Western Engineering

Assignment-4

Course Name: Computational Methods in Mechanical Engineering

MME 9621

Submitted by

Jagdeep Singh

STUDENT ID: 250911825

Course Instructor: Dr. Mohammad Hossain

Department of Mechanical and Materials Engineering

Western University, London, Ontario

March- 2017

Q1.

Function Vectorized RK4 Method

```
function [tout Yout]=VectorRK4(Fode,t,Y,NStep,h)
for k=1:NStep
s1=Fode(t(k),Y(k,:));
s2=Fode(t(k)+h/2,Y(k,:)+h/2.*s1');
s3=Fode(t(k)+h/2,Y(k,:)+h/2.*s2');
s4=Fode(t(k)+h,Y(k,:)+h.*s3');
Y(k+1,:)=Y(k,:)+h*(s1/6+s2/3+s3/3+s4/6)';
t(k+1)=t(k)+h;
end
Yout=Y;tout=t;
```

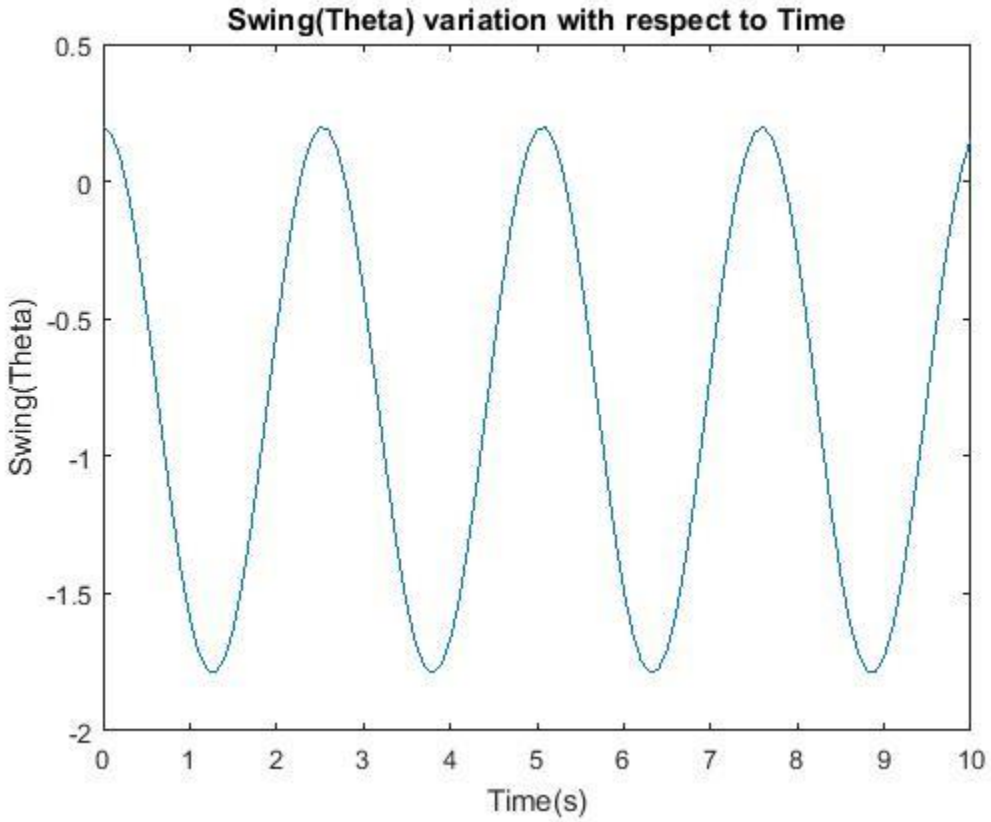
Main Program:

```
clc;clear all;close all; format long e;
h=0.1;t0=0;tn=10;g=9.8;b=10;l=2;
Node=2; % number of ODE
Fode=@(t,y)[y(2);(1/l).*(-b*cos(y(1))-g*sin(y(1)))];
NStep=abs(tn-t0)/h;
t=zeros(NStep,1);Yout=zeros(NStep,Node); % Pre-allocation
t(1)=t0;y(1)=0.2; y(2)=0; y0=[y(1) y(2)];% initial condition
Y(1,:)=y0; % to store results of y1 and y2 in a singe array
[t Y]=VectorRK4(Fode,t,Y,NStep,h)
figure(1);
plot(t,Y(:,1));
title('Swing(Theta) variation with respect to Time')
xlabel('Time(s)');ylabel('Swing(Theta)')
```

Results

Time(Sec)	Swing(Theat)	Dtheta/dt(Derivative)
0	**0.2000**	**0.0000**
0.1	0.1707	-0.5836
0.2	0.0841	-1.1436
0.3	-0.0563	-1.6519
0.4	-0.2435	-2.0745
-----	------	-----
3	-0.3952	-2.2969
3.1	-0.6359	-2.4910
3.2	-0.8876	-2.5146
3.3	-1.1330	-2.3649
3.4	-1.3554	-2.0603
3.5	-1.5410	-1.6337
-----	-----	-----
9.6	-0.5224	2.4213
9.7	-0.2923	2.1554
9.8	-0.0957	1.7580
9.9	0.0561	1.2663
10	**0.1556**	**0.7155**

Plot (Swing Variation with respect to Time):



Computation time for RK4 Method is 0.024235 seconds

Method 2: ODE45 (MATLAB Function)

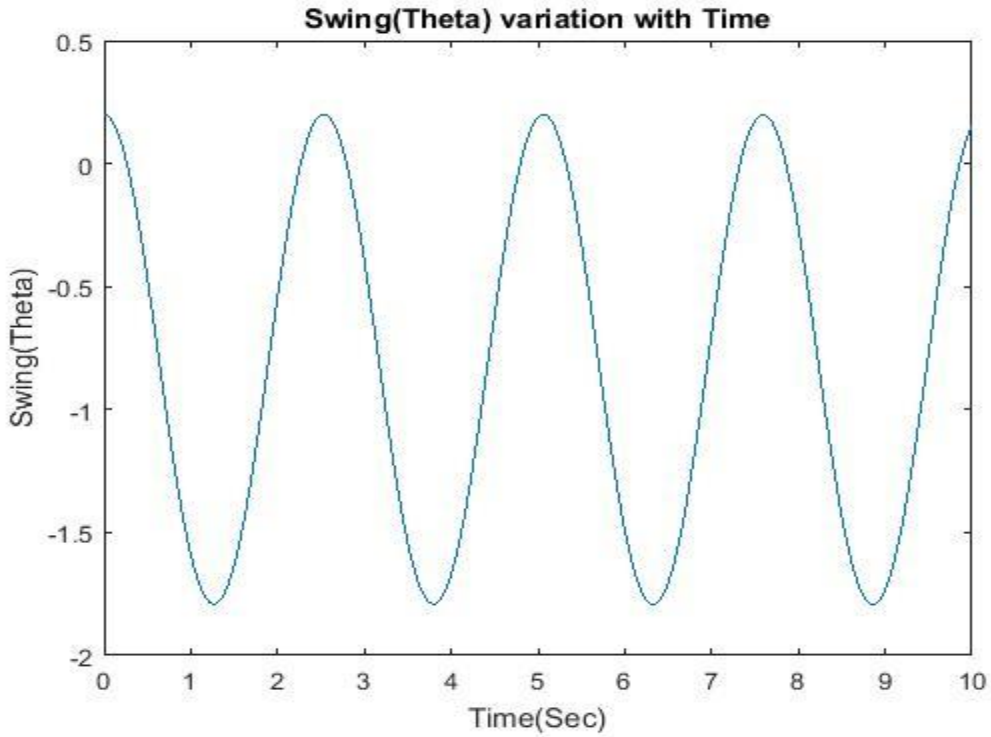
```
function Fode=ncc(t,y,g,b,l)
Fode=[y(2);(1/l).*(-b*cos(y(1))-g*sin(y(1)))];
```

Main Program

```
clc; clear all;close all; format long e;
g=9.8;b=10;l=2;
tspan=[0 10]; % time interval
y0=[0.2 0]; % initial condition
options=odeset('RelTol',1e-6,'AbsTol',1e-7);
tic
[t y]=ode45(@ncc,tspan,y0,options,g,b,l);
toc
No_of_steps_needed=length(t);
figure(1);
plot(t,y(:,1))
title('Swing(Theta) variation with Time')
xlabel('Time(Sec)'); ylabel('Swing(Theta)');
```

Results

Time(Sec)	Swing(Theat)	Dtheta/dt(Derivative)
0	**0.2000**	**0.0000**
0.000214837	0.2000	-0.0013
0.000429675	0.2000	-0.0025
0.000644512	0.2000	-0.0038
0.00085935	0.2000	-0.0050
0.001933537	0.2000	-0.0114
0.003007724	0.2000	-0.0177
0.004081911	0.2000	-0.0240
0.005156098	0.1999	-0.0303
-----	------	-----
9.728403369	-0.2321	2.0541
9.746102978	-0.1963	1.9861
9.763802587	-0.1618	1.9144
9.781502196	-0.1286	1.8394
-----	------	-----
9.949282457	0.1124	0.9995
9.966188305	0.1285	0.9057
9.983094152	0.1430	0.8107
10	**0.1559**	**0.7148**



Computation time for ODE 45 is 0.231571 seconds.

