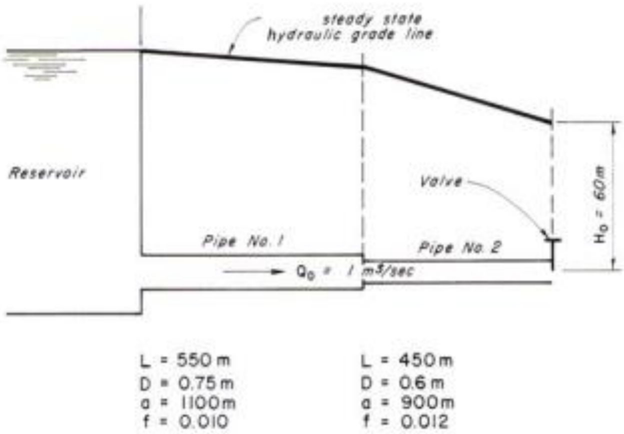
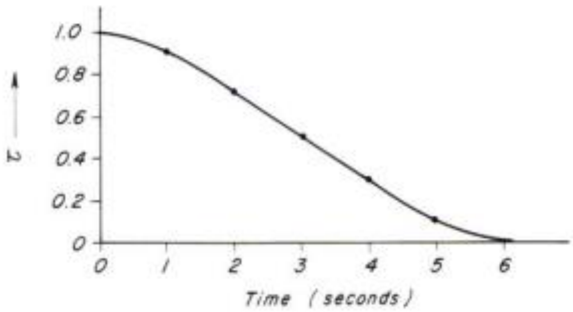
(1) Hydraulic Transient Problem

#Error: Level small.



(a) Piping system



(b) Valve closure curve

Fig. 3-23. **Series piping system.**

```
clc;

clear all;
L=[550 450];
D=[0.75 0.6];
a=[1100 900];
f=[0.01 0.012];
g=9.81;
Qo=1; % steady state discharge
Num_pipe=2; % number of pipes
Num_reach=2; %number of reach at which each pipe is
divided into
Num_node=Num_pipe*(Num_reach+1);
H_ds=60; % reservior head, m
T_last=10; % time upto which computations should be done
in sec
delta_t=L(1)/(a(1)*Num_reach); % in sec
Num_timestep=T_last/delta_t;
Num_timenode=Num_timestep+1;

%%
% pipe cs area
CS_area=zeros(1,2);
CS_area(1)=(pi/4)*D(1)^2;
CS_area(2)=(pi/4)*D(2)^2;

% x coordinate vector
x_pipe1=linspace(0,L(1),3);
x_pipe2=linspace(L(1),L(1)+L(2),3);
x_vector=[x_pipe1 x_pipe2];

%pipe coff Ca
Ca=zeros(1,2);
Ca(1)=(g*CS_area(1))/a(1);
Ca(2)=(g*CS_area(2))/a(2);

% coff R
R=zeros(1,2);
R(1)=f(1)/(2*D(1)*CS_area(1));
R(2)=f(2)/(2*D(2)*CS_area(2));

% Valve opening condition
Tau=zeros(1,Num_timenode);
```

```
Tau(1)=1;Tau(5)=0.9;Tau(9)=0.7;Tau(13)=0.5;Tau(17)=0.3;
Tau(21)=0.1;Tau(25)=0;
for i=0:1:5
    Tau(4*i+2)=Tau(4*i+1)+(Tau(4*i+5)-Tau(4*i+1))*(1/4);
 Tau(4*i+3)=Tau(4*i+1)+(Tau(4*i+5)-Tau(4*i+1))*(2/4);
  Tau(4*i+4)=Tau(4*i+1)+(Tau(4*i+5)-Tau(4*i+1))*(3/4);
end
%% Calculation for steady state conditions
H=zeros(Num_timenode,Num_node);
Q=zeros(Num_timenode,Num_node);
Q(1,:)=Qo;
H(1,6)=H_ds;
H(1,5)=H_ds+(f(2)*(x_vector(6)-
x_vector(5))*Qo^2)/(2*D(2)*g*CS_area(2)^2);
H(1,4)=H(1,5)+(f(2)*(x_vector(5)-
x_vector(4))*Qo^2)/(2*D(2)*g*CS_area(2)^2);
H(1,3)=H(1,4);
H(1,2)=H(1,3)+(f(1)*(x_vector(3)-
x_vector(2))*Qo^2)/(2*D(1)*g*CS_area(1)^2);
H(1,1)=H(1,2)+(f(1)*(x_vector(2)-
x_vector(1))*Qo^2)/(2*D(1)*g*CS_area(1)^2);
%junction loss and other minor losses are neglected

%% Boundary conditions
Cn=zeros(Num_timenode,Num_node);
Cp=zeros(Num_timenode,Num_node);
Cv=zeros(Num_timenode,Num_node);
while i<Num_timenode
    i=i+1;
for i=2:Num_timenode

% FOR UPSTREAM RESERVOIR***
    Cn(i,1)=Q(i-1,2)-(g*CS_area(1))/a(1)*H(i-1,2)-
R(1)*delta_t*Q(i-1,2)*abs(Q(i-1,2));
    Q(i,1)=Ca(1)*H(1,1)+Cn(1);
    H(i,1)=H(1,1);

% Interior nodes
    Cp(i,2)=Q(i-1,1)+((g*CS_area(1))/a(1))*H(i-1,1)-
R(1)*delta_t*Q(i-1,1)*abs(Q(i-1,1));
    Cn(i,2)=Q(i-1,3)-((g*CS_area(1))/a(1))*H(i-1,3)-
R(1)*delta_t*Q(i-1,3)*abs(Q(i-1,3));
```

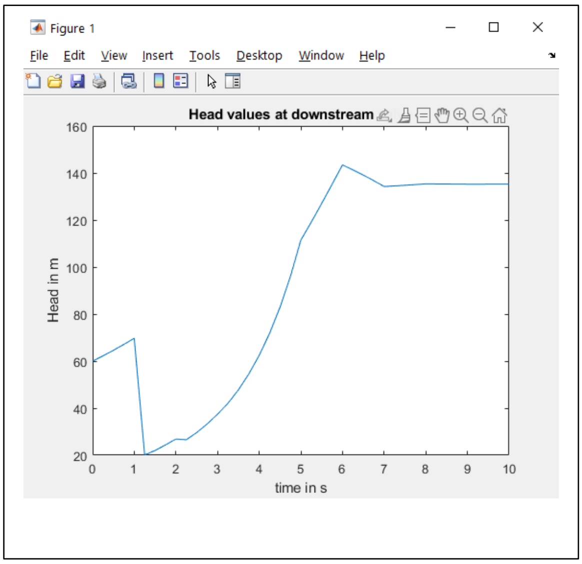
```
     Q(i,2)=0.5*(Cp(i,2)+Cn(i,2));
     H(i,2)=(Q(i,2)-Cn(i,2))/Ca(1);

    Cp(i,5)=Q(i-1,4)+((g*CS_area(2))/a(2))*H(i-1,4)-
R(2)*delta_t*Q(i-1,4)*abs(Q(i-1,4));
    Cn(i,5)=Q(i-1,6)-((g*CS_area(2))/a(2))*H(i-1,6)-
R(2)*delta_t*Q(i-1,6)*abs(Q(i-1,6));
    Q(i,5)=0.5*(Cp(i,5)+Cn(i,5));
    H(i,5)=(Q(i,5)-Cn(i,5))/Ca(2);

% Junction condition
    Cp(i,3)=Q(i-1,2)+((g*CS_area(1))/a(1))*H(i-1,2)-
R(1)*delta_t*Q(i-1,2)*abs(Q(i-1,2));
    Cn(i,4)=Q(i-1,5)-((g*CS_area(2))/a(2))*H(i-1,5)-
R(2)*delta_t*Q(i-1,5)*abs(Q(i-1,5));
    H(i,3)=(Cp(i,3)-Cn(i,4))/(Ca(1)+Ca(2));
    H(i,4)=H(i,3);
    Q(i,3)=Cp(i,3)-Ca(1)*H(i,3);
    Q(i,4)=Q(i,3);
% Valve BC
    Cp(i,6)=Q(i-1,5)+((g*CS_area(2))/a(2))*H(i-1,5)-
R(2)*delta_t*Q(i-1,5)*abs(Q(i-1,5));
    Cv(i,6)=((Tau(i)*Qo)^2)/(Ca(2)*H(1,6));
    Q(i,6)=0.5*(-
Cv(i,6)+sqrt(Cv(i,6)^2+4*Cp(i,6)*Cv(i,6)));
    H(i,6)=(Cp(i,6)-Q(i,6))/Ca(2);
end
end
Q
H
time=0:delta_t:T_last;
H_valve=transpose(H(:,6));
plot(time,H_valve);
title("Head values at downstream valve");
xlabel('time in s');
ylabel('Head in m');
```

Output
Q =

1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
0.2665	1.0000	1.0000	1.0000	1.0000	0.9931
0.2665	0.6350	1.0000	1.0000	0.9931	0.9858
0.2665	0.6350	0.6730	0.6730	0.9859	0.9782
0.2665	0.6729	0.6650	0.6650	0.6606	0.9703
0.2665	0.6649	0.6590	0.6590	0.6518	0.4923
0.2665	0.6591	0.6502	0.6502	0.4916	0.4848
0.2665	0.6502	0.4722	0.4722	0.4841	0.4779
0.2665	0.4726	0.4648	0.4648	0.4586	0.4685
0.2665	0.4652	0.4573	0.4573	0.4493	0.4326
0.2665	0.4574	0.4478	0.4478	0.4315	0.4217
0.2665	0.4478	0.4284	0.4284	0.4203	0.4097
0.2665	0.4285	0.4171	0.4171	0.4067	0.3950
0.2665	0.4172	0.4042	0.4042	0.3919	0.3768
0.2665	0.4042	0.3889	0.3889	0.3744	0.3569
0.2665	0.3889	0.3709	0.3709	0.3541	0.3336
0.2665	0.3709	0.3500	0.3500	0.3302	0.3061
0.2665	0.3501	0.3255	0.3255	0.3021	0.2735
0.2665	0.3255	0.2965	0.2965	0.2689	0.2351
0.2665	0.2965	0.2622	0.2622	0.2296	0.1898
0.2665	0.2623	0.2217	0.2217	0.1832	0.1363
0.2665	0.2218	0.1738	0.1738	0.1285	0.1057
0.2665	0.1739	0.1174	0.1174	0.0964	0.0728
0.2665	0.1174	0.0871	0.0871	0.0617	0.0375
0.2665	0.0871	0.0550	0.0550	0.0283	0
0.2665	0.0550	0.0212	0.0212	-0.0067	0
0.2665	0.0212	-0.0142	-0.0142	-0.0071	0
0.2665	-0.0142	-0.0105	-0.0105	-0.0075	0
0.2665	-0.0105	-0.0067	-0.0067	-0.0034	0
0.2665	-0.0067	-0.0026	-0.0026	0.0008	0
0.2665	-0.0026	0.0017	0.0017	0.0009	0
0.2665	0.0017	0.0013	0.0013	0.0009	0
0.2665	0.0013	0.0008	0.0008	0.0004	0
0.2665	0.0008	0.0003	0.0003	-0.0001	0
0.2665	0.0003	-0.0002	-0.0002	-0.0001	0
0.2665	-0.0002	-0.0002	-0.0002	-0.0001	0
0.2665	-0.0002	-0.0001	-0.0001	-0.0001	0
0.2665	-0.0001	-0.0000	-0.0000	0.0000	0
0.2665	-0.0000	0.0000	0.0000	0.0000	0
0.2665	0.0000	0.0000	0.0000	0.0000	0
0.2665	0.0000	0.0000	0.0000	0.0000	0

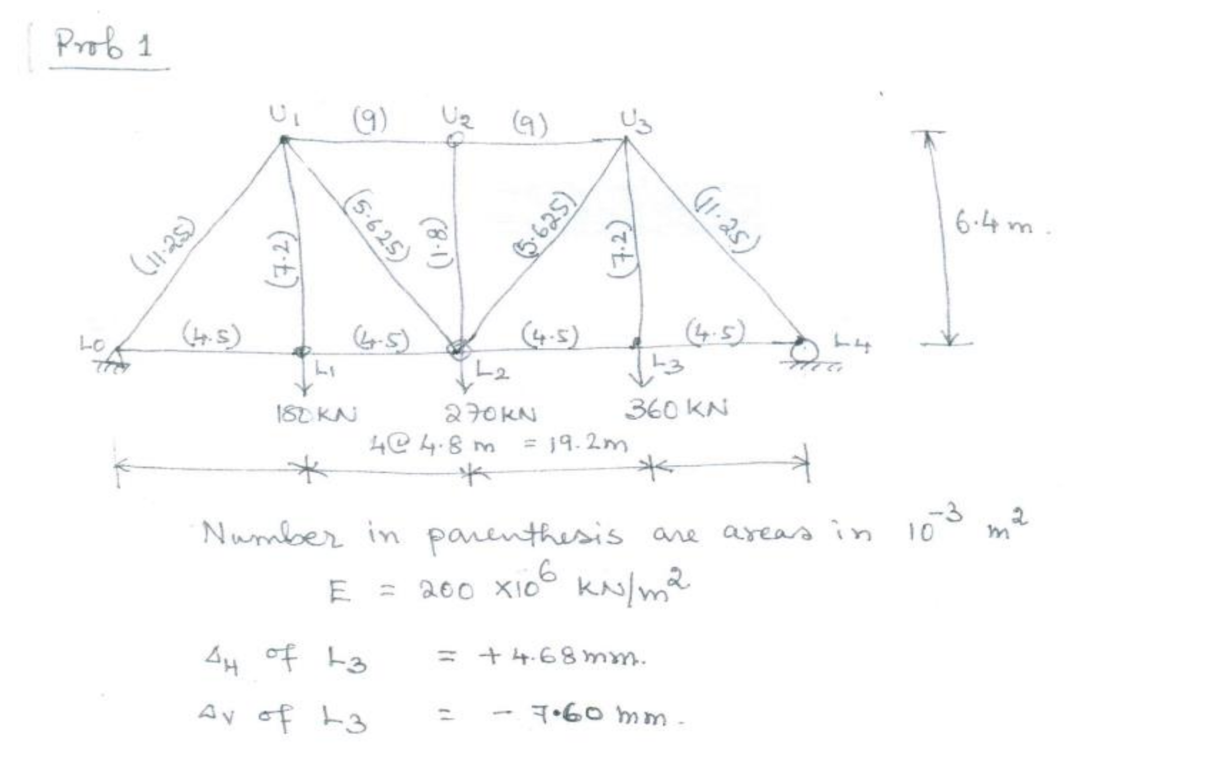
H =

67.6530	66.6955	65.7380	65.7380	62.8690	60.0000
67.6530	66.6955	65.7380	65.7380	62.8690	62.2457
67.6530	-25.9390	65.7380	65.7380	65.0949	64.6074
67.6530	-25.9390	-35.9586	-35.9586	67.4359	67.1039
67.6530	-35.5487	-33.9217	-33.9217	-33.2267	69.7326
67.6530	-33.5169	-32.4674	-32.4674	-30.9089	20.1247
67.6530	-32.0426	-30.2062	-30.2062	20.6086	22.0370
67.6530	-29.7922	14.9842	14.9842	22.4705	24.3631
67.6530	15.2989	16.8715	16.8715	18.7465	26.8713
67.6530	17.1773	18.9457	18.9457	21.2772	26.5761
67.6530	19.1527	21.3759	21.3759	26.7438	29.6404
67.6530	21.5755	26.2962	26.2962	29.7066	33.2856
67.6530	26.4843	29.1721	29.1721	32.8155	37.4377
67.6530	29.3514	32.4720	32.4720	36.8774	42.0600
67.6530	32.6382	36.3626	36.3626	41.6859	47.7715
67.6530	36.5183	40.9530	40.9530	47.2224	54.5087
67.6530	41.0971	46.2560	46.2560	53.7383	62.4495
67.6530	46.3870	52.5004	52.5004	61.4417	71.8264
67.6530	52.6169	59.8707	59.8707	70.5439	82.9223
67.6530	59.9715	68.5805	68.5805	81.3045	96.0215
67.6530	68.6641	78.8811	78.8811	94.0111	111.4456
67.6530	78.9467	91.0524	91.0524	108.9783	119.0880
67.6530	91.0998	105.4041	105.4041	116.1020	127.0262
67.6530	105.4343	113.1116	113.1116	123.4399	135.1668
67.6530	113.1297	121.2771	121.2771	132.1675	143.4334
67.6530	121.2852	129.8617	129.8617	141.2663	141.3371
67.6530	129.8655	138.8333	138.8333	139.0307	139.0994
67.6530	138.8347	137.9146	137.9146	136.6667	136.7244
67.6530	137.9148	136.9328	136.9328	135.6085	134.2341
67.6530	136.9327	135.8904	135.8904	134.5003	134.4926
67.6530	135.8904	134.7974	134.7974	134.7746	134.7665
67.6530	134.7974	134.9109	134.9109	135.0636	135.0565
67.6530	134.9108	135.0311	135.0311	135.1928	135.3607
67.6530	135.0311	135.1583	135.1583	135.3282	135.3291
67.6530	135.1583	135.2919	135.2919	135.2946	135.2956
67.6530	135.2919	135.2780	135.2780	135.2593	135.2602
67.6530	135.2780	135.2633	135.2633	135.2435	135.2230
67.6530	135.2633	135.2477	135.2477	135.2270	135.2269
67.6530	135.2477	135.2314	135.2314	135.2311	135.2310
67.6530	135.2314	135.2331	135.2331	135.2354	135.2353
67.6530	135.2331	135.2349	135.2349	135.2373	135.2398



2.Finite Element Problems

2.1 Truss 1:

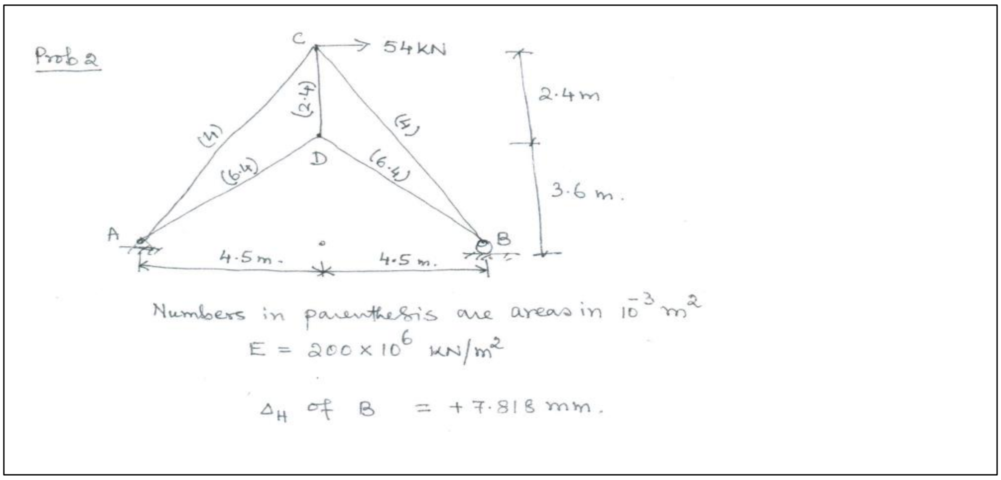



```
clear all;
clc;
E=200e9*ones(13,1);
A=(1/1000)*[4.5;4.5;4.5;4.5;11.25;7.2;5.625;1.8;5.625;7.2;11.25;9;9];
% area of each element
Node=[0 0;4.8 0;9.6 0;14.4 0;19.2 0;14.4 6.4;9.6 6.4;4.8 6.4];
%coordinates of nodes
elcon=[1 2 E(1) A(1);2 3 E(2) A(2); 3 4 E(3) A(3); 4 5 E(4) A(4);1 8 E(5)
A(5);2 8 E(6) A(6); 3 8 E(7) A(7);3 7 E(8) A(8); 3 6 E(9) A(9); 4 6 E(10)
A(10);5 6 E(11) A(11);8 7 E(12) A(12); 7 6 E(13) A(13)];
%element connectivity matrix. nodes connected to each element.
UBC=[1 1 0;1 2 0;5 2 0];
% displacement boundary conditions.[node no    x or y   displacement value]
FBC=[2 2 -180; 3 2 -270; 4 2 -360; 5 1 0];
%force boundary condition
numnode=size(Node,1);
%returns the number of rows in the array or matrix.
% Numnode means total number of nodes=8 . if size(Node,2) was there
% it may give number of columns i.e 2
numel=size(elcon,1) ;
%number of elements
Kg=zeros(2*numnode);
%global stiffness matrix is 2 times number of nodes square matrix.
Fg=zeros(2*numnode,1);
%global force vector.
Ug=zeros(2*numnode,1);
%global displacement vector
```

```
for el=1:numel
    n1=elcon(el,1);
    n2=elcon(el,2);
    %from elcon matrix, two node numbers are being stored from each element
number.
    x1=Node(n1,1);
    y1=Node(n1,2);
    % coordinates of node 1 of an element
    x2=Node(n2,1);
    y2=Node(n2,2);
    % coordinates of node 2 of an element
    theta=atan2(y2-y1,x2-x1);
    %atan2 represents 4 quadrant inverse tangent.
    L=sqrt((x2-x1)^2+(y2-y1)^2);
    C=cos(theta);
    S=sin(theta);
    kel=(A(el,1)*E(el,1)/L)*[C^2 C*S -C^2 -C*S;C*S S^2 -C*S -S^2;-C^2 -C*S C^2
C*S;-C*S -S^2 C*S S^2];
    k1=2*n1-1;
    k2=2*n1;
    k3=2*n2-1;
    k4=2*n2;
    %k1, k2,k3,k4 represents the positions of displacement terms of elemental
    %stiffness matrix in global stiffness matrix
    Kg(k1:k2,k1:k2)=Kg(k1:k2,k1:k2)+kel(1:2,1:2);
    Kg(k1:k2,k3:k4)=Kg(k1:k2,k3:k4)+kel(1:2,3:4);
    Kg(k3:k4,k1:k2)=Kg(k3:k4,k1:k2)+kel(3:4,1:2);
    Kg(k3:k4,k3:k4)=Kg(k3:k4,k3:k4)+kel(3:4,3:4);
end
Kg([1 2 10],:)=[];
Kg(:,[1 2 10])=[];
K=Kg;
%applying boundary conditions
K;
f=[0;-180000;0;-270000;0;-360000;0;0;0;0;0;0;0];
u=K\f;
displacement=[u(5,1) u(6,1)] %in m
```

```
Output:

displacement =


   0.0047   -0.0076
```

2.2 Trusss-2




```
clear all;
clc;
E=10^9*[200; 200; 200;200;200];
A=(1/1000)*[6.4; 6.4; 4; 2.4;4];
% area of each element
Node=[0,0;4.5,3.6;9,0;4.5,6];
%coordinates of nodes
elcon=[1,2;2,3;3 4;2 4;1 4];
%element connectivity matrix. nodes connected to each element.
UBC=[1 1 0;1 2 0; 3 2 0];
% displacement boundary conditions.[node no    x or y   displacement
value]
FBC=[4 1 54000];
%force boundary condition
numnode=size(Node,1);
%returns the number of rows in the array or matrix.
% Numnode means total number of nodes=8 . if size(Node,2) was there
% it may give number of columns i.e 2
numel=size(elcon,1);
%number of elements
Kg=zeros(2*numnode);
%global stiffness matrix is 2 times number of nodes square matrix.
Fg=zeros(2*numnode,1);
%global force vector.
Ug=zeros(2*numnode,1);
%global displacement vector
```

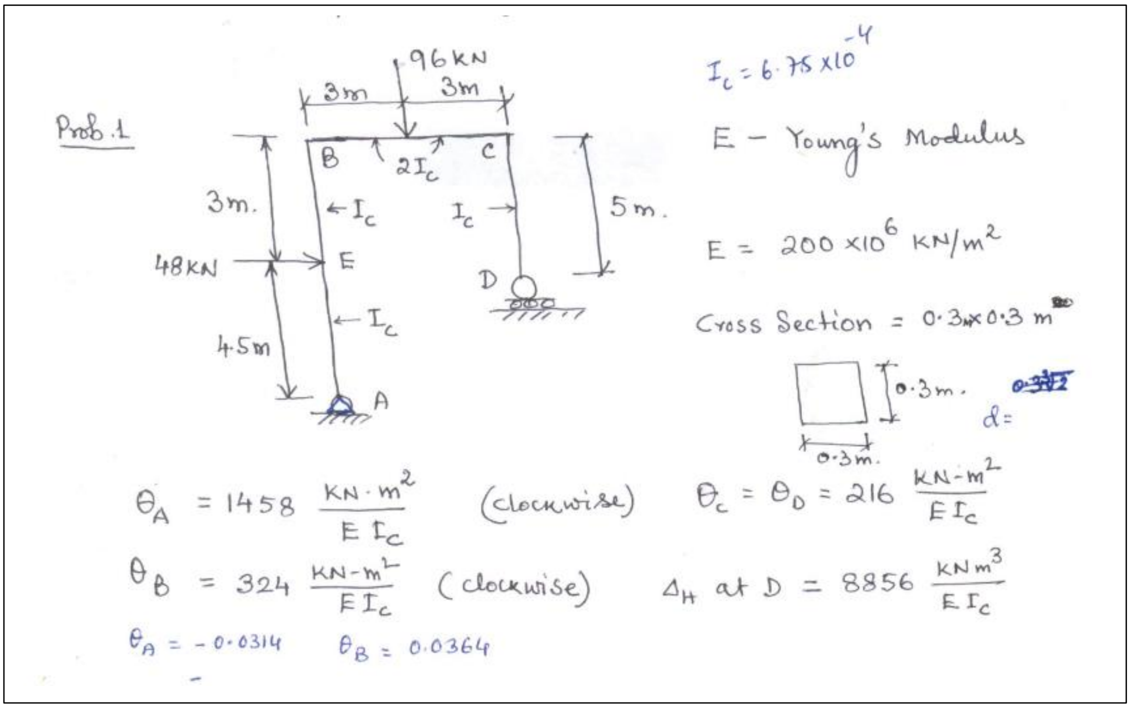
```
for el=1:numel
    n1=elcon(el,1);
    n2=elcon(el,2) ;
    %from elcon matrix, two node numbers are being stored from each element
number.
    x1=Node(n1,1);
    y1=Node(n1,2);
    % coordinates of node 1 of an element
    x2=Node(n2,1);
    y2=Node(n2,2);
    % coordinates of node 2 of an element
    theta=atan2d(y2-y1,x2-x1);
    %atan2 represents 4 quadrant inverse tangent.
    L=sqrt((x2-x1)^2+(y2-y1)^2);
    C=cosd(theta);
    S=sind(theta);
    kel=(A(el,1)*E(el,1)/L)*[C^2 C*S -C^2 -C*S;C*S S^2 -C*S -S^2;-C^2 -C*S C^2
C*S;-C*S -S^2 C*S S^2];
k1=2*n1-1;
k2=2*n1;
k3=2*n2-1;
k4=2*n2;
%k1, k2,k3,k4 represents the positions of displacement terms of elemental
%stiffness matrix in global stiffness matrix
Kg(k1:k2,k1:k2)=Kg(k1:k2,k1:k2)+kel(1:2,1:2);
Kg(k1:k2,k3:k4)=Kg(k1:k2,k3:k4)+kel(1:2,3:4);
Kg(k3:k4,k1:k2)=Kg(k3:k4,k1:k2)+kel(3:4,1:2);
Kg(k3:k4,k3:k4)=Kg(k3:k4,k3:k4)+kel(3:4,3:4);
end
Kg([1 2 6],:)=[];
Kg(:,[1 2 6])=[];
K=Kg;
%applying boundary conditions
K;
f=[0;0;0;54000;0];
u=K\f;
ux_B=u(3) % Horizontal displacement at B in m
```

Output:

ux_B =

0.0078

2.3. Frame




```
clc
clear all
E=200e9;
A=0.09;
Ic=6.75*10^(-4);
I_VEC=[Ic; Ic; 2*Ic; 2*Ic; Ic];% Area moment of inertia
node=[0 0;0 4.5;0,7.5;3,7.5;6,7.5;6,2.5]; % node coordinates
elcon=[1 2;2 3;3 4;4,5; 5,6]; % element connectivity matrix
numel=size(elcon,1); %number of elements
numnode=size(node,1); %number of nodes
k_global=zeros(3*numnode);
for el=1:numel
    n1=elcon(el,1);
    n2=elcon(el,2);
    % n1,n2 represent node numbers for 'el'th element.
    x1=node(n1,1);
    y1=node(n1,2);
    % (x1,y1) coordinates of node 1 of an element
    x2=node(n2,1);
    y2=node(n2,2);
    % coordinates of node 2 of an element
    theta=atan2d(y2-y1,x2-x1);
    %atan2 represents 4 quadrant inverse tangent.
    L=sqrt((x2-x1)^2+(y2-y1)^2); % LENGTH OF THE ELEMENT
    C=cosd(theta);
    S=sind(theta);
    I=I_VEC(el,1); % area moment of inertia for 'el'th element
```

```
T1=(A*E)/L;
    T2=(12*E*I)/L^3;
    T3=(4*E*I)/L;
    T4=(6*E*I)/L^2;
    T5=(2*E*I)/L;
    k_el_lcs=[T1 0 0 -T1 0 0;
        0 T2 T4 0 -T2 T4;
        0 T4 T3 0 -T4 T5;
        -T1 0 0 T1 0 0;
        0 -T2 -T4 0 T2 -T4;
        0 T4 T5 0 -T4 T3]; %elemental stiffness matrix in local
    % co ordinate system
    TRANSFORMATION_MAT=zeros(6);
    TRANSFORMATION_MAT(1:2,1:2)=[C S;-S C];
    TRANSFORMATION_MAT(4:5,4:5)=[C S;-S C];
    TRANSFORMATION_MAT(3,3)=1;
    TRANSFORMATION_MAT(6,6)=1;
    k_el_gcs=TRANSFORMATION_MAT.'*k_el_lcs*TRANSFORMATION_MAT
    k1=3*n1-2;
    k2=3*n1-1;
    k3=3*n1;
    k4=3*n2-2;
    k5=3*n2-1;
    k6=3*n2;
    k_global(k1:k6,k1:k6)=k_global(k1:k6,k1:k6)+k_el_gcs
    %Kg(k1:k2,k3:k4)=Kg(k1:k2,k3:k4)+kel(1:2,3:4)
    %Kg(k3:k4,k1:k2)=Kg(k3:k4,k1:k2)+kel(3:4,1:2)
    %Kg(k3:k4,k3:k4)=Kg(k3:k4,k3:k4)+kel(3:4,3:4)
end
f_global=[0; 0; 0; 48000; 0; 0; 0; 0; 0; 0; -96000; 0; 0; 0; 0; 0; 0; 0];
k_global([1 2 17],:)=[];
k_global(:,[1 2 17])=[];
f_global([1 2 17],:)=[];
u=k_global\f_global
Theta_ABCD=[u(1);u(7);u(13);u(15)] %in radian
ux_D=u(14) %in m
```

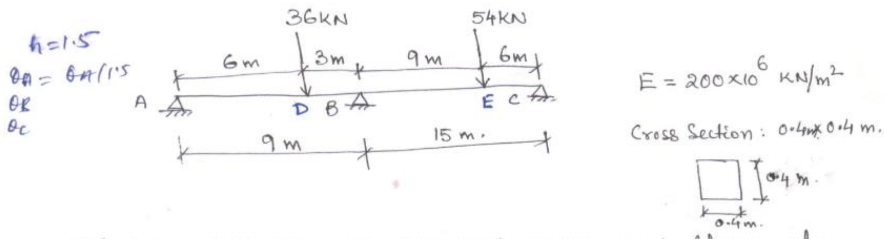
OUTPUT:

```
u =

  -0.0108
   0.0432
  -0.0000
  -0.0072
   0.0576
  -0.0000
  -0.0024
   0.0576
  -0.0034
   0.0002
   0.0576
  -0.0000
   0.0016
   0.0656
   0.0016
```

```
Theta_ABCD =

  -0.0108
  -0.0024
   0.0016
   0.0016
ux_D =

   0.0656
```

2.4 Beam

Prob.2



Calculate deflection at the mid-spans and slopes at the supports. Draw bending moment and shear force diagram for the continuous beam.

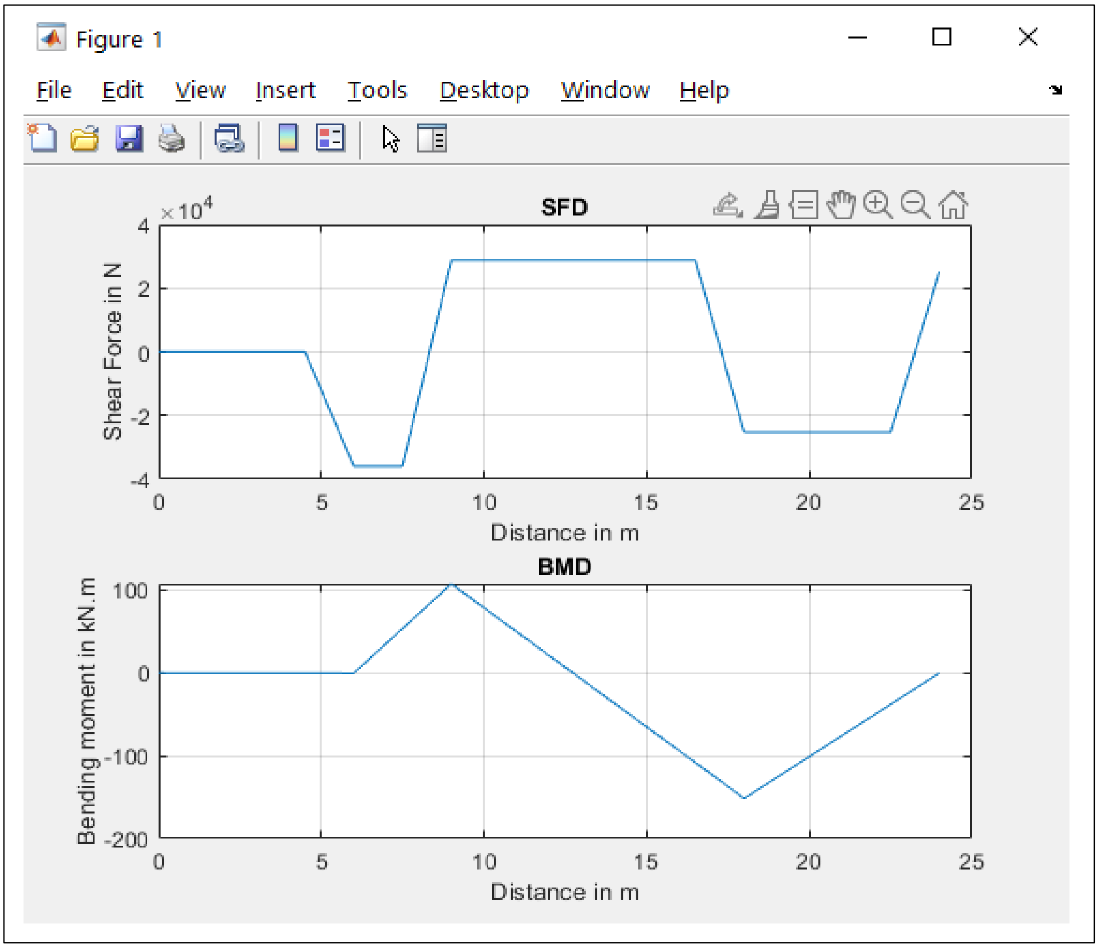
$x = 4.5$ m	0.1845 mm	$\theta_A = 0.00035°$
$x = 16.5$	-4.8558 "	$\theta_B = -0.0287°$
$x = 12$ m	-4.8339 "	$\theta_C = \cancel{0.010}\ 0.1025°$

$V_A = 50$ N
$V_D = 3.595\times10^4$ N
$V_B = 2.897\times10^4$ N
$V_E = 2.523\times10^4$ N
$V_C = 1.1298\times10^{-9}$ N

```
%% Informations given
L=24;
E=200e6;
I=(0.4)^4/12;
F01=36;%Forces are in KN
F02=54;
Node=(0:1.5:L)';% coordinates for nodes
% element connectivity matrix
elcon=zeros(16,2);
for i=1:16
    elcon(i,1)=i;
    elcon(i,2)=i+1;
end
%%
numnode=size(Node,1); %number of nodes
numel=size(elcon,1); %number of elements
Kg=zeros(2*numnode);
Fg=zeros(2*numnode,1);
Ug=zeros(2*numnode,1);
```

```
for el=1:numel
    n1=elcon(el,1); %1st node for an element
    n2=elcon(el,2); %2nd node for an element
    le=L/numel;
    kel=((E*I)/le^3)*[12 6*le -12 6*le; 6*le 4*le^2 -6*le 2*le^2; -12 -6*le 12
-6*le; 6*le 2*le^2 -6*le 4*le^2];
    k1=2*n1-1;
    k2=2*n1;
    k3=2*n2-1;
    k4=2*n2;
    %k1, k2,k3,k4 represents the positions of displacement terms of elemental
    %stiffness matrix in global stiffness matrix
    Kg(k1:k2,k1:k2)=Kg(k1:k2,k1:k2)+kel(1:2,1:2);
    Kg(k1:k2,k3:k4)=Kg(k1:k2,k3:k4)+kel(1:2,3:4);
    Kg(k3:k4,k1:k2)=Kg(k3:k4,k1:k2)+kel(3:4,1:2);
    Kg(k3:k4,k3:k4)=Kg(k3:k4,k3:k4)+kel(3:4,3:4);
end
Kg;
Kg([1 13 33],:)=[];
Kg(:,[1 13 33])=[];
Fg(9,1)=-36;
Fg(25,1)=-54;
Fg([1 13 33],:)=[];
u=Kg\Fg;
U_y=1000*u([6,15,21],:) %Vertical displacements at 4.5m, 12m, 16.5m.
U_theta=(180/pi)*u([1,12,31],:) %Rotation at A,B and C.
%% SFD & BMD using FEM
%Getting total displacement vector
u_total=zeros(2*numnode,1);
u_total(2:12,:)=u(1:11,:);
u_total(14:32)=u(12:30,:);
u_total(34,1)=u(31,1);
%To store SF and BM in "F_EL" for each node
F_EL=[];
for i=1:numel
    u_elemental=[u_total(2*i-1);u_total(2*i);u_total(2*i+1);u_total(2*i+2)];
    f_el=kel*u_elemental;
    F_EL=[F_EL;f_el];
end
for i=1:numel-1 %Because, last two rows for V and M would not come twice.No
need to delete.
    F_EL([2*i+1 2*i+2],:)=[];
end
%To extract SF and BM vector seperately from "F_EL"
BM=[];
SF=[];
for i=1:numnode
    BM=[BM;F_EL(2*i)];
    SF=[SF;F_EL(2*i-1)];
end
%Plotting
subplot(2,1,1)
plot(Node,1000*SF);
xlabel('Distance in m');
ylabel('Shear Force in N');
title('SFD');
grid on;
subplot(2,1,2)
plot(Node,BM);
xlabel('Distance in m');
ylabel('Bending moment in kN.m');
title('BMD');
grid on;
```

```
Output:
U_y =

   0.1845
  -1.8339
  -4.8558


U_theta =

   0.0023
  -0.0192
   0.0683
```



