

Scilab Textbook Companion for
Numerical Analysis
by I. Jacques And C. Judd¹

Created by
Pragya Chordia And Shubham Mittal
Int. MSc. App. Math. (Pursuing)
Mathematics
IIT Roorkee
College Teacher
Prof. Roshan Lal
Cross-Checked by
Santosh Kumar, IITB

August 10, 2013

¹Funded by a grant from the National Mission on Education through ICT, <http://spoken-tutorial.org/NMEICT-Intro>. This Textbook Companion and Scilab codes written in it can be downloaded from the "Textbook Companion Project" section at the website <http://scilab.in>

Book Description

Title: Numerical Analysis

Author: I. Jacques And C. Judd

Publisher: Chapman And Hall

Edition: 1

Year: 1987

ISBN: 0-412-27950-9

Scilab numbering policy used in this document and the relation to the above book.

Exa Example (Solved example)

Eqn Equation (Particular equation of the above book)

AP Appendix to Example(Scilab Code that is an Appednix to a particular Example of the above book)

For example, Exa 3.51 means solved example 3.51 of this book. Sec 2.3 means a scilab code whose theory is explained in Section 2.3 of the book.

Contents

List of Scilab Codes	4
1 Introduction	8
2 Linear Algebraic Equation	11
3 Non linear algebraic equations	23
4 Eigenvalues and eigenvectors	38
5 Methods of approximation theory	56
6 Numerical Differentiation and Integration	83
7 Ordinary Differential Equations Initial value problem	98
8 Ordinary Differential Equations boundary value problem	108

List of Scilab Codes

Exa 1.1	Illustrating big errors caused by small errors	8
Exa 1.4	Calculating Induced instability through deflation method	10
Exa 2.1	Illustrates the effect of the partial pivoting	11
Exa 2.2	Decomposition in LU form	13
Exa 2.3	LU factorization method for solving the system of equation	13
Exa 2.4	LU factorisation method for solving the system of equation	15
Exa 2.5	Choleski decomposition	16
Exa 2.6	Jacobi method	17
Exa 2.7	Gauss Seidel method	18
Exa 2.8	Successive over relaxation method	19
Exa 2.9	Gauss Seidel and SOR method	20
Exa 3.1	Bisection Method	23
Exa 3.2	False positioning method	25
Exa 3.3	fixed point iteration method	26
Exa 3.4	Type of convergence	28
Exa 3.5	Newton Method	29
Exa 3.6	Secant Method	32
Exa 3.7	System of Non Linear Equations	34
Exa 3.8	System of Non Linear Equations	36
Exa 4.1	Power Method of finding largest Eigen value	38
Exa 4.2	Power Method of finding largest Eigen value	39
Exa 4.3	Convergence of Inverse Iteration	39
Exa 4.4	Deflation	40
Exa 4.5	Threshold serial Jacobi Method	41
Exa 4.6	The Gerchgorin circle	43
Exa 4.7	Sturm sequence property	45

Exa 4.8	Gerschgorins first theorem	46
Exa 4.9	Givens Method	48
Exa 4.10	Householder Matrix	49
Exa 4.11	Householder methods	50
Exa 4.12	stable LR method	51
Exa 4.13	Orthogonal decomposition QR method	51
Exa 4.14	Reduction to upper Hessenberg form	52
Exa 4.15	Redduction to upper Hessenberg form and calculating eigen values	54
Exa 5.1	Lagranges Method of interpolation	56
Exa 5.2	Theoretical bound on error	57
Exa 5.3	Divided difference	58
Exa 5.4	Polynomial Interpolation Divided Differnce form	59
Exa 5.5	Construction of Forward Difference Table	60
Exa 5.6	Illustration of Newtons Forward Difference Formula	61
Exa 5.7	Illustration of Central Difference Formula	62
Exa 5.8	Hermite Interpolation	64
Exa 5.9	Hermite cubic Interpolation	65
Exa 5.10	Illustration cubic spline interpolation with equal difference	66
Exa 5.11	Illustration cubic spline interpolation with unequal difference	68
Exa 5.12	Alternating way of constructing cubic splines	70
Exa 5.13	Linear Least square aproximation method	71
Exa 5.14	Quadratic Least square aproximation method	74
Exa 5.15	Least square aproximation method with exponential functions	76
Exa 5.16	Least square approximation to continuous functions	77
Exa 5.17	Gram Schmidt process for finding orthogonal functions	78
Exa 5.18	Gram Schmidt process for cubic polynomial least squares approx	80
Exa 6.1	Numerical Differentiation	83
Exa 6.2	Numerical Differentiation	84
Exa 6.3	Numerical Integration	85
Exa 6.4	Numerical Integration	87
Exa 6.5	Trapezoidal Rule	88
Exa 6.6	Simpson Rule	89
Exa 6.7	Rombergs Interpolation	90

