CHAPTER 2
Solution of the Vibration Equation

1. Introduction

The differential equations that govern the vibration system is given by:

$$m\ddot{x} + c\dot{x} + kx = f(t) \tag{1}$$

where
m: Inertia coefficient
c: Damping coefficient
k Stiffness coefficient
x : Displacement
$\dot{x}$: Velocity $= \frac{dx}{dt}$
$\ddot{x}$: Acceleration $= \frac{d^2x}{dt^2} = \frac{d\dot{x}}{dt}$
$f(t)$: Forcing function that might depend on time.

In the linear theory of vibration, m, c and k are constant coefficients. If the forcing function $f(t)$ is equal to zero, Eq. 1 is described as homogeneous, second order differential equation. If the forcing function $f(t)$ is not equals to zero, Eq. 1 is described as nonhomogeneous, second order differential equation. The nonhomogeneous differential equation corresponds to the case of forced vibration and the homogeneous differential equation corresponds to the case of free vibration. In the following sections we present methods for obtaining solutions for both homogeneous and nonhomogeneous differential equations.

2. Solution of homogeneous differential Equation with constant coefficients

In this section, techniques for solving linear, homogeneous, second order differential equations with constant coefficients are discussed. Whenever the right-hand side of Eq. 1 is identically zero, that is

$$f(t) = 0 \tag{2}$$

The equation is called a homogeneous differential equation. In this case Eq. 1 reduces to

$$m\ddot{x} + c\dot{x} + kx = 0 \tag{3}$$

By a solution of Eq. 3 we mean a function x(t) which, with its derivatives, satisfies the differential equation. A solution to Eq. 3 can be obtained by trial and error. A trial solution is to assume the function $x(t)$ in the following form

$$x(t) = Ae^{pt} \tag{4}$$

The general solution of the differential equation, provided that the roots of the differential equation are not equal, can be written as

$$x(t) = A_1 e^{p_1 t} + A_2 e^{p_2 t} \tag{5}$$

where

$$p_1 = \frac{-c + \sqrt{c^2 - 4mk}}{2m} \tag{6}$$

$$p_2 = \frac{-c - \sqrt{c^2 - 4mk}}{2m} \tag{7}$$

That is a complete solution of the second-order ordinary differential equations contains two arbitrary constants A_1 and A_2. These arbitrary constants can be determined from the initial conditions, as discussed in later sections.

Clearly the solution of the differential equation depends on the roots p_1 and p_2.

There are three different cases for the roots p_1 and p_2 as shown in Table 1.

Table 1.1. Different Cases of the solution of the second order homogeneous differential equation with constant coefficients.

Case 1	Case 2	Case 3
Overdamped System	Critically Damped System	Underdamped System
Real distinct roots	Repeated roots	Complex Conjugate Roots
p_1 and p_2 are real numbers and $p_1 \neq p_2$	p_1 and p_2 are real numbers and $p_1 = p_2$	p_1 and p_2 are complex conjugate numbers and $p_1 \neq p_2$
$c^2 > 4mk$	$c^2 = 4mk$	$c^2 < 4mk$
High damping Coefficient		Small damping Coefficient

The solution will be in the form of

Case 1	Case 2	Case 3
Overdamped System	Critically Damped System	Underdamped System
$x(t) = A_1 e^{p_1 t} + A_2 e^{p_2 t}$	$x(t) = (c_1 + c_2 t) e^{p_1 t}$	$x(t) = X e^{\alpha t} \sin{(\beta t + \phi)}$
A_1 and A_2 from initial conditions	c_1 and c_2 from initial conditions	X and ϕ from initial conditions
$p_1 = \frac{-c + \sqrt{c^2 - 4mk}}{2m}$	$p_1 = \frac{-c}{2m}$	$\alpha = -\frac{c}{2m}$
$p_2 = \frac{-c - \sqrt{c^2 - 4mk}}{2m}$		$\beta = \frac{1}{2m}\sqrt{4mk - c^2}$

If the initial Conditions are given as

$$x_o = x(t = 0), \quad v_o = \dot{x}(t = 0),$$

the coefficients A_1, A_2, c_1, c_2, X and ϕ will be calculated as follows

Case 1	Case 2	Case 3
Overdamped System	Critically Damped System	Underdamped System
$x(t) = A_1 e^{p_1 t} + A_2 e^{p_2 t}$	$x(t) = (c_1 + c_2 t) A_1 e^{p_1 t}$	$x(t) = X e^{\alpha t} \sin{(\beta t + \phi)}$
$A_1 = \frac{x_o p_2 - v_o}{p_2 - p_1}$	$c_1 = x_o$	$X = \sqrt{x_0^2 + \left(\frac{v_o - \alpha x_o}{\beta}\right)^2}$
$A_2 = \frac{v_o - p_1 x_o}{p_2 - p_1}$	$c_2 = v_o - p_1 x_o$	$\phi = \tan^{-1}\frac{\beta x_o}{v_o - \alpha x_o}$

Example 1.1

a. Find the solution of the following homogeneous second-order ordinary differential equation

$$\ddot{x} - 4\dot{x} + 3x = 0$$

b. If the initial conditions are $x_o = 2$ and $v_o = 0$, plot the response.

Solution.

a. m=1, *c*=-4, *k*=3

$$c^2 \quad ? \quad 4mk$$

$$16 > 4\times1\times3$$

Since $c^2 > 4mk$, the system is over damped.

$$p_1 = \frac{-c + \sqrt{c^2 - 4mk}}{2m}$$

$$p_1 = \frac{-(-4) + \sqrt{4^2 - 4\times1\times3}}{2\times1}$$

$$p_1 = 3$$

$$p_2 = \frac{-c - \sqrt{c^2 - 4mk}}{2m}$$

$$p_2 = \frac{-(-4) - \sqrt{4^2 - 4\times1\times3}}{2\times1}$$

$$p_2 = 1$$

The solution is

$$x(t) = A_1 e^{p_1 t} + A_2 e^{p_2 t}$$

$$x(t) = A_1 e^{3t} + A_2 e^{t}$$

b. The initial conditions are $x_o = 2$ and $v_o = 0$

$$A_1 = \frac{x_o p_2 - v_o}{p_2 - p_1}$$

$$A_1 = \frac{2\times 1 - 0}{1 - 3}$$

$$A_1 = -1$$

$$A_2 = \frac{v_o - p_1 x_o}{p_2 - p_1}$$

$$A_2 = \frac{0 - 3\times 2}{1 - 3}$$

$$A_2 = 3$$

The solution is

$$x(t) = A_1 e^{3t} + A_2 e^{t}$$

$$x(t) = -e^{3t} + 3e^{t}$$

The solution is shown in Fig. 1.1.

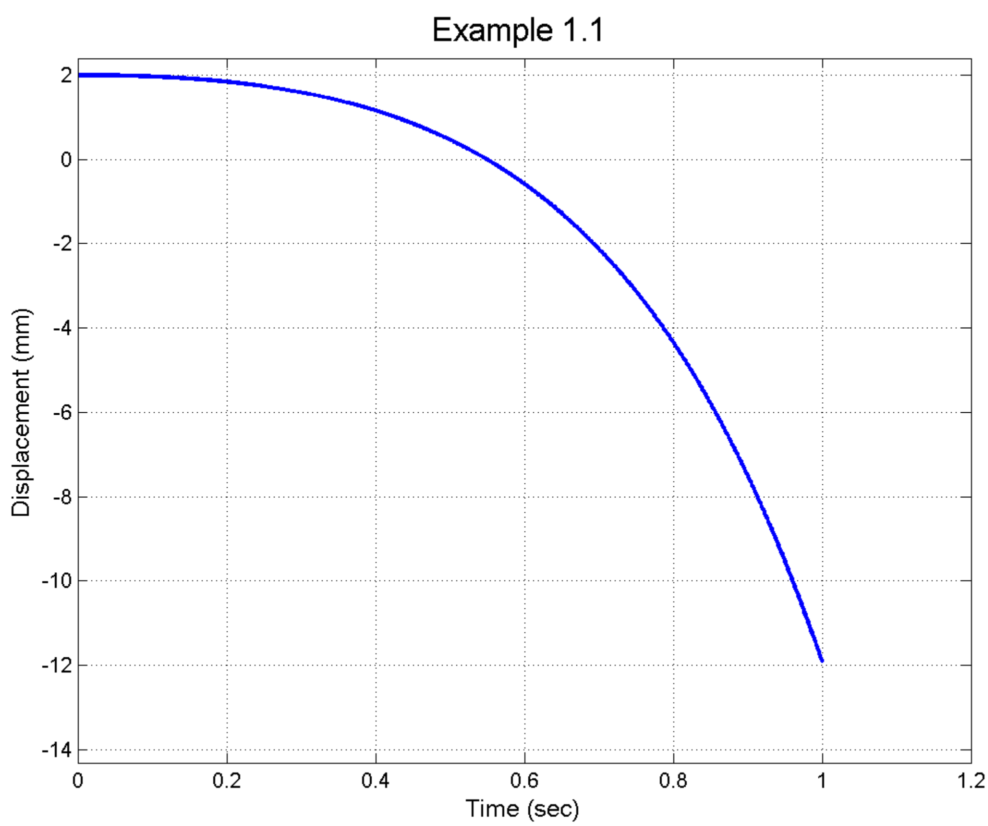

Fig. 1.1. Time response of example 1.1.

Matlab Code of Example 1.1:

Matlab Program 1.1

```
% Example 1.1
clear all; clf; clc
Tf=1; %Final time, sec
%time Step
dt=1e-3;
no_data_points=Tf/dt; %number of data points to plot
for i=1:no_data_points+1
t(i)=(i-1)*dt;
x(i)=-exp(3*t(i))+3*exp(t(i));
end
%plotting
figure(1);plot(t,x,'linewidth', 2);
xlabel('Time (sec)','FontSize',12);
ylabel('Displacement (mm)','FontSize',12);
axis([0 1.2*Tf 1.2*min(x) 1.2*max(x)])
grid on
title('Example 1.1 ','FontSize',16);
saveas(gcf, 'Example 1_1.tiff');
```

Example 1.2

a. Find the solution of the following homogeneous second-order ordinary differential equation

$$\ddot{x} + 6\dot{x} + 9x = 0$$

b. If the initial conditions are $x_o = 0$ and $v_o = 1$, plot the response.