Method	No. of Steps	Computational Time(s)
Vectorized RK4	100	0.024235
ODE45	509	0.231571

In Vectorized RK4, the mesh is controlled by user by giving the step size that's why it took 100 step for solution for h=0.1(Step Size) for 10 seconds. On the other hand, ODE45 is based on the adaptive mesh and it took 509 steps to solve for 10 seconds. Therefore, RK4 Method is faster than ODE45. Though, ODE45 method is more accurate than RK4 method because of Adaptive mesh control and it discretizes the domain into more number of division and therefore, the solution obtained by ODE45 is more accurate. However, the number of steps evaluation is more in ODE45.

Q2:

Function Vectorized RK4 Method:1

```
function [tout Yout]=VectorRK4(Fode,t,Y,NStep,h)
for k=1:Nstep
s1=Fode(t(k),Y(k,:);
s2=Fode(t(k)+h/2,Y(k,:)+h/2.*s1');
s3=Fode(t(k)+h/2,Y(k,:+h/2.*s2');
s4=Fode(t(k)+h,Y(k,:+h.*s3');
Y(k+1,:)=Y(k,:)+h*(s1/6+s2/3+s3/3+s4/6)';
t(k+1)=t(k)+h;
end
Yout=Y;tout=t;
```

Main Program

```
clc;clear all; close all; format long e;
h=0.1;t0=0; tn=20; k1=3000;k2=2400;k3=1800;m1=12000;m2=10000;m3=8000;
Node=6; % number of ODE
Nstep=abs(tn-t0)/h;
t=zeros(Nstep,1);Yout=zeros(Nstep,Node); % Pre-allocation
Fode=@(t,y)[y(4);y(5);y(6);(-k1/m1)*y(1)+(k2/m1)*(y(2)-y(1));(k2/m2)*(y(1)-
y(2))+(k3/m2)*(y(3)-y(2));(k3/m3)*(y(2)-y(3))];
t(1)=t0;y(1)=0; y(2)=0; y(3)=0;y(4)=1;y(5)=0;y(6)=0; y0=[y(1) y(2) y(3) y(4)
y(5) y(6)];% initial condition
Y(1,:)=y0;
tic
[t Y]=VectorRK4(Fode,t,Y,Nstep,h)
Toc
```

%Plots

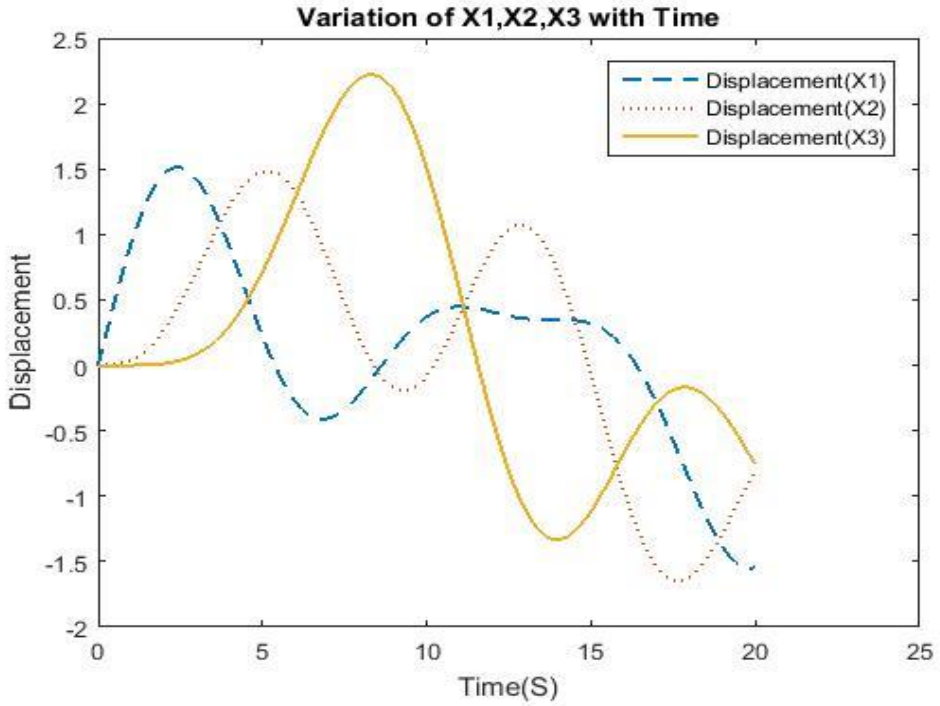
```
figure(1);plot(t,Y(:,1),'-','LineWidth',1.2)
hold on
plot(t,Y(:,2),':','LineWidth',1.2)
hold on
plot(t,Y(:,3),'LineWidth',1.2)
legend('Displacement(X1)','Displacement(X2)','Displacement(X3)')
xlabel('Time(S)');ylabel('Displacement');
title('Variation of X1,X2,X3 with Time')
figure(2);plot(t,Y(:,4),'-','LineWidth',1.2)
hold on
plot(t,Y(:,5),':','LineWidth',1.2)
hold on
plot(t,Y(:,6),'LineWidth',1.2)
```

```
legend('Velocity(DX1/Dt)','Velocity(DX2/Dt)','Velocity(DX3/Dt)')
title('Variation of V1,V2,V3 with Time')
xlabel('Time(S)'); ylabel('Velocity');
hold off
figure(3);
plot3(Y(:,1),Y(:,2),Y(:,3),'-','LineWidth',1.5)
xlabel('Displacement(X1)');ylabel('Displacement(X2)');zlabel('Displacement(X3
)')
title('3D Phase Plane Plot for Displacement')
```

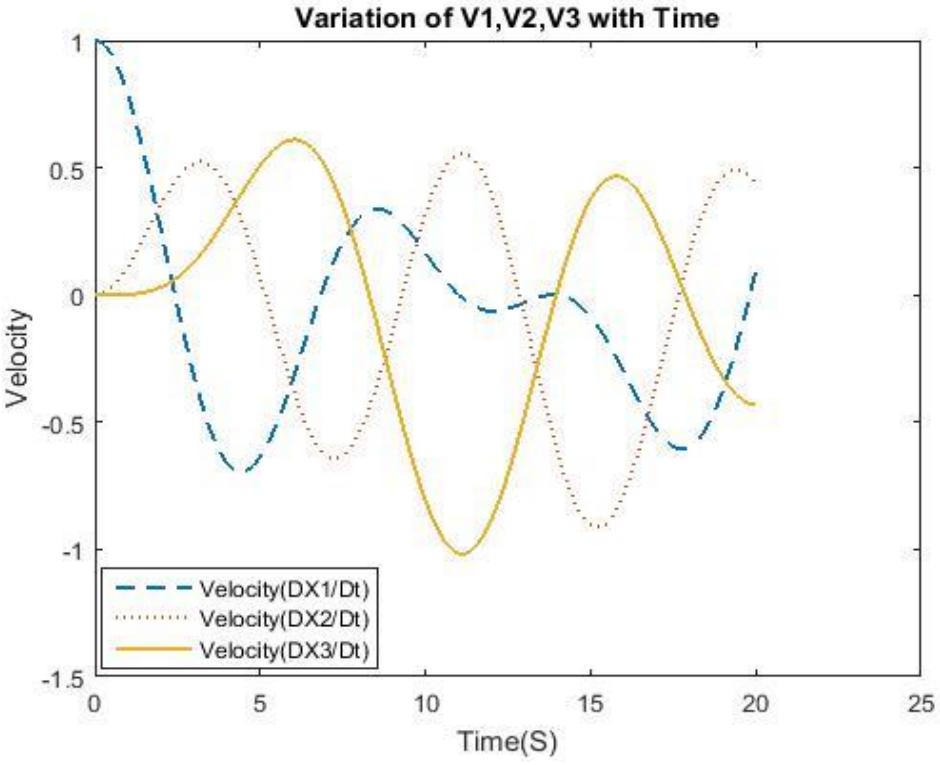
Results:

Time	X1	X2	X3	DX1/Dt(V1)	DX2/Dt(V2)	DX3/Dt(V3)
0	**0**	**0.0000**	**0.0000**	**1.0000**	**0.0000**	**0.0000**
0.1	0.09993	0.0000	0.0000	0.9978	0.0012	0.0000
0.2	0.1994	0.0003	0.0000	0.9910	0.0048	0.0000
0.3	0.29798	0.0011	0.0000	0.9798	0.0107	0.0000
0.4	-0.3323	1.2440	1.4074	-0.2467	-0.4442	0.6076
6.3	-0.355	1.1979	1.4680	-0.2068	-0.4779	0.6027
6.4	-0.3737	1.1485	1.5279	-0.1669	-0.5090	0.5954
6.5	-0.3884	1.0962	1.5870	-0.1273	-0.5372	0.5856
6.6	0.41392	0.8531	-0.3209	-0.0622	0.4308	-0.9111
----	----	----	----	----	----	----
20	**-1.541**	**-0.7995**	**-0.7603**	**0.0945**	**0.4416**	**-0.4324**

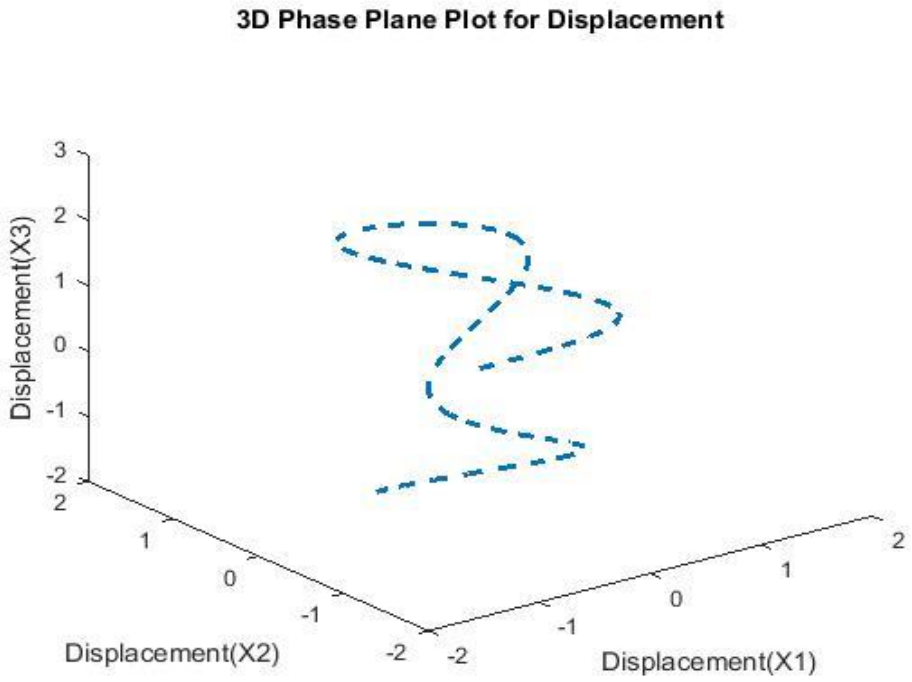
Plot 1: Displacement vs Time



Plot 2: Velocity vs Time



Plot 3: 3D Phase Plane Plot for Displacements:



Method 2: ODE45 "MATLAB" Function

```
clc; clear all;close all; format long e;
k1=3000;k2=2400;k3=1800;m1=12000;m2=10000;m3=8000;
tspan=[0 20]; % time Span
y0=[0 0 0 1 0 0];% initial condition
Fode=@(t,y)[y(4);y(5);y(6);(-k1/m1)*y(1)+(k2/m1)*(y(2)-y(1));(k2/m2)*(y(1)-
y(2))+(k3/m2)*(y(3)-y(2));(k3/m3)*(y(2)-y(3))];
options=odeset('RelTol',1e-6,'AbsTol',1e-7);
tic
[t y]=ode45(Fode,tspan,y0,options);
toc
N_Step=length(t);
```

%Plots

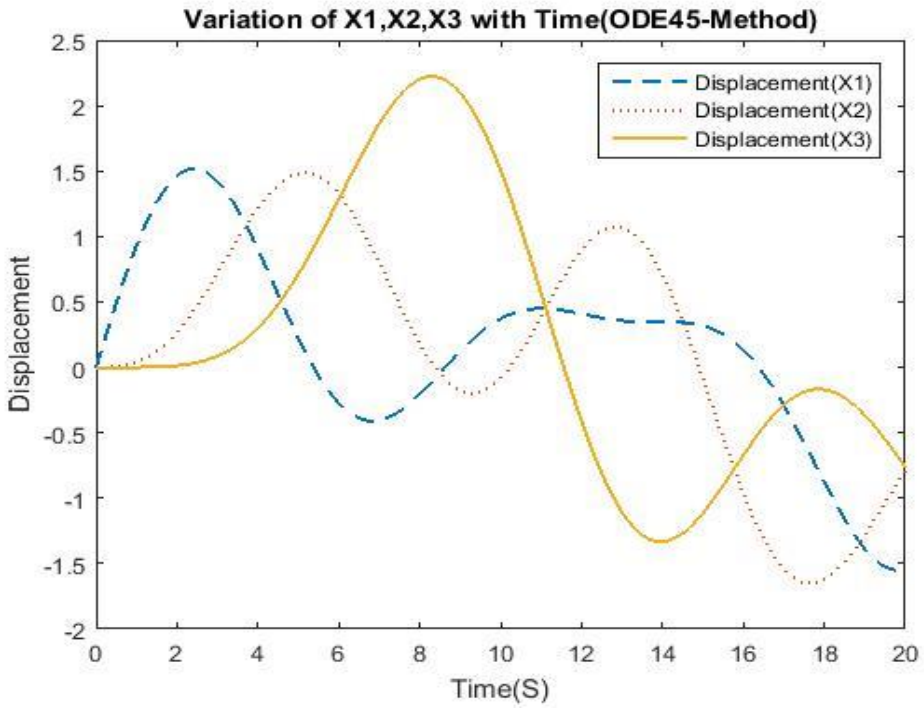
```
figure(1);plot(t,y(:,1),'--','LineWidth',1.2)
hold on
plot(t,y(:,2),':','LineWidth',1.2)
hold on
plot(t,y(:,3),'LineWidth',1.2)
legend('Displacement(X1)','Displacement(X2)','Displacement(X3)')
xlabel('Time(S)');ylabel('Displacement');
title('Variation of X1,X2,X3 with Time(ODE45-Method)')
figure(2);plot(t,y(:,4),'--','LineWidth',1.2)
hold on
plot(t,y(:,5),':','LineWidth',1.2)
hold on
plot(t,y(:,6),'LineWidth',1.2)
legend('Velocity(DX1/Dt)','Velocity(DX2/Dt)','Velocity(DX3/Dt)')
```

```
title('Variation of V1,V2,V3 with Time(ODE45)')
xlabel('Time(S)'); ylabel('Velocity');
hold off
figure(3);
plot3(y(:,1),y(:,2),y(:,3),'--','LineWidth',1.5)
xlabel('Displacement(X1)');ylabel('Displacement(X2)');zlabel('Displacement(X3
)')
title('3D Phase Plane Plot for Displacement(ODE45)')
```