Exa 6.8	Rombergs Method	91
Exa 6.9	Simpsons Adaptive Quatrature	92
Exa 6.10	Simpsons Adaptive Quatrature	93
Exa 6.11	Gaussian Quadrature Rule	96
Exa 6.12	Gaussian Quadrature Rule	96
Exa 7.1	Eulers Method	98
Exa 7.2	Eulers trapezoidal predictor corrector pair	99
Exa 7.3	Mid point formula	100
Exa 7.4	Illustraion of Taylor Series for approximation	101
Exa 7.5	3 Step Adams Bashforth and 2 step Adam Moulton formula	102
Exa 7.10	Runge Kutta Methods	103
Exa 7.11	Eulers Methods	104
Exa 7.12	Eulers trapezoidal predictor corrector pair	105
Exa 7.13	4 Stage Runge Kutta method	106
Exa 8.1	The finite difference method	108

List of Figures

1.1	Illustrating big errors caused by small errors	9
3.1	Bisection Method	24
3.2	False positioning method	26
3.3	Type of convergence	30
3.4	Newton Method	32
3.5	Secant Method	34
4.1	The Gerchgorin circle	44
5.1	Linear Least square aproximation method	73
5.2	Quadratic Least square aproximation method	75
5.3	Least square approximation to continuous functions	79
6.1	Numerical Differentiation	84
6.2	Numerical Differentiation	86

Chapter 1

Introduction

Scilab code Exa 1.1 Illustrating big errors caused by small errors

```
1 //Illustrating that a small error in data provided
   can result in big errors.
2 //with original equations
3 //X+Y=2 & X+1.01Y=2.01
4 clear;
5 clc;
6 close();
7 A=[1 1;1 1.01];
8 B=[2 2.01]';
9 x=A\B;
10 disp(x, 'Solutions are :')
11 x=linspace(-0.5,1.5);
12 y1=2-x;
13 y2=(2.01-x)/1.01;
14 subplot(2,1,1);
15 plot(x,y1)
16 plot(x,y2, 'r')
17 xtitle('plot of correct equations','x axis','y axis'
   )
18 //with the equations having some error in data
19 //X+Y=2 & X+1.01Y=2.02
```