2.4. Beam (Using Different method)

// Error: Level medium ☹

```
clear all;
clc;
E=200e9;
I=0.4^4/12;
L=24;
q=0;
xi=[-sqrt(1/3); sqrt(1/3)];
w=[1; 1];
%% element connectivity matrix
elcon=zeros(16,2);
for i=1:16
    elcon(i,1)=i;
    elcon(i,2)=i+1;
end
numel=size(elcon,1);
le=L/numel;
elcon;
numnode=numel+1;
%% coordinates for nodes
Node_coordinate=zeros(numnode,1);
for i=1:numnode
    Node_coordinate(i,1)=Node_coordinate(i,1)+(i-1)*le;
end
Node_coordinate;
%% Initialization
Kg=zeros(2*numnode);
%Ug=zeros(2*numnode,1);
Fg=zeros(2*numnode,1);
%% Shape Functions
%N1=0.25*(2-3*xi+xi^3)
%N2=(le/8)*(1-xi-xi^2+xi^3)
%N3=0.25*(2+3*xi-xi^3)
%N4=(le/8)*(-1-xi+xi^2+xi^3)
%% 2nd derivative of shape functions for 1st gauss point
ddN1_1=1.5*xi(1);
ddN2_1=(le/8)*(-2+6*xi(1));
ddN3_1=-1.5*xi(1);
ddN4_1=(le/8)*(2+6*xi(1));
ddN_VEC_1=[ddN1_1; ddN2_1; ddN3_1; ddN4_1];
%% 2nd derivative of shape functions for 2nd gauss point
ddN1_2=1.5*xi(2);
ddN2_2=(le/8)*(-2+6*xi(2));
ddN3_2=-1.5*xi(2);
ddN4_2=(le/8)*(2+6*xi(2));
ddN_VEC_2=[ddN1_2; ddN2_2; ddN3_2; ddN4_2];
```

```
%% elemental stiffness matrix
kel=zeros(4);
for i=1:4
    for j=1:4
kel(i,j)=E*I*(2/le)^3*(ddN_VEC_1(i)*ddN_VEC_1(j)*w(1)+ddN_VEC_2(i)*ddN_VEC_2(j)*w(2));
    end
end
kel;
%% for 1st gauss point
N1_1=0.25*(2-3*xi(1)+xi(1)^3);
N2_1=(le/8)*(1-xi(1)-xi(1)^2+xi(1)^3);
N3_1=0.25*(2+3*xi(1)-xi(1)^3);
N4_1=(le/8)*(-1-xi(1)+xi(1)^2+xi(1)^3);
N_VEC_1=[N1_1;N2_1;N3_1;N4_1];
%% for 2nd gauss point
N1_2=0.25*(2-3*xi(2)+xi(2)^3);
N2_2=(le/8)*(1-xi(2)-xi(2)^2+xi(2)^3);
N3_2=0.25*(2+3*xi(2)-xi(2)^3);
N4_2=(le/8)*(-1-xi(2)+xi(2)^2+xi(2)^3);
N_VEC_2=[N1_2;N2_2;N3_2;N4_2];
%% elemental force vector for distributed load
fel=zeros(4,1);
for i=1:4
fel(i)=(le/2)*q*(N_VEC_1(i)*w(1)+N_VEC_2(i)*w(2));
end
for el=1:numel % el=element number, numel=total no of elements
    n1=elcon(el,1);
    n2=elcon(el,2);
k1=2*n1-1;
k2=2*n1;
k3=2*n2-1;
k4=2*n2;
%k1, k2,k3,k4 represents the positions of displacement terms of elemental
%stiffness matrix in global stiffness matrix
Kg(k1:k4,k1:k4)=Kg(k1:k4,k1:k4)+kel(1:4,1:4);
Fg(k1:k4,1)=Fg(k1:k4,1)+fel(1:4,1);
end
Fg(5,1)=-36000;
Fg(13,1)=-54000;
Kg([1,13,33],:)=[];
Kg(:,[1,13,33])=[];
Fg([1,13,33],:)=[];
Kg;
Fg;
Ug=Kg\Fg;
U_y=Ug([6,15,21],:) %Vertical displacements at 4.5m, 12m, 16.5m in m.
U_theta=(180/pi)*Ug([1,12,31],:) %Rotation at A,B and C IN DEGREES.
```

Output: *//(incorrect)!*

```
U_y =

  1.0e-03 *

 -0.8780
  0.4556
  0.5933
```

```
U_theta =


 -0.0205

  0.0121

 -0.0060
```

3.COMPUTATIONAL HYDRAULICS

(1) Gauss Elimination

Example

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 \\ 0 & 1 & 3 & -1 & 0 \\ 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{Bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \\ \phi_5 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 12 \\ 11 \\ 28 \\ 9 \end{Bmatrix}$$

Solution:

$$\begin{Bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \\ \phi_5 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 3 \\ 5 \\ 7 \\ 9 \end{Bmatrix}$$

```
clc
clear
function phi=gausselim(A, r)
//Linear Equation: A*phi=r
[nr_A,nc_A]=size(A)//size of A
[nr_r,nc_r]=size(r)//size of r
//" [nr_A,nc_A]=size(A) " retrieves the number of rows and columns of a matrix A and assigns them to the variables "nr_A" and "nc_A", respectively.

if nc_A<>nr_A then
   error('A is not a square matrix')
   abort;
 //if nc_A is not equal to nr_A , it means that matrix A is not square
end
if nc_A<>nr_r then
   error('Not compatible matrices')
   abort;
end
n=nc_A
```

```
//Forward Elimination
for k=1:1:n-1
  for i=k+1:1:n
   gam=A(i,k)/A(k,k)
   for j=k+1:n
      A(i,j)=A(i,j)-gam*A(k,j)
   end
   r(i)=r(i)-gam*r(k)
end
end
//Backward Substitution
phi(n)=r(n)/A(n,n)
for i=n-1:-1:1
   sumj=r(i)
   for j=i+1:n
      sumj=sumj-A(i,j)*phi(j)
end
phi(i)=sumj/A(i,i)
end
endfunction
A=[1 0 0 0 0
 1 2 1 0 0
 0 1 3 -1 0
 0 0 1 2 1
 0 0 0 0 1];
 r=[1
 12
 11
 28
 9];
 phi=gausselim(A,r)
disp(phi)
```

```
Output:
  1.
  3.
  5.
  7.0000000
  9.
```

2. LU Decomposition

Example

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 \\ 0 & 1 & 3 & -1 & 0 \\ 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{Bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \\ \phi_5 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 12 \\ 11 \\ 28 \\ 9 \end{Bmatrix}$$

Solution:

$$\begin{Bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \\ \phi_5 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 3 \\ 5 \\ 7 \\ 9 \end{Bmatrix}$$

```
clc
clear
function phi=ludcomp(A, r)
//Linear Equation:A*phi=r

[nr_A,nc_A]=size(A)//size of A
[nr_r,nc_r]=size (r)//size of r

if nc_A<>nr_A then
   error('A is not a square matrix')
   abort;
end
if nc_A<>nr_r then
   error('Not compatible matrices')
   abort;
end
n=nc_A
```

```
//Decomposition
for k=1:1:n-1
   for i=k+1:1:n
      gam=A(i,k)/A(k,k)
      A(i,k)=gam
      for j=k+1:n
         A(i,j)=A(i,j)-gam*A(k,j)
      end
   end
end
//Forward substitution
psi(1)=r(1)
for i=2:1:n
   sumj=r(i)
   for j=1:i-1
      sumj=sumj-A(i,j)*psi(j)
   end
   psi(i)=sumj
end
//Backward substitution
phi(n)=psi(n)/A(n,n)
for i=n-1:-1:1
   sumj=psi(i)
   for j=i+1:n
      sumj=sumj-A(i,j)*phi(j)
   end
   phi(i)=sumj/A(i,i)
end
endfunction
A=[1 0 0 0 0;1 2 1 0 0;0 1 3 -1 0;0 0 1 2 1;0 0 0 0 1];
 r=[1
 12
 11
 28
 9];
phi=ludcomp(A,r)
```

```
Output:

phi

 phi  =


   1.

   3.

   5.

   7.0000000

   9.
```

3. Tridiagonal matrix method

Example

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 \\ 0 & 1 & 3 & -1 & 0 \\ 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{Bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \\ \phi_5 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 12 \\ 11 \\ 28 \\ 9 \end{Bmatrix}$$

Solution:

$$\begin{Bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \\ \phi_5 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 3 \\ 5 \\ 7 \\ 9 \end{Bmatrix}$$

```
clc
clear
function phi=tdmasolv(b, d, a, r)

//n:number of rows
n=length(d);

a(1)=a(1)/d(1);
r(1)=r(1)/d(1);
//forward elimination
for i=2:n-1
   fact=d(i)-b(i)*a(i-1);
   a(i)=a(i)/fact;
   r(i)=(r(i)-b(i)*r(i-1))/fact;
end

r(n)=(r(n)-b(n)*r(n-1))/(d(n)-b(n)*a(n-1));

//Backward substitution
phi(n)=r(n);
for i=n-1:-1:1
   phi(i)=r(i)-a(i)*phi(i+1);
end
endfunction
d=[1 2 3 2 1];
b=[0 1 1 1 0];
a=[0 1 -1 1 0];
r=[1 12 11 28 9];
phi=tdmasolv(b,d,a,r)
disp(phi)
```

Output:

```
1.
  3.
  5.
  7.
  9.
```

4. Jacobi's Method:

Example

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 \\ 0 & 1 & 3 & -1 & 0 \\ 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{Bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \\ \phi_5 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 12 \\ 11 \\ 28 \\ 9 \end{Bmatrix}$$

Solution:

$$\begin{Bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \\ \phi_5 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 3 \\ 5 \\ 7 \\ 9 \end{Bmatrix}$$

```
clc
clear
function [count, rmse, phi]=jacobi(A, r, phio,
eps_max)
//Linear Equation: A*phi=r

[nr_A,nc_A]=size(A)//size of A
[nr_r,nc_r]=size (r)//size of r

if nc_A<>nr_A then
   error('A is not a square matrix')
   abort;
end
if nc_A<>nr_r then
   error('Not compatible matrices')
   abort;
end
n=nc_A

count=0
rmse=1
```

```
while rmse>eps_max
   rmse=0
   for i=1:n
      res(i)=r(i)
      for j=1:n
         res(i)=res(i)-A(i,j)*phio(j)
      end
      phi(i)=phio(i)+res(i)/A(i,i)
      rmse=rmse+(res(i)/A(i,i)).^2
   end
   phio=phi
   rmse=sqrt(rmse/n)
   count=count+1;
end
endfunction

A=[1 0 0 0 0
1 2 1 0 0
0 1 3 -1 0
0 0 1 2 1
0 0 0 0 1];
r=[1
12
11
28
9];
phio=[0
0
0
0
0];
eps_max=1e-6
[count,rmse,phi]=jacobi(A,r,phio,eps_max)
disp(phi)
```

Output:

1.

3.

5.

7.

9.

```
//Explanation of the loop
//phio = [0; 0; 0]
//res1 = r(1) - A(1,1)*phio(1) - A(1,2)*phio(2) -
A(1,3)*phio(3)
//    = 1 - 2*0 - 1*0 - 1*0
//    = 1
//phi(1) = phio(1) + res1/A(1,1)
//       = 0 + 1/2
//       = 0.5
//
//res2 = r(2) - A(2,1)*phio(1) - A(2,2)*phio(2) -
A(2,3)*phio(3)
//    = 1 - 1*0 - 2*0 - 1*0
//    = 1
//phi(2) = phio(2) + res2/A(2,2)
//       = 0 + 1/2
//       = 0.5
//
//res3 = r(3) - A(3,1)*phio(1) - A(3,2)*phio(2) -
A(3,3)*phio(3)
//    = 1 - 1*0 - 1*0 - 3*0
//    = 1
//phi(3) = phio(3) + res3/A(3,3)
//       = 0 + 1/3
//       = 0.333
//
//RMSE = (res1/A(1,1))^2 + (res2/A(2,2))^2 +
(res3/A(3,3))^2
//     = (1/2)^2 + (1/2)^2 + (1/3)^2
//     = 7/36
```

5. Gauss-Seidal Method

Example

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 \\ 0 & 1 & 3 & -1 & 0 \\ 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{Bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \\ \phi_5 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 12 \\ 11 \\ 28 \\ 9 \end{Bmatrix}$$

Solution:

$$\begin{Bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \\ \phi_5 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 3 \\ 5 \\ 7 \\ 9 \end{Bmatrix}$$

```
clc
clear
function [count, rmse, phi]=gseidal(A, r, phio, eps_max,
omega)
   //Linear Equation:A*phi=r

[nr_A,nc_A]=size(A)//size of A
[nr_r,nc_r]=size (r)//size of r

if nc_A<>nr_A then
   error('A is not a square matrix')
   abort;
end
if nc_A<>nr_r then
   error('Not compatible matrices')
   abort;
end
n=nc_A

count=0
rmse=1
phi=phio
```

```
while rmse>eps_max
   rmse=0
   for i=1:1:n
      resi=r(i)
      for j=1:n
         resi=resi-A(i,j)*phi(j)
      end
      phi(i)=phi(i)+omega*resi/A(i,i)
      rmse=rmse+(omega*resi/A(i,i)).^2
   end
   rmse=sqrt(rmse/n)
   count=count+1;
end
endfunction
A=[1 0 0 0 0
 1 2 1 0 0
 0 1 3 -1 0
 0 0 1 2 1
 0 0 0 0 1];
 r=[1
 12
 11
 28
 9];

 phio=[0
0
0
0
0];
eps_max=1e-6;
omega=1
[count,rmse,phi]=gseidal(A,r,phio,eps_max,omega)
disp("solution",phi)
disp("rmse error",rmse)
disp("count",count)
```

```
Output:
"solution="

  1.
  3.
  5.
  7.
  9.

 "rmse error="

  0.

 "count="

  5.
```

6. Newton-Raphson Method

Example

$$\begin{pmatrix} \phi_1 & \phi_1 & 0 & 0 & 0 \\ 1 & \phi_2^2 & -\phi_2 & 0 & 0 \\ 0 & -\phi_2 & \phi_3 & \phi_3 & 0 \\ 0 & 0 & -\phi_3 & \phi_4 & \phi_5 \\ 0 & 0 & 0 & 1 & \phi_5 \end{pmatrix} \begin{Bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \\ \phi_5 \end{Bmatrix} = \begin{Bmatrix} 3 \\ 3 \\ 17 \\ 27 \\ 29 \end{Bmatrix}$$

```
clc
clear
function phi=ludcomp(A, r)
//Linear Equation:A*phi=r

[nr_A,nc_A]=size(A)//size of A
[nr_r,nc_r]=size (r)//size of r

if nc_A<>nr_A then
   error('A is not a square matrix')
   abort;
end
if nc_A<>nr_r then
   error('Not compatible matrices')
   abort;
end
n=nc_A
//Decomposition
for k=1:1:n-1
   for i=k+1:1:n
      gam=A(i,k)/A(k,k)
      A(i,k)=gam
      for j=k+1:n
         A(i,j)=A(i,j)-gam*A(k,j)
      end
   end
end
```

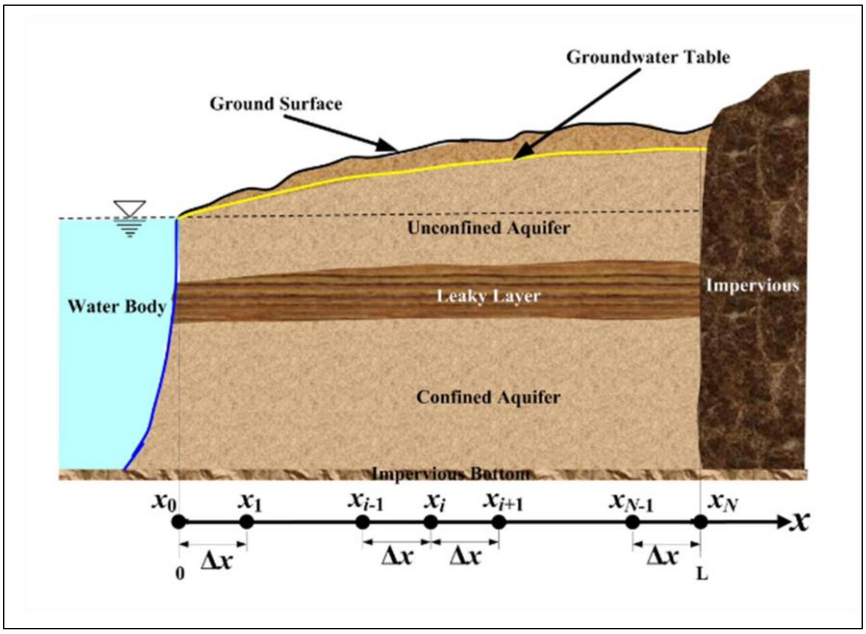
```
//Forward substitution
psi(1)=r(1)
for i=2:1:n
   sumj=r(i)
   for j=1:i-1
      sumj=sumj-A(i,j)*psi(j)
   end
   psi(i)=sumj
end

//Backward substitution
phi(n)=psi(n)/A(n,n)
for i=n-1:-1:1
   sumj=psi(i)
   for j=i+1:n
      sumj=sumj-A(i,j)*phi(j)
   end
   phi(i)=sumj/A(i,i)
end
endfunction
//.....
function FV=Fun(phi)
   FV(1)=phi(1)^2+phi(1)*phi(2)-3
   FV(2)=phi(1)+phi(2)^3-phi(2)*phi(3)-3
   FV(3)=-phi(2)^2+phi(3)^2+phi(3)*phi(4)-17
   FV(4)=-phi(3)^2+phi(4)^2+phi(4)*phi(5)-27
   FV(5)=phi(4)+phi(5)^2-29
endfunction
function JV=JM(phi)
   JV=zeros(length(phi),length(phi))
   //1
   JV(1,1)=2*phi(1)+phi(2)
   JV(1,2)=phi(1)
   //2
   JV(2,1)=1
   JV(2,2)=3*phi(2)^2-phi(3)
   JV(2,3)=-phi(3)
```

```
//3
JV(3,2)=-2*phi(2)
JV(3,3)=2*phi(3)+phi(4)
JV(3,4)=phi(3)
//4
JV(4,3)=-2*phi(3)
JV(4,4)=2*phi(4)+phi(5)
JV(4,5)=phi(4)
//5
JV(5,4)=1
JV(5,5)=2*phi(5)
endfunction
function [count, rmse, phi]=newtonrn(phio, n, eps_max)
count=0
rmse=1
while rmse>eps_max
   rmse=0
   JV=JM(phio)
   FV=Fun(phio)
   dphi=ludcomp(JV,-FV)
   for i=1:n
   phi(i)=phio(i)+dphi(i)
   rmse=rmse+(dphi(i)).^2
end
phio=phi
rmse=sqrt(rmse/n)
count=count+1
end
endfunction
```

```
//
phio1=[1
1
1
1
1];
phio2=[0
0
0
0
0];
phio3=[10
1
100
6
11];
eps_max=1e-6;
n=5
[count,rmse,phi]=newtonrn(phio3,n,eps_max)
FV=Fun(phi)
disp(phi)
```

```
Output:
1.0000000
  2.0000000
  3.0000000
  4.0000000
  5.0000000
```

(7) GW_1D_GAUSS_ELIMINATION



Mathematical Conceptualization

The differential equation describing the head distribution in the aquifer is given as ,

$$\frac{d^2h}{dx^2} = \frac{C_{\text{conf}}}{T}(h - h_{wt}) \tag{1}$$

where,
h = head,
T = aquifer transmissivity,
C_{conf} = hydraulic conductivity/thickness of confining layer,
h_{wt} = overlying water table elevation $(c_0 + c_1x + c_2x^2)$.

Boundary Conditions

- Left Boundary is specified head/ Dirichlet boundary: $h(x = 0) = h_s$
- Right Boundary is impervious/ no-flow/ Neumann Boundary: $\frac{dh}{dx}\Big|_L = 0$

Mathematical Conceptualizatio

Data Values

$C_{\text{conf}} = 10^{-11}$
$T = 2 \times 10^{-5}$
$c_0 = 90$
$c_1 = 0.06$
$c_2 = -0.00003$
$h_s = 90$
$L = 1000$

```
//Groundwater 1d using Gauss elimination method
clc
clear
function phi=gausselim(A, r)
//Linear Equation: A*phi=r
[nr_A,nc_A]=size(A)//size of A
[nr_r,nc_r]=size(r)//size of r

if nc_A<>nr_A then
   error('A is not a square matrix')
   abort;
end
if nc_A<>nr_r then
   error('Not compatible matrices')
   abort;
end
n=nc_A
//Forward Elimination
for k=1:1:n-1
  for i=k+1:1:n
   gam=A(i,k)/A(k,k)
   for j=k+1:n
      A(i,j)=A(i,j)-gam*A(k,j)
   end
   r(i)=r(i)-gam*r(k)
end
end
//Backward Substitution
phi(n)=r(n)/A(n,n)
for i=n-1:-1:1
   sumj=r(i)
   for j=i+1:n
      sumj=sumj-A(i,j)*phi(j)
end
phi(i)=sumj/A(i,i)
end
endfunction
```

```
//....Problem dependent parameters
nnode=21; //number of nodes
xl=0;//coordinate of left end boundary xL
xr=1000;
cconf=1e-11;
T=2e-5;
hs=90;
c0=90;
c1=0.06;
c2=-0.00003;
//....
x=linspace(xl,xr,nnode);//In between left and right boundary, nnode numbers of x coordinates has been generated.
delx=x(2)-x(1);//gird size
//.....Initialization of matrices...
h=zeros(nnode,1);
A=zeros(nnode,nnode);
r=zeros(nnode,1);
//Left boundary
A(1,1)=1.0;
r(1)=hs;
//Interior nodes
for i=2:nnode-1
   A(i,i-1)=1.0/(delx^2);
   A(i,i)=-((cconf/T)+2.0/(delx^2));
   A(i,i+1)=1.0/(delx^2);
   r(i)=-(cconf/T)*(c0+c1*x(i)+c2*x(i)^2);
end
//Right boundary
//Two point
A(nnode,nnode)=1/delx;
A(nnode,nnode-1)=-1/delx;
r(nnode)=0;

disp(A)
disp(r)
h=gausselim(A,r)
plot(x,h','-r')
disp(max(h))
```

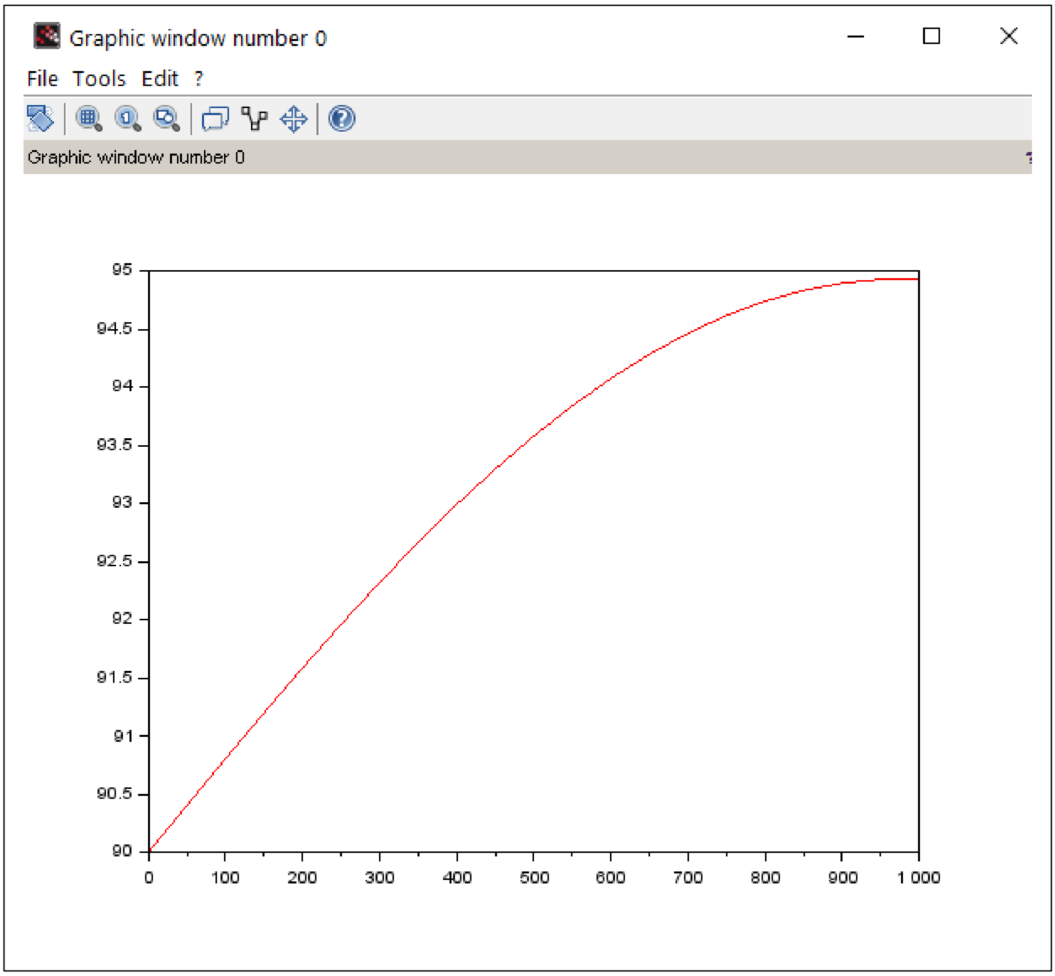
Output: //A

```
column 1 to 10
  1.      0.         0.         0.         0.         0.         0.         0.         0.         0.
  0.0004  -0.0008005  0.0004   0.         0.         0.         0.         0.         0.         0.
  0.      0.0004     -0.0008005  0.0004   0.         0.         0.         0.         0.         0.
  0.      0.         0.0004     -0.0008005  0.0004   0.         0.         0.         0.         0.
  0.      0.         0.         0.0004     -0.0008005  0.0004   0.         0.         0.         0.
  0.      0.         0.         0.         0.0004     -0.0008005  0.0004   0.         0.         0.
  0.      0.         0.         0.         0.         0.0004     -0.0008005  0.0004   0.         0.
  0.      0.         0.         0.         0.         0.         0.0004     -0.0008005  0.0004   0.
  0.      0.         0.         0.         0.         0.         0.         0.0004     -0.0008005  0.0004
  0.      0.         0.         0.         0.         0.         0.         0.         0.0004     -0.0008005
  0.      0.         0.         0.         0.         0.         0.         0.         0.         0.0004
  0.      0.         0.         0.         0.         0.         0.         0.         0.         0.
  0.      0.         0.         0.         0.         0.         0.         0.         0.         0.
  0.      0.         0.         0.         0.         0.         0.         0.         0.         0.
  0.      0.         0.         0.         0.         0.         0.         0.         0.         0.
  0.      0.         0.         0.         0.         0.         0.         0.         0.         0.
  0.      0.         0.         0.         0.         0.         0.         0.         0.         0.
  0.      0.         0.         0.         0.         0.         0.         0.         0.         0.
  0.      0.         0.         0.         0.         0.         0.         0.         0.         0.
  0.      0.         0.         0.         0.         0.         0.         0.         0.         0.
  0.      0.         0.         0.         0.         0.         0.         0.         0.         0.
     column 11 to 20
  0.        0.         0.         0.         0.         0.         0.         0.         0.         0.
  0.        0.         0.         0.         0.         0.         0.         0.         0.         0.
  0.        0.         0.         0.         0.         0.         0.         0.         0.         0.
  0.        0.         0.         0.         0.         0.         0.         0.         0.         0.
  0.        0.         0.         0.         0.         0.         0.         0.         0.         0.
  0.        0.         0.         0.         0.         0.         0.         0.         0.         0.
  0.        0.         0.         0.         0.         0.         0.         0.         0.         0.
  0.        0.         0.         0.         0.         0.         0.         0.         0.         0.
  0.        0.         0.         0.         0.         0.         0.         0.         0.         0.
  0.0004     0.         0.         0.         0.         0.         0.         0.         0.         0.
 -0.0008005  0.0004    0.         0.         0.         0.         0.         0.         0.         0.
  0.0004    -0.0008005  0.0004    0.         0.         0.         0.         0.         0.         0.
  0.        0.0004    -0.0008005  0.0004    0.         0.         0.         0.         0.         0.
  0.        0.         0.0004    -0.0008005  0.0004    0.         0.         0.         0.         0.
  0.        0.         0.         0.0004    -0.0008005  0.0004    0.         0.         0.         0.
  0.        0.         0.         0.         0.0004    -0.0008005  0.0004    0.         0.         0.
  0.        0.         0.         0.         0.         0.0004    -0.0008005  0.0004    0.         0.
  0.        0.         0.         0.         0.         0.         0.0004    -0.0008005  0.0004    0.
  0.        0.         0.         0.         0.         0.         0.         0.0004    -0.0008005  0.0004
  0.        0.         0.         0.         0.         0.         0.         0.         0.0004    -0.0008005
  0.        0.         0.         0.         0.         0.         0.         0.         0.        -0.02
     column 21
  0.
  0.
```

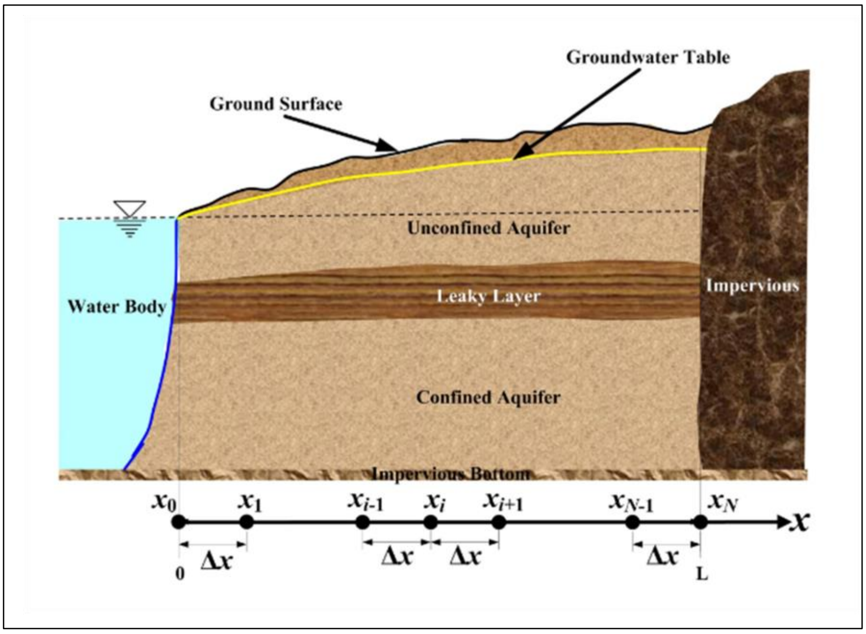
```
0.
0.
0.
0.
0.
0.
0.
0.
0.
0.
0.
0.
0.
0.
0.
0.
0.
0.0004
0.02
```

```
//r
90.
  -0.0000465
  -0.0000478
  -0.0000492
  -0.0000504
  -0.0000516
  -0.0000526
  -0.0000537
  -0.0000546
  -0.0000555
  -0.0000562
  -0.0000570
  -0.0000576
  -0.0000582
  -0.0000586
  -0.0000591
  -0.0000594

//max(h)

 94.925688
```

```
h
 h  =
   90.
   90.404091
   90.805031
   91.199852
   91.585767
   91.960164
   92.320605
   92.664822
   92.990714
   93.296344
   93.579938
   93.839882
   94.074719
   94.283150
   94.464029
   94.616363
   94.739311
   94.832183
   94.894439
   94.925688
   94.925688
```



(8) GW_1D_TDMA



Mathematical Conceptualization

The differential equation describing the head distribution in the aquifer is given as ,

$$\frac{d^2h}{dx^2} = \frac{C_{\text{conf}}}{T}(h - h_{wt}) \tag{1}$$

where,
h = head,
T = aquifer transmissivity,
C_{conf} = hydraulic conductivity/thickness of confining layer,
h_{wt} = overlying water table elevation $(c_0 + c_1x + c_2x^2)$.

Boundary Conditions

- Left Boundary is specified head/ Dirichlet boundary: $h(x = 0) = h_s$
- Right Boundary is impervious/ no-flow/ Neumann Boundary: $\frac{dh}{dx}\big|_L = 0$

Mathematical Conceptualizatio

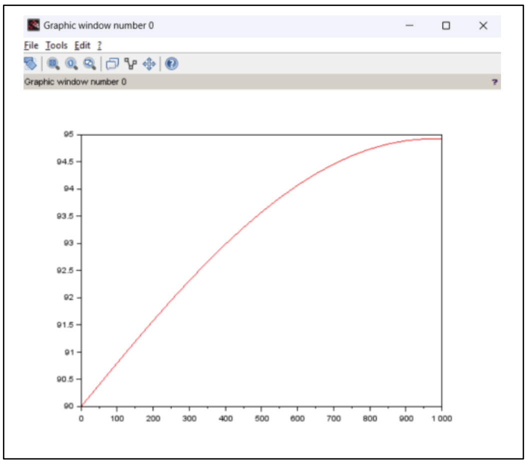
Data Values

$C_{\text{conf}} = 10^{-11}$
$T = 2 \times 10^{-5}$
$c_0 = 90$
$c_1 = 0.06$
$c_2 = -0.00003$
$h_s = 90$
$L = 1000$

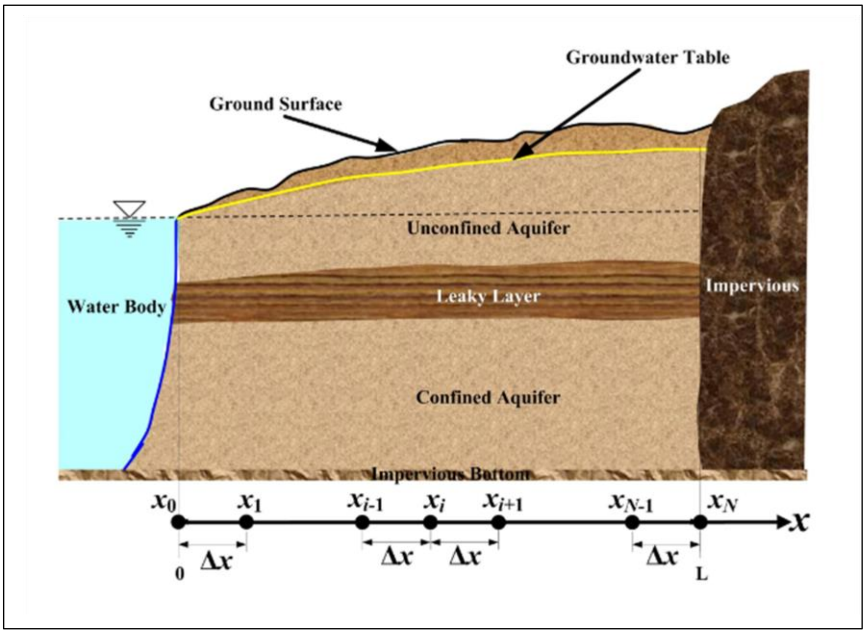
```
//Groundwater 1D using Tridiagonal matrix method
clc
clear
function phi=tdmasolv(b, d, a, r)

//n:number of rows
n=length(d);

a(1)=a(1)/d(1);
r(1)=r(1)/d(1);
//forward elimination
for i=2:n-1
   fact=d(i)-b(i)*a(i-1);
   a(i)=a(i)/fact;
   r(i)=(r(i)-b(i)*r(i-1))/fact;
end

r(n)=(r(n)-b(n)*r(n-1))/(d(n)-b(n)*a(n-1));

//Backward substitution
phi(n)=r(n);
for i=n-1:-1:1
   phi(i)=r(i)-a(i)*phi(i+1);
end
endfunction
//....Problem dependent parameters
nnode=21; //number of nodes
xl=0;//coordinate of left end boundary xL
xr=1000;
cconf=1e-11;
T=2e-5;
hs=90;
c0=90;
c1=0.06;
c2=-0.00003;
```

```
//....
x=linspace(xl,xr,nnode);//In between left and right boundary, nnode numbers of x coordinates has been generated.
delx=x(2)-x(1);//gird size
//.....Initialization of matrices...
h=zeros(nnode,1);
a=zeros(nnode,1);
d=zeros(nnode,1);
b=zeros(nnode,1);
r=zeros(nnode,1);
//Left boundary
d(1,1)=1.0;
r(1)=hs;
//Interior nodes
for i=2:nnode-1
   b(i,1)=1.0/(delx^2);
   d(i,1)=-((cconf/T)+2.0/(delx^2));
   a(i,1)=1.0/(delx^2);
   r(i)=-(cconf/T)*(c0+c1*x(i)+c2*x(i)^2);
end
//Right boundary
//Two point
d(nnode,1)=1/delx;
b(nnode,1)=-1/delx;
r(nnode)=0;

h=tdmasolv(b,d,a,r)
plot(x,h','-r')
disp("Head at nodes",h)
disp("maximum head",max(h))
```

```
Output:
"Head at nodes"

  90.
  90.404091
  90.805031
  91.199852
  91.585767
  91.960164
  92.320605
  92.664822
  92.990714
  93.296344
  93.579938
  93.839882
  94.074719
  94.283150
  94.464029
  94.616363
  94.739311
  94.832183
  94.894439
  94.925688
  94.925688

 "maximum head"

  94.925688
```



(9) GW_1D_Gauss_Seidel:



Mathematical Conceptualization

The differential equation describing the head distribution in the aquifer is given as ,

$$\frac{d^2h}{dx^2} = \frac{C_{\text{conf}}}{T}(h - h_{wt}) \tag{1}$$

where,
h = head,
T = aquifer transmissivity,
C_{conf} = hydraulic conductivity/thickness of confining layer,
h_{wt} = overlying water table elevation $(c_0 + c_1 x + c_2 x^2)$.

