ADVANCED PROGRAMMING IN SCILAB

Chapterwise Solutions

CHAPTER 1

1) It will give an error of ‘Inconsistent multiplication’.

2) The correct answers are,

a) 4. 10. 18.
b) 32.
c) 4. 5. 6.
8. 10. 12.
12. 15. 18.

3) The command will print the elements of all the rows and columns.

```
1.      2.      3.
4.      5.      6.
```

4) B = 8.84 9.32 3.61

5) B = 8. 9. 3.

6) The result of the operations is tabulated in *Table 1.6*.

Table 1.6*: Solution for *Exercise 6

$\mathbf{A} = \begin{bmatrix} \mathbf{1} & \mathbf{2} & \mathbf{3} \\ \mathbf{4} & \mathbf{5} & \mathbf{6} \\ \mathbf{7} & \mathbf{8} & \mathbf{9} \end{bmatrix}$	
***SciLab* Command**	**Output**

`A(1,:) = 2*A(1,:)`	`A =`
	`2.  4.  6.`
	`4.  5.  6.`
	`7.  8.  9.`

`A(1,:) = A(1,:) + A(2,:)`	`A =`
	`5.  7.  9.`
	`4.  5.  6.`
	`7.  8.  9.`

`A(:,2) = 3*A(:,2)`	`A =`
	`1.   6.   3.`
	`4.  15.   6.`
	`7.  24.   9.`

`A(:,1) = A(:,1) - 0.5*A(:,2)`	`A =`
	`0.    2.   3.`
	`1.5   5.   6.`
	`3.    8.   9.`

7) `A(length(A))`

8) `A = - 1.    2.    3.`

9) The output is written below.

```
1.    2.
3.    4.
5.    6.
```

10) The *SciLab* command will be,

```
A(1:2,2:4) = 2
```

11) The *SciLab* command will be,

```
A = [ones(1, 3) ; zeros(1,3); ones(1, 3)];
```

12) Consider two points having the following position vectors.

$$\vec{A} = a_1\hat{\imath} + a_2\hat{\jmath} + a_3\hat{k}$$
$$\vec{B} = b_1\hat{\imath} + b_2\hat{\jmath} + b_3\hat{k}$$

The distance between these two points is given by,

$$\sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + (a_3 - b_3)^2}$$

The *SciLab* code for determining this distance between the given points is given below.

```
function d = distance(a,b)
d = sqrt(sum((a-b).^2))
endfunction
A = [1 2 3];
B = [4 5 6];
distance(a,b)
```

13) The *SciLab* program is written below.

```
function SC = sum_cube(a)
SC = sum(a.^3)
endfunction
A = [1 2 3];
sum_cube(A);
```

14) If `[a b c]` and `[d e f]` are two vectors, then their scalar or dot product is defined by,

```
[a b c].[d e f]= a*d + b*e + c*f
```

Therefore, for two vectors $\vec{A}$ and $\vec{B}$, the function for determining the scalar product can be written in the following manner.

```
function SP = scalar_product(A,B);
SP = sum(A.*B);
endfunction
```

15) Suppose, $\vec{c} = \vec{a} \times \vec{b}$
The function and the *SciLab* program for determining the cross product of two vectors is given by,

```
function [c] = cross_product(a,b)
c = [a(2)*b(3)-a(3).*b(2) a(3)*b(1)-a(1)*b(3) a(1)*b(2)-
a(2)*b(1)];
endfunction
a = [1 2 3];
b = [4 5 6];
c = cross_product(a,b)
```

16) The angle between two vectors $\vec{A}$ and $\vec{B}$ is θ such that,

$$\cos\theta = \frac{\vec{A}\cdot\vec{B}}{|\vec{A}||\vec{B}|}$$

The *SciLab* program for determining this angle is written below.

```
function angle = angle_between_vectors(A,B)
length_A = (A(1)^2 + A(2)^2 + A(3)^2)^0.5;
length_B = (B(1)^2 + B(2)^2 + B(3)^2)^0.5;
angle = acosd((sum(A.*B))/(length_A*length_B)) ;
endfunction
A = [3 4 2];
B = [2 2 -3];
angle_between_vectors(A,B)
```

17) Consider a parallelepiped whose adjacent sides are the vectors $\vec{A}$, B and $\vec{C}$ (see *Figure 1.10*). The volume of the parallelepiped is given by the scalar triple product of these three vectors, i.e.

$$V = \left|\left(\vec{A}\times\vec{B}\right)\cdot\vec{C}\right|$$

The *SciLab* program written below calculates the volume of a parallelepiped where the three vectors are equal to,

$$\vec{A} = 2\hat{\imath} + 2\hat{\jmath} + 2\hat{k}$$
$$\vec{B} = 5\hat{\imath} + 3\hat{\jmath} - \hat{k}$$
$$\vec{C} = 2\hat{\imath} + 4\hat{\jmath} + 6\hat{k}$$

```
A = [2, 2, 2];
B = [5, 3, -1];
C = [2, 4, 6];
abs(A* cross(B,C)') = abs(B* cross(C,A)') = abs(C*
cross(B,A)')
```

The answer will be equal to 8.

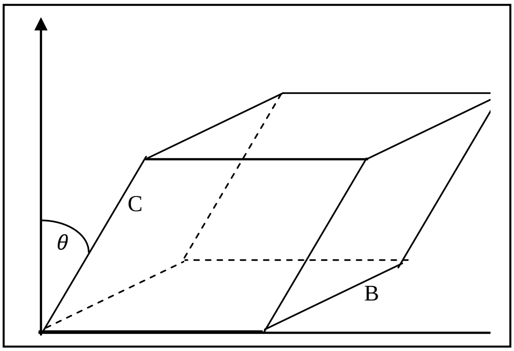

***Figure 1.10*:** Diagram for *Exercise 17*

18) The *SciLab* program is written below.

```
A = [1 2 3]
P = poly(0,'a')
B = p*A
```

The answer will be equal to, `B = a         2a        3a`

19) The *SciLab* program is written below.

```
A = [1 2 3]
P = poly(0,'a')
u = [1+p, (1+p)^2, (1+p)^3 ]
B = A.*u
```

The answer will be equal to,

```
B =
                    2                2    3
 1 + a     2 + 4a + 2a     3 + 9a + 9a + 3a
```

20) The *SciLab* program is written below.

```
function [integral] = indef_integral_poly(p)
coefficient = coeff(p);
a = 1;
for i = 1:length(coefficient);
      a = [a i];
end
new_coeff = [0 coefficient];
new_coeff = new_coeff./a;
integral = poly(new_coeff,varn(p),'coeff');
endfunction

//Define the polynomial
```

```
polynomial = poly([1],'x','coeff')

//Call the function
indef_integral_poly(polynomial)

//Determine the value of the integral at x = 2.
horner(indef_integral_poly(polynomial),2)
```

The integral will be equal to ‘x’. Its value at $x = 2$ will be equal to 2.

21) The function to evaluate the indefinite integral remains same as in the previous question. The *SciLab* program is written below.

```
polynomial = poly([0 1],'x','coeff')
Answer_1 = indef_integral_poly(polynomial)
Answer_2 = horner(indef_integral_poly(polynomial),2)
```

The answer will be equal to,
Answer_1 = $0.5x^2$
Answer_2 = 2

22) The *SciLab* program is written below.

```
q = poly(3,'x','r')
```

The answer will be equal to,

```
q = - 3 + x
```

23) The *SciLab* program is written below.

```
m = input("Enter the number of rows in the matrix : ");
n = input("Enter the number of columns in the matrix :
");
disp("Enter the elements of the matrix row wise");
for i = 1:1:m;
      for j = 1:1:n;
            A(i,j) = input(" ");
      end
end
disp(A,"Matrix A : ");
det(A)
det(A’)
```

The output of this program is as follows.

```
A =
0.    7.    2.
5.    1.    2.
6.    5.    2.
det(A) = 52
det(A’) = 52
```

24) The *SciLab* program is written below.

```
A = [1 2 + %i 3 ; 2 - %i 4 5*%i ; 3 -5*%i 6]
if A' == A then
      disp("Matrix is Hermitian");
else
      disp("Matrix is not Hermitian");
end
```

The output of this program is as follows.

```
A =
1.          2. + i      3.
2. - i      4.          5.i
3.        - 5.i         6.

A' =
1.          2. + i      3.
2. - i      4.          5.i
3.        - 5.i         6.
Matrix is Hermitian
```

25) For an orthogonal matrix (A)

$$AA^T = A^T A = I$$

The *SciLab* program is written below.

```
A = [sind(30) cosd(30) ; -cosd(30) sind(30)]
A.'*A
A*A.'
```

The output of this program is as follows.

```
A =
0.5            0.8660254
- 0.8660254      0.5

A.’*A =
1.  0.
0.  1.

A*A.’ =
1.  0.
```

```
0.   1.
```

26) For a unitary matrix (A)

$$AA^{C^T} = A^{C^T}A = AA^{+} = A^{+}A = I$$

The *SciLab* program is written below.

```
A = [%i 0 ; 0 %i]
A*A'
A'*A
```

The output of this program is written below.

```
    A =
    i      0
0      i
    A*A' =
        1.     0
0     1.
A'*A =
1.     0
0     1.
```

27) The *SciLab* program is written below.

```
A = [1 2+%i 3 ; 2-%i 4 5*%i ; 3 -5*%i 6]
format(5)
[eigen_vector,eigen_value] = spec(A)
clean(eigen_vector(:,1)'*eigen_vector(:,3))
clean(eigen_vector(:,1)'*eigen_vector(:,2))
clean(eigen_vector(:,1)'*eigen_vector(:,1))
```

The output of this program is written below.

```
eigen_value  =
    - 2.83    0.      0.
    0.      3.23    0.
    0.      0.      10.6
eigen_vector  =
    0.61 - 0.24i   - 0.42 + 0.58i    0.19 + 0.13i
     - 0.14 + 0.53i    0.35 + 0.47i    0.08 + 0.59i
     - 0.51          - 0.39            0.77
clean(eigen_vector(:,1)'*eigen_vector(:,3)) = 0
clean(eigen_vector(:,1)'*eigen_vector(:,2)) = 0
clean(eigen_vector(:,1)'*eigen_vector(:,1)) = 1
```

28) The conversion of components of a vector $(\vec{A})$ from Cartesian system (A_x, A_y, A_z) to cylindrical system (A_r, A_θ, A_z) is given by,

$$\begin{pmatrix} A_r \\ A_\theta \\ A_z \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} A_x \\ A_y \\ A_z \end{pmatrix}$$

The *SciLab* program written below determines the components of the given vector in cylindrical coordinate system.

```
exec('vectors.sci',-1);
A = [2 -3 4];
              //Coordinates of the point
B = cartesian_to_cylindrical(A);
x = B(1).*cosd(B(2));        //Define x in terms of (r,θ)
y = B(1).*sind(B(2));        //Define y in terms of (r,θ)
z = B(3)
A = [x.*y z y];
//Components of the vector
Answer   =   [cosd(B(2))   sind(B(2))   0   ;   -sind(B(2))
cosd(B(2)) 0 ; 0 0 1]*[A(1) ; A(2) ; A(3)]
```

The answer will be equal to,
$A_r = 6.6564024$
$A_\theta = 2.773501$
$A_z = 3$

31) The *SciLab* program is written below.

```
A = [90 -40 50 ; -40 80 -30 ; -50 -30 100];
C = [5 ; 0 ; 0];
B = A\C
```

The answer will be equal to,
$I_1 = 0.0522828$
$I_2 = 0.0405007$
$I_3 = 0.0382916$

32) Let mass of the cube is 'M'. It is distributed uniformly. Therefore mass per unit volume will be equal to $\frac{M}{a^3}$. The elements of the moment of inertia tensor can be obtained in the following manner.

$$I_{xx} = \frac{M}{a^3} \int_{x=0}^{a} \int_{y=0}^{a} \int_{z=0}^{a} (y^2 + z^2)\, dz\, dy\, dx = \frac{2}{3} M a^2$$

$$I_{yy} = \frac{M}{a^3} \int_{y=0}^{a} \int_{x=0}^{a} \int_{z=0}^{a} (x^2 + z^2)\, dz\, dx\, dy = \frac{2}{3} M a^2$$

$$I_{zz} = \frac{M}{a^3}\int_{z=0}^{a}\int_{y=0}^{a}\int_{x=0}^{a}(x^2+y^2)\,dx\,dy\,dz = \frac{2}{3}Ma^2$$

$$I_{xy} = I_{yx} = -\frac{M}{a^3}\int_{z=0}^{a}\int_{y=0}^{a}\int_{x=0}^{a} xy\,dx\,dy\,dz = -\frac{1}{4}Ma^2$$

$$I_{xz} = I_{zx} = -\frac{M}{a^3}\int_{y=0}^{a}\int_{z=0}^{a}\int_{x=0}^{a} xz\,dx\,dz\,dy = -\frac{1}{4}Ma^2$$

$$I_{yz} = I_{zy} = -\frac{M}{a^3}\int_{x=0}^{a}\int_{y=0}^{a}\int_{z=0}^{a} yz\,dz\,dy\,dx = -\frac{1}{4}Ma^2$$

This gives,

$$I = Ma^2\begin{pmatrix} 2/3 & -1/4 & -1/4 \\ -1/4 & 2/3 & -1/4 \\ -1/4 & -1/4 & 2/3 \end{pmatrix}$$

The *SciLab* program to determine the principal axes is written below.

```
A = [2/3 -1/4 -1/4 ; -1/4 2/3 -1/4 ; -1/4 -1/4 2/3];
[eigen_vector,eigen_value] = spec(A);
```

The result of this short *SciLab* code can be interpreted in the following manner.

- The eigen value matrix will be,

$$\begin{pmatrix} 0.166667 & 0 & 0 \\ 0 & 0.916667 & 0 \\ 0 & 0 & 0.916667 \end{pmatrix}$$

- This implies that the eigen values are,

$$(0.166667Ma^2, 0.916667Ma^2, 0.916667Ma^2)$$

- The eigen vector matrix will be,

$$\begin{pmatrix} -0.5773503 & 0.1243009 & 0.8069795 \\ -0.5773503 & -0.7610152 & -0.2958421 \\ -0.5773503 & 0.6367143 & -0.5111375 \end{pmatrix}$$

- The eigen vector corresponding to the first eigen value can be written as,

$$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

- This implies that the direction of principal axis is along $\widehat{e_x} + \widehat{e_y} + \widehat{e_z}$. This represents the diagonal of the cube and is the symmetry (principal) axis of the rotating cube. Similarly the other principal axes can be found from the other two eigen vectors.

33) If the cube is rotating about its center, then the moment of inertia tensor will be a diagonal matrix and off diagonal elements will be zero. The direction of total angular momentum will always be parallel to the angular velocity. The elements of the moment of inertia tensor can be obtained in the following manner.

$$I_{xx} = \frac{M}{a^3} \int_{x=0}^{a} \int_{y=0}^{a} \int_{z=0}^{a} (y^2 + z^2)\, dz\, dy\, dx = \frac{1}{6} M a^2$$

$$I_{yy} = \frac{M}{a^3} \int_{y=0}^{a} \int_{x=0}^{a} \int_{z=0}^{a} (x^2 + z^2)\, dz\, dx\, dy = \frac{1}{6} M a^2$$

$$I_{zz} = \frac{M}{a^3} \int_{z=0}^{a} \int_{y=0}^{a} \int_{x=0}^{a} (x^2 + y^2)\, dx\, dy\, dz = \frac{1}{6} M a^2$$

$$I_{xy} = I_{yx} = -\frac{M}{a^3} \int_{z=0}^{a} \int_{y=0}^{a} \int_{x=0}^{a} xy\, dx\, dy\, dz = 0$$

$$I_{xz} = I_{zx} = -\frac{M}{a^3} \int_{y=0}^{a} \int_{z=0}^{a} \int_{x=0}^{a} xz\, dx\, dz\, dy = 0$$

$$I_{yz} = I_{zy} = -\frac{M}{a^3} \int_{x=0}^{a} \int_{y=0}^{a} \int_{z=0}^{a} yz\, dz\, dy\, dx = 0$$

This gives,

$$I = \frac{Ma^2}{6} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

34) The graph shown in *Figure 1.11* is expected for this exercise.

35) The graph shown in *Figure 1.12* is expected for this exercise.

36) The graph shown in *Figure 1.13* is expected for this exercise.

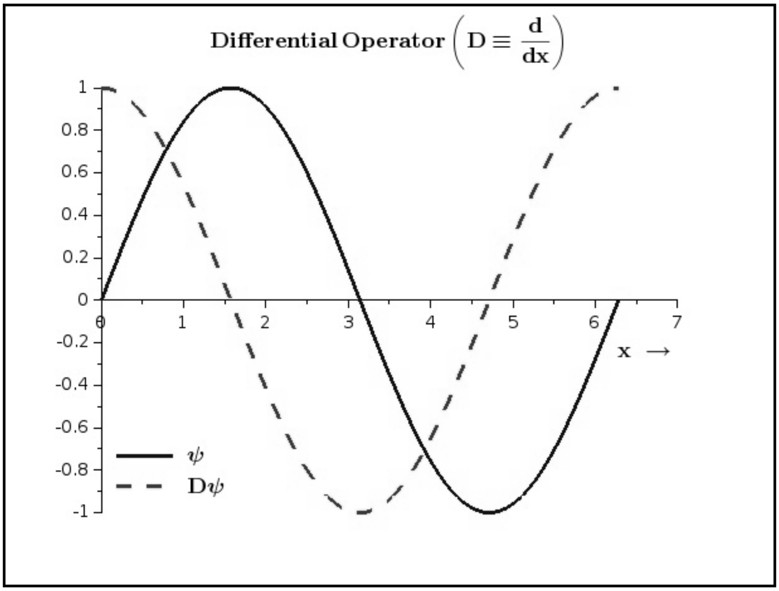

***Figure 1.11*:** Solution for *Exercise 34*

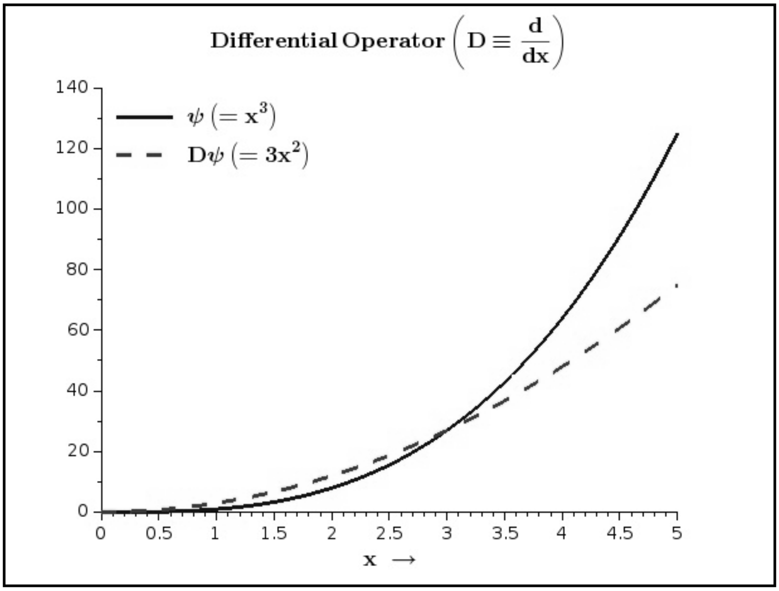

***Figure 1.12*:** Solution for *Exercise 35*

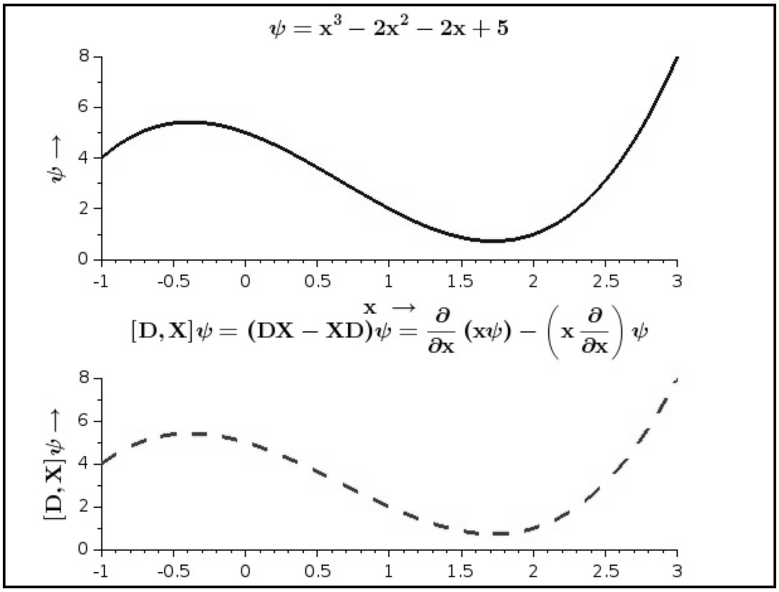

***Figure 1.13*:** Solution for *Exercise 36*

37) The *SciLab* program is written below.

```
minimum_x = 0;
maximum_x = 1;
N = 500;
x = linspace(minimum_x,maximum_x,N)';
h = x(2) - x(1);

function y = func(x)
y = x;
endfunction

D = ( diag(ones((N-1),1),1) - diag(ones((N-1),1),-1)
)/(2*h);
D(1,1) = -1/h;
D(1,2) = 1/h;
D(2,1) = -1/(2*h);
D(N,N-1) = -1/h;
D(N-1,N) = 1/(2*h);
D(N,N) = 1/h;
D*(diag(1-x.^2)*D*func(x))
```

38) The graph shown in *Figure 1.14* is expected for this exercise.

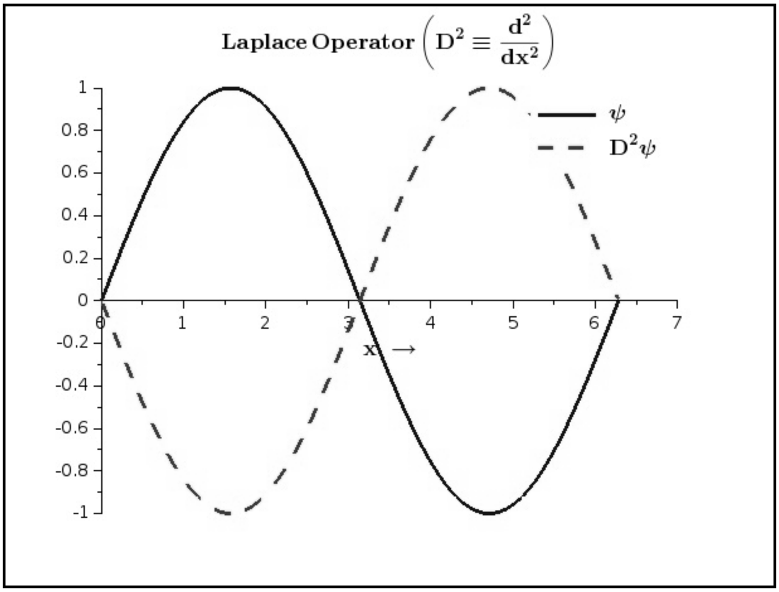

Figure 1.14: Solution for *Exercise 38*

39) The graph shown in *Figure 1.15* is expected for this exercise.

