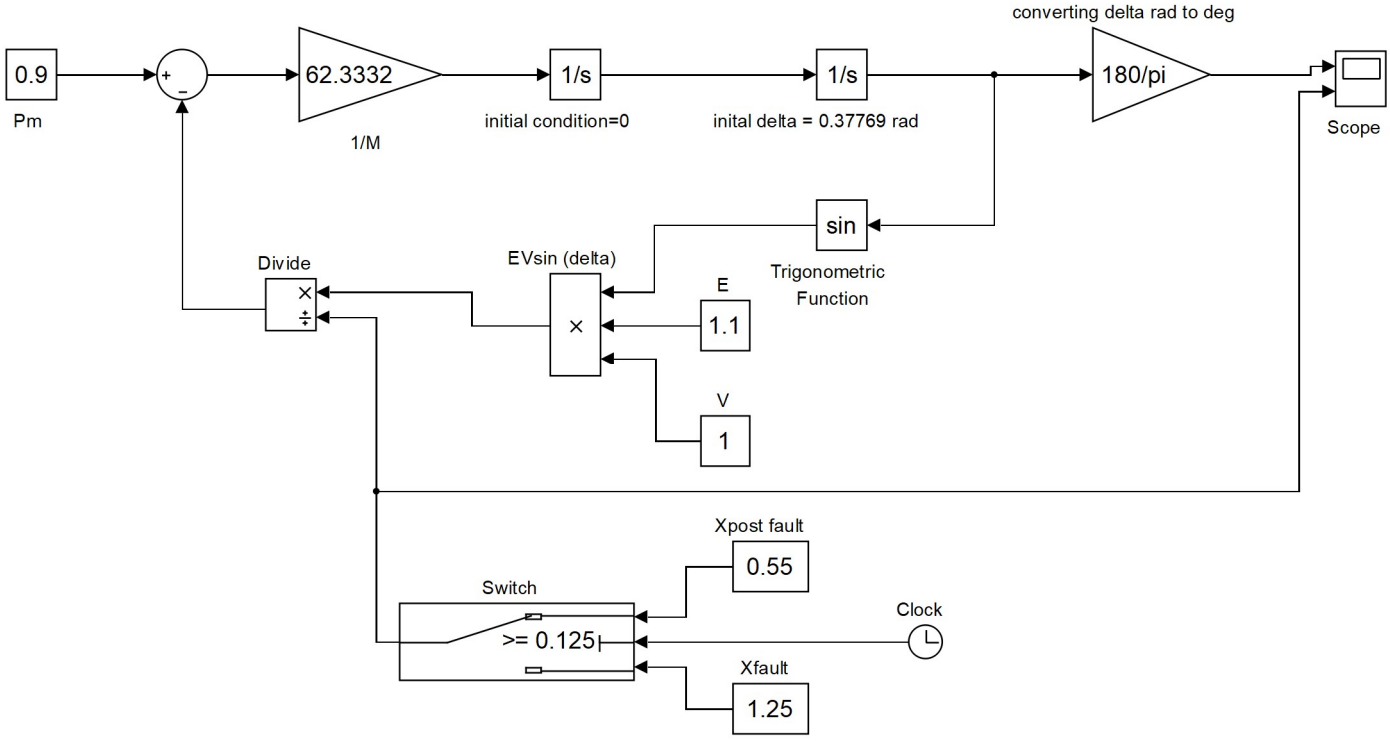
Theory: Transient stability is the ability of the system to regain synchronism after a large and sudden disturbance. SIMULINK is a software package developed by MathWorks which is used for modelling and simulating dynamical systems which can be linear and nonlinear, either in continuous time frame or sampled time frame or even a hybrid of the two. It provides a very easy drag-drop type GUI to build the models in block diagram form.