Boundary Conditions

- Left Boundary is specified head/ Dirichlet boundary: $h(x = 0) = h_s$
- Right Boundary is impervious/ no-flow/ Neumann Boundary: $\frac{dh}{dx}\Big|_L = 0$

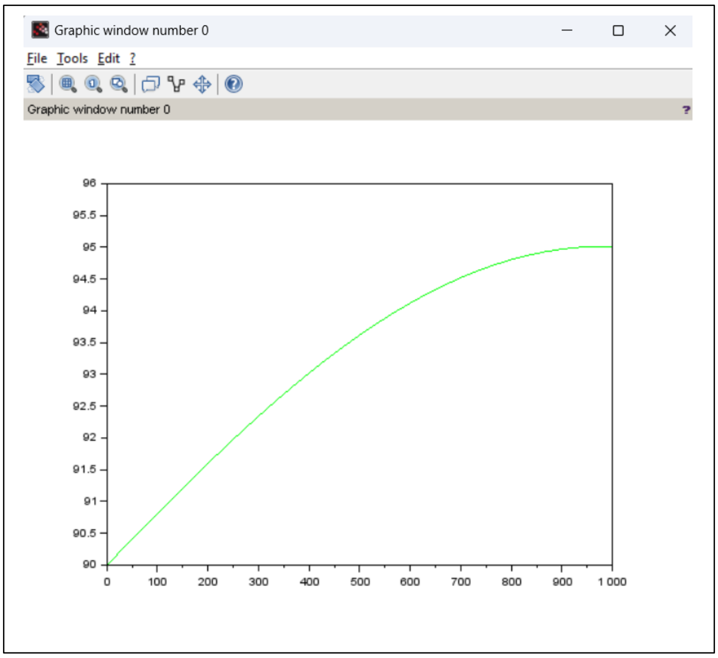
Mathematical Conceptualizatio
Data Values

$C_{\text{conf}} = 10^{-11}$
$T = 2 \times 10^{-5}$
$c_0 = 90$
$c_1 = 0.06$
$c_2 = -0.00003$
$h_s = 90$
$L = 1000$

```
clc
clear
function [count, rmse, phi]=gseidel(A, r, phio, eps_max, omega)
   //Linear Equation:A*phi=r

[nr_A,nc_A]=size(A)//size of A
[nr_r,nc_r]=size (r)//size of r

if nc_A<>nr_A then
   error('A is not a square matrix')
   abort;
end
if nc_A<>nr_r then
   error('Not compatible matrices')
   abort;
end
n=nc_A

count=0
rmse=1
phi=phio
while rmse>eps_max
   rmse=0
   for i=1:1:n
      resi=r(i)
      for j=1:n
         resi=resi-A(i,j)*phi(j)
      end
      phi(i)=phi(i)+omega*resi/A(i,i)
      rmse=rmse+(omega*resi/A(i,i)).^2
   end
   rmse=sqrt(rmse/n)
   count=count+1;
   disp("Count rmse",[count rmse])

end
endfunction
```

```
//....Problem dependent parameters
nnode=31; //number of nodes
xl=0;//coordinate of left end boundary xL
xr=1000;
cconf=1e-11;
T=2e-5;
hs=90;
c0=90;
c1=0.06;
c2=-0.00003;
x=linspace(xl,xr,nnode);//In between left and right boundary, nnode
numbers of x coordinates has been generated.
delx=x(2)-x(1);//gird size
//.....Initialization of matrices...
h=zeros(nnode,1);
A=zeros(nnode,nnode);
r=zeros(nnode,1);
//Left boundary
A(1,1)=1.0;
r(1)=hs;
//Interior nodes
for i=2:nnode-1
   A(i,i-1)=(1.0/(delx^2))*delx^2;
   A(i,i)=-((cconf/T)+2.0/(delx^2))*delx^2;
   A(i,i+1)=(1.0/(delx^2))*delx^2;
   r(i)=-(cconf/T)*(c0+c1*x(i)+c2*x(i)^2)*delx^2;//delx^2 is multiplied
with each term for scaling
end
//Right boundary
//Two point
A(nnode,nnode)=(1/delx)*delx;
A(nnode,nnode-1)=-(1/delx)*delx;//delx is multiplied for scaling
r(nnode)=0;
//....
ho=hs*ones(nnode,1); //Initial guess =value specified at left hand
boundary
eps_max=1e-6
omega=1
[count,rmse,h]=gseidel(A,r,ho,eps_max,omega)
plot(x,h','-g')
[L,U,E]=lu(A)
D=diag(diag(A))
for i=1:nnode
   L(i,i)=0
   U(i,i)=0
end
c=-inv(D)*(L+U)
eigv=max(abs(spec(c)))
disp("Eigenvalue",eigv)
```

```
Output:
"Count rmse"

  1.   0.0117472

 "Count rmse"

  2.   0.0117093

 "Count rmse"

  3.   0.0116714

 "Count rmse"

  4.   0.0116335

 "Count rmse"

  5.   0.0115956
          .
          .
          .
          .
"Count rmse"

  2858.   0.000001

 "Count rmse"

  2859.   0.000001


 "Eigenvalue"

  0.9740632
```



(10) GW_LAPLACE_2D

Problem Definition

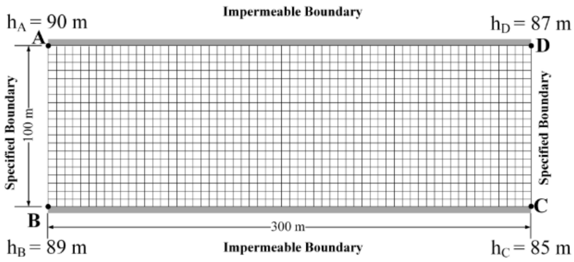


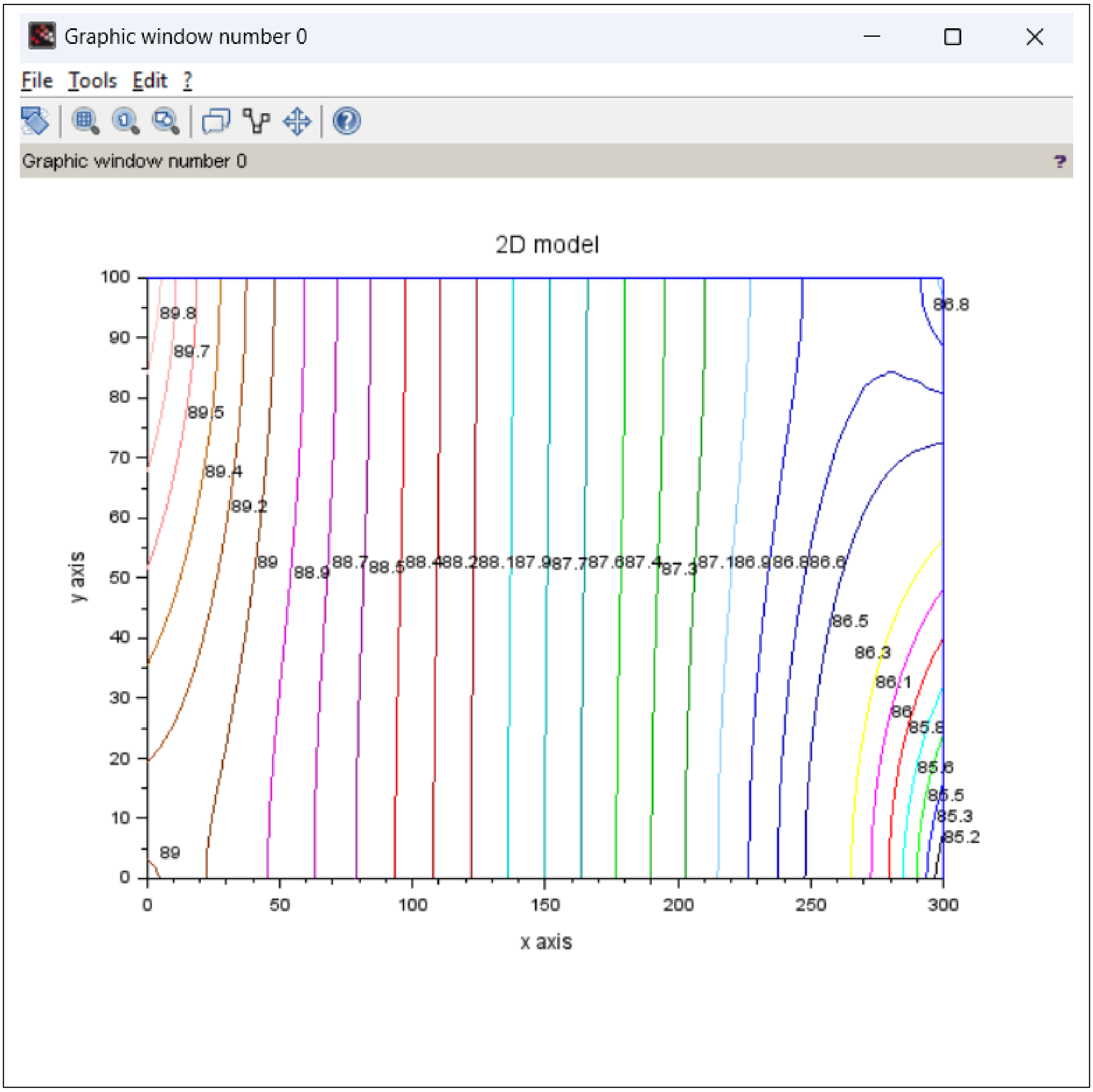
Figure: Homogeneous Aquifer System (Dhar and Patil, 2012)

```
//GW LAPLACE 2D
clc
clear
//...problem dependent parameters...
mnode=31;
nnode=21;
Lx=300;
Ly=100;
hA=90;
hB=89;
hC=85;
hD=87;
//calculated values of parameters
delta_x=Lx/(mnode-1);
delta_y=Ly/(nnode-1);
x=0:delta_x:Lx;
y=0:delta_y:Ly;
alphax=1/(delta_x)^2;
alphay=1/(delta_y)^2;
mnnode=mnode*nnode;
A=zeros(mnnode,mnnode);
r=zeros(mnnode,1);
for j=1:nnode
for i=1:mnode
l=i+(j-1)*mnode; //single index notation
//corner points are considered at specified boundary condition
```

```
//node A
if (i==1 & j==nnode) then
A(l,l)=1;
r(l)=hA;
end
//node B
if (i==1 &j==1) then
A(l,l)=1;
r(l)=hB;
end
//Node c
if (i==mnode & j==1)then
A(l,l)=1;
r(l)=hC;
end
//Node D
if (i==mnode & j==nnode)then
A(l,l)=1;
r(l)=hD;
end
//Interior points
if (i>1 & i<mnode) then
if(j>1 & j<nnode)then
   A(l,l-mnode)=alphay;
   A(l,l-1)=alphax;
   A(l,l)=-2*(alphax+alphay)
   A(l,l+1)=alphax;
   A(l,l+mnode)=alphay;
   r(l)=0;
end
end
//specified LBC
if (i==1) then
   if (j>1 & j<nnode)then
      A(l,l)=1.0;
      r(l)=hB+(hA-hB)*(j-1)*(delta_y/Ly);
   end
end
//specified RBC
if (i==mnode) then
   if (j>1 & j<nnode)then
      A(l,l)=1.0;
      r(l)=hC+(hD-hC)*(j-1)*(delta_y/Ly);
   end
end
```

```
//No flow BBC
if (j==1)then
   if (i>1 & i<mnode)then
      //3 point boundary
      A(l,l)=-3/(2*delta_y);
      A(l,l+mnode)=4/(2*delta_y);
      A(l,l+2*mnode)=-1/(2*delta_y);
      r(l)=0;
   end
end
//No flow TBC
if (j==nnode) then
   if (i>1 & i<mnode)then
      //3 point boundary
      A(l,l)=-3/(2*delta_y);
      A(l,l-mnode)=4/(2*delta_y);
      A(l,l-2*mnode)=-1/(2*delta_y);
      r(l)=0;
   end
end
end
end
//Solution of system
h=A\r
for j=1:nnode
   for i=1:mnode
      l=i+(j-1)*mnode;
      hdata(i,j)=h(l);
   end
end
contour(x,y,hdata,30)
xtitle("2D model","x axis","y axis");
plot([0 300],[100 100],'-')
plot([300,300],[0 100],'-')
```

Output:



Graphic window number 0
File Tools Edit ?
Graphic window number 0
2D model
y axis
x axis
100
90
80
70
60
50
40
30
20
10
0
0
50
100
150
200
250
300
89.8
89.7
89.5
89.4
89.2
89
88.9
86.8
86.5
86.3
86.1
86
85.8
85.6
85.5
85.3
85.2
89

(11) GW_STEADY_LAPLACE_2D_ITERATIVE

Problem Definition

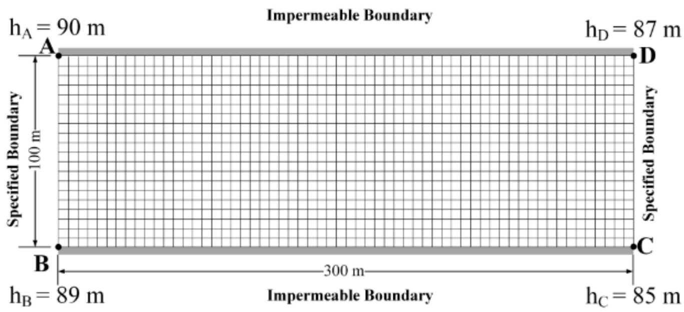


Figure: Homogeneous Aquifer System (Dhar and Patil, 2012)

```
clc
clear
//...problem dependent parameters...
mnode=31;
nnode=21;
Lx=300;
Ly=100;
hA=90;
hB=89;
hC=85;
hD=87;
omega=1.0;
//calculated values of parameters
delta_x=Lx/(mnode-1);
delta_y=Ly/(nnode-1);
x=0:delta_x:Lx;
y=0:delta_y:Ly;
alphax=1/(delta_x)^2;
alphay=1/(delta_y)^2;
eps_max=1e-3;
```

```
//initialization
h=hA*ones(mnode,nnode);
count=0;
rmse=1
while rmse>eps_max
   rmse=0;
   for j=1:nnode
      for i=1:mnode
         if (i>1 & i<mnode) then
            if (j>1 & j<nnode) then //interor nodes
               cencoff=-2*(alphax+alphay);
               res=-alphay*h(i,j-1)-alphax*h(i-1,j)+2*(alphax+alphay)*h(i,j)-
alphax*h(i+1,j)-alphay*h(i,j+1);
               h(i,j)=h(i,j)+omega*res/cencoff
               rmse=rmse+(omega*res/cencoff).^2;
            end
         end
         //Node A
         if (i==1 & j==nnode)then
            h(i,j)=hA;
         end
         //Node B
         if (i==1 & j==1) then h(i,j)=hB; end
         //Node C
         if (i==mnode & j==1) then h(i,j)=hC; end
         //Node D
         if (i==mnode & j==nnode) then h(i,j)=hD;end
         //Specified LBC
         if (i==1) then
            if (j>1 & j<nnode)then
               h(i,j)=hB+(hA-hB)*(j-1)*(delta_y/Ly);
            end
         end
         //Specified RBC
         if (i==mnode) then
            if (j>1 & j<nnode) then
               h(i,j)=hC+(hD-hC)*(j-1)*(delta_y/Ly);
            end
         end
```

```
//Neuman BBC
if (j==1) then
if(i>1 & i<mnode) then
h(i,j)=(4*h(i,j+1)-h(i,j+2))/3;
end
end
//Neumann TBC
if (j==nnode) then
if (i>1 & i<mnode)then
//3 point
h(i,j)=(4*h(i,j-1)-h(i,j-2))/3;
end
end
end
end

rmse=sqrt(rmse/(mnode*nnode)); //overall rmse value
count=count+1
disp([count rmse])
end

contour(x,y,h,50)
xtitle("2D model","x axis","y axis");
plot([0 300],[100 100],'-')
plot([300,300],[0 100],'-')
```

Output:

1. 0.0158392
2. 0.1121481
3. 0.0803617
4. 0.0624525
5. 0.0513152

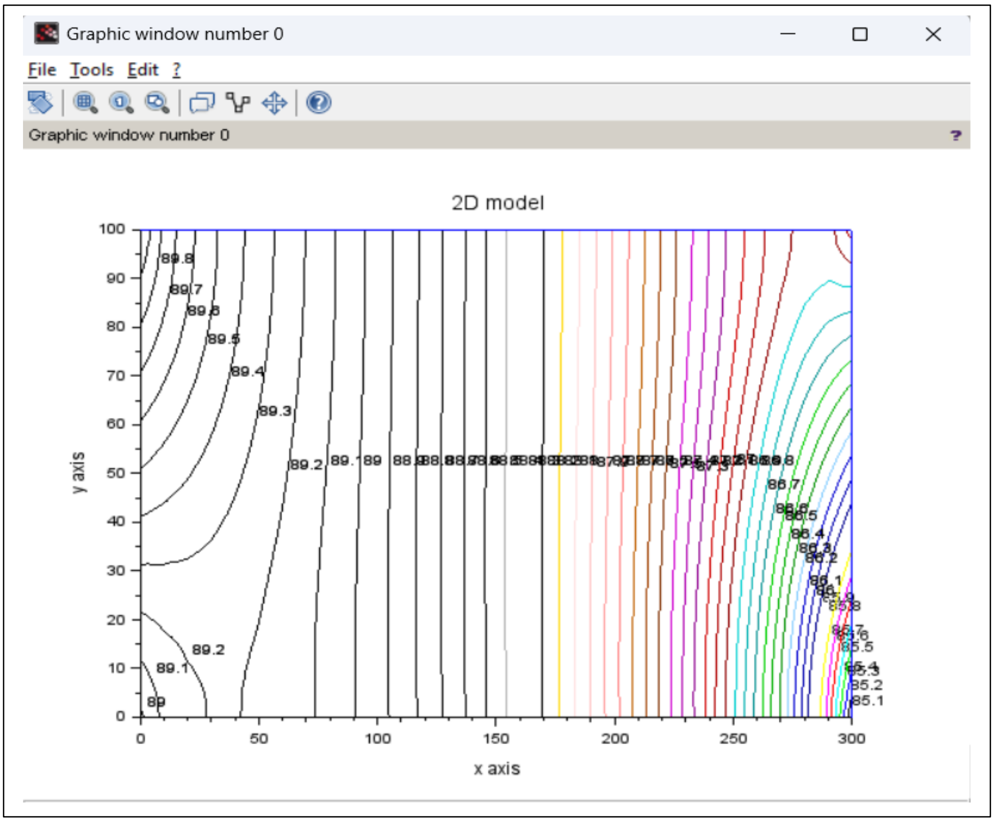
.

.

.

.

663. 0.0010074
664. 0.0010054
665. 0.0010033
666. 0.0010012
667. 0.0009991



(12) GW_UNSTEADY_2D_EXPLICIT

//Error level: low. (Some cross-contours in graph).

Problem Definition

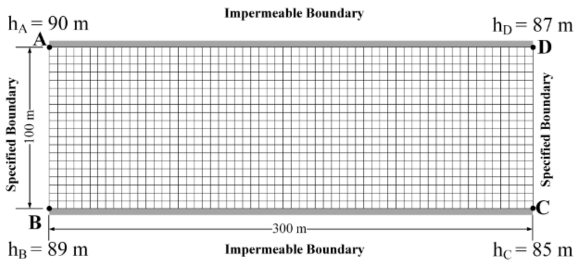


Figure: Homogeneous Aquifer System (Dhar and Patil, 2012)

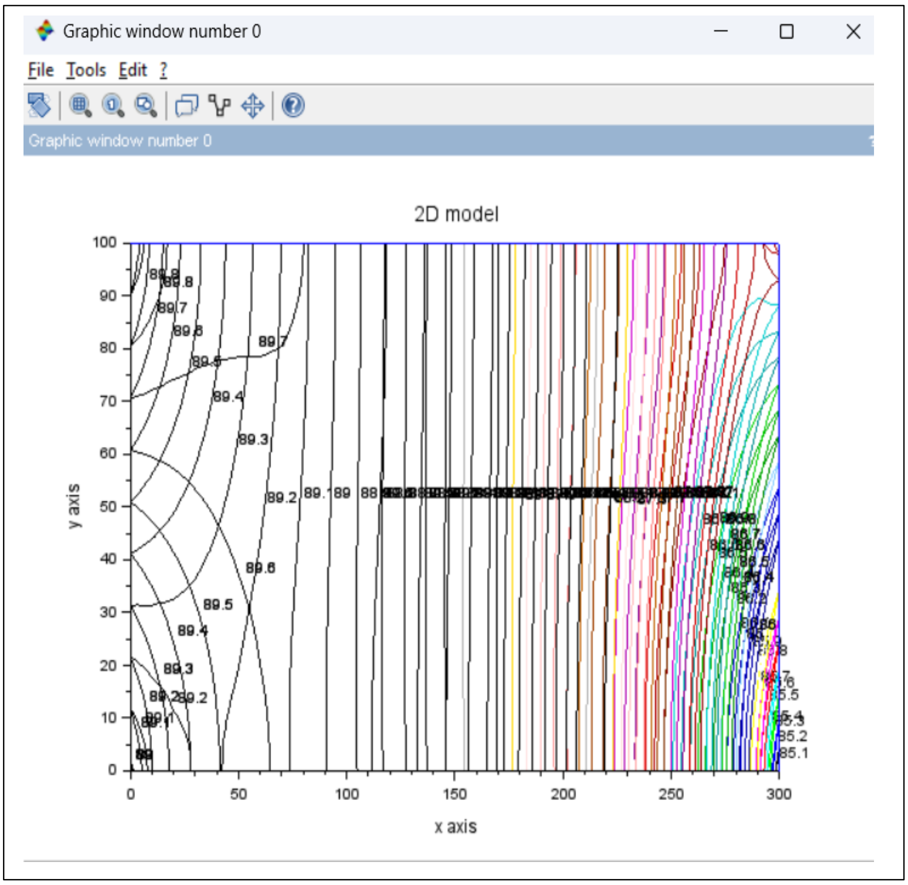
```
clc
clear
//....Problem dependent paramaters...
mnode=31;
nnode=21;
Lx=300;
Ly=100;
hA=90;
hB=89;
hC=85;
hD=87;
Time_max=5;
eps_max=1e-3;
S=5e-5;//storativity
T=200;//Transmissivity
//Calculated parameter values
delta_x=Lx/(mnode-1);
delta_y=Ly/(nnode-1);
x=0:delta_x:Lx;
y=0:delta_y:Ly;
delta_t=0.5;//we assumed. May need to be recalculated based on
alphax and alphay values.
alphax=(T*delta_t)/(S*delta_x^2);
alphay=(T*delta_t)/(S*delta_y^2);
sumalpha=alphax+alphay;
```

```
while sumalpha>0.5
   delta_t=delta_t/2
   alphax=(T*delta_t)/(S*delta_x^2);
   alphay=(T*delta_t)/(S*delta_y^2);
   sumalpha=alphax+alphay;//this while loop is to ensure stability
criteria of explicit problem and to take appropriate delta_t value.
End
disp("delta_t",delta_t)

//Initialization
ho=hA*ones(mnode,nnode);//ho for old time level
hn=zeros(mnode,nnode);//hn for new time level
//Boundary Condition
for j=1:nnode
   //Dirichlet LBC
   ho(1,j)=hB+(hA-hB)*(j-1)*(delta_y/Ly);
   //Dirichlet RBC
   ho(mnode,j)=hC+(hD-hC)*(j-1)*(delta_y/Ly);
end
count=0;
rmse=1;
t=0;
//Time loop
while t<Time_max
   t=t+delta_t;
   //Interior node
   for j=1:nnode
      for i=1:mnode
         if (i>1 & i<mnode)then
            if (j>1 & j<nnode)then
               hn(i,j)=alphay*ho(i,j-1)+alphax*ho(i-1,j)+(1-
2*(alphax+alphay))*ho(i,j)+alphax*ho(i+1,j)+alphay*ho(i,j+1);
end
end
end
end
//Boundary nodes
for j=1:nnode
   for i=1:mnode
```

```
//node A
   if (i==1 & j==nnode)then
      hn(i,j)=hA;
   end
   //Node B
   if (i==1 & j==1) then hn(i,j)=hB; end
   //Node C
   if (i==mnode & j==1) then hn(i,j)=hC; end
   //Node D
   if (i==mnode & j==nnode) then hn(i,j)=hD;end
   //Specified LBC
   if (i==1) then
      if (j>1 & j<nnode)then
         hn(i,j)=hB+(hA-hB)*(j-1)*(delta_y/Ly);
      end
   end
   //Specified RBC
   if (i==mnode) then
      if (j>1 & j<nnode) then
         hn(i,j)=hC+(hD-hC)*(j-1)*(delta_y/Ly);
      end
   end
//Neuman BBC
if (j==1) then
if(i>1 & i<mnode) then
hn(i,j)=(4*hn(i,j+1)-hn(i,j+2))/3;
end
end
//Neumann TBC
if (j==nnode) then
if (i>1 & i<mnode)then
//3 point
hn(i,j)=(4*hn(i,j-1)-hn(i,j-2))/3; //Boundary conditions are
evaluated based on new time level values
end
end
end
end
```

```
rmse=0;
for j=1:nnode
   for i=1:mnode
      rmse=rmse+(hn(i,j)-ho(i,j)).^2;
      ho(i,j)=hn(i,j);
end
end
rmse=sqrt(rmse/(mnode*nnode));
disp("t rmse",[t rmse])
if (rmse<eps_max) then
   break       //while loop terminated
end
end
contour(x,y,hn,50)
xtitle("2D model","x axis","y axis");
plot([0 300],[100 100],'-')
plot([300,300],[0 100],'-')
```

```
Output:
"delta_t"
  0.0000019

 "t rmse"
  0.0000019   0.0559856

 "t rmse"
  0.0000038   0.0476177

 "t rmse"
  0.0000057   0.0411457
        .
        .
        .
        .
"t rmse"
  0.0012608   0.0010001

 "t rmse"
  0.0012627   0.0009991
```



(13) GW_UNSTEADY_2D_IMPLICIT_ITERATIVE

Problem Definition

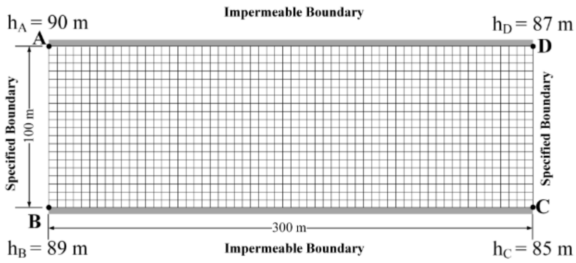


Figure: Homogeneous Aquifer System (Dhar and Patil, 2012)

```
clc
clear
//....Problem dependent paramaters...
mnode=31;
nnode=21;
Lx=300;
Ly=100;
hA=90;
hB=89;
hC=85;
hD=87;
Time_max=5;
eps_max=1e-3;
S=5e-5;//storativity
T=200;//Transmissivity
omega=1;
delta_x=Lx/(mnode-1);
delta_y=Ly/(nnode-1);
x=0:delta_x:Lx;
y=0:delta_y:Ly;
delta_t=0.5;//Implicit scheme is unconditionally stable.No need to update
alphax=(T*delta_t)/(S*delta_x^2);
alphay=(T*delta_t)/(S*delta_y^2);
```

```
//Initialization
ho=hA*ones(mnode,nnode);
hn=zeros(mnode,nnode);
//Boundary Condition
for j=1:nnode
   //Dirichlet LBC
   ho(1,j)=hB+(hA-hB)*(j-1)*(delta_y/Ly);
   //Dirichlet RBC
   ho(mnode,j)=hC+(hD-hC)*(j-1)*(delta_y/Ly);
end
//Time loop
t=0;
while t<Time_max
   t=t+delta_t;
   count=0;
   rmse=1;
   //space loop
   while rmse>eps_max
      rmse=0;
      for j=1:nnode

      for i=1:mnode
         if (i>1 & i<mnode) then
            if (j>1 & j<nnode) then //interor nodes
               cencoff=-(1+2*(alphax+alphay));
               res=-ho(i,j)-alphay*hn(i,j-1)-alphax*hn(i-
1,j)+(1+2*(alphax+alphay)*hn(i,j))-alphax*hn(i+1,j)-
alphay*hn(i,j+1);
               hn(i,j)=hn(i,j)+omega*res/cencoff
               rmse=rmse+(omega*res/cencoff).^2;
            end
         end
```

```
//node A
        if (i==1 & j==nnode)then
           hn(i,j)=hA;
        end
        //Node B
        if (i==1 & j==1) then hn(i,j)=hB; end
        //Node C
        if (i==mnode & j==1) then hn(i,j)=hC; end
        //Node D
        if (i==mnode & j==nnode) then hn(i,j)=hD;end
        //Specified LBC
        if (i==1) then
           if (j>1 & j<nnode)then
              hn(i,j)=hB+(hA-hB)*(j-1)*(delta_y/Ly);
           end
        end
        //Specified RBC
        if (i==mnode) then
           if (j>1 & j<nnode) then
              hn(i,j)=hC+(hD-hC)*(j-1)*(delta_y/Ly);
           end
        end
 //Neuman BBC
        if(j==1)then
           if (i>1 & i<mnode) then
              res=hn(i,j+1)-hn(i,j);
              hn(i,j)=hn(i,j)+omega*(hn(i,j+1)-
hn(i,j));//evaluated considering 2 points
              rmse=rmse+(omega*res).^2; //sum of individual
errors
           end
        end
        //Neuman TBC
        if(j==nnode)then
           if (i>1 & i<mnode) then
              res=hn(i,j-1)-hn(i,j);
              hn(i,j)=hn(i,j)+omega*(hn(i,j-1)-hn(i,j));
              rmse=rmse+(omega*res).^2;
           end
        end
     end
  end
```

```
rmse=sqrt(rmse/(mnode*nnode)); //overall rmse value
count=count+1
disp([count rmse])
end
rmse=0;
for j=1:nnode
   for i=1:mnode
      rmse=rmse+(hn(i,j)-ho(i,j)).^2;
      ho(i,j)=hn(i,j);
end
end

if (rmse<eps_max) then
   break       //while loop terminated
end
end
contour(x,y,hn,30)
xtitle("2D model","x axis","y axis");
plot([0 300],[100 100],'-')
plot([300,300],[0 100],'-')
```

Output:

//count rmse

1. 2.5735614
2. 3.0996063
3. 2.3321854
4. 1.8629285
5. 1.5528867

.

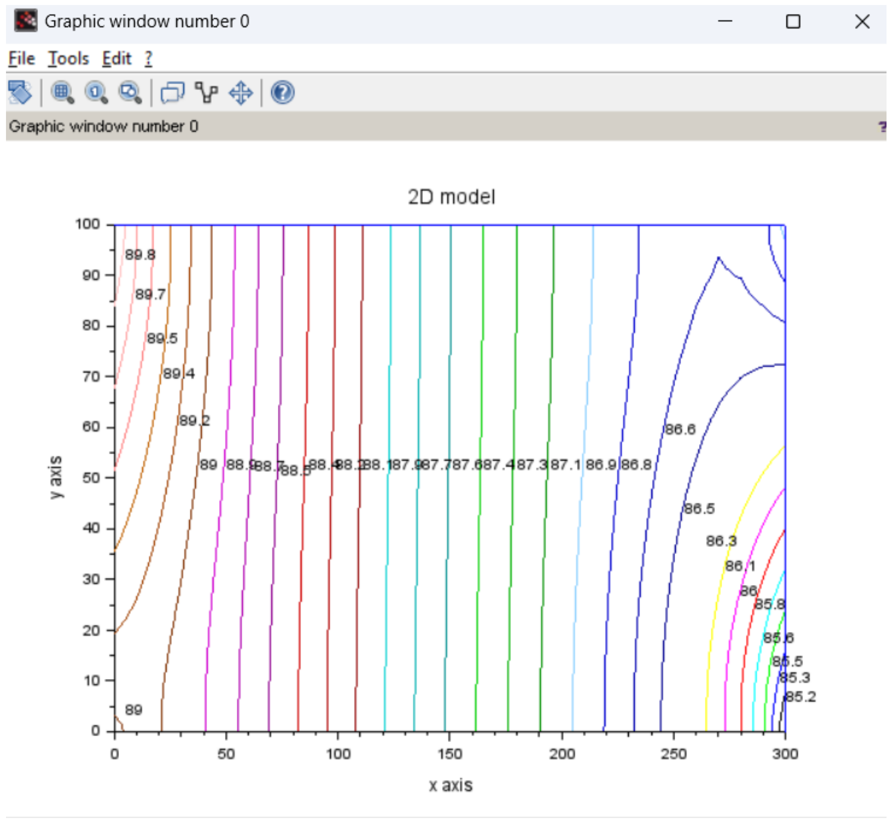
.

.

2 425. 0.0010022

2426. 0.0010001

2427. 0.000998



(14) GW unsteady 2D iterative implicit FVM

//Error level: …… (Syntax error, no output is coming). ☹

Problem Definition

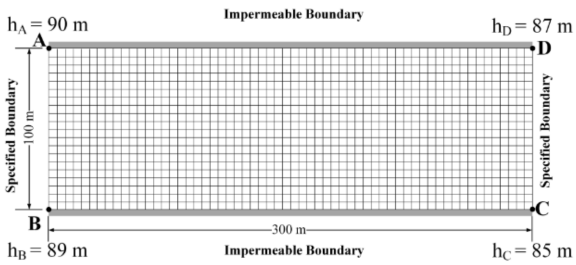


Figure: Homogeneous Aquifer System (Dhar and Patil, 2012)

```
clc
clear
//GW unsteady 2D iterative implicit FVM
//....Problem dependent parameters..
mnode=31;
nnode=21;
mcell=mnode-1;
ncell=nnode-1;
Lx=300;
Ly=100;
hA=90;
hB=89;
hC=85;
hD=87;
Time_max=50;
eps_max=1e-3;
S=5e-5;//storativity
T=200;//Transmissivity
omega=1;
delta_x=Lx/(mnode-1);
delta_y=Ly/(nnode-1);
x=delta_x/2:delta_x:(Lx-delta_x/2);
y=delta_y/2:delta_y:(Ly-delta_y/2);
delta_t=0.5;
alphax=(T*delta_t)/(S*delta_x^2);
alphay=(T*delta_t)/(S*delta_y^2);
```

```
//modified corner heads (Because, these values should be defined
at cell centered positions, not at the corners anymore)
hAm=hB+((hA-hB)/Ly)*(Ly-delta_y/2); //Ly-delta_y/2 is the
distance between corner point B and cell centered A.
hBm=hB+((hA-hB)/Ly)*(delta_y/2);
hCm=hC+((hD-hC)/Ly)*(delta_y/2);
hDm=hC+((hD-hC)/Ly)*(Ly-delta_y/2);
//Initialization
ho=hA*ones(mcell,ncell);//old time level
hn=hA*ones(mcell,ncell);//New time level
//Time loop
t=0;
while t<Time_max
   t=t+delta_t;
   count=0;
   rmse=1;
//space loop
while rmse>eps_max
   rmse=0;
   for j=1:ncell
      for i=1:mcell
//Interior nodes
if(i>1 & i<mcell) then
   if(j>1 & j<ncell)then
      a_S=alphay;
      a_W=alphax;
      a_P=-1-(2*(alphax+alphay));
      a_E=alphax;
      a_N=alphay;
      r_P=-ho(i,j);
      res=r_P-(a_S*hn(i,j-1)+a_W*hn(i-
1,j)+a_P*hn(i,j)+a_E*hn(i+1,j)+a_N*hn(i,j+1));
end
end
//cell A
if (i==1 & j==ncell) then
  a_S=alphay;
  a_W=0;
  a_P=-1-4*alphax-alphay;
  a_E=(4/3)*alphax;
  a_N=0;
end
```

```
//Cell B
if (i==1 & j==1) then
   a_S=0;
   a_W=0;
   a_P=-1-4*alphax-alphay;
   a_E=(4/3)*alphax;
   a_N=alphay;
   r_P=-ho(i,j)-(8/3)*alphax*hBm;
   res=r_P-(a_P*hn(i,j)+a_E*hn(i+1,j)+a_N*hn(i,j+1));
end
//Cell C
if (i==mcell & j==1) then
   a_P=-1-4*alphax-alphay;
   res=(-ho(i,j)-(8/3)*alphax*hCm)-((4/3)*alphax*hn(i-1,j)-
(1+4*alphax+alphay)*hn(i,j)+alphay*hn(i,j+1));
end
//Cell D
if (i==mcell & j==ncell) then
   a_P=-(1+4*alphax+alphay);
   res=(-ho(i,j)-(8/3)*alphax*hDm)-(alphay*hn(i,j-
1)+(4/3)*alphax*hn(i-1,j)-(1+4*alphax+alphay)*hn(i,j));
end
//Specified LBC
if (i==1) then
   if (j>1 & j<ncell)then
    a_P=-(1+4*alphax+2*alphay);
   hvl=hBm+(hAm-hBm)*(j-1)*((delta_y)/(Ly-delta_y));
//hvl=head value at the left boundary.As FVM considers cell
centered values dimensions of the domain length=(Lx-delta_x) and
width=(Ly-delta_y)
   res=(-ho(i,j)-(8/3)*alphax*hvl)-(alphay*hn(i,j-1)-
(1+4*alphax+2*alphay)*hn(i,j)+(4/3)*alphax*hn(i+1,j)+alphay*
hn(i,j+1));
end
end
```

```
//Specified RBC
if (i==mcell) then
   if (j>1 & j<ncell)then
      a_P=-(1+4*alphax+2*alphay);
      hvr=hCm+(hDm-hCm)*(j-1)*delta_y/(Ly-delta_y);
      res=(-ho(i,j)-(8/3)*alphax*hvr)-(alphay*hn(i,j-1)-
(1+4*alphax+2*alphay)*hn(i,j)+(4/3)*alphax*hn(i-
1,j)+alphay*hn(i,j+1))
end
//Neumann TBC
if (j==ncell) then
   if (i>1 & i<mcell)then
      a_P=-(1+2*alphax+alphay);
      res=-ho(i,j)-(alphay*hn(i,j-1)+alphax*hn(i-1,j)-
(1+2*alphax+alphay)*hn(i,j)+alphax*hn(i+1,j));
end
end
//Neumann BBC
if (j==1) then
   if (i>1 & i<mcell)then
      a_P=-(1+2*alphax+alphay);
      res=-ho(i,j)-(alphax*hn(i-1,j)-
(1+2*alphax+alphay)*hn(i,j)+alphax*hn(i+1,j)++alphay*hn(i,j+1));
end
end
//Update
hn(i,j)=hn(i,j)+omega*res/a_P;
rmse=rmse+(omega*res/a_P).^2;//Total rmse
end
end
rmse=sqrt(rmse/(mcell*ncell));//Actual rmse
count =count +1;
disp([count rmse])
end
```

```
//To check whether things are converging with time or not
rmse_t=0;
for j=1:ncell
   for i=1:mcell
      rmse_t=rmse_t+(hn(i,j)-ho(i,j)).^2;
      ho(i,j)=hn(i,j);
   end
end
//Condition for steady state
if (rmse_t<eps_max) then
   break
end
end
//Boundary information
hdata=zeros(ncell+2,mcell+2);//corner points need to be added to
this matrix to get the picture of full domain. those points were
excluded during FVM formulation.
//Internal cells
for j=2:ncell+1;
   for i=mcell+1
      hdata(i,j)=hn(i-1,j-1);//because, after updating, one extra row
of nodes added to the bottom.
   end
end
//Input the corner values at 'hdata' matrix
hdata(1,ncell+2)=hA;
hdata(1,1)=hB;
hdata(mcell+2,1)=hC;
hdata(mcell+2,ncell+2)=hD;

//Left and right boundary
for j=2:ncell+1
   hdata(1,j)=hBm+(hAm-hBm)*((j-2)*delta_y)/(Ly-
delta_y);//left boundary. along a column of 'hdata',1st
node,h(1,1)=hB; 2nd node h(2,1)=hBm; etc.

hdata(mcell+2,j)=hCm+(hDm-hCm)*(j-2)*(delta_y/(Ly-delta_y));
//Right boundary
end
```

```
//Bottom and top boundary
for i=2:mcell+1
   hdata(i,1)=(9*hn(i-1,1)-hn(i-1,2))/8;//Bottom boundary
   hdata(i,ncell+2)=(9*hn(i-1,1)-hn(i-1,ncell-1))/8;

end
end
xn=[0 x Lx];
yn=[0 y Ly];
contour(xn,yn,hdata,30)
xtitle("2D FVM","x axis","y axis");
plot([0 300],[100 100],'-')
plot([300,300],[0 100],'-')
```

```
Output:

at line    91 of function contour2d ( C:\Program Files\scilab-6.1.1\modules\graphics\macros\contour2d.sci line 103 )

at line    91 of function contour   ( C:\Program Files\scilab-6.1.1\modules\graphics\macros\contour.sci line 104 )

at line   165 of executed file C:\Users\sayak\OneDrive\Desktop\CH_assignments\GW_UNSTEADY_2D_IMPLICIT_ITERATIVE_FVM.sce


contour2di: Wrong size for input arguments: Incompatible sizes.
```

(15) GVF_FORWARD_EULER

Problem Statement

Given

Channel Cross-Section Type: Rectangular

$y_0 = 0.8m$
$B = 15m$
$g = 9.81m/s^2$
$S_0 = 0.0008$
$n = 0.015$
$L_x = 200m$
$Q = 20m^3/s$

Required

Identify the type of GVF Profile: ?
Plot of the GVF Profile.

```
//GRADUALLY VARIED FLOW-FORWARD EULER
clc
clear all
//Data given
Q=20;
S0=0.0008;
B=15;//m, Channel width @rectangular channel
Lx=200;//length of the channel reach
y0=0.8;//initial depth at x=0
n=0.015;
g=9.81;
mnode=201;//number of nodes in the x directions
global('Q','S0','n','B','g')//global variables . for writing multiple functions later, we can directly utilize these constant values.

//**********************************************
//calculation of critical depth
yc=(Q^2/(g*B^2))^(1/3);//for rectangular channel.equation (73)
disp("Critical Depth",yc)
```

```
//***************************************************
//calculation of normal depth using NR method [from equation
(75.1)]

//To define G and G' functions
function Gval=Gfunc(y)
   Gval=(S0^(1/2)*B^(5/3)/n)*(y/(B+2*y))^(2/3)*y-Q;//eqn(74)
endfunction
function Gpval=Gderi(y)
   term1=(S0^(1/2)*B^(5/3))/(3*n);
   term2=y^(2/3)*(5*B+6*y);
   term3=(B+2*y)^(5/3);
//
Gpval=(S0^(1/2)*B^(5/3))/(3*n)*y^(2/3)*(5*B+6*y)/(B+2*y)^(5/3)
;//G' from eqn (76)
Gpval=term1*term2/term3;
endfunction

//Newton-Raphson method (to solve nonlinear 'Gval' eqaution for
getting yn)
eps_max=1e-6;
aerror=1; //absolute error
yn=yc; //To start the iteration loop, critical depth is our initial guess.
while abs(aerror)>eps_max
   aerror=(Gfunc(yn)/Gderi(yn));
   yn=yn-aerror; //yn(p)=yn(p-1)-(G(yn(p-1))/G'(yn(p-
1)))..eqn(75.1))
end
disp("Normal Depth",yn)
//****************************************

//dydx function calculation from Governing Equation
function dydx=psi(x, y)  //equation (73)
   A_y=B*y;
   P_y=B+2*y;
   R_y=A_y/P_y;//Area, Perimeter and hydraulic radius are function
of y
   Sf=(n^2*Q^2)/(R_y^(4/3)*A_y^2);//friction slope
   Frs=(Q^2*B)/(g*A_y^3);//Fr^2
   dydx=(S0-Sf)/(1-Frs);
endfunction
```

```
//*********************************************
//Forward Euler method [to get GVF profile for nodes along x
direction]
xc=linspace(0,Lx,mnode);//x coordinate vector of nodes
delta_x=Lx/(mnode-1);
yv=zeros(mnode,1);//Initialization of flow depth vector
yv(1)=y0;//Input Boundary value in yv vector
//input other entries
for i=2:mnode
  yv(i)=yv(i-1)+delta_x*psi(xc(i-1),yv(i-1));
end

plot(xc,yv,"-b")
set(gca(),"auto_clear","off")
plot([0 Lx],[yn yn],'-m')
set(gca(),"auto_clear","off")
plot([0 Lx],[yc yc],'-b')
xtitle("GVF forward Euler Method","x axis(m)","flow
depth(m)")
```

Output:

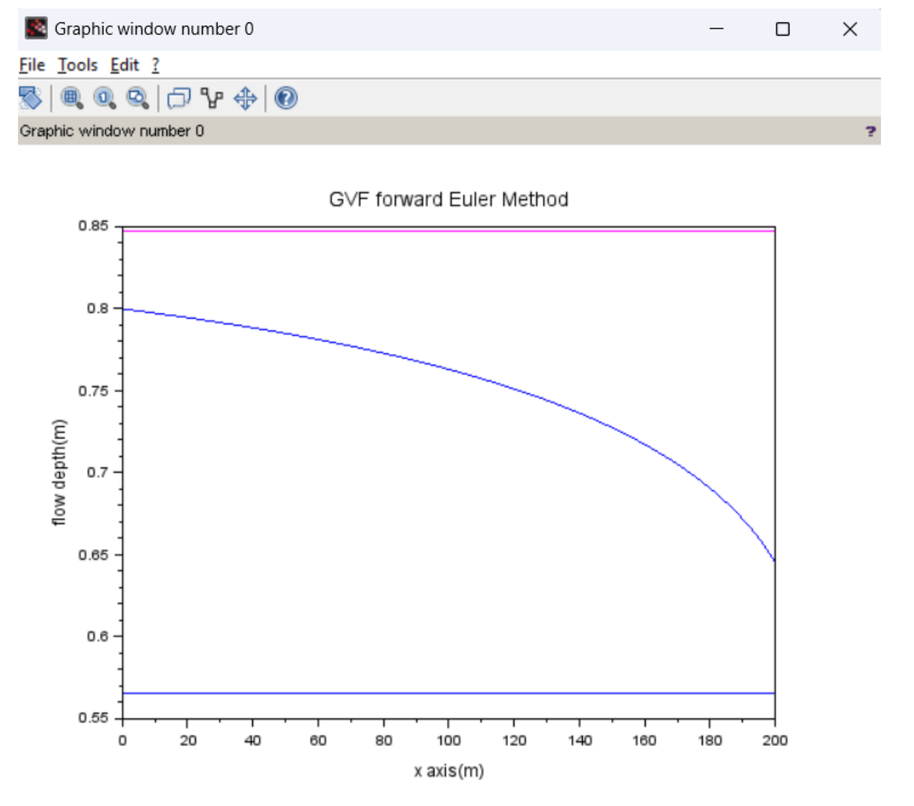
"Critical Depth"

0.5658954

"Normal Depth"

0.8478039

WARNING: Transposing row vector X to get compatible dimensions.