Solution.

a. m=1, *c*=6, *k*=9

$$c^2 \quad ? \quad 4mk$$

$$36 = 4\times1\times9$$

Since $c^2 = 4mk$, the system is critically damped.

$$p_1 = \frac{-c}{2m} = \frac{-6}{2\times1} = -3$$

The solution is

$$x(t) = (c_1 + c_2 t)e^{p_1 t}$$

$$x(t) = (c_1 + c_2 t)e^{-3t}$$

b. The initial conditions are $x_o = 0$ and $v_o = 1$

$$c_1 = x_o = 0$$

$$c_2 = v_o - p_1 x_o = 1 - (-3)\times 0 = 1$$

The solution is

$$x(t) = (c_1 + c_2 t)e^{-3t}$$

$$x(t) = (0 + 1\times t)e^{-3t}$$

$$x(t) = te^{-3t}$$

The solution is shown in Fig. 1.2.

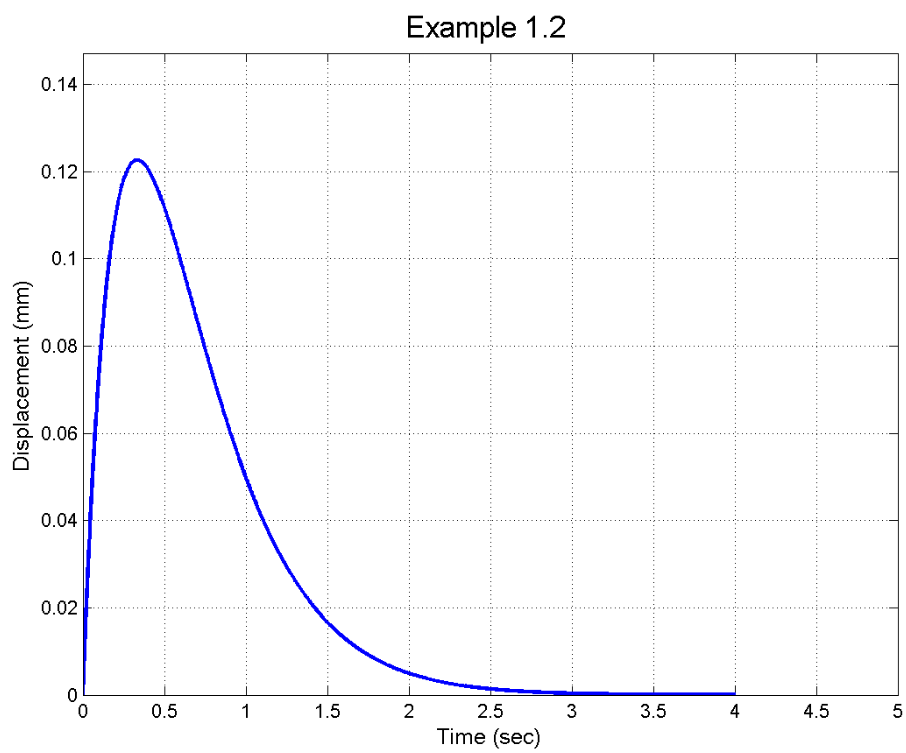

Fig. 1.2. Time response of example 1.2.

Matlab Code of Example 1.2:

Matlab Program 1.2

```
% Example 1.2
clear all; clf; clc
Tf=4; %Final time, sec
%time Step
dt=1e-3;
no_data_points=Tf/dt; %number of data points to plot
for i=1:no_data_points+1
t(i)=(i-1)*dt;
x(i)=t(i).*exp(-3*t(i));
end
%plotting
figure(1);plot(t,x,'linewidth', 2);
xlabel('Time (sec)','FontSize',12);
ylabel('Displacement (mm)','FontSize',12);
axis([0 Tf+1 .8*min(x) 1.2*max(x)])
grid on
title('Example 1.2 ','FontSize',16);
saveas(gcf, 'Example 1_2.tiff');
```

Example 1.3

a. Find the solution of the following homogeneous second-order ordinary differential equation

$$5\ddot{x} + 2\dot{x} + 50x = 0$$

b. If the initial conditions are $x_o = 0.01$ and $v_o = 3$, plot the response.

Solution.

a. m=5, *c*=2, *k*=50

$$c^2 \quad ? \quad 4mk$$

$$4 < 4{\times}5{\times}50$$

Since $c^2 < 4mk$, the system is underdamped.

$$\alpha = -\frac{c}{2m} = -\frac{2}{2{\times}5} = -0.2$$

$$\beta = \frac{1}{2m}\sqrt{4mk - c^2} = \frac{1}{2{\times}5}\sqrt{4{\times}5{\times}50 - 2^2} = 3.156$$

The solution is

$$x(t) = Xe^{\alpha t}\sin(\beta t + \phi)$$

$$x(t) = Xe^{-0.2t}\sin(3.156t + \phi)$$

b. The initial conditions are $x_o = 0.01$ and $v_o = 3$

$$X = \sqrt{x_0^2 + \left(\frac{v_o - \alpha x_o}{\beta}\right)^2}$$

$$X = \sqrt{0.01^2 + \left(\frac{3-(-0.2)\times 0.01}{3.156}\right)^2} = 0.9512$$

$$\phi = \tan^{-1}\frac{\beta x_o}{v_o - \alpha x_o}$$

$$\phi = \tan^{-1}\frac{3.156\times 0.01}{3-(-0.2)\times 0.01} = 0.6023°$$

The compete solution is then given by

$$x(t) = Xe^{-0.2t}\sin(3.156t + \phi)$$

$$x(t) = 0.9512e^{-0.2t}\sin(3.156t + 0.6023)$$

The solution is shown in Fig. 1.3.

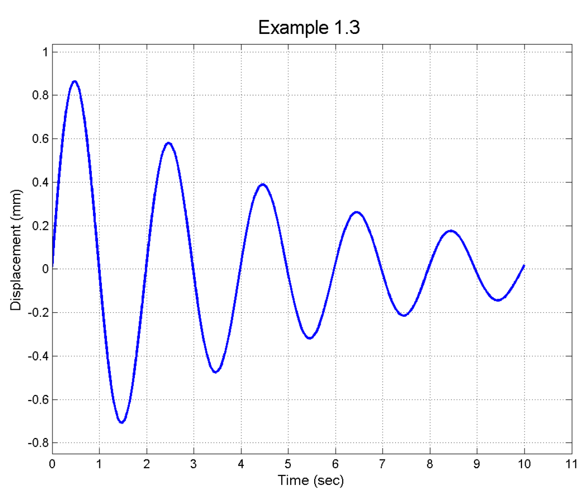

Fig. 1.3. Time response of example 1.3.

Matlab Code of Example 1.3:

Matlab Program 1.3

```
% Example 1.3
clear all; clf; clc
Tf=10; %Final time, sec
%time Step
dt=1e-3;
no_data_points=Tf/dt; %number of data points to plot
for i=1:no_data_points+1
t(i)=(i-1)*dt;
x(i)=0.9512*exp(-0.2*t(i))*sin(3.156*t(i)+.6023*pi/180);
end
%plotting
figure(1);plot(t,x,'linewidth', 2);
xlabel('Time (sec)','FontSize',12);
ylabel('Displacement (mm)','FontSize',12);
axis([0 Tf+1 1.2*min(x) 1.2*max(x)])
grid on
title('Example 1.3 ','FontSize',16);
saveas(gcf, 'Example 1_3.tiff');
```

General Matlab code for solving solve homogeneous second order differential equation of single degree of freedom vibratory system

Matlab Program 1.4

```
% General Program to plot the response of Second Order ODE
of a vibratory system
clear all; clf; clc
%Inputs
m=5; % mass
c=2; %damping
k=50; %stiffness
Tf=10; %Final time, sec
dt=1e-3; %time Step
%initial conditions
xo=0.01; % initial displacement
vo=3; % initial velocity
no_data_points=Tf/dt; %number of data points to plot
C= c^2-4*m*k;
% Overdamped System C^2>4mk
if C>0
p1= (-c+sqrt(c^2-4*m*k))/(2*m);
p2= (-c-sqrt(c^2-4*m*k))/(2*m);
A1=(xo*p2-vo)/(p2-p1);
```

```
A2=(vo-xo*p1)/(p2-p1);
for i=1:no_data_points+1
t(i)=(i-1)*dt;
x(i)=A1*exp(p1*t(i))+A2*exp(p2*t(i));
end
end
% Critically damped System C^2=4mk
if C==0
p1_1= (-c/(2*m));
c1=xo;
c2=vo-xo*p1_1;
for i=1:no_data_points+1
t(i)=(i-1)*dt;
x(i)=(c1+c2*t(i))*exp(p1_1*t(i));
end
end
% Undrdamped System C^2<4mk
if C<0
alpha= (-c)/(2*m);
beta= (sqrt(4*m*k-c^2))/(2*m);
X=sqrt(xo^2+((vo-alpha*xo)/beta)^2);
phi=atand((beta*xo)/(vo-alpha*xo));
for i=1:no_data_points+1
t(i)=(i-1)*dt;
x(i)=X*exp(alpha*t(i))*sin(beta*t(i)+phi*pi/180);
end
end
%plotting
figure(1);plot(t,x,'linewidth', 2);
xlabel('Time (sec)','FontSize',12);
ylabel('Displacement (mm)','FontSize',12);
axis([0 Tf+0.2 1.2*min(x) 1.2*max(x)])
grid on
```

Dsolve Command in matlab

`Dsolve` command in Matlab can be used to find the analytical solution of first and second order ODE as shown in the following two examples.

Example 1.4

By using Dsolve command in Matlab, solve the following ODE:

$$\dot{x}(t) = 2t - x, \qquad x(0) = 1$$

Matlab Program 1.5

```
% Example 1-4
X=dsolve('Dx-2*t+x=0','x(0)=1','t')
X=simple(X);
pretty(X)
ezplot(X,(0:0.05:0.3))
```

Matlab Output

```
X =

2*t + 3*exp(-t) - 2


  2 t + 3 exp(-t) - 2
```

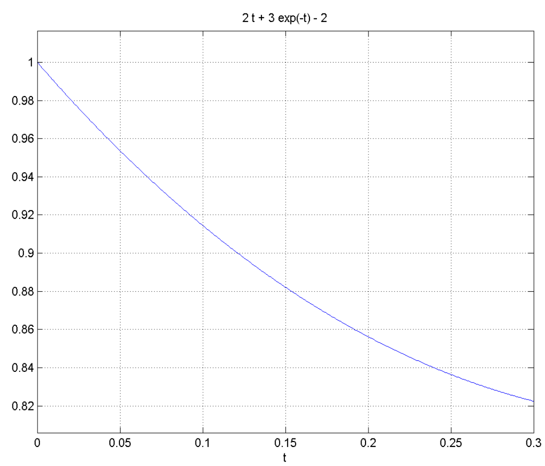

Fig. 1.4. Time response of example 1.4.