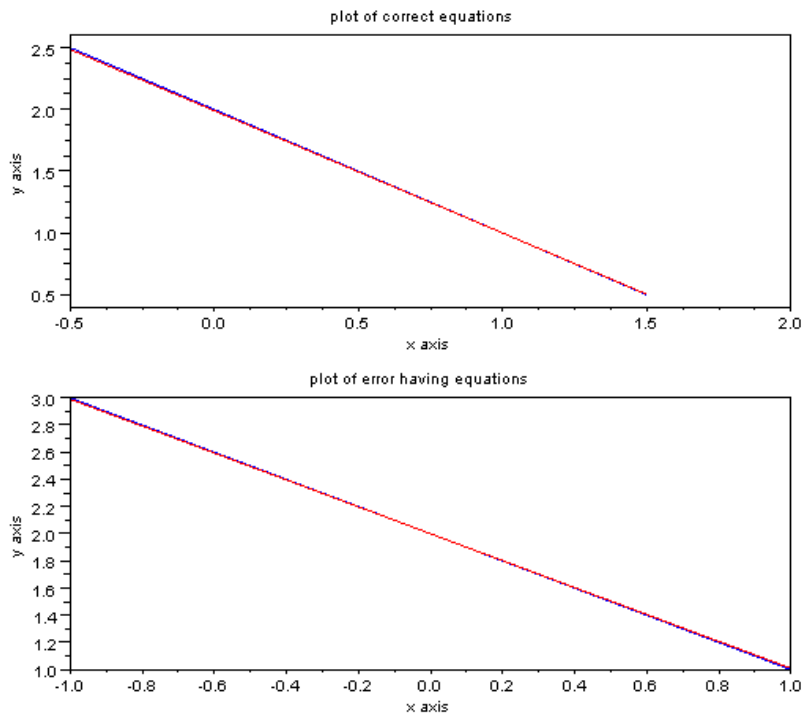


Figure 1.1: Illustrating big errors caused by small errors

```

20 A=[1 1;1 1.01];
21 B=[2 2.02]';
22 x=A\B;
23 disp(x,'Solutions are :')
24 subplot(2,1,2);
25 x=linspace(-1,1);
26 y1=2-x;
27 y2=(2.02-x)/1.01;
28 plot(x,y1)
29 plot(x,y2,'r')
30 xtitle('plot of error having equations','x axis','y
axis')

```

Scilab code Exa 1.4 Calculating Induced instability through deflation method

```
1 //illustrating the induced instability through the
   deflation method of polynomial factorisation.
2 clear;
3 clc;
4 close();
5 x=poly(0, 'x');
6 p3=x^3-13*x^2+32*x-20; //Given Polynomial
7 roots(p3)
8 //suppose that an estimate of its largest zero is
   taken as 10.1.Now divide p3 by (x-10.1)
9 p2=x^2-2.9*x+2.71; //the quotient
10 roots(p2)
11 disp('induced a large error in roots')
```

Chapter 2

Linear Algebraic Equation

Scilab code Exa 2.1 Illustrates the effect of the partial pivoting

```
1 //Illustrates the effect of the partial pivoting
   using 3 significant //figure arithmetic with
   rounding
2 //first done without pivoting and then with partial
   pivoting
3 clear;
4 close();
5 clc;
6 A
   =[0.610,1.23,1.72;1.02,2.15,-5.51;-4.34,11.2,-4.25];

7 B=[0.792;12.0;16.3];
8 C=[A,B];
9 format('v',10);
10 n=3;
11 for k=1:n-1
12     for i=k+1:n
13         c=C(i,k)/C(k,k);
14         for j=k:n+1
15             C(i,j)=C(i,j)-c*C(k,j);
16         end
```

```

17     end
18 end
19 x3=C(3,4)/C(3,3);
20 x2=(C(2,4)-C(2,3)*x3)/C(2,2);
21 x1=(C(1,4)-C(1,3)*x3-C(1,2)*x2)/C(1,1);
22 disp([x1,x2,x3], 'Answers without partial pivoting :
      ')

23
24
25 C=[A,B];
26 format('v',5);
27 n=3;
28 for k=1:n-1
29     m = max(abs(A(:,k)));
30     for l=k:n
31         if C(l,k)==m
32             temp = C(l,:);
33             C(l,:)= C(k,:);
34             C(k,:)=temp;
35             break;
36         end
37     end
38     for i=k+1:n
39         c=C(i,k)/C(k,k);
40         for j=k:n+1
41             C(i,j)=C(i,j)-c*C(k,j);
42         end
43     end
44 end
45 x3=C(3,4)/C(3,3);
46 x2=(C(2,4)-C(2,3)*x3)/C(2,2);
47 x1=(C(1,4)-C(1,3)*x3-C(1,2)*x2)/C(1,1);
48 disp([x1,x2,x3], 'Answers using partial pivoting : ')

```

Scilab code Exa 2.2 Decomposition in LU form

```
1 //Illustrates the decomposition of every matrix into
   product of lower //and upper triangular matrix
   if diagonal elements of any one is '1' //then L
   and U could uniquely be determined.
2 clear;
3 close();
4 clc;
5 format('v',5);
6 A = {4,-2,2;4,-3,-2;2,3,-1};
7 L(1,1)=1;L(2,2)=1;L(3,3)=1;
8 for i=1:3
9     for j=1:3
10        s=0;
11        if j>=i
12            for k=1:i-1
13                s=s+L(i,k)*U(k,j);
14            end
15            U(i,j)=A(i,j)-s;
16        else
17            for k=1:j-1
18                s=s+L(i,k)*U(k,j);
19            end
20            L(i,j)=(A(i,j)-s)/U(j,j);
21        end
22    end
23 end
24 disp(L, 'L =')
25 disp(U, 'U =')
```