(16) GVF_EULER_CAUCHY

Problem Statement

Given

Channel Cross-Section Type: Rectangular

$y_0 = 0.8m$

$B = 15m$

$g = 9.81m/s^2$

$S_0 = 0.0008$

$n = 0.015$

$L_x = 200m$

$Q = 20m^3/s$

Required

Identify the type of GVF Profile: ?

Plot of the GVF Profile.

```
//GRADUALLY VARIED FLOW- EULER CAUCHY METHOD
clc
clear all
//Data given
Q=20;
S0=0.0008;
B=15;//m, Channel width @rectangular channel
Lx=200;//length of the channel reach
y0=0.8;//initial depth at x=0
n=0.015;
g=9.81;
mnode=201;//number of nodes in the x directions
global('Q','S0','n','B','g')//global variables . for writing multiple functions later, we can directly utilize these constant values.

//**********************************************
//calculation of critical depth
yc=(Q^2/(g*B^2))^(1/3);//for rectangular channel.equation (73)
disp("Critical depth",yc)
```

```
//**************************************************
//calculation of normal depth using NR method [from equation (75.1)]

//To define G and G' functions
function Gval=Gfunc(y)
   Gval=(S0^(1/2)*B^(5/3)/n)*(y/(B+2*y))^(2/3)*y-Q;//eqn(74)
endfunction
function Gpval=Gderi(y)
   term1=(S0^(1/2)*B^(5/3))/(3*n);
   term2=y^(2/3)*(5*B+6*y);
   term3=(B+2*y)^(5/3);
//
Gpval=(S0^(1/2)*B^(5/3))/(3*n)*y^(2/3)*(5*B+6*y)/(B+2*y)^(5/3);/
/G' from eqn (76)
Gpval=term1*term2/term3;
endfunction

//Newton-Raphson method (to solve nonlinear 'Gval' eqaution for
getting yn)
eps_max=1e-6;
aerror=1; //absolute error
yn=yc; //To start the iteration loop, critical depth is our initial guess.
while abs(aerror)>eps_max
   aerror=(Gfunc(yn)/Gderi(yn));
   yn=yn-aerror; //yn(p)=yn(p-1)-(G(yn(p-1))/G'(yn(p-1)))..eqn(75.1))
end
disp("Normal Depth",yn)
//****************************************

//dydx function calculation from Governing Equation
function dydx=psi(x, y)  //equation (73)
   A_y=B*y;
   P_y=B+2*y;
   R_y=A_y/P_y;//Area, Perimeter and hydraulic radius are function of
y
   Sf=(n^2*Q^2)/(R_y^(4/3)*A_y^2);//friction slope
   Frs=(Q^2*B)/(g*A_y^3);//Fr^2
   dydx=(S0-Sf)/(1-Frs);
endfunction
```

```
//*********************************************
//Euler-Cauchy method [to get GVF profile for nodes along x direction]
xc=linspace(0,Lx,mnode);//x coordinate vector of nodes
delta_x=Lx/(mnode-1);
yv=zeros(mnode,1);//Initialization of flow depth vector
yv(1)=y0;//Input Boundary value in yv vector
//input other entries
for i=2:mnode
   K1=delta_x*psi(xc(i-1),yv(i-1)); //K1 and K2 for ith point are
calculated from previous (i-1)th point data.(explicit)
   K2=delta_x*psi(xc(i-1)+delta_x,yv(i-1)+K1)

   yv(i)=yv(i-1)+0.5*(K1+K2);//Equation (78)
end

plot(xc,yv,"-b")
set(gca(),"auto_clear","off")
plot([0 Lx],[yn yn],'-m')
set(gca(),"auto_clear","off")
plot([0 Lx],[yc yc],'-b')
xtitle("GVF Euler Cauchy Method","x axis(m)","flow depth")
```

Output:

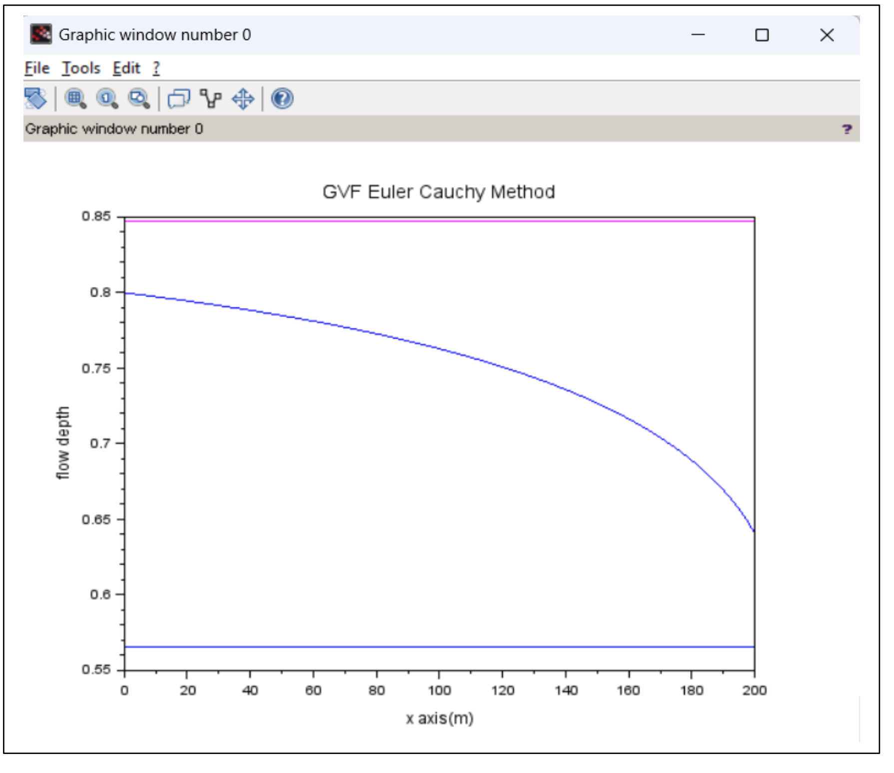
"Critical depth"

0.5658954

"Normal Depth"

0.8478039

WARNING: Transposing row vector X to get compatible dimensions



(17) GVF_Modified_EULER

Problem Statement

Given

Channel Cross-Section Type: Rectangular
$y_0 = 0.8m$
$B = 15m$
$g = 9.81m/s^2$
$S_0 = 0.0008$
$n = 0.015$
$L_x = 200m$
$Q = 20m^3/s$

Required

Identify the type of GVF Profile: ?
Plot of the GVF Profile.

```
//GRADUALLY VARIED FLOW-MODIFIED EULER
clc
clear all
//Data given
Q=20;
S0=0.0008;
B=15;//m, Channel width @rectangular channel
Lx=200;//length of the channel reach
y0=0.8;//initial depth at x=0
n=0.015;
g=9.81;
mnode=201;//number of nodes in the x directions
global('Q','S0','n','B','g')//global variables . for writing multiple functions later, we can directly utilize these constant values.

//**********************************************
//calculation of critical depth
yc=(Q^2/(g*B^2))^(1/3);//for rectangular channel.equation (73)
disp("yc",yc)
```

```
//************************************************
//calculation of normal depth using NR method [from equation (75.1)]

//To define G and G' functions
function Gval=Gfunc(y)
   Gval=(S0^(1/2)*B^(5/3)/n)*(y/(B+2*y))^(2/3)*y-Q;//eqn(74)
endfunction
function Gpval=Gderi(y)
   term1=(S0^(1/2)*B^(5/3))/(3*n);
   term2=y^(2/3)*(5*B+6*y);
   term3=(B+2*y)^(5/3);
// Gpval=(S0^(1/2)*B^(5/3))/(3*n)*y^(2/3)*(5*B+6*y)/(B+2*y)^(5/3);//G' from eqn (76)
Gpval=term1*term2/term3;
endfunction

//Newton-Raphson method (to solve nonlinear 'Gval' eqaution for getting yn)
eps_max=1e-6;
aerror=1; //absolute error
yn=yc; //To start the iteration loop, critical depth is our initial guess.
while abs(aerror)>eps_max
   aerror=(Gfunc(yn)/Gderi(yn));
   yn=yn-aerror; //yn(p)=yn(p-1)-(G(yn(p-1))/G'(yn(p-1)))..eqn(75.1))
end
disp("yn",yn)
//****************************************
//dydx function calculation from Governing Equation
function dydx=psi(x, y)  //equation (73)
   A_y=B*y;
   P_y=B+2*y;
   R_y=A_y/P_y;//Area, Perimeter and hydraulic radius are function of y
   Sf=(n^2*Q^2)/(R_y^(4/3)*A_y^2);//friction slope
   Frs=(Q^2*B)/(g*A_y^3);//Fr^2
   dydx=(S0-Sf)/(1-Frs);
endfunction
```

```
//********************************************
//Modified Euler method [to get GVF profile for nodes along x direction]
xc=linspace(0,Lx,mnode);//x coordinate vector of nodes
delta_x=Lx/(mnode-1);
yv=zeros(mnode,1);//Initialization of flow depth vector
yv(1)=y0;//Input Boundary value in yv vector
//input other entries
for i=2:mnode
   K1=delta_x*psi(xc(i-1),yv(i-1)); //K1 and K2 for ith point are calculated from previous (i-1)th point data.(explicit)
   K2=delta_x*psi(xc(i-1)+0.5*delta_x,yv(i-1)+0.5*K1)

   yv(i)=yv(i-1)+K2;//Equation (78)
end

plot(xc,yv,"-r")
set(gca(),"auto_clear","off")
plot([0 Lx],[yn yn],'-m')
set(gca(),"auto_clear","off")
plot([0 Lx],[yc yc],'-b')
xtitle("GVF Modified Euler Method","x axis(m)","flow depth")
```

Output:

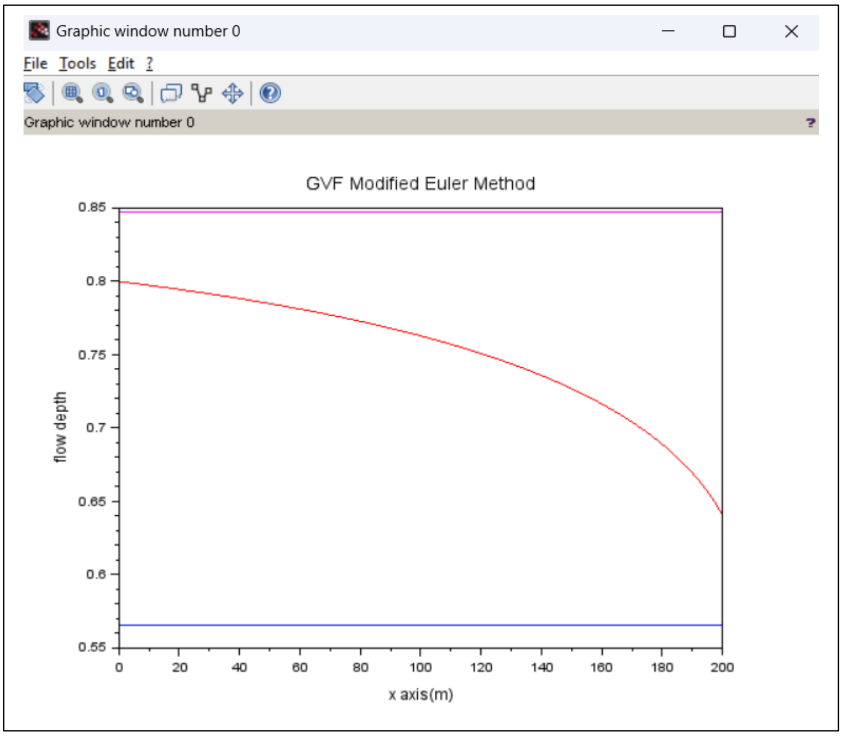
"yc"

0.5658954

"yn"

0.8478039

WARNING: Transposing row vector X to get compatible dimensions



(18) GVF_IMPLICIT_BACKWARD_EULER

Problem Statement

Given

Channel Cross-Section Type: Rectangular

$y_0 = 0.8m$

$B = 15m$

$g = 9.81m/s^2$

$S_0 = 0.0008$

$n = 0.015$

$L_x = 200m$

$Q = 20m^3/s$

Required

Identify the type of GVF Profile: ?

Plot of the GVF Profile.

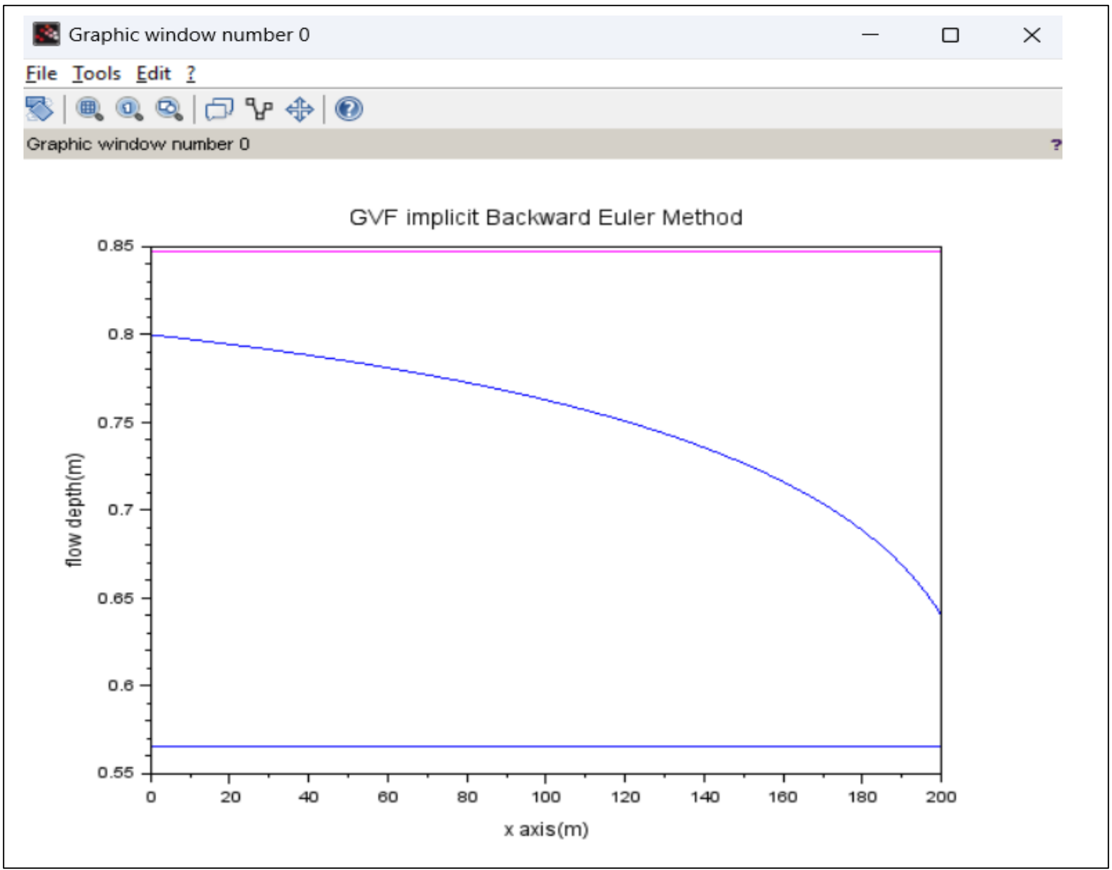
```
//GRADUALLY VARIED FLOW-IMPLICIT BACKWARD EULER
clc
clear
//Data given
Q=20;
S0=0.0008;
B=15;//m, Channel width @rectangular channel
Lx=200;//length of the channel reach
y0=0.8;//initial depth at x=0
n=0.015;
g=9.81;
mnode=201;//number of nodes in the x directions
global('Q','S0','n','B','g')//global variables . for writing multiple functions later, we can directly utilize these constant values.

//************************************************
//calculation of critical depth
yc=(Q^2/(g*B^2))^(1/3);//for rectangular channel.equation (73)
disp("yc",yc)
```

```
//*********************************************
//calculation of critical depth
yc=(Q^2/(g*B^2))^(1/3);//for rectangular channel.equation
(73)
disp(yc)
//************************************************
//calculation of normal depth using NR method [from equation
(75.1)]

//To define G and G' functions
function Gval=Gfunc(y)
  Gval=(S0^(1/2)*B^(5/3)/n)*(y/(B+2*y))^(2/3)*y-
Q;//eqn(127)
endfunction
function Gpval=Gderi(y)
  term1=(S0^(1/2)*B^(5/3))/(3*n);
  term2=y^(2/3)*(5*B+6*y);
  term3=(B+2*y)^(5/3);
//
Gpval=(S0^(1/2)*B^(5/3))/(3*n)*y^(2/3)*(5*B+6*y)/(B+2*y)^(5
/3);//G'
Gpval=term1*term2/term3;//from eqn (129)
endfunction

//Newton-Raphson method (to solve nonlinear 'Gval' eqaution for
getting yn)
eps_max=1e-6;
aerror=1; //absolute error
yn=yc; //To start the iteration loop, critical depth is our initial
guess.
while abs(aerror)>eps_max
  aerror=(Gfunc(yn)/Gderi(yn));
  yn=yn-aerror; //yn(p)=yn(p-1)-(G(yn(p-1))/G'(yn(p-
1)))..eqn(75.1))
end
disp("yn",yn)
```

```
//*****************************************

//dydx function calculation from Governing Equation
function dydx=psi(x, y)  //equation (73)
   A_y=B*y;
   P_y=B+2*y;
   R_y=A_y/P_y;//Area, Perimeter and hydraulic radius are function of
y
   Sf=(n^2*Q^2)/(R_y^(4/3)*A_y^2);//friction slope
   Frs=(Q^2*B)/(g*A_y^3);//Fr^2
   dydx=(S0-Sf)/(1-Frs);
endfunction
//function Fval=Ffunc(y,x,delta_x,yp)
//    Fval=y-delta_x*psi(x,y)-yp //y=new value,yp=previous known
value.Equation(134)
//endfunction
//function Fpval=Fderi(y,delta_x) //derivative of Fval wrt y(new)
//    term1=(1-Q^2/(B^2*g*y^3))^(-1);
//    term2=(2*n^2*Q^2)/(B^2*y^3);
//    term3=(B*y/(B+2*y))^(-4/3);
//    term4=(4*n^2*Q^2)/(3*B^2*y^2);
//    term5=(B*y/(B+2*y))^(-7/3);
//    term6=((B/(B+2*y))-(2*B*y/(B+2*y)^2));
//    term7=(3*Q^2)/(B^2*g*y^4);
//    term8=(1-Q^2/(B^2*g*y^3))^(-2);
//    term9=S0;
//    term10=(n^2*Q^2)/(B^2*y^2);
//    term11=(B*y/(B+2*y))^(4/3);
//    Fpval=1-delta_x*(term1*(term2*term3+term4*term5*term6)-
term7*term8*(term9-term10*term11));
//endfunction
```

```
/////function psipval=psideri(x,y,delta_x)
function psipval=psideri(x, y, delta_x)//derivative of psival wrt
y(new)
   term1=(1-Q^2/(B^2*g*y^3))^(-1);
   term2=(2*n^2*Q^2)/(B^2*y^3);
   term3=(B*y/(B+2*y))^(-4/3);
   term4=(4*n^2*Q^2)/(3*B^2*y^2);
   term5=(B*y/(B+2*y))^(-7/3);
   term6=((B/(B+2*y))-(2*B*y/(B+2*y)^2));
   term7=(3*Q^2)/(B^2*g*y^4);
   term8=(1-Q^2/(B^2*g*y^3))^(-2);
   term9=S0;
   term10=(n^2*Q^2)/(B^2*y^2);
   term11=(B*y/(B+2*y))^(4/3);
   psipval=(1-delta_x*(term1*(term2*term3+term4*term5*term6)-
term7*term8*(term9-term10*term11))-1)/(-delta_x);//using
eqn(135.1)
   endfunction
//*********************************************
//Implicit Backward Euler method [to get GVF profile for nodes along
x direction]
xc=linspace(0,Lx,mnode);//x coordinate vector of nodes
delta_x=Lx/(mnode-1);
yv=zeros(mnode,1);//Initialization of flow depth vector
yv(1)=y0;//Input Boundary value in yv vector
for i=2:mnode
   K1=delta_x*(1-1*delta_x*psideri(xc(i-1)+1*delta_x,yv(i-
1),delta_x))^(-1)*psi(xc(i-1)+1*delta_x,yv(i-1));//using semi-implicit
eqn (138) and putting values from (135.1) and (136.1)
   yv(i)=yv(i-1)+1*K1;
end
//plot
plot(xc,yv,"-b")
set(gca(),"auto_clear","off")
plot([0 Lx],[yn yn],'-m')
set(gca(),"auto_clear","off")
plot([0 Lx],[yc yc],'-b')
xtitle("GVF implicit Backward Euler Method","x axis(m)","flow
depth(m)")
```

```
"yc"

  0.5658954

 "yn"

  0.8478039
WARNING: Transposing row vector X to get compatible dimensions
```



(19) GVF_RK2

Problem Statement

Given

Channel Cross-Section Type: Rectangular
$y_0 = 0.8m$
$B = 15m$
$g = 9.81m/s^2$
$S_0 = 0.0008$
$n = 0.015$
$L_x = 200m$
$Q = 20m^3/s$

Required

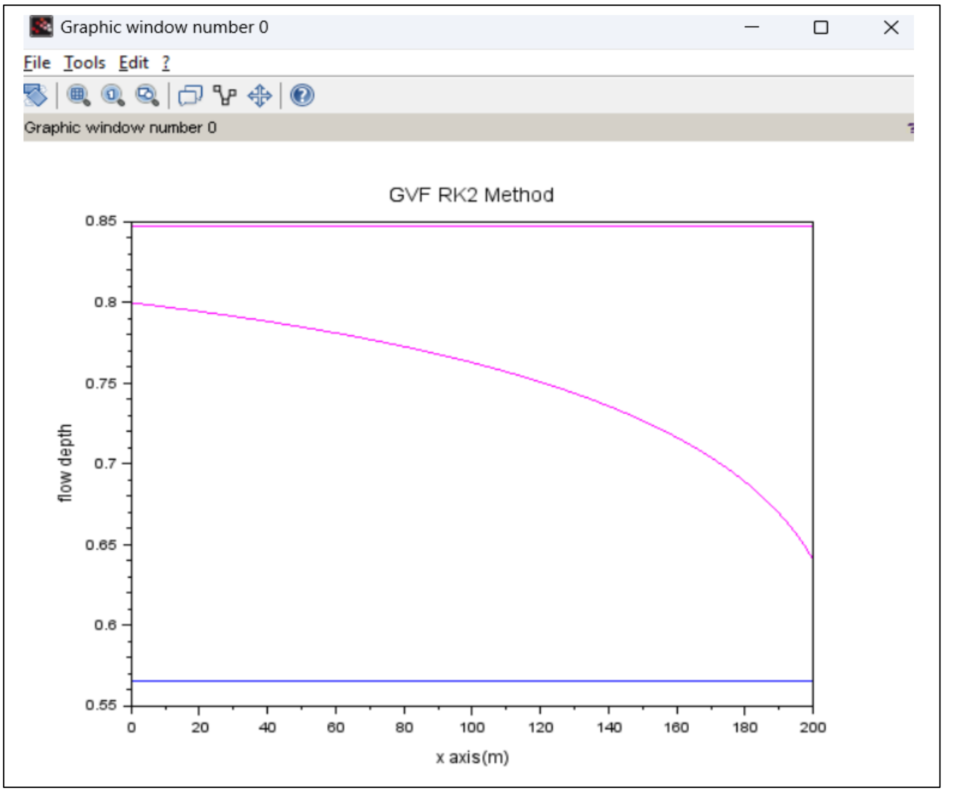
Identify the type of GVF Profile: ?
Plot of the GVF Profile.

```
//GRADUALLY VARIED FLOW- RK_2 METHOD
clc
clear all
//Data given
Q=20;
S0=0.0008;
B=15;//m, Channel width @rectangular channel
Lx=200;//length of the channel reach
y0=0.8;//initial depth at x=0
n=0.015;
g=9.81;
mnode=201;//number of nodes in the x directions
global('Q','S0','n','B','g')//global variables . for writing multiple functions later, we can directly utilize these constant values.

//**********************************************
//calculation of critical depth
yc=(Q^2/(g*B^2))^(1/3);//for rectangular channel.equation (73)
disp("yc",yc)
```

```
//*****************************************************
//calculation of normal depth using NR method [from equation (75.1)]

//To define G and G' functions
function Gval=Gfunc(y)
   Gval=(S0^(1/2)*B^(5/3)/n)*(y/(B+2*y))^(2/3)*y-Q;//eqn(74)
endfunction
function Gpval=Gderi(y)
   term1=(S0^(1/2)*B^(5/3))/(3*n);
   term2=y^(2/3)*(5*B+6*y);
   term3=(B+2*y)^(5/3);
//
Gpval=(S0^(1/2)*B^(5/3))/(3*n)*y^(2/3)*(5*B+6*y)/(B+2*y)^(5/3);//G'
from eqn (76)
Gpval=term1*term2/term3;
endfunction

//Newton-Raphson method (to solve nonlinear 'Gval' eqaution for getting
yn)
eps_max=1e-6;
aerror=1; //absolute error
yn=yc; //To start the iteration loop, critical depth is our initial guess.
while abs(aerror)>eps_max
   aerror=(Gfunc(yn)/Gderi(yn));
   yn=yn-aerror; //yn(p)=yn(p-1)-(G(yn(p-1))/G'(yn(p-1)))..eqn(75.1))
end
disp("yn",yn)
//*****************************************

//dydx function calculation from Governing Equation
function dydx=psi(x, y)  //equation (73)
   A_y=B*y;
   P_y=B+2*y;
   R_y=A_y/P_y;//Area, Perimeter and hydraulic radius are function of y
   Sf=(n^2*Q^2)/(R_y^(4/3)*A_y^2);//friction slope
   Frs=(Q^2*B)/(g*A_y^3);//Fr^2
   dydx=(S0-Sf)/(1-Frs);
endfunction
```

```
//*********************************************
//Euler-Cauchy method [to get GVF profile for nodes along x direction]
xc=linspace(0,Lx,mnode);//x coordinate vector of nodes
delta_x=Lx/(mnode-1);
yv=zeros(mnode,1);//Initialization of flow depth vector
yv(1)=y0;//Input Boundary value in yv vector
//input other entries
for i=2:mnode
   K1=delta_x*psi(xc(i-1),yv(i-1)); //K1 and K2 for ith point are calculated
from previous (i-1)th point data.(explicit)
   K2=delta_x*psi(xc(i-1)+(2/3)*(delta_x),yv(i-1)+(2/3)*K1)

   yv(i)=yv(i-1)+0.25*(K1+3*K2);//Equation (80)
end

plot(xc,yv,"-m")
set(gca(),"auto_clear","off")
plot([0 Lx],[yn yn],'-m')
set(gca(),"auto_clear","off")
plot([0 Lx],[yc yc],'-b')
xtitle("GVF RK2 Method","x axis(m)","flow depth")
```

```
"yc"

  0.5658954

 "yn"

  0.8478039

WARNING: Transposing
row vector X to get
compatible dimensions
```



(20) GVF_RK3

Problem Statement

Given

Channel Cross-Section Type: Rectangular
$y_0 = 0.8m$
$B = 15m$
$g = 9.81m/s^2$
$S_0 = 0.0008$
$n = 0.015$
$L_x = 200m$
$Q = 20m^3/s$

Required

Identify the type of GVF Profile: ?
Plot of the GVF Profile.

```
//GRADUALLY VARIED FLOW- RK_3 METHOD
clc
clear all
//Data given
Q=20;
S0=0.0008;
B=15;//m, Channel width @rectangular channel
Lx=200;//length of the channel reach
y0=0.8;//initial depth at x=0
n=0.015;
g=9.81;
mnode=201;//number of nodes in the x directions
global('Q','S0','n','B','g')//global variables . for writing multiple functions later, we can directly utilize these constant values.

//*********************************************
//calculation of critical depth
yc=(Q^2/(g*B^2))^(1/3);//for rectangular channel.equation (73)
disp("yc",yc)
```

```
//*************************************************
//calculation of normal depth using NR method [from equation (75.1)]

//To define G and G' functions
function Gval=Gfunc(y)
   Gval=(S0^(1/2)*B^(5/3)/n)*(y/(B+2*y))^(2/3)*y-Q;//eqn(74)
endfunction
function Gpval=Gderi(y)
   term1=(S0^(1/2)*B^(5/3))/(3*n);
   term2=y^(2/3)*(5*B+6*y);
   term3=(B+2*y)^(5/3);
//
Gpval=(S0^(1/2)*B^(5/3))/(3*n)*y^(2/3)*(5*B+6*y)/(B+2*y)^(5/3);/
/G' from eqn (76)
Gpval=term1*term2/term3;
endfunction

//Newton-Raphson method (to solve nonlinear 'Gval' eqaution for
getting yn)
eps_max=1e-6;
aerror=1; //absolute error
yn=yc; //To start the iteration loop, critical depth is our initial guess.
while abs(aerror)>eps_max
   aerror=(Gfunc(yn)/Gderi(yn));
   yn=yn-aerror; //yn(p)=yn(p-1)-(G(yn(p-1))/G'(yn(p-1)))..eqn(75.1))
end
disp("yn",yn)
//****************************************

//dydx function calculation from Governing Equation
function dydx=psi(x, y)  //equation (73)
   A_y=B*y;
   P_y=B+2*y;
   R_y=A_y/P_y;//Area, Perimeter and hydraulic radius are function of
y
   Sf=(n^2*Q^2)/(R_y^(4/3)*A_y^2);//friction slope
   Frs=(Q^2*B)/(g*A_y^3);//Fr^2
   dydx=(S0-Sf)/(1-Frs);
endfunction
```

```
//**********************************************
//RK3 method [to get GVF profile for nodes along x direction]
xc=linspace(0,Lx,mnode);//x coordinate vector of nodes
delta_x=Lx/(mnode-1);
yv=zeros(mnode,1);//Initialization of flow depth vector
yv(1)=y0;//Input Boundary value in yv vector
//input other entries
for i=2:mnode
   K1=delta_x*psi(xc(i-1),yv(i-1)); //K1 and K2 for ith point are calculated from previous (i-1)th point data.(explicit)
   K2=delta_x*psi(xc(i-1)+(0.5)*(delta_x),yv(i-1)+(0.5)*K1);
   K3=delta_x*psi(xc(i-1)+(delta_x),yv(i-1)-K1+2*K2);

   yv(i)=yv(i-1)+(1/6)*(K1+4*K2+K3);//Equation (81)
end

plot(xc,yv,"-b")
set(gca(),"auto_clear","off")
plot([0 Lx],[yn yn],'-m')
set(gca(),"auto_clear","off")
plot([0 Lx],[yc yc],'-b')
xtitle("GVF RK3 Method","x axis(m)","flow depth")
```

Output:

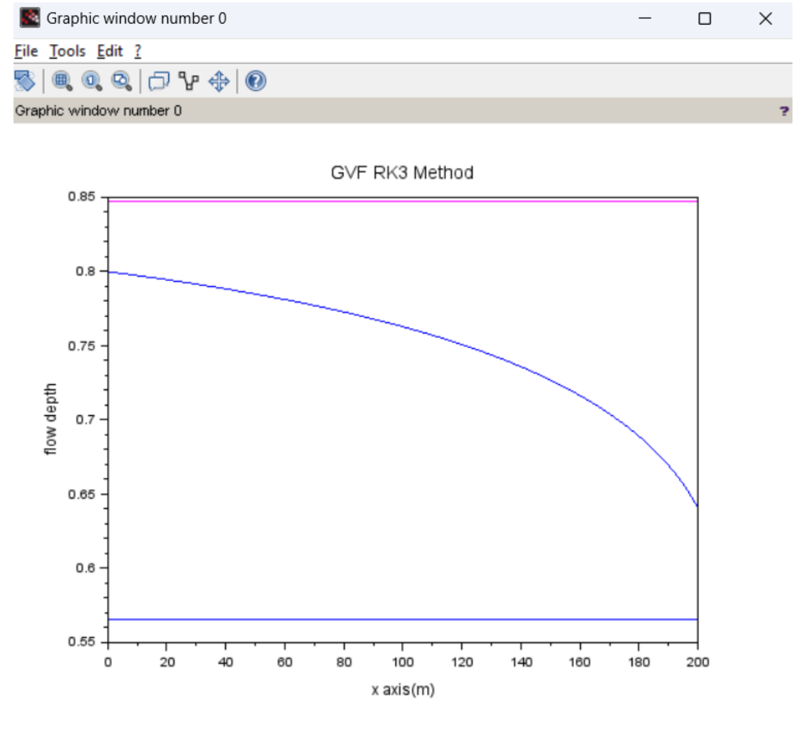
"yc"

0.5658954

"yn"

0.8478039

WARNING: Transposing row vector X to get compatible dimensions



(21) GVF_RK4

Problem Statement

Given

Channel Cross-Section Type: Rectangular
$y_0 = 0.8m$
$B = 15m$
$g = 9.81m/s^2$
$S_0 = 0.0008$
$n = 0.015$
$L_x = 200m$
$Q = 20m^3/s$

Required

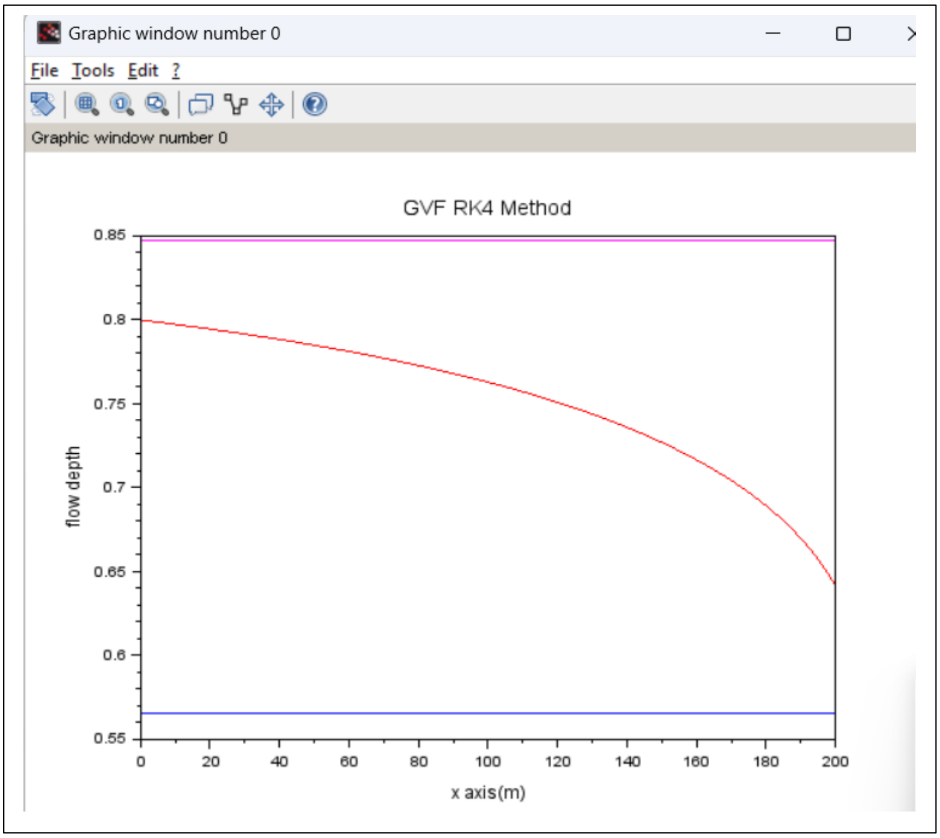
Identify the type of GVF Profile: ?
Plot of the GVF Profile.

```
//GRADUALLY VARIED FLOW- RK_4 METHOD
clc
clear all
//Data given
Q=20;
S0=0.0008;
B=15;//m, Channel width @rectangular channel
Lx=200;//length of the channel reach
y0=0.8;//initial depth at x=0
n=0.015;
g=9.81;
mnode=201;//number of nodes in the x directions
global('Q','S0','n','B','g')//global variables . for writing multiple functions later, we can directly utilize these constant values.

//*************************************************
//calculation of critical depth
yc=(Q^2/(g*B^2))^(1/3);//for rectangular channel.equation (73)
disp("yc",yc)
```

```
//**************************************************
//calculation of normal depth using NR method [from equation (75.1)]

//To define G and G' functions
function Gval=Gfunc(y)
   Gval=(S0^(1/2)*B^(5/3)/n)*(y/(B+2*y))^(2/3)*y-Q;//eqn(74)
endfunction
function Gpval=Gderi(y)
   term1=(S0^(1/2)*B^(5/3))/(3*n);
   term2=y^(2/3)*(5*B+6*y);
   term3=(B+2*y)^(5/3);
//
Gpval=(S0^(1/2)*B^(5/3))/(3*n)*y^(2/3)*(5*B+6*y)/(B+2*y)^(5/3);//G' from eqn (76)
Gpval=term1*term2/term3;
endfunction

//Newton-Raphson method (to solve nonlinear 'Gval' eqaution for getting yn)
eps_max=1e-6;
aerror=1; //absolute error
yn=yc; //To start the iteration loop, critical depth is our initial guess.
while abs(aerror)>eps_max
   aerror=(Gfunc(yn)/Gderi(yn));
   yn=yn-aerror; //yn(p)=yn(p-1)-(G(yn(p-1))/G'(yn(p-1)))..eqn(75.1))
end
disp("yn",yn)
//****************************************
//dydx function calculation from Governing Equation
function dydx=psi(x, y)  //equation (73)
   A_y=B*y;
   P_y=B+2*y;
   R_y=A_y/P_y;//Area, Perimeter and hydraulic radius are function of y
   Sf=(n^2*Q^2)/(R_y^(4/3)*A_y^2);//friction slope
   Frs=(Q^2*B)/(g*A_y^3);//Fr^2
   dydx=(S0-Sf)/(1-Frs);
endfunction
```

```
//*********************************************
//RK3 method [to get GVF profile for nodes along x direction]
xc=linspace(0,Lx,mnode);//x coordinate vector of nodes
delta_x=Lx/(mnode-1);
yv=zeros(mnode,1);//Initialization of flow depth vector
yv(1)=y0;//Input Boundary value in yv vector
//input other entries
for i=2:mnode
   K1=delta_x*psi(xc(i-1),yv(i-1)); //K1 and K2 for ith point are
calculated from previous (i-1)th point data.(explicit)
   K2=delta_x*psi(xc(i-1)+(0.5)*(delta_x),yv(i-1)+(0.5)*K1);
   K3=delta_x*psi(xc(i-1)+(0.5)*(delta_x),yv(i-1)+(0.5)*K2);
   K4=delta_x*psi(xc(i-1)+delta_x,yv(i-1)+K3);
   yv(i)=yv(i-1)+(1/6)*(K1+4*K2+K3);//Equation (82)
end

plot(xc,yv,"-r")
set(gca(),"auto_clear","off")
plot([0 Lx],[yn yn],'-m')
set(gca(),"auto_clear","off")
plot([0 Lx],[yc yc],'-b')
xtitle("GVF RK4 Method","x axis(m)","flow depth")
```

```
Output:

"yc"

  0.5658954

 "yn"

  0.8478039

WARNING: Transposing row vector X to get compatible
```



(22) GVF_IMPLICIT_RK2

Problem Statement

Given

Channel Cross-Section Type: Rectangular

$y_0 = 0.8m$

$B = 15m$

$g = 9.81m/s^2$

$S_0 = 0.0008$

$n = 0.015$

$L_x = 200m$

$Q = 20m^3/s$

Required

Identify the type of GVF Profile: ?