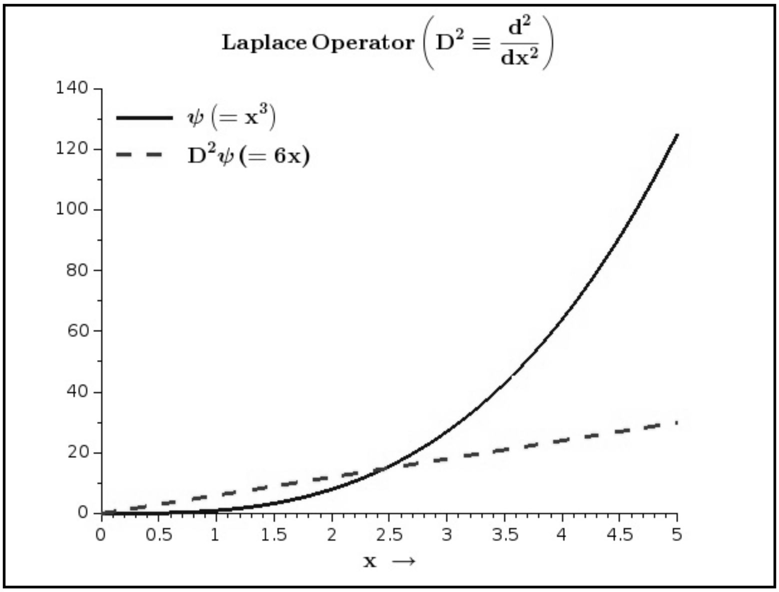

Figure 1.15: Solution for *Exercise 39*

40) The first three wave functions are shown in *Figure 1.16*. The energy eigen value for these wave functions will be equal to,

$$E_0 = 1.3810434, E_1 = 4.1560752, E_2 = 7.0092436$$

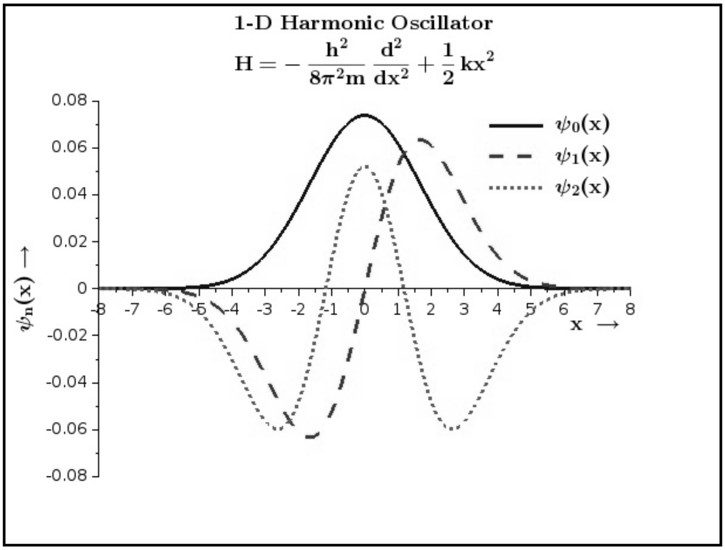

Figure 1.16: Solution for *Exercise 40*

41) The first three wave functions are shown in *Figure 1.17*. The energy eigen value for these wave functions will be equal to,

$$E_1 = -13.594455, \; E_2 = -3.4022647, \; E_3 = -1.5123143$$

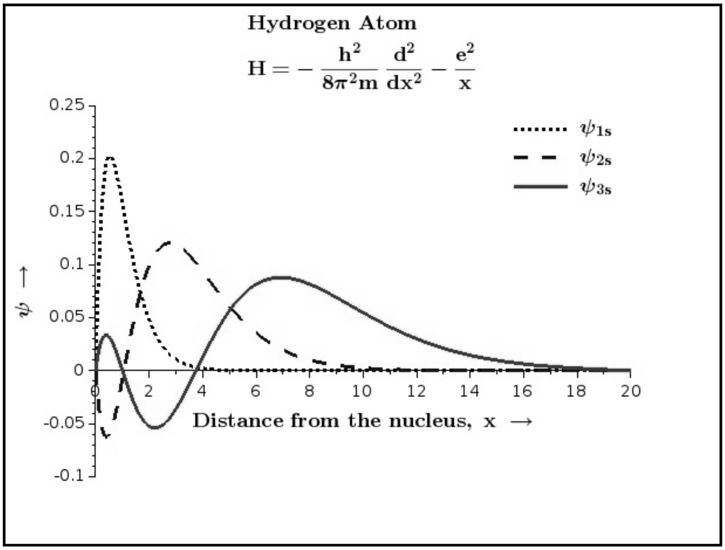

Figure 1.17: Solution for *Exercise 41*

CHAPTER 2

4) The *SciLab* program is written below. The histogram is shown in *Figure 2.44*.

```
data=rand(1,1000,'normal');     //Generate random number
x = -5:5;
y = histplot(x,data,style = 2)      //Generate histogram
histplot(x,data,style = 2);             //Plot histogram
plot2d(x(1:length(y))+0.5,y)        //Mark the frequency
```

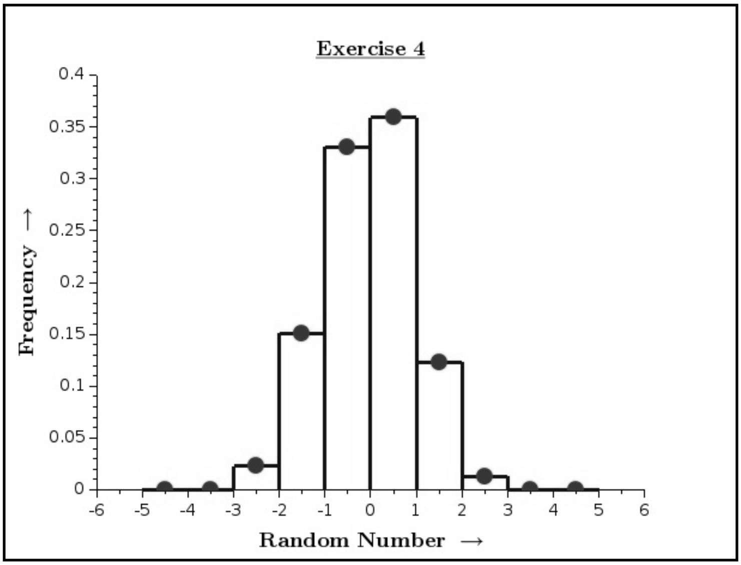

***Figure 2.44*:** Solution for *Exercise 4*

5) The *SciLab* program is written below.

```
x = 0:0.3:%pi;
y = sin(x);
em = 0.1*sqrt(y);
ep = 0.1*sqrt(y);
plot(x,y,'ro-','markersize',15)
a = gca();
a.box='off';
errbar(x, y, em, ep);
h=gce();
h.thickness = 2;
newcolour = 2;
h.segs_color = newcolour *ones(h.segs_color);
xgrid(3)
```

6) The *SciLab* program is written below. The solution is shown in *Figure 2.45*.

```
x = [0 0.5 1 1 2 3 3.5 4.51];
y = [0 0.51 0.51 0.2 -0.31 -0.25 0.2 0.25];

data = [x;y];                              //Give data set
x_new = [0.25 0.7 2.3 3.3 4];
y_new = interpln(data,x_new);        //Interpolated values
plot2d(data(1,:),data(2,:))           //Plot original data
plot2d(x_new,y_new)                //Plot interpolated data
```

The values of the dependent variable at intermediate points are,

```
y_new = 0.255 0.51 -0.292 0.02 0.2247
```

7) The *SciLab* program is written below. The solution is shown in *Figure 2.46*.

```
x = [0 1.2 2.1 2.9 3.5 4.9 6 7.1];
y = [0 0.51 0.51 0.93 0.31 0.15 0.4 0.25];
data = [x;y];                            //Given data set
data_smooth=smooth(data,0.1);  //Interpolate/smooth data
plot2d(data(1,:),data(2,:))       //Mark/plot data points
//Draw smooth curve
plot2d(data_smooth(1,:),data_smooth(2,:));
```

8) The *SciLab* program is written below. The solution is shown in *Figure 2.47*.

```
data = rand(1,1000,'normal');
x = -5:5;
//Generate histogram
y = histplot(x,data,style=2,normalization=%f)

//Plot histogram
histplot(x,data,style=2,normalization=%f)

//Original data
data1 = [x(1:length(y))+0.5;y];

//Smooth data
data1_smooth = smooth(data1,0.1);

//Plot smooth data
plot2d(data1_smooth(1,:),data1_smooth(2,:))

//Mark the frequency
plot2d(data1(1,:),data1(2,:),5)
```

9) The *SciLab* program is written below.

```
x = 0:0.01:2*%pi;
y = cos(x);
plot2d(x,y,2)
title('Cosine
Wave','fontsize',7,'color','black','fontname','times
italic','edgecolor','red','backgroundcolor','yellow' );
```

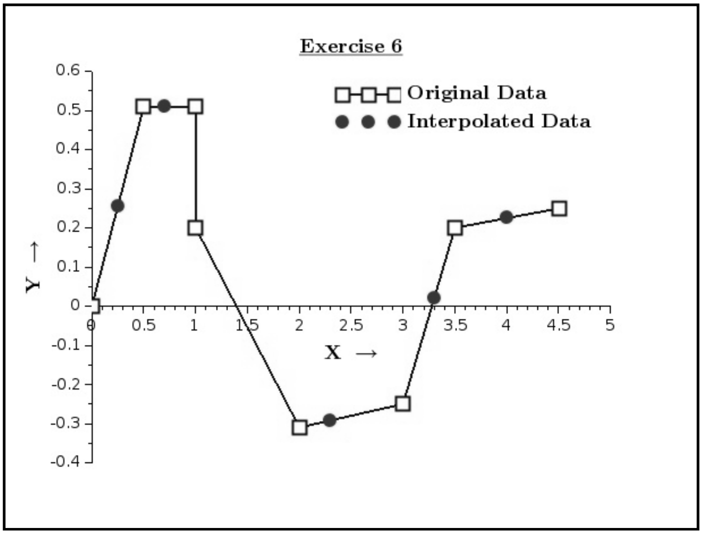

***Figure 2.45*:** Solution for *Exercise 6*

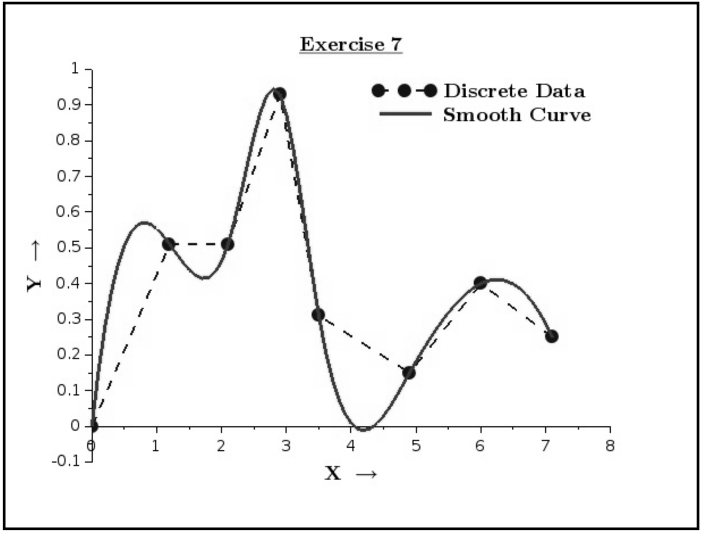

***Figure 2.46*:** Solution for *Exercise 7*

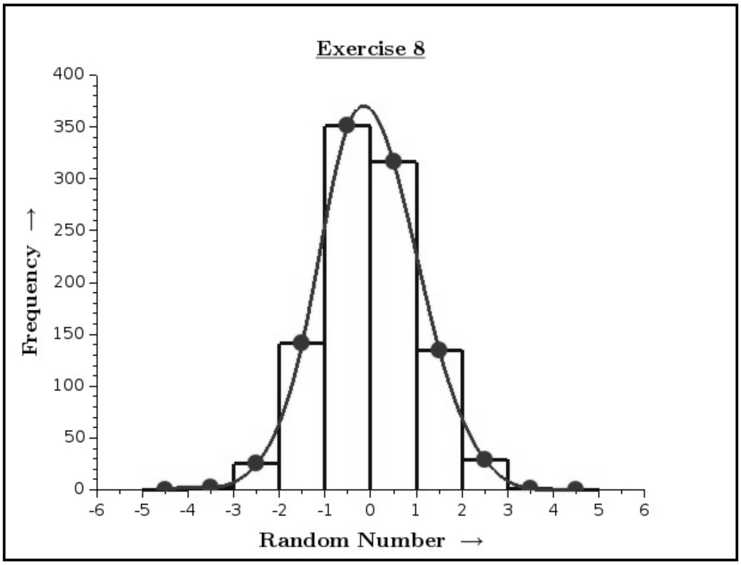

***Figure 2.47*:** Solution for *Exercise 8*

10) The graph shown in *Figure 2.48* is expected.

11) The resultant graph is given in *Figure 2.49.*

12) The *SciLab* program is written below. The graph is shown in *Figure 2.50.*

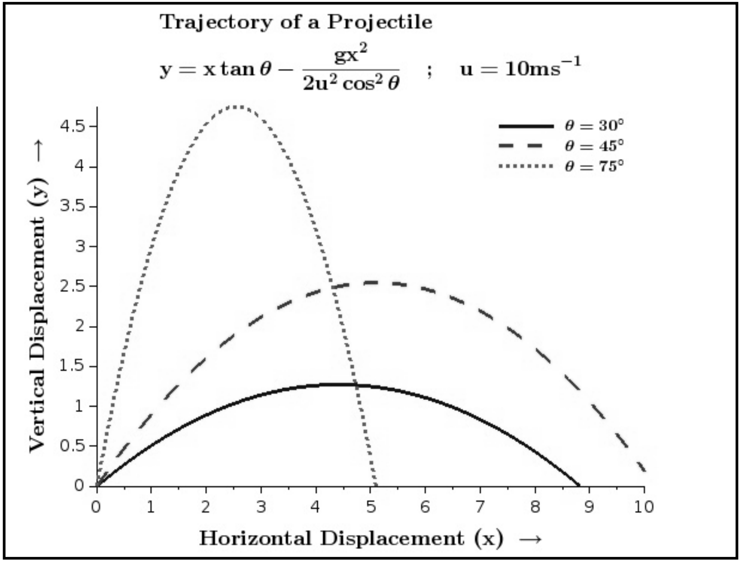

***Figure 2.48*:** Solution for *Exercise 10*

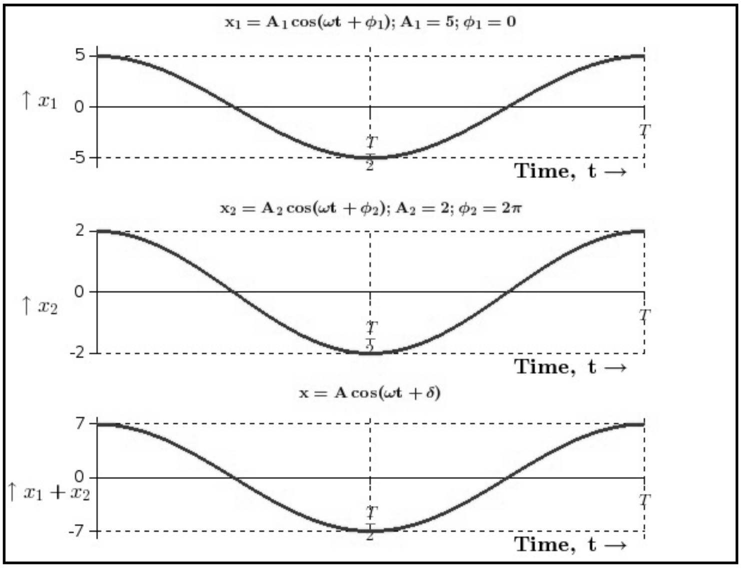

***Figure 2.49*:** Solution for *Exercise 11*

```
C = 100d-6;                  //Value of capacitor (in farad)
R = 100;                          //Value of resistor (in Ω)
tau = C*R;                                  //Time constant
Vs = 5;                                    //Source voltage
t = 0: 0.001: 7*tau;              //Time range for plotting
V = Vs * exp(-t/tau);            //Voltage across capacitor
i = (Vs-V)/R;                      //Current in the circuit
subplot(211)
plot2d(t/tau,V/Vs);

subplot(212)
plot2d(t/tau,i/max(i));
```

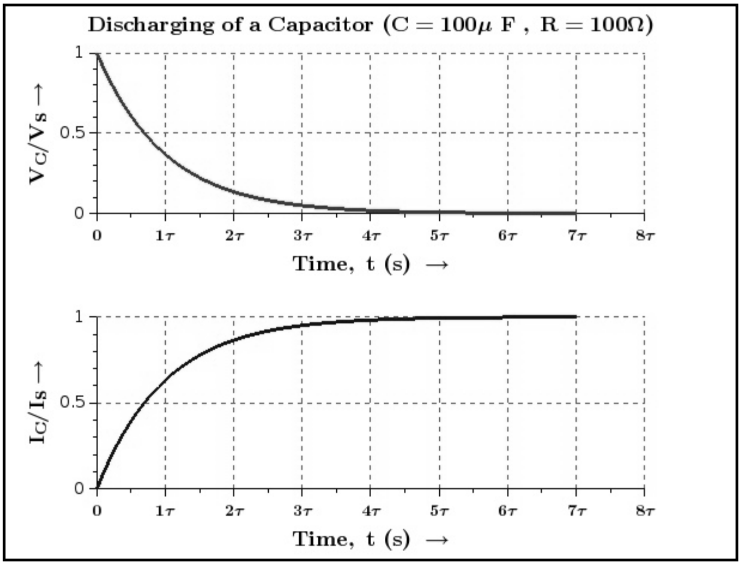

***Figure 2.50*:** Solution for *Exercise 12*

13) The *SciLab* program is written below. The graph is shown in *Figure 2.51*.

```
function y = dirac(x)
y = exp((-(x-
a).^(2))/(2.*sigma.*sigma))/sqrt(2*%pi.*sigma.*sigma)
endfunction

sigma = 0.2;                         //Standard Deviation
a = 2;                                     //Shift factor
x = a-2:0.01:a+2;                   //Range of x-variable
z = dirac(x);                     //Evaluate the function
plot2d(x,z)                           //Plot the function
sigma = 0.1;                         //Standard Deviation
z = dirac(x);                     //Evaluate the function
plot2d(x,z)                           //Plot the function
```

14) The graph shown in *Figure 2.52* is expected for this exercise.

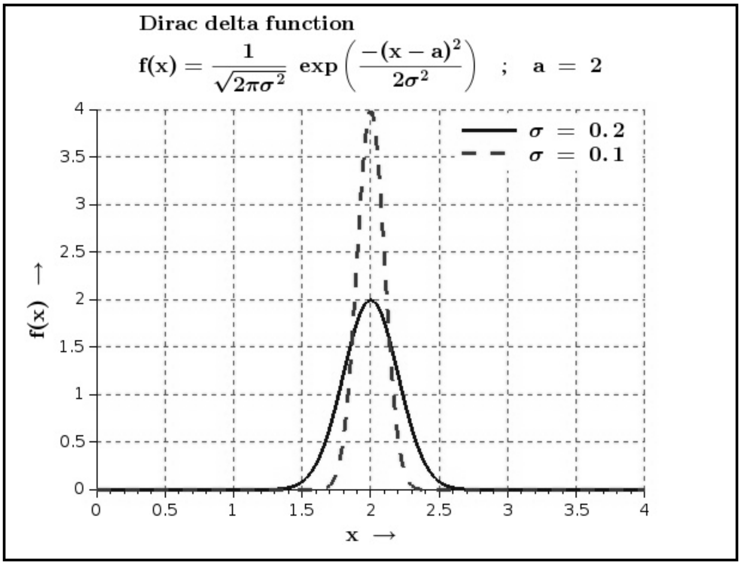

Figure 2.51**:** Solution for *Exercise 13*

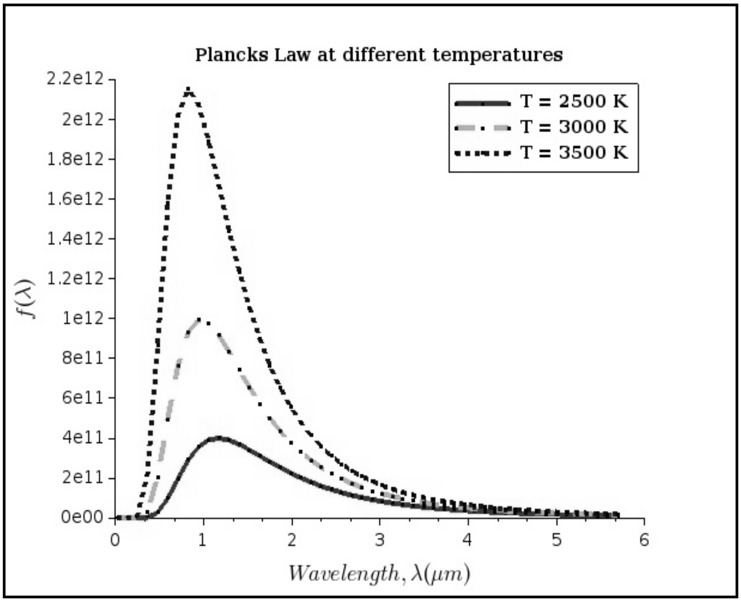

Figure 2.52**:** Solution for *Exercise 14*

15) The *SciLab* program is written below.

```
x = [1 0 0 1 1];
y = [1 1 0 0 1];
```

```
z = [0 0 0 0 0];
param3d(x,y,z,alpha=75);

x = [1 0 0 1 1];
y = [1 1 0 0 1];
z = [0.5 0.5 0.5 0.5 0.5];
param3d(x,y,z,alpha=75);

x = [1 0 0 1 1];
y = [1 1 0 0 1];
z = [1 1 1 1 1];
param3d(x,y,z,alpha=75);
```

16) The graph shown in *Figure 2.53* is expected for this exercise.

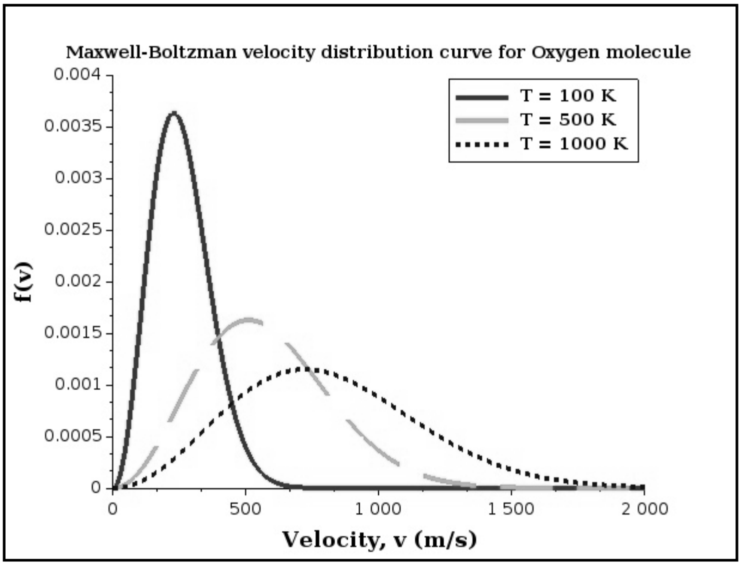

***Figure 2.53*:** Solution for *Exercise 16*

17) The graph shown in *Figure 2.54* is expected for this exercise.

18) The graph shown in *Figure 2.55* is expected for this exercise.

19) The graph shown in *Figure 2.56* is expected for this exercise.

20) The graph shown in *Figure 2.57* is expected for this exercise.

21) The graph shown in *Figure 2.58* is expected for this exercise.

22) The graph shown in *Figure 2.59* is expected for this exercise.

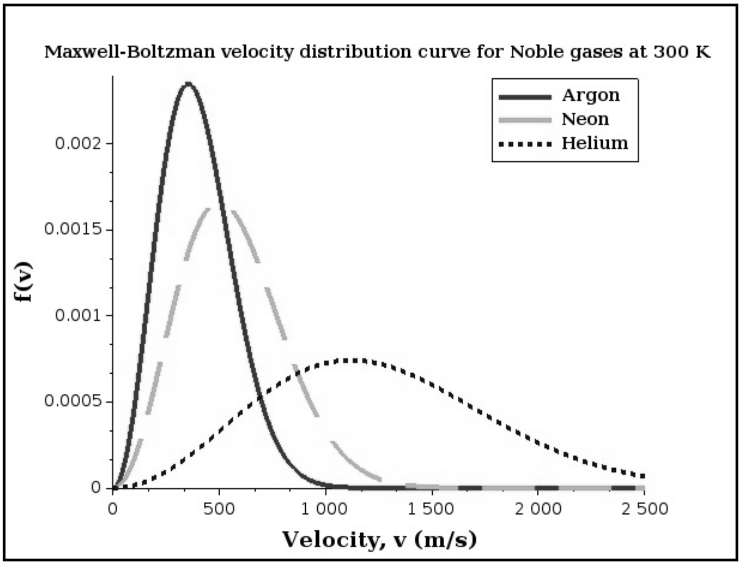

***Figure 2.54*:** Solution for *Exercise 17*

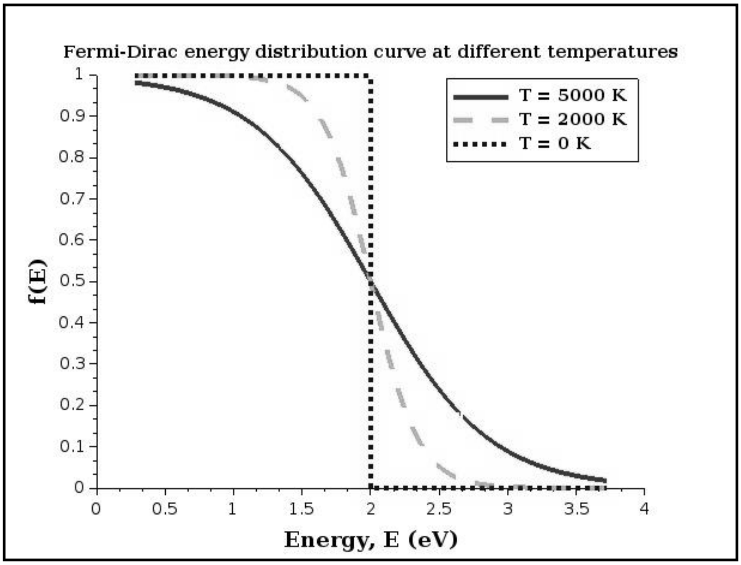

***Figure 2.55*:** Solution for *Exercise 18*

23) The graph shown in *Figure 2.60* is expected for this exercise.

24) The *SciLab* programs are written below.

Part (a)

```
function f = func(vector)
```

```
f(1) = vector(1) * vector(2);
f(2) = -vector(2).^3;
f(3) = (vector(1)-1) * vector(3);
endfunction

vector = [5 3 8];
sum(diag(numderivative(func,vector)))
```

The answer will be equal to, -20.

Part (b)

```
function f = func(vector)
f(1) = vector(1) * vector(1);
f(2) = vector(2) * vector(2);
f(3) = vector(3) * vector(3);
endfunction

vector = [2 1 3];
sum(diag(numderivative(func,vector)))
```

The answer will be equal to, 12.