Scilab code Exa 2.3 LU factorization method for solving the system of equation

```

1 //Applying LU factorization method for solving the
   system of equation
2
3 clear;
4 close();
5 clc;
6 format('v',5);
7 A = [4,-2,2;4,-3,-2;2,3,-1];
8 for l=1:3
9     L(l,l)=1;
10 end
11 for i=1:3
12     for j=1:3
13         s=0;
14         if j>=i
15             for k=1:i-1
16                 s=s+L(i,k)*U(k,j);
17             end
18             //disp(s,'sum :');
19             U(i,j)=A(i,j)-s;
20         else
21             //s=0;
22             for k=1:j-1
23                 s=s+L(i,k)*U(k,j);
24             end
25             L(i,j)=(A(i,j)-s)/U(j,j);
26         end
27     end
28 end
29 b=[6;-8;5];
30 c=L\b;
31 x=U\c;
32 disp(x,'Solution of equations :')

```

Scilab code Exa 2.4 LU factorisation method for solving the system of equation

```
1 //Application of LU factorisation method for solving
   the system of equation
2 //In this case A(1,1)=0 so to avoid the division by
   0 we will have to interchange the rows.
3
4 clear;
5 close();
6 clc;
7 format('v',5);
8 A = {2,2,-2,4;0,1,5,3;1,5,7,-10;-1,1,6,-5};
9 for l=1:4
10     L(1,1)=1;
11 end
12 for i=1:4
13     for j=1:4
14         s=0;
15         if j>=i
16             for k=1:i-1
17                 s=s+L(i,k)*U(k,j);
18             end
19             //disp(s,'sum :');
20             U(i,j)=A(i,j)-s;
21         else
22             //s=0;
23             for k=1:j-1
24                 s=s+L(i,k)*U(k,j);
25             end
26             L(i,j)=(A(i,j)-s)/U(j,j);
27         end
28     end
29 end
30 b=[4;-6;14;0];
31 c=L\b;
32 x=U\c;
33 disp(x,'Solution of equations :')
```

Scilab code Exa 2.5 Choleski decomposition

```
1 //Solving the problem using Choleski decomposition
2 //Decomposition of a matrix "A" to L and L'
3
4 clear;
5 close();
6 clc;
7 format('v',7)
8 A = [4,2,-2;2,10,2;-2,2,3];
9 n = 3;
10 for i = 1:n
11     for j = 1:i
12         s=0;
13         if i==j
14             for k = 1:j-1
15                 s=s+L(j,k)*L(j,k);
16             end
17             L(j,j)= sqrt(A(j,j)-s);
18         else
19             for k = 1:j-1
20                 s=s+L(i,k)*L(j,k);
21             end
22             L(i,j)= (A(i,j)-s)/L(j,j);
23         end
24     end
25 end
26 U = L';
27 disp(L,'Lower triangular matrix is :')
28 disp(U,'Upper triangular matrix is :')
```