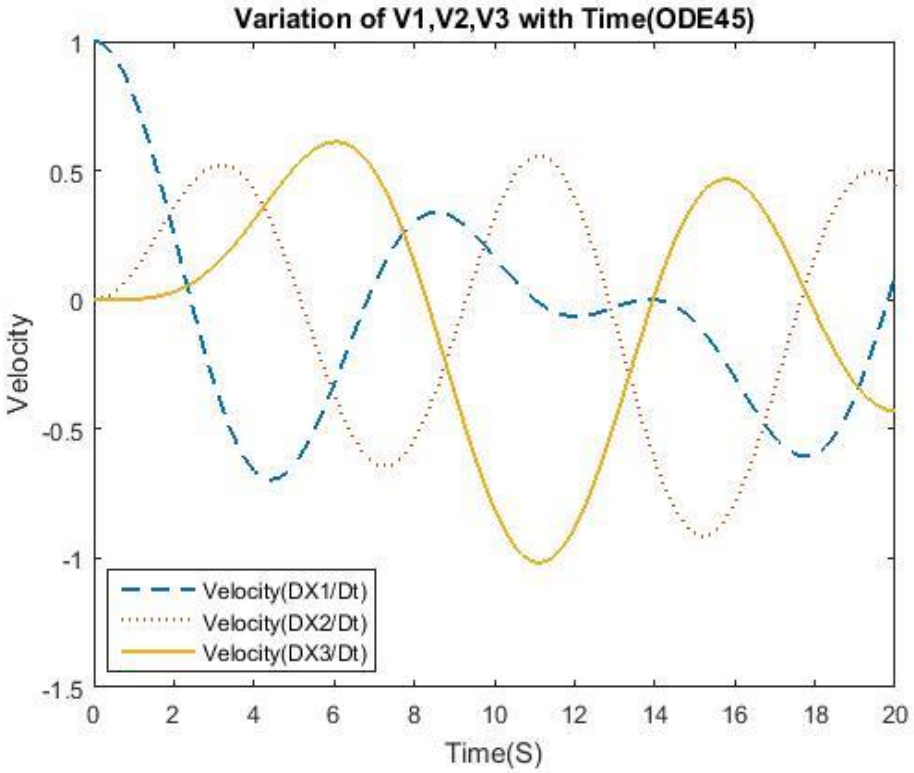
Results

Time	X1	X2	X3	DX1/Dt(V1)	DX2/Dt(V2)	DX3/Dt(V3)
0	0	0.0000	0.0000	1	0	0
0.001262	0.001262	0.0000	0.0000	0.999999642	1.91E-07	5.71E-15
0.002524	0.0025234	0.0000	0.0000	0.999998567	7.64E-07	9.13E-14
0.003786	0.003786	0.0000	0.0000	0.999996775	1.72E-06	4.62E-13
0.005048	0.0050476	0.0000	0.0000	0.999994267	3.06E-06	1.46E-12
----	----	----	----	----	----	----
15.35191	0.276501	-0.4043	-0.9699	-0.149806389	-0.909128853	0.439883505
15.40741	0.267869	-0.4547	-0.9453	-0.161372581	-0.905057849	0.44647826
15.4629	0.258584	-0.5047	-0.9204	-0.173272521	-0.899688005	0.45213651
15.5184	0.248630	-0.5545	-0.8951	-0.185486419	-0.893031923	0.456858061
----	----	----	----	----	----	----
20	**-1.54099**	**-0.7995**	**-0.7603**	**0.094540302**	**0.441622667**	**0.432445896**

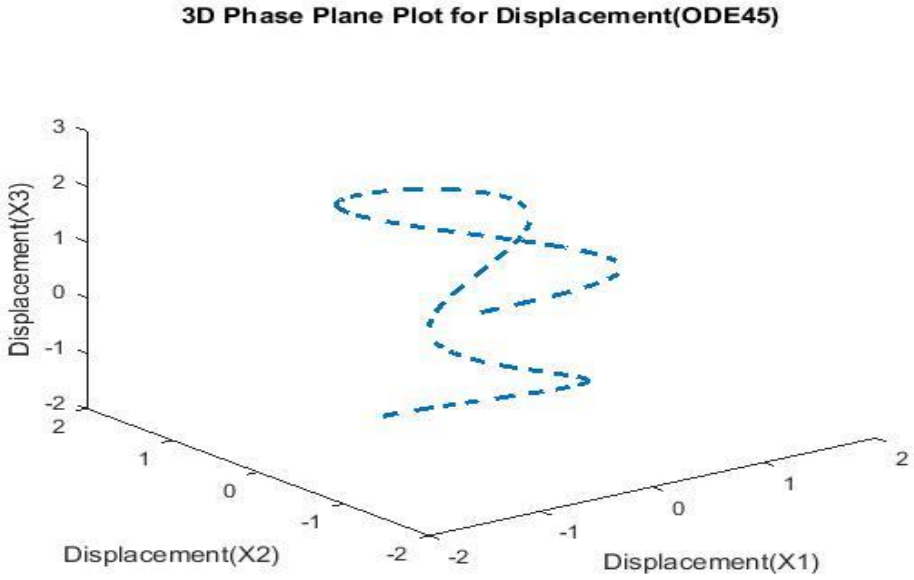
Plot 1: Displacement vs Time



Plot 2: Velocity vs Time



Plot 3: 3D Phase Plane Plot for Displacements:



Comparison between Vectorized RK4 and ODE45

Method	No. of Steps	Computational Time(s)
Vectorized RK4	200	0.041617
ODE45	377	0.466359

In Vectorized RK4, the mesh is controlled by user by giving the step size that's why it took 200 step for solution for h=0.1(Step Size) for 20 seconds. On the other hand, ODE45 is based on the adaptive mesh and it took 377 steps to solve for 10 seconds. Therefore, RK4 Method is faster than ODE45. Though, ODE45 method is more accurate than RK4 method because of Adaptive mesh control and it discretizes the domain into more number of division and therefore, the solution obtained by ODE45 is more accurate. However, the number of steps evaluation is more in ODE45.

Note: The unit of spring constant (k) taken in the problem is N/m.

Q3:

Method 1:

Main Function of ODE

```
function fode=defFunc(x,y,E,I,w,l)
fode=[y(2);y(3);y(4);abs(-w/(E*I))];
```

Main Function for Boundary Condition

```
function res=defBC(ya,yb,E,I,w,l)
res=[ya(1)-0;yb(1)-0;ya(3)-0;yb(3)-0];
```

Main Program

```
clc;close all; clear all; format long e;
E=2e+11; I=3e-4;w=15e+3;l=3;
bl=0.0005;br=3;
N=10;
x=linspace(bl,br,N);
yinit=[0 0 0 0];
solinit=bvpinit(x,yinit);
tic
sol=bvp4c(@defFunc,@defBC,solinit,[],E,I,w,l);
t1=toc
xfinal=sol.x;
y_sol=sol.y(1,:);
figure(1);
plot(xfinal,y_sol,'LineWidth',1.6)
xlabel('length of beam');ylabel('Deflection');
title('Defelction Curve for Simply supported Beam')
%Mesh Refinement of deflection curve
N=300;
x=linspace(bl,br,N);
y=deval(sol,x);
y=y';
figure(2);
plot(x,y(:,1),'LineWidth',1.6)
xlabel('Length of beam(m)');ylabel('Deflection(m)');
```

```
title('Smooth curve of deflection at N=300');
title('Defelction Curve for Simply supported Beam')
```

Analytical Solution:

```
clc; close all; clear all;
syms y w ee ii   %double e and double I used to avoid complex number and Exp
y=dsolve('D4y=-w/(ee*ii)','y(0)=0','y(3)=0','D2y(0)=0','D2y(3)=0','x')
```

OUTPUT

```
y = (w*x^3)/(4*ee*ii) - (9*w*x)/(8*ee*ii) - (w*x^4)/(24*ee*ii)
```

Error Computation

Max Deflection Value for Numerical Solution

```
Max_y_sol=max(y_sol)
```

```
Max_y_sol=2.634562199133655e-04
```

Max Deflection Value for Analytic Solution

```
Max_y=max(y)
```

```
Max_y=2.636319305999925e-04
```