The exact solution is

$$x(t) = 3e^{-t} + 2t - 2$$

Example 1.5

By using Dsolve command in Matlab, solve the following ODE:

$$3\ddot{x} + \dot{x} + 2x = 0$$

subject to the initial conditions $x(0) = 0, \dot{x}(0) = 0.25$ over the time interval $0 \leq t \leq 20$ sec by using 4th order Runge-Kutta method.

Matlab Program 1.6

```
% Example 1-5
clc; clear all; clf;
tf=20;
```

```
X=dsolve('3*D2x+1*Dx+2*x=0','Dx(0)=0.25','x(0)=0','t');
X=simple(X)
pretty(X)
ezplot(X,(0:0.01:20))
grid on
saveas(gcf, 'Example 1_5.tiff');
```

Matlab Output

```
X =

(3*23^(1/2)*exp(-t/6)*sin((23^(1/2)*t)/6))/46

                          /    1/2    \
      1/2     /    t \    | 23     t |
  3 23    exp| - - | sin| -------  |
              \    6 /    \     6     /
  ---------------------------------
                   46
```

The exact solution is

$$x(t) = 0.31e^{-\frac{t}{6}}\ \sin\,(0.79t)$$

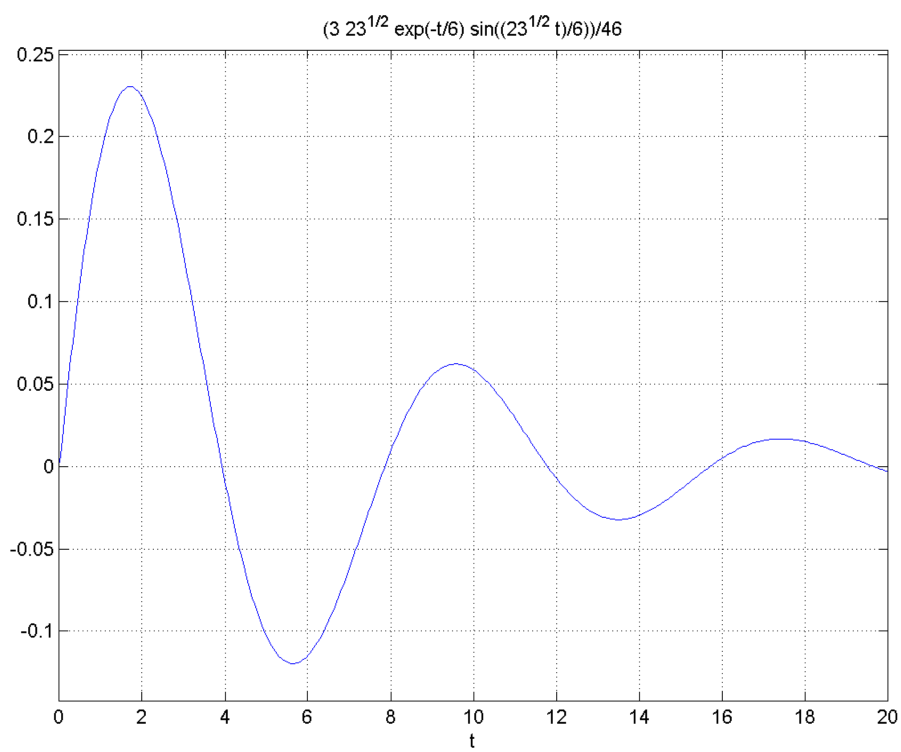

Fig. 1.5. Time response of example 1.5.

3. Solution of nonhomogeneous differential Equation with constant coefficients

The nonhomogeneous differential Equation with constant coefficients of a vibratory system is written as:

$$m\ddot{x} + c\dot{x} + kx = f(t) \tag{8}$$

where $f(t)$ is the forcing function. The solution of the equation consists of two parts. First the complementary solution x_c of Eq.8 where the right hand side is equal to zero, that is, $f(t) = 0$. Methods for obtaining the complementary solutions were discussed in the proceeding sections. The second part of the solution is the particular solution, x_p. The complete solution of Eq.8 can be written as

x =complementary solution + particular solution

$$x = x_c + x_p \tag{9}$$

where x_c is the solution of the equation

$$m\ddot{x}_c + c\dot{x}_c + kx_c = 0 \tag{10}$$

and x_p is the solution of the equation

$$m\ddot{x}_p + c\dot{x}_p + kx_p = f(t) \tag{11}$$

The particular solution x_p can be found by the method of undetermined coefficients.

The solution of different force functions is given below

1. For a constant Force, the forcing function will be

$$f(t) = F_0 \tag{12}$$

   and the particular solution, x_p will be

$$x_p = \frac{F_0}{k} \tag{13}$$

2. For a sinusoidal Force, the forcing function will be

$$f(t) = F_0 \ \sin(\omega t) \tag{14}$$

   and the particular solution, x_p will be

$$x_p = \frac{F_0}{\sqrt{(k - m\omega^2)^2 + (c\omega)^2}} \sin\left(wt - \left(tan^{-1}\left(\frac{c\omega}{k - m\omega^2}\right)\right)\right) \tag{15}$$

4. Numerical Simulation of the time response

The solution of the vibration problems is often plotted versus time in order to visualize the physical vibration and to obtain an idea of the nature of the response. For simple vibration problems, there is analytical (closed form solution for the displacement as a function of time) solution. However for real life problems, the equations are more complex and sometimes nonlinear that is difficult or impossible to solve analytically. The use of numerical solution (integration) greatly enhances the understanding of vibration. Just as a picture is worth a thousand words, a numerical simulation or plot can enable a completely dynamic understanding of vibration phenomena. Computer calculations and simulations are presented at the end of each chapter.

The free response of any system is usually computed by simple numerical means such as Euler's method, Heun's or Runge-Kutta methods. The basis of the numerical solutions of ordinary differential equations is to essentially undo calculus by representing each derivative by a small but finite difference. A numerical solution of an ordinary differential equation is a procedure for constructing approximate values: $x_1, x_2, \ldots, xn$, of the solution x(t) at the discrete values of time: $t_o < t_1 \ldots < t_n$. Effectively, a numerical procedure produces a list of discrete values $x_i = x(t_i)$ that approximates the solution, as shown in Fig.1, rather than a continuous function x(t), which is the exact solution.

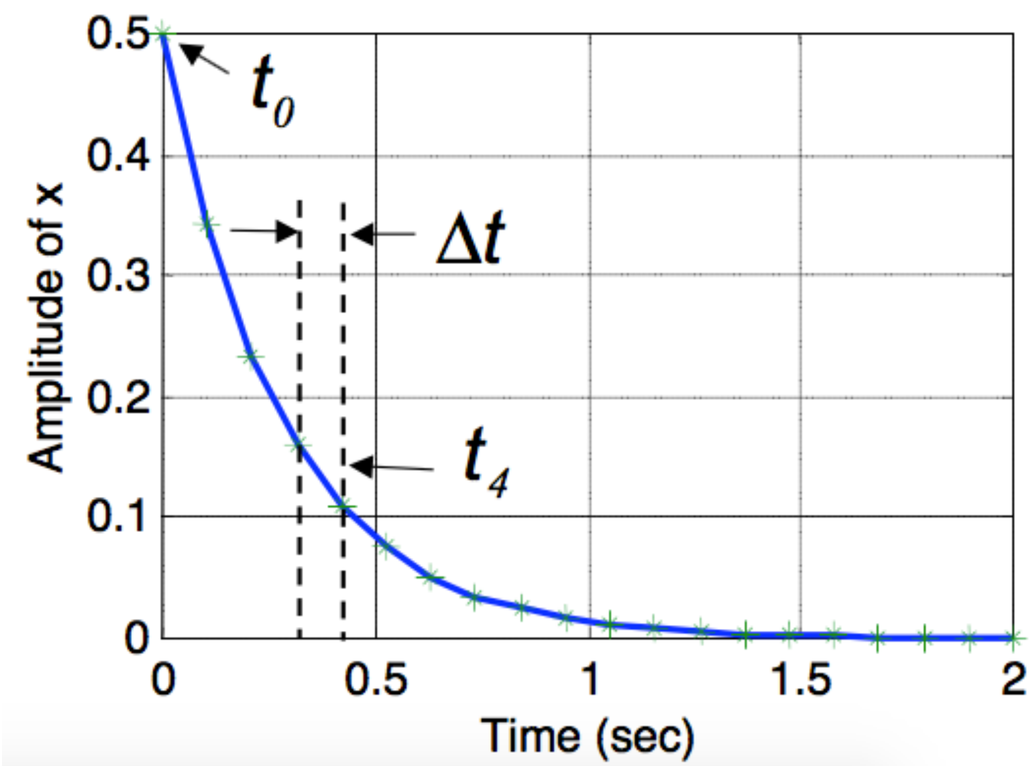

Fig. 1.6. Time discretization of time response.

For a single degree of freedom system of the form

$$m\ddot{x} + c\dot{x} + kx = 0 \qquad x(0) = x_o \qquad v(0) = v_o \tag{16}$$

the initial values x_o (initial displacement) and v_o (initial velocity) form the first two points of the numerical solution. The equation will be solved for values of time t between $t = 0$ and $t = T_f$, where T_f is the total length of time over which the solution is of interest. The time interval $T_f - 0$ is then divided into n intervals (so that $\Delta t = T_f/n$). Then Eq.16 is calculated at the values of $t_o = 0,\ t_1 = \Delta t,\ t_2 = 2\Delta t, \dots,\ t_n = n\Delta t = T_f$ to produce an approximate representation, or simulation, of the solution.
The concept of a numerical solution is easiest to grasp by first examining the numerical solution of a first order scalar differential equation. To this end consider the first order differential equation

$$\dot{x}(t) = f(x,t), \quad x(0) = x_o \tag{17}$$

5. Euler's Method for first order ODE
The Euler's method proceeds from the definition of the slope form of the derivative at $t = t_i$ is

$$\dot{x}(t_i) = \dot{x}_i = \frac{dx(t_i)}{dt} = \lim_{\Delta t=0}\frac{x(t_{i+1}) - x(t_i)}{\Delta t} = \frac{x_{i+1} - x_i}{\Delta t} = f(t_i, x_i)$$

$$f(t_i, x_i) = A_i$$

where A_i is the derivative at time t_i and $x(t_i) = x_i$

$$\frac{x_{i+1} - x_i}{\Delta t} = A_i$$
$$x_{i+1} = x_i + \Delta t A_i$$

Euler's Equations

$$\begin{aligned} t_{i+1} &= t_i + \Delta t \\ A_i &= f(t_i, x_i) \\ x_{i+1} &= x_i + \Delta t A_i \end{aligned} \tag{18}$$