Scilab code Exa 2.6 Jacobi method

```
1 //Solving the problem using Jacobi method
2 //the first case in converging and the 2nd is
   diverging ..... drawback
3 //of jacobi method
4 //the ans is correct to 2D place
5
6 clear;
7 close();
8 clc;
9 format('v',7);
10 x1=[0,0];
11 x2=[0,0];
12 x3=[0,0];
13 x1(1,2)=0.2*(6-2*x2(1,1)+x3(1,1));
14 x2(1,2)=0.16667*(4-x1(1,1)+3*x3(1,1));
15 x3(1,2)=0.25*(7-2*x1(1,1)-x2(1,1));
16 i=1;
17 while (abs(x1(1,1)-x1(1,2))>0.5*10^-2 | abs(x2(1,1)-
   x2(1,2))>0.5*10^-2 | abs(x3(1,1)-x3(1,2))
   >0.5*10^-2 )
18     x1(1,1)=x1(1,2);
19     x2(1,1)=x2(1,2);
20     x3(1,1)=x3(1,2);
21     x1(1,2)=0.2*(6-2*x2(1,1)+x3(1,1));
22     x2(1,2)=0.16667*(4-x1(1,1)+3*x3(1,1));
23     x3(1,2)=0.25*(7-2*x1(1,1)-x2(1,1));
24     i=i+1;
25 end
26 disp([x1; x2; x3], 'Answers are :')
27 disp(i, 'Number of Iterations :')
```

```

28
29
30 x1=[0,0];
31 x2=[0,0];
32 x3=[0,0];
33 x1(1,2)=4-6*x2(1,1)+3*x3(1,1);
34 x2(1,2)=0.5*(6-5*x1(1,1)+x3(1,1));
35 x3(1,2)=0.25*(7-2*x1(1,1)-x2(1,1));
36 i=1;
37 while (abs(x1(1,1)-x1(1,2))>0.5*10^-2 | abs(x2(1,1)-
      x2(1,2))>0.5*10^-2 | abs(x3(1,1)-x3(1,2))
      >0.5*10^-2 )
38     x1(1,1)=x1(1,2);
39     x2(1,1)=x2(1,2);
40     x3(1,1)=x3(1,2);
41     x1(1,2)=(4-6*x2(1,1)+3*x3(1,1));
42     x2(1,2)=0.5*(6-5*x1(1,1)+x3(1,1));
43     x3(1,2)=0.25*(7-2*x1(1,1)-x2(1,1));
44     i=i+1;
45 end
46 disp([x1; x2; x3], 'Answers are :')
47 disp(i, 'Number of Iterations :')

```

Scilab code Exa 2.7 Gauss Seidel method

```

1 //the problem is solved using Gauss-Seidel method
2 //it is fast convergent as compared to jacobi method
3
4 clear;
5 close();
6 clc;
7 format('v',7);
8 x1=[0,0];

```

```

9 x2=[0,0];
10 x3=[0,0];
11 x1(1,2)=0.2*(6-2*x2(1,1)+x3(1,1));
12 x2(1,2)=0.16667*(4-x1(1,2)+3*x3(1,1));
13 x3(1,2)=0.25*(7-2*x1(1,2)-x2(1,2));
14 i=1;
15 while (abs(x1(1,1)-x1(1,2))>0.5*10^-2 | abs(x2(1,1)-
        x2(1,2))>0.5*10^-2 | abs(x3(1,1)-x3(1,2))
        >0.5*10^-2 )
16     x1(1,1)=x1(1,2);
17     x2(1,1)=x2(1,2);
18     x3(1,1)=x3(1,2);
19     x1(1,2)=0.2*(6-2*x2(1,1)+x3(1,1));
20     x2(1,2)=0.16667*(4-x1(1,2)+3*x3(1,1));
21     x3(1,2)=0.25*(7-2*x1(1,2)-x2(1,2));
22     i=i+1;
23 end
24 disp([x1; x2; x3], 'Answers are :')
25 disp(i, 'Number of Iterations :')

```

Scilab code Exa 2.8 Successive over relaxation method

```

1 //The method used to solve is SOR(Successive over-
  relaxation) method
2 //only marginal improvement is possible for this
  particular system since
3 //Gauss-Seidel iteration itself converges quite
  rapidly
4
5 clear;
6 close();
7 clc;
8 format('v',7);