25) The *SciLab* programs are written below.
Part (a)

```
function f = func(vector)
f(1) = vector(1) * vector(2).^2;
f(2) = -vector(1) * vector(2) * vector(3);
f(3) = vector(1).^2 * vector(3).^2;
endfunction

vector = [2 1 3];
a = numderivative(func,vector)
A = [a(3,2)-a(2,3) a(1,3)-a(3,1) a(2,1)-a(1,2)]
```

The answer will be equal to, A = 2   -36   -7

Part (b)

```
function f = func(vector)
f(1) = vector(2).^2 * vector(3).^2;
f(2) = -vector(1) * vector(3);
f(3) = vector(1) * vector(2);
endfunction

vector = [2 1 3];
a = numderivative(func,vector)
```

```
A = [a(3,2)-a(2,3) a(1,3)-a(3,1) a(2,1)-a(1,2)]
```

The answer will be equal to, A = 4 5 -21

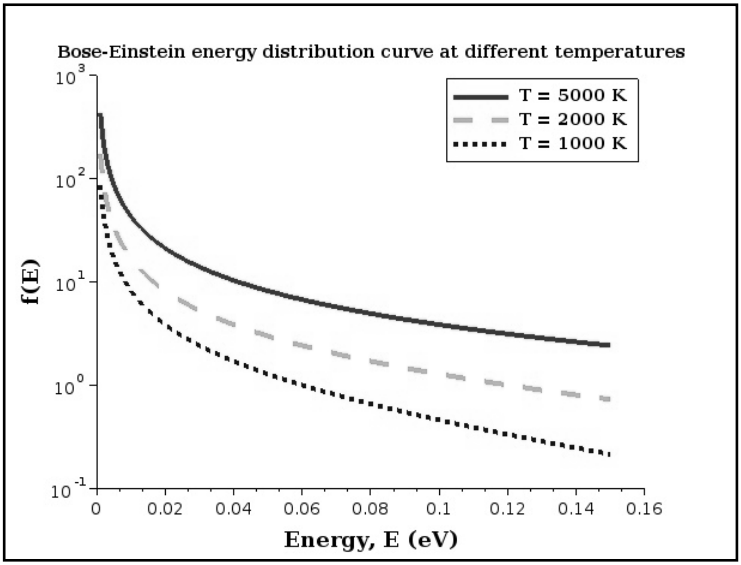

Figure 2.56: Solution for *Exercise 19*

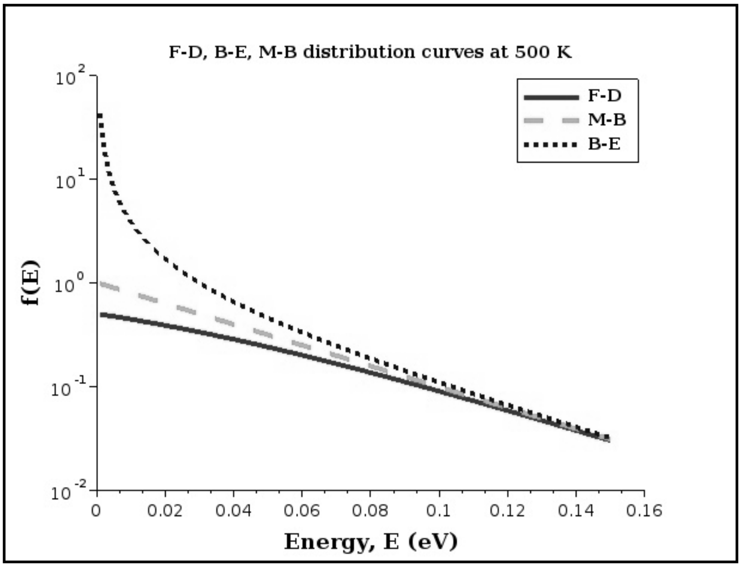

Figure 2.57: Solution for *Exercise 20*

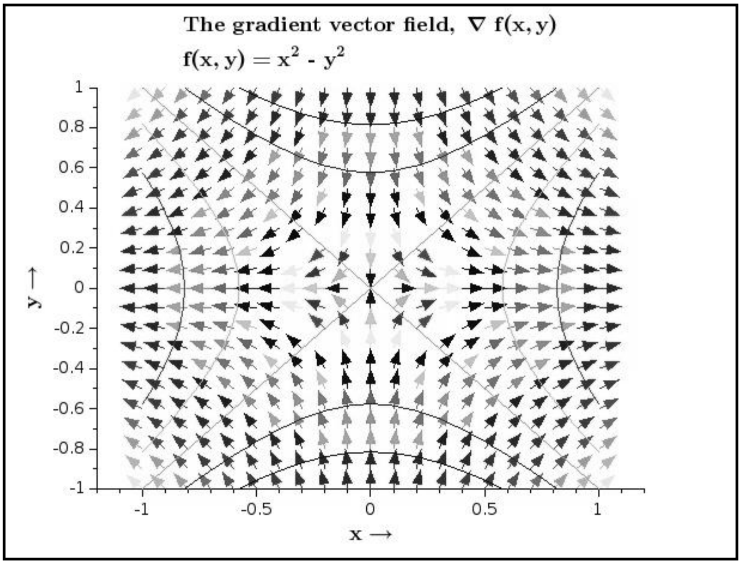

***Figure 2.58*:** Solution for *Exercise 21*

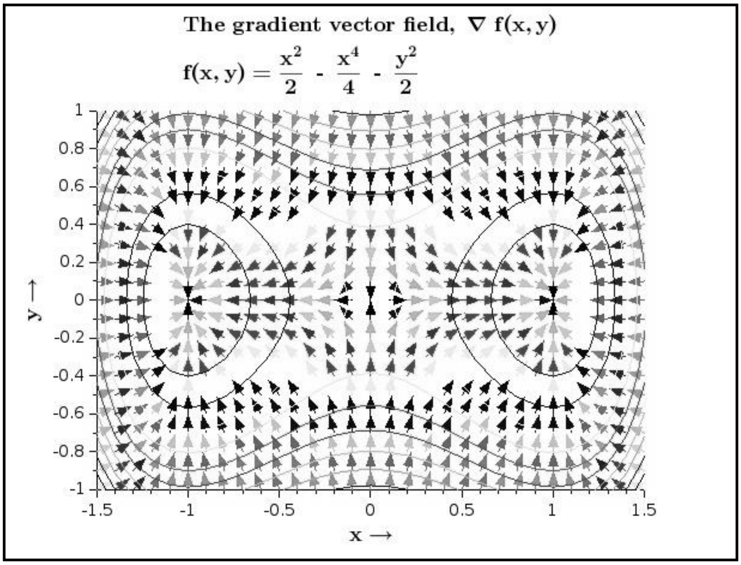

***Figure 2.59*:** Solution for *Exercise 22*

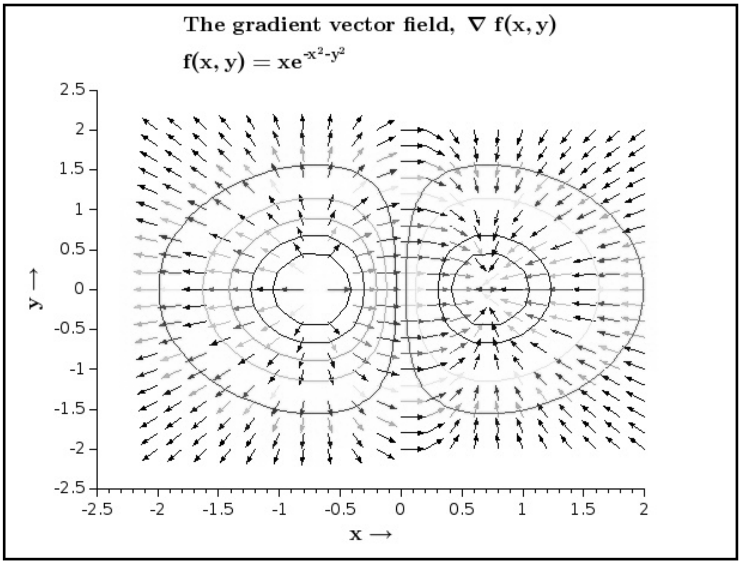

***Figure 2.60*:** Solution for *Exercise 23*

CHAPTER 3

1) The *SciLab* program is written below and graph is shown in *Figure 3.20.*

```
//Generate a data set having a linear trend and a
negative slope
x = linspace(1,10,10);
n = length(x);                                           //n = 10
y = -rand(1,n).*x                      //Generate random y-values
y = y - min(y)                               //Rescale the y-axis
plot2d(x,y)                                       //Plot the data

//Steps to determine the value of slope and constant
x1 = sum(x);                                 //$\sum_{i=1}^{n} x_i$
x2 = sum(x.*x);                              //$\sum_{i=1}^{n} {x_i}^2$
x1y1 = sum(x.*y);                            //$\sum_{i=1}^{n} x_i\, y_i$
y1 = sum(y);                                 //$\sum_{i=1}^{n} y_i$

//Define the two matrices
A = [x2 x1; x1 n];
B = [x1y1; y1];
C = A\B;
m = C(1);                                                //Slope
c = C(2);                                             //Constant
z = x(1)/2:0.01:x(n)+x(1)/2.0;
fplot2d(z,bestfit)                    //Plot the best fit curve
```

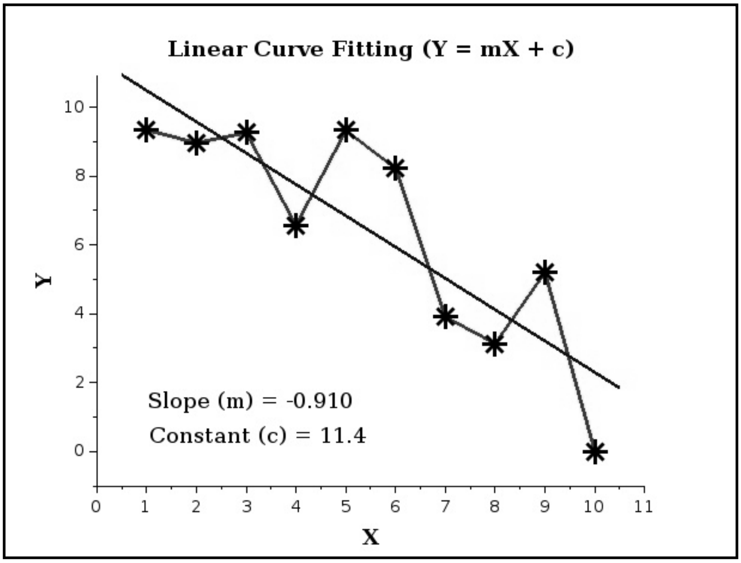

***Figure 3.20*:** Graph for *Exercise 1*

3) The *SciLab* program is written below. The best fit curve is shown in *Figure 3.21*.

```
x = linspace(0.1,1,10);
n = length(x);

//Generate random y-values around $e^{-3x}$
y = %e^(-3*x)+(min(y)*rand(1,n));
plot2d(x,y)

//Best fit parameters and plot the best fit curve
[bestfit,m,c] = exponential_fit(x,y)
plot2d(x,bestfit)
```

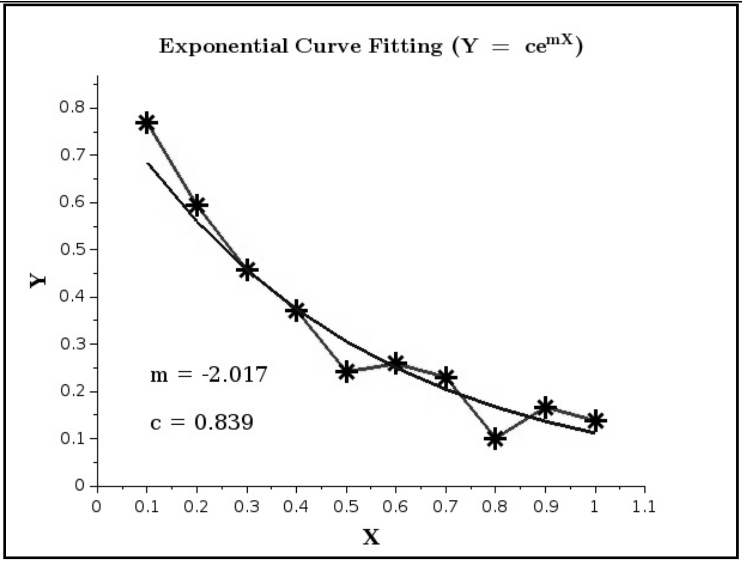

Figure 3.21**:** Graph for *Exercise 3*

4) The *SciLab* program is written below and the curve is shown in *Figure 3.22*.

```
// Define function for best fit
function s = bestfit(z)
s = alpha*(z^m);
endfunction

n = input("Please enter the value of data points, n = ")

x = [10 20 30 40 50 60 70 80 90 100];
y = [55 210 440 794 1205 1812 2451 3172 4022 5020];

// Calculate $\sum_{i=1}^{n} X_i = \sum_{i=1}^{n} \log(x_i)$
```

```
x1 = 0;
for i = 1:n;
    x1 = x1 + log(x(i));
end

// Calculate $\sum_{i=1}^{n} {X_i}^2 = \sum_{i=1}^{n} \{\log(x_i)\}^2$
x2 = 0;
for i = 1:n;
    x2 = x2 + (log(x(i)))^2;
end
//Calculate $\sum_{i=1}^{n} X_i Y_i = \sum_{i=1}^{n} \log(x_i)\log(y_i)$
x1y1 = 0;
for i = 1:n;
    x1y1 = x1y1 + log(x(i))*log(y(i));
end

//Calculate $\sum_{i=1}^{n} Y_i = \sum_{i=1}^{n} log(y_i)$
y1 = 0;
for i = 1:n;
    y1 = y1 + log(y(i));
end

//Calculate slope $(m = \beta)$
m = (n*x1y1-(x1*y1))/(n*x2-(x1)^2);

//Calculate $\alpha$
alpha = exp(((x2*y1)-(x1*x1y1))/(n*x2 -(x1)^2));

//Range for plotting and plot the best fit curve
z= x(1)-x(1)/2:(x(2)-x(1))*0.1:x(n)+1.5*x(1);
fplot2d(z,bestfit);

//Plot the data points
for i = 1:n;
    plot(x(i),y(i),'r-*')
end
```

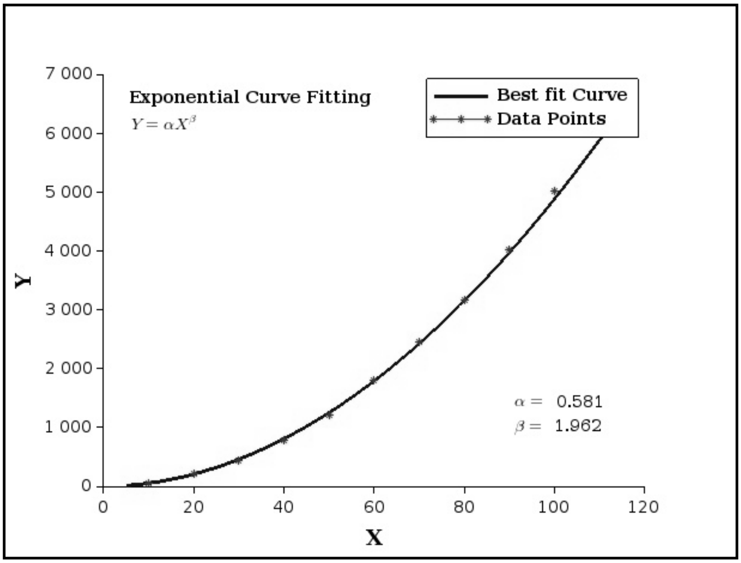

***Figure 3.22*:** Graph for *Exercise 4*

8) The *SciLab* program is written below. The solution is shown in *Figure 3.23*.

```
x = [2 4 6 8 10 12 14 16 18];
y = [0 0.2 1.1 1.2 1 2 1.9 1.8 2.5];
plot2d(x, y)

function y = model(x,constant)
y = constant(1)*x + exp(-constant(2)*x).*sin(x +
constant(3));
endfunction

function err = model_error(constant,z)
  err = z(2) - model(z(1),constant);
endfunction

z = [x ; y];
constant_trial = [0 1 1.5]';

[best_fit_constant, err] = datafit(model_error, z,
constant_trial);

x = linspace(0, 20, 100);
y = model(x,best_fit_constant);
plot2d(x, y)
```

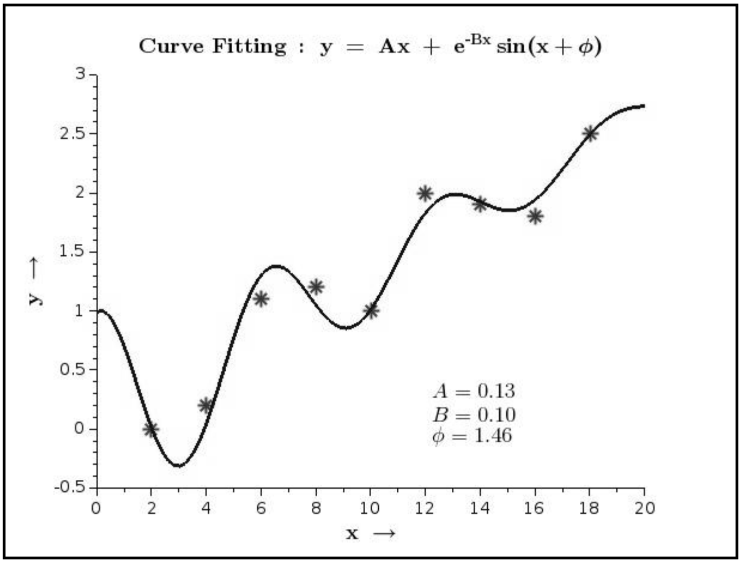

Figure 3.23**:** Solution for *Exercise 8*

9) The *SciLab* program is written below. The solution is shown in *Figure 3.24*.

```
x = [1 2 3 4 5 6 7 8 9 10 11 12];
y = [4.1 3.8 3 1.1 0.9 1.1 1.1 1 2 4.8 5.1 5];
plot2d(x, y)

function a = model(x,constant_trial)
i = 1;
for y = min(x):0.01:max(x)

if y <= constant_trial(1) then
    a(i) = constant_trial(2)
elseif (y>constant_trial(1))&(y<=constant_trial(3)) then
    m = (constant_trial(2)-
constant_trial(5))/(constant_trial(3)-constant_trial(1))
    a(i) = -m*(y-constant_trial(1)) + constant_trial(2);
elseif (y > constant_trial(3)) &  (y <=
constant_trial(4)) then
    a(i) = constant_trial(5);
elseif (y > constant_trial(4)) & (y<constant_trial(6))
then
    m = (constant_trial(7)-
constant_trial(5))/(constant_trial(6)-constant_trial(4))
    a(i) = m*(y-constant_trial(4)) + constant_trial(5);
else
    a(i) = constant_trial(7);
```

```
end
i = i+1;
end
endfunction

function err = model_error(constant,z)
err = z(2) - model(z(1),constant);
endfunction

z = [x ; y];
constant_trial = [2 4 4 8 1 10 5]';

[best_fit_constant, err] = datafit(model_error, z,
constant_trial);

x1 = 0:0.01:13;
y1 = model(x1,best_fit_constant);

plot2d(x1,y1)
```

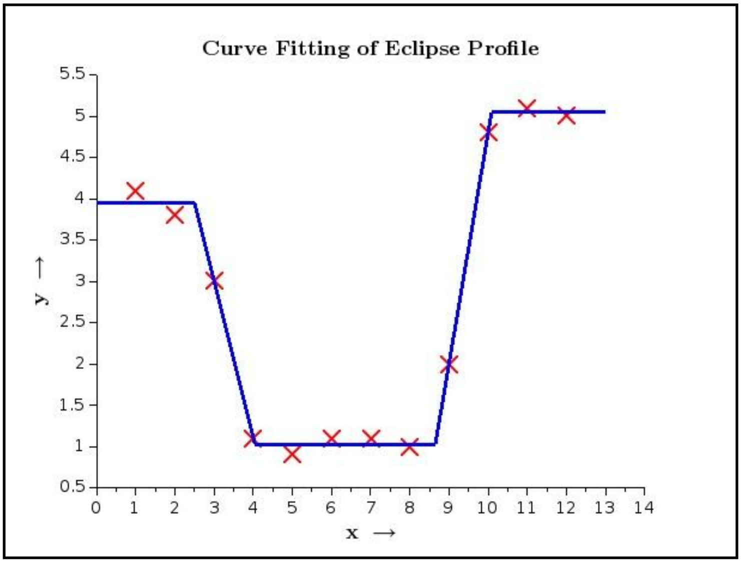

***Figure 3.24*:** Solution for *Exercise 9*

CHAPTER 4

1) The exact solution of the differential equation is,

$$Y = \exp(-X)$$

Figure 4.25 shows the solution of the differential equation obtained by using the '`ode`' in-built function of *SciLab*. The 'X' range is taken to be 0 to 1. The *SciLab* program written for this graph is as follows.

```
//Define the function for differential equation
function dy = f(x,y);
dy = -y;
endfunction

x(1) = 0;                                   //Initial value of X
y(1) = 1;                                   //Initial value of Y
final = 1;                                    //Final value of X
h = 0.1;                                             //Step size

j = x(1);
k = 1;
while(j<=max(x));
    ode_result(k) = ode(y(1),x(1),j,f);
    j = j+h;
    k = k+1;
end

plot2d(x,ode_result)
```

For the same function and initial values, the functions for Euler's and Runge-Kutta methods can be loaded by using the following command.

```
exec('differentiation.sci',-1)
```

These functions are then invoked through the following commands.

```
[x,y] = euler(x(1),y(1),h,final);
plot2d(x,y)

[x,y] = modeuler1(x(1),y(1),h,final);
plot2d(x,y)

[x,y] = rk2(x(1),y(1),h,final);
plot2d(x,y)

[x,y] = rk4(x(1),y(1),h,final);
plot2d(x,y)
```

It is clear from *Figure 4.25*, that Runge-Kutta methods give more accurate results as compared with the Euler's methods for the same step size.

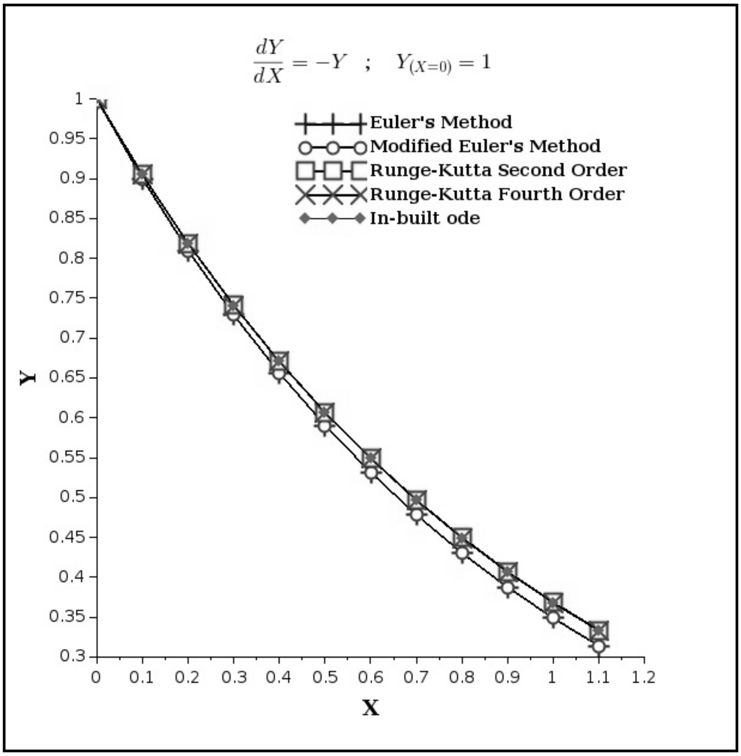

***Figure 4.25*:** Solution for *Exercise 1*

2) The exact solution of this differential equation is,

$$y = 5e^{-2t}$$

Figure 4.26 shows the comparison of the analytical solution with the estimates from Euler's method. The graph corresponding to the analytical solution is obtained by using the following commands.

```
t(1) = 0;                           //Initial value of t
y(1) = 5;                           //Initial value of y
final = 2;                            //Final value of t

//Define the function for differential equation
function ydot = f(t,y)
ydot = -2*y;
endfunction

//Plot the analytical solution
t = 0:0.2:2;
plot2d(t,y(1)*exp(-2*t));
```

For the same function and initial values, the function for Euler's method can be loaded using the following command. The function for the Euler's method is then invoked for different step size.

```
exec('differentiation.sci',-1)
h = 0.2;
[t,y] = euler(t(1),y(1),h,final);
plot2d(t,y)

h = 0.8;
[t,y] = euler(t(1),y(1),h,final);
plot2d(t,y,)
```

It is clear from *Figure 4.26* that a better estimate of the solution is obtained if the step size is small.

3) The exact solution of this differential equation is,

$$y = \frac{1}{1 + e^{-x}}$$

Figure 4.27 shows the comparison of the analytical solution with the estimates from Euler's method. The graph corresponding to the analytical solution is obtained by using the following commands.

```
x(1) = -4;                          //Initial value of x
y(1) = 0.018;                       //Initial value of y
final = 4;                            //Final value of t

//Define the function for differential equation
function yprime = f(x,y)
yprime = exp(x)./((1 + exp(x)).^2);
endfunction

//Plot the analytical solution
x = -4:0.4:4;
plot2d(x,1 ./(1 + exp(-x)))
```

For the same function and initial values, the function for Euler's method can be loaded by using the following command. The function for the Euler's method is then invoked for a step size of 1.

```
exec('differentiation.sci',-1)

h = 1;
[x,y] = euler(x(1),y(1),h,final);
plot2d(x,y)
```

It is clear from *Figure 4.27* that Euler's method gives a large amount of error when the step size is large.

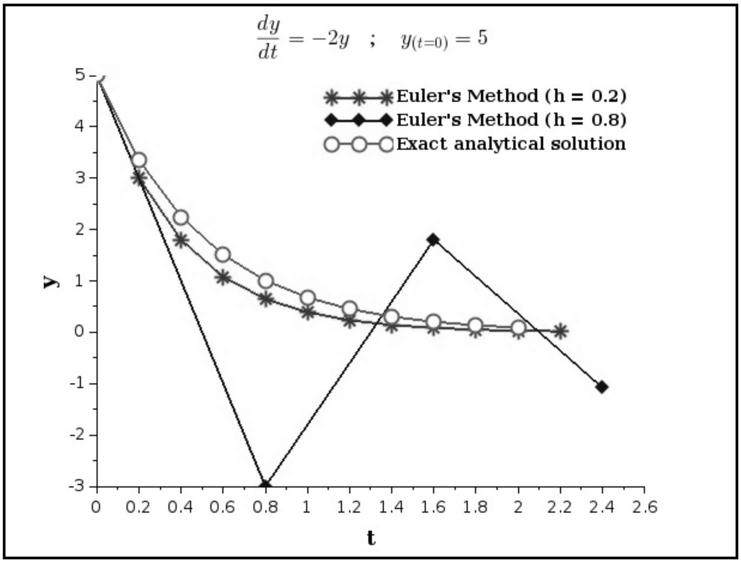

Figure 4.26: Solution for *Exercise 2*

4) The differential equation taken for a concave down solution curve is,

$$\frac{dy}{dx} = -2x$$

The differential equation taken for a concave up solution curve is,

$$\frac{dy}{dx} = 2x$$

The solution curves shown in *Figures 4.28 - 4.29* are expected for these differential equations.

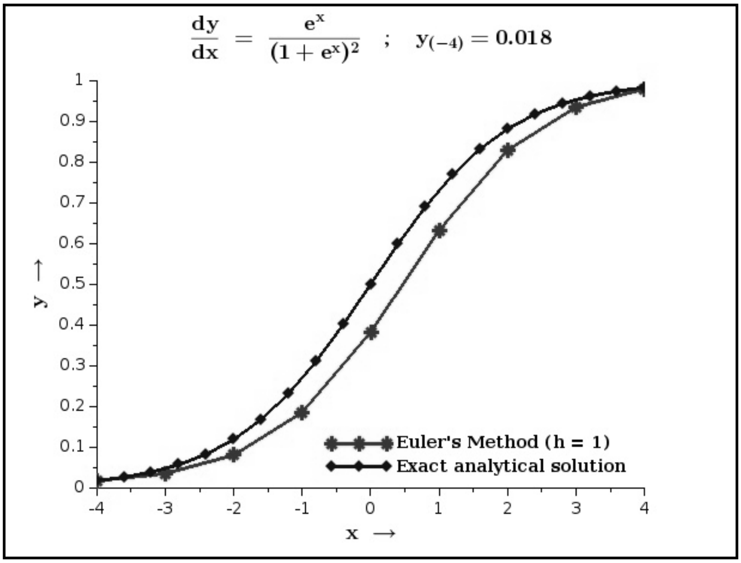

***Figure 4.27*:** Solution for *Exercise 3*

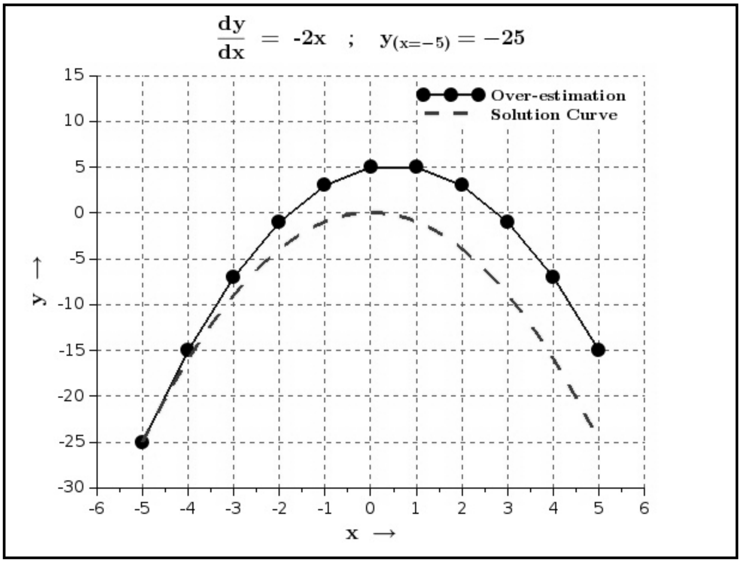

***Figure 4.28*:** Solution for *Exercise 4*

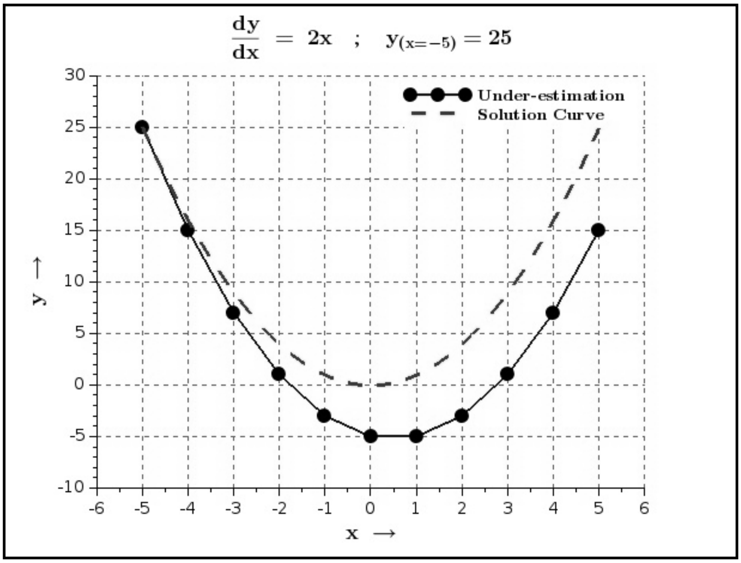

***Figure 4.29*:** Solution for *Exercise 4*

5) The procedure to solve the differential equation for the orthogonal curves is exactly same as discussed in the text (*Section 4.8.2*). The solution curves shown in *Figure 4.30* are expected for this exercise.