Plot of the GVF Profile.

```
//GRADUALLY VARIED FLOW-IMPLICIT RK2
clc
clear
//Data given
Q=20;
S0=0.0008;
B=15;//m, Channel width @rectangular channel
Lx=200;//length of the channel reach
y0=0.8;//initial depth at x=0
n=0.015;
g=9.81;
mnode=201;//number of nodes in the x directions
global('Q','S0','n','B','g')//global variables . for writing multiple functions later, we can directly utilize these constant values.

//*************************************************
//calculation of critical depth
yc=(Q^2/(g*B^2))^(1/3);//for rectangular channel.equation (73)
disp("yc",yc)
```

```
//***************************************************
//calculation of normal depth using NR method [from equation (75.1)]

//To define G and G' functions
function Gval=Gfunc(y)
   Gval=(S0^(1/2)*B^(5/3)/n)*(y/(B+2*y))^(2/3)*y-Q;//eqn(127)
endfunction
function Gpval=Gderi(y)
   term1=(S0^(1/2)*B^(5/3))/(3*n);
   term2=y^(2/3)*(5*B+6*y);
   term3=(B+2*y)^(5/3);
//
Gpval=(S0^(1/2)*B^(5/3))/(3*n)*y^(2/3)*(5*B+6*y)/(B+2*y)^(5/3);//G'
Gpval=term1*term2/term3;//from eqn (129)
endfunction

//Newton-Raphson method (to solve nonlinear 'Gval' eqaution for getting
yn)
eps_max=1e-6;
aerror=1; //absolute error
yn=yc; //To start the iteration loop, critical depth is our initial guess.
while abs(aerror)>eps_max
   aerror=(Gfunc(yn)/Gderi(yn));
   yn=yn-aerror; //yn(p)=yn(p-1)-(G(yn(p-1))/G'(yn(p-1)))..eqn(75.1))
end
disp("yn",yn)
//****************************************

//dydx function calculation from Governing Equation
function dydx=psi(x, y)  //equation (73)
   A_y=B*y;
   P_y=B+2*y;
   R_y=A_y/P_y;//Area, Perimeter and hydraulic radius are function of y
   Sf=(n^2*Q^2)/(R_y^(4/3)*A_y^2);//friction slope
   Frs=(Q^2*B)/(g*A_y^3);//Fr^2
   dydx=(S0-Sf)/(1-Frs);
endfunction
```

```
//function Fval=Ffunc(y,x,delta_x,yp)
//    Fval=y-delta_x*psi(x,y)-yp //y=new value,yp=previous known value.Equation(134)
//endfunction
//function Fpval=Fderi(y,delta_x) //derivative of Fval wrt y(new)
//    term1=(1-Q^2/(B^2*g*y^3))^(-1);
//    term2=(2*n^2*Q^2)/(B^2*y^3);
//    term3=(B*y/(B+2*y))^(-4/3);
//    term4=(4*n^2*Q^2)/(3*B^2*y^2);
//    term5=(B*y/(B+2*y))^(-7/3);
//    term6=((B/(B+2*y))-(2*B*y/(B+2*y)^2));
//    term7=(3*Q^2)/(B^2*g*y^4);
//    term8=(1-Q^2/(B^2*g*y^3))^(-2);
//    term9=S0;
//    term10=(n^2*Q^2)/(B^2*y^2);
//    term11=(B*y/(B+2*y))^(4/3);
//    Fpval=1-delta_x*(term1*(term2*term3+term4*term5*term6)-term7*term8*(term9-term10*term11));
//endfunction
/////function psipval=psideri(x,y,delta_x)
```

```
function psipval=psideri(x, y)//derivative of psival wrt y(new)
   term1=(1-Q^2/(B^2*g*y^3))^(-1);
   term2=(2*n^2*Q^2)/(B^2*y^3);
   term3=(B*y/(B+2*y))^(-4/3);
   term4=(4*n^2*Q^2)/(3*B^2*y^2);
   term5=(B*y/(B+2*y))^(-7/3);
   term6=((B/(B+2*y))-(2*B*y/(B+2*y)^2));
   term7=(3*Q^2)/(B^2*g*y^4);
   term8=(1-Q^2/(B^2*g*y^3))^(-2);
   term9=S0;
   term10=(n^2*Q^2)/(B^2*y^2);
   term11=(B*y/(B+2*y))^(4/3);
   psipval=(term1*(term2*term3+term4*term5*term6)-
term7*term8*(term9-term10*term11));//using eqn(135.1)
   endfunction
//*********************************************
//Implicit Backward Euler method [to get GVF profile for nodes along x
direction]
xc=linspace(0,Lx,mnode);//x coordinate vector of nodes
delta_x=Lx/(mnode-1);
yv=zeros(mnode,1);//Initialization of flow depth vector
yv(1)=y0;//Input Boundary value in yv vector
for i=2:mnode
   K1=delta_x*(1-0.5*delta_x*psideri(xc(i-1)+0.5*delta_x,yv(i-1)))^(-
1)*psi(xc(i-1)+0.5*delta_x,yv(i-1));//using semi-implicit eqn (138) and
putting values from (135.1) and (136.1)
   yv(i)=yv(i-1)+1*K1;
end
//plot
plot(xc,yv,"-g")
set(gca(),"auto_clear","off")
plot([0 Lx],[yn yn],'-m')//Normal depth line
set(gca(),"auto_clear","off")
plot([0 Lx],[yc yc],'-b') //Critical depth line
xtitle("GVF implicit RK2","x axis(m)","flow depth(m)")
```

Output:

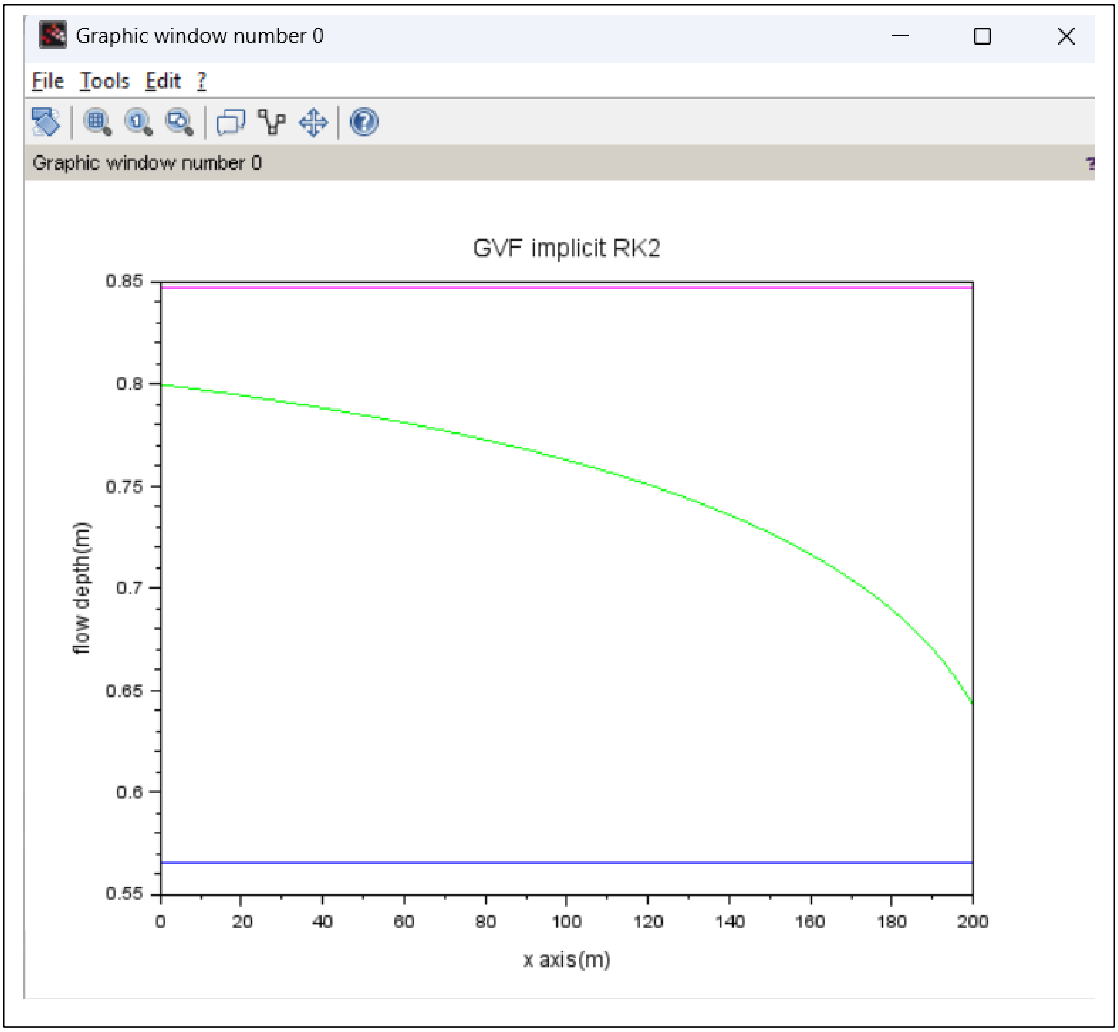
"yc"

0.5658954

"yn"

0.8478039

WARNING: Transposing row vector X to get compatible dimensions



(23) steady_1D_channel_single

//Error level: Medium. (Flow profile is not correct). Recheck! ☹

Problem Statement
Single Channel

Given

Channel Cross-Section Type: Rectangular
$B = 15m$
$g = 9.81m/s^2$
$S_0 = 0.0008$
$n = 0.015$
$L_x = 200m$
$Q = 20m^3/s$
$y_d = 0.60m$

Required

Estimate the flow depth across the channel reach.

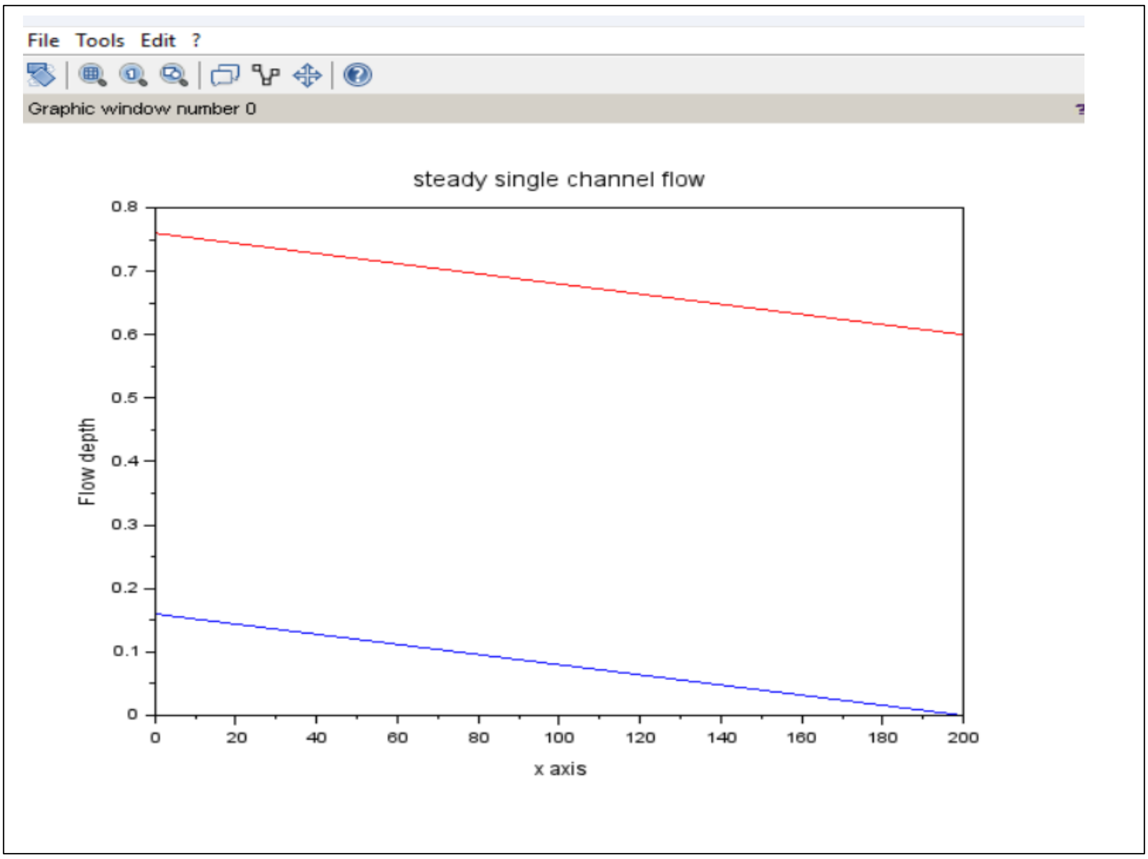
```
clc
clear
//~~~~~~given data~~~~~~~~
Q=20; //m3/s
S0=0.0008;
n=0.015;
B=15;//m
g=9.81;//m/s^2
Lx=200 //m
yd=0.6;//m
mnode=201;
//no of nodes in x direction
eps_max=1e-6;
global('Q','S0','n','B','g')

//~~~~~~~~problem dependent parameters~~~~~~
alpha=1;
xc=linspace(0,Lx,mnode);
delta_x=Lx/(mnode-1);
zv(mnode)=0;
//elevation at the end section is considered to be at datum.
```

```
for i=mnode-1:-1:1
   zv(i)=zv(i+1)+S0*delta_x
   //downsream section was considered to be the end section
which has lowest elevation. Elv. linealy increases in u/s direction.
end

yv=zeros(mnode,1);
 //It is the variable (water depth) defined for (Nl+1) number of
nodes.
C1=alpha*Q^2/(2*g);
C2=(1/2)*n^2*Q^2*delta_x;
//c1 and c2 are constants. See equation (117)

global('C1','C2','delta_x')

function Av=areav(y)
   Av=B*y;
   //Cross-section area
endfunction

function dAv=dareav(y)
   dAv=B;
   //dAv gives dA/dy.
endfunction

function Rv=HRv(y)
   Rv=B*y/(B+2*y);
   //'Rv' gives hydraulic radius.
endfunction

function dRv=dHRv(y)
   dRv=B^2/(B+2*y)^2;
   //'dRv' gives dR/dy.
endfunction
```

```
//...........................
function Mliv=Mli(y1, y2)
   Mliv=(y2-y1)-S0*delta_x+C1*(areav(y2)^(-2)-areav(y1)^(-
2))+C2*(HRv(y2)^(-4/3)*areav(y2)^(-2)+HRv(y1)^(-
4/3)*areav(y1)^(-2));
   //It is momentum function M_{l,i}."-S0*delta_x" gives the the
difference of the elevations of the two sections i.e."zv(2)-zv(1)".
See equation (117).
endfunction

function dMdyiv=dMdyi(y)
   term1=(2/areav(y)^3*dareav(y));
   term2=2*areav(y)^(-3)*HRv(y)^(-4/3)*dareav(y);
   term3=(4/3)*areav(y)^(-2)*HRv(y)^(-7/3)*dHRv(y);
   dMdyiv=1+C1*term1-C2*(term2+term3);
   //dMdyiv is nothing but dM_{l,i}/dy_{l,i}. See equation (118)
endfunction

function dMdyip1v=dMdyip1(y)
   term1=(2/areav(y)^3)*dareav(y);
   term2=2*areav^(-3)*HRv^(-4/3)*dareav(y);
   term3=(4/3)*areav(y)^(-2)*HRv^(-7/3)*dHRv(y);
   dMdyip1v=1-C1*term1-C2*(term2+term3);
   //This gives dM_{l,i}/dy_{l,i+1}. See equation(119).
endfunction
A=zeros(mnode,mnode)
//Elements of jacobian matrix should be inserted in this 'A'
matrix.
r=zeros(mnode,1);
count=0;
rmse=1;
yv=yd*ones(mnode,1);
//Downstream end value (yd) is taken as the initial guess value.
```

```
//~~~~~~~~~~~Space loop~~~~~~~~~~~~
while rmse>eps_max
   //While rmse>eps_max, then only we should iterate.
rmse=0;
for i=1:mnode-1
   A(i,i)=dMdyi(yv(i));
   A(i,i+1)=dMdyip1(yv(i+1));
   r=Mli(yv(i),yv(i+1));
end
//~~~~~Subcritical boundary conditions~~~~~
A(mnode,mnode)=1;
r(mnode)=-(yv(mnode)-yd);
//Here, 'r' vector in "[A]{dely}={r}" equation contains "negative of
momentum function"(see equation 123). From 116.1, we have d/s
boundary condition i.e., "DB_{l,Nl+1}=y_{l,Nl+1}-y_{d}". Hence, value
of 'r(mnode)' is justified.

dely=A\r;
for i=1:mnode
   yv(i)=yv(i)+dely(i);
   // Initially, yv(i)=yd; for all values of i. Then, it got modified.
   rmse=rmse+dely(i)^2;
   //Actually, using this loop, we are summing up the square of all
the errors. RMSE was calculated later.( I.T--> initial rmse should be
considered as 0. why 1?)
end

rmse=sqrt(rmse/mnode);
count=count+1;
//disp('COUNT RMSE')
//disp([count; rmse])
end

//~~~~~~figures and plots~~~~~~~~
 plot(xc',yv+zv,"-r")
 plot([0 200],[zv(1) zv(mnode)],'b-')
 xtitle("steady single channel flow","x axis","Flow depth")
```

Output: *// (It should not look like a straight line. It should be curved.)*



(24) steady_1D_channel_series

//Error level: Medium. Flow profile is not matching properly. Recheck!

Problem Statement

Channels in Series

Given

Channel Cross-Section Type: Rectangular

$B = 15m$

$g = 9.81m/s^2$

$S_{01} = 0.0004$

$S_{02} = 0.0008$

$n_1 = 0.01$

$n_2 = 0.015$

$L_{x1} = 100m$

$L_{x2} = 100m$

$Q = 20m^3/s$

$y_d = 0.60m$

Required

Estimate the flow depth across the channels in series.

```
clc
clear
//Question is at(5 no.-p-24). Lecture 37
//This problem considers constant discharge Q. flow depth 'y'
is the only variable here.
//~~~~~~Given Data~~~~~~
chl=2;
//We have two channels in series.
Q=20;//m^3/S
S0=[0.0004 0.0008];
n=[0.010 0.015]
B=15;//m
g=9.81;//m\s^2
Lx=[100 100];
yd=0.6;//m
mnode=[101 101];
eps_max=1e-6;
global('Q','B','g')
//Here, S0 and n are varying. So, we don't keep those as global
variables.
```

```
//~~~~~problem dependent parameters~~~~~~~
alpha=[1,1];
yv=zeros(sum(mnode),1);
// Here,sum(mnode)=sum of all elements in 'mnode' matrix.Therefore,
yv=zeros(202,1). Because, total number of sections=101*2=202.
for l=1:chl
//Here, l is the channel numbering. chl is number of channels(=2 here).
   delta_x(l)=Lx(l)/(mnode(l)-1);
   C1(l)=alpha(l)*Q^2/(2*g);
   C2=(1/2)*n(l)^2*Q^2*delta_x(l);
   //Here, two loops will run.  for l=1 and l=2.
end
mc=sum(mnode);
// It gives total number of sections (i.e.sum of all elements in 'mnode'
matrix). Because, we need to get those many 'depth' values.

for l=chl:-1:1
// i.e. l=2:-1:1 ,for this case.

   for i=mnode(l):-1:1
      if(l==chl & i==mnode(l))then
         zv(mc)=0;
 //Here, 'l' represents channel numbering. Number of channel is 2. So,
chl=2. Therefore, this condition means l==2 and i=mnode(2)=101; i.e.
the downstream section.
      end
      if(l<>chl & i==mnode(l)) then
// end section of a channel other than the last channel.
         mc=mc-1;
         zv(mc)=zv(mc+1)
// This 'if' loop will be executed (i.e. will take a value ,other than zero,
for 101-th row of the 'zv' vector) when the next 'if' loop will evaluate
the zv value of the 1st section of the second channel(i.e. 102-th entry of
'zv' vector).
      end
      if (i<>mnode(l)) then
         mc=mc-1;
         zv(mc)=zv(mc+1)+S0(l)*delta_x(l);
```

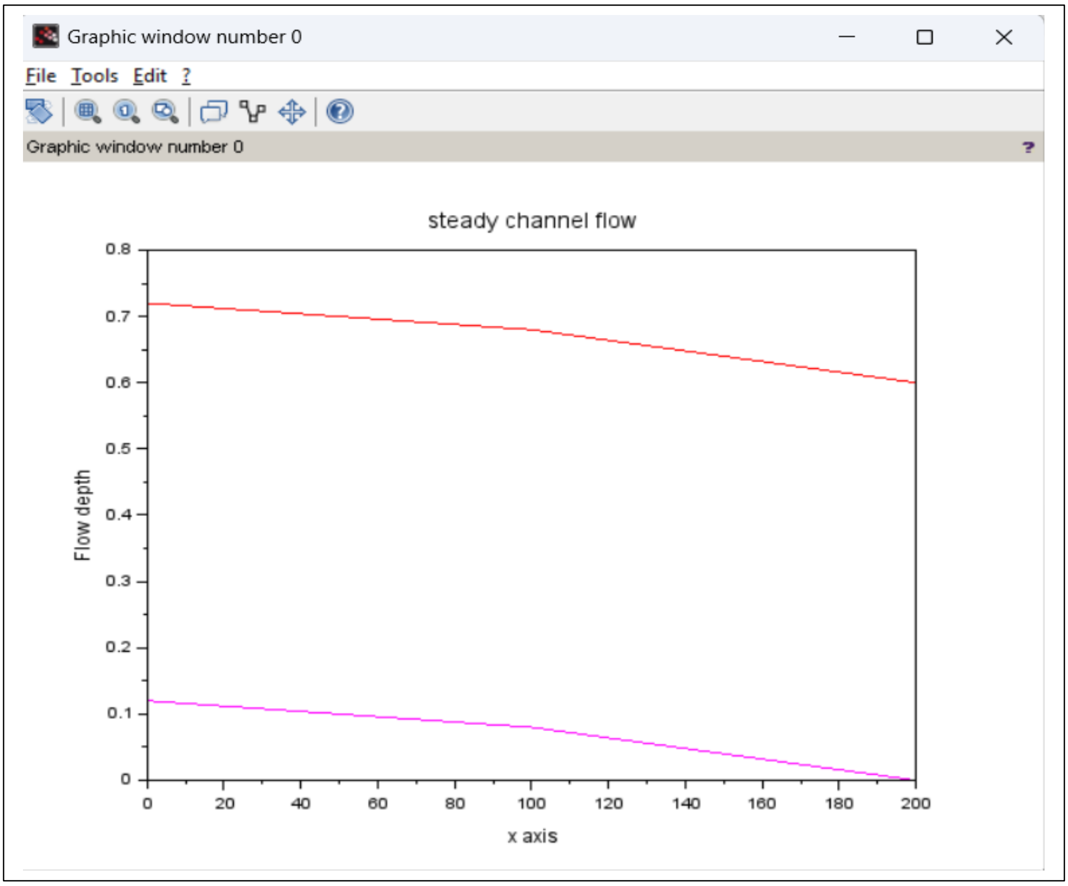
```
//This 'if' loop is for all other general setions. In first iteration, this 3rd 'if' loop will evaluate nothing. Only 1st 'if' loop will be executed for l=2 & i==mnode(2)=101-->zv=0. for 2nd loop,
//l=2,i=mnode-1=100 and mc=201; (i.e. zv(201)=zv(202)+S0(2)*delta_x(2).
//Then, l=2,i=99 and mc=200; (i.e. zv(200)=zv(201)+S0(2)*delta_x(2)...and so on.....
//When "l=2, i=1 and mc=102; (i.e. zv(102)=zv(103)+S0(2)*delta_x(2)" is done, then 2nd if loop will take non-zero value.i.e. l=1,i=101 & mc=102-1=101--> zv(101)=zv(101+1)=zv(102). Because last section of a channel and the 1st section of the very next channel will have same elevation.
      end
   end
end
xv=[linspace(0,Lx(1),mnode(1)) linspace(Lx(1),Lx(1)+Lx(2),mnode(2))]

function Av=areav(y)
   Av=B*y;
   //Cross-section area
endfunction

function dAv=dareav(y)
   dAv=B;
   //dAv gives dA/dy.
endfunction

function Rv=HRv(y)
   Rv=B*y/(B+2*y);
   //'Rv' gives hydraulic radius.
endfunction

function dRv=dHRv(y)
   dRv=B^2/(B+2*y)^2;
   //'dRv' gives dR/dy.
endfunction
```

```
//.........................
function Mliv=Mli(y1, y2, S0, delta_x, C1, C2);
//Previously, "S0,delta_x,C1,C2" were global values. But, now these
are different for different channels.
  Mliv=(y2-y1)-S0*delta_x+C1*(areav(y2)^(-2)-areav(y1)^(-
2))+C2*(HRv(y2)^(-4/3)*areav(y2)^(-2)+HRv(y1)^(-
4/3)*areav(y1)^(-2));
endfunction

function dMdyiv=dMdyi(y, C1, C2)
  term1=(2/areav(y)^3*dareav(y));
  term2=2*areav(y)^(-3)*HRv(y)^(-4/3)*dareav(y);
  term3=(4/3)*areav(y)^(-2)*HRv(y)^(-7/3)*dHRv(y);
  dMdyiv=1+C1*term1-C2*(term2+term3);
  //dMdyiv is nothing but dM_{l,i}/dy_{l,i}. See equation (118)
endfunction

function dMdyip1v=dMdyip1(y, C1, C2)
  term1=(2/areav(y)^3)*dareav(y);
  term2=2*areav(y)^(-3)*HRv^(-4/3)*dareav(y);
  term3=(4/3)*areav(y)^(-2)*HRv^(-7/3)*dHRv(y);
  dMdyip1v=1-C1*term1-C2*(term2+term3);
  //This gives dM_{l,i}/dy_{l,i+1}. See equation(119).
endfunction

A=zeros(sum(mnode),sum(mnode))
//Elements of jacobian matrix should be inserted in this 'A' matrix.No
of rows and columns in A matrix= total number of nodes/sections in
all channels.
r=zeros(sum(mnode),1);
count=0;
rmse=1;
yv=yd*ones(sum(mnode),1);
//Downstream end value (yd) is taken as the initial guess value.
```

```
//~~~~~~~~~~~Space loop~~~~~~~~~~~~
while rmse>eps_max
rmse=0;
mc=0
//Here, mc is nothing but the counter of iteration.
for l=1:chl
   for i=1:mnode(l)
      mc=mc+1
      if (l==chl & i==mnode(l)) then
         A(mc,mc)=1;
         r(mc)=-(yv(mc)-yd);
//This if loop is applicable for downstream section of the last channel.
For this 'if' loop,  mc=0+1=1. if l=2 & i=mnode(2)=101 then,
A(202,202)=1; r(202)=-(yv(1)-yd).
// This satisfy the equation "DB_{2,mnode}=yv_{l,Nl+1}-y_{d}"(see
equation 116.1).So, r(202)=-DB_{2,mnode}=-(yv(1)-yd);(See equation
123).
//Clearly, this 'if' loop will be executed at last iteration for "l=chl &
i=mnode(= 101)".
      else
         if (i==mnode(l)) then
            A(mc,mc)=1;
            A(mc,mc+1)=-1;
            r(mc)=-(yv(mc)-yv(mc+1));
//This 'if' loop is applicable for the last section of all channel(s) other
than end one.For this problem, this 'if' loop will be executed for 101-th
iteration i.e  when mc=101.
//Here, for l=1, if i==mnode(1)=101,mc=101--> A(101,101)=1,
A(101,102)=-1, r(101)=-(yv(101)-yv(102))
//This should satisfy the junction condition (Depth continuity)
"y_{1,mnode}=y_{2,1}". Therefore in equation form,"Mliv=y_{1,mnode}-
y_{2,1}". So,r(101)=-Mliv=-(y_{1,mnode}-y_{2,1}).(See equation 123).
      else
         A(mc,mc)=dMdyi(yv(mc),C1(l),C2(l));
         A(mc,mc+1)=dMdyip1(yv(mc+1),C1(l),C2(l));
         r(mc)=-Mli(yv(mc),yv(mc+1),S0(l),delta_x(l),C1(l),C2(l));
      end
   end
   end
end
```

```
dely=A\r;
for i=1:sum(mnode)
   yv(i)=yv(i)*dely(i);
   rmse=rmse+dely(i)^2;
end
rmse=sqrt(rmse/sum(mnode));
count=count+1;
disp([count rmse])
end
//Figures and Plots
plot(xv,yv+zv,"-r")
plot([Lx(1) Lx(1)+Lx(2)],[zv(mnode(1)+1)
zv(sum(mnode))],'m-')
plot([0,Lx(1)],[zv(1),zv(mnode(1))],'m-')
xtitle('steady channel flow', 'x axis','Flow depth')
```

Output: *// (It should not look like a straight line. It should be curved.)*



(25) Steady_1D_channel_network

//Error level: Medium. Flow profile is not matching properly. Recheck!

Problem Statement

Channels in Series

Given

Channel Cross-Section Type: Rectangular

$B = 15m$

$g = 9.81m/s^2$

$S_{01} = 0.0004$

$S_{02} = 0.0008$

$n_1 = 0.01$

$n_2 = 0.015$

$L_{x1} = 100m$

$L_{x2} = 100m$

$Q = 20m^3/s$

$y_d = 0.60m$

Required

Estimate the flow depth across the channels in series.

```
clc
clear
//Lecture 39. variables: y and Q Both.
//~~~~~~Given Data~~~~~~~
junc=1;
chl=2;
QI=20;//m^3/s
S0=[0.0004 0.0008];
n=[0.010 0.015];
B=15;//m
g=9.81;//m/s^2
Lx=[100 100];
yd=0.6;//m
//Depth at the downstream section.
mnode=[101 101];
eps_max=1e-3;
global('B','g')
//here two channels are there in series. So, C1, C2, S0, n values are
different for the channels. Also, Q is considered as variable, no constant
discharge for this problem.
juni=[1 2 101 1];
//I.T-->Coordinate of the junction =(channel number, node
number)=(1,101)=(2,1).
```

```
//~~~~~~~~Problem Dependent Parameters~~~~~~~~~~
alpha=[1,1];
yv=yd*ones(sum(mnode),1);
Qv=QI*ones(sum(mnode),1);
//We will initialize the problem with yd and QI values of depth and discharge.Both
are (202*1) matrices.
gv=zeros(2*sum(mnode),1);
//"gv" is the general variable with both y and Q.

//~~~~~~General Identification matrix~~~~~
 idv=0;
 for l=1:chl
    for i=1:mnode(l)
       idv=idv+1
       gid(l,i)=idv
//Previously, we used local numbering of a section i.e.(l,i).Now, this local
numberings are converted to the global numbering and stored in 'gid' matrix. Eg.
for l=chl=2, for i=mnode(2)=101, gid(2,101)=202.
    end
 end
 for l=1:chl
    for i=1:mnode(l)
    gv(2*gid(l,i)-1)=yv(gid(l,i))
    gv(2*gid(l,i))=Qv(gid(l,i))
    //Here, yv and Qv vectors store depth and dischage informations of the sections
(size 202*1 for each). Where, 'gv'(size 404*1) is the "general variable" vector which
collects data from 'yv' and 'Qv' vectors.
    //Here, equation looks like, "J*del_phi=r".Whare, J=Jacobian matrix,
   //del_phi={del_y1; del_Q1; del_y2; del_Q2;......},
   //r=[-UB_{1,1}; -C_{1,1}; -M_{1,1}; -C_{1,2}; -M_{1,2}; .........;-C_{chl,mnode(chl)}; -
M_{chl,mnode(chl)};-DB_{chl,mnode(chl)}]
    end
 end
for l=1:chl
   delta_x(l)=Lx(l)/(mnode(l)-1);
   D1(l)=alpha(l)/(2*g);
   D2(l)=(1/2)*n(l)^2*delta_x(l);
//These three are channel reach dependent parameters.
end
```