```

```

9  x1=[0,0];
10 x2=[0,0];
11 x3=[0,0];
12 w =0.9;
13 x1(1,2)=x1(1,1)+0.2*w*(6-5*x1(1,1)-2*x2(1,1)+x3(1,1)
    );
14 x2(1,2)=x2(1,1)+0.16667*w*(4-x1(1,2)-6*x2(1,1)+3*x3
    (1,1));
15 x3(1,2)=x3(1,1)+0.25*w*(7-2*x1(1,2)-x2(1,2)-4*x3
    (1,1));
16 i=1;
17 while (abs(x1(1,1)-x1(1,2))>0.5*10^-2 | abs(x2(1,1)-
    x2(1,2))>0.5*10^-2 | abs(x3(1,1)-x3(1,2))
    >0.5*10^-2 )
18     x1(1,1)=x1(1,2);
19     x2(1,1)=x2(1,2);
20     x3(1,1)=x3(1,2);
21     x1(1,2)=x1(1,1)+0.2*w*(6-5*x1(1,1)-2*x2(1,1)+x3
        (1,1));
22     x2(1,2)=x2(1,1)+0.16667*w*(4-x1(1,2)-6*x2(1,1)
        +3*x3(1,1));
23     x3(1,2)=x3(1,1)+0.25*w*(7-2*x1(1,2)-x2(1,2)-4*x3
        (1,1));
24     i=i+1;
25 end
26 disp([x1; x2; x3], 'Answers are:')
27 disp(i, 'Number of Iterations :')

```

Scilab code Exa 2.9 Gauss Seidel and SOR method

```

1 //Solving four linear system of equations with Gauss
    -Seidel and SOR method
2 //the convergence is much faster in SOR method

```

```

3
4 clear;
5 close();
6 clc;
7 format('v',7);
8 x1=[0,0];
9 x2=[0,0];
10 x3=[0,0];
11 x4=[0,0];
12 x1(1,2)=-0.33333*(1-x2(1,1)-3*x4(1,1));
13 x2(1,2)=0.16667*(1-x1(1,2)-x3(1,1));
14 x3(1,2)=0.16667*(1-x2(1,2)-x4(1,1));
15 x4(1,2)=-0.33333*(1-3*x1(1,2)-x3(1,2));
16 i=1;
17 while (abs(x1(1,1)-x1(1,2))>0.5*10^-2 | abs(x2(1,1)-
    x2(1,2))>0.5*10^-2 | abs(x3(1,1)-x3(1,2))
    >0.5*10^-2 | abs(x4(1,1)-x4(1,2))>0.5*10^-2)
18     x1(1,1)=x1(1,2);
19     x2(1,1)=x2(1,2);
20     x3(1,1)=x3(1,2);
21     x4(1,1)=x4(1,2);
22     x1(1,2)=-0.33333*(1-x2(1,1)-3*x4(1,1));
23     x2(1,2)=0.16667*(1-x1(1,2)-x3(1,1));
24     x3(1,2)=0.16667*(1-x2(1,2)-x4(1,1));
25     x4(1,2)=-0.33333*(1-3*x1(1,2)-x3(1,2));
26     i=i+1;
27 end
28 disp([x1; x2; x3; x4], 'Answers are:')
29 disp(i, 'Number of Iterations :')
30
31
32 w=1.6;
33 x1=[0,0];
34 x2=[0,0];
35 x3=[0,0];
36 x4=[0,0];
37 x1(1,2)=x1(1,1)-0.33333*w*(1+3*x1(1,1)-x2(1,1)-3*x4
    (1,1));

```

```

38 x2(1,2)=x2(1,1)+0.16667*w*(1-x1(1,2)-6*x2(1,2)-x3
    (1,1));
39 x3(1,2)=x3(1,1)+0.16667*w*(1-x2(1,2)-6*x3(1,2)-x4
    (1,1));
40 x4(1,2)=x4(1,1)-0.33333*w*(1-3*x1(1,2)-x3(1,2)+3*x4
    (1,1));
41 i=1;
42 while (abs(x1(1,1)-x1(1,2))>0.5*10^-2 | abs(x2(1,1)-
    x2(1,2))>0.5*10^-2 | abs(x3(1,1)-x3(1,2))
    >0.5*10^-2 | abs(x4(1,1)-x4(1,2))>0.5*10^-2)
43     x1(1,1)=x1(1,2);
44     x2(1,1)=x2(1,2);
45     x3(1,1)=x3(1,2);
46     x4(1,1)=x4(1,2);
47     x1(1,2)=x1(1,1)-0.33333*w*(1+3*x1(1,1)-x2(1,1)
        -3*x4(1,1));
48     x2(1,2)=x2(1,1)+0.16667*w*(1-x1(1,2)-6*x2(1,2)-
        x3(1,1));
49     x3(1,2)=x3(1,1)+0.16667*w*(1-x2(1,2)-6*x3(1,2)-
        x4(1,1));
50     x4(1,2)=x4(1,1)-0.33333*w*(1-3*x1(1,2)-x3(1,2)
        +3*x4(1,1));
51     i=i+1;
52 end
53 disp([x1; x2; x3; x4], 'Answers are :')
54 disp(i, 'Number of Iterations :')