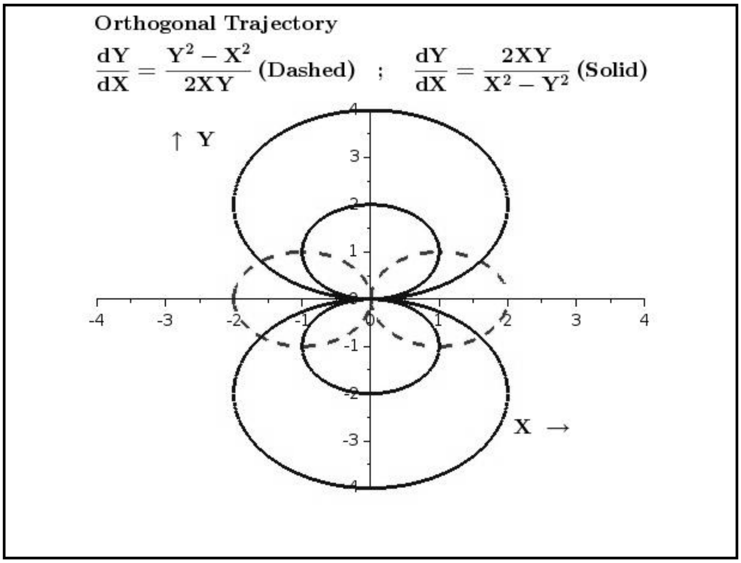

***Figure 4.30*:** Solution for *Exercise 5*

6) The analytical solution of the differential equation is,

$$x^2 + y^2 = 1$$

This is equation of a circle and it can also be written in terms of polar coordinates.

$$x = \sin\theta$$
$$y = \cos\theta$$

The *SciLab* program for generating the analytical solution is,

```
angle = 0:0.2:2*%pi;
plot2d(sin(angle), cos(angle));
```

The two first order equations can be written in the form of a function.

```
function xdot = f1(t,x,y)
xdot = y;
endfunction

function ydot = f2(t,x,y)
ydot = -x;
endfunction
```

The initial conditions can be defined as,

```
t(1) = 0;
x(1) = 1;
y(1) = 0;
final = 2*%pi;
h = 0.2;
```

The functional form of the Euler's method and the Runge-Kutta method can be loaded using the following command and are then evoked and the result is plotted by using the following commands.

```
exec('differentiation.sci',-1)

[t,x,y] = euler2(t(1),x(1),y(1),h,final);
plot2d(x,y);

[x,y,z] = rk42(x(1),y(1),z(1),h,final);
plot2d(x,y)
```

Figure 4.31 and *Figure 4.32* shows the comparative graphs for the Euler's method and the Runge-Kutta method.

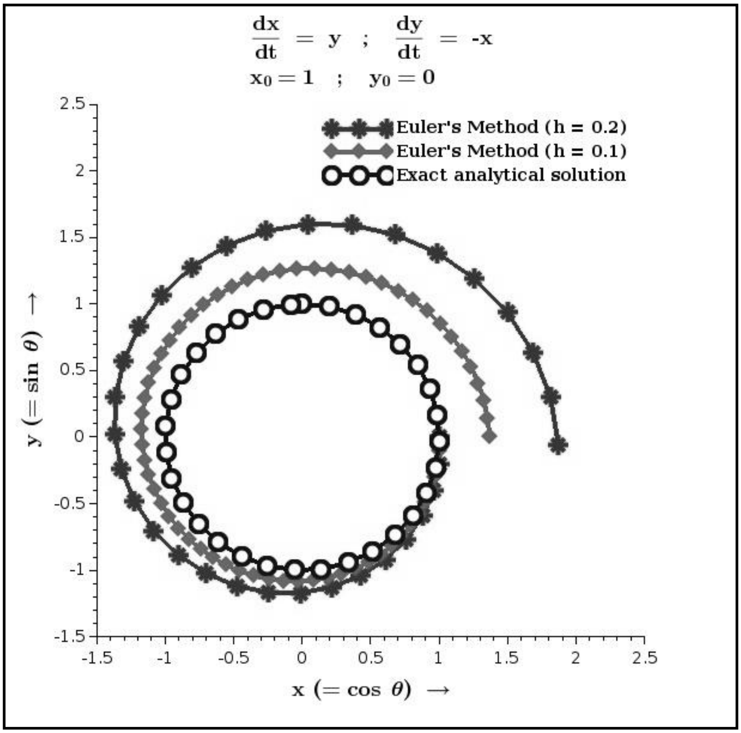

Figure 4.31**:** Solution for *Exercise 6*

7) The *SciLab* program is written below and the solution curve is shown in *Figure 4.33*.

```
exec('differentiation.sci',-1)
time(1) = 0;                                    //Initial time
temp(1) = 80;                            //Initial temperature
Ts = 20;                          //Temperature of surrounding
final_time = 100;                                 //Final time
h = 0.1;                                           //Step size
alpha = 0.02;

function T_dt = f(t,T);
T_dt = -alpha*(T-Ts);
endfunction

[x,y] = rk4(time(1),temp(1),h,final_time);
plot2d(x,y,2)

alpha = 0.04;
[x,y] = rk4(time(1),temp(1),h,final_time);
plot2d(x,y,5)
```

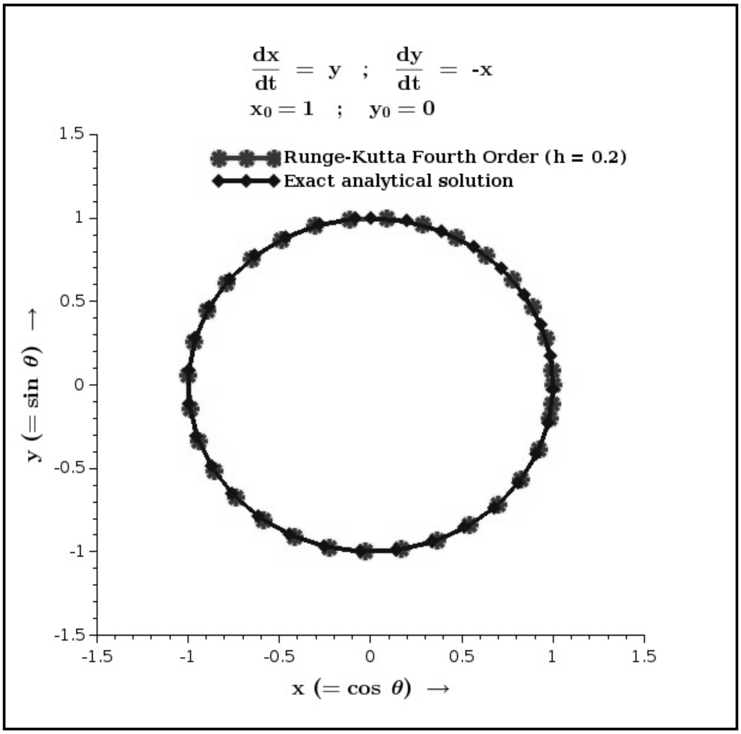

Figure 4.32**:** Solution for *Exercise 6*

8) The solution curve in *Figure 4.34* is expected for the logistic growth model.

9) The analytical solution of the differential equation is,

$$y = 3\cos x + 5\sin x$$

The second order differential equation can be re-written in the form of coupled first order differential equations in the following manner.

Let,

$$\frac{dy}{dx} = z$$

This implies

$$\frac{dz}{dx} = \frac{d^2y}{dx^2} = -y$$

These two equations can be written in the form of a function.

```
function yprime = f1(x,y,z)
yprime = z;
endfunction

function zprime = f2(x,y,z)
zprime = -y;
endfunction
```

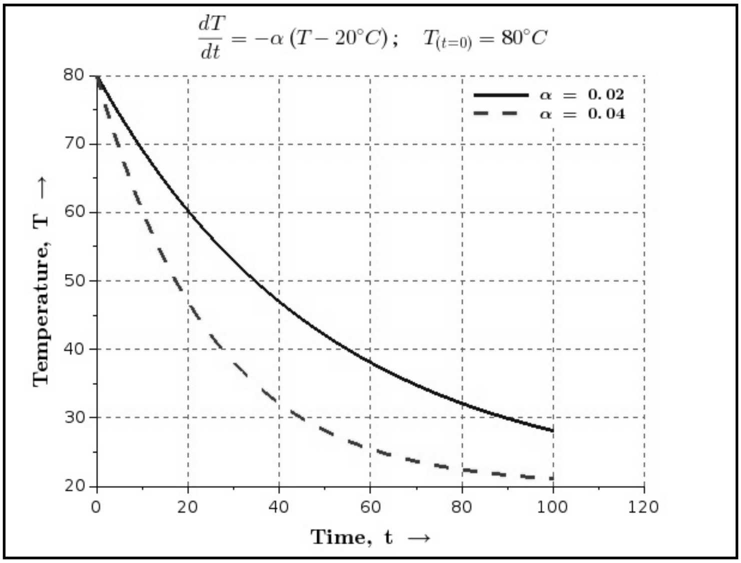

Figure 4.33**:** Solution for *Exercise 7*

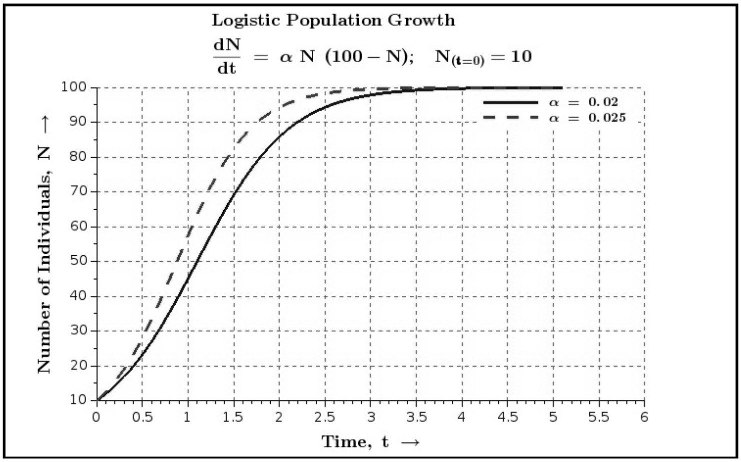

Figure 4.34**:** Solution for *Exercise 8*

The initial conditions can be defined as,

```
x(1) = 0;
y(1) = 3;
z(1) = 5;
final = 3*%pi;
h = 0.1;
```

The functional form of the Euler's method and the Runge-Kutta method can be loaded by using the following command.

```
exec('differentiation.sci',-1)
```

These functions are called and the result is plotted by using the following commands.

```
[x,y,z] = euler2(x(1),y(1),z(1),h,final);
plot2d(x,y);

[x,y,z] = rk42(x(1),y(1),z(1),h,final);
plot2d(x,y)
```

The analytical solution can be plotted by using the following command.

```
x = 0:0.1:3*%pi;
plot2d(x,3*cos(x) + 5*sin(x));
```

The in-built function of *SciLab* can be used in the following manner.

```
y0 = [3;5];

function dy = f(x,y);
dy(1) = y(2);
dy(2) = -y(1);
endfunction

x = 0:0.1:3*%pi;
y = ode(y0,0,x,f);
plot2d(x,y(1,:));
```

Figures 4.35, 4.36 and 4.37 show the comparative graphs for Euler's method, Runge-Kutta method and the in-built *SciLab* function respectively.

10) The phase space plot is shown in *Figure 4.38.*

11) For critically damped case, $c^2 = 4mk$
This implies, $c = \sqrt{4mk} = 160$
Therefore, damping constant in the following *SciLab* program is taken to be equal to 160. For comparison, over-damped case is taken to be double the critical damping and under-damped case is taken to be half the critical damping. The graph is shown in *Figure 4.39.*

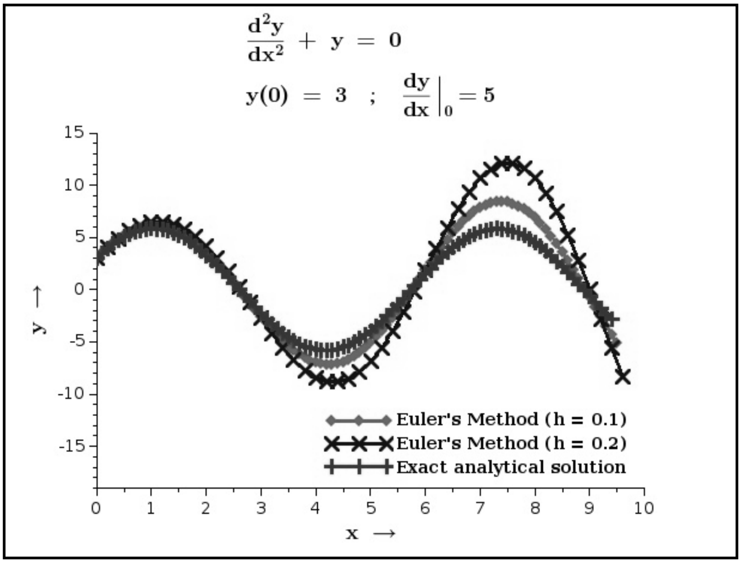

***Figure 4.35*:** Solution for *Exercise 9*

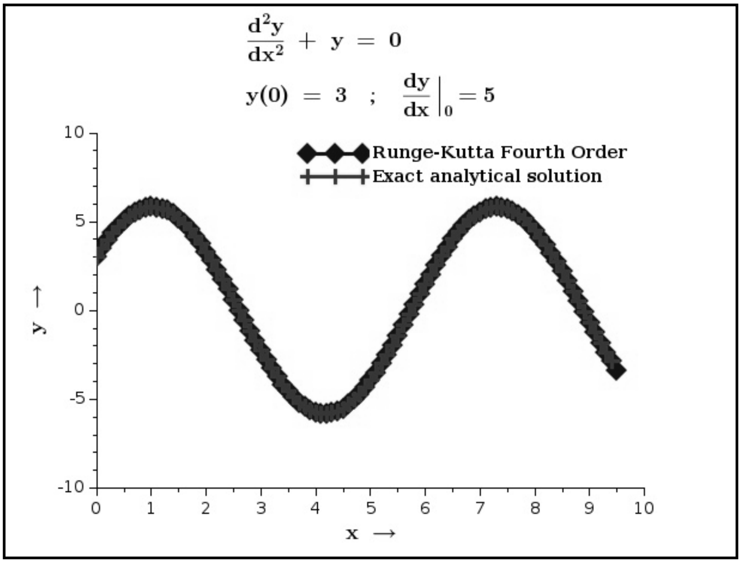

***Figure 4.36*:** Solution for *Exercise 9*

```
exec('differentiation.sci',-1)

function x_dot = f1(t,x,y)
x_dot = y;
endfunction
```

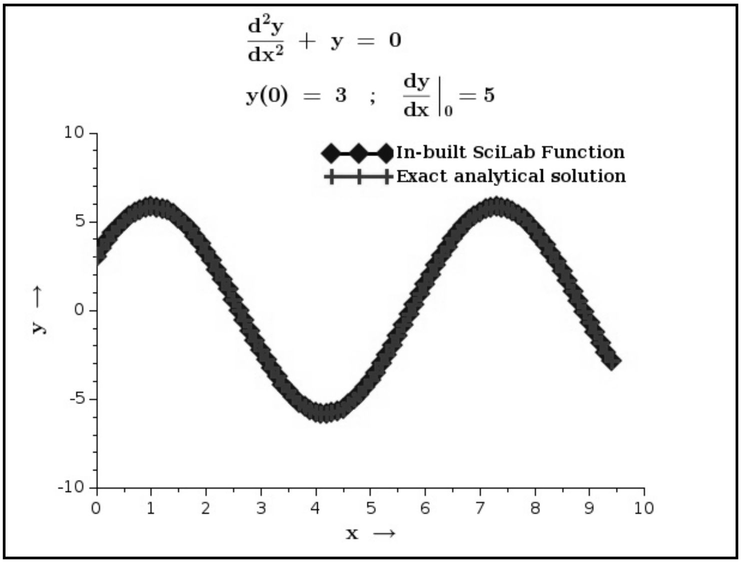

***Figure 4.37*:** Solution for *Exercise 9*

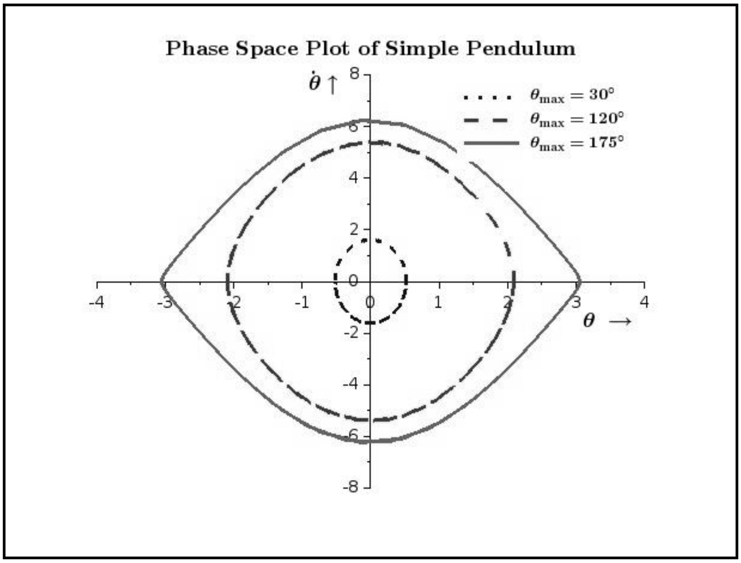

***Figure 4.38*:** Solution for *Exercise 10*

```
function y_dot = f2(t,x,y)
y_dot = -(c*y + k*x)/m;
endfunction

t0 = 0;                                          //Initial time
```

```
x0 = 1;                               //Initial position
xdot_0 = 0;                           //Initial velocity
final = 8;                                  //Final time
h = 0.1;                                     //Step size
m = 50;                             //Mass of the object
k = 128;                               //Spring constant

c = sqrt(160*160/2);
[t,x,y] = rk42(t0,x0,xdot_0,h,final);
plot2d(t,x);

c = 160;
[t,x,y] = rk42(t0,x0,xdot_0,h,final);
plot2d(t,x);

c = sqrt(2*160*160);
[t,x,y] = rk42(t0,x0,xdot_0,h,final);
plot2d(t,x);
```

12) The general form of the second order differential equation is,

$$\frac{d^2y}{dx^2} + f(x)\frac{dy}{dx} + g(x)y = r(x)$$

As explained in the text, it is necessary to define the functions $f(x)$, $g(x)$ and $r(x)$ for the given differential equation; give the boundary conditions; and then call the function for the finite difference method. The following *SciLab* program shows this for part (a) of the question.

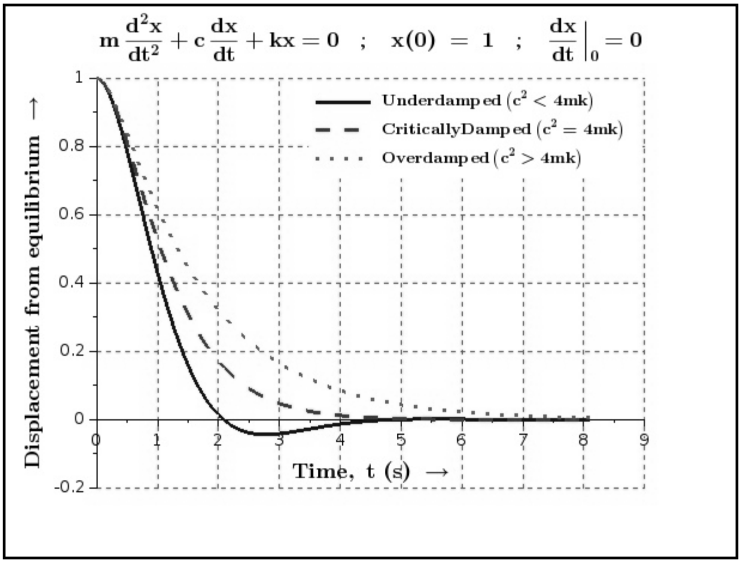

***Figure 4.39*:** Solution for *Exercise 11*

```
function def_f = f(x)             //Define function f(x)
def_f = 0;
endfunction

function def_g = g(x)             //Define function g(x)
def_g = 1;
endfunction

function def_r = r(x)             //Define function r(x)
def_r = 0;
endfunction

exec('differentiation.sci',-1);
a = 0;                              //Initial value of x
b = %pi/2;                            //Final value of x
ya = 1;                             //Initial value of y
yb = 1;                               //Final value of y
h = 0.01;                                    //Step Size

//Call the function for finite difference method
[x,y] = boundary(a,b,h,ya,yb,f,g,r);

plot2d(x,y)                             //Plot the result
```

Figure 4.40 shows the solution curve for the given differential equation.

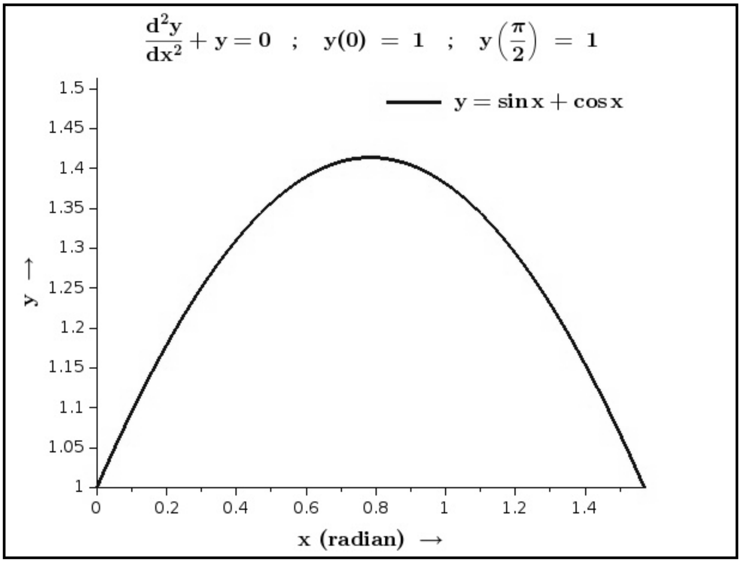

Figure 4.40: Solution for *Exercise 12(a)*

The following graphs are expected for the parts, (b), (c) and (d) of this question.

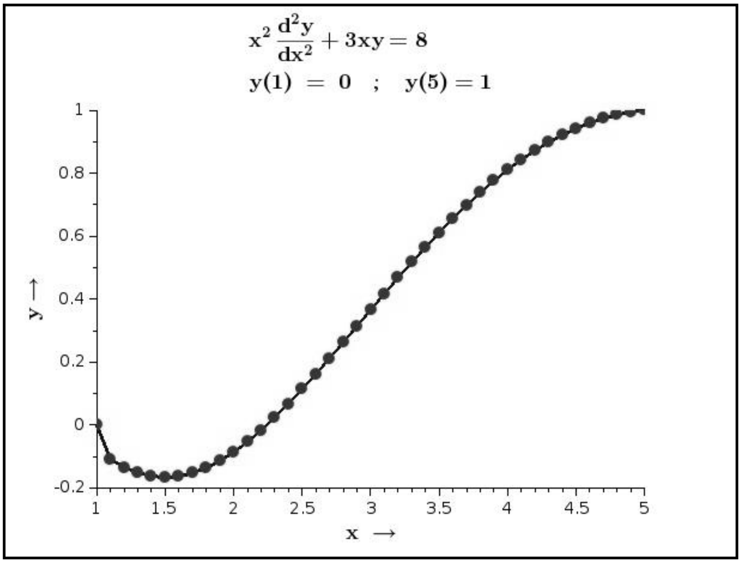

***Figure 4.41*:** Solution for *Exercise 12(b)*

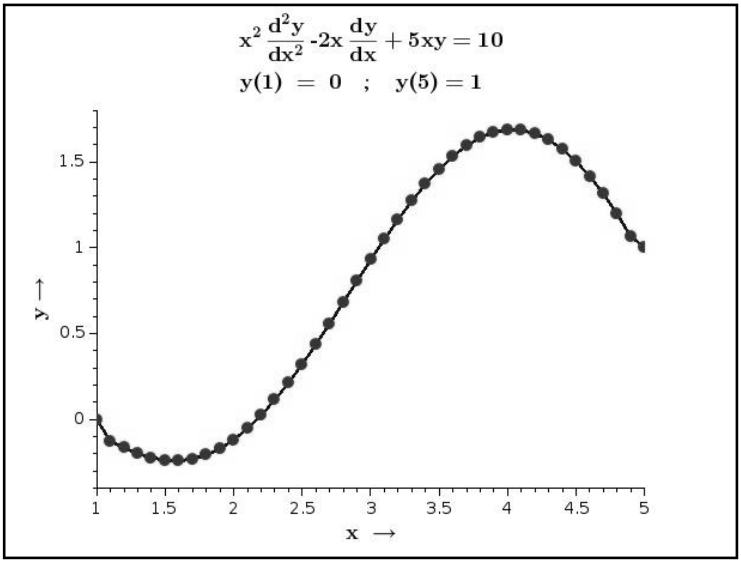

***Figure 4.42*:** Solution for *Exercise 12(c)*

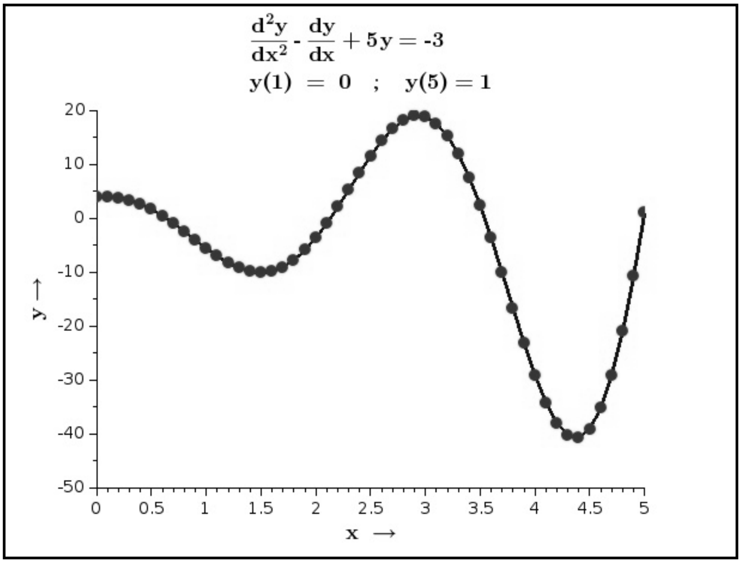

***Figure 4.43*:** Solution for *Exercise 12(d)*

13) The force due to air resistance is proportional to the speed of the object, and it acts in the direction opposite to motion. Therefore, the acceleration of the freely falling object will be given by,

$$\frac{d^2x}{dt^2} = \frac{dv}{dt} = -g - \alpha v$$

The *SciLab* program to determine the velocity profile of the object is written below. The velocity-time graph is shown in *Figure 4.44*.

```
//Define system of equations for freely falling object
function xdot = f_1(t,x)
xdot(1) = x(2);
xdot(2) = -g;
endfunction

//Define system of equations for object falling under
air resistance, α is taken to be 0.7
function xdot = f_2(t,x)
xdot(1) = x(2);
xdot(2) = -g-(0.7*x(2));
endfunction

g = 9.82;                     //Acceleration due to gravity
height_initial = 100;                    //Initial height
v_initial = 0;                         //Initial velocity
t_initial = 0;                             //Initial time
```

```
t = t_initial:0.3:((v_initial) +
(sqrt((v_initial*v_initial)+2*g*height_initial)))/g;

//Call the in-built function
x = ode([height_initial;v_initial],t_initial,t,f_1);

//Plot the velocity profile of freely falling object
plot2d(t,x(2,:));

//Call the in-built function
x = ode([height_initial;v_initial],t_initial,t,f_2);

//Plot the velocity profile of object falling under air
resistance
plot2d(t,x(2,:));
```

14) The graph shown in *Figure 4.45* is expected for this exercise.