Example 2.1

By using Euler's method, solve the following ODE:

$$\dot{x}(t) = 5x, \qquad x(0) = 1, \qquad \Delta t = 0.1$$

Solution

$$x_o = 1 \quad , \quad t_o = 0$$

$$t_{i+1} = t_i + \Delta t = t_i + 0.1$$

$$A_i = f(t_i, x_i) = 5x_i$$

$$x_{i+1} = x_i + \Delta t A_i$$

For i=0:

$$t_1 = t_0 + \Delta t = 0 + 0.1 = 0.1$$

$$A_0 = f(t_0, x_0) = 5x_0 = 5\times 1 = 5$$

$$x_1 = x_0 + \Delta t A_0 = 1 + 0.1\times 5 = 1.5$$

For i=1:

$$t_2 = t_1 + \Delta t = 0.1 + 0.1 = 0.2$$

$$A_1 = f(t_1, x_1) = 5x_1 = 5\times 1.5 = 7.5$$

$$x_2 = x_1 + \Delta t A_1 = 1.5 + 0.1\times 7.5 = 2.15$$

Iteration, i	t_i	x_i	A_i	$\Delta t A_i$
0	0	1	$5\times 1 = 5$	$0.1\times 5 = 0.5$
1	0.1	$1 + 0.5 = 1.5$	$5\times 1.5 = 7.5$	$0.1\times 7.5 = 0.75$
2	0.2	$1.5 + 0.75 = 2.25$		

Exact solution can be found by using the following Matlab program

Matlab Program 1.7

```
X=dsolve('Dx-5*x=0','x(0)=1','t')
X=simple(X);
pretty(X)
```

Te exact solution is:

$$x(t) = e^{5t}$$

A comparison between Euler method numerical solution and the exact solution is shown in Fig. 1.7.

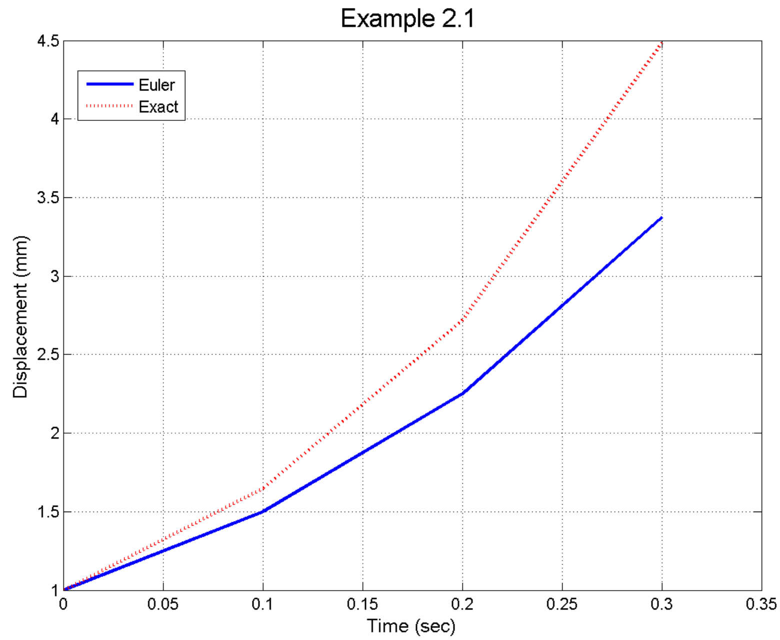

Fig. 1.7. Time response of example 2.1.

Matlab Code for Example 2.1:

Matlab Program 1.8

```
% Euler Method Program to plot the response of first Order
ODE
% Example 2_1
clear all; clf; clc
% xdot=ax+bt+c
%Inputs
a=5;
b=0; %damping
c=0; %stiffness
Tf=10; %Final time, sec
dt=1e-1; %time Step
%initial conditions
x(1)=1; % initial displacement
x_exact(1)=x(1);
no_data_points=Tf/dt; %number of data points to plot
% for i=1:no_data_points+1
for i=1:3
t(i)=(i-1)*dt;
A(i)= a*x(i)+b*t(i)+c;
x(i+1)=x(i)+A(i)*dt;
```

```
x_exact(i+1)=1*exp(a*(t(i)+dt));
end
t(i+1)=(i)*dt;
%plotting
figure(1);plot(t,x,'linewidth', 2);
hold on
plot(t,x_exact,'r:','linewidth', 2);
xlabel('Time (sec)','FontSize',12);
ylabel('Displacement (mm)','FontSize',12);
% axis([0 Tf+0.2 1.2*min(x) 1.2*max(x)])
grid on
legend('Euler', 'Exact','location', 'best')
title('Example 2.1 ','FontSize',16);
saveas(gcf, 'Example 2_1.tiff');
```

Example 2.2

By using Euler's method, solve the following ODE:

$$\dot{x}(t) = 2t - x, \qquad x(0) = 1, \qquad \Delta t = 0.1$$

Solution

$$x_o = 1 \quad , \quad t_o = 0$$

$$t_{i+1} = t_i + \Delta t = t_i + 0.1$$

$$A_i = f(t_i, x_i) = 2t_i - x_i$$

$$x_{i+1} = x_i + \Delta t A_i$$

For i=0:

$$t_1 = t_0 + \Delta t = 0 + 0.1 = 0.1$$

$$A_0 = f(t_0, x_0) = 2t_0 - x_0 = 2\times 0 - 1 = -1$$

$$x_1 = x_0 + \Delta t A_0 = 1 + 0.1\times(-1) = 0.9$$

For i=1

$$t_2 = t_1 + \Delta t = 0.1 + 0.1 = 0.2$$

$$A_1 = f(t_1, x_1) = 2t_1 - x_1 = 2\times 0.1 - 0.9 = -0.7$$

$$x_2 = x_1 + \Delta t A_1 = 0.9 + 0.1\times(-0.7) = 0.83$$

i	t_i	x_i	A_i	$\Delta t A_i$
0	0	1	-1	-0.1
1	0.1	0.9	-0.7	-0.07
2	0.2	0.83		

It was shown in example 1.4 that the exact solution of this differential equation is

$$x(t) = 3e^{-t} + 2t - 2$$

A comparison between Heun method numerical solution and the exact solution is shown in Fig. 1.8.

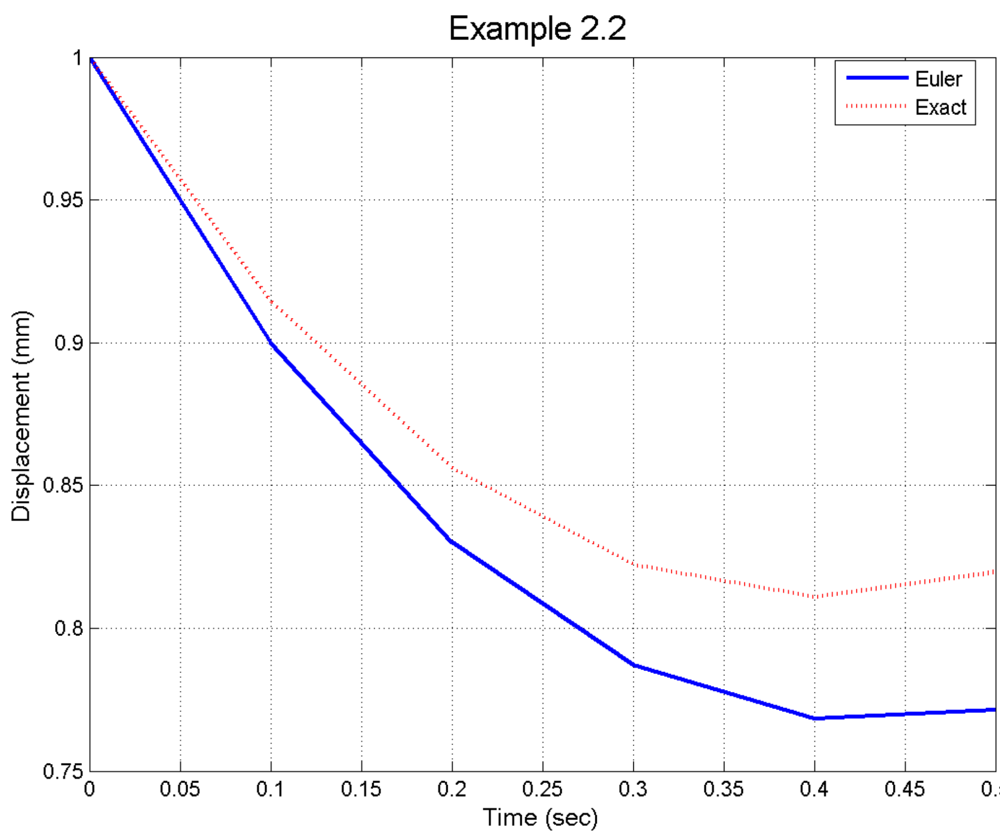

Fig. 1.8. Time response of example 2.2.

Matlab Code for Example 2.2:

Matlab Program 1.10

```
% Euler Method Program to plot the response of first Order
ODE
% Example 2_2
```

```
clear all; clf; clc
% xdot=ax+bt+c
%Inputs
a=-1;
b=2;
c=0;
Tf=0.5; %Final time, sec
dt=1e-1; %time step
%initial conditions
x(1)=1; % initial displacement
x_exact(1)=x(1);
error(1)=abs(x(1)-x_exact(1))/(x_exact(1))*100;
no_data_points=Tf/dt; %number of data points to plot
for i=1:no_data_points
t(i)=(i-1)*dt;
A(i)= a*x(i)+b*t(i)+c;
x(i+1)=x(i)+A(i)*dt;
x_exact(i+1)=3*exp(-(t(i)+dt))+2*(t(i)+dt)-2;
error(i+1)=abs(x_exact(i+1)-x(i+1))/(x_exact(i+1))*100;
end
t(i+1)=(i)*dt;
%plotting
figure(1);plot(t,x,'linewidth', 2);
hold on
figure(1);plot(t,x_exact,'r:','linewidth', 2);
xlabel('Time (sec)','FontSize',12);
ylabel('Displacement (mm)','FontSize',12);
grid on
legend('Euler', 'Exact','location', 'best')
title('Example 2.2 ','FontSize',16);
saveas(gcf, 'Example 2_2.tiff');
```

Is Euler's method accurate?

Euler is a first order method that assumes that the slope is constant in the time step and uses the slope at the beginning. The error in Euler method is first order error and is related of the time step. That means if you halve the time step the error will halve. So by decreasing the time step, the error will decrease, but there are better methods of numerical integration with a higher order error. The better method will find a better slope.

6. Heun's (Modified Euler's) Method for first order ODE

This method calculates two slopes, the slope at the beginning and end of the time step. Then by averaging the two slopes we will get a better slope than Euler method.