Absolute Error

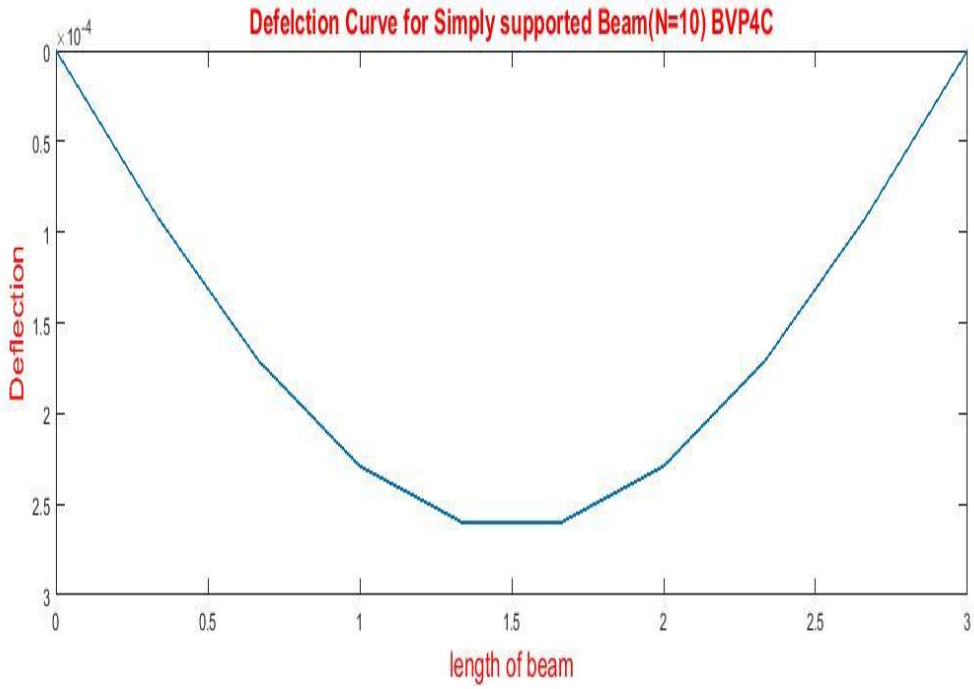
```
Absolute_Error=abs((Error_y)-(Error_y_sol))
```

Absolute_Error = 1.757106866269432e-07

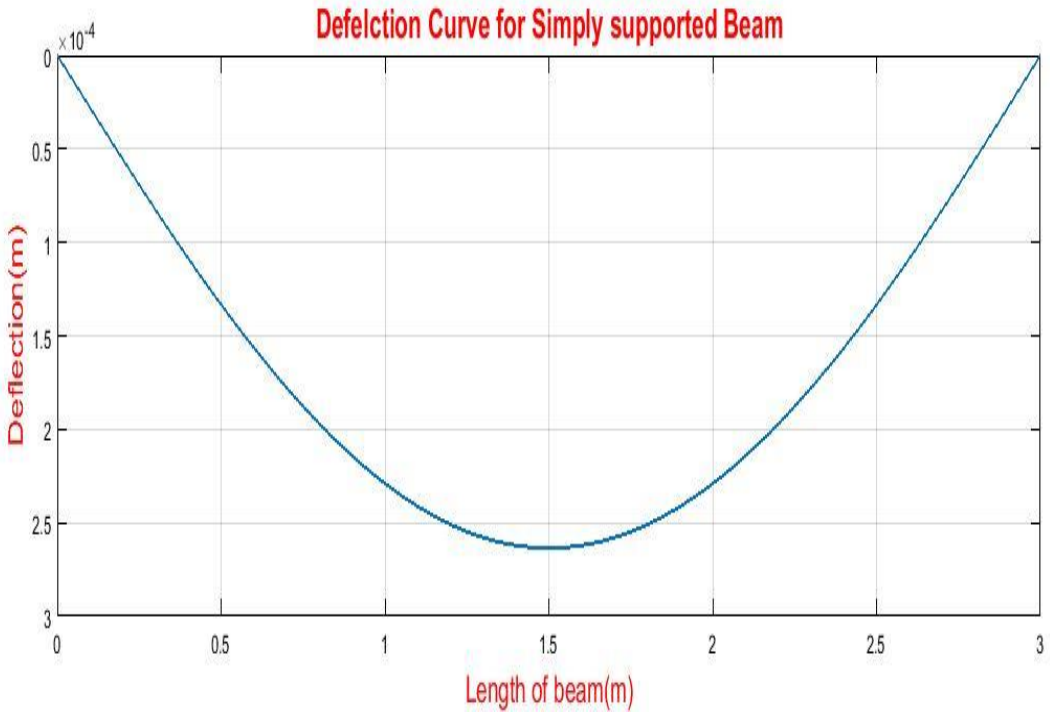
Computational Time = 0.277847 Seconds

Results

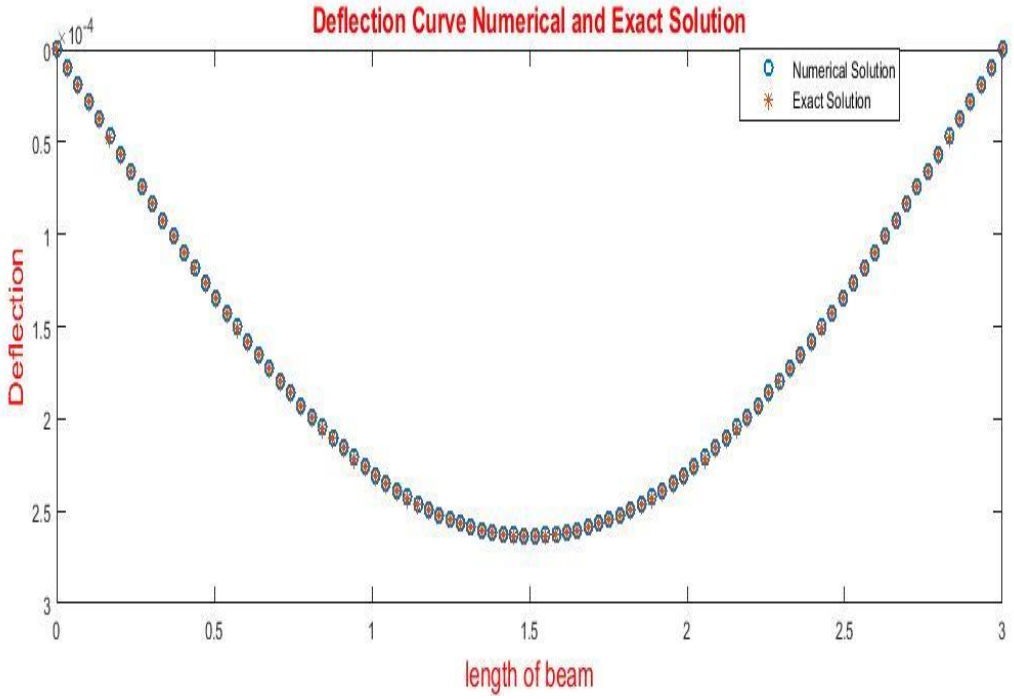
1. **Deflection Plot for Simply supported beam for N=10 steps(Mesh Size)**



2. **Deflection Plot for Simply supported beam for N=300 steps(Mesh Refinement)**



3. **Comparison of Numerical Solution with Analytic Solution:**



Q4.

$$\frac{\partial}{\partial y}\left(\mu\frac{\partial u}{\partial y}\right)-\frac{\partial p}{\partial x}=0$$

Discretize the above equation into algebraic form by applying suitable boundary conditions:

u =0 at y =H/2, and -H/2

Where; Pressure gradient is constant

$$\frac{\partial p}{\partial x}=-K$$

Domain

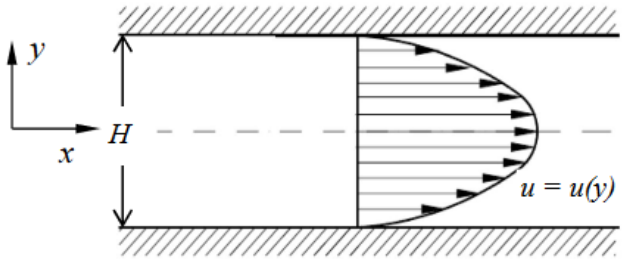


Figure 4. Plane Poiseuille flow.

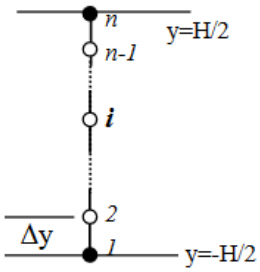


Figure 5. Uniform grid for full-channel height (*H*).