15) The graph shown in *Figure 4.46* is expected for this exercise.

16) The graph shown in *Figure 4.47* is expected for this exercise.

17) The *SciLab* program is written below. The solution curve is shown in *Figure 4.48.*

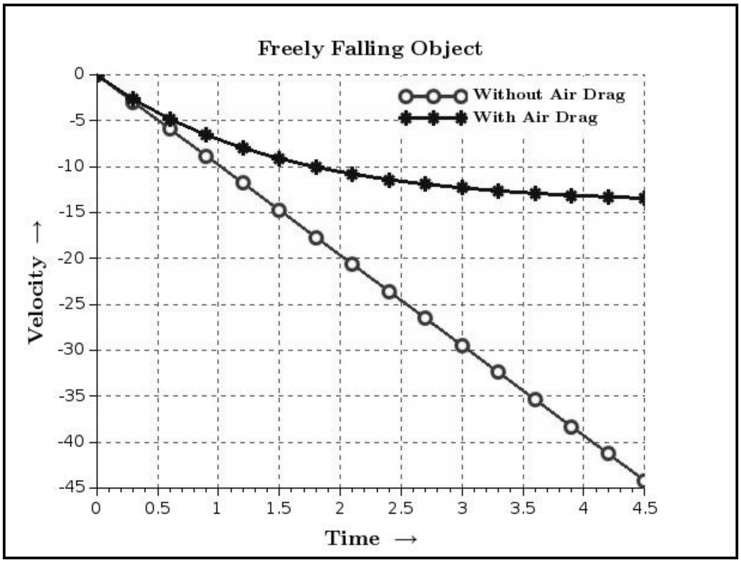

***Figure 4.44*:** Solution for *Exercise 13*

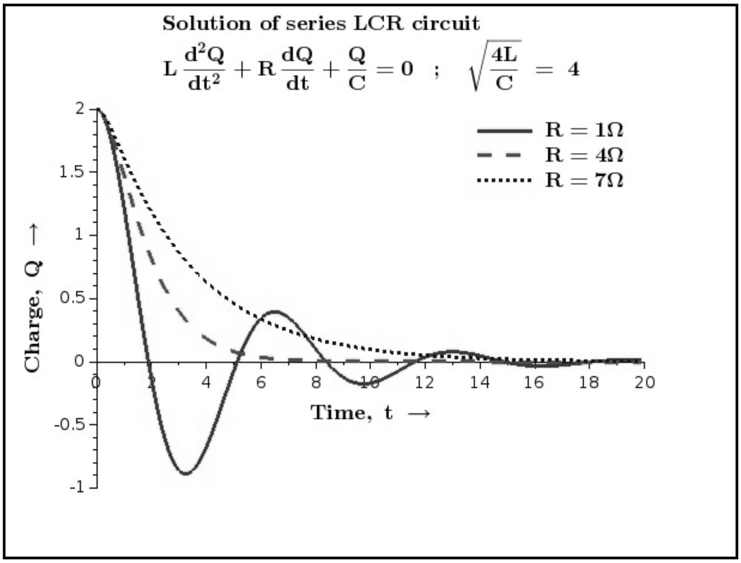

***Figure 4.45*:** Solution for *Exercise 14*

```
function ydash = f(x,y)
ydash(1) = y(2);
ydash(2) = y(3);
ydash(3) = 5*sin(2*x) - 3*y(2);
endfunction
y_0 = 0;
ydash_0 = 0;
ydash_dash_0 = 0;
x = 0:0.1:48;
y = ode([y_0 ; ydash_0 ; ydash_dash_0],0,x,f);
plot2d(x,y(1,:))
```

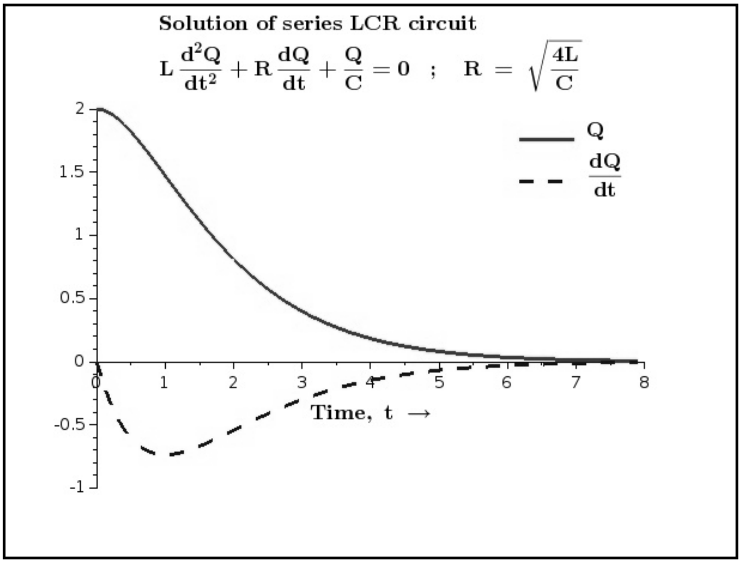

***Figure 4.46*:** Solution for *Exercise 15*

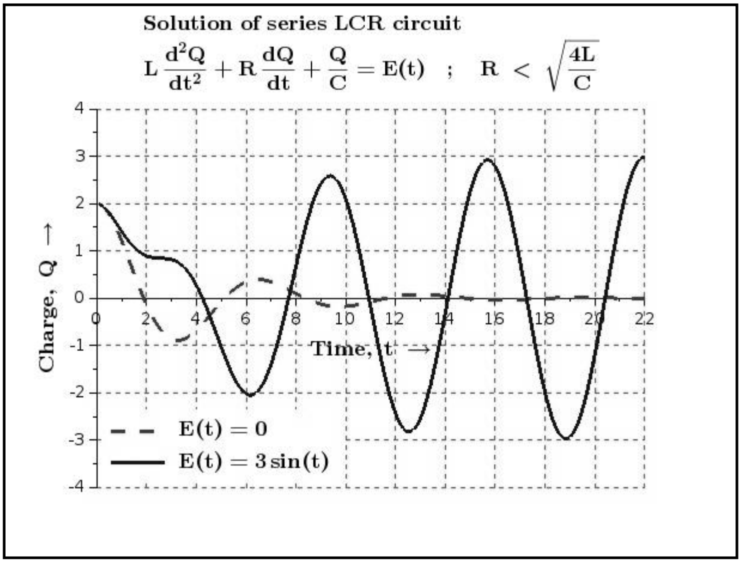

***Figure 4.47*:** Solution for *Exercise 16*

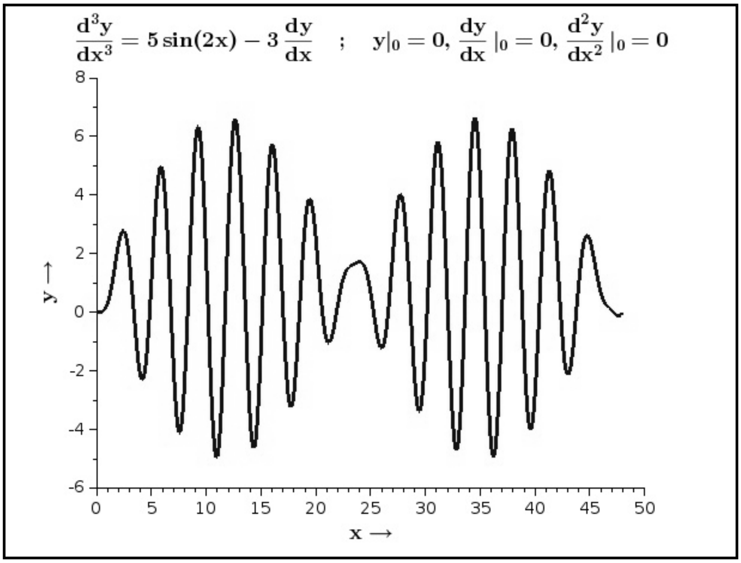

***Figure 4.48*:** Solution for *Exercise 17*

18) Based on the notations used in *Section 4.8.10.2*, the following *SciLab* commands can be used for each of the parts of this question.

a) To show that the wave functions are orthogonal, use the following *SciLab* commands. The output of each command is given in the table below.

Command	**Output**
sum(eigenvector(:,1).* eigenvector(:,1))	1
sum(eigenvector(:,1).* eigenvector(:,2))	- 6.356D-16
sum(eigenvector(:,1).* eigenvector(:,3))	4.642D-16

b) Following *SciLab* commands can be used to determine the value of Bohr radius.

Objective	***SciLab* Command**	**Output**

• Calculate the maximum probability
• Distance at which electron probability is maximum
	`[max_value_1,max_index_1] = max(eigenvector (:,1).* eigenvector (:,1))`	max_value_1 = 0.0204578
		max_index_1 = 26
		r(max_index_1) = 0.52

c) The following *SciLab* commands can be used to determine the electron probability in the 1s orbital.

Objective	***SciLab* Command**	**Output**

Probability that electron lies in the range,
$0 \leq r \leq r_{Bohr}$
	s = 0;
	for i = 1 : max_index_1;
	s = sum(eigenvector(i,1).* eigenvector(i,1)) + s;
	end
		s = 0.3242

Probability that electron lies in the range,
$r_{Bohr} \leq r \leq 2r_{Bohr}$
	probability_range = find(r >= r(max_index_1) & r <= 2*r(max_index_1));
	s = 0;
	for i = min(probability_range) : max(probability_range);
	s = sum(eigenvector(i,1).* eigenvector(i,1)) + s;
	end
		s = 0.4420

Probability that electron lies in the range,
$0 \leq r \leq 10r_{Bohr}$
	probability_range = find(r <= 10*r(max_index_1));
	s = 0;
	for i = min(probability_range) : max(probability_range);
	s = sum(eigenvector(i,1).* eigenvector(i,1)) + s;
	end
		s = 0.9999

Probability that electron lies in the range,
$r_{Bohr} \leq r \leq 10r_{Bohr}$
	probability_range = find(r >= r(max_index_1) & r <= 10*r(max_index_1));
	s = 0;
	for i = min(probability_range) : max(probability_range);
	s = sum(eigenvector(i,1).* eigenvector(i,1)) + s;
	end
		s = 0.6961

d) The following *SciLab* commands can be used to determine the electron probabilities in the 2s orbital.

Objective	***SciLab* Command**	**Output**

Probability that electron lies in the range,
$0 \leq r \leq r_{Bohr}$
	s = 0;
	for i = 1 : max_index_1;
	s = sum(eigenvector(i,2).* eigenvector(i,2)) + s;
	end
		s = 0.0344

Probability that electron lies in the range,
$4r_{Bohr} \leq r \leq 6r_{Bohr}$
	probability_range = find(r >= 4*r(max_index_1) & r <= 6*r(max_index_1));
	s = 0;
	for i = min(probability_range) : max(probability_range);
	s = sum(eigenvector(i,2).* eigenvector(i,2)) + s;
	end
		s = 0.3513

e) The following *SciLab* commands can be used to determine the electron probabilities in the 3s orbital.

Objective	***SciLab* Command**	**Output**

Probability that electron lies in the range, $0 \le r \le r_{Bohr}$	s = 0;
	for i = 1 : max_index_1;
	s = sum(eigenvector(i,3).* eigenvector(i,3)) + s;
	end
		s = 0.0099

Probability that electron lies in the range, $4r_{Bohr} \le r \le 6r_{Bohr}$	probability_range = find(r >= 4*r(max_index_1) & r <= 6*r(max_index_1));
	s = 0;
	for i = min(probability_range) : max(probability_range);
	s = sum(eigenvector(i,3).* eigenvector(i,3)) + s;
	end
		s = 0.0590

Probability that electron lies in the range, $12r_{Bohr} \le r \le 14r_{Bohr}$	probability_range = find(r >= 12*r(max_index_1) & r <= 14*r(max_index_1));
	s = 0;
	for i = min(probability_range) : max(probability_range);
	s = sum(eigenvector(i,3).* eigenvector(i,3)) + s;
	end
		s = 0.1962

19) The *SciLab* program to determine the behavior of wave function for the ground state of electron for different values of screening constant 'a' is given below. The graph of the wave function is shown in *Figure 4.49*.

```
a = 0;                                        //Lower boundary
b = 8;                                        //Upper boundary
h = 0.02;                                         //Step size
n = (b-a)/h;                              //Number of intervals
m = 0.511d6;                          //Mass of electron (eV/c^2)
hbar = 1973;                                  //ħc (in eV Ȧ)
e = 3.795;                   //Electron charge (in (eV Ȧ)^(1/2))
alpha = 2*m/(hbar*hbar);
a1 = 7;                         //Screening constant (in Ȧ)
V = -alpha*e*e;
A = zeros(n,n);
r = zeros(1,n);

r(1) = r(1) + h;
A(1,1) = 2 + (V*h*h*exp(-r(1)/a1)/r(1));
A(1,2) = -1;

for i = 2:n-1;
    r(i) = r(i-1) + h;
    A(i,i-1) = -1;
    A(i,i) = 2 + (V*h*h*exp(-r(i)/a1)/r(i));
    A(i,i+1) = -1;
```

```
end

r(n) = r(n-1) + h;
A(n,n-1) = -1;
A(n,n) = 2 + (V*h*h*exp(-r(n)/a1)/r(n));

[c,d] = spec(A);
E = diag(d)/(alpha*h*h);

plot2d(r,c(:,1))
xgrid(13)
```

Value of ground state energy for different values of screening constant is given below.
Screening Constant = 3Ȧ
Energy = -9.3824434 eV

Screening Constant = 5Ȧ
Energy = - 10.943102 eV

Screening Constant = 7Ȧ
Energy = - 11.662933 eV

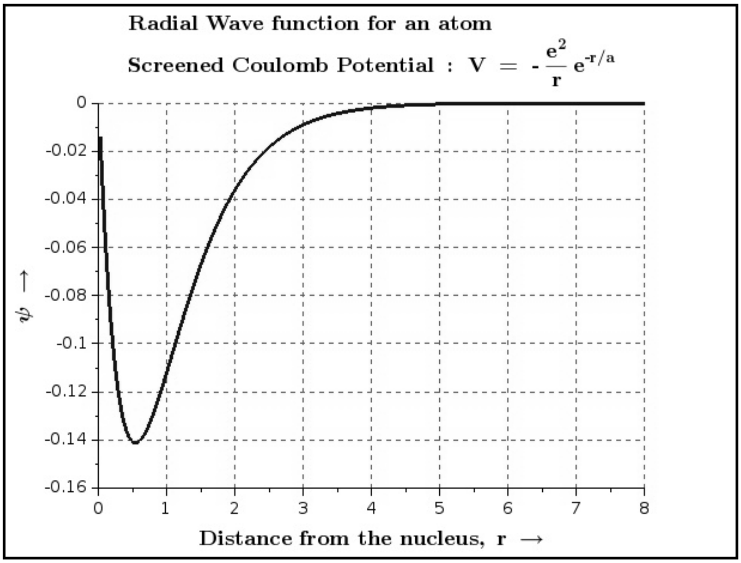

Figure 4.49**:** Graph for *Exercise 19*

(20) The following energy eigen values will be obtained if boundary is taken between 0 to 16.

Energy state	Energy eigen values (eV)
E_1	$4.14 \left(= \frac{3\hbar\omega}{2}\right)$
E_2	$9.66 \left(= \frac{7\hbar\omega}{2}\right)$
E_3	$15.18 \left(= \frac{11\hbar\omega}{2}\right)$

22) The *SciLab* program for an-harmonic oscillator is written below. The wave function for the ground state is shown in *Figure 4.50*.

```
a = 0;                                          //Lower boundary
b = 5;                                          //Upper boundary
h = 0.01;                                            //Step size
n = (b-a)/h;                                //Number of intervals
m = 940;                         //Mass of neutron (in Mev/c^2)
hbar = 197.3;                                  //ħc (in MeV-fm)
k = 100;                     //Positive constant (in MeV-fm^-2)
b = 30;                    //Perturbation factor (in MeV-fm^-3)
alpha = 2*m/(hbar*hbar);
A = zeros(n,n);
r = zeros(1,n);

r(1) = r(1) + h;
A(1,1) = 2 +
(h*h*alpha*((0.5*k*r(1)^2)+((1/3)*b*r(1)^3)));
A(1,2) = -1;

for i = 2:n-1;
   r(i) = r(i-1) + h;
   A(i,i-1) = -1;
   A(i,i) = 2 +
(h*h*alpha*((0.5*k*r(i)^2)+((1/3)*b*r(i)^3)));
   A(i,i+1) = -1;
end

r(n) = r(n-1) + h;
A(n,n-1) = -1;
A(n,n) = 2 +
(h*h*alpha*((0.5*k*r(n)^2)+((1/3)*b*r(n)^3)));

[c,d] = spec(A);
E = diag(d)/(alpha*h*h);
plot2d(r,c(:,1))
```

Ground state energy for different values of perturbation constant (b) are given below.

B	E(1) (in MeV)
0	96.526684
10	100.24487
30	106.89398

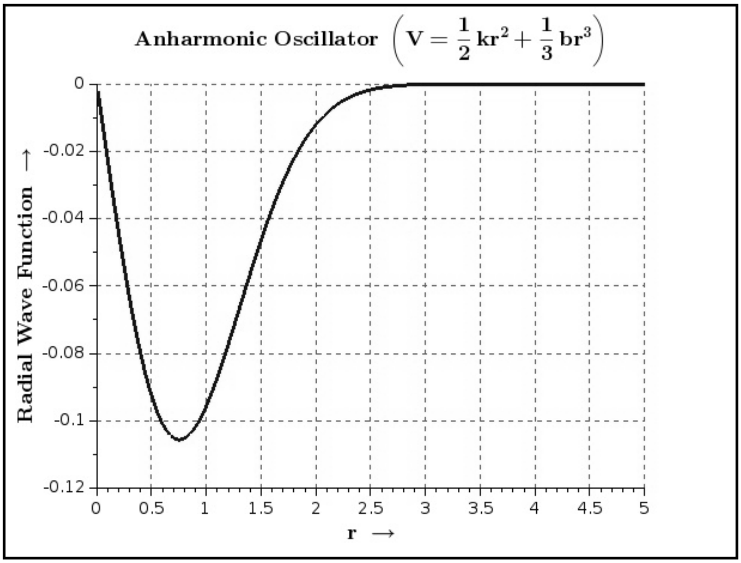

Figure 4.50*: Graph for *Exercise 22

23) The *SciLab* program for the Morse potential is written below. The wave function for the ground state is shown in *Figure 4.51*.

```
a = 0;                                      //Lower boundary
b = 10;                                     //Upper boundary
h = 0.01;                                       //Step size
n = (b-a)/h;                           //Number of intervals
m = 940d6;                       //Mass of neutron (in eV/c^2)
hbar = 1973;                               //ħc (in eV-Å)
D = 0.755501;                          //Dissociation energy
aa = 1.44;                              //Width of potential
r0 = 0.131349                    //Equilibrium bond distance
alpha = 2*m/(hbar*hbar);

A = zeros(n,n);
r = zeros(1,n);
```

```
r(1) = r(1) + h;
A(1,1) = 2 + (h*h*alpha*(D*((exp(-2*aa*((r(1)-
r0)/r(1))))-(exp(-aa*((r(1)-r0)/r(1)))))));
A(1,2) = -1;

for i = 2:n-1;
    r(i) = r(i-1) + h;
    A(i,i-1) = -1;
    A(i,i) = 2 + (h*h*alpha*(D*((exp(-2*aa*((r(i)-
r0)/r(i))))-(exp(-aa*((r(i)-r0)/r(i)))))));
    A(i,i+1) = -1;
end

r(n) = r(n-1) + h;
A(n,n-1) = -1;
A(n,n) = 2 + (h*h*alpha*(D*((exp(-2*aa*((r(n)-
r0)/r(n))))-(exp(-aa*((r(n)-r0)/r(n)))))));

[c,d] = spec(A);
E = diag(d)/(alpha*h*h);

plot2d(r,c(:,1))
```

The energy eigen values of the vibrating hydrogen molecule in different energy states are given below.

Radial Range $(\boldsymbol{a} < r < b)$	**Energy level**	**Energy eigen value (eV)**
$-5 < r < 5$	Ground state = E(1)	-0.1869
$0 < r < 10$	Ground state = E(1)	-0.1545
	E(2)	-0.1429
	E(3)	-0.1388
	E(4)	-0.1330

24) The Lagrangian for damping motion of a simple pendulum shown is given by,

$$L = e^{\alpha t}\left(\frac{1}{2}ml^2\dot{\theta}^2 - mgl(1-\cos\theta)\right)$$

Therefore the Lagrange's equation of motion becomes,

$$e^{\alpha t}\left(\ddot{\theta} + \alpha\dot{\theta} + \frac{g}{l}\sin\theta\right) = 0$$

This implies,

$$\ddot{\theta} = -\alpha\dot{\theta} - \frac{g}{l}\sin\theta$$

The *SciLab* program is written below. The position-time and the phase space graphs are shown in *Figures 4.52* and *4.53* respectively.

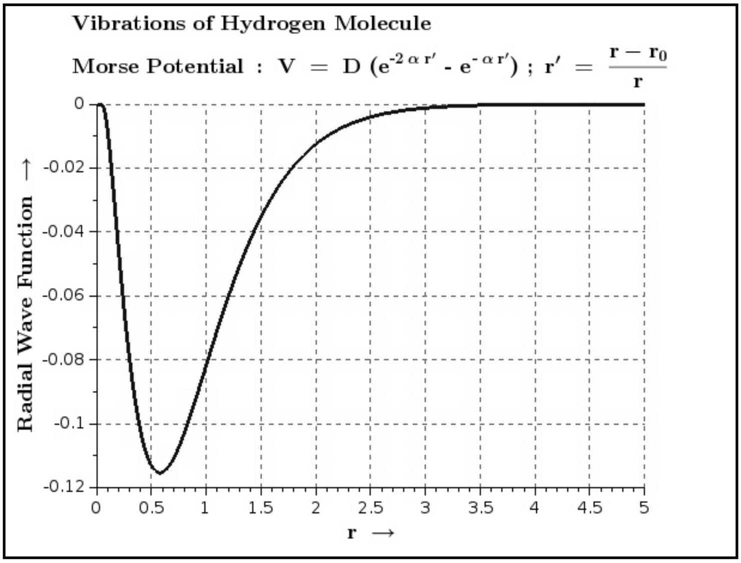

***Figure 4.51*:** Graph for *Exercise 23*

```
function ydot = f(t,y)
ydot(1) = y(2);
ydot(2) = -(g/l)*sin(y(1)) - 0.2*y(2);
endfunction

l = 1;
g = 9.82;
theta_0 = 0.5;
theta_dot_0 = 1.0;
t = 0:0.01:40;
y = ode([theta_0 ; theta_dot_0],0,t,f);

plot2d(t,y(1,:))
plot2d(y(1,:),y(2,:))
```

25) The Lagrangian for spring pendulum is given by,

$$L = \frac{1}{2}m\left[\dot{r}^2 + (l+r)^2\dot{\theta}^2\right] - \frac{1}{2}kr^2 - mg(l+r)(1-\cos\theta) + mgr$$

Therefore the Lagrange's equations of motion becomes,

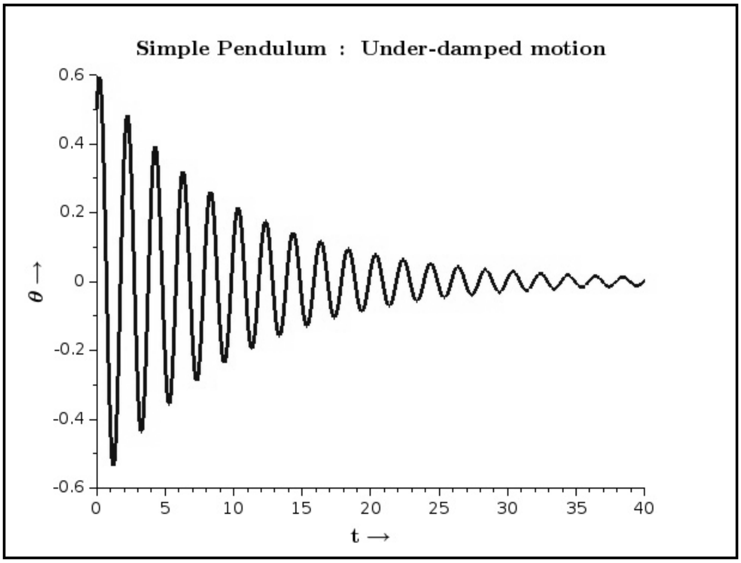

Figure 4.52**:** Solution for *Exercise 24*

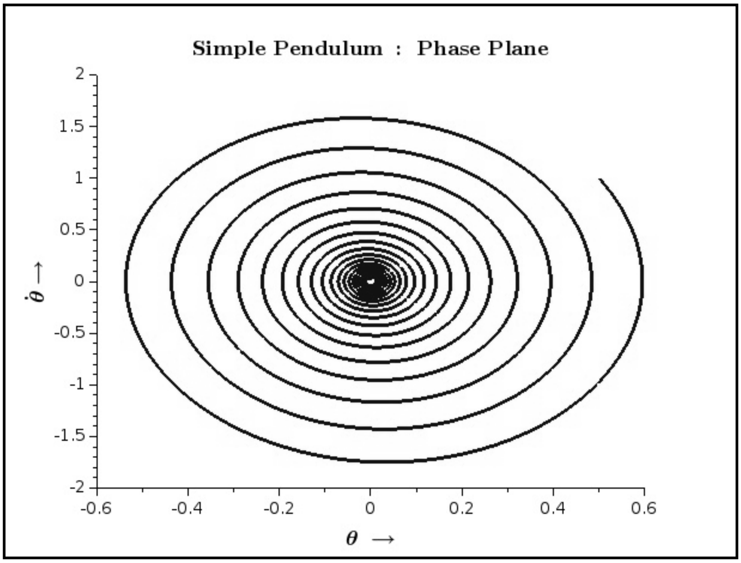

Figure 4.53**:** Solution for *Exercise 24*

$$\ddot{r} = (l + r)\dot{\theta}^2 + g\cos\theta - \frac{k}{m}r$$
$$\ddot{\theta}^2 = -\frac{2}{l+r}\dot{r}\dot{\theta} - \frac{g}{l+r}\sin\theta$$