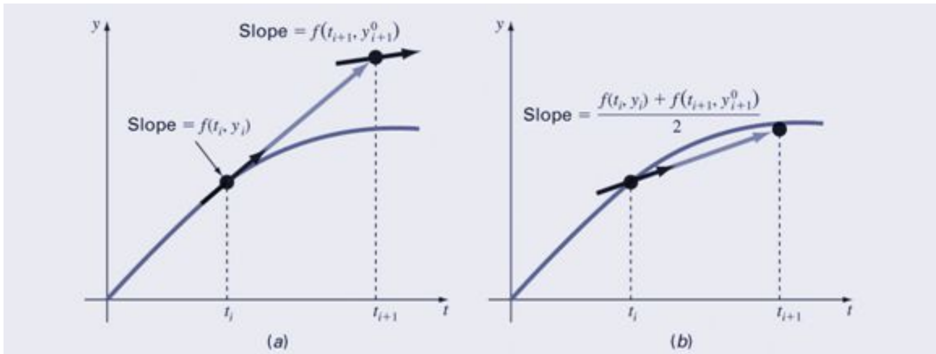

Fig. 1.9. Heun numerical integration method.

Modified Euler (Heun) Equations

$$t_{i+1} = t_i + \Delta t$$

$$B_i = f(t_i, x_i)$$

$$\tilde{x}_{i+1} = x_i + \Delta t B_i$$

$$C_i = f(t_{i+1}, \tilde{x}_{i+1}) \tag{19}$$

$$A_i = \frac{B_i + C_i}{2}$$

$$x_{i+1} = x_i + \Delta t A_i$$

The condensed form of Modified Euler (Heun) Equations is:

$$t_{i+1} = t_i + \Delta t$$

$$\tilde{x}_{i+1} = x_i + \Delta t f(t_i, x_i)$$

$$x_{i+1} = x_i + \Delta t \left(\frac{f(t_i, x_i) + f(t_{i+1}, \tilde{x}_{i+1})}{2} \right)$$

Example 2.3

By using Heun's method solve the following ODE:

$$\dot{x}(t) = 2t - x, \qquad x(0) = 1, \qquad \Delta t = 0.1$$

Solution

$$x_o = 1 \quad , \quad t_o = 0$$

$$t_{i+1} = t_i + \Delta t = t_i + 0.1$$

$$B_i = f(t_i, x_i) = 2t_i - x_i$$

$$\tilde{x}_{i+1} = x_i + \Delta t B_i$$

$$C_i = f(t_{i+1}, \tilde{x}_{i+1}) = 2t_{i+1} - \tilde{x}_{i+1}$$

$$A_i = \frac{B_i + C_i}{2}$$

$$x_{i+1} = x_i + \Delta t A_i$$

For i=0:

$$t_1 = t_0 + \Delta t = 0 + 0.1 = 0.1$$

$$B_0 = f(t_0, x_0) = 2t_0 - x_0 = 2\times 0 - 1 = -1$$

$$\tilde{x}_1 = x_0 + \Delta t B_0 = 1 + 0.1\times(-1) = 0.9$$

$$C_0 = f(t_1, \tilde{x}_1) = 2t_1 - \tilde{x}_1 = 2\times 0.1 - 0.9 = -0.7$$

$$A_0 = \frac{B_0 + C_0}{2} = \frac{-1 + (-0.7)}{2} = -0.85$$

$$x_1 = x_0 + \Delta t A_0 = 1 + 0.1\times(-0.85) = 0.915$$

For i=1

$$t_2 = t_1 + \Delta t = 0.1 + 0.1 = 0.2$$

$$B_1 = f(t_1, x_1) = 2t_1 - x_1 = 2\times 0.1 - 0.915 = -0.715$$

$$\tilde{x}_2 = x_1 + \Delta t B_1 = 0.915 + 0.1\times(-0.7) = 0.845$$

$$C_1 = f(t_2, \tilde{x}_2) = 2t_2 - \tilde{x}_2 = 2\times 0.2 - 0.845 = -0.445$$

$$A_1 = \frac{B_1 + C_1}{2} = \frac{-0.715 + (-0.445)}{2} = -0.58$$

$$x_2 = x_1 + \Delta t A_1 = 0.915 + 0.1\times(-0.58) = 0.857$$

In a tabular form

Iteration, i	t_i	x_i	B_i	$\tilde{x}_{i+1}$	C_i	A_i	$\Delta t A_i$
0	0	1	-1	0.9	-0.7	-.85	-0.085
1	0.1	0.915	-0.715	0.845	-0.445	-.058	-.0058
2	0.2	0.857					

A comparison between 4th order Runge-Kutta method numerical solution and the exact solution is shown in Fig. 1.10.

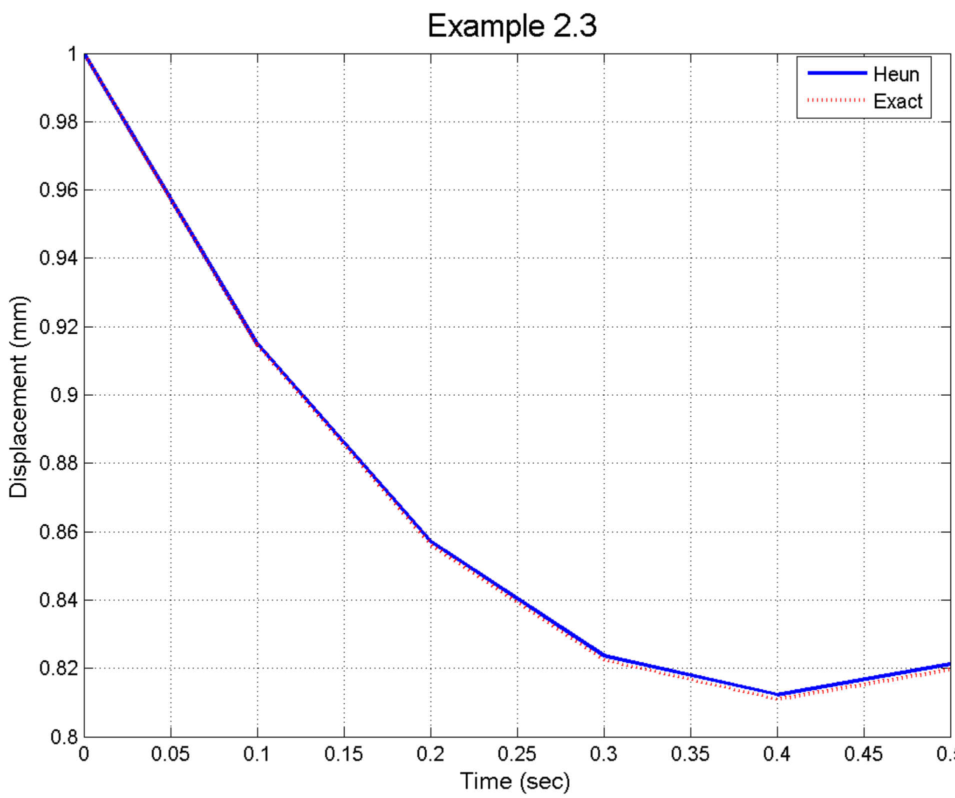

Fig. 1.10. Time response of example 2.3.

Matlab Code for Example 2.3:

Matlab Program 1.11

```
% Heun Method (Modified Euler) Program to plot the response
of first Order ODE
% Example 2_3
clear all; clf; clc
% xdot=ax+bt+c
%Inputs
a=-1;
```

```
b=2;
c=0;
Tf=0.5; %Final time, sec
dt=1e-1; %time Step
%initial conditions
x(1)=1; % initial displacement
x_exact(1)=x(1);
error(1)=abs(x(1)-x_exact(1))/(x_exact(1))*100;
no_data_points=Tf/dt; %number of data points to plot
for i=1:no_data_points
t(i)=(i-1)*dt;
B(i)= a*x(i)+b*t(i)+c;
x_1=x(i)+B(i)*dt;
C(i)= a*x_1+b*(t(i)+dt)+c;
A(i)= (B(i)+C(i))/2;
x(i+1)=x(i)+A(i)*dt;
x_exact(i+1)=3*exp(-(t(i)+dt))+2*(t(i)+dt)-2;
error(i+1)=abs(x_exact(i+1)-x(i+1))/(x_exact(i+1))*100;
end
t(i+1)=(i)*dt;
%plotting
figure(1);plot(t,x,'linewidth', 2);
hold on
figure(1);plot(t,x_exact,'r:','linewidth', 2);
xlabel('Time (sec)','FontSize',12);
ylabel('Displacement (mm)','FontSize',12);
grid on
legend('Heun', 'Exact','location', 'best')
title('Example 2.3 ','FontSize',16);
saveas(gcf, 'Example 2_3.tiff');
```

7. 4th order Runge-Kutta Method for first order ODE

Euler's method and the improved Euler's method are the simplest examples of a whole family of numerical methods to approximate the solutions of differential equations called Runge-Kutta methods. In this section we will give third and fourth order Runge-Kutta methods and discuss how Runge-Kutta methods are developed. Euler's method and the improved Euler's method both try to approximate Euler's method approximates the slope of the secant line by the slope of the tangent line at the left endpoint

The improved Euler's method uses the average of the slopes at the left endpoint and the approximate right endpoint (that is the right endpoint as computed by Euler's method) to approximate the slope of the secant line. We don't have to stop there either. We can keep finding slopes at different points and computing weighted averages to approximate the slope of the tangent line. Numerical methods to approximate the solution of differential equations in this fashion are called Runge-Kutta methods (after the mathematicians Runge and Kutta).