By applying Taylor series;

We get the central differencing finite method scheme of second order derivative of velocity field

$$\frac{\partial^2 u}{\partial y^2} = -K/\mu$$

$$\frac{u_{k+1} - uy_k + u_{k-1}}{h^2} = -K/\mu$$

$$u_{k+1} - 2u_k + u_{k-1} + \frac{K}{\mu}h^2 = 0$$

For k=1 to 5

$$u_2 - 2u_1 + u_0 + \frac{K}{\mu}h^2 = 0$$

$$u_3 - 2u_2 + u_1 + \frac{K}{\mu}h^2 = 0$$

$$u_4 - 2u_3 + u_2 + \frac{K}{\mu}h^2 = 0$$

$$u_5 - 2u_4 + u_3 + \frac{K}{\mu}h^2 = 0$$

$$u_6 - 2u_5 + u_4 + \frac{K}{\mu}h^2 = 0$$

Applying boundary condition, or at wall no slip boundary condition u_0 and $u_6 = 0$

$$u_2 - 2u_1 + \frac{K}{\mu}h^2 = 0$$

$$u_3 - 2u_2 + u_1 + \frac{K}{\mu}h^2 = 0$$

$$u_4 - 2u_3 + u_2 + \frac{K}{\mu}h^2 = 0$$

$$u_5 - 2u_4 + u_3 + \frac{K}{\mu}h^2 = 0$$

$$-2u_5 + u_4 + \frac{K}{\mu}h^2 = 0$$

Matrix Form:

$$A = \begin{bmatrix} -2 & 1 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 1 & -2 \end{bmatrix}$$

B = - $\frac{K}{\mu}h^2$ (Matrix-B) column vector

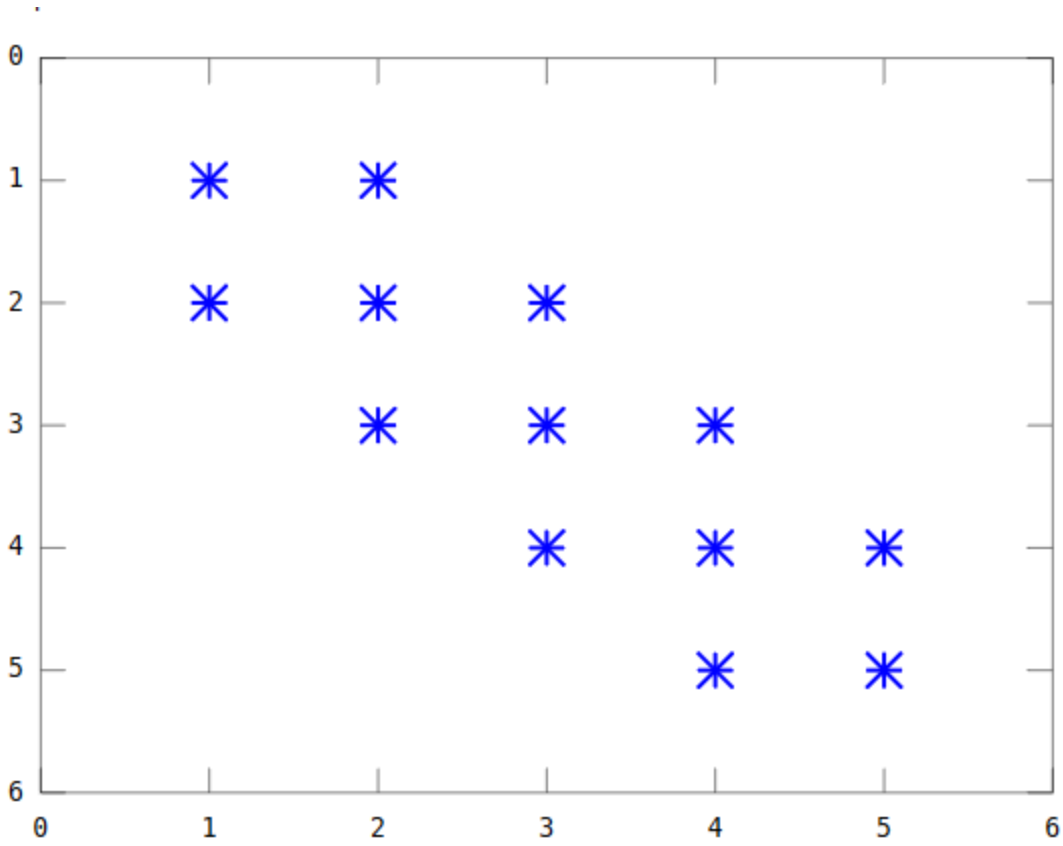
The matrix A is Tridiagonal Matrix with sub and super diagonal are non-zero. This type of matrix can be solved with the help of Tridiagonal Method.

Super diagonal =[0 1 1 1 1]

Main Diagonal =[-2 -2 -2 -2 -2]

Sub diagonal =[1 1 1 1 0]

Structure of Matrix A



Solution of Velocity field

U1	8.33e-02
U2	1.33e-01
U3	1.49e-01
U4	1.33e-01
U5	8.33e-02

At walls u0 and u6 = 0

Solver: Tridiagonal

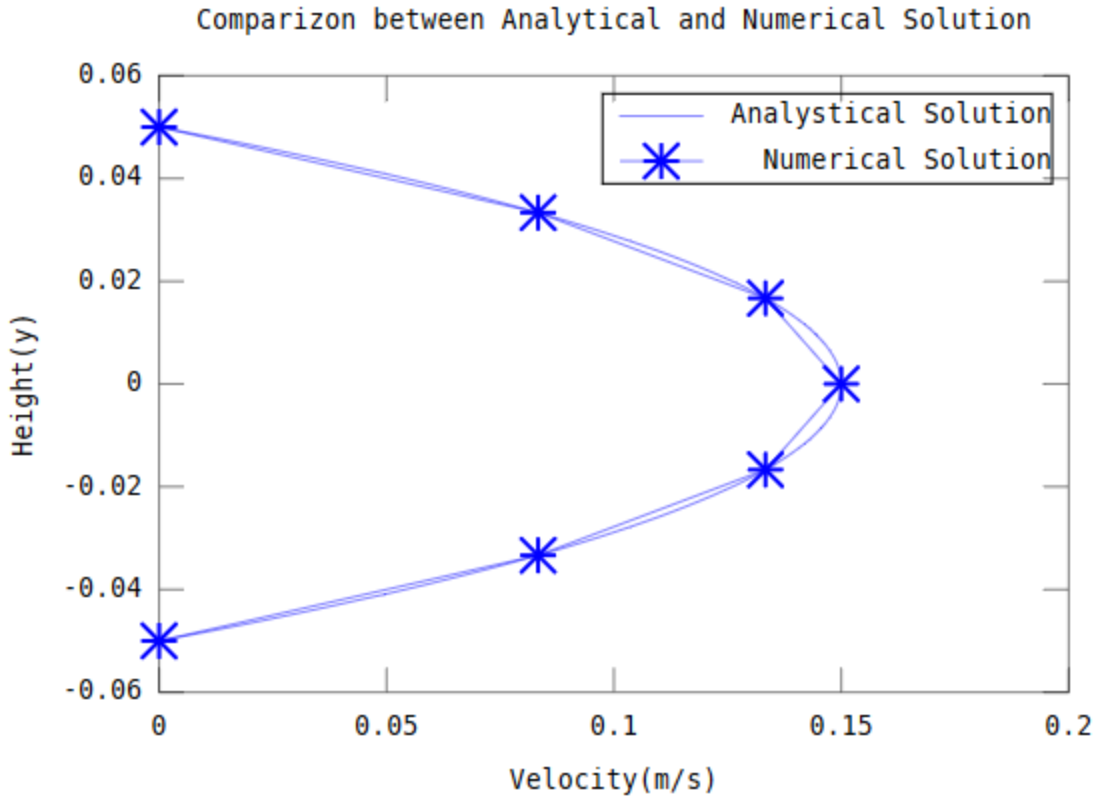
```
function [xout]=Tridiagonal(a,b,c,d)
```

```
N=length(b);
for k=2:N
    multiplier=a(k)/b(k-1);
    b(k)=b(k)-multiplier*c(k-1);
    d(k)=d(k)-multiplier*d(k-1);
end
x(N)=d(N)/b(N);
for k=N-1:-1:1
    x(k)=(d(k)-c(k)*x(k+1))/b(k);
end
xout=x.';
```

Main Program

```
clc; clear all; close all;format long e
    uav=0.1;H=0.1;d=1.2;mu=4e-5;h=(0.1/6);
    K=(12*mu*uav)/(H^2);
    z=(K/mu)*(h^2);
    A=[-2 1 0 0 0;1 -2 1 0 0;0 1 -2 1 0;0 0 1 -2 1;0 0 0 1 -2];
    a=[0 1 1 1 1];
    b=[-2 -2 -2 -2 -2];
    c=[1 1 1 1 0];
    d=[-z -z -z -z -z];
    conditon_number_A=cond(A)
    tic
    [xout]=Tridiagonal(a,b,c,d)
    toc
    spy(A)

y=linspace(-0.05,0.05,100)
u=(1.5*uav*(1-((y.^2)/(H/2)^2)));
u_num=[0;xout;0]
tspan=linspace(-0.05,0.05,7);
figure(1)
plot(u,y);
hold on
plot(u_num,tspan,'-*');
xlabel('Velocity');ylabel('Height')
title('Comparizon between Analytical and Numerical Solution')
legend('Analystical Solution','Numerical Solution')
hold off
```



f)