The *SciLab* program is written below. The spring phase-plane and the pendulum phase-plane graphs are shown in *Figures 4.54* and *4.55*, respectively.

```
function ydot = f(t,y)
ydot(1) = y(2);
ydot(2) = (l+y(1))*(y(4).^2) + g*cos(y(3)) - (k/m)*y(1);
ydot(3) = y(4);
ydot(4) = -g*sin(y(3))/(l+y(1)) - 2*y(2)*y(4)/(l+y(1));
endfunction
l = 3;
g = 9.82;
k = 5;
m = 3;
r_0 = 4;
rdot_0 = 0;
theta_0 = 0.2;
theta_dot_0 = 0;
t = 0:0.1:60;
y = ode([r_0 ; rdot_0 ; theta_0 ; theta_dot_0],0,t,f);
plot2d(y(1,:),y(2,:))
plot2d(y(3,:),y(4,:))
```

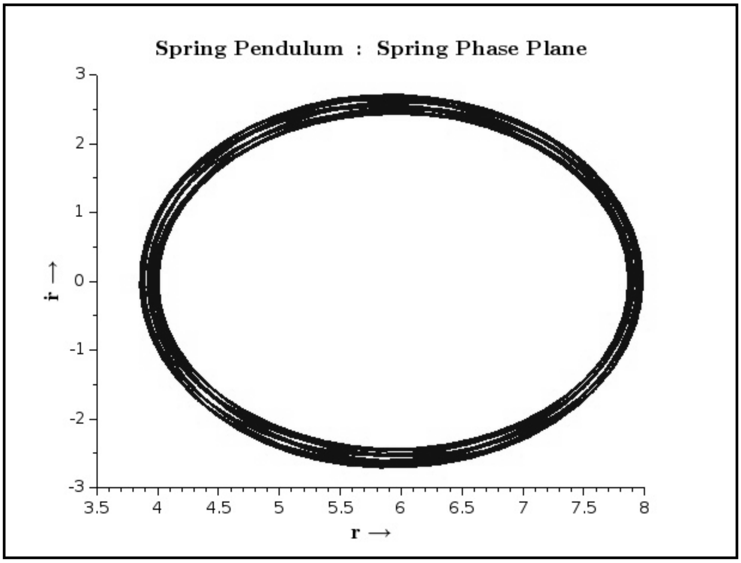

***Figure 4.54*:** Solution for *Exercise 25*

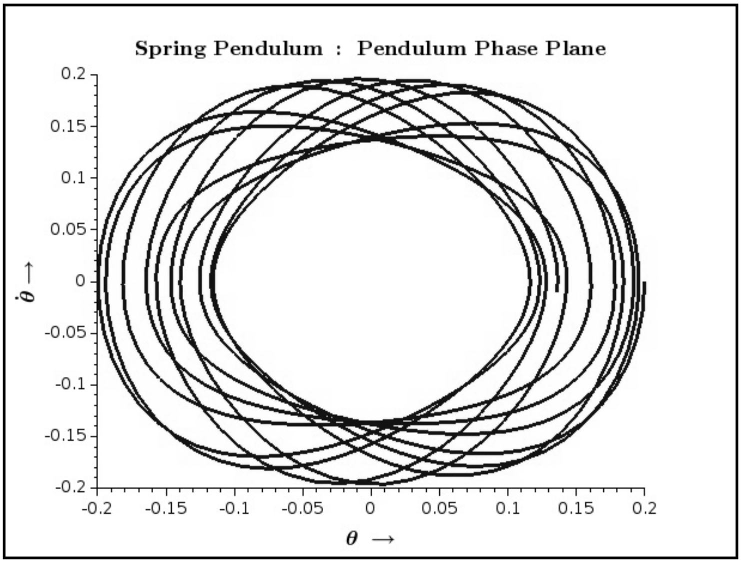

***Figure 4.55*:** Solution for *Exercise 25*

26) The position coordinates of the two masses are,

- $x_1 = l_1 \sin\theta_1$
- $x_2 = l_1 \sin\theta_1 + l_2 \sin\theta_2$
- $y_1 = -l_1 \cos\theta_1$
- $y_2 = -l_1 \cos\theta_1 - l_2 \cos\theta_2$

- The kinetic energy of the system is equal to,
$$K = \frac{1}{2}m_1\left(\dot{x_1}^2 + \dot{y_1}^2\right) + \frac{1}{2}m_2\left(\dot{x_2}^2 + \dot{y_2}^2\right)$$
- This implies,
$$K = \frac{1}{2}m_1\dot{\theta_1}^2 {l_1}^2 + \frac{1}{2}m_2\left[\dot{\theta_1}^2 {l_1}^2 + \dot{\theta_2}^2 {l_2}^2 + 2l_1 l_2 \dot{\theta}_1 \dot{\theta}_2 \cos(\theta_1 - \theta_2)\right]$$
- The potential energy of the system is equal to,
$$V = -(m_1 + m_2) g l_1 \cos\theta_1 - m_2 l_2 g \cos\theta_2$$
- The Lagrangian of the system is given by,
$$L = K - V$$
- Therefore the Lagrange's equation of motion will be equal to,

$$\ddot{\theta_1} = \frac{\begin{array}{c} -m_2 l_1 \dot{\theta_1}^2 \sin(\theta_1 - \theta_2)\cos(\theta_1 - \theta_1) + g m_2 \sin\theta_2 \cos(\theta_1 - \theta_2) \\ -m_2 l_2 \dot{\theta_2}^2 \sin(\theta_1 - \theta_2) - (m_1 + m_2) g \sin\theta_1 \end{array}}{(m_1 + m_2) l_1 - m_2 l_1 \cos^2(\theta_1 - \theta_2)}$$

$$\ddot{\theta_2} = \frac{\begin{array}{c} m_2 l_2 \dot{\theta_2}^2 \sin(\theta_1 - \theta_2)\cos(\theta_1 - \theta_1) + g(m_1 + m_2)\sin\theta_1 \cos(\theta_1 - \theta_2) \\ +(m_1 + m_2) l_1 \dot{\theta_1}^2 \sin(\theta_1 - \theta_2) - (m_1 + m_2) g \sin\theta_2 \end{array}}{(m_1 + m_2) l_2 - m_2 l_2 \cos^2(\theta_1 - \theta_2)}$$

The *SciLab* program is written below. The graph is shown in *Figure 4.56*.

```
function ydot = f(t,y)
ydot(1) = y(2);

num_1 = -m_2*l_1*(y(2)^2)*sin(y(1)-y(3))*cos(y(1)-y(3))
+ g*m_2*sin(y(3))*cos(y(1)-y(3)) -
m_2*l_2*(y(4)^2)*sin(y(1)-y(3)) - (m_1+m_2)*g*sin(y(1));
den_1 = l_1*(m_1+m_2) - m_2*l_1*(cos(y(1)-y(3))^2);
ydot(2) = num_1/den_1;

ydot(3) = y(4);

num_2 = m_2*l_2*(y(4)^2)*sin(y(1)-y(3))*cos(y(1)-y(3)) +
g*sin(y(1))*cos(y(1)-y(3))*(m_1+m_2) +
l_1*(y(2)^2)*sin(y(1)-y(3))*(m_1+m_2) -
(m_1+m_2)*g*sin(y(3));
den_2 = l_2*(m_1+m_2) - m_2*l_2*(cos(y(1)-y(3))^2);
ydot(4) = num_2/den_2;
endfunction

g = 9.82;
l_1 = 1;
l_2 = 2;
m_1 = 2;
m_2 = 1;
theta_1_0 = %pi;
theta_1_dot_0 = 0;
theta_2_0 = %pi/2;
theta_2_dot_0 = 0;

t = 0:0.01:50;
y = ode([theta_1_0 ; theta_1_dot_0 ; theta_2_0 ;
theta_2_dot_0],0,t,f);

x1 = l_1*sin(y(1,:));
y1 = -l_1*cos(y(1,:));
x2 = l_1*sin(y(1,:))+l_2*sin(y(3,:));
y2 = -l_1*cos(y(1,:)) - l_2*cos(y(3,:));

plot2d(x1,y1)

plot2d(x2,y2)
```

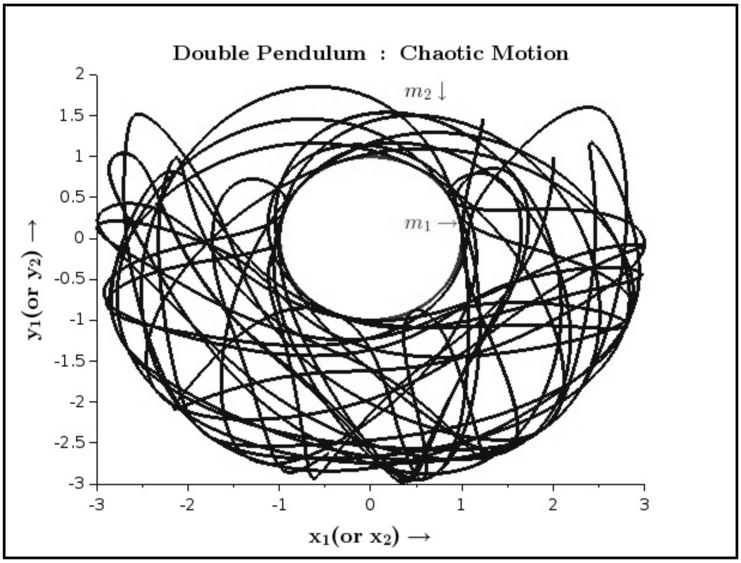

***Figure 4.56*:** Solution for *Exercise 26*

27) The coordinates of the pendulum attached to a rotating pivot are given by,

$$x = a\cos\omega t + b\sin\theta$$
$$y = a\sin\omega t - b\cos\theta$$

Here,

- The center of the pivot is taken as the center of the coordinate system.
- a is the radius of the pivot
- ω is the angular frequency of the rotating pivot
- b is the length of the pendulum
- θ is the angular displacement of the pendulum

The equation of motion of this system is given by,

$$\ddot{\theta} = \frac{a}{b}\omega^2\cos(\theta - \omega t) - \frac{g}{b}\sin\theta$$

The *SciLab* program is written below. The solution curve is shown is *Figure 4.57*.

```
function ydot = f(t,y)
ydot(1) = y(2);
ydot(2) = (a/b)*omega*omega*cos(y(1)-(omega*t)) -
(g/b)*sin(y(1))
endfunction

a = 0.2;
b = 1;
omega = 20;
g = 9.82;
theta_0 = %pi/6;
```

```
theta_dot_0 = 0;
t = 0:0.01:10;
y = ode([theta_0 ; theta_dot_0],0,t,f);
plot2d(t,y(1,:))
```

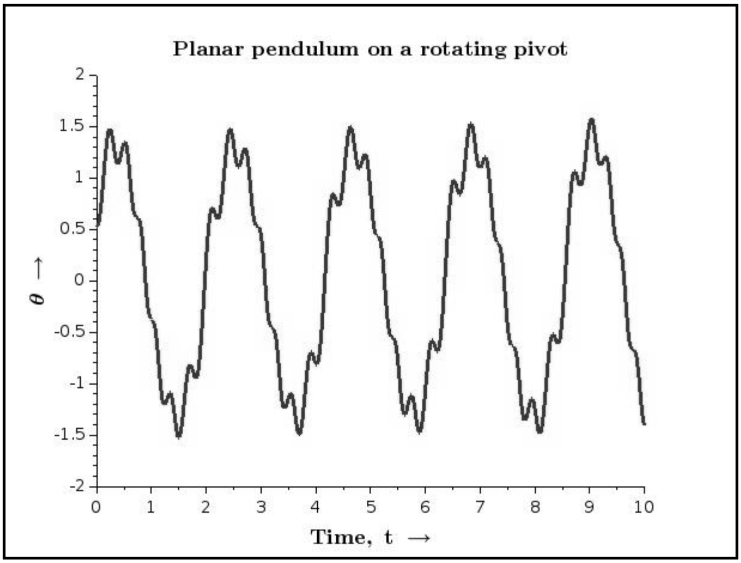

Figure 4.57: Solution for *Exercise 27*

28) Suppose the pendulum of mass m and length l is attached to a moving pivot having mass M. The pivot is located at a distance x from the reference point and is moving along the x-axis. If the angular displacement of the pendulum is θ then the coordinates of the bob of the pendulum are given by,

$$x_p = x + l\sin\theta$$
$$y_p = -l\cos\theta$$

The Lagrange's equation of motion is given by,

$$\ddot{\theta} = -\frac{\ddot{x}}{l}\cos\theta - \frac{g}{l}\sin\theta$$

The *SciLab* program is written below. For this program it is assumed that the pivot is moving such that $x = a\cos\omega t$. This implies that $\ddot{x} = -a\omega^2\cos\omega t$. The solution curve is shown in *Figure 4.58*.

```
function ydot = f(t,y)
ydot(1) = y(2);
ydot(2) = -(g/l)*sin(y(1)) +
(a*omega*omega*cos(omega*t)*cos(y(1)))/l;
endfunction

l = 1;
```

```
g = 9.82;
a = 0.2;
omega = 100;
theta_0 = 0.1;
theta_dot_0 = 0;

t = 0:0.01:3;
y = ode([theta_0 ; theta_dot_0],0,t,f);
plot2d(t,y(1,:))
```

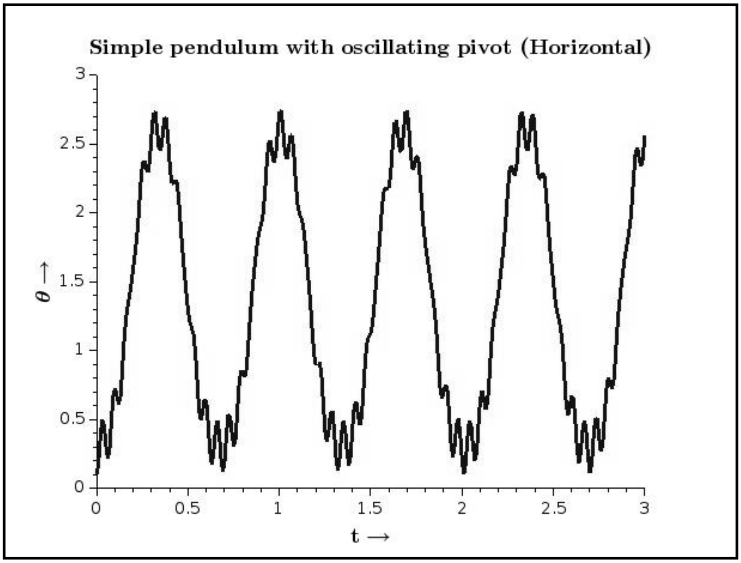

***Figure 4.58*:** Solution for *Exercise 28*

29) The solution to this question is similar to the previous exercise, with the exception that pivot is now moving along the y-axis instead of the x-axis.

30) The Lagrange's equation of motion for this system is,

$$\ddot{x_1} = \frac{-g\sin\theta\cos\theta}{\frac{m+M}{m} - \cos^2\theta}$$

$$\ddot{x_2} = \frac{g\sin\theta}{1 - \frac{m\cos^2\theta}{m+M}}$$

The graph shown in *Figure 4.59* is expected for this exercise.

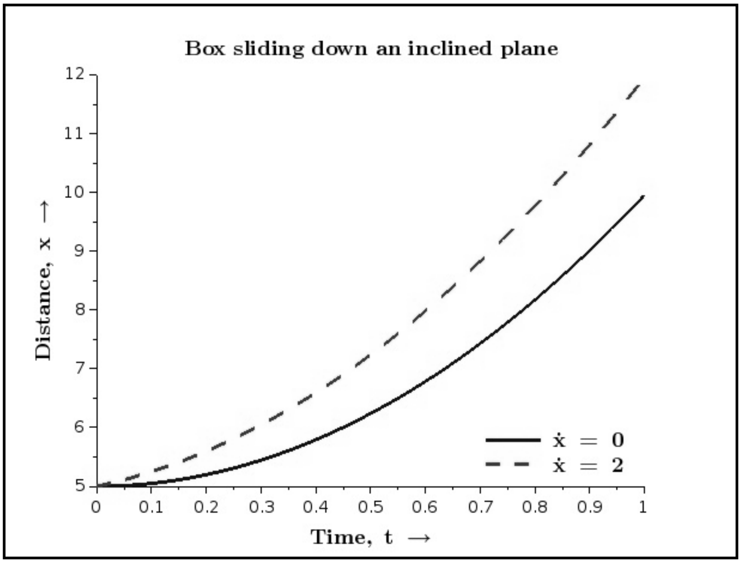

***Figure 4.59*:** Solution for *Exercise 30*

CHAPTER 5

1) The following *SciLab* program calculates the given integral by using different rules of integration.

```
exec('integrate.sci',-1);
function y = f(x)
y = sin(x);
endfunction

a = 0;                                             //Lower limit
b = %pi/2;                                         //Upper limit

intg(a,b,f)
integrate('f(x)','x',a,b)

//Even  number  of  intervals  have  been  taken  only  for
comparison of the trapezoidal method with the Simpson’s
1/3 Rule
h = (b-a)/2;
Y = trapezoidal(f,a,b,h)

Y = simpson_1_3(f,a,b,h)

h = (b-a)/3;          //Step size for Simpson’s 3/8 Rule
Y = simpson_3_8(f,a,b,h)
```

The values of the integral from different methods and different step sizes have been tabulated in *Table 5.4*.

Table 5.4*: Result for *Exercise 1

Method	Step size	Value of the integral
In-built Function – `intg`		1
In-built Function – `Integrate`		1
Trapezoidal Rule	$\frac{b-a}{2}$	0.9480594
	$\frac{b-a}{10}$	0.9979430

Method	Step size	Value of the integral
Simpson’s 1/3 Rule	$\frac{b-a}{2}$	1.0022799
	$\frac{b-a}{10}$	1.0000034
Simpson’s 3/8 Rule	$\frac{b-a}{3}$	1.0010049
	$\frac{b-a}{30}$	1.0000001

2) Suppose the definite integral to be evaluated is,

$$\int_{0}^{4} x^3\, dx$$

The *SciLab* program is written below and the significance of step size is shown in *Figures 5.12* and *5.13*.

```
exec('integrate.sci',-1);

a = 0;                                              //Lower limit
b = 4;                                              //Upper limit
h = 2;                                      //Number of intervals
step = (b-a)/h;                                       //Step size

for i = 1:h           //Loop for plotting the trapezoids
    if pmodulo(i,2) == 0 then
        j = 4;
    elseif pmodulo(i,2) == 1 then
        j = 7;
    end
    x1 = a+(step*(i-1));
    x2 = a+(step*i);
    y1 = x1^3;
    y2 = x2^3;
    xpts = [x1, x2, x2, x1];
    ypts = [y1, y2, 0, 0];
    scf(0);
    plot2d(x1,y1);
    xfpoly(xpts,ypts,j);
end

x = a:0.01:b;                          //x-range for plotting
plot2d(x,x.^3)                            //Plot the function
```

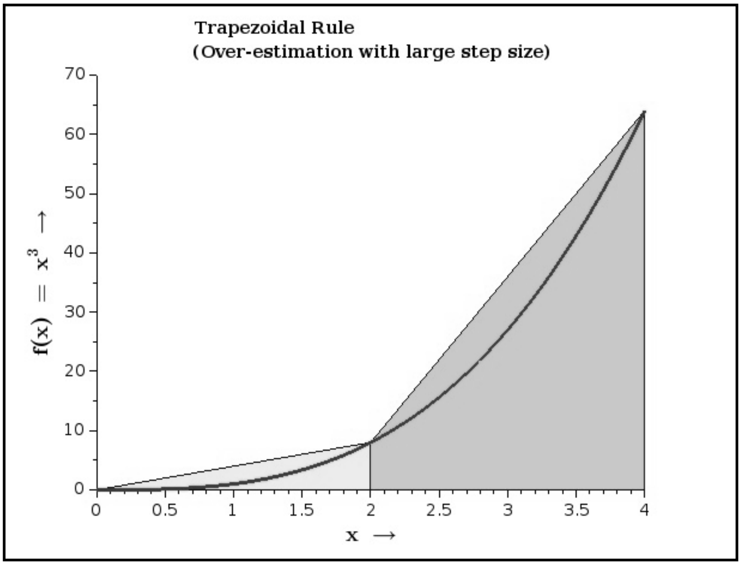

***Figure 5.12*:** Graph for solution of *Exercise 2*

The analytical solution of the given integral is equal to 64. But trapezoidal method gives a value of 80 when the entire interval is divided into 2 sub-intervals as shown in *Figure 5.12*. It clearly shows that a large step size in this case results into over-estimation of the value of integral. A more accurate result is obtained if the number of sub-intervals is increased to 8 (shown in *Figure 5.13*).

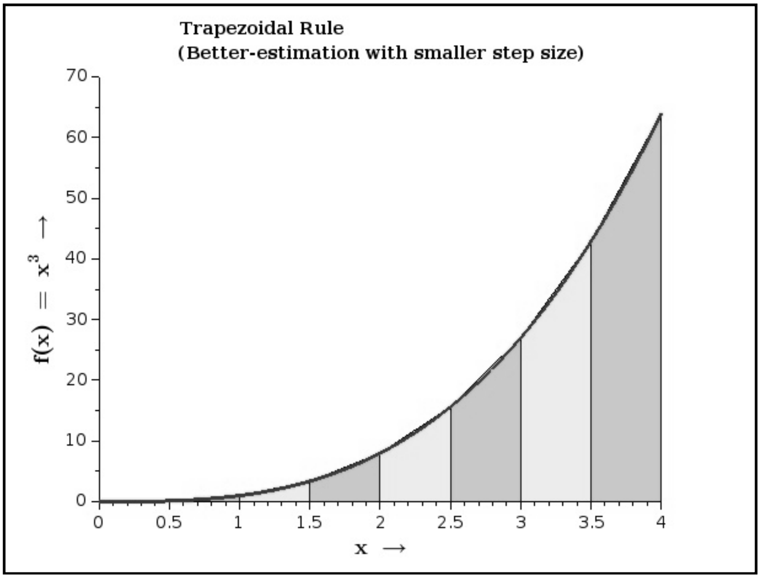

***Figure 5.13*:** Graph for solution of *Exercise 2*

3) According to the Debye’s model, the molar specific heat is given by,

$$C_v = 9Nk\left(\frac{T}{T_D}\right)^3 \int_0^{T_D/T} \frac{x^4 e^x}{(e^x - 1)^2}\, dx$$

The following *SciLab* program plots the molar specific heat of the metal on the y-axis and temperature on the x-axis. The graph is shown in *Figure 5.14*. In this program,

- The temperature is varied from 0.5 K to about 3 times the higher Debye temperature, i.e. from 0.5 K to 900 K.
- At high temperatures, the Debye’s formula approaches the Dulong-Petit law, according to which the molar specific heat is equal to $3Nk = 24.94\,JK^{-1}$.

```
function [Cv] = DB(T)
m = integrate('(y**4)*exp(y)/((exp(y)-1)^2)','y',0,TD/T)
Cv = 9*m*N*k*(T/TD)^3;
endfunction

k = 1.381e-23;             //Boltzmann constant (in J/K)
N = 6.022e23;                        //Avogadro’s number

n = input("Enter the number of elements for the graph :
")

for i = 1:n;
    element = input("Enter the name of the element :
","string");
    TD = input("Enter the Debye temperature (in Kelvin)
: ");
    x = [0.5 : 0.1 : 900.0];
    fplot2d(x,DB);
    A(i) = string(element);
end

legend(A);
```

The input parameters are written below,

```
Enter the number of elements for the graph : 2
Enter the name of the element : Copper
Enter the Debye temperature (in Kelvin) : 340
Enter the name of the element : Sodium
Enter the Debye temperature (in Kelvin) : 157
```

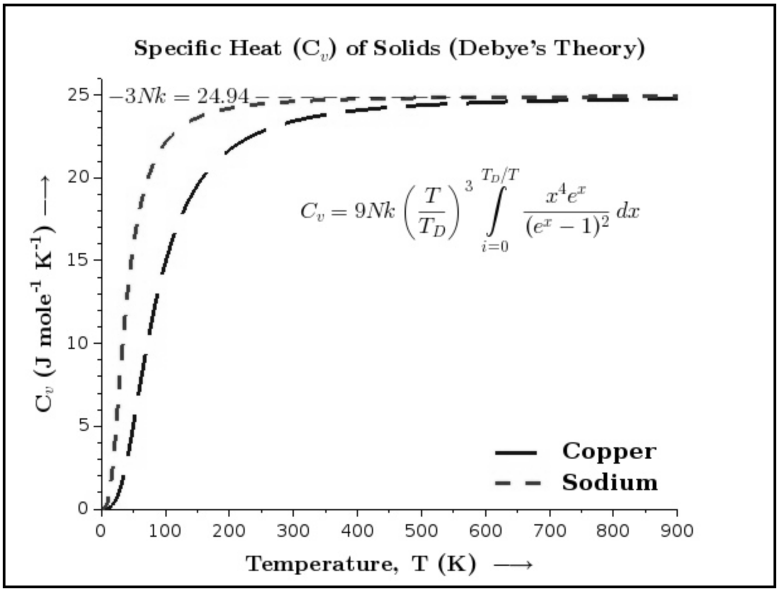

Figure 5.14: Graph for *Exercise 3*

4) According to the Debye's model, the molar specific heat is given by,

$$C_v = 9Nk\left(\frac{T}{T_D}\right)^3 \int_0^{T_D/T} \frac{x^4e^x}{(e^x-1)^2}\,dx$$

The following *SciLab* program plots the molar specific heat of the metal on the y-axis and $\left(\frac{T}{T_D}\right)$ on the x-axis. The graph is shown in *Figure 5.15*. In this program,

- The temperature is varied $0.01T_D$ to $3T_D$. This implies that the values on the x-axis range from 0.01 to 3.0.
- This graph shows that if the x-axis is in terms of $\frac{T}{T_D}$, then the curves for copper and sodium overlap.

```
function [Cv] = DB(alpha)
m = integrate('(y**4)*exp(y)/((exp(y)-
1)^2)','y',0,1.0/alpha)
Cv = 9*m*N*k*(alpha)^3;
endfunction

k = 1.381e-23;              //Boltzmann constant (in J/K)
N = 6.022e23;                          //Avogadro's number
n = input("Enter the number of elements for the graph :
")

for i = 1:n
```

```
    element = input("Enter the name of the element :
","string");
    TD = input("Enter the debye temperature (in Kelvin)
: ");
    x = [0.01 : 0.01 : 3.0];
    fplot2d(x,DB);
    A(i) = string(element);
end

legend(A);
```