This method calculates four slopes, the slope at the beginning, middle and end of the time step. Then by using the weighted average of the four slopes we will get a better slope than Euler method.
By considering the following first order ODE:

$$\dot{x}(t) = f(x,t), \quad x(0) = x_o \tag{20}$$

$$x_{i+1} = x_i + \tfrac{\Delta t}{6}\left(S_{i,1} + 2S_{i,2} + S_{i,3} + S_{i,4}\right) \tag{21}$$

where

$$S_{i,1} = f(t_i, x_i)$$
$$S_{i,2} = f\left(t_i + \frac{\Delta t}{2}, x_i + \frac{\Delta t}{2} S_{i,1}\right)$$
$$S_{i,3} = f\left(t_i + \frac{\Delta t}{2}, x_i + \frac{\Delta t}{2} S_{i,2}\right)$$
$$S_{i,4} = f\left(t_i + \Delta t, x_i + \Delta t S_{i,3}\right)$$

4th order Runge-Kutta Equations:

$$\begin{gathered}
t_{i+1} = t_i + \Delta t \\
S_{i,1} = f(t_i, x_i) \\
\tilde{x}_{i+1,1} = x_i + \Delta t/2 \times S_{i,1} \\
S_{i,2} = f\left(t_i + \Delta t/2\,, \tilde{x}_{i+1,1}\right) \\
\tilde{x}_{i+1,2} = x_i + \Delta t/2 \times S_{i,2} \\
S_{i,3} = f\left(t_i + \Delta t/2\,, \tilde{x}_{i+1,2}\right) \\
\tilde{x}_{i+1,3} = x_i + \Delta t \times S_{i,3} \\
S_{i,4} = f\left(t_i + \Delta t, \tilde{x}_{i+1,3}\right) \\
Q_i = \frac{S_{i,1} + 2S_{i,2} + 2S_{i,3} + S_{i,4}}{6} \\
x_{i+1} = x_i + \Delta t Q_i
\end{gathered} \tag{22}$$

An example of the first calculation is shown below for i=0

$$t_1 = t_0 + \Delta t$$

$$S_{0,1} = f(t_0, x_0)$$

$$\tilde{x}_{1,1} = x_0 + \Delta t/2 \times S_{0,1}$$

$$S_{0,2} = f\left(t_0 + \Delta t/2\,, \tilde{x}_{1,1}\right)$$

$$\tilde{x}_{1,2} = x_0 + \Delta t/2 \times S_{0,2}$$

$$S_{0,3} = f\left(t_0 + \Delta t/2\,, \tilde{x}_{1,2}\right)$$

$$\tilde{x}_{1,3} = x_i + \Delta t \times S_{0,3}$$

$$S_{0,4} = f\left(t_i + \Delta t, \tilde{x}_{1,3}\right)$$

$$Q_0 = \frac{S_{0,1} + 2S_{0,2} + 2S_{0,3} + S_{0,4}}{6}$$

$$x_1 = x_0 + \Delta t Q_0$$

The solution was computed by using Matlab software and the response is shown in Fig. 1.11.

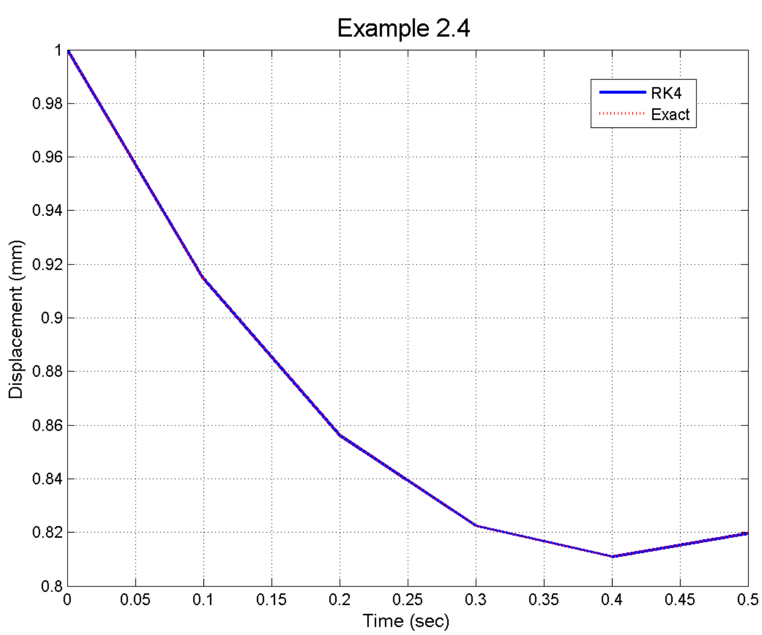

Fig. 1.11. Time response of example 2.4.

Matlab Code for Example 2.4:

Matlab Program 1.12

```
% Runge-Kutta 4th order Program to plot the response of
first Order ODE
% Example 2_4
clear all; clf; clc
% xdot=ax+bt+c
%Inputs
a=-1;
b=2;
c=0;
Tf=0.5; %Final time, sec
dt=1e-1; %time Step
%initial conditions
x(1)=1; % initial displacement
x_exact(1)=x(1);
error(1)=abs(x(1)-x_exact(1))/(x_exact(1))*100;
no_data_points=Tf/dt; %number of data points to plot
for i=1:no_data_points
t(i)=(i-1)*dt;
S1(i)= a*x(i)+b*t(i)+c;
x_1=x(i)+S1(i)*dt/2;
S2(i)= a*x_1+b*(t(i)+dt/2)+c;
x_2=x(i)+S2(i)*dt/2;
S3(i)= a*x_2+b*(t(i)+dt/2)+c;
x_3=x(i)+S3(i)*dt;
S4(i)= a*x_3+b*(t(i)+dt)+c;
Q(i)= (S1(i)+2*S2(i)+2*S3(i)+S4(i))/6;
x(i+1)=x(i)+Q(i)*dt;
x_exact(i+1)=3*exp(-(t(i)+dt))+2*(t(i)+dt)-2;
error(i+1)=abs(x_exact(i+1)-x(i+1))/(x_exact(i+1))*100;
end
t(i+1)=(i)*dt;
%plotting
figure(1);plot(t,x,'linewidth', 2);
hold on
figure(1);plot(t,x_exact,'r:','linewidth', 2);
xlabel('Time (sec)','FontSize',12);
ylabel('Displacement (mm)','FontSize',12);
grid on
legend('RK4', 'Exact','location', 'best')
title('Example 2.4 ','FontSize',16);
saveas(gcf, 'Example 2_4.tiff');
```

Comparison of the numerical integration methods

Iteration	Euler	Heun	RK4	Exact
1	1	1	1	1
2	0.9	0.915	0.914513	0.914512
3	0.83	0.857075	0.856193	0.856192
4	0.787	0.823653	0.822455	0.822455
5	0.7683	0.812406	0.810961	0.81096

The error percentage for each method is show in the following table

Iteration	Euler	Heun	RK4
1	0	0	0
2	1.586885	0.053334	2.69E-05
3	3.059156	0.103101	5.20E-05
4	4.310835	0.145687	7.34E-05
5	5.260448	0.178272	8.98E-05

The time response plot for each method is shown in the Fig. 1.12.

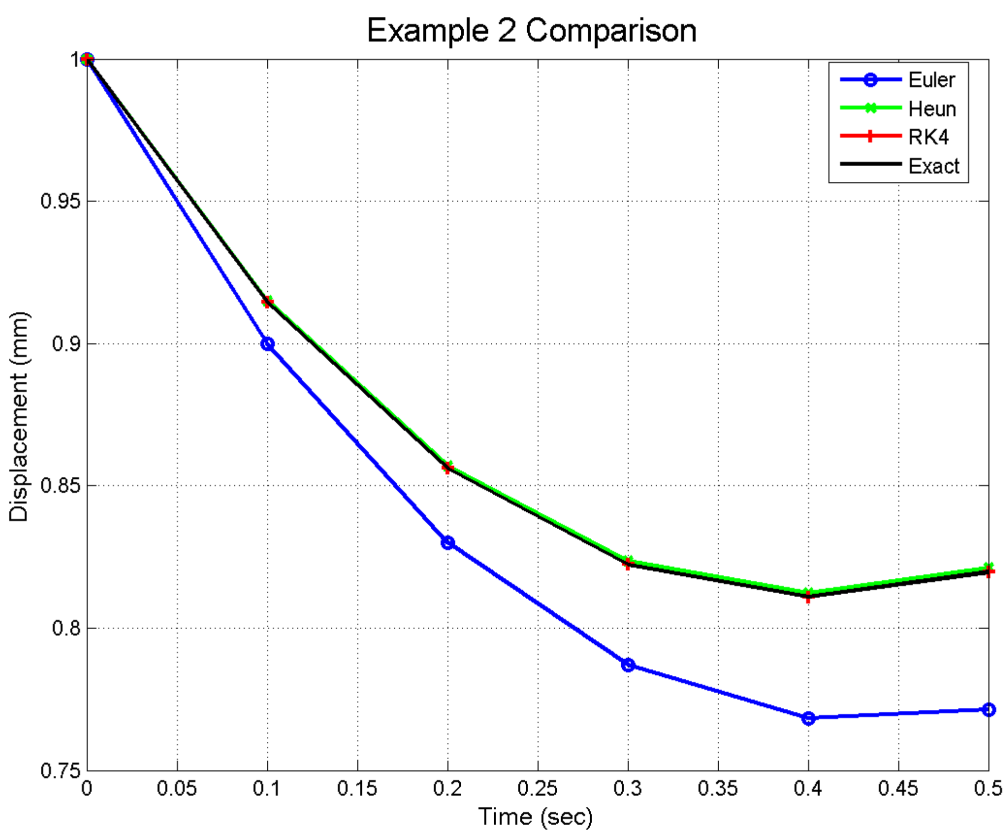

Fig. 1.12. Comparison mf the time response for Euler, Heun's, 4th order Runge-Kutta and exact solution.

8. Numerical Integration of second order DOE

The Euler's, Heun's, Runge-Kutta methods can be applied to first order systems only. So that it will be necessary to convert the second order vibration equation into two first order equations. The non-homogeneous differential equation with constant coefficients of a vibratory system is written as:

$$m\ddot{x} + c\dot{x} + kx = 0 \qquad x(0) = x_o \qquad v(0) = v_o \tag{23}$$

To achieve this, new variables y_1 and y_2 are defined as follows

$$y_1 = x(t) \; and \; y_2 = \dot{x}(t)$$

Hence

$$x = y_1 \qquad \dot{x} = y_2 = \dot{y}_1 \qquad \ddot{x} = \dot{y}_2$$

Substitute into Eq.23

$$m\dot{y}_2 + cy_2 + ky_1 = f(t)$$

$$\dot{y}_2 = -\frac{c}{m}y_2 - \frac{k}{m}y_1 + \frac{f(t)}{m}$$

Therefore, we are going to solve the next two first order equations:

$$\dot{y}_1 = y_2$$

$$\dot{y}_2 = -\frac{c}{m}y_2 - \frac{k}{m}y_1 + f(t) \tag{24}$$

In matrix form:

$$\begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{f(t)}{m} \end{bmatrix}$$

Equation 24 can be written as

$$\dot{y} = Ay + B(t) \tag{25}$$

$$y(0) = \begin{bmatrix} y_1(0) \\ y_2(0) \end{bmatrix} = \begin{bmatrix} x(0) \\ \dot{x}(0) \end{bmatrix} = \begin{bmatrix} x_o \\ v_o \end{bmatrix}$$

where

$$A = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix}, B = \begin{bmatrix} 0 \\ \frac{f(t)}{m} \end{bmatrix}$$

$$\dot{y} = \begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} \quad y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} x \\ \dot{x} \end{bmatrix}$$