U3=1.49e-01 (Max Value)[Numerical]

$$u_{max} = \frac{3}{2} u_{av} .$$

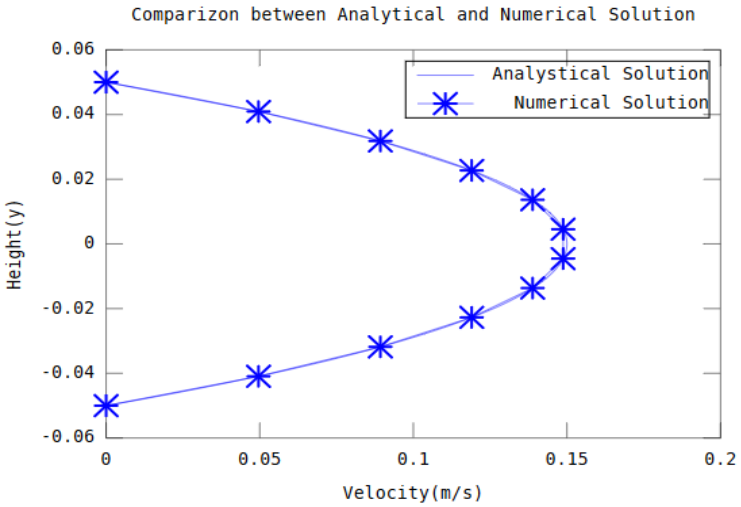
$\dot{U}max=0.1\dot{5}$ (m/s)

U3~ Umax
Therefore, the results are in good agreement with the analytic solution.

g) Grid Independent test

Taking n=10

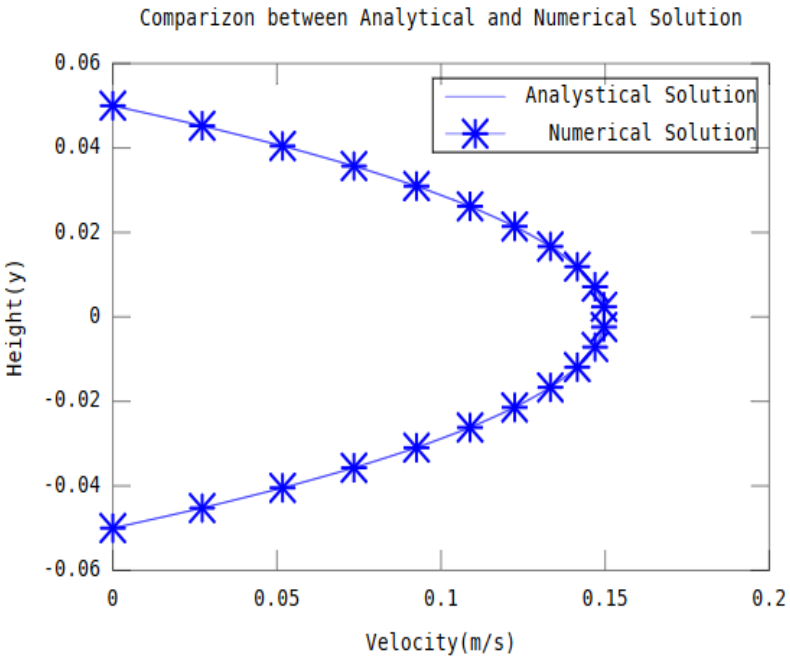
```
0.00000000000000e+00
4.95867768595041e-02
8.92561983471075e-02
1.19008264462810e-01
1.38842975206612e-01
1.48760330578512e-01
1.48760330578512e-01
1.38842975206612e-01
1.19008264462810e-01
8.92561983471074e-02
4.95867768595041e-02
0.00000000000000e+00
```



Taking n=20

u_num =

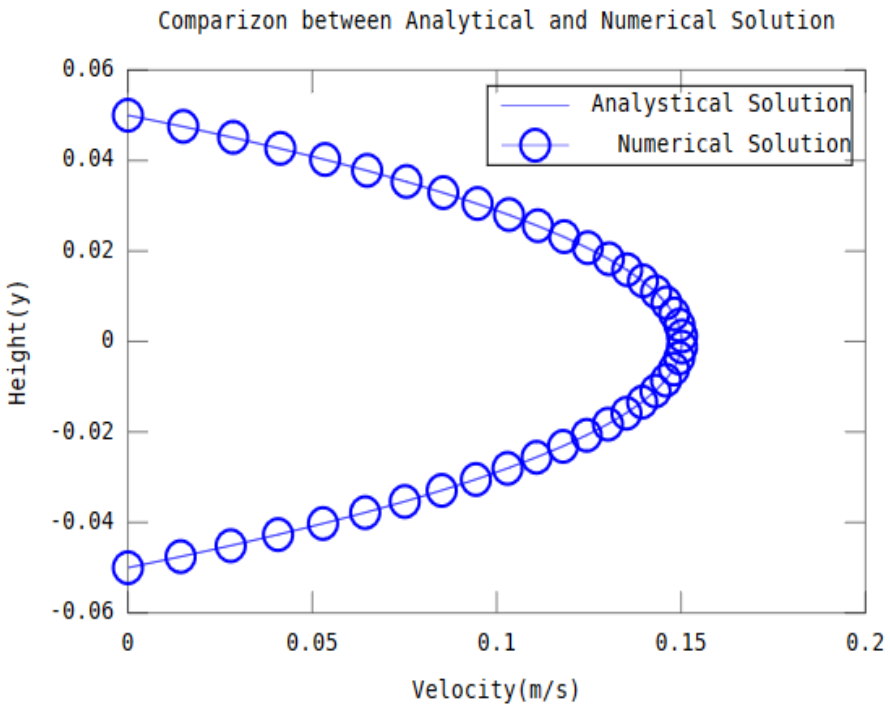
```
0.00000000000000e+00
2.72108843537415e-02
5.17006802721089e-02
7.34693877551020e-02
9.25170068027211e-02
1.08843537414966e-01
1.22448979591837e-01
1.33333333333333e-01
1.41496598639456e-01
1.46938775510204e-01
1.49659863945578e-01
1.49659863945578e-01
1.46938775510204e-01
1.41496598639456e-01
1.33333333333333e-01
1.22448979591837e-01
1.08843537414966e-01
9.25170068027211e-02
7.34693877551020e-02
5.17006802721088e-02
2.72108843537415e-02
0.00000000000000e+00
```