The input parameters are written below,

```
Enter the number of elements for the graph : 2
Enter the name of the element : Copper
Enter the debye temperature (in Kelvin) : 340
Enter the name of the element : Sodium
Enter the debye temperature (in Kelvin) : 157
```

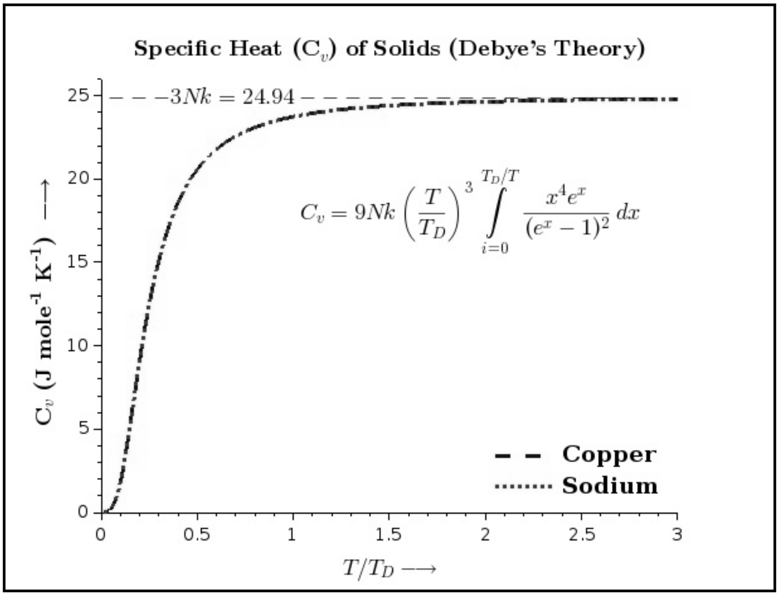

Figure 5.15*: Graph for *Exercise 4

7) The *SciLab* program is written below.
The diffraction patterns for different widths of the slit are shown in *Figures 5.16 (a – d).*

```
function y = f1(x)                     //Fresnel's Integral
y = cos(%pi*x*x/2);
endfunction
```

```
function y = f2(x)                  //Fresnel's Integral
y = sin(%pi*x*x/2);
endfunction

slit = 1;                                               //Δv
i = 1;
for v = -slit:0.01:2*slit;
    v1(i) = v;
    x_up(i) = integrate('f1(x)','x',0,v);
    y_up(i) = integrate('f2(x)','x',0,v);
    x_down(i) = integrate('f1(x)','x',0,slit-v);
    y_down(i) = integrate('f2(x)','x',0,slit-v);
    intensity(i) = (x_up(i)+x_down(i))^2 +
(y_up(i)+y_down(i))^2;
i = i+1;
end
plot2d(v1-(slit/2),intensity);
```

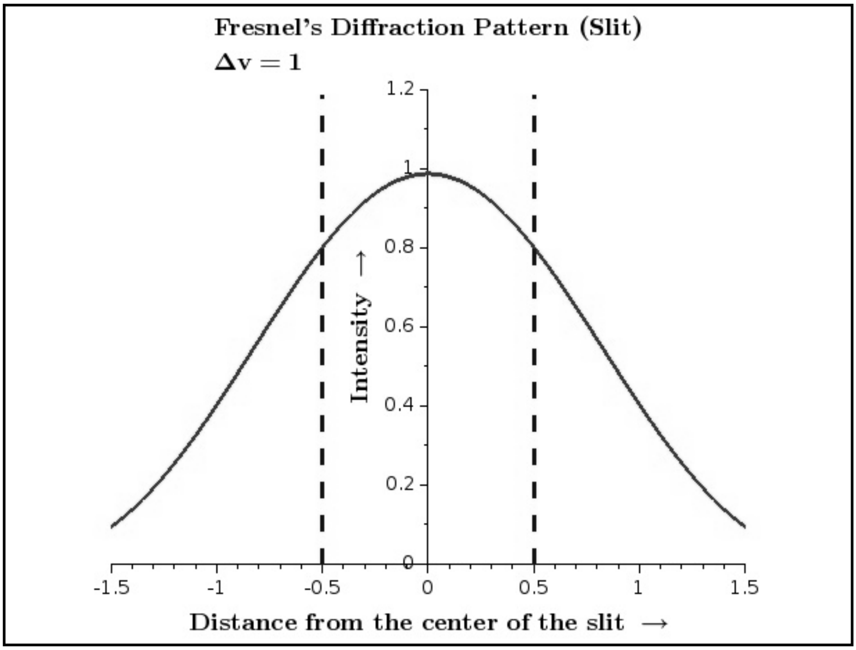

***Figure 5.16 (a)*:** Fresnel's Diffraction pattern due to slit of width 1

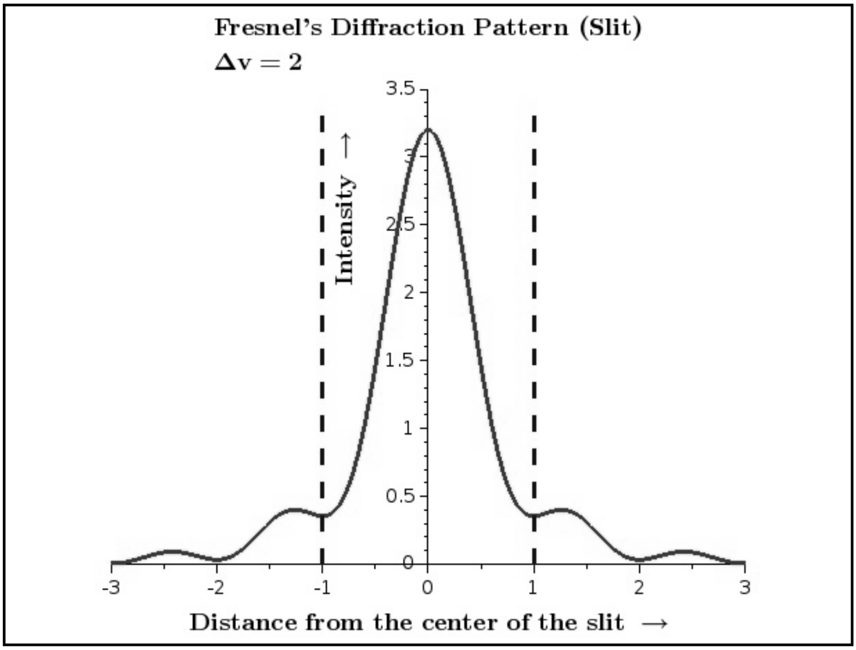

***Figure 5.16 (b)*:** Fresnel's Diffraction pattern due to slit of width 2

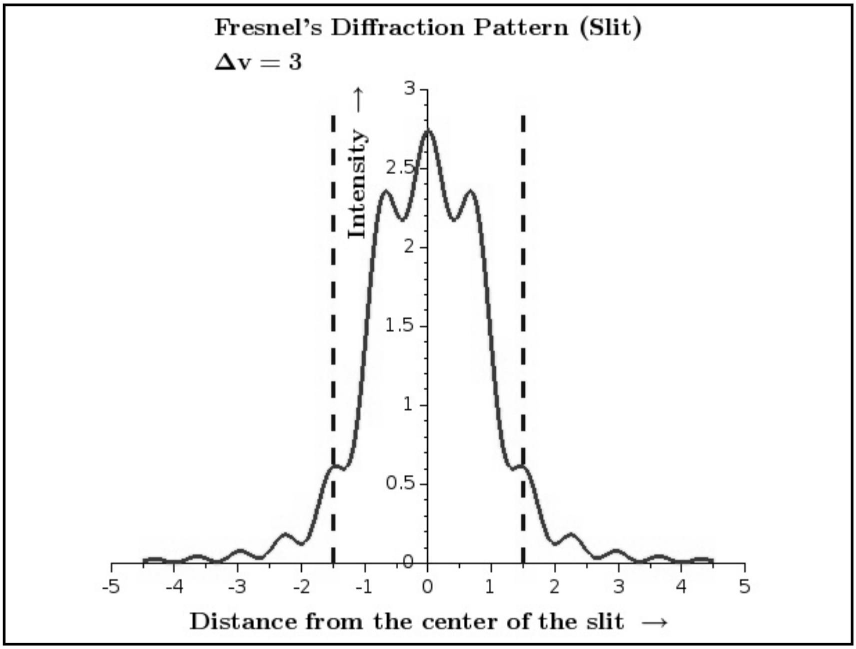

***Figure 5.16 (c)*:** Fresnel's Diffraction pattern due to slit of width 3

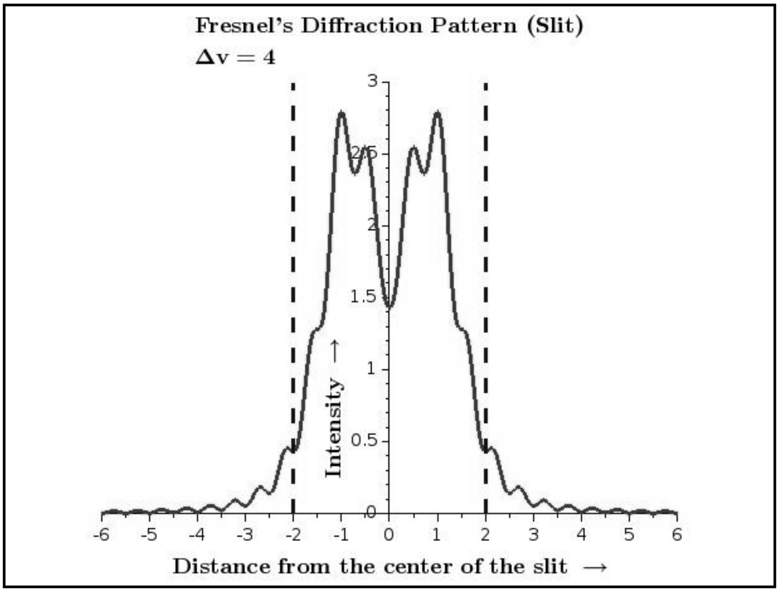

***Figure 5.16 (d)*:** Fresnel's Diffraction pattern due to slit of width 4

8) The *SciLab* program is written below. The diffraction patterns for different widths of the wire are shown in *Figures 5.17(a – d).*

```
function y = f1(x)                    //Fresnel's Integral
y = cos(%pi*x*x/2);
endfunction

function y = f2(x)                    //Fresnel's Integral
y = sin(%pi*x*x/2);
endfunction

slit = 1;                                                //Δv
i = 1;
for v = -4.4:0.01:slit+4.4;
    v1(i) = v;
    x_up(i) = integrate('f1(x)','x',0,v);
    y_up(i) = integrate('f2(x)','x',0,v);
    x_down(i) = integrate('f1(x)','x',0,slit-v);
    y_down(i) = integrate('f2(x)','x',0,slit-v);
    intensity(i) = 0.5*((1-x_up(i)-x_down(i))^2 + (1-
y_up(i)-y_down(i)))^2;
    i = i+1;
end

plot2d(v1-(slit/2),intensity);
```

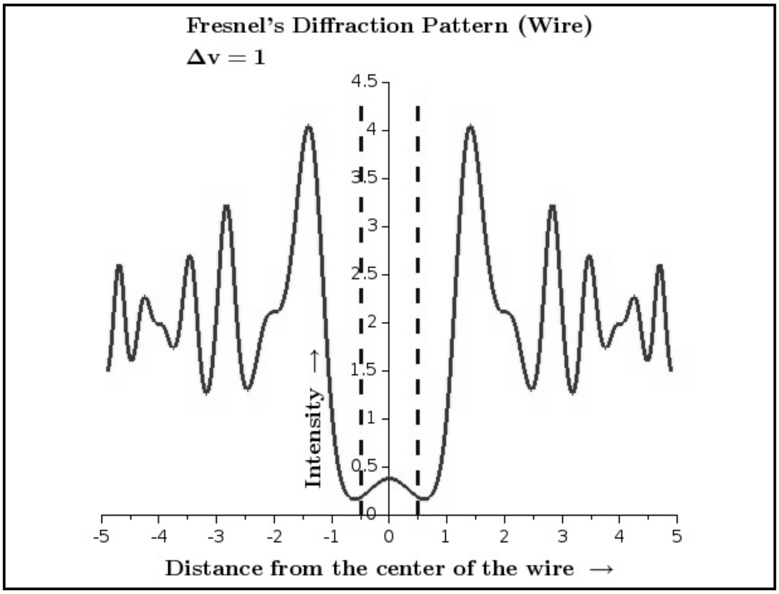

***Figure 5.17 (a)*:** Fresnel's Diffraction due to a wire of width 1

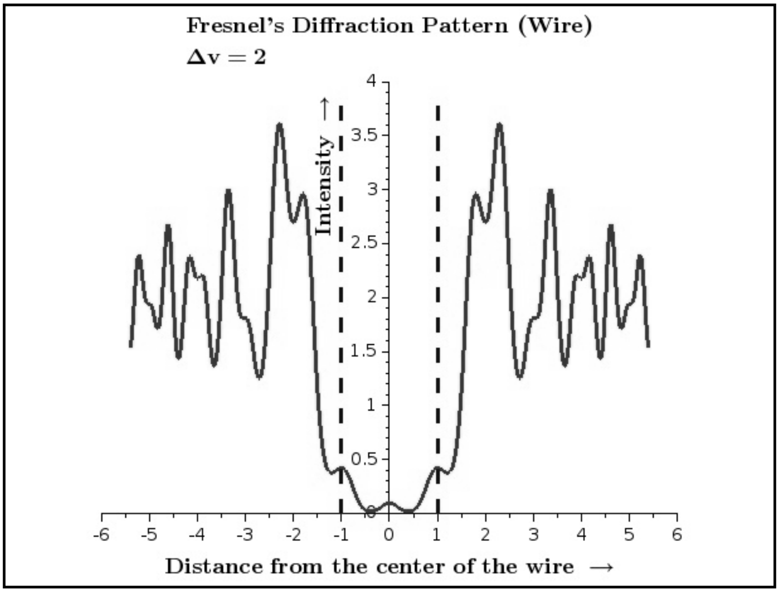

***Figure 5.17 (b)*:** Fresnel's Diffraction due to a wire of width 2

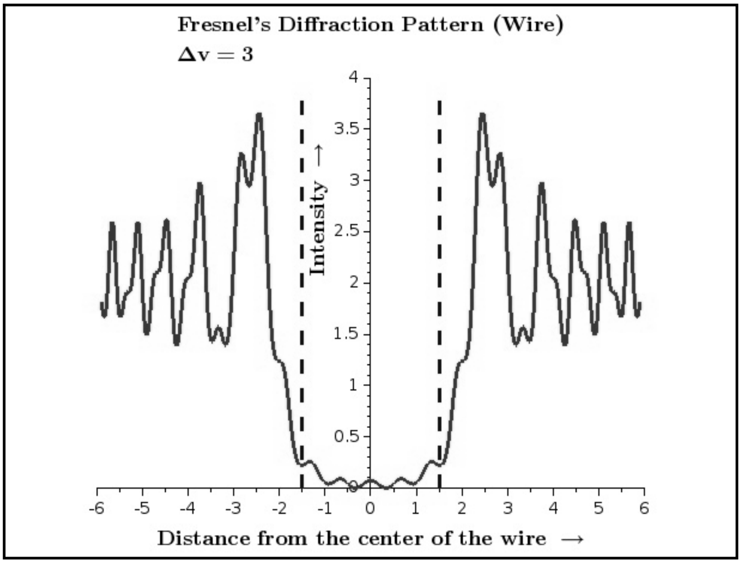

***Figure 5.17 (c)*:** Fresnel's Diffraction due to a wire of width 3

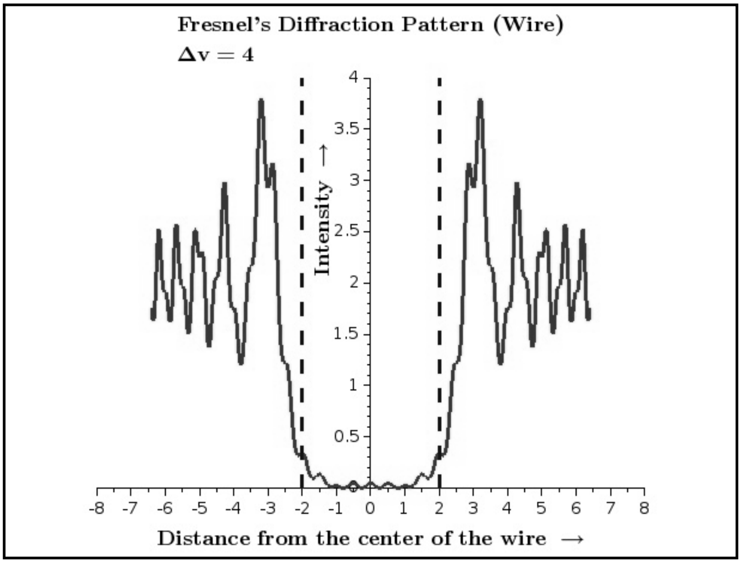

***Figure 5.17 (d)*:** Fresnel's Diffraction due to a wire of width 4

11) The *SciLab* program for differentiation of a triangular wave is written below. The graph is shown in *Figure 5.18*. Note that periodic functions will be discussed in detail in the Chapter on Fourier analysis.

```
period = 2*%pi;              //Period of triangular wave
```

```
function a = periodic(f,T,x)          //Periodic function
if (x >= 0) & (x <= T) then
    a = f(x);
elseif x < 0 then
    x_new = x + T;
    a = periodic(f,T,x_new);
elseif x > T then
    x_new = x - T;
    a = periodic(f,T,x_new);
end
endfunction
function y = f(x)                       //Triangular wave
if x < period*0.5 then
    y = x;
else
    y = period-x
end
endfunction
x = [0:0.01:2*period];           //Range of 'x' variable
for i = 1:length(x)
    y(i) = periodic(f,period,x(i));
end
dy = diff(y)/0.01;                      //Differentiation
x1 = x(1:$-1);
plot2d(x1,dy)                //Plot the first derivative
plot2d(x,y')                  //Plot the triangular wave
```

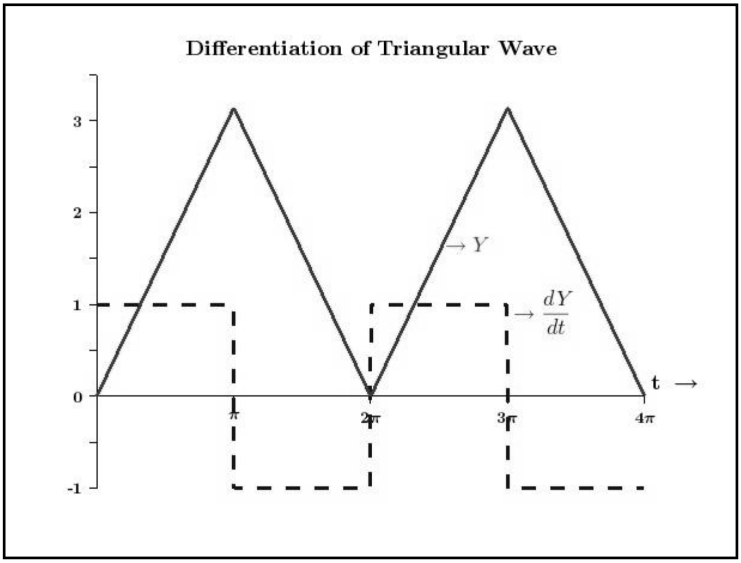

Figure 5.18: Solution of *Exercise 11*

12) The *SciLab* program is written below. The displacement, velocity and acceleration profiles are shown in *Figure 5.19*.

```
t = [0:0.01:1];

displacement = 5*t.^3 + t.^2 + 1.0;
velocity = diff(displacement)/0.01;
acceleration = diff(velocity)/0.01;

t1 = t(1:$-1);
t2 = t1(1:$-1);

plot2d(t,displacement)
plot2d(t1,velocity)
plot2d(t2,acceleration)
```

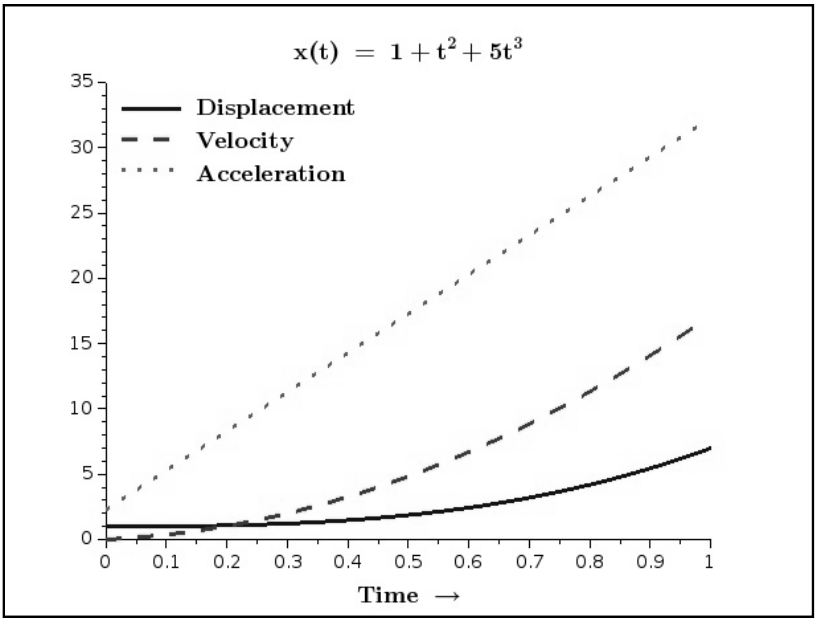

***Figure 5.19*:** Solution for *Exercise 12*

13) The output graph is shown in *Figure 5.20*.

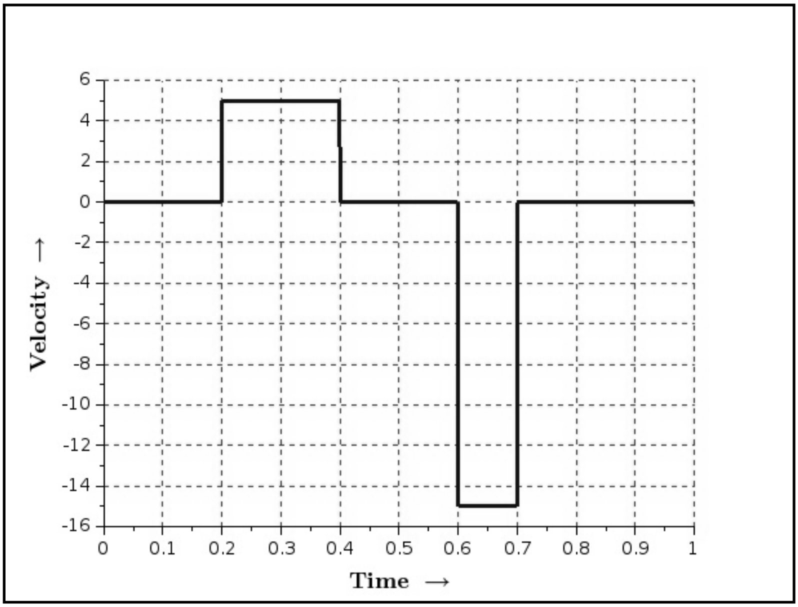

***Figure 5.20*:** Solution for *Exercise 13*

CHAPTER 6

1) The *SciLab* program is written below and the graph is shown in *Figure 6.10*.

```
i = -1;
j = 0;
for n = 1:3
    x = 0.001:0.5:10;
    i = i+1;
    j = j+1;
    y = besselj(i,x);
    plot2d(x,y)
end
```

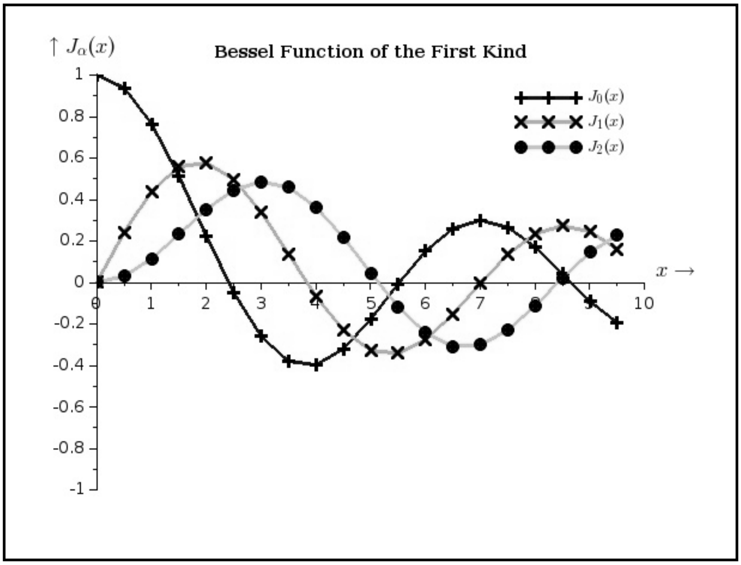

***Figure 6.10*:** Solution of *Exercise 1*

2) The *SciLab* program is written below.

```
function y = J_alpha(alpha, x)
series = 0;
for n = 0:10;
    series_alpha = ( ((-1)^n)/(factorial(n)*factorial(n
+ alpha)) )*(x/2)^(2*n);
    series = series + series_alpha;
end
y = ((x/2)^alpha).*series;
endfunction
```

The result is given below.

```
    J_alpha(0,1) = 0.7651977
    J_alpha(0,2) = 0.2238908
```

3) The interpolated value of Bessel functions can be determined by using the following Newton's forward difference formula for interpolation.

$$y_n(x) = y_0 + p\Delta y_0 + \frac{p(p-1)}{2!}\Delta^2 y_0 + .. + \frac{p(p-1)..(p-n+1)}{n!}\Delta^n y_0$$

Here,

- A range of data points $(x_0, y_0), (x_1, y_1), (x_2, y_2) \ldots (x_n, y_n)$ is given such that the values of x are equidistant.
- Therefore the step size is $h = x_1 - x_0 = x_2 - x_1$ and so on.
- It is required to determine an interpolated value at a random value of x lying in the given range. This value is labeled as $y_n(x)$.
- Therefore, $p = \frac{x - x_0}{h}$
- Δy_0 corresponds to the first forward difference.
- $\Delta^2 y_0$ corresponds to difference of the first forward differences. It is therefore called as the second forward difference.
- $\Delta^n y_0$ corresponds to n^{th} forward difference

The *SciLab* program for determining the value of $J_0(9.95)$ is written below.

```
x = [5,6,7,8,9,10,11,12];
for i = 1:length(x);
    y(1,i) = besselj(0,x(i));
end
for i = 2:length(x);
    for j = 1:(length(x)-(i-1));
        y(i,j) = y(i-1,j+1)-y(i-1,j);
    end
end
x_given = 9.95;
h = x(2) - x(1);
p = (x_given - x(1))/h;
sum = y(1,1);
m = 1;
for i = 2:length(x);
    m = m*(p-i+2)
    sum = sum + ((m*y(i,1))/factorial(i-1));
end
mprintf("\n Interpolated value of Bessel function at x =
"+string(x_given)+" is equal to %f",sum);
mprintf("\n Actual value of Bessel function at x =
"+string(x_given)+" is equal to %f",besselj(0,x_given));
```

The result is as follows.

```
Interpolated value of Bessel function at x = 9.95 is
equal to -0.243471
Actual value of Bessel function at x = 9.95 is equal to
-0.243450
```

For $J_0(25.2)$, change the x-range to equally spaced values around 25.
The result will be as follows.

```
Interpolated value of Bessel function at x = 25.2 is
equal to 0.119655
Actual value of Bessel function at x = 25.2 is equal to
0.119157
```

4) The *SciLab* program is written below.

```
p = zeros(5,1);
for order = 1:5;
   p(order) = legendre_poly_gamma(order,'x');
end;
```

The roots of these polynomials can be obtained by using the in-built *SciLab* function, ‘`roots`’. The result is tabulated below (*Table 6.3*).