The matrix A defined in this way is called the state matrix and the vector y is called the state vector. The position y_1 and the velocity y_1 are called the state variables.
Now the Euler and Range-Kutta methods of numerical solution can be applied to Eq.25 as follows:

Euler Method

$$t_{i+1} = t_i + \Delta t$$

$$S_i = Ay(i) + B(t) \tag{26}$$

$$y(i+1) = y(i) + \Delta t S_i$$

Runge-Kutta Method

$$t_{i+1} = t_i + \Delta t$$

$$S_{i,1} = \Delta t \times f\big(t_i, y(i)\big)$$

$$f\big(t_i, y(i)\big) = Ay(i) + B(t_i)$$

$$\tilde{y}_1(i+1) = y(i) + S_{i,1}/2$$

$$S_{i,2} = f\big(t_i + \Delta t/2\,, \tilde{y}_1(i+1)\big)$$

$$f\big(t_i + \Delta t/2\,, \tilde{y}_1(i+1)\big) = A\tilde{y}_1(i+1) + B(t_i + \Delta t/2)$$

$$\tilde{y}_2(i+1) = y(i) + S_2/2$$

$$S_{i,3} = f\big(t_i + \Delta t/2\,, \tilde{y}_2(i+1)\big) \tag{27}$$

$$f\big(t_i + \Delta t/2\,, \tilde{y}_2(i+1)\big) = A\tilde{y}_2(i+1) + B(t_i + \Delta t/2)$$

$$\tilde{y}_3(i+1) = y(i) + S_{i,3}$$

$$S_{i,4} = f\big(t_i + \Delta t, \tilde{y}_3(i+1)\big)$$

$$f\big(t_i + \Delta t, \tilde{y}_3(i+1)\big) = A\tilde{y}_3(i+1) + B(t_i + \Delta t)$$

$$Q_i = \frac{S_{i,1} + 2S_{i,2} + 2S_{i,3} + S_{i,4}}{6}$$

$$y(i+1) = y(i) + Q_i$$

Example 3.1

Plot the response of $3\ddot{x} + \dot{x} + 2x = 0$ subject to the initial conditions $x(0) = 0$, $\dot{x}(0) = 0.25$ over the time interval $0 \le t \le 20$ sec by using 4th order Runge-Kutta method.

Solution

The first step is to write the equation of motion in first-order form:

$$3\ddot{x} + \dot{x} + 2x = 0$$

$$\ddot{x} = -\frac{1}{3}\dot{x} - \frac{2}{3}x$$

Let

$$y_1 = x(t) \ and \ y_2 = \dot{x}(t)$$

Hence

$$\dot{y}_1 = y_2 = \dot{x}$$

$$\dot{y}_2 = \ddot{x} = -\frac{1}{3}\dot{x} - \frac{2}{3}x$$

$$\dot{y}_1 = y_2 = \dot{x}$$

$$\dot{y}_2 = \ddot{x} = -\frac{1}{3}\dot{x} - \frac{2}{3}x \tag{28}$$

$$\dot{y} = \begin{bmatrix}\dot{y}_1\\ \dot{y}_2\end{bmatrix} = \begin{bmatrix}\dot{x}\\ \ddot{x}\end{bmatrix} \quad y = \begin{bmatrix}y_1\\ y_2\end{bmatrix} = \begin{bmatrix}x\\ \dot{x}\end{bmatrix}$$

In matrix form:

$$\begin{bmatrix}\dot{y}_1\\ \dot{y}_2\end{bmatrix} = \begin{bmatrix}0 & 1\\ -\frac{2}{3} & -\frac{1}{3}\end{bmatrix}\begin{bmatrix}y_1\\ y_2\end{bmatrix} + \begin{bmatrix}0\\ 0\end{bmatrix}$$

Eq. 28 can be written as

$$\dot{y} = Ay + B(t)$$

$$y(0) = \begin{bmatrix}y_1(0)\\ y_2(0)\end{bmatrix} = \begin{bmatrix}x(0)\\ \dot{x}(0)\end{bmatrix} = \begin{bmatrix}x_o\\ v_o\end{bmatrix} = \begin{bmatrix}0\\ 0.25\end{bmatrix}$$

where

$$A = \begin{bmatrix} 0 & 1 \\ -\frac{2}{3} & -\frac{1}{3} \end{bmatrix}, B = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Numerical iterations:
Assume $\Delta t = 0.01$

For $i = 0$:

$$t_1 = t_0 + \Delta t = 0 + 0.01 = 0.01$$

$$S_{i,1} = \Delta t \times f\big(t_0, y(0)\big) = \Delta t \times f\big(0, y(0)\big)$$

$$f\big(0, y(0)\big) = \ Ay(0) + B(0)$$

$$f\big(0, y(0)\big) = \begin{bmatrix} 0 & 1 \\ -\frac{2}{3} & -\frac{1}{3} \end{bmatrix} y(0) + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$y(0) = \begin{bmatrix} y_1(0) \\ y_2(0) \end{bmatrix} = \ \begin{bmatrix} x_o \\ v_o \end{bmatrix} = \begin{bmatrix} 0 \\ 0.25 \end{bmatrix}$$

$$f\big(0, y(0)\big) = \begin{bmatrix} 0 & 1 \\ -\frac{2}{3} & -\frac{1}{3} \end{bmatrix} \begin{bmatrix} 0 \\ 0.25 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$f\big(0, y(0)\big) = \begin{bmatrix} 0.25 \\ -0.083 \end{bmatrix}$$

$$S_{i,1} = \Delta t \times f\big(0, y(0)\big) = 0.01 \times \begin{bmatrix} 0.25 \\ -0.083 \end{bmatrix} = \begin{bmatrix} 0.0025 \\ -0.00083 \end{bmatrix}$$

$$\tilde{y}_1(1) = y(0) + \frac{S_{i,1}}{2} = y(0) + \frac{\Delta t \times f\big(0, y(0)\big)}{2} = \begin{bmatrix} 0 \\ 0.25 \end{bmatrix} + \frac{1}{2} \times \begin{bmatrix} 0.0025 \\ -0.00083 \end{bmatrix}$$

$$\tilde{y}_1(1) = \begin{bmatrix} 0.00125 \\ 0.24958 \end{bmatrix}$$

$$S_{i,2} = \Delta t \times f\big(t_o + \Delta t/2\,, \tilde{y}_1(1)\big) = \Delta t \times f\big(0 + 0.01/2\,, \tilde{y}_1(1)\big)$$

$$S_{i,2} = \Delta t \times f\big(0.005, \tilde{y}_1(1)\big)$$

$$f\big(0.005, \tilde{y}_1(1)\big) = \ A\tilde{y}_1(1) + B(0.005)$$

$$f\big(0.005, \tilde{y}_1(1)\big) = \begin{bmatrix} 0 & 1 \\ -\frac{2}{3} & -\frac{1}{3} \end{bmatrix} \tilde{y}_1(1) + + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$f\big(0.005, \tilde{y}_1(1)\big) = \begin{bmatrix} 0 & 1 \\ -\frac{2}{3} & -\frac{1}{3} \end{bmatrix} \begin{bmatrix} 0.00125 \\ 0.24958 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$f\big(0.005, \tilde{y}_1(1)\big) = \begin{bmatrix} 0.24958 \\ -0.084 \end{bmatrix}$$

$$S_{i,2} = \Delta t \times f\big(0.005, \tilde{y}_1(1)\big) = \begin{bmatrix} 0.0024958 \\ -0.00084 \end{bmatrix}$$

$$\tilde{y}_2(1) = y(0) + \frac{S_{i,2}}{2} = y(0) + \frac{\Delta t \times f\big(0.005, \tilde{y}_1(1)\big)}{2} = \begin{bmatrix} 0 \\ 0.25 \end{bmatrix} + \frac{1}{2} \times \begin{bmatrix} 0.0024958 \\ -0.00084 \end{bmatrix}$$

$$\tilde{y}_2(1) = \begin{bmatrix} 0.001248 \\ 0.24958 \end{bmatrix}$$

$$S_{i,3} = \Delta t \times f\big(t_o + \Delta t/2\,, \tilde{y}_2(1)\big) = \Delta t \times f\big(0 + 0.01/2\,, \tilde{y}_2(1)\big)$$

$$S_{i,3} = \Delta t \times f\big(0.005, \tilde{y}_2(1)\big)$$

$$f\big(0.005, \tilde{y}_2(1)\big) = A\tilde{y}_2(1) + B(0.005)$$

$$f\big(0.005, \tilde{y}_1(1)\big) = \begin{bmatrix} 0 & 1 \\ -\frac{2}{3} & -\frac{1}{3} \end{bmatrix} \tilde{y}_2(1) + + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$f\big(0.005, \tilde{y}_1(1)\big) = \begin{bmatrix} 0 & 1 \\ -\frac{2}{3} & -\frac{1}{3} \end{bmatrix} \begin{bmatrix} 0.001248 \\ 0.24958 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$f\big(0.005, \tilde{y}_1(1)\big) = \begin{bmatrix} 0.24958 \\ -0.084 \end{bmatrix}$$

$$S_{i,3} = \Delta t \times f\big(0.005, \tilde{y}_1(1)\big) = \begin{bmatrix} 0.0024958 \\ -0.00084 \end{bmatrix}$$

$$\tilde{y}_3(1) = y(0) + S_3 = y(0) + \Delta t \times f\big(0.005, \tilde{y}_1(1)\big) = \begin{bmatrix} 0 \\ 0.25 \end{bmatrix} + \begin{bmatrix} 0.0024958 \\ -0.00084 \end{bmatrix}$$

$$\tilde{y}_3(1) = \begin{bmatrix} 0.0025958 \\ 0.24916 \end{bmatrix}$$

$$S_{i,4} = \Delta t \times f\big(t_o + \Delta t, \tilde{y}_3(1)\big) = \Delta t \times f\big(0 + 0.01, \tilde{y}_3(1)\big)$$

$$S_{i,4} = \Delta t \times f\big(0.01, \tilde{y}_3(1)\big)$$

$$f\big(0.01, \tilde{y}_3(1)\big) = A\tilde{y}_3(1) + B(0.01)$$

$$f\left(0.01, \tilde{y}_3(1)\right) = \begin{bmatrix} 0 & 1 \\ -\frac{2}{3} & -\frac{1}{3} \end{bmatrix} \tilde{y}_3(1) + + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$f\left(0.01, \tilde{y}_3(1)\right) = \begin{bmatrix} 0 & 1 \\ -\frac{2}{3} & -\frac{1}{3} \end{bmatrix} \begin{bmatrix} 0.0025958 \\ 0.24916 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$f\left(0.01, \tilde{y}_3(1)\right) = \begin{bmatrix} 0.24916 \\ -0.0847 \end{bmatrix}$$

$$S_{i,4} = \Delta t \times f\left(0.05, \tilde{y}_3(1)\right) = \begin{bmatrix} 0.0024916 \\ -0.000847 \end{bmatrix}$$

$$Q_1 = \frac{S_{i,1} + 2S_{i,2} + 2S_{i,3} + S_{i,4}}{6}$$

$$Q_1 = \frac{\begin{bmatrix} 0.0025 \\ -0.00083 \end{bmatrix} + 2\times \begin{bmatrix} 0.0024958 \\ -0.00084 \end{bmatrix} + 2\times \begin{bmatrix} 0.0024958 \\ -0.00084 \end{bmatrix} + \begin{bmatrix} 0.0024916 \\ -0.000847 \end{bmatrix}}{6}$$

$$Q_1 = \begin{bmatrix} 0.0024958 \\ -0.00084 \end{bmatrix}$$

$$y(i+1) = y(i) + Q_i$$

$$y(1) = y(0) + Q_1$$

$$y(1) = \begin{bmatrix} 0 \\ 0.25 \end{bmatrix} + \begin{bmatrix} 0.0024958 \\ -0.00084 \end{bmatrix}$$

$$y(1) = \begin{bmatrix} 0.0024958 \\ 0.24916 \end{bmatrix}$$

The Matlab Code for solving Second order, vibration system ODE of Example 3.1:

Matlab Program 1.13

```
%Matlab Code for Single Degree of freedom vibratory System
by Using RK4
%Example 3_1
function Ex3_1
clear all; clf; clc;
global m k c
global dt t t_rk

%Inputs
m = 3; % mass (Kg)
```

```
k = 2; % Stiffness (N/mm)
c = 1; % damping coefficient (N.s/mm)
%initial conditions
xo=0; % initial displacement (mm)
vo=0.25; % initial velocity (mm/s)
% time
tf = 20; %(sec)
dt = 0.01; %(sec)
xstate = [xo; vo]; %(mm; mm/sec)
for i=1:tf/dt
t = dt*i;
xstate = RK(xstate);
%results
z(i) = xstate(1); % Displacement(mm)
q(i) = xstate(2); % Velocity(mm/sec)
end
% Calculating Exact Solution
C= c^2-4*m*k;
% Overdamped System C^2>4mk
if C>0
p1= (-c+sqrt(c^2-4*m*k))/(2*m);
p2= (-c-sqrt(c^2-4*m*k))/(2*m);
A1=(xo*p2-vo)/(p2-p1);
A2=(vo-xo*p1)/(p2-p1);
for i=1:tf/dt+1
t1(i)=(i-1)*dt;
x_exact(i)=A1*exp(p1*t1(i))+A2*exp(p2*t1(i));
end
end
% Critically damped System C^2=4mk
if C==0
p1_1= (-c/(2*m));
c1=xo;
c2=vo-xo*p1_1;
for i=1:tf/dt1
t1(i)=(i-1)*dt;
x_exact(i)=(c1+c2*t1(i))*exp(p1_1*t1(i));
end
end
% Undrdamped System C^2<4mk
if C<0
alpha= (-c)/(2*m);
beta= (sqrt(4*m*k-c^2))/(2*m);
X=sqrt(xo^2+((vo-alpha*xo)/beta)^2);
phi=atand((beta*xo)/(vo-alpha*xo));
for i=1:tf/dt+1
t1(i)=(i-1)*dt;
x_exact(i)=X*exp(alpha*t1(i))*sin(beta*t1(i)+phi*pi/180);
```

```
end
end
%plotting
plot(0:dt:tf,[xo,z],'linewidth', 2);
hold on
plot(t1,x_exact,'r:','linewidth', 2);
xlabel('Time (sec)','FontSize',12); ylabel('Displacement
(mm/sec)','FontSize',12);
title(' Transient Response','FontSize',15)
legend('Numerical', 'Exact','location', 'best')
grid on
saveas(gcf, 'Example 3_1.tiff');
figure(2)
plot(0:dt:tf,[xo,z],'linewidth', 2);
hold on
plot(0:dt:tf,[vo,q],'r:','linewidth', 2); xlabel('Time
(sec)','FontSize',12);
title(' Transient Response','FontSize',15)
legend('Displacement', 'Velocity','location', 'best')
grid on
saveas(gcf, 'Example 3_1_velocity.tiff');
function xstate= RK(xstate)
global dt t t_rk
t_rk=(t-dt);
S1 = dt * x_dot( xstate       );
t_rk=(t-dt)+dt/2;
S2 = dt * x_dot( xstate + S1/2);
t_rk=(t-dt)+dt/2;
S3 = dt * x_dot( xstate + S2/2);
t_rk=(t-dt)+dt;
S4 = dt * x_dot( xstate + S3  );
Q=1/6 *(S1 + 2*S2 + 2*S3 + S4);
xstate = xstate + Q;
function xdot=x_dot(X)
global m k c
% forcing function
F_t= force;
A=[0, 1; -k/m, -c/m];
b=[0; F_t/m];
xdot = A*X+b;
function F_of_t =force
global t_rk
F_of_t=0;
% F_of_t=240000*t_rk;
```

The time response is plotted in Fig. 1.13.

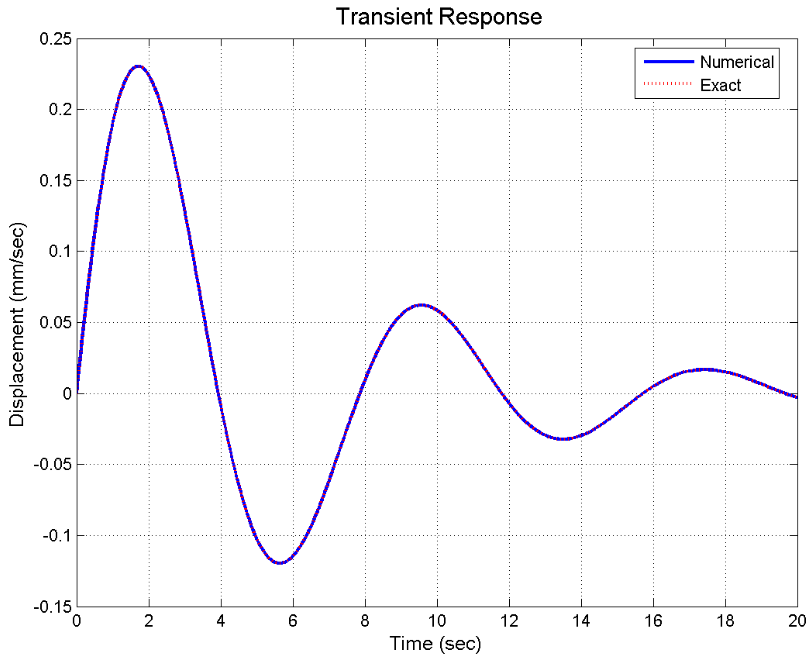

Fig. 1.13. Time response of example 3.1.

It was shown in example 1.5 that the exact solution of this differential equation is

$$x(t) = 0.31e^{-\frac{t}{6}}\sin(0.79t)$$

9. Enhancements to Runge-Kutta integration method

Runge-Kutta method can be enhanced by treating the time step Δt as a variable, Δt_i. At each time t_i, the value of Δt_i is adjusted based on how rapidly the solution of $x(t)$is changing. If the solution is not changing very rapidly, a large value of Δt_i is allowed without increasing the formula error. On the other hand, if $x(t)$is changing rapidly, a small Δt_i must be chosen to keep the formula error small. Such step sizes can be chosen automatically as part of the computer code for implementing the numerical solution.
Matlab has two different Runge-Kutta based simulations: `ode23` and `ode45`. These are automatic step-size integration methods (i.e., Δt is chosen automatically).

Example 3.1 can be solved by using `ode45` function in matlab as follows

Matlab Program 1.14

Create and save the following code as `sdof.m`

```
% function Ex_3_1_ODE45
function xdot=sdof(t,x)
```

```
m = 3; % mass (Kg)
k = 2; % Stiffness (N/mm)
c = 1; % damping coefficient (N.s/mm)
xdot=zeros(2,1);
xdot(1)=x(2);
xdot(2)=(-k/m)*x(1)+(-c/m)*x(2);
```

In the same directory as sdof.m, create a new m-file as follows:

```
% Program to solve 1 DOF vibration system by using RK-4
clear all; clf; clc;
to=0;
tf=20;
xo=[0 0.25];
[t,x]=ode45('sdof', [to tf], xo);
plot(t,x(:,1),'linewidth', 2); xlabel('Time
(sec)','FontSize',12);
hold on
plot(t,x(:,2),'r:','linewidth', 2); xlabel('Time
(sec)','FontSize',12);
title(' Transient Response','FontSize',15)
legend('Displacement', 'Velocity','location', 'best')
grid on
saveas(gcf, 'Example 3_1_ODE45.tiff');
```

The time response by using ODE45 function is shown in Fig. 1.14.

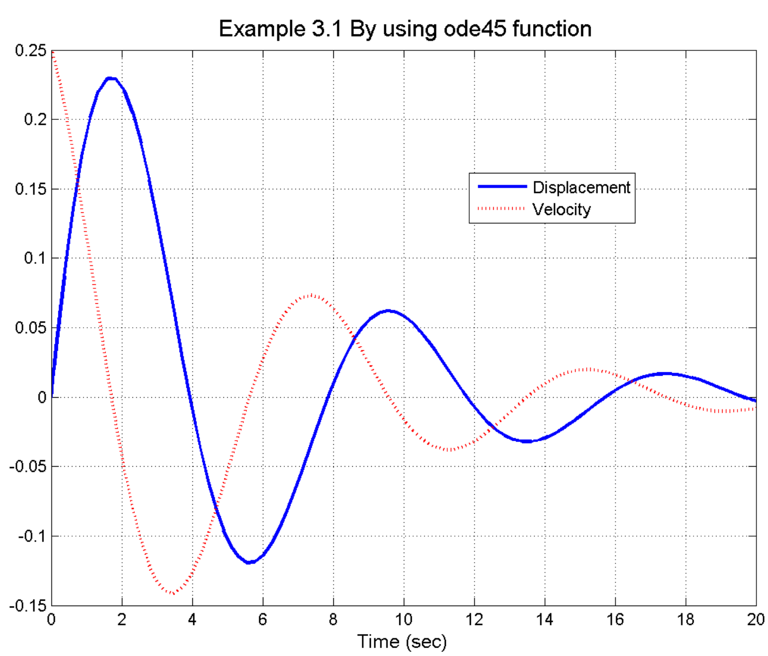

Fig. 1.14. Time response of example 3.1 by using ode45 Matlab function.