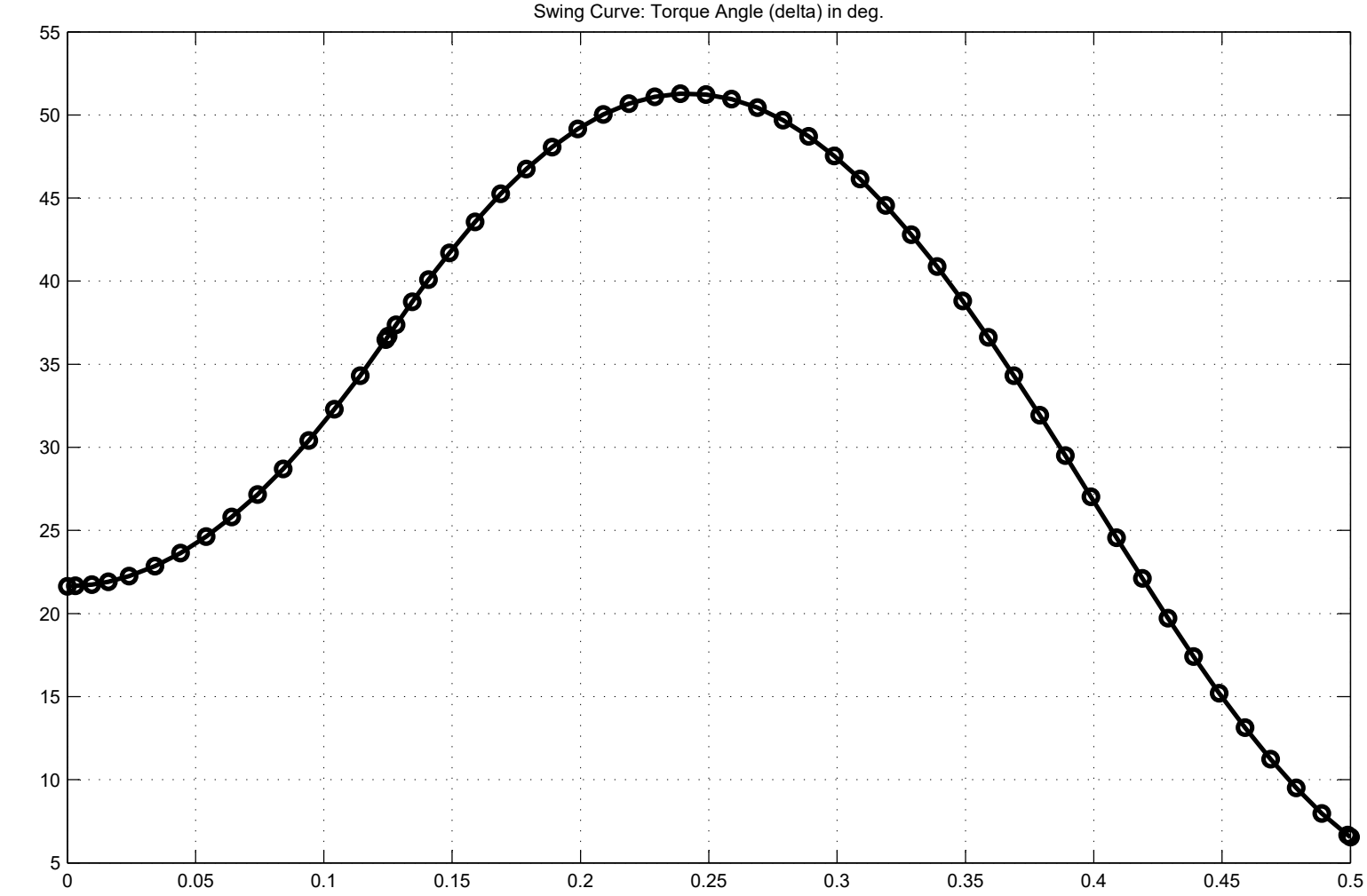
Case Study:

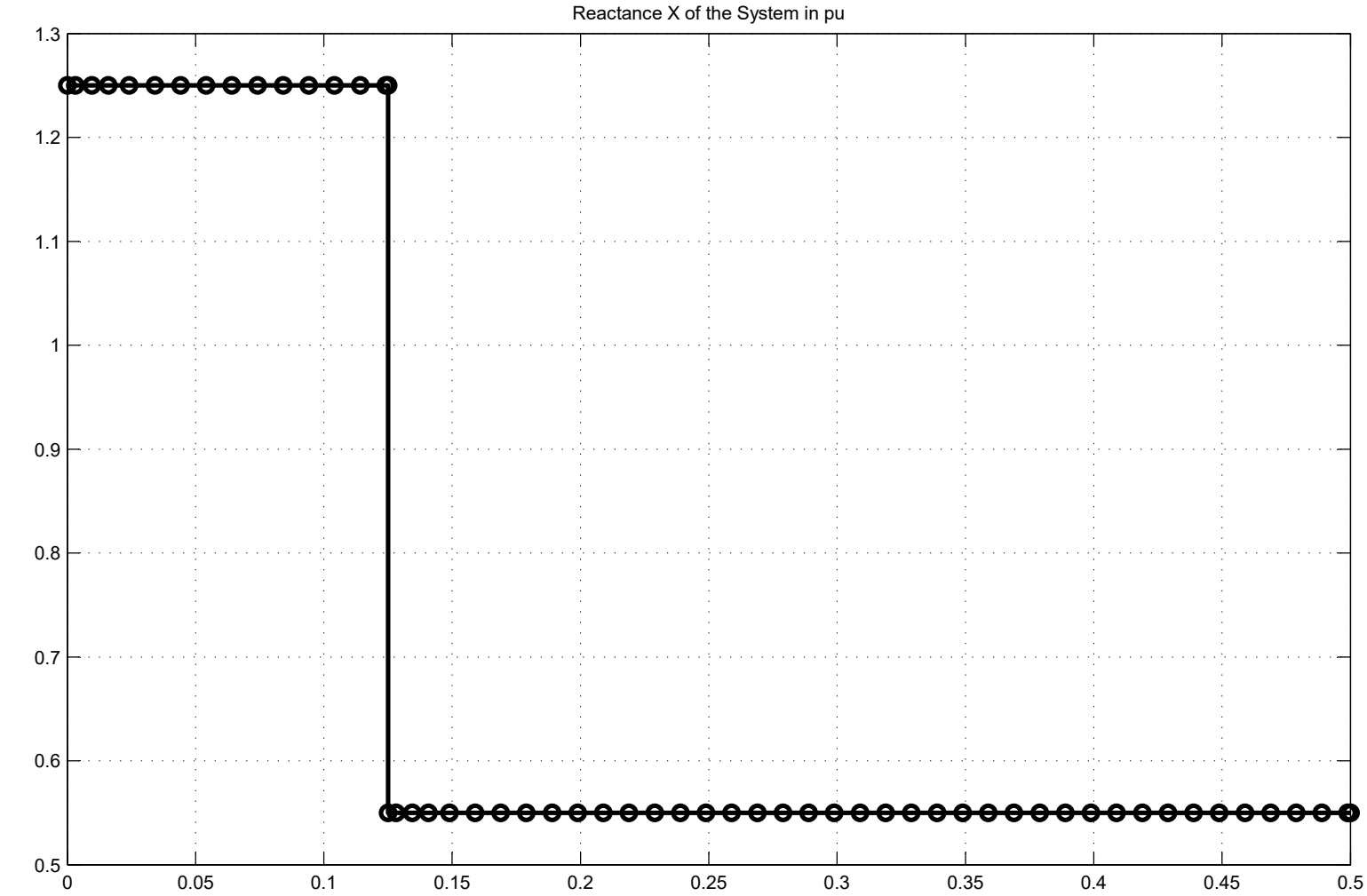
A 20 MVA, 50 Hz generator delivers 18 MW over a double circuit line to an infinite bus. The generator has kinetic energy of 2.52 MJ/MVA at rated speed. The generator transient reactance is 0.35 pu. Each transmission circuit has a reactance of 0.2 pu on a 20 MVA base. The generator voltage behind the transient reactance is 1.1 pu and infinite bus voltage is 1 pu. A three-phase short circuit occurs at the mid-point of one of the transmission lines. Carry out the swing equation solution using Simulink.

Different Blocks to be used:

- **Sum:** found under Math Operations in Simulink library. It is used to add two or more inputs.
- **Gain:** found under Math Operations in Simulink library. It is used to scale the given input.
- **Product:** found under Math Operations in Simulink library. It is used to take product of two or more inputs.
- **Divide:** found under Math Operations in Simulink library. It is used to divide two inputs.
- **Scope:** found under Sinks in Simulink library. It is used to plot a quantity with respect to time.
- **Switch:** found in Commonly Used Blocks in Simulink Library. It has 3 inputs. The 2^{nd} input is the control input. If this 2^{nd} input value is greater than or less than (as per logic selected) the threshold value, the input 1^{st} is allowed to pass through, else input 3^{rd} is passed through.
- **Clock:** found under Sources in Simulink library. It is used to supply time as input.
- **Constant:** found under Sources in Simulink library. It is used to provide a constant input.
- **Integrator:** found under Continuous in Simulink library. It is used to take integration of an input.
- **Trigonometric function:** found under Math Operations in Simulink library. It is used to insert trigonometric function. The angle taken is in rad.


converting delta rad to deg
0.9
Pm
62.3332
1/M
1/s
initial condition=0
1/s
inital delta = 0.37769 rad
180/pi
Scope
sin
Trigonometric
Function
Divide
EVsin (delta)
E
1.1
V
1
Xpost fault
0.55
Switch
>= 0.125
Clock
Xfault
1.25





Time offset: 0