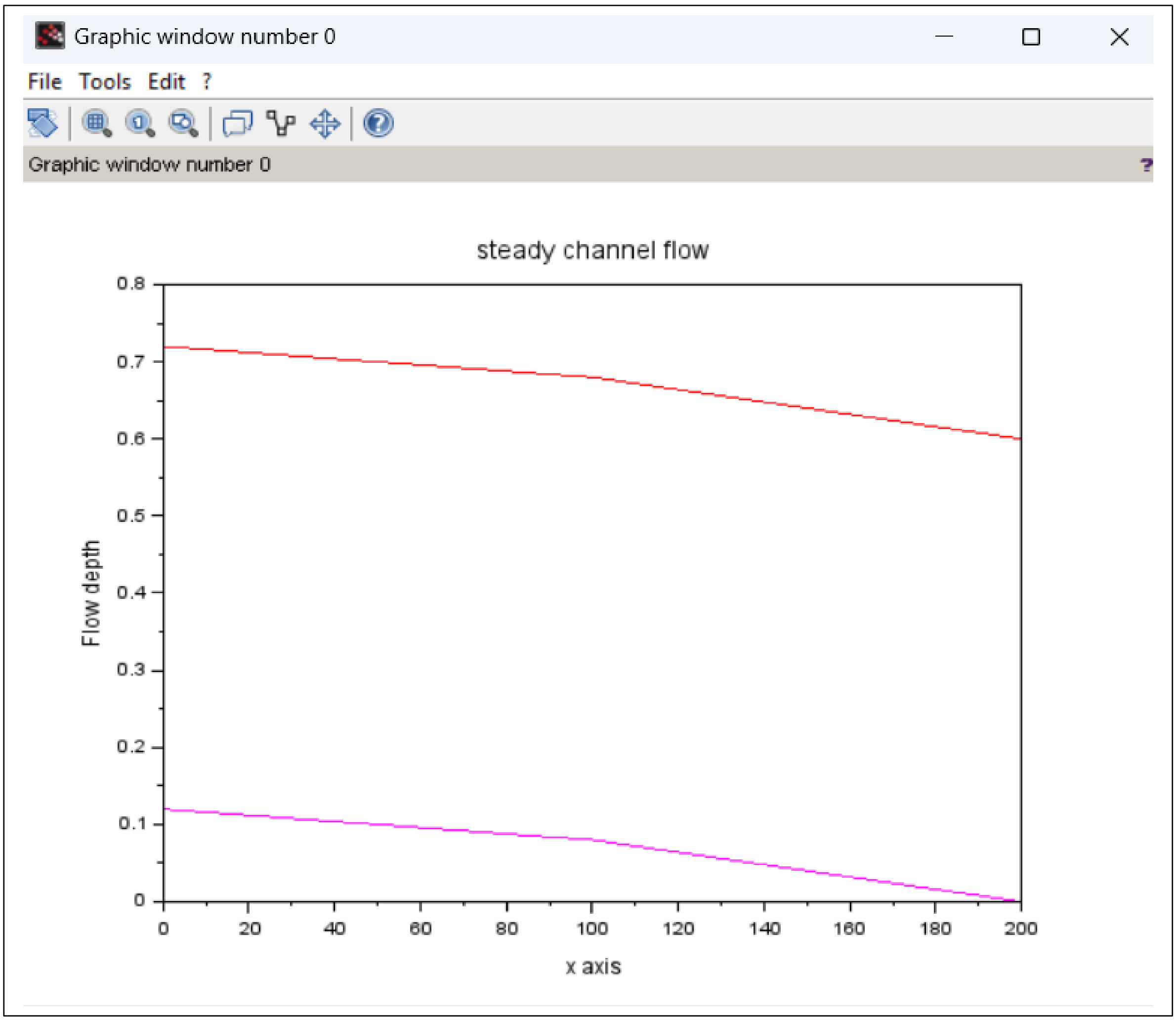
```
mc=sum(mnode);
for l=chl:-1:1
   for i=mnode(l):-1:1
      if (l==chl & i==mnode(l)) then
         zv(mc)=0;
//This 1st 'if' loop evaluates the bed elevation for end section of the last channel reach.
      end
      if(l<>chl & i==mnode(l)) then
         mc=mc-1;
         zv(mc)=zv(mc+1);
//This 'if' loop evaluates the bed elevation for end section of other channel reaches.
      end
      if(i<>mnode(l)) then
         mc=mc-1;
         zv(mc)=zv(mc+1)+S0(l)*delta_x(l);
      end
   end
end
xv=[linspace(0,Lx(1),mnode(1)) linspace(Lx(1),Lx(1)+Lx(2),mnode(2))]
function Av=areav(y)
   Av=B*y;
   //Cross-section area
endfunction

function dAv=dareav(y)
   dAv=B;
   //dAv gives dA/dy.
endfunction

function Rv=HRv(y)
   Rv=B*y/(B+2*y);
   //'Rv' gives hydraulic radius.
endfunction

function dRv=dHRv(y)
   dRv=B^2/(B+2*y)^2;
   //'dRv' gives dR/dy.
endfunction
```

```
//..........................
function Mliv=Mli(y1, Q1, y2, Q2, S0, delta_x, D1, D2)

   Mliv=(y2-y1)-S0*delta_x+D1*(Q2*abs(Q2)*areav(y2)^(-2)-
Q1*abs(Q1)*areav(y1)^(-2))+D2*(Q2*abs(Q2)*HRv(y2)^(-4/3)*areav(y2)^(-
2)+Q1*abs(Q1)*HRv(y1)^(-4/3)*areav(y1)^(-2));
 //See Equation (146).
 //Here, coefficients are D1 and D2, not C1 and C2. C1 and C2 was considered for the
code "steady_1D_channel_series", Where 'discharge Q' was not considered as a
variable.
 /////////////Why Q|Q| in Kineric head term?????
endfunction

function dMdyiv=dMdyi(y, Q, D1, D2)
   term1=(2*Q^2/areav(y)^(3))*dareav(y);
   term2=2*Q^2*areav(y)^(-3)*HRv(y)^(-4/3)*dareav(y);
   term3=(4/3)*Q^2*areav(y)^(-2)*HRv(y)^(-7/3)*dHRv(y);
   dMdyiv=-1+D1*term1-D2*(term2+term3);
   //From equation (147.1).Evaluates dM_{l,i}/dy_{l,i}.
endfunction

function dMdyip1v=dMdyi(y, Q, D1, D2)
   term1=(2*Q^2/area(y)^(3))*dareav(y);
   term2=2*Q^2*areav(y)^(-3)*HRv(y)^(-4/3)*dareav(y);
   term3=(4/3)*Q^2*areav(y)^(-2)*HRv(y)^(-7/3)*dHRv(y);
   dMdyiv=-1-D1*term1-D2*(term2+term3);
   //From equation (147.3). Evaluates dM_{l,i}/dy_{l,i+1}.
endfunction

function dMdQip1v=dMdQip1(y, Q, D1, D2)
   term1=2*Q*areav(y)^(-3);
   term2=2*Q*areav(y^(-2)*HRv(y)^(-4/3));
   dMdQip1v=D1*term1+D2*term2;
   //From equation (147.4).
endfunction

function dMdQiv=dMdQi(y, Q, D1, D2)
   term1=2*Q*areav(y)^(-3);
   term2=2*Q*areav(y^(-2)*HRv(y)^(-4/3));
   dMdQip1v=-D1*term1+D2*term2;
   //From equation (147.2).
endfunction
```

```
A=zeros(2*sum(mnode),2*sum(mnode));
r=zeros(2*sum(mnode),1);
// We have total 101*2=202 nodes in two channels. So, we have 202*2=404
variables in total (considering y & Q).
count=0;
rmse=1;

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~Space
Loop~~~~~~~~~~~~~~~~~~~~~~~~~~~
while rmse>eps_max
   rmse=0;
   eqn=1;
//Equation number.
//~~~Upstream Boundary~~~~
   A(1,2)=1;
   r(1)=-(Qv(1)-QI);
   //See (5no-P 133). Here, UB_{1,1}=Q_{1,1}-Q_u.
A(1,1)=del(UB)/del_(y_{1,1})=0. So, this satisfy "A*phi=r", means
"Jacobian*del_(y and Q)=-(C_{l,i} and M_{l,i})". See (5no-P 33).
//~~~~~~~~~~~~Equations Corresponding to segments~~~~~~~~~~
for l=1:chl
   for i=1:mnode(l)-1
// Because, we get one continuity and one momentum equation
corresponding to each segment. and for lth channel reach, number of
segments=mnode(l)-1. Here, (l,i) means the segment number which have
(l,i)th and (l,i+1)th sections.
      eqn=eqn+1;
//~~~~~~Jacobians for Continuity eqn~~~~~~~
      A(eqn,2*gid(l,i)-1)=0;
//del(C_{l,i})/del(y_{l,i}).See equation (156).
      A(eqn,2*gid(l,i))=-1;
//del(C_{l,i})/del(Q_{l,i}).
      A(eqn,2*gid(l,i+1)-1)=0;
//del(C_{l,i})/del(y_{l,i+1}).
      A(eqn,2*gid(l,i+1))=1;
//del(C_{l,i})/del(Q_{l,i+1}).
      r(eqn)=0;
// Value of each continuity functions (i.e, C_{l,i}) is zero! See equation (143) &
(155.1).
      eqn=eqn+1;
//This 'increment of counter' is for 'momentum equation' of the same
segment.
```

```
//~~~~~~~Jacobians for Momentum eqn~~~~~~~~
      A(eqn,2*gid(l,i)-1)=dMdyi(yv(gid(l,i)),Qv(gid(l,i)),D1(l),D2(l));
//del(M_{l,i})/del(y_{l,i}).See page 37 & equation (147.1).
       A(eqn,2*gid(l,i))=dMdQi(yv(gid(l,i)),Qv(gid(l,i)),D1(l),D2(l));
//del(M_{l,i})/del(Q_{l,i}).See equation (147.2).
        A(eqn,2*gid(l,i+1)-
1)=dMdyip1(yv(gid(l,i+1)),Qv(gid(l,i+1)),D1(l),D2(l));
//del(M_{l,i})/del(y_{l,i+1}).See equation (147.3).

A(eqn,2*gid(l,i+1))=dMdQip1(yv(gid(l,i+1)),Qv(gid(l,i+1)),D1(l),D2(l));
//del(M_{l,i})/del(Q_{l,i+1}).See equation (147.4).

        r(eqn)=-
Mli(yv(gid(l,i)),Qv(gid(l,i)),yv(gid(l,i+1)),Qv(gid(l,i+1)),S0(l),delta_x(l),D1(l)
,D2(l));
        end
end
// Explanation of 'equations corresponding to segments' of 'space loop':
//1st iteration (When l=1 & i=1):      Continuity part:eqn=1+1=2 (i.e. 2nd row
of Jacobian matrix). gid(1,1)=1 and gid(1,2)=2. Therefore, A(2,1)=0; A(2,2)=-
1; A(2,3)=0; A(2,4)=1.        Momentum part: eqn=2+1=3 (i.e. 3rd row of
Jacobian matrix). gid(1,1)=1. To evaluate A(3,1), A(3,2), A(3,3)and A(3,4)
equation (147)'s are used.
//2nd iteration(When l=1 & i=2): then eqn=4. gid(1,2)=2 and gid(1,3)=3.
Therefore, A(4,3)=0; A(4,4)=-1; A(4,5)=0; A(4,6)=1....and so on...

//~~~~~~~~~Junction Condition~~~~~~
//////Doubt: If junction condition executes after all segments, is the position
of junction equation not already occupied???

//~~~~Junction Continuity~~~
eqn=eqn+1;
A(eqn,2*gid(juni(1),juni(3)))=-1;
A(eqn,2*gid(juni(2),juni(4)))=1;
r(eqn)=-(Qv(gid(juni(2),juni(4)))-Qv(gid(juni(1),juni(3))));
////I.T=>[eqn=203 should be here(Because, previously 202 eqns are there i.e.
1 u/s condition and 200 equations for 1st 100 segments and 1 junction energy
condition (because, delta_y comes before delta_Q.).
//A(202,2*gid(1,101))=A(202,2*101)=A(202,202)=-1 ;
//A(202,gid(2,1))=A(202,2*102)=A(202,204)=1 and
//r(202)=-(Qv(102)-Qv(101))=-(Qv_{2,1}-Qv_{1,101})
//So, equation becomes, -delta_Qv(101)+delta_Qv(102)=-(Qv(102)-Qv(101)).
****Remember, delta_Qv(101), delta_Qv(102) these are difference of current
and previous iteration values from Newton-Raphson's method, not the
difference between two node values.]
```

```
//~~~~Junction Energy~~~
eqn=eqn+1;
A(eqn,2*gid(juni(1),juni(3))-1)=1;
A(eqn,2*gid(juni(2),juni(4))-1)=-1;
r(eqn)=-(yv(gid(juni(1),juni(3)))-yv(gid(juni(2),juni(4))));
/////I.T--> eqn=202 should be here. Because, junction energy comes before junction
continuity (i.e, delta_y comes before delta_Q). But, in code it gets more counter(as,
eqn=eqn+1 used). Is it right?????

////I.T--> [We have total 101+101=202 sections=>202*2=404 variables. We have 1
upstream discharge condition; 100+100=200 segments=>200*2=400
equations(Continuity & momentum); 2 junction conditions (Continuity &
momentum); 1 downstrem depth condition i.e. 1+400+2+1=404 equations].

//~~~~~Downstream Boundary~~~~~~~~~
eqn=eqn+1;
A(eqn,2*gid(2,mnode(2))-1)=-1;
// d/s depth boundary, coff corresponding to delta_y(202).
r(eqn)=-(yd-yv(gid(2,mnode(2))));
//So, d/s becomes, -delta_y(202)=-(yd-yv(202))

delyQ=A\r;
for i=1:2*sum(mnode)
//Because, 2*sum(mnode)=2*202=404 is total number of variables.
   gv(i)=gv(i)+delyQ(i);
//All the variables are being updated in each iteration.
   rmse=rmse+delQ(i)^2;
end

//Initial Value
for l=1:chl
   for i=1:mnode(l)
      yv(gid(l,i))=gv(2*gid(l,i)-1);
      Qv(gid(l,i))=gv(2*gid(l,i));
   end
end
   rmse=sqrt(rmse/sum(mnode));
   count=count+1;
   disp([count rmse])
end
//figure
plot(xv,yv+zv,"-r")
plot([Lx(1) Lx(1)+Lx(2)],[zv(mnode(1)+1) zv(sum(mnode))],'m-')
plot([0,Lx(1)],[zv(1),zv(mnode(1))],'m-')
xtitle('steady channel flow', 'x axis','Flow depth')
```


Graphic window number 0
File Tools Edit ?
Graphic window number 0
steady channel flow
Flow depth
x axis
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0 20 40 60 80 100 120 140 160 180 200

(26) Steady_1D_channel_network (1):

Problem Statement
Channels in Series

Given

Channel Cross-Section Type: Rectangular

$B = 15m$

$g = 9.81m/s^2$

$S_{01} = 0.0004$

$S_{02} = 0.0008$

$n_1 = 0.01$

$n_2 = 0.015$

$L_{x1} = 100m$

$L_{x2} = 100m$

$Q = 20m^3/s$

$y_d = 0.60m$

Required

Estimate the flow depth across the channels in series.

```
clc
clear
//Lecture 39. variables: y and Q Both.
//~~~~~~Given Data~~~~~~~
junc=1;
chl=2;
QI=20;//m^3/s
S0=[0.0004 0.0008];
n=[0.010 0.015];
B=15;//m
g=9.81;//m/s^2
Lx=[100 100];
yd=0.6;//m
//Depth at the downstream section.
mnode=[101 101];
eps_max=1e-6;
global('B','g')
//here two channels are there in series. So, C1,C2,S0,n values are different for the channels. Also Q is considered as variable, no constant dischargefor this problem.
juni=[1 2 101 1];
//I.T-->Coordinate of the junction =(channel number, node number)=(1,101)=(2,1).
```

```
//~~~~~~~~Problem Dependent Parameters~~~~~~~~~~
alpha=[1,1];
yv=yd*ones(sum(mnode),1);
Qv=QI*ones(sum(mnode),1);
//We will initialize the problem with yd and QI values of depth and
discharge.Both are (202*1) matrices.
gv=zeros(2*sum(mnode),1);
//"gv" is the general variable with both y and Q.

//~~~~~~General Identification matrix~~~~~
 idv=0;
 for l=1:chl
    for i=1:mnode(l)
       idv=idv+1
       gid(l,i)=idv
//Previously, we used local numbering of a section i.e.(l,i).Now, this local
numberings are converted to the global numbering and stored in 'gid'
matrix. Eg. for l=chl=2, for i=mnode(2)=101, gid(2,101)=202.
    end
 end
 for l=1:chl
    for i=1:mnode(l)
    gv(2*gid(l,i)-1)=yv(gid(l,i))
    gv(2*gid(l,i))=Qv(gid(l,i))
    //Here, yv and Qv vectors store depth and dischage informations of the
sections (size 202*1 for each). Where, 'gv'(size 404*1) is the "general
variable" vector which collects data from 'yv' and 'Qv' vectors.
    //Here, equation looks like, "J*del_phi=r".Whare, J=Jacobian matrix,
   //del_phi={del_y1; del_Q1; del_y2; del_Q2;......},
   //r=[-UB_{1,1}; -C_{1,1}; -M_{1,1}; -C_{1,2}; -M_{1,2}; .........;-
C_{chl,mnode(chl)}; -M_{chl,mnode(chl)};-DB_{chl,mnode(chl)}]
    end
 end

for l=1:chl
   delta_x(l)=Lx(l)/(mnode(l)-1);
   D1(l)=alpha(l)/(2*g);
   D2(l)=(1/2)*n(l)^2*delta_x(l);
//These three are channel reach dependent parameters.
end
```

```
mc=sum(mnode);
for l=chl:-1:1
   for i=mnode(l):-1:1
      if (l==chl & i==mnode(l)) then
         zv(mc)=0;
//This 1st 'if' loop evaluates the bed elevation for end section of the last
channel reach.
      end
      if(l<>chl & i==mnode(l)) then
         mc=mc-1;
         zv(mc)=zv(mc+1);
//This 'if' loop evaluates the bed elevation for end section of other channel
reaches.
      end
      if(i<>mnode(l)) then
         mc=mc-1;
         zv(mc)=zv(mc+1)+S0(l)*delta_x(l);
      end
   end
end
xv=[linspace(0,Lx(1),mnode(1)) linspace(Lx(1),Lx(1)+Lx(2),mnode(2))]
function Av=areav(y)
   Av=B*y;
   //Cross-section area
endfunction

function dAv=dareav(y)
   dAv=B;
   //dAv gives dA/dy.
endfunction

function Rv=HRv(y)
   Rv=B*y/(B+2*y);
   //'Rv' gives hydraulic radius.
endfunction
```

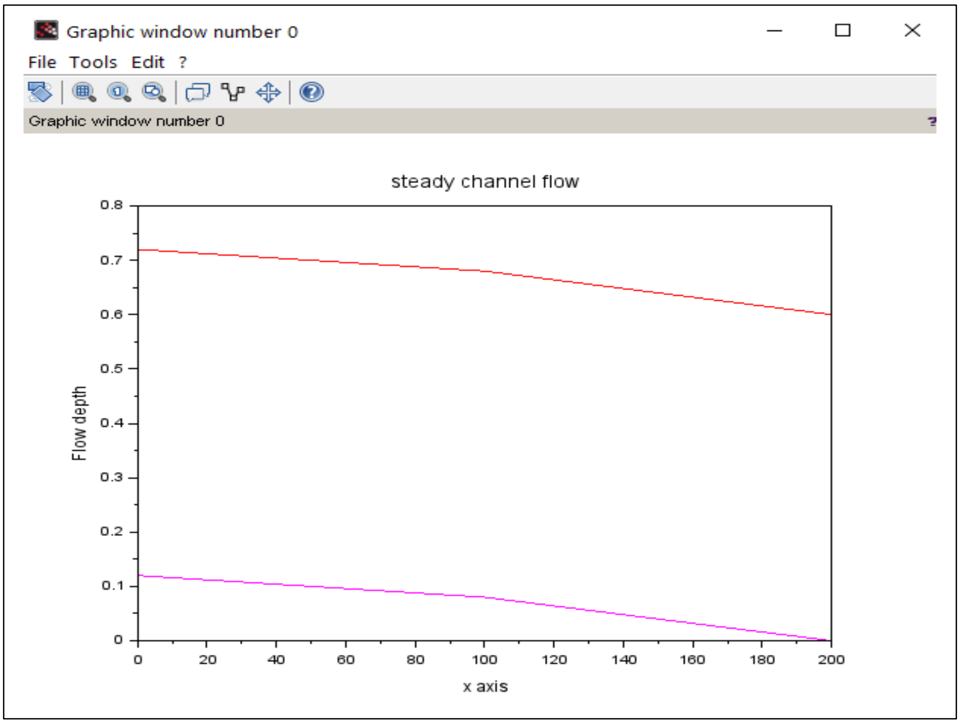
```
function dRv=dHRv(y)
   dRv=B^2/(B+2*y)^2;
   //'dRv' gives dR/dy.
endfunction

//..........................
function Mliv=Mli(y1, Q1, y2, Q2, S0, delta_x, D1, D2)

   Mliv=(y2-y1)-S0*delta_x+D1*(Q2*abs(Q2)*areav(y2)^(-2)-
Q1*abs(Q1)*areav(y1)^(-2))+D2*(Q2*abs(Q2)*HRv(y2)^(-
4/3)*areav(y2)^(-2)+Q1*abs(Q1)*HRv(y1)^(-4/3)*areav(y1)^(-2));
 //See Equation (146).
 //Here, coefficients are D1 and D2, not C1 and C2. C1 and C2 was
considered for the code "steady_1D_channel_series", Where 'discharge Q'
was not considered as a variable.
 //////////////Why Q|Q| in Kineric head term?????
endfunction

function dMdyiv=dMdyi(y, Q, D1, D2)
   term1=(2*Q^2/areav(y)^(3))*dareav(y);
   term2=2*Q^2*areav(y)^(-3)*HRv(y)^(-4/3)*dareav(y);
   term3=(4/3)*Q^2*areav(y)^(-2)*HRv(y)^(-7/3)*dHRv(y);
   dMdyiv=-1+D1*term1-D2*(term2+term3);
   //From equation (147.1).Evaluates dM_{l,i}/dy_{l,i}.
endfunction

function dMdyip1v=dMdyi(y, Q, D1, D2)
   term1=(2*Q^2/area(y)^(3))*dareav(y);
   term2=2*Q^2*areav(y)^(-3)*HRv(y)^(-4/3)*dareav(y);
   term3=(4/3)*Q^2*areav(y)^(-2)*HRv(y)^(-7/3)*dHRv(y);
   dMdyiv=-1-D1*term1-D2*(term2+term3);
   //From equation (147.3). Evaluates dM_{l,i}/dy_{l,i+1}.
endfunction

function dMdQip1v=dMdQip1(y, Q, D1, D2)
   term1=2*Q*areav(y)^(-3);
   term2=2*Q*areav(y)^(-2)*HRv(y)^(-4/3);
   dMdQip1v=D1*term1+D2*term2;
   //From equation (147.4).
endfunction
```

```
function dMdQiv=dMdQi(y, Q, D1, D2)
   term1=2*Q*areav(y)^(-3);
   term2=2*Q*areav(y)^(-2)*HRv(y)^(-4/3);
   dMdQip1v=-D1*term1+D2*term2;
   //From equation (147.2).
endfunction

A=zeros(2*sum(mnode),2*sum(mnode));
r=zeros(2*sum(mnode),1);
// We have total 101*2=202 nodes in two channels. So, we have 202*2=404
variables in total (considering y & Q).
count=0;
rmse=1;

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Space
Loop~~~~~~~~~~~~~~~~~~~~~~~~~~~~
while rmse>eps_max
   rmse=0;
   eqn=1;
//Equation number.
//~~~Upstream Boundary~~~~
   A(1,2)=1;
   r(1)=-(Qv(1)-QI);
   //See (5no-P 133). Here, UB_{1,1}=Q_{1,1}-Q_u.
A(1,1)=del(UB)/del_(y_{1,1})=0. So, this satisfy "A*phi=r", means
"Jacobian*del_(y and Q)=-(C_{l,i} and M_{l,i})". See (5no-P 33).
//~~~~~~~~~~~~Equations Corresponding to segments~~~~~~~~~~~
for l=1:chl
   for i=1:mnode(l)-1
// Because, we get one continuity and one momentum equation
corresponding to each segment. and for lth channel reach, number of
segments=mnode(l)-1. Here, (l,i) means the segment number which have
(l,i)th and (l,i+1)th sections.
      eqn=eqn+1;
```

```
//~~~~~~Jacobians for Continuity eqn~~~~~~~
      A(eqn,2*gid(l,i)-1)=0;
//del(C_{l,i})/del(y_{l,i}).See equation (156).
      A(eqn,2*gid(l,i))=-1;
//del(C_{l,i})/del(Q_{l,i}).
      A(eqn,2*gid(l,i+1)-1)=0;
//del(C_{l,i})/del(y_{l,i+1}).
      A(eqn,2*gid(l,i+1))=1;
//del(C_{l,i})/del(Q_{l,i+1}).
      r(eqn)=0;
// Value of each continuity functions (i.e, C_{l,i}) is zero! See equation (143)
& (155.1).
      eqn=eqn+1;
//This 'increment of counter' is for 'momentum equation' of the same
segment.

//~~~~~~~Jacobians for Momentum eqn~~~~~~~~
      A(eqn,2*gid(l,i)-1)=dMdyi(yv(gid(l,i)),Qv(gid(l,i)),D1(l),D2(l));
//del(M_{l,i})/del(y_{l,i}).See page 37 & equation (147.1).
       A(eqn,2*gid(l,i))=dMdQi(yv(gid(l,i)),Qv(gid(l,i)),D1(l),D2(l));
//del(M_{l,i})/del(Q_{l,i}).See equation (147.2).
       A(eqn,2*gid(l,i+1)-
1)=dMdyip1(yv(gid(l,i+1)),Qv(gid(l,i+1)),D1(l),D2(l));
//del(M_{l,i})/del(y_{l,i+1}).See equation (147.3).

A(eqn,2*gid(l,i+1))=dMdQip1(yv(gid(l,i+1)),Qv(gid(l,i+1)),D1(l),D2(l));
//del(M_{l,i})/del(Q_{l,i+1}).See equation (147.4).

       r(eqn)=-
Mli(yv(gid(l,i)),Qv(gid(l,i)),yv(gid(l,i+1)),Qv(gid(l,i+1)),S0(l),delta_x(l),D1(
l),D2(l));
       end
end
// Explanation of 'equations corresponding to segments' of 'space loop':
//1st iteration (When l=1 & i=1):      Continuity part:eqn=1+1=2 (i.e. 2nd
row of Jacobian matrix). gid(1,1)=1 and gid(1,2)=2. Therefore, A(2,1)=0;
A(2,2)=-1; A(2,3)=0; A(2,4)=1.       Momentum part: eqn=2+1=3 (i.e. 3rd row
of Jacobian matrix). gid(1,1)=1. To evaluate A(3,1), A(3,2), A(3,3)and A(3,4)
equation (147)'s are used.
//2nd iteration(When l=1 & i=2): then eqn=4. gid(1,2)=2 and gid(1,3)=3.
Therefore, A(4,3)=0; A(4,4)=-1; A(4,5)=0; A(4,6)=1....and so on...
```

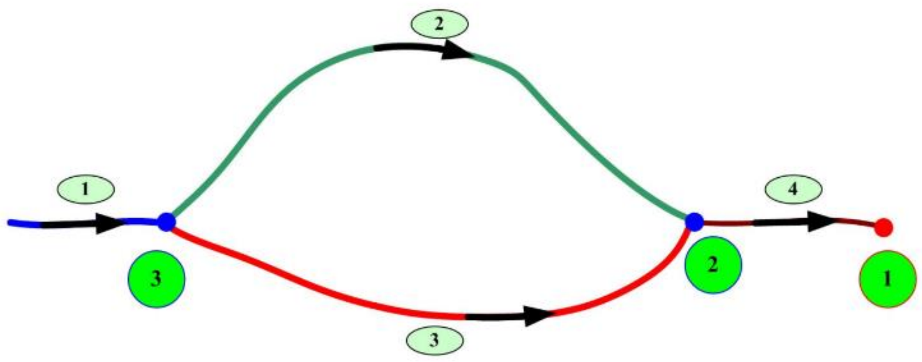
```
//~~~~~~~~~Junction Condition~~~~~~
///////Doubt: If junction condition executes after all segments, is the position of junction equation not already occupied???

//~~~~Junction Continuity~~~
eqn=eqn+1;
A(eqn,2*gid(juni(1),juni(3)))=-1;
A(eqn,2*gid(juni(2),juni(4)))=1;
r(eqn)=-(Qv(gid(juni(2),juni(4)))-Qv(gid(juni(1),juni(3))));
////I.T=>[eqn=203 should be here(Because, previously 202 eqns are there i.e. 1 u/s condition and 200 equations for 1st 100 segments and 1 junction energy condition (because, delta_y comes before delta_Q.).
//A(202,2*gid(1,101))=A(202,2*101)=A(202,202)=-1 ;
//A(202,gid(2,1))=A(202,2*102)=A(202,204)=1 and
//r(202)=-(Qv(102)-Qv(101))=-(Qv_{2,1}-Qv_{1,101})
//So, equation becomes, -delta_Qv(101)+delta_Qv(102)=-(Qv(102)-Qv(101)).
****Remember, delta_Qv(101), delta_Qv(102) these are difference of current and previous iteration values from Newton-Raphson's method, not the difference between two node values.]

//~~~~Junction Energy~~~
eqn=eqn+1;
A(eqn,2*gid(juni(1),juni(3))-1)=1;
A(eqn,2*gid(juni(2),juni(4))-1)=-1;
r(eqn)=-(yv(gid(juni(1),juni(3)))-yv(gid(juni(2),juni(4))));
/////I.T--> eqn=202 should be here. Because, junction energy comes before junction continuity (i.e, delta_y comes before delta_Q). But, in code it gets more counter(as, eqn=eqn+1 used). Is it right?????

////I.T--> [We have total 101+101=202 sections=>202*2=404 variables. We have 1 upstream discharge condition; 100+100=200 segments=>200*2=400 equations(Continuity & momentum); 2 junction conditions (Continuity & momentum); 1 downstrem depth condition i.e. 1+400+2+1=404 equations].
//~~~~~Downstream Boundary~~~~~~~~~
eqn=eqn+1;
A(eqn,2*gid(2,mnode(2))-1)=-1;
// d/s depth boundary, coff corresponding to delta_y(202).
r(eqn)=-(yd-yv(gid(2,mnode(2))));
//So, d/s becomes, -delta_y(202)=-(yd-yv(202))
```

```
delyQ=A\r;
for i=1:2*sum(mnode)
//Because, 2*sum(mnode)=2*202=404 is total number of variables.
   gv(i)=gv(i)+delyQ(i);
//All the variables are being updated in each iteration.
   rmse=rmse+delQ(i)^2;
end

//Initial Value
for l=1:chl
   for i=1:mnode(l)
      yv(gid(l,i))=gv(2*gid(l,i)-1);
      Qv(gid(l,i))=gv(2*gid(l,i));
   end
end
   rmse=sqrt(rmse/sum(mnode));
   count=count+1;
   disp([count rmse])
end
//figure
plot(xv,yv+zv,"-r")
plot([Lx(1) Lx(1)+Lx(2)],[zv(mnode(1)+1) zv(sum(mnode))],'m-')
plot([0,Lx(1)],[zv(1),zv(mnode(1))],'m-')
xtitle('steady channel flow', 'x axis','Flow depth')
```



(27) Steady_1D_channel_network_Without_reverse

Problem Statement



Problem Statement

Channel Data

Channel	length (m)	width (m)	Side Slope m_1	Side Slope m_2	reach(m)	n	S_0	Connectivity JN_1	Connectivity JN_2
1	100	50	2	2	25	0.012	0.0005	0	3
2	1500	30	2	2	75	0.0125	0.0004	3	2
3	500	20	2	2	25	0.013	0.0012	3	2
4	100	20	2	2	25	0.0135	0.0005	2	1

Junction Number	Depth (m)	Discharge (m^3/s)
1	3	-250
2	-99999	-99999
3	-99999	-99999

Required

Estimate the flow depth and discharge across the channels.

```
clc
clear
function Av=areav(y, B, m1, m2)
   Av=B*y+(1/2)*(m1+m2)*y^2;
endfunction
function dAv=dareav(y, B, m1, m2)
   dAv=B+(m1+m2)*y;
//dAv means dA/dy. See (157.1).
endfunction
function Rv=HRv(y, B, m1, m2)
   Rv=(B*y+(1/2)*(m1+m2)*y^2)/(B+(sqrt(1+m1^2)+sqrt(1+m2^2))*y);
// it is hydraulic radius. see equation (155)
endfunction
function dRv=dHRv(y, B, m1, m2)
   Tw=B+(m1+m2)*y;
//Top width. See equation (155)
   Pm=(B+(sqrt(1+m1^2)+sqrt(1+m2^2))*y);
//Wetted perimeter. See equation (155)
   Rh=HRv(y,B,m1,m2);
   dPdy=(sqrt(1+m1^2)+sqrt(1+m2^2));
//dP/dy. See equation (157.2).
   dRv=(Tw/Pm)-(Rh/Pm)*dPdy;
//dRv means dR/dy. See equation (120)for its derrivation.
endfunction
function Mliv=Mli(y1, Q1, y2, Q2, S0, delta_x, D1, D2, B, m1, m2)
   Mliv=(y2-y1)-S0*delta_x+D1*(Q2^2*areav(y2,B,m1,m2)^(-2)-
Q1^2*areav(y1,B,m1,m2)^(-2))+D2*(Q2*abs(Q2)*HRv(y2,B,m1,m2)^(-
4/3)*areav(y2,B,m1,m2)^(-2)+Q1*abs(Q1)*HRv(y1,B,m1,m2)^(-
4/3)*areav(y1,B,m1,m2)^(-2));
endfunction

function dMdyiv=dMdyi(y, Q, D1, D2, B, m1, m2)
   term1=(2*Q^2/areav(y,B,m1,m2)^(3))*dareav(y,B,m1,m2);
   term2=2*Q*abs(Q)*areav(y,B,m1,m2)^(-3)*HRv(y,B,m1,m2)^(-
4/3)*dareav(y,B,m1,m2);
   term3=(4/3)*Q*abs(Q)*areav(y,B,m1,m2)^(-2)*HRv(y,B,m1,m2)^(-
7/3)*dHRv(y,B,m1,m2);
   dMdyiv=-1+D1*term1-D2*(term2+term3);
   //From equation (147.1).Evaluates dM_{l,i}/dy_{l,i}.
endfunction
```

```
function dMdyip1v=dMdyi(y, Q, D1, D2, B, m1, m2)
   term1=(2*Q^2/area(y,B,m1,m2)^(3))*dareav(y,B,m1,m2);
   term2=2*Q*abs(Q)*areav(y,B,m1,m2)^(-3)*HRv(y,B,m1,m2)^(-
4/3)*dareav(y,B,m1,m2);
   term3=(4/3)*Q*abs(Q)*areav(y,B,m1,m2)^(-2)*HRv(y,B,m1,m2)^(-
7/3)*dHRv(y,B,m1,m2);
   dMdyiv=-1-D1*term1-D2*(term2+term3);
   //From equation (147.3). Evaluates dM_{l,i}/dy_{l,i+1}.
endfunction

function dMdQip1v=dMdQip1(y, Q, D1, D2, B, m1, m2)
   term1=2*Q*areav(y,B,m1,m2)^(-3);
   term2=2*abs(Q)*areav(y,B,m1,m2)^(-2)*HRv(y,B,m1,m2)^(-4/3);
   dMdQip1v=D1*term1+D2*term2;
   //From equation (147.4).
endfunction

function dMdQiv=dMdQi(y, Q, D1, D2, B, m1, m2)
   term1=2*Q*areav(y,B,m1,m2)^(-3);
   term2=2*abs(Q)*areav(y,B,m1,m2)^(-2)*HRv(y,B,m1,m2)^(-4/3);
   dMdQip1v=-D1*term1+D2*term2;
   //From equation (147.2).
endfunction

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//Channel reach: start + , end - .
//Flow depth condition: 1
//Flow rate (discharge) condition: 2
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~Given Data~~~~~~~~~
g=9.81; //m/s^2
global('g')
yd=3; //m
Qd=250; //m^3/s
eps_max=1e-6;
```

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
junn=3;//number of junctions
chln=4;//number of channels
//~~~~Chl | Length | Width | m1 | m2 | Segment | n | S0 | JN1 | JN2 |
chl_inf=[1  100 50 2 2 25 0.0120 0.0005 0 3
      2 1500 30 2 2 75 0.0125 0.0004 3 2
      3  500 20 2 2 25 0.0130 0.0012 3 2
      4  100 40 2 2 25 0.0135 0.0005 2 1];
jun_inf=[yd -Qd
      -99999 -99999
      -99999 -99999];
jun_con= [1 -4 0 0
       3 4 -3 -2
       3 -1 2 3];
//In 'jun_con' matrix, 1st column represents number of channels connected to
that junction.
//Positive sign means 1st section of that channel is connected to that junction.
Negative sign means (Nl+1)th section of that channel is connected to that
junction.
alpha=[1 1 1 1];

//~~~~~~~Derived informations from 'chl_inf' matrix~~~~~~~~~~
Lx=chl_inf(1:chln,2);
//We know, 'chln' means channel number=4.
B=chl_inf(1:chln,3);
m1=chl_inf(1:chln,4);
m2=chl_inf(1:chln,5);
delta_x=chl_inf(1:chln,6);
n=chl_inf(1:chln,7);
S0=chl_inf(1:chln,8);

mnode=Lx./delta_x+1;
//Here, mnode is calculated at point by point basis. Here, Lx=[100; 1500; 500;
100], delta_x=[25;75; 25; 25] results mnode=[5; 21; 21; 5].

//~~~~~~~Problem Dependent Parameters~~~~~~~
yv=yd*ones(sum(mnode),1);
Qv=Qd*ones(sum(mnode),1);
//Problem is initialized with downstream depth and discharge values.
gv=zeros(2*sum(mnode),1);
//'gv' will store the general variable with y and Q.
```

```
//~~~General identification matrix~~~
idv=0;
for l=1:chln
   for i=1:mnode(l)
      idv=idv+1;
      gid(l,i)=idv;
//Number of rows in 'gid' matrix= no. of channels=4 and number of columns= maximum no. of sections in a channel reach=max(5,21,21,5)=21. So, 'gid' is a 4*21 matrix.
   end
end
for l=1:chln
    for i=1:mnode(l)
    gv(2*gid(l,i)-1)=yv(gid(l,i))
    gv(2*gid(l,i))=Qv(gid(l,i))
//Initial Values of 'yv' and 'Qv' matries are stored in 'gv' or general variable matrix togetherly.
    end
end

for l=1:chln
   D1(l)=alpha(l)/(2*g);
   D2(l)=(1/2)*n(l)^2*delta_x(l);
end

A=zeros(2*sum(mnode),2*sum(mnode));
//'A' will store coff(s) of jacobian matrix.
r=zeros(2*sum(mnode),1);

Count=0;
rmse=1;

//~~~~~~~~~~~~Space loop~~~~~~~~~~~~~~~~~~
while rmse>eps_max
   rmse=0;
   eqn=0;//Equation number
```

```
//~~~~~Equations corresponding to
segments(2N1+2N2+2N3+2N4)~~~~~~
for l=1:chln
for i=1:mnode(l)-1
//~~~~~~Jacobians for Continuity eqn~~~~~~~
   eqn=eqn+1;
   A(eqn,2*gid(l,i)-1)=0;
//del(C_{l,i})/del(y_{l,i}).See equation (156).
   A(eqn,2*gid(l,i))=-1;
//del(C_{l,i})/del(Q_{l,i}).
   A(eqn,2*gid(l,i+1)-1)=0;
//del(C_{l,i})/del(y_{l,i+1}).
   A(eqn,2*gid(l,i+1))=1;
//del(C_{l,i})/del(Q_{l,i+1}).
   r(eqn)=0;

//~~~~~~~Jacobians for Momentum eqn~~~~~~~~
   eqn=eqn+1;
   A(eqn,2*gid(l,i)-
1)=dMdyi(yv(gid(l,i)),Qv(gid(l,i)),D1(l),D2(l),B(l),m1(l),m2(l));
//del(M_{l,i})/del(y_{l,i}).See page 37 & equation (147.1).

A(eqn,2*gid(l,i))=dMdQi(yv(gid(l,i)),Qv(gid(l,i)),D1(l),D2(l),B(l),m1(l),m2
(l));
//del(M_{l,i})/del(Q_{l,i}).See equation (147.2).
    A(eqn,2*gid(l,i+1)-
1)=dMdyip1(yv(gid(l,i+1)),Qv(gid(l,i+1)),D1(l),D2(l),B(l),m1(l),m2(l));
//del(M_{l,i})/del(y_{l,i+1}).See equation (147.3).