Takin N=40

u_num =

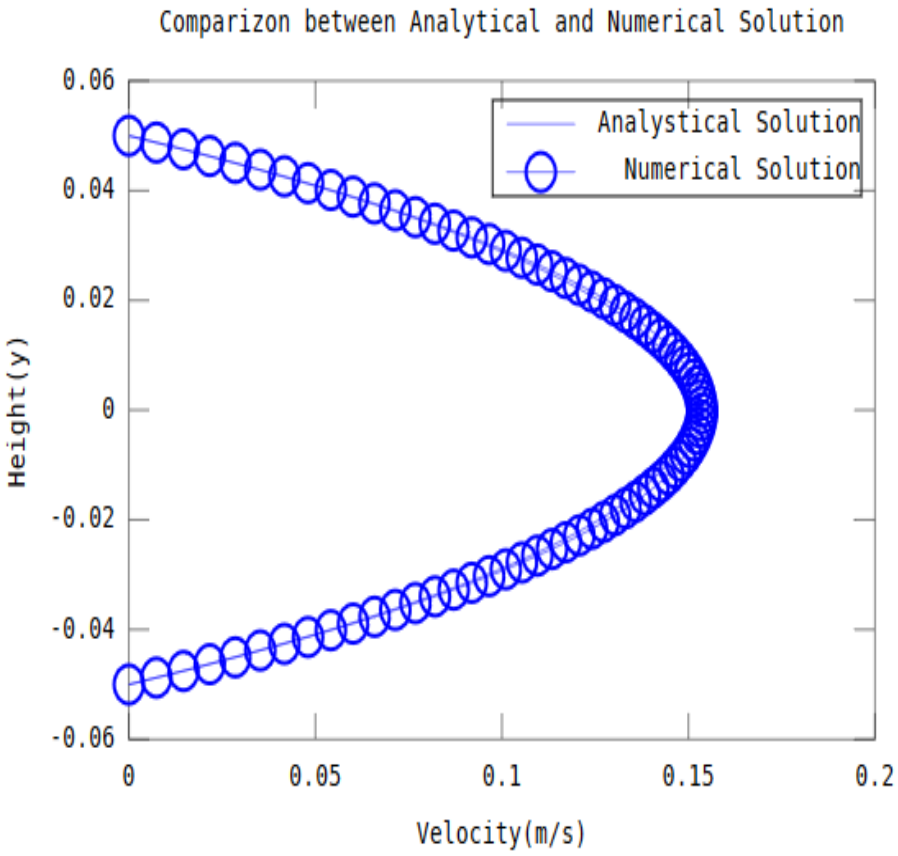
0.00000000000000e+00
1.42946271818458e-02
2.78753935665471e-02
4.07422991541039e-02
5.28953439445160e-02
6.43345279377836e-02
7.50598511339067e-02
8.50713135328852e-02
9.43689151347191e-02
1.02952655939409e-01
1.10822535946953e-01
1.17978555157354e-01
1.24420713570609e-01
1.30149011186721e-01
1.35163448005687e-01
1.39464024027509e-01
1.43050739252187e-01
1.45923593679720e-01
1.48082587310108e-01
1.49527720143352e-01
1.50258992179451e-01
1.50276403418406e-01
1.49579953860216e-01
1.48169643504882e-01
1.46045472352403e-01
1.43207440402779e-01
1.39655547656011e-01
1.35389794112099e-01
1.30410179771042e-01
1.24716704632840e-01
1.18309368697494e-01
1.11188171965003e-01
1.03353114435368e-01
9.48041961085877e-02
8.55414169846632e-02
7.55647770635942e-02
6.48742763453807e-02
5.34699148300226e-02
4.13516925175199e-02
2.85196094078727e-02
1.49736655010809e-02
0.00000000000000e+00



Taking N=80

u_num =

0.00000000000000e+00
7.40853641326124e-03
1.46341738781914e-02
2.16769123947906e-02
2.85367519630587e-02
3.52136925829957e-02
4.17077342546017e-02
4.80188769778767e-02
5.41471207528206e-02
6.00924655794335e-02
6.58549114577153e-02
7.14344583876661e-02
7.68311063692858e-02
8.20448554025745e-02
8.70757054875321e-02
9.19236566241587e-02
9.65887088124542e-02
1.01070862052419e-01
1.05370116344052e-01
1.09486471687355e-01
1.13419928082326e-01
1.17170485528966e-01
1.20738144027276e-01
1.24122903577254e-01
1.27324764178901e-01
1.30343725832217e-01
1.33179788537202e-01
1.35832952293856e-01
1.38303217102179e-01
1.40590582962171e-01
1.42695049873832e-01
1.44616617837162e-01
1.46355286852161e-01
1.47911056918829e-01
1.49283928037166e-01
1.50473900207171e-01
1.51480973428846e-01
1.52305147702190e-01
1.52946423027202e-01
1.53404799403884e-01
1.53680276832234e-01
1.53589956363923e-01
1.53316736947280e-01
1.52860618582306e-01
1.52221601269002e-01
1.51399685007366e-01
1.50394869797399e-01
1.49207155639101e-01
1.47836542532472e-01
1.46283030477512e-01
1.44546619474221e-01
1.42627309522599e-01
1.40525100622646e-01
1.38239992774362e-01



```
1.35771985977747e-01
1.33121080232801e-01
1.30287275539523e-01
1.27270571897915e-01
1.24070969307976e-01
1.20688467769705e-01
1.17123067283104e-01
1.13374767848171e-01
1.09443569464908e-01
1.05329472133313e-01
1.01032475853387e-01
9.65525806251304e-02
9.18897864485426e-02
8.70440933236238e-02
8.20155012503739e-02
7.68040102287929e-02
7.14096202588810e-02
6.58323313406379e-02
6.00721434740638e-02
5.41290566591587e-02
4.80030708959225e-02
4.16941861843553e-02
3.52024025244571e-02
2.85277199162277e-02
2.16701383596674e-02
1.46296578547760e-02
7.40627840155350e-03
0.00000000000000e+00
```

Grid Independent Test:

NO. of Nodes	U_max	Percentage Difference
5	1.499e-1	0.8
10	1.487e-1	
20	1.496e-1	0.2
40	1.499e-1	
80	1.499e-1	0

The grid independence test has been carried out using five different number of nodes as shown in table above. At 40 number of nodes, we get the optimum solution of U_max which is in good agreement with analytic solution also. Further increase in the number of nodes doesn't effect the value of U_max upto 4 decimal places. Therefore, we can solve the problem by taking 40 number of nodes to reduce the computational time.

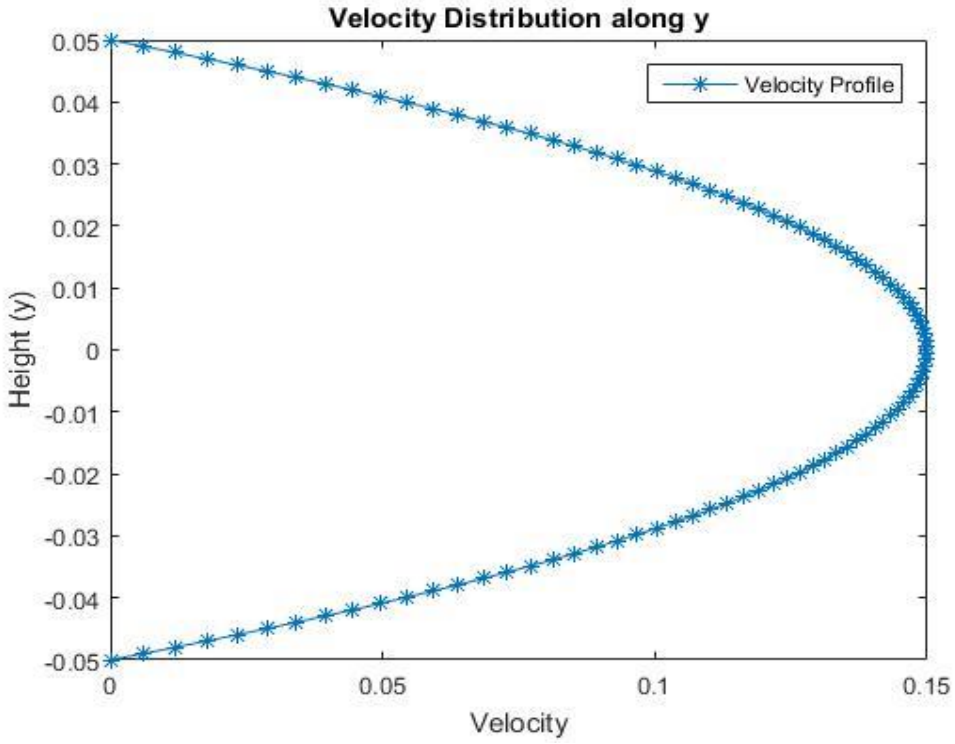
h) BVP4C Method

```
function fode=ppf(x,y,k,mu,H,uav)
fode=[y(2);-k/mu];
function fb=ppfBC(ya,yb,k,mu,H,uav)
fb=[ya(1)-0;yb(1)-0];
```

Main Program

```
clc; clear all; close all;
bl=-0.05;br=0.05;
H=0.1;mu=4e-5;uav=0.1;
k=(12*uav*mu)/(H^2);
```

```
N=100;
x=linspace(bl,br,N);
yinit=[0 0];
solinit=bvpinit(x,yinit);
tic
sol=bvp4c(@ppf,@ppfBC,solinit,[],k,mu,H,uav)
t1=toc
xfinal=sol.x;
y_sol=sol.y(1,:);
plot(y_sol,xfinal,'-*')
legend('Velocity Profile')
xlabel('Velocity');ylabel('Height (y)');
title('Velocity Distribution along y');
```



1. **Computational Speed**=0.226700 seconds
2. **Error**= 0% upto 4 decimal places by applying Error norm and calculated the absolute error
   maxofNumericalSolution =0.1500
   maxofanalytic Sol =0.1500
   Absolute Error=abs(maxofanalytic Sol- maxofNumericalSolution)

3. **Tridiagonal Method** calculation for N=80 took only 0.0060989 seconds

   However, BVP4C is more accurate for lower number of nodes also.