Table 6.3*: Result for *Exercise 4

roots(p(1))	0.
roots(p(2))	- 0.5774 0.5774
roots(p(3))	- 0.7746 0.7746 0
roots(p(4))	- 0.8611 0.8611 - 0.3400 0.3400
roots(p(5))	- 0.9062 - 0.5385 0.9062 0.5385 0

CHAPTER 7

1) The *SciLab* program is written below. The graph is shown in *Figure 7.21*.

```
exec('fourier.sci',-1);
period = 4;                                //Periodicity
function y = f(x)                    //Periodic function
if x < 0 then
    y = x;
else
    y = -x
end
endfunction
x = [-2*period : 0.01 : 2*period];
for i = 1:length(x)
    y(i) = periodic1(f,0.5*period,x(i));
end
plot2d(x,y')
```

2) The *SciLab* program is written below.

```
exec('fourier.sci',-1)
period = 10;                               //Periodicity
function y = f(x)
if (x > -0.25*period) & (x < 0.25*period) then
    y = 0.5;
else
    y = 0
end
endfunction
x = [-1.5*period:0.01:1.5*period];
for i = 1:length(x)
    y(i) = periodic1(f,0.5*period,x(i));
end
plot2d(x,y)
```

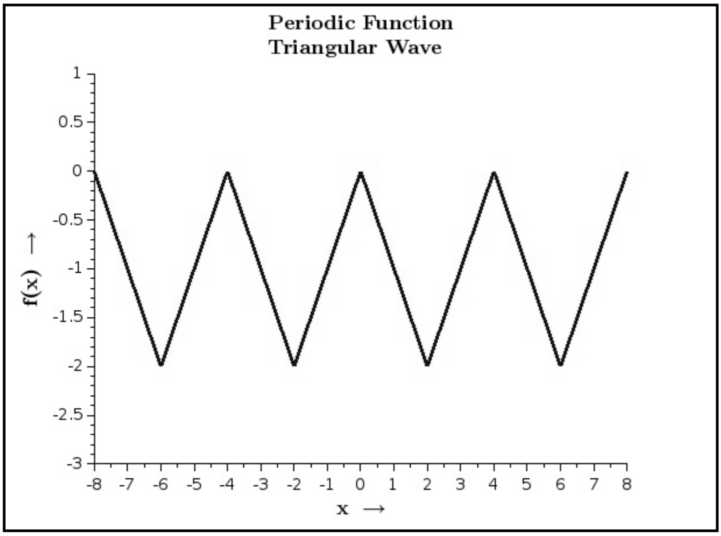

***Figure 7.21*:** Solution for *Exercise 1*

3) The *SciLab* program is written below. The graph is shown in *Figure 7.22*.

```
exec('fourier.sci',-1);
period = 4;                                        //Periodicity
function y = f(x)                            //Periodic Function
y = x;
endfunction
x = [-2*period : 0.01 : 2*period];
for i = 1:length(x)
    y(i)= periodic1(f,0.5*period,x(i));
end
plot2d(x,y')
```

5) The *SciLab* program is written below. The Fourier series expansion is shown in *Figure 7.23*. The value of Fourier series coefficients a_0 and b_n will be equal to zero. The values of other coefficients determined from the program have been given in *Table 7.3*.

```
exec('fourier.sci',-1);
w = %pi;                                //Base angular frequency
period = (2*%pi)/w;                                //Base period
n = 5;                                     //Number of harmonics
x = 0:0.01:2*period;                        //Range for plotting
function y = f(x);                         //Define the function
      y = cos(w*x) + cos(2*w*x);
endfunction
plot2d(x,f(x));                              //Plot the function
```

```
[a0,a,b] = fourier2(period,n,f);
```

Table 7.3*: Fourier coefficients for *Exercise 5

n	a_n
1	1
2	1
3	0
4	0
5	0

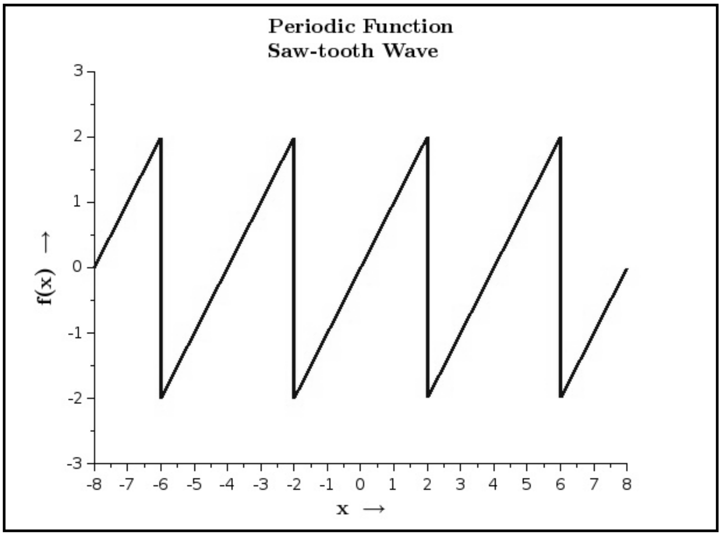

Figure 7.22: Solution for *Exercise 3*

7) The *SciLab* program is written below and the harmonics are shown in *Figure 7.24.*

```
exec('fourier.sci',-1);
function y = f(x)
y = sign(sin(2*%pi*(1/(2*%pi))*x));
endfunction
period = 2*%pi;
x = 0:0.01:2*period;
plot2d(x,f(x)
n = 1;
[a0,a,b] = fourier2(period,n,f);
n = 3;
[a0,a,b] = fourier2(period,n,f);
```

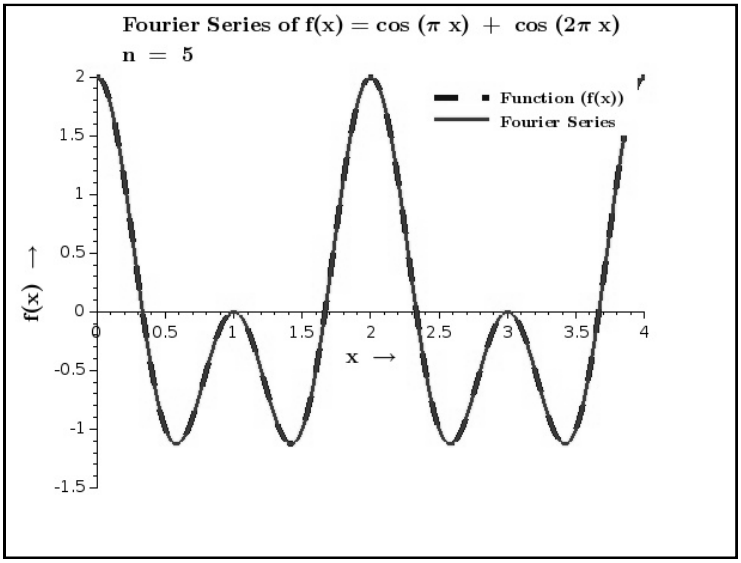

Figure 7.23: Solution for *Exercise 5*

The square wave is an odd function. Therefore the Fourier series coefficients, a_0 and a_n will be equal to zero. The other coefficients are given in *Table 7.4*. For a square wave having amplitude (A), these coefficients are in accordance with the theoretically expected values given by,

$$b_n = \begin{cases} \dfrac{4A}{n\pi} & \text{if n is odd} \\ 0 & \text{if n is even} \end{cases}$$

From *Figure 7.24*, it is clear that,

- Addition of higher order harmonics gives a better approximation of the original function.
- Only odd harmonics are present in this approximation. This is due to the symmetric nature of the function.

Table 7.4*: Fourier coefficients for *Exercise 7

n	b_n
1	1.2732

n	b_n
2	0
3	0.4244

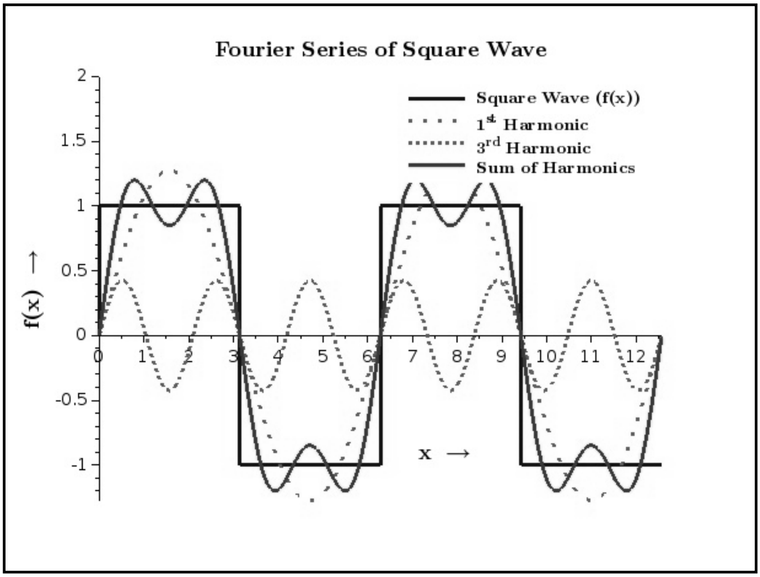

***Figure 7.24*:** Solution for *Exercise 7*

9) The *SciLab* program is written below. The Fourier series expansion is shown in *Figure 7.25.*

```
period = 2;
function a = periodic(f,T,x)
if (x >= 0) & (x <= T) then
    a = f(x);
elseif x < 0 then
    x_new = x + T;
    a = periodic(f,T,x_new);
elseif x > T then
    x_new = x - T;
    a = periodic(f,T,x_new);
end
endfunction

function y = f(x)
if x < period*0.2 then
      y = 5*x;
elseif x < period*0.8 then
```

```
      y = 2
else
      y = 2-5*(x-1.6)
end
endfunction

x = [0:0.01:2*period];

for i = 1:length(x)
    y(i) = periodic(f,period,x(i));
end
plot2d(x,y')

n = 7;
[a0,a,b] = fourier2(period,n,f);
```

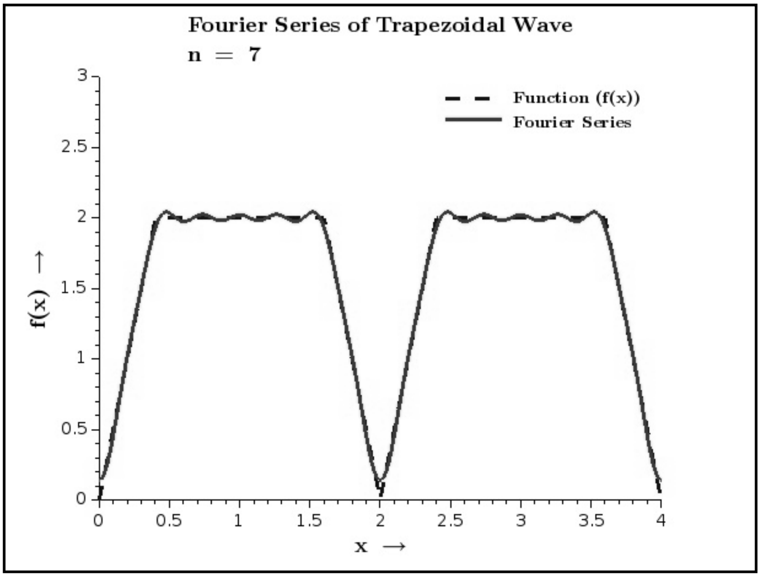

Figure 7.25**:** Solution for *Exercise 9*

12) The *SciLab* program is written below. The Fourier Transform is shown in *Figure 7.26*.

```
sample_rate = 100;
i = 1;
for t = -1:1/sample_rate:1;
    time(i) = t;
    func(i) = sin(20*%pi*t)/exp(2*%pi.*t.*t);
    i = i+1;
end
```

```
subplot(211)
plot2d(time,func)
X = fft(func);
N = length(time);
f = 0:40;
subplot(212)
plot2d(f,abs(X(1:length(f))))
```

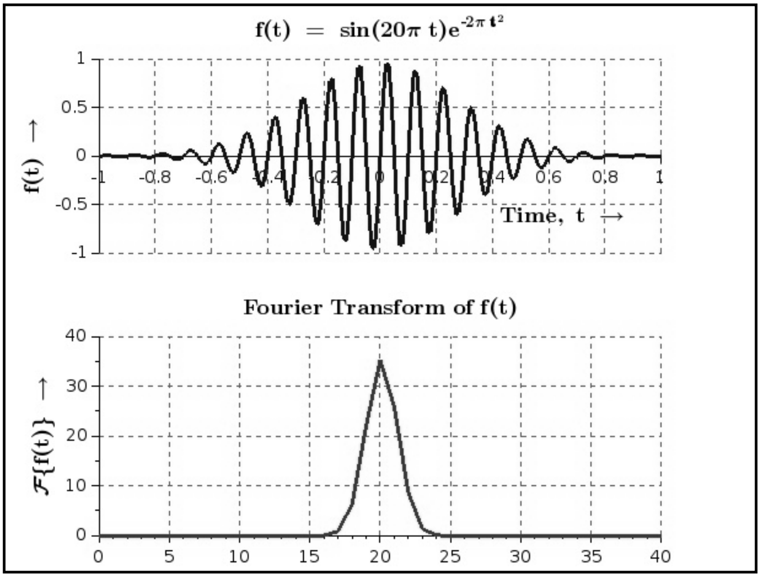

***Figure 7.26*:** Solution for *Exercise 12*

CHAPTER 8

1) The *SciLab* programs are written below.
Part (a)

```
exec('numerical_techniques.sci',-1)
A = [1 2 11 ; 1 -1 5];
```

The Gauss-Seidel method will not converge.

Part (b)

```
exec('numerical_techniques.sci',-1)
A = [3 1 1 8 ; 1 5 -3 2 ; 2 -1 4 12];
```

The solution will be equal to 1, 2 and 3.

Part (c)

```
exec('numerical_techniques.sci',-1)
A = [2 4 6 14 ; 3 -2 1 -3 ; 4 2 -1 -4];
```

The Gauss-Seidel method will not converge.

2) The *SciLab* programs are written below.
Part (a)

```
exec('numerical_techniques.sci',-1)
A = [4 -2 ; 2 1];
B = [6 ; 45];
gauss_elimination(A,B)
```

The answer will come out to be, $x = 12,\ \ y = 21$

Part (b)

```
exec('numerical_techniques.sci',-1)
A = [6 -3 ; 1 5];
B = [-21 ; 46];
gauss_elimination(A,B)
```

The answer will come out to be, $x = 1,\ \ y = 9$

Part (c)

```
exec('numerical_techniques.sci',-1)
A = [1 1 -1 ; 4 -1 5 ; 3 2 -2];
B = [6 ; 8 ; 14];
```

```
gauss_elimination(A,B)
```

The answer will come out to be, $x = 2,\ \ y = 5,\ z = 1$

Part (d)

```
exec('numerical_techniques.sci',-1)
A = [0 2 3 ; 1 0 -2 ; 4 3 0];
B = [13 ; -5 ; 10];
gauss_elimination_pivot(A,B)
```

The answer will come out to be, $x = 1,\ \ y = 2,\ z = 3$

Part (e)

```
exec('numerical_techniques.sci',-1)
A = [1 3 0 ; 1 0 3 ; 0 2 1];
B = [9 ; -3 ; 2];
gauss_elimination_pivot(A,B)
```

The answer will come out to be, $x = 3,\ \ y = 2,\ z = -2$

Part (f)

```
exec('numerical_techniques.sci',-1)
A = [1 1 0 ; 0 1 1 ; 1 0 1];
B = [3 ; 5 ; 4];
gauss_elimination_pivot(A,B)
```

The answer will come out to be, $x = 1,\ \ y = 2, z = 3$

3) The *SciLab* program is written below.

```
exec('numerical_techniques.sci',-1)
A = [3 4 0 ; 6 8 2 ; 1 1 3];
gauss_inverse(A)
```

The answer will come out to be,

$$A^{-1} = \begin{bmatrix} 11 & -6 & 4 \\ -8 & 4.5 & -3 \\ -1 & 0.5 & 0 \end{bmatrix}$$

4) The *SciLab* programs are written below.
Part (a)

```
exec('numerical_techniques.sci',-1)

function func = f(x)
```

```
func = 5*x + log(x) - 100
endfunction

Newton_Raphson(15,1e-4,1e-4)
```

The root will be equal to 19.406875

Part (b)

```
exec('numerical_techniques.sci',-1)

function f = f(x)
f = %e^x - x.^3
endfunction

Newton_Raphson(3,1e-4,1e-5)
Newton_Raphson(1.5,1e-4,1e-5)
```

Roots are equal to 4.536404 and 1.857184

Part (c)

```
exec('numerical_techniques.sci',-1)

function f = f(x)
f = 2.^x - x.^2
endfunction

Newton_Raphson(-1,1e-4,1e-5)
Newton_Raphson(1,1e-4,1e-5)
Newton_Raphson(3,1e-4,1e-5)
```

Roots are equal to -0.766663, 2.000000 and 4.000000

Part (d)

```
exec('numerical_techniques.sci',-1)
function f = f(x)
f = x.*cos(x) + sin(x)
endfunction

Newton_Raphson(-0.5,1e-4,1e-5)
Newton_Raphson(1,1e-4,1e-5)
Newton_Raphson(3,1e-4,1e-5)
Newton_Raphson(7,1e-4,1e-5)
Newton_Raphson(10,1e-4,1e-5)
Newton_Raphson(13,1e-4,1e-5)
```

The first few positive roots are equal to 0.000000, 2.028748, 4.913147, 7.978699, 11.085510 and 14.207458.

5) The *SciLab* program is written below. The graph is shown in *Figure 8.5*.

```
exec('numerical_techniques.sci',-1)

function func = f(x)                      //Define the function
func = x.*tan(x) - 1
endfunction

x = 0:0.1:8;                             //x-range for plotting
plot2d(x,f(x))                              //Plot the function

Bisection_Method(0.5,1.5,100,1d-4);
Bisection_Method(3,4,100,1d-4);
Bisection_Method(6,7,100,1d-4);

Secant_Method(0.5,1.5,1e-4)
Secant_Method(3,4,1e-4)
Secant_Method(6,7,1e-4)

Regula_Falsi_Method(0.5,1.5,1e-4)
Regula_Falsi_Method(3,4,1e-4)
Regula_Falsi_Method(6,7,1e-4)

Newton_Raphson(0.5,1e-4,1e-4)
Newton_Raphson(3,1e-4,1e-4)
Newton_Raphson(6,1e-4,1e-4)
```

The roots determined from different methods are given in *Table 8.2*.

Table 8.2*: Result for *Exercise 5

Method	Approximate Root
Bisection Method	0.860291
	3.425598
	6.437317
Secant Method	0.860334
	3.425618
	6.437298

Method	Approximate Root
Regula Falsi Method	0.860305
	3.425602
	6.437292
Newton Raphson Method	0.860334
	3.425621
	6.437304

6) The *SciLab* program is written below. The graphs are shown in *Figure 8.6*.

```
exec('numerical_techniques.sci',-1)

function func = f(x)
func = 2*sin(x) - x
endfunction

Newton_Raphson(1.5,1e-4,1e-4)

x = 0:0.1:3;

plot2d(x,x)
plot2d(x,2*sin(x))
plot2d(x,f(x))
plot2d([1.895494,1.895494],[0,1.895494],13)
```

The root will be equal to 1.895494

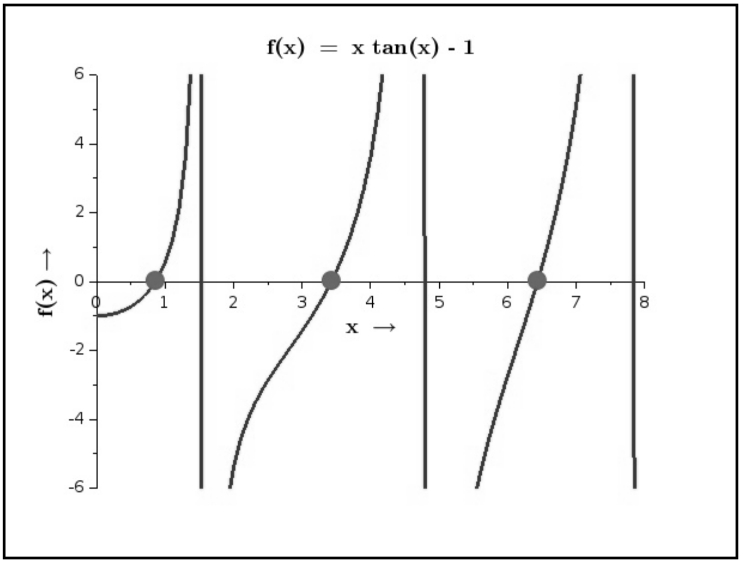

***Figure 8.5*:** Solution for *Exercise 5*

7) The *SciLab* programs are written below.
Part (a)

```
exec('numerical_techniques.sci',-1)

function f = f(x)
f = x.^2 - 3
endfunction
Bisection_Method(1,2,100,1d-6);
Newton_Raphson(1,1e-4,1e-4)
Secant_Method(1,2,1e-2)
Regula_Falsi_Method(0,1,1e-6)
```

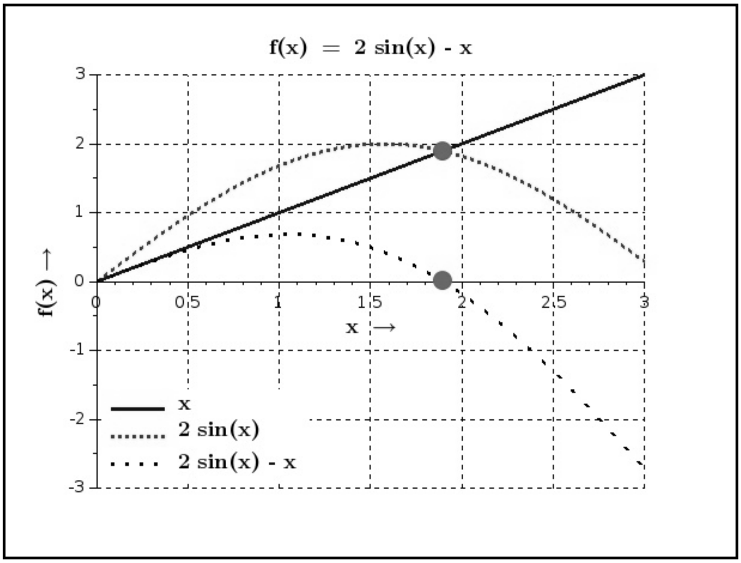

Figure 8.6: Solution for *Exercise 6*

Part (b)

```
exec('numerical_techniques.sci',-1)

function f = f(x)
f = 4.*x.^2 - 3
endfunction

Bisection_Method(0,1,100,1d-6);
Newton_Raphson(1,1e-4,1e-5)
Secant_Method(0,1,2,1e-4)
Regula_Falsi_Method(0,1,1e-6)
```

Part (c)

```
exec('numerical_techniques.sci',-1)

function f = f(x)
f = x.^3 - 2
endfunction

Bisection_Method(1,2,100,1d-5);
Newton_Raphson(1,1e-4,1e-5)
Secant_Method(1,2,1e-4)
Regula_Falsi_Method(0,1,1e-6)
```

Part (d)

```
exec('numerical_techniques.sci',-1)

function f = f(x)
f = x.^4 - 0.8
endfunction

Bisection_Method(0,1,100,1d-6);
Newton_Raphson(1,1e-4,1e-5)
Secant_Method(1,2,1e-4)
Regula_Falsi_Method(0,1,1e-6)
```

Part (e)

```
exec('numerical_techniques.sci',-1)

function f = f(x)
f = x.^5 - 5
endfunction

Bisection_Method(1,2,100,1d-6);
Newton_Raphson(1,1e-4,1e-5)
Secant_Method(1,2,1e-4)
Regula_Falsi_Method(0,1,1e-6)
```

8) The *SciLab* programs are written below.
Part (a)

```
exec('numerical_techniques.sci',-1)

function f = f(x)
f = 2.*x.^3 - x.^2 - 5*x + 1
endfunction

Bisection_Method(-2,0,100,1d-4);
Bisection_Method(0,1,100,1d-4);
Bisection_Method(1,2,100,1d-4);

Newton_Raphson(-2,1e-4,1e-5)
Newton_Raphson(0,1e-4,1e-5)
Newton_Raphson(1.5,1e-4,1e-5)
```

The three roots are equal to, -1.454773, 0.195374 and 1.759408

Part (b)

```
exec('numerical_techniques.sci',-1)

function f = f(x)
    f = 5.*x.^4  - 13*x.^3 - 1
endfunction

Bisection_Method(-2,0,100,1d-4);
Bisection_Method(0,1,100,1d-4);

Newton_Raphson(-0.5,1e-4,1e-5)
Newton_Raphson(1,1e-4,1e-5)
```

The roots are equal to -0.405212 and 2.611176.

9) The *SciLab* programs are written below.

Part (a)

```
exec('numerical_techniques.sci',-1)

function f = f(x)
f = (3.*x.*x - 1)/2.0
endfunction

Bisection_Method(0,1,100,1d-4);
```

The positive root is equal to 0.577332

Part (b)

```
exec('numerical_techniques.sci',-1)

function f = f(x)
f = (5.*x.^3 - 3.*x)/2.0
endfunction

Bisection_Method(-0.2,0.2,100,1d-4);
Bisection_Method(0.1,1,100,1d-6);
```

The positive roots are equal to 0.000000 and 0.774597

Part (c)

```
exec('numerical_techniques.sci',-1)

function f = f(x)
f = (35.*x.^4 - 30.*x.^2 + 3)/8.0
endfunction
```

```
Bisection_Method(0,0.5,100,1d-6);
Bisection_Method(0.5,1,100,1d-6);
```

The positive roots are equal to 0.339980 and 0.861136

Part (d)

```
exec('numerical_techniques.sci',-1)

function f = f(x)
f = (63.*x.^5 - 70.*x.^3 + 15*x)/8.0
endfunction

Bisection_Method(0,0.6,100,1d-6);
Bisection_Method(0.2,0.6,100,1d-6);
Bisection_Method(0.6,1,100,1d-6);
```

The positive roots are equal to 0.000000, 0.538470 and 0.906181.

10) The velocity of the object is given by,

$$v(t) = 3t^2 - 81$$

Therefore, in order to find the time at which the velocity becomes equal to zero, we will have to determine the root of the following equation,

$$t^2 - 27 = 0$$

The *SciLab* program is written below.

```
exec('numerical_techniques.sci',-1)

function f = f(t)
f = t.^2  - 27
endfunction

Newton_Raphson(5,1e-4,1e-5)
```