A(eqn,2*gid(l,i+1))=dMdQip1(yv(gid(l,i+1)),Qv(gid(l,i+1)),D1(l),D2(l),B(l
),m1(l),m2(l));
//del(M_{l,i})/del(Q_{l,i+1}).See equation (147.4).

    r(eqn)=-
Mli(yv(gid(l,i)),Qv(gid(l,i)),yv(gid(l,i+1)),Qv(gid(l,i+1)),S0(l),delta_x(l),D1
(l),D2(l),B(l),m1(l),m2(l));
    end
end
```

```
//~~~~~~~Junction Continuity Conditions~~~~~~~
for j=1:junn
   eqn=eqn+1;
   if(jun_inf(j,2)<>-99999) then
      r(eqn)=-jun_inf(j,2);
//////How??????????? I.T--> r=-DB=-(Q_{4,N4+1}+Q_d)...See Equation(159).
But here, r=-(-Q_d)=Q_d....
   else
      r(eqn)=0;
//'r(eqn)=0' -->it means, no quantity is added there.
end

//   jun_con= [1 -4 0 0
//            3 4 -3 -2
//            3 -1 2 3];
for l=1:jun_con(j,1)
//1st column gives the information about number of channel connected to
particular junction.
if (abs(jun_con(j,l+1))>eps_max) then
//Checks whether a non-zero value is there or not?!zero value means that,
no channel is connected.
   if (jun_con(j,l+1)>0) then
//Checks whether the value is positive or negative. If positive, then the 1st
section of the channel is connected to that junction. If negative, then the end
section of the channel is connected to that junction.
      jn_node=1;
      A(eqn,2*gid(abs(jun_con(j,l+1)),jn_node))=-1;
      r(eqn)=r(eqn)-Qv(gid(abs(jun_con(j,l+1)),jn_node));
   end
   if (jun_con(j,l+1)<0) then
         jn_node=mnode(abs(jun_con(j,l+1)));
         A(eqn,2*gid(abs(jun_con(j,l+1)),jn_node))=1;
         r(eqn)=r(eqn)+Qv(gid(abs(jun_con(j,l+1)),jn_node));
   end
end
end
  r(eqn)=-r(eqn);
end
```

```
//Explanation of the junction continuity condition:
/////For 1st iteration: for j=1;
// eqn=eqn+1;
//jun_inf(j,2)=jun_inf(1,2)=-250<>-99999;
// r(eqn)=-jun_inf(j,2)=-jun_inf(1,2)=-(-250)=250;
// l=1(i.e. for j=1 & l=1),
//if abs(jun_con(j,l+1))=abs(jun_con(1,2))=abs(-4)=4 > eps_max;
// if (jun_con(j,l+1)=jun_con(1,2)=-4<0)
// then
jn_node=mnode(abs(jun_con(j,l+1)))=mnode(abs(jun_con(1,2)))=mnode(4)
=5;
//A(eqn,2*gid(abs(jun_con(j,l+1)),jn_node))=A(eqn,2*gid(4,5))=A(eqn,2*52
)=A(eqn,104)=1;
//r(eqn)=r(eqn)+Qv(gid(abs(jun_con(j,l+1)),jn_node))=250+Qv(52);
//??????????????///Therefore, equation becomes,
1*del_Qv(52)=250+Qv(52). But in my opinion, from equation 159 and 160,
DBQ=Qv(52)+250 & d(DBQ)/d(Qv(52))=1. Using the relation, J*phi=-
F(phi)(See 164 and 165), 1*del_Qv(52)=-(250+Qv(52))

/////For 2nd iteration: for j=2;
// eqn=eqn+1;
//jun_inf(j,2)=jun_inf(1,2)=-99999;
// r(eqn)=0;
// l=1(i.e. for j=2 & l=1),
//if (abs(jun_con(j,l+1)))=abs(jun_con(2,2))=abs(4)=4 > eps_max;
// if (jun_con(j,l+1)=jun_con(2,2)=4>0)
// then jn_node=1;
//A(eqn,2*gid(abs(jun_con(j,l+1)),jn_node))=A(eqn,2*gid(4,1))=A(eqn,2*48
)=A(eqn,96)=-1;
//r(eqn)=r(eqn)+Qv(gid(abs(jun_con(j,l+1)),jn_node))=250+Qv(gid(abs(ju
n_con(2,2)),1))=250+Qv(gid(4,1))=250-Qv(48);
```

```
//~~~~~~~~~~~Junction Energy condition~~~~~~~~

//   jun_con= [1 -4 0 0
//            3 4 -3 -2
//            3 -1 2 3];
for j=1:junn

   if(jun_inf(j,1)<>-99999) then
      eqn=eqn+1;
      if(jun_con(j,2)>0)then jn_nodel=1; end
      if(jun_con(j,2)<0)then jn_nodel=mnode(abs(jun_con(j,2))); end
      A(eqn, 2*gid(abs(jun_con(j,2)),jn_nodel)-1)=1;
      r(eqn)=yv(gid(abs(jun_con(j,2)),jn_nodel))-jun_inf(j,1);
      r(eqn)=-r(eqn);
   end

   if(jun_con(j,2)>1) then
      for l=1:jun_con(j,1)-1
      eqn=eqn+1;
      if(jun_con(j,2)>0)then jn_nodel=1; end
      if(jun_con(j,2)<0)then jn_nodel=mnode(abs(jun_con(j,2))); end
      A(eqn, 2*gid(abs(jun_con(j,2)),jn_nodel)-1)=1;
      r(eqn)=yv(gid(abs(jun_con(j,2)),jn_nodel));

      if(jun_con(j,l+2)>0)then jn_node2=1; end
      if(jun_con(j,l+2)<0)then jn_node2=mnode(abs(jun_con(j,l+2)));
end
      A(eqn, 2*gid(abs(jun_con(j,l+2)),jn_nodel)-1)=-1;
      r(eqn)=r(eqn)-yv(gid(abs(jun_con(j,l+2)),jn_node2));
      r(eqn)=-r(eqn);
   end
end
end
//~~~~~~~~~Delta gv~~~~~~~
delyQ=A\r;
for i=1:2*sum(mnode)
   gv(i)=gv(i)+delyQ(i);
   rmse=rmse+delyQ(i)^2;
end
```

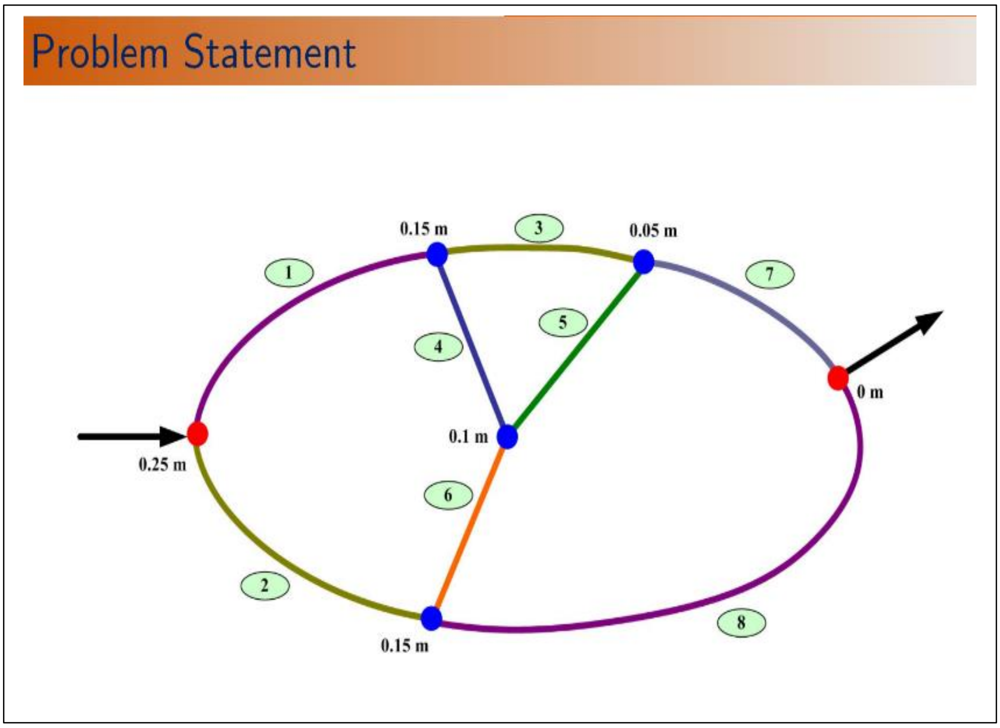
```
//~~~~~Update Value~~~~~
for l=1:chln
   for i=1:mnode(l)
      yv(gid(l,i))=gv(2*gid(l,i)-1);
      Qv(gid(l,i))=gv(2*gid(l,i));
   end
end
rmse=sqrt(rmse/sum(mnode));
count=count+1;
disp([count rmse])
end

//Print output
for l=1:chln
   disp(['channel number:' string(l)])
   disp('Section  Distance(m)  Depth(m)  Discharge(m^3/s)')
   for i=1:mnode(l)
   disp([i (i-1)*delta_x(l) yv(gid(l,i)) Qv(gid(l,i))])
end
end

////Wrong output!!!!!!!!!!!
```

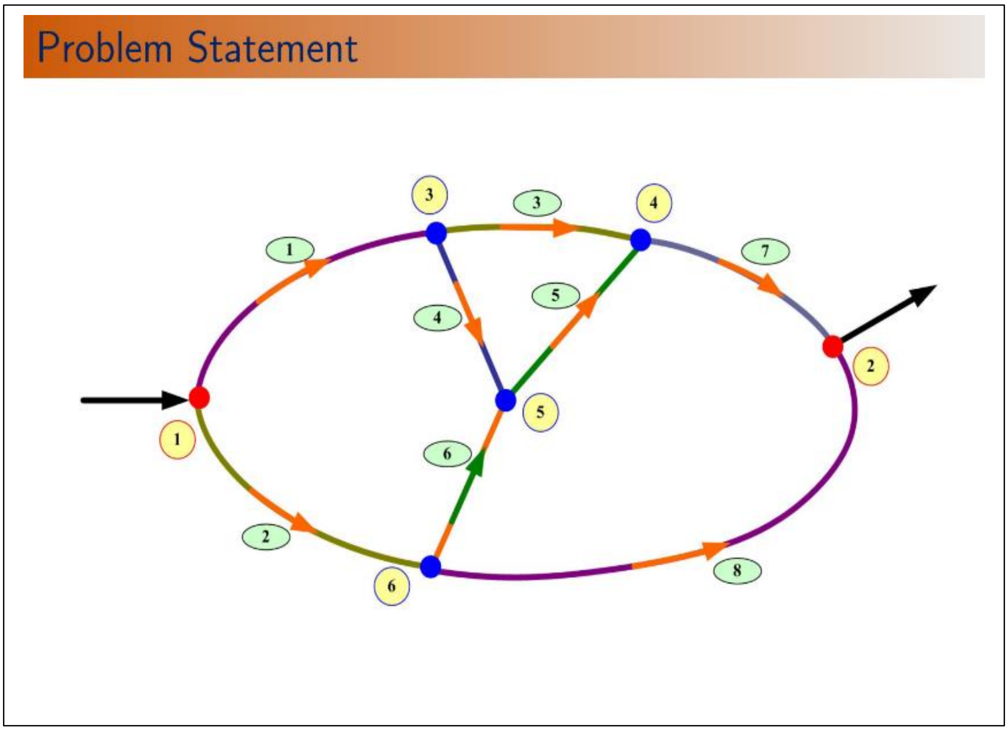
```
"channel number:"  "1"
"Section  Distance(m)  Depth(m)  Discharge(m^3/s)"
 1.   0.    3.   250.
 2.   25.   3.   250.
 3.   50.   3.   250.
 4.   75.   3.   250.
 5.   100.  3.   250.
.. . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . .
"channel number:"  "4"
"Section  Distance(m)  Depth(m)  Discharge(m^3/s)"
 1.   0.    3.   250.
 2.   25.   3.   250.
 3.   50.   3.   250.
 4.   75.   3.   250.
 5.   100.  3.   250.
```


```
"channel number:"  "1"
"Section  Distance(m)  Depth(m)  Discharge(m^3/s)"
 1.   0.    3.   250.
 2.   25.   3.   250.
 3.   50.   3.   250.
 4.   75.   3.   250.
 5.   100.  3.   250
.. . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . .
"channel number:"  "4"
"Section  Distance(m)  Depth(m)  Discharge(m^3/s)"
 1.   0.    3.   250.
 2.   25.   3.   250.
 3.   50.   3.   250.
 4.   75.   3.   250.
 5.   100.  3.   250.
```


!

(28) Steady_1D_channel_network_With_reverse





Problem Statement

Channel Data

Channel	length (m)	width (m)	Side Slope		reach(m)	n	S_0	Connectivity	
			m_1	m_2				JN_1	JN_2
1	200	30	0	0	50	0.013	0.0005	1	3
2	200	40	0	0	50	0.013	0.0005	1	6
3	200	20	0	0	50	0.012	0.0005	3	4
4	100	20	0	0	25	0.014	0.0005	3	5
5	100	20	0	0	25	0.013	0.0005	5	4
6	100	25	0	0	25	0.013	0.0005	6	5
7	100	30	0	0	25	0.014	0.0005	4	2
8	300	30	0	0	75	0.014	0.0005	6	2

Problem Statement

Junction Data

Junction Number	Depth (m)	Discharge (m^3/s)	Bed Elevation (m)
1	-99999	250	0.25
2	5	-250	0
3	-99999	-99999	0.15
4	-99999	-99999	0.05
5	-99999	-99999	0.10
6	-99999	-99999	0.15

Required

Estimate the flow depth and discharge across the channels.

```
clc
clear
function Av=areav(y, B, m1, m2)
   Av=B*y+(1/2)*(m1+m2)*y^2;
endfunction
function dAv=dareav(y, B, m1, m2)
   dAv=B+(m1+m2)*y;
//dAv means dA/dy. See (157.1).
endfunction
function Rv=HRv(y, B, m1, m2)
   Rv=(B*y+(1/2)*(m1+m2)*y^2)/(B+(sqrt(1+m1^2)+sqrt(1+m2^2))*y);
// it is hydraulic radius. see equation (155)
endfunction
function dRv=dHRv(y, B, m1, m2)
   Tw=B+(m1+m2)*y;
//Top width. See equation (155)
   Pm=(B+(sqrt(1+m1^2)+sqrt(1+m2^2))*y);
//Wetted perimeter. See equation (155)
   Rh=HRv(y,B,m1,m2);
   dPdy=(sqrt(1+m1^2)+sqrt(1+m2^2));
//dP/dy. See equation (157.2).
   dRv=(Tw/Pm)-(Rh/Pm)*dPdy;
//dRv means dR/dy. See equation (120)for its derrivation.
endfunction
function Mliv=Mli(y1, Q1, y2, Q2, S0, delta_x, D1, D2, B, m1, m2)
   Mliv=(y2-y1)-S0*delta_x+D1*(Q2^2*areav(y2,B,m1,m2)^(-2)-
Q1^2*areav(y1,B,m1,m2)^(-2))+D2*(Q2*abs(Q2)*HRv(y2,B,m1,m2)^(-
4/3)*areav(y2,B,m1,m2)^(-2)+Q1*abs(Q1)*HRv(y1,B,m1,m2)^(-
4/3)*areav(y1,B,m1,m2)^(-2));
endfunction

function dMdyiv=dMdyi(y, Q, D1, D2, B, m1, m2)
   term1=(2*Q^2/areav(y,B,m1,m2)^(3))*dareav(y,B,m1,m2);
   term2=2*Q*abs(Q)*areav(y,B,m1,m2)^(-3)*HRv(y,B,m1,m2)^(-
4/3)*dareav(y,B,m1,m2);
   term3=(4/3)*Q*abs(Q)*areav(y,B,m1,m2)^(-2)*HRv(y,B,m1,m2)^(-
7/3)*dHRv(y,B,m1,m2);
   dMdyiv=-1+D1*term1-D2*(term2+term3);
   //From equation (147.1).Evaluates dM_{l,i}/dy_{l,i}.
endfunction
```

```
function dMdyip1v=dMdyi(y, Q, D1, D2, B, m1, m2)
   term1=(2*Q^2/area(y,B,m1,m2)^(3))*dareav(y,B,m1,m2);
   term2=2*Q*abs(Q)*areav(y,B,m1,m2)^(-3)*HRv(y,B,m1,m2)^(-
4/3)*dareav(y,B,m1,m2);
   term3=(4/3)*Q*abs(Q)*areav(y,B,m1,m2)^(-
2)*HRv(y,B,m1,m2)^(-7/3)*dHRv(y,B,m1,m2);
   dMdyiv=-1-D1*term1-D2*(term2+term3);
   //From equation (147.3). Evaluates dM_{l,i}/dy_{l,i+1}.
endfunction

function dMdQip1v=dMdQip1(y, Q, D1, D2, B, m1, m2)
   term1=2*Q*areav(y,B,m1,m2)^(-3);
   term2=2*abs(Q)*areav(y,B,m1,m2)^(-2)*HRv(y,B,m1,m2)^(-
4/3);
   dMdQip1v=D1*term1+D2*term2;
   //From equation (147.4).
endfunction

function dMdQiv=dMdQi(y, Q, D1, D2, B, m1, m2)
   term1=2*Q*areav(y,B,m1,m2)^(-3);
   term2=2*abs(Q)*areav(y,B,m1,m2)^(-2)*HRv(y,B,m1,m2)^(-
4/3);
   dMdQip1v=-D1*term1+D2*term2;
   //From equation (147.2).
endfunction

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//Channel reach: start + , end - .
//Flow depth condition: 1
//Flow rate (discharge) condition: 2
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~Given Data~~~~~~~~~
g=9.81; //m/s^2
global('g')
yd=5; //m
Qd=250; //m^3/s
Qu=250; //m^3/s
eps_max=1e-6;
```

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
junn=6;//number of junctions
bjn=2;//Out of 6 junctions, 2 are boundary junctions.
chln=8;//number of channels
//~~~~Chl | Length | Width | m1 | m2 | Segment | n | S0 | JN1 | JN2
|~~~~~~~~~
chl_inf=[1  200 30 0 0 50 0.0130 0.0005 1 3
       2  200 40 0 0 50 0.0130 0.0005 1 6
       3  200 20 0 0 50 0.0120 0.0005 3 4
       4  100 20 0 0 25 0.0140 0.0005 3 5
       5  100 20 0 0 25 0.0130 0.0005 5 4
       6  100 25 0 0 25 0.0130 0.0005 6 5
       7  100 30 0 0 25 0.0140 0.0005 4 2
       8  300 50 0 0 75 0.0140 0.0005 6 2];
//~~~~~~Specified flow depth | Specified Discharge | Bed elevation~~~~~~
jun_inf=[-99999 Qu 0.25
       yd -Qd 0
       -99999 -99999 0.15
       -99999 -99999 0.05
       -99999 -99999 0.1
       -99999 -99999 0.15];
//Upstream junction is numbered as 1 & downstream junction as 2.
jun_con= [2 1 2 0
        2 -7 -8 0
        3 -1 3 4
        3 -3 -5 7
        3 -4 -6 5
        3 -2 6 8];
alpha=ones(chln,1);
//~~~~~~~Derived informations from 'chl_inf' matrix~~~~~~~~~~
Lx=chl_inf(1:chln,2);
//We know, 'chln' means channel number=8.
B=chl_inf(1:chln,3);
m1=chl_inf(1:chln,4);
m2=chl_inf(1:chln,5);
delta_x=chl_inf(1:chln,6);
n=chl_inf(1:chln,7);
S0=chl_inf(1:chln,8);

mnode=Lx./delta_x+1;
//Here, mnode is calculated at point by point basis.
```

```
//~~~~~~~~~~~z values~~~~~~~~~~~~~
for l=1:chln
   if (jun_inf(chl_inf(l,9),3)>jun_inf(chl_inf(l,10),3)) then
      fact=-1;
// 9th column of 'chl_inf' matrix represents the junction connected to the
1st section of that particular channel. 10th column represents the
'junction' connected to the last section of that particular channel. It
checks, which junction is at higher elevation between this two...
   else
      fact=+1;
end
zv(l,1)=jun_inf(chl_inf(l,9),3);
for i=2:mnode(l)
   zv(l,i)=zv(l,i-1)+fact*S0(l)*delta_x(l);
//Determines the elevations of the intermediate sections of l-th channel.
   end
end
///Explanation of the 'z values' loop:
//1st iteration:
//for l=1:chln=1:8=> Take,l=1
//    if (jun_inf(chl_inf(1,9),3)=(jun_inf(1,3))=0.25 >
jun_inf(chl_inf(1,10),3))=(jun_inf(3,3))=0.15 then
//       fact=-1;
//end
//zv(1,1)=jun_inf(chl_inf(1,9),3)=jun_inf(1,3)=0.25;
//for i=2:mnode(1)=2:5 (take i=2)
//    zv(1,2)=zv(1,i-1)+fact*S0(1)*delta_x(1)=zv(1,1)+(-
1)*S0(1)*delta_x(1)=0.25-0.0005*50=0.225;
//    end
//end
//......and so on.....

//~~~~~~~Problem Dependent Parameters~~~~~~~
yv=yd*ones(sum(mnode),1);
Qv=Qd*ones(sum(mnode),1);
//Problem is initialized with downstream depth and discharge values.
gv=zeros(2*sum(mnode),1);
//'gv' will store the general variable with y and Q.
```

```
//~~~General identification matrix~~~
idv=0;
for l=1:chln
   for i=1:mnode(l)
      idv=idv+1;
      gid(l,i)=idv;
//Number of rows in 'gid' matrix= no. of channels=8 and number of columns= maximum no. of sections in a channel reach=max(5,5,5,5,5,5,5,5)=5. So, 'gid' is a 8*5 matrix. Unlike the previous code (Steady_1D_channel_network_Without_reverse), we have no 'zero" entry in 'gid' matrix. So, we have total 40 sections.
   end
end
for l=1:chln
    for i=1:mnode(l)
    gv(2*gid(l,i)-1)=yv(gid(l,i))
    gv(2*gid(l,i))=Qv(gid(l,i))
//Initial Values of 'yv' and 'Qv' matries are stored in 'gv' or general variable matrix togetherly.
    end
end

for l=1:chln
   D1(l)=alpha(l)/(2*g);
   D2(l)=(1/2)*n(l)^2*delta_x(l);
end

A=zeros(2*sum(mnode),2*sum(mnode));
//'A' will store coff(s) of jacobian matrix.
r=zeros(2*sum(mnode),1);

Count=0;
rmse=1;
//~~~~~~~~~~~~Space loop~~~~~~~~~~~~~~~~~~
while rmse>eps_max
   rmse=0;
   eqn=0;//Equation number
```

```
//~~~~~Equations corresponding to
segments(2N1+2N2+2N3+2N4+2N5+2N6+2N7+2N8)~~~~~~
for l=1:chln
for i=1:mnode(l)-1
//~~~~~~Jacobians for Continuity eqn~~~~~~~
   eqn=eqn+1;
   A(eqn,2*gid(l,i)-1)=0;
//del(C_{l,i})/del(y_{l,i}).See equation (156).
   A(eqn,2*gid(l,i))=-1;
//del(C_{l,i})/del(Q_{l,i}).
   A(eqn,2*gid(l,i+1)-1)=0;
//del(C_{l,i})/del(y_{l,i+1}).
   A(eqn,2*gid(l,i+1))=1;
//del(C_{l,i})/del(Q_{l,i+1}).
   r(eqn)=0;

//~~~~~~~Jacobians for Momentum eqn~~~~~~~~
   eqn=eqn+1;
   A(eqn,2*gid(l,i)-
1)=dMdyi(yv(gid(l,i)),Qv(gid(l,i)),D1(l),D2(l),B(l),m1(l),m2(l));
//del(M_{l,i})/del(y_{l,i}).See page 37 & equation (147.1).

A(eqn,2*gid(l,i))=dMdQi(yv(gid(l,i)),Qv(gid(l,i)),D1(l),D2(l),B(l),m1(l),
m2(l));
//del(M_{l,i})/del(Q_{l,i}).See equation (147.2).
    A(eqn,2*gid(l,i+1)-
1)=dMdyip1(yv(gid(l,i+1)),Qv(gid(l,i+1)),D1(l),D2(l),B(l),m1(l),m2(l));
//del(M_{l,i})/del(y_{l,i+1}).See equation (147.3).

A(eqn,2*gid(l,i+1))=dMdQip1(yv(gid(l,i+1)),Qv(gid(l,i+1)),D1(l),D2(l),
B(l),m1(l),m2(l));
//del(M_{l,i})/del(Q_{l,i+1}).See equation (147.4).

    r(eqn)=-
Mli(yv(gid(l,i)),Qv(gid(l,i)),yv(gid(l,i+1)),Qv(gid(l,i+1)),S0(l),delta_x(l),
D1(l),D2(l),B(l),m1(l),m2(l));
    end
end
jeqn=0;
```

```
//~~~~~~Junction Continuity Condition~~~~~~~
for j=1:junn
   if (jun_inf(j,2)~=-99999) then
//Checks whether the junction has specified didschsrge value or not.then r(eqn)=That specified discharge value.
      eqn=eqn+1;
      jeqn=jeqn+1;
      r(eqn)=jun_inf(j,2);
   else
      if(j>bjn) then
//'bjn' means number of boundary junction and j>bjn means it is applicable for internal junctions where no specified depth or discharge condition exists.
         eqn=eqn+1;
         jeqn=jeqn+1;
         r(eqn)=0;
   end
end

for l=1:jun_con(j,1)
//1st column gives the information about number of channel connected to particular junction.
if (abs(jun_con(j,l+1))>eps_max) then
//Checks whether a non-zero value is there or not?!zero value means that, no channel is connected.
   if (jun_con(j,l+1)>0) then
//Checks whether the value is positive or negative. If positive, then the 1st section of the channel is connected to that junction. If negative, then the end section of the channel is connected to that junction.
      jn_node=1;
      A(eqn,2*gid(abs(jun_con(j,l+1)),jn_node))=-1;
      r(eqn)=r(eqn)-Qv(gid(abs(jun_con(j,l+1)),jn_node));
   end
   if (jun_con(j,l+1)<0) then
         jn_node=mnode(abs(jun_con(j,l+1)));
         A(eqn,2*gid(abs(jun_con(j,l+1)),jn_node))=1;
         r(eqn)=r(eqn)+Qv(gid(abs(jun_con(j,l+1)),jn_node));
   end
end
end
  r(eqn)=-r(eqn);
end
```

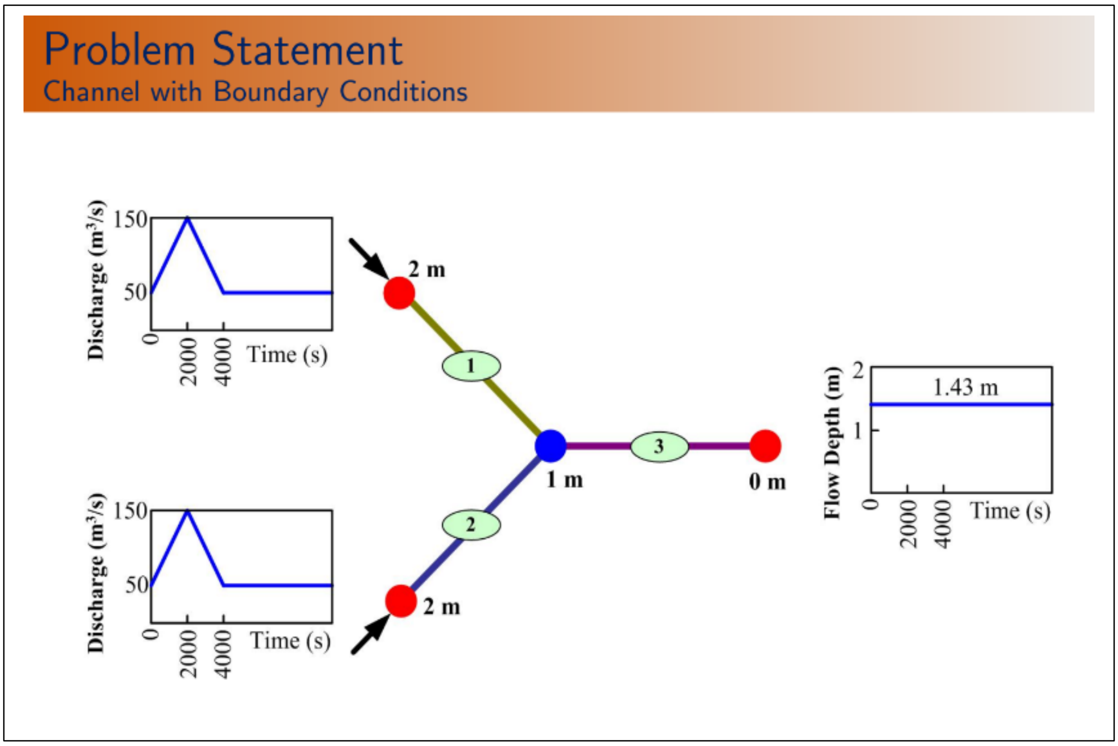
```
//~~~~~~~~~~~Junction Energy condition~~~~~~~~
//   jun_con= [1 -4 0 0
//           3 4 -3 -2
//           3 -1 2 3];
for j=1:junn

   if(jun_inf(j,1)<>-99999) then
//1st column of "jun_inf" matrix represents the specified depth at boundary
junctions (if unspecified, represented as -99999).
      eqn=eqn+1;
      if(jun_con(j,2)>0)then jn_nodel=1; end
      if(jun_con(j,2)<0)then jn_nodel=mnode(abs(jun_con(j,2))); end
      A(eqn, 2*gid(abs(jun_con(j,2)),jn_nodel)-1)=1;
      r(eqn)=yv(gid(abs(jun_con(j,2)),jn_nodel))-jun_inf(j,1);
      r(eqn)=-r(eqn);
   end

   if(jun_con(j,1)>1) then
//It means, if more than one channel is connected to that junction.
      for l=1:jun_con(j,1)-1
//It runs for (number of channels connected to that junction-1) times.
Because, one channel is already calculated for the channel represented by
"jun_con(j,2)" elements in previous loop.
      eqn=eqn+1;
//"
      if(jun_con(j,2)>0)then jn_nodel=1; end
      if(jun_con(j,2)<0)then jn_nodel=mnode(abs(jun_con(j,2))); end
      A(eqn, 2*gid(abs(jun_con(j,2)),jn_nodel)-1)=1;
      r(eqn)=yv(gid(abs(jun_con(j,2)),jn_nodel));
//"
//?????????Why "  "  part is required? I think, it has been already calculated.
      if(jun_con(j,l+2)>0)then jn_node2=1; end
// Minimum and maximum values of l are:(1 & jun_con(j,1)-1). So, it will run
for 1+2=3rd column to last non-zero column (depends on how many channels
are connected to that particular junction) of "jun_con" matrix.
      if(jun_con(j,l+2)<0)then jn_node2=mnode(abs(jun_con(j,l+2))); end
      A(eqn, 2*gid(abs(jun_con(j,l+2)),jn_node2)-1)=-1;
      r(eqn)=r(eqn)-yv(gid(abs(jun_con(j,l+2)),jn_node2));
      r(eqn)=-r(eqn);
   end
end
end
```

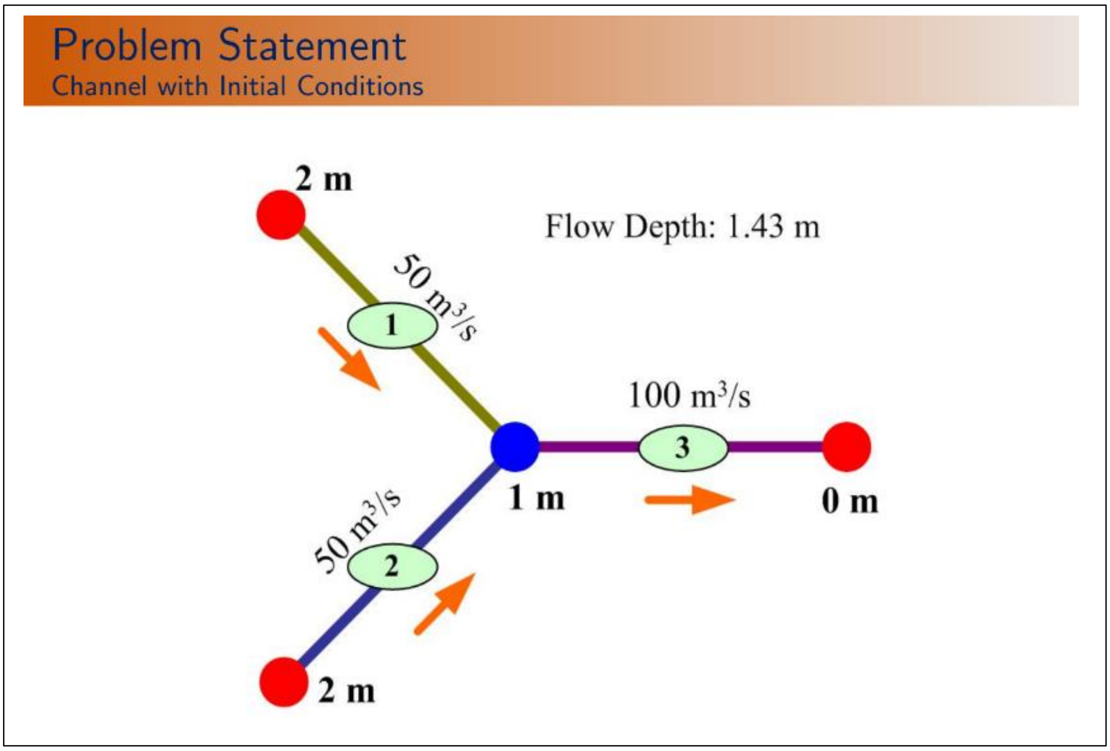
```
//~~~~Del y Q~~~
delyQ=inv(A'*A)*(A'*r);
// Here, we have 80 unknowns. But we get 81 equations. (See '5no-P-43' to know why). So we get,
//    [A]_(81*80) * [delyQ]_(80*1)=[r]_(81*1)
// But, this matrix operation is not possible.So, multiplying both  side of the equation with 'transpose of A',
//    [A']_(80*81) * [A]_(81*80) * [delyQ]_(80*1)=[A']_(80*81) * [r]_(81*1)
//    [A'A]_(80*80) * [delyQ]_(80*1)=[A'r]_(80*1)
//     [delyQ]_(80*1)= inv([A'A]_(80*80)) * [A'r]_(80*1)

for i=1:2*sum(mnode)
   gv(i)=gv(i)+delyQ(i);
   rmse=rmse+delyQ(i)^2;
end
//~~~~~Update Value~~~~~
for l=1:chln
   for i=1:mnode(l)
      yv(gid(l,i))=gv(2*gid(l,i)-1);
      Qv(gid(l,i))=gv(2*gid(l,i));
   end
end
rmse=sqrt(rmse/sum(mnode));
count=count+1;
disp([count rmse])
end
disp(eqn)
//Print output
for l=1:chln
   disp(['channel number:' string(l)])
   disp('Section  Distance(m)  Depth(m)  Discharge(m^3/s)')
   for i=1:mnode(l)
   disp([i (i-1)*delta_x(l) yv(gid(l,i)) Qv(gid(l,i))])
end
end

//Wrong Output!!!!!!!!!!!!!!
```

```
"channel number:"  "1"

 "Section  Distance(m)  Depth(m)  Discharge(m^3/s)"
  1.   0.   3.   250.
  2.   25.   3.   250.
  3.   50.   3.   250.
  4.   75.   3.   250.
  5.   100.   3.   250

  .. . . . . . . . . . . . . .

  . . . . . . . . . . . . . . .

"channel number:"  "4"
 "Section  Distance(m)  Depth(m)  Discharge(m^3/s)"
  1.   0.   3.   250.
  2.   25.   3.   250.
  3.   50.   3.   250.
  4.   75.   3.   250.
  5.   100.   3.   250.
```

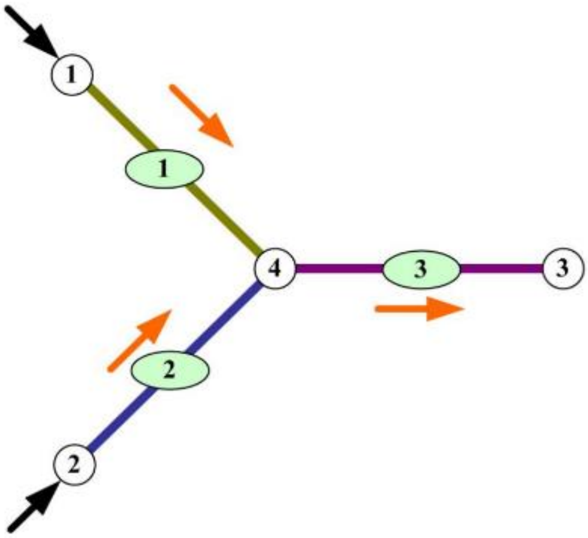
!

(29) Unsteady_1D_channel_network_With_reverse





Problem Statement
Configuration 1



Problem Statement
Channel Data (Zhang, 2005)

Channel	length (m)	width (m)	Side Slope m_1	Side Slope m_2	reach(m)	n	S_0	Connectivity JN_1	Connectivity JN_2
1	5000	50	0	0	500	0.025	0.0002	1	4
2	5000	50	0	0	500	0.025	0.0002	2	4
3	5000	100	0	0	500	0.025	0.0002	4	3

Problem Statement

Junction Data

Junction Number	Depth (m)	Discharge (m^3/s)	Bed Elevation (m)
1	-99999	2	2
2	-99999	2	2
3	1	-99999	0
4	-99999	-99999	1

Required

Plot the discharge and depth hydrographs at x = 4000 m from internal junction node in Channel reach 3 of the network.

```
clc
clear
function Av=areav(y, B, m1, m2)
   Av=B*y+(1/2)*(m1+m2)*y^2;
endfunction
function dAv=dareav(y, B, m1, m2)
   dAv=B+(m1+m2)*y;
//dAv means dA/dy. See (157.1).
endfunction
function Rv=HRv(y, B, m1, m2)

Rv=(B*y+(1/2)*(m1+m2)*y^2)/(B+(sqrt(1+m1^2)+sqrt(1+m2^2))
*y);
// it is hydraulic radius. see equation (155)
endfunction
function dRv=dHRv(y, B, m1, m2)
   Tw=B+(m1+m2)*y;
//Top width. See equation (155)
   Pm=(B+(sqrt(1+m1^2)+sqrt(1+m2^2))*y);
//Wetted perimeter. See equation (155)
   Rh=HRv(y,B,m1,m2);
   dPdy=(sqrt(1+m1^2)+sqrt(1+m2^2));
//dP/dy. See equation (157.2).
   dRv=(Tw/Pm)-(Rh/Pm)*dPdy;
//dRv means dR/dy. See equation (120)for its derrivation.
endfunction
```

```
function Cliv=Cli(y1, Q1, y2, Q2, y1o, Q1o, y2o, Q2o, zv1, zv2, theta, psi,
delta_t, delta_x, alpha1, alpha2, B, m1, m2, nm)
//"y1o,Q1o,y2o,Q2o" repesents previous time level values.
   term1=areav(y2,B,m1,m2)-areav(y2o,B,m1,m2);
   term2=areav(y1,B,m1,m2)-areav(y1o,B,m1,m2);
   term3=Q2-Q1;
   term4=Q2o-Q1o;
   Cliv=(psi/delta_t)*term1+((1-
psi)/delta_t)*term2+(theta/delta_x)*term3+((1-
theta)/delta_x)*term4;
//See equation (194). As there is no extraction or injection at any junction;
so "q" term in equation (194) was taken as zero.
endfunction

//~~~~~~~~~~Continuity Functions~~~~~~~~~~~~~~~~~~
function dCdyiv=dCdyi(y1, Q1, y2, Q2, y1o, Q1o, y2o, Q2o, zv1, zv2,
theta, psi, delta_t, delta_x, alpha1, alpha2, B, m1, m2, nm)
   dCdyiv=((1-psi)/delta_t)*dareav(y1,B,m1,m2);
   //See equation 195(.1)
endfunction

function dCdyip1v=dCdyip1(y1, Q1, y2, Q2, y1o, Q1o, y2o, Q2o, zv1,
zv2, theta, psi, delta_t, delta_x, alpha1, alpha2, B, m1, m2, nm)
   dCdyip1v=((psi)/delta_t)*dareav(y2,B,m1,m2);
//See equation 195(.3)
endfunction

function dCdQiv=dCdQi(y1, Q1, y2, Q2, y1o, Q1o, y2o, Q2o, zv1, zv2,
theta, psi, delta_t, delta_x, alpha1, alpha2, B, m1, m2, nm)
dCdQiv=-theta/delta_x;
//See equation 195(.2)
endfunction

function dCdQip1v=dCdQip1(y1, Q1, y2, Q2, y1o, Q1o, y2o, Q2o, zv1,
zv2, theta, psi, delta_t, delta_x, alpha1, alpha2, B, m1, m2, nm)
 dCdQip1v=theta/delta_x;
//See equation 195(.4)
endfunction
```

```
//~~~~~~~~~~~~~~Momentum
Functions~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
function Mliv=Mli(y1, Q1, y2, Q2, y1o, Q1o, y2o, Q2o, zv1, zv2, theta, psi,
delta_t, delta_x, alpha1, alpha2, B, m1, m2, nm)
   Av1=areav(y1,B,m1,m2);
   Av2=areav(y2,B,m1,m2);
   Av1o=areav(y1o,B,m1,m2);
   Av2o=areav(y2o,B,m1,m2);
//Av1,Av2 are corresponding to higher time level and  Av1o,Av2o are
corresponding to Previous time level.

   Rh1=HRv(y1,B,m1,m2);
   Rh2=HRv(y2,B,m1,m2);
   Rh1o=HRv(y1o,B,m1,m2);
   Rh2o=HRv(y2o,B,m1,m2);

   term11=(Q2/Av2-Q2o/Av2o);
   term11=(Q1/Av1-Q1o/Av1o);
   term21=((alpha2/2)*Q2^2*Av2^(-2))-((alpha1/2)*Q1^2*Av1^(-2));
   term22=((alpha2/2)*Q2o^2*Av2o^(-2))-((alpha1/2)*Q1o^2*Av1o^(-
2));
   term31=(y2+zv2)-(y1+zv1);
   term32=(y2o+zv2)-(y1o+zv1);
   term41=nm^2*Q2*abs(Q2)*Rh2^(-4/3)*Av2^(-2);
   term42=nm^2*Q1*abs(Q1)*Rh1^(-4/3)*Av1^(-2);
   term43=nm^2*Q2o*abs(Q2o)*Rh2o^(-4/3)*Av2o^(-2);
   term44=nm^2*Q1o*abs(Q1o)*Rh1o^(-4/3)*Av1o^(-2);

   Mliv=(psi/delta_t)*term11+((1-
psi)/delta_t)*term12+(theta/delta_x)*term21+((1-
theta)/delta_x)*term22+(theta*g/delta_x)*term31+((1-
theta)*g/delta_x)*term32+theta*psi*g*term41+theta*(1-
psi)*g*term42+(1-theta)*psi*g*term43+(1-theta)*(1-psi)*g*term44;
//See equation 196.
endfunction
```

```
function dMdyiv=dMdyi(y1, Q1, y2, Q2, y1o, Q1o, y2o, Q2o, zv1, zv2,
theta, psi, delta_t, delta_x, alpha1, alpha2, B, m1, m2, nm)
  Av1=areav(y1,B,m1,m2);
  Rh1=HRv(y1,B,m1,m2);
  dAv1=dareav(y1,B,m1,m2);
  dRh1=dHRv(y1,B,m1,m2);

  term1=(Q1/Av1^2)*dAv1;
  term2=(Q1^2/Av1^3)*dAv1;
  term3=theta*g/delta_x;
  term41=2*Q1*abs(Q1)*dAv1*Rh1^(-4/3)*Av1^(-3);
  term42=(4/3)*Q1*abs(Q1)*dRh1*Rh1^(-7/3)*Av1^(-2);
  dMdyiv=-((1-psi)/delta_t)*term1+(theta*alpha1/delta_x)*term2-
term3-theta*(1-psi)*g*nm^2*(term41+term42);
//See equation 197(.1)
endfunction

function dMdyip1v=dMdyip1(y1, Q1, y2, Q2, y1o, Q1o, y2o, Q2o, zv1,
zv2, theta, psi, delta_t, delta_x, alpha1, alpha2, B, m1, m2, nm)
  Av2=areav(y2,B,m1,m2);
  Rh2=HRv(y2,B,m1,m2);
  dAv2=dareav(y2,B,m1,m2);
  dRh2=dHRv(y2,B,m1,m2);

  term1=(Q2/Av2^2)*dAv2;
  term2=(Q2^2/Av2^3)*dAv2;
  term3=theta*g/delta_x;
  term41=2*Q2*abs(Q2)*dAv2*Rh2^(-4/3)*Av2^(-3);
  term42=(4/3)*Q2*abs(Q2)*dRh2*Rh2^(-7/3)*Av2^(-2);
  dMdyip1v=-(psi/delta_t)*term1-
(theta*alpha2/delta_x)*term2+term3-
theta*psi*g*nm^2*(term41+term42);
//See equation 197(.3)
endfunction
```

```
function dMdQiv=dMdQi(y1, Q1, y2, Q2, y1o, Q1o, y2o, Q2o, zv1, zv2, theta, psi, delta_t, delta_x, alpha1, alpha2, B, m1, m2, nm)
   Av1=areav(y1,B,m1,m2);
   Rh1=HRv(y1,B,m1,m2);
   term1=Av1^(-1);
   term2=Q1/Av1^2;
   term3=abs(Q1)*Rh1^(-4/3)*Av1^(-2);
   dMdQiv=((1-psi)/delta_t)*term1-
(theta*alpha1/delta_x)*term2+2*theta*(1-psi)*g*nm^2*term3;
//See equation 197(.2)
endfunction

function dMdQip1v=dMdQip1(y1, Q1, y2, Q2, y1o, Q1o, y2o, Q2o, zv1, zv2, theta, psi, delta_t, delta_x, alpha1, alpha2, B, m1, m2, nm)
   Av2=areav(y2,B,m1,m2);
   Rh2=HRv(y2,B,m1,m2);
   term1=Av2^(-1);
   term2=Q2/Av2^2;
   term3=abs(Q2)*Rh2^(-4/3)*Av2^(-2);

dMdQip1v=(psi/delta_t)*term1+(theta*alpha2/delta_x)*term2+2*theta*ps
i*g*nm^2*term3;
//See equation 197(.4)
endfunction

//~~~~~~~~Boundary values
Functions~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
function bndv=bndcon(typ, jnum, tv)
//'typ' means type. if typ=1, means we specify depth. If typ=2, we specify the
discharge value. "jnum" specify the junction number(See 5 no-page-49)."tv"
means time value.

if (typ==1 & jnum==3) then
   bndv=1.43;
// Depth at downstream has a fixed value.
end
```

```
if (typ==2 & jnum==1) then
   if (tv<2000) then
      bndv=50+100/2000*tv;
   end
   if (tv>=2000) then
      bndv=150-(100/2000)*(tv-2000);
   end
   if (tv>4000) then
      bndv=50;
   end
end

if (typ==2 & jnum==2) then
   if (tv<2000) then
      bndv=50+100/2000*tv;
   end
   if (tv>=2000) then
      bndv=150-(100/2000)*(tv-2000);
   end
   if (tv>4000) then
      bndv=50;
   end
end
// At junction number 1 and 2, there are varying discharge boundary
conditions with time.
endfunction


// Channel reach: Start + End -
//~~~~~~~~~~~~~~Given Data~~~~~~~~~
g=9.81;//m.s^(-2)
global('g')
eps_max=1e-6;
t_max=25001;//s
delta_t=250;//s
theta=0.5;
// To get Intermediate value between 'n' and 'n+1'th time level.
psi=0.5;
// To get Intermediate value between 'i' and 'i+1'th node
```

```
junnn=4;
bjn=3;
chln=3;
//~~~~Chl | Length | Width | m1 | m2 | Segment | n | S0 | JN1 | JN2 |
chl_inf=[1 5000  50 0 0 500 0.025 0.0002 1 4
        2 5000  50 0 0 500 0.025 0.0002 2 4
        3 5000 100 0 0 500 0.025 0.0002 4 3];
//~~~~~~Specified flow depth | Specified Discharge | Bed elevation~~~~~~
jun_inf=[-99999 2 2
        -99999 2 2
        1 -99999 0
        -99999 -99999 1];
//"1" is symbolized as specified depth."2" is symbolized as specified
discharge."-99999" means no value specified.
jun_con=[1 1 0 0
        1 2 0 0
        1 -3 0 0
        3 3 -1 -2];
//In 'jun_con' matrix, 1st column represents number of channels connected
to that junction.
//Positive sign means 1st section of that channel is connected to that
junction. Negative sign means (Nl+1)th section of that channel is
connected to that junction.
alpha=ones(chln,1);

//Derived informations
Lx=chl_inf(1:chln,2);
//We know, 'chln' means channel number=4.
B=chl_inf(1:chln,3);
m1=chl_inf(1:chln,4);
m2=chl_inf(1:chln,5);
delta_x=chl_inf(1:chln,6);
nm=chl_inf(1:chln,7);
S0=chl_inf(1:chln,8);

mnode=Lx./delta_x+1;
//mnode is total number of sections for a particular channel. Here, mnode
is calculated at point by point basis.
```

```
//~~~~~~~~z values~~~~~~~~~~
for l=1:chln
   if (jun_inf(chl_inf(l,9),3)>jun_inf(chl_inf(l,10),3)) then
      fact=-1;
// 9th column of 'chl_inf' matrix represents the junction connected to the
1st section of that particular channel. 10th column represents the
'junction' connected to the last section of that particular channel. It
checks, which junction is at higher elevation between this two...
   else
      fact=+1;
end
zv(l,1)=jun_inf(chl_inf(l,9),3);
for i=2:mnode(l)
   zv(l,i)=zv(l,i-1)+fact*S0(l)*delta_x(l);
//Determines the elevations of the intermediate sections of l-th channel.
   end
end

//~~~~~~~Problem Dependent Parameters~~~~~~~
yv=zeros(sum(mnode),1);
Qv=zeros(sum(mnode),1);
yo=zeros(sum(mnode),1);
Qo=zeros(sum(mnode),1);
// yv, Qv are values corresponding to future time level and yo, Qo are
values corresponding to old time level.

//~~~General identification matrix~~~
idv=0;
for l=1:chln
   for i=1:mnode(l)
      idv=idv+1;
      gid(l,i)=idv;
//Number of rows in 'gid' matrix= no. of channels=4 and number of
columns= maximum no. of sections in a channel
reach=max(11,11,11)=11. So, 'gid' is a 3*11 matrix.
   end
end
```

```
//~~~~~~~Initial values~~~~~~~~~
for l=1:chln
   for i=mnode(l)
      if (l==1 & l==2) then
         yo(gid(l,i))=1.4300;
         Qo(gid(l,i))=50.00;
// Initial depth and discharge values for channel 1 and 2.
      else
         yo(gid(l,i))=1.4300;
         Qo(gid(l,i))=100.00;
// Initial depth and discharge values for channel 3.
      end
   end
end

yv=yo;
Qv=Qo;
//At first, yv and Qv vector stores initial values.
gv=zeros(2*sum(mnode),1);

for l=1:chln
    for i=1:mnode(l)
    gv(2*gid(l,i)-1)=yv(gid(l,i));
    gv(2*gid(l,i))=Qv(gid(l,i));
//Initial Values of 'yv' and 'Qv' matries are stored in 'gv' or general
variable matrix togetherly.
    end
end
//~~~~~~~~~~~~~~~~~~ Time loop~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
tv=0;//zero time level.
tcount=0;
while tv<t_max
   tv=tv+delta_t;
   A=zeros(2*sum(mnode),2*sum(mnode));
//Specifying the size of the jacobian matrix.
   r=zeros(2*sum(mnode),1);
   disp(['Time in seconds:' string(tv)])
   count=0;
   rmse=1;
```

```
//~~~~~~~~Space Loop~~~~~~~~~~
while rmse>eps_max
   rmse=0;
//????? In line 286, rmse=1 and here rmse=0. What does it mean???
   eqn=0;
//Equation number.

//~~~~Equations corresponding to segments (2N1+2N2+2N3)~~~~
for l=1:chln
   for i=1:mnode(l)-1
      //~~~~~~~Continuity~~~~~~~
      eqn=eqn+1;
      A(eqn,2*gid(l,i)-
1)=dCdyi(yv(gid(l,i)),Qv(gid(l,i)),yv(gid(l,i+1)),Qv(gid(l,i+1)),yo(gid(l,i
)),Qo(gid(l,i)),yo(gid(l,i+1)),Qo(gid(l,i+1)),zv(l,i),zv(l,i+1),theta,psi,delt
a_t,delta_x(l),alpha(l),B(l),m1(l),m2(l),nm(l));
//del(C_{l,i})/del(y_{l,i}).See page 52 & equation 195(.1). Everything
written within the brackets are variables corresponding to i-th segment
of l-th channel (i.e.(l,i)).

A(eqn,2*gid(l,i))=dCdQi(yv(gid(l,i)),Qv(gid(l,i)),yv(gid(l,i+1)),Qv(gid(l,
i+1)),yo(gid(l,i)),Qo(gid(l,i)),yo(gid(l,i+1)),Qo(gid(l,i+1)),zv(l,i),zv(l,i+1
),theta,psi,delta_t,delta_x(l),alpha(l),B(l),m1(l),m2(l),nm(l));
//del(C_{l,i})/del(Q_{l,i}).See equation 195(.2).
     A(eqn,2*gid(l,i+1)-
1)=dCdyip1(yv(gid(l,i)),Qv(gid(l,i)),yv(gid(l,i+1)),Qv(gid(l,i+1)),yo(gid
(l,i)),Qo(gid(l,i)),yo(gid(l,i+1)),Qo(gid(l,i+1)),zv(l,i),zv(l,i+1),theta,psi,d
elta_t,delta_x(l),alpha(l),B(l),m1(l),m2(l),nm(l));
//del(C_{l,i})/del(y_{l,i+1}).See equation 195(.3).

A(eqn,2*gid(l,i+1))=dCdQip1(yv(gid(l,i)),Qv(gid(l,i)),yv(gid(l,i+1)),Qv(
gid(l,i+1)),yo(gid(l,i)),Qo(gid(l,i)),yo(gid(l,i+1)),Qo(gid(l,i+1)),zv(l,i),zv
(l,i+1),theta,psi,delta_t,delta_x(l),alpha(l),B(l),m1(l),m2(l),nm(l));
//del(C_{l,i})/del(Q_{l,i+1}).See equation 195(.4).

     r(eqn)=-
Cli(yv(gid(l,i)),Qv(gid(l,i)),yv(gid(l,i+1)),Qv(gid(l,i+1)),yo(gid(l,i)),Qo(
gid(l,i)),yo(gid(l,i+1)),Qo(gid(l,i+1)),zv(l,i),zv(l,i+1),theta,psi,delta_t,de
lta_x(l),alpha(l),B(l),m1(l),m2(l),nm(l));
```

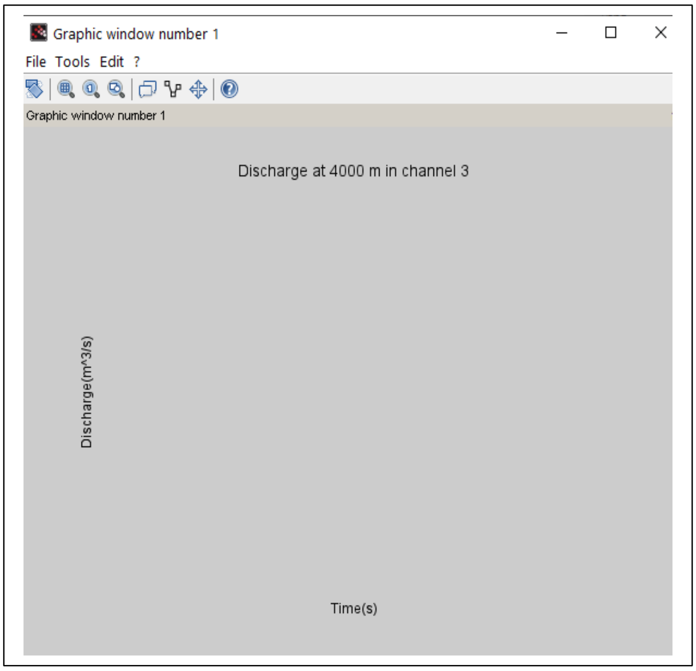
```
//~~~~~~~Momentum~~~~~~~
eqn=eqn+1;
A(eqn,2*gid(l,i)-
1)=dMdyi(yv(gid(l,i)),Qv(gid(l,i)),yv(gid(l,i+1)),Qv(gid(l,i+1)),yo(gid(l,i)),Q
o(gid(l,i)),yo(gid(l,i+1)),Qo(gid(l,i+1)),zv(l,i),zv(l,i+1),theta,psi,delta_t,delt
a_x(l),alpha(l),B(l),m1(l),m2(l),nm(l));
//del(M_{l,i})/del(y_{l,i}).See page 52 & equation 197(.1). Everything written
within the brackets are variables corresponding to i-th segment of l-th
channel (i.e.(l,i)).
A(eqn,2*gid(l,i))=dMdQi(yv(gid(l,i)),Qv(gid(l,i)),yv(gid(l,i+1)),Qv(gid(l,i+1
)),yo(gid(l,i)),Qo(gid(l,i)),yo(gid(l,i+1)),Qo(gid(l,i+1)),zv(l,i),zv(l,i+1),theta,
psi,delta_t,delta_x(l),alpha(l),B(l),m1(l),m2(l),nm(l));
//del(M_{l,i})/del(Q_{l,i}).See equation 197(.2).
A(eqn,2*gid(l,i+1)-
1)=dMdyip1(yv(gid(l,i)),Qv(gid(l,i)),yv(gid(l,i+1)),Qv(gid(l,i+1)),yo(gid(l,i)
),Qo(gid(l,i)),yo(gid(l,i+1)),Qo(gid(l,i+1)),zv(l,i),zv(l,i+1),theta,psi,delta_t,d
elta_x(l),alpha(l),B(l),m1(l),m2(l),nm(l));
//del(M_{l,i})/del(y_{l,i+1}).See equation 197(.3).
A(eqn,2*gid(l,i+1))=dMdQip1(yv(gid(l,i)),Qv(gid(l,i)),yv(gid(l,i+1)),Qv(gid
(l,i+1)),yo(gid(l,i)),Qo(gid(l,i)),yo(gid(l,i+1)),Qo(gid(l,i+1)),zv(l,i),zv(l,i+1),
theta,psi,delta_t,delta_x(l),alpha(l),B(l),m1(l),m2(l),nm(l));
//del(M_{l,i})/del(Q_{l,i+1}).See equation 197(.4).

r(eqn)=-
Mli(yv(gid(l,i)),Qv(gid(l,i)),yv(gid(l,i+1)),Qv(gid(l,i+1)),yo(gid(l,i)),Qo(gid(
l,i)),yo(gid(l,i+1)),Qo(gid(l,i+1)),zv(l,i),zv(l,i+1),theta,psi,delta_t,delta_x(l),a
lpha(l),B(l),m1(l),m2(l),nm(l));
end
end


//~~~~~~Junction Continuity condition~~~~~~~
for j=1:junn
   dcond=0; //Initially zero,=1 if discharge condition is there.
   if (jun_inf(j,2)==2) then
// At the 2nd column of jun_inf matrix, discharge values of the junctions are
shown. "2" number symbolizes discharge and -99999 represents no specified
value.
```

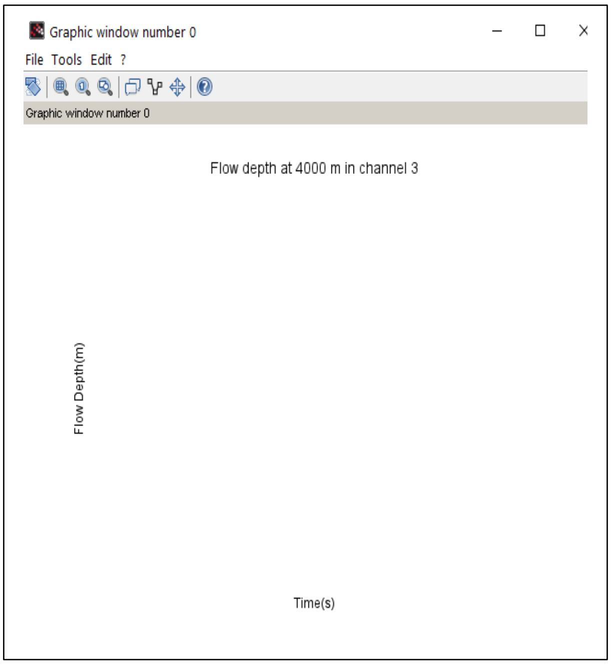
```
eqn=eqn+1;
r(eqn)=bndcon(2,j,tv);
//We know "bndv=bndcon(typ,jnum,tv)". For example bndcon(2,1,0)
means discharge boundary condition of 1st junction at time=0s.
dcond=1;
else
if(j>bjn) then
//"bjn" means number of boundary junctions(i.e=3). Those are
numbered as 1,2 and 3. So, this condition implies the internal junctions.
   eqn=eqn+1;
   r(eqn)=0;
   dcond=1;
end
end

if (dcond==1) then
for l=1:jun_con(j,1)
   if(abs(jun_con(j,l+1))>eps_max) then
      if (jun_con(j,l+1)>0) then
         jun_node=1;
         A(eqn,2*gid(abs(jun_con(j,l+1)),jn_node))=-1;
r(eqn)=r(eqn)-Qv(gid(abs(jun_con(j,l+1)),jn_node));
end
if (jun_con(j,l+1)<0) then
   jn_node=mnode(abs(jun_con(j,l+1)));
   A(eqn,2*gid(abs(jun_con(j,l+1)),jn_node))=1;
   r(eqn)=r(eqn)+Qv(gid(abs(jun_con(j,l+1)),jn_node));
end
end
end
r(eqn)=-r(eqn);
end
end
```

```
//~~~~~~~~~Junction Energy Condition~~~~~~~~~~
for j=1:junn

   if(jun_inf(j,1)==1) then
//1st column of "jun_inf" matrix represents the specified depth at
boundary junctions (if specified,represented as 1. If unspecified,
represented as -99999).
      eqn=eqn+1;
      if(jun_con(j,2)>0)then jn_nodel=1; end
      if(jun_con(j,2)<0)then jn_nodel=mnode(abs(jun_con(j,2))); end
//Above two 'if' statements check whether the junction is connected to 1st
section or the last section of the channel.
      A(eqn, 2*gid(abs(jun_con(j,2)),jn_nodel)-1)=1;
      r(eqn)=yv(gid(abs(jun_con(j,2)),jn_nodel))-bndcon(1,j,tv);
      r(eqn)=-r(eqn);
   end

//Explanation of the above loop:
//Let us start with j=3. Because Only for that We have specified depth
condition.
//for j=3;
//if(jun_inf(j,1)== jun_inf(3,1)==1) then
//      eqn=eqn+1;
//      if(jun_con(j,2)=jun_con(3,2)=-3<0)
//          then
jn_nodel=mnode(abs(jun_con(j,2)))=mnode(abs(jun_con(3,2)))=mnode(ab
s(-3))=mnode(3)=11;
//       end
//       A(eqn, 2*gid(abs(jun_con(j,2)),jn_nodel)-1)=A(eqn,
2*gid(abs(jun_con(3,2)),jn_nodel)-1)=A(eqn, 2*gid(3,11)-1)=A(eqn,2*33-
1)=A(eqn,65)=1;
//        r(eqn)=yv(gid(abs(jun_con(j,2)),jn_nodel))-
bndcon(1,j,tv)=yv(gid(abs(jun_con(3,2)),jn_nodel))-
bndcon(1,j,tv)=yv(gid(3,11))-bnd(1,3,0)=yv(33)-1.43;
//         r(eqn)=-r(eqn)=1.43-yv(33)
//So, as per this loop, d/s equation becomes, 1.delta_y(33)=1.43-yv(33).

//?????????? But in my opinion,Mli=yd-y(33), dMdyi=-1. So, acoording to
the rule dMdyi*delta_y(33)=-Mli => (-1).delta_y(33)=-(yd-y(33)) =>
"delta_y(33)=y(33)-yd".    this is different!
```

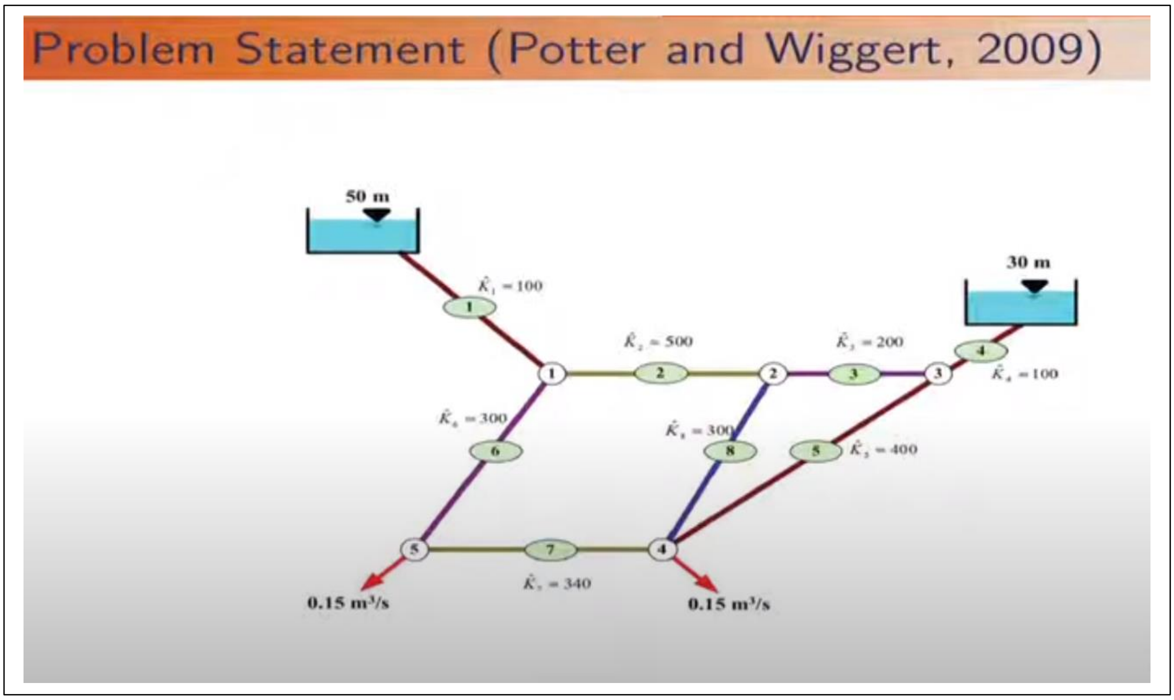
```
if(jun_con(j,1)>1) then
//It means, if more than one channel is connected to that junction.
   for l=1:jun_con(j,1)-1
//It runs for (number of channels connected to that junction-1) times.
Because, one channel is already calculated for the channel represented by
"jun_con(j,2)" elements in previous loop.
   eqn=eqn+1;
//"
   if(jun_con(j,2)>0)then jn_nodel=1; end
   if(jun_con(j,2)<0)then jn_nodel=mnode(abs(jun_con(j,2))); end
   A(eqn, 2*gid(abs(jun_con(j,2)),jn_nodel) -1)=1;
   r(eqn)=yv(gid(abs(jun_con(j,2)),jn_nodel));
//"
//?????????Why "  "  part is required? I think, it has been already calculated.
Ans: No, junction with multiple channel is seperately calculated. This part is
exclusively for junction 4.
   if(jun_con(j,l+2)>0)then jn_node2=1; end
// Minimum and maximum values of l are:(1 & jun_con(j,1)-1). So, it will
run for 1+2=3rd column to last non-zero column (depends on how many
channels are connected to that particular junction) of "jun_con" matrix.
   if(jun_con(j,l+2)<0)then jn_node2=mnode(abs(jun_con(j,l+2))); end
   A(eqn, 2*gid(abs(jun_con(j,l+2)),jn_node2)-1)=-1;
   r(eqn)=r(eqn)-yv(gid(abs(jun_con(j,l+2)),jn_node2));
   r(eqn)=-r(eqn);
end
end
end
//~~~~~del y Q~~~~~~~~
delyQ=A\r;
for i=1:2*sum(mnode)
gv(i)=gv(i)+delyQ(i);
rmse=rmse+delyQ(i)^2;
end
//~~~~~Update Value~~~~~
for l=1:chln
   for i=1:mnode(l)
      yv(gid(l,i))=gv(2*gid(l,i)-1);
      Qv(gid(l,i))=gv(2*gid(l,i));
//We should update those values so that we can use those values for the
next iteration.
   end
end
```

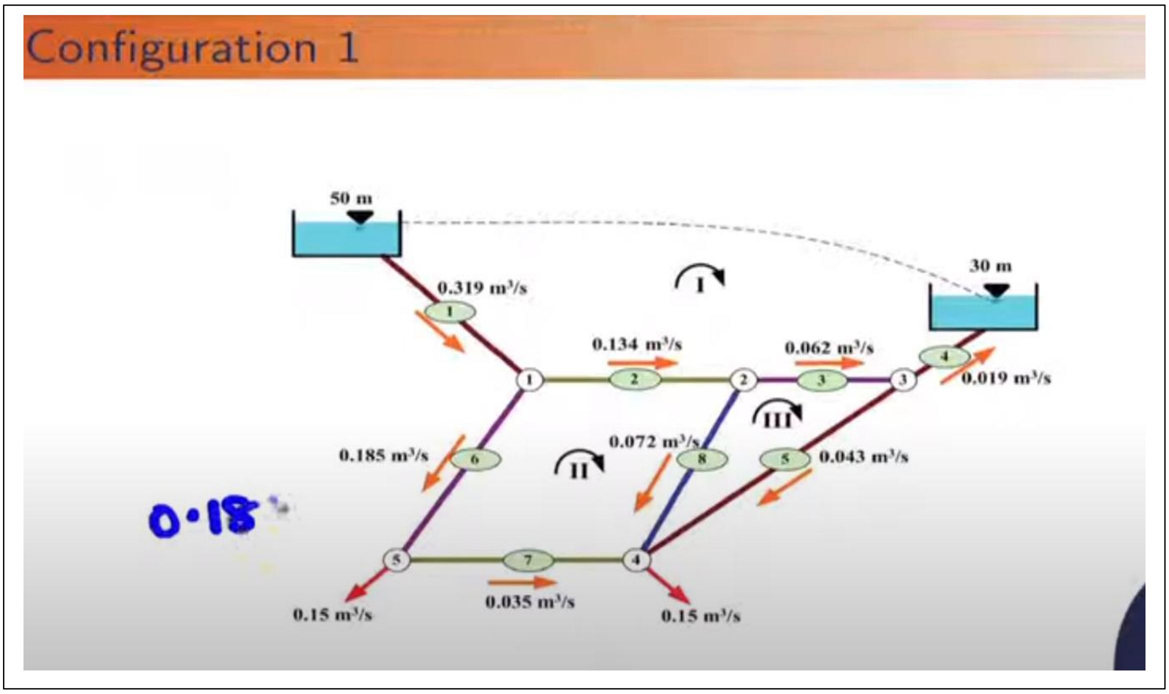
```
rmse=sqrt(rmse/sum(mnode));
count=count+1;
//disp([count rmse])
end

tcount=tcount+1;
//~~~~~~Update values for time n<--n+1~~~~~~
for l=1:chln
for i=1:mnode(l)
   yo(gid(l,i))=yv(gid(l,i));
   Qo(gid(l,i))=Qv(gid(l,i));
end
end

timev(tcount)=tv;
ysim(tcount)=yv(31);
Qsim(tcount)=Qv(31);

end

//Print output
plot(timev,ysim)
xtitle('Flow depth at 4000 m in channel 3','Time(s)', 'Flow Depth(m)');
figure
plot(timev,Qsim)
xtitle('Discharge at 4000 m in channel 3','Time(s)', 'Discharge(m^3/s)');
```





(30) Steady_1D_pipe_network





$$loop_{con} = \begin{bmatrix} 4 & -1 & -2 & -3 & -4 \\ 4 & 2 & 8 & -7 & -6 \\ 3 & 3 & 5 & -8 & 0 \end{bmatrix}$$

```
clc
clear
//~~~Given Data~~~`
g=9.81; //m/s^(-2)
global('g')
eps_max=1e-6;
betav=2; //exponent of discharge.
pipen=8; //Number of pipes.
nmaxpl=4;//Maximum number of pipes connected to a particular loop.
junn=5; //Number of junctions.
ploop=1; //Number of pseudo loop.
iloop=2; //Number of interior loop.
loopn=ploop+iloop; //Total number of loops.
Qi=[0.319 0.134 0.062 0.019 0.043 0.185 0.035 0.072]; //initial discharge
head_diff=[20]; //m , in pseudo loop
loop_con=[4 -1 -2 -3 -4
        4  2  8 -7 -6
        3  3  5 -8  0];
//In 'loop_con' marix, 1st column represents the number of pipes connected to
that loop. Other shows the pipes connected to that. + sign means flow in
clockwise direction.
pipe_con=zeros(pipen,loopn);
//'pipe_con' should be a 8*3 matrix.
for l=1:loopn
//'l' stands for loop numbering.(i.e, l=1:3).
       for i=1:loop_con(l,1)
//"loop_con(l,1)" represents the number of pipes connected to l-th loop.
           pipe_con(abs(loop_con(l,i+1)),l)=l;
//"for i=1:loop_con(l,1), abs(loop_con(l,i+1))" covers all the 'pipe nos'
connected to lth loop. We can see, each row of "loop_con"  matrix is dedicated
to a particular pipe no. It shows, in which loop a particular pipe is connected.
Output: pipe_con=
//   1.   0.   0.
//   1.   2.   0.
//   1.   0.   3.
//   1.   0.   0.
//   0.   0.   3.
//   0.   2.   0.
//   0.   2.   0.
//   0.   2.   3.
       end
end
```

```
Kv=[100 500 200 100 400 300 400 300];
//'Kv' means "Ki(cap)", which considers the total loss thing.

//~~~~~~Problem dependent parameters~~~~~~~~
Qv=zeros(loopn,nmaxpl);
//'Qv' is a (3*4) matrix.

//~~~~~General Identification Matrix~~~~
for l=1:loopn
   for i=1:loop_con(l,1)
      Qv(l,i)=sign(loop_con(l,i+1))*Qi(abs(loop_con(l,i+1)));
//'loop_con(l,i+1)' because channel numbering starts from 2nd column. For
each 'pipe loop(l)', i runs for 'loop_con(l,1)' times. Because, 'loop_con(l,1)' is
the number of pipes connected to l-th loop. Here, Qv considers the sign of
discharge also. Because, in loop_con matrix we considered the pipe
numberings with appropriate sign (Clockwise + ,Anticlockwise -)
   end
end

count=0;
rmse=1;


//~~~~~~~~Space Loop~~~~~~~~~~~~

while rmse>eps_max
   rmse=0;
   delQ=zeros(loopn,1);
//We are considering a particular delQ value for all pipes in each loop.

for l=1:loopn
   nr=0;
   dr=0;
//numerator and denominator initialized as zero values.
   if (l<=ploop)then
//Because, we have assigned the pseudo loop as loop no 1. As ploop=1,
therefore, l<=1 implies the pseudo loop , where a particular head difference is
present between two reservoirs.
      nr=nr+head_diff(l);
   end
```

```
for i=1:loop_con(l,1)
//"loop_con(l,1)" represents number of pipes connected to l-th loop.
   nr=nr+Kv(abs(loop_con(l,i+1)))*Qv(l,i)*abs(Qv(l,i))^(betav-1);
// In "Kv(abs(loop_con(l,i+1)))" i+1 is taken because,numbering of
connected pipes are started from 2nd column of 'loop_con' matrix.
   dr=dr+betav*Kv(abs(loop_con(l,i+1)))*abs(Qv(l,i))^(betav-1);
//See equation (412) for "nr" and "dr" calculation.
end
delQ(l)=-(nr/dr);
end
//From output, delQ= [0.0000010; 0.0000007; 0.0000009]
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
for l=1:loopn
//Considers loops
for j=1:loop_con(l,1)
//Considers all pipes connected to that loop.
   for k=1:loopn
//Related to 'pipe_con' matrix. Considers a pipe connected to how many
loops.
      if (pipe_con(abs(loop_con(l,j+1)),k)<>0)then
         if (pipe_con(abs(loop_con(l,j+1)),k)==l)then //this condition is
important.
            Qv(l,j)=Qv(l,j)+delQ(pipe_con(abs(loop_con(l,j+1)),k));
         else
            Qv(l,j)=Qv(l,j)-delQ(pipe_con(abs(loop_con(l,j+1)),k));
         end
      end
   end
end
end
```

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//Explanation of the above loop:
//when l=1,j=1,k=1,   pipe_con(abs(loop_con(1,2)),1)=pipe_con(1,1)=1;
Qv(1,1)=Qv(1,1)+delQ(pipe_con(abs(loop_con(1,2)),1))=Qv(1,1)+delQ(1
);
//when l=1,j=1,k=2,   pipe_con(abs(loop_con(1,2)),2)=pipe_con(1,2)=0; -
->stop
//when l=1,j=1,k=3,   pipe_con(abs(loop_con(1,2)),3)=pipe_con(1,3)=0; -
->stop
//when l=1,j=2,k=1,   pipe_con(abs(loop_con(1,3)),1)=pipe_con(2,1)=1;
Qv(1,2)=Qv(1,2)+delQ(pipe_con(abs(loop_con(1,3)),1))=Qv(1,2)+delQ(1
);
//when l=1,j=2,k=2,   pipe_con(abs(loop_con(1,3)),2)=pipe_con(2,2)=2;
Qv(1,2)=Qv(1,2)+delQ(pipe_con(abs(loop_con(1,3)),2))=Qv(1,2)-
delQ(2);
//when l=1,j=2,k=3,   pipe_con(abs(loop_con(1,3)),3)=pipe_con(2,2)=0;-
-->stop
//when l=1,j=3,k=1,   pipe_con(abs(loop_con(1,4)),1)=pipe_con(3,1)=1;
Qv(1,3)=Qv(1,3)+delQ(pipe_con(abs(loop_con(1,4)),1))=Qv(1,3)+delQ(1
);
//when l=1,j=3,k=2,   pipe_con(abs(loop_con(1,4)),2)=pipe_con(3,2)=0; -
-> stop
//when l=1,j=3,k=3,   pipe_con(abs(loop_con(1,4)),3)=pipe_con(3,3)=3;
Qv(1,3)=Qv(1,3)+delQ(pipe_con(abs(loop_con(1,4)),3))=Qv(1,3)-
delQ(3);
////when l=2,j=1,k=1,
pipe_con(abs(loop_con(2,2)),1)=pipe_con(2,1)=1;
Qv(2,1)=Qv(2,1)+delQ(pipe_con(abs(loop_con(2,2)),1))=Qv(2,1)-
delQ(1);
//Similarly other iterations will be there. I.T ---> (As 'l' stands for loop
and 'j' stands for pipe, so from "loop_con" matrix, we can say that
Qv(1,2)*,Qv(2,1)** both gives discharge of pipe 2. Similarly
Qv(1,3),Qv(3,1) both gives discharge of pipe 3. Qv(2,2),Qv(2,3) both
gives discharge of pipe 8. Those pairs should be added together to get
the discharges of pipe no 2,3 & 8.) →Wrong Idea,Discharge value of a
pipe for any loop has same value. Because, for any loop it considers all
effects.
// * In loop_con matrix, row 1,2 ,3 stands foe loop 1, loop2, loop3
respectively. So, Qv(1,2) means 2nd pipe of row 1 (i.e, pipe represented
by (1,3)th element. As pipes are started from 2nd column, 1st column
represents the total number of pipes connected to that particular loop).
// ** Similarly, Qv(2,1) means 1st pipe of row 2 (i.e, pipe represented by
(2,2)th element
```

```
for l=1:loopn
   rmse=rmse+delQ(l)^2;
end

rmse=sqrt(rmse/loopn);
count=count+1;
disp([count rmse])
end

//Print output
for l=1:loopn
   disp(['Loop Number:' string(l)])
   disp('Pipe Number  Discharge(m^3/s)')
   for i=1:loop_con(l,1)
      disp([abs(loop_con(l,i+1)) Qv(l,i)])
      end
   end
```

```
Output:          1.   0.0004983
                 2.   0.0003078
                 3.   0.0001496
                 4.   0.0001064
                 5.   0.0000546
                 6.   0.0000386
                 7.   0.0000212
                 8.   0.0000146
                 9.   0.0000084
                 10.   0.0000057
                 11.   0.0000034
                 12.   0.0000022
                 13.   0.0000013
                 14.   0.0000009
                "Loop Number:"  "1"
                 "Pipe Number  Discharge(m^3/s)"
                 1.  -0.3184094
                 2.  -0.1345158
                 3.  -0.0624709
                 4.  -0.0184094
                 "Loop Number:"  "2"
                 "Pipe Number  Discharge(m^3/s)"
                 2.   0.1345158
                 8.   0.072045
                 7.  -0.0338936
                 6.  -0.1838936
                 "Loop Number:"  "3"
                 "Pipe Number  Discharge(m^3/s)"
                 3.   0.0624709
                 5.   0.0440615
                 8.  -0.072045
```