```

Chapter 3

Non linear algebraic equations

Scilab code Exa 3.1 Bisection Method

```
1 //Bisection Method
2 clc;
3 clear;
4 close();
5 format('v',9);
6 b(1)=1;a(1)=0;k=5;
7 deff('[fx]=bisec(x)', 'fx =(x+1).^2.*exp(x.^2-2)-1');
8 x = linspace(0,1);
9 plot(x,((x+1).^2).*(exp(x.^2-2))-1);
10 //in interval [0,1] changes its sign thus has a root
11 //k = no of decimal place of accuracy
12 //a = lower limit of interval
13 //b = upper limit of interval
14 //n = no of iterations required
15 n = log2((10^k)*(b-a));
16 n = ceil(n);
17 disp(n, 'Number of iterations : ')
18 for i = 1:n-1
19     N(i) = i;
20     c(i) = (a(i)+b(i))/2;
21     bs(i) = bisec(c(i));
```

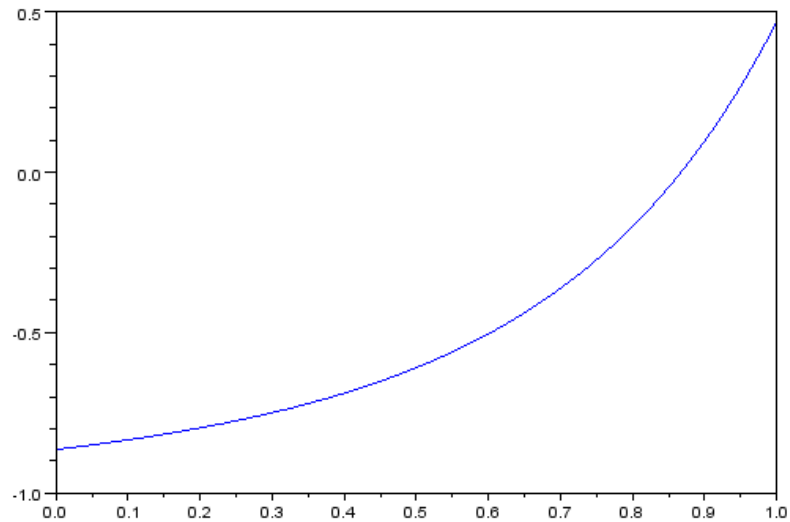



Figure 3.1: Bisection Method

```

22     if (bisec(b(i))*bisec(c(i))<0)
23         a(i+1)=c(i);
24         b(i+1)=b(i);
25     else
26         b(i+1)=c(i);
27         a(i+1)=a(i);
28     end
29 end
30 N(i+1)=i+1;
31 c(i+1) = (a(i+1)+b(i+1))/2;
32 bs(i+1) = bisec(c(i));
33 ann = [N a b c bs];
34 disp(ann , 'The Table : ');
35 disp(c(i),'The root of the function is : ')

```

Scilab code Exa 3.2 False positioning method

```
1 //The solution using false position method
2 clc;
3 clear;
4 close();
5 b(1)=1;a(1)=0;k=5;i=1;
6 format('v',9);
7 deff('[fx]=bisec(x)', 'fx =(x+1)^2*exp(x^2-2)-1');
8 x = linspace(0,1);
9 plot(x,((x+1).^2).*(exp(x.^2-2))-1);
10 //in interval [0,1] changes its sign thus has a root
11 //k = no of decimal place of accuracy
12 //a = lower limit of interval
13 //b = upper limit of interval
14 c(i) = (a(i)*bisec(b(i))-b(i)*bisec(a(i)))/(bisec(b(
    i))-bisec(a(i)));
15 bs(1)=bisec(c(1));
16 N(1) = 1;
17 a(i+1)=c(i);
18 b(i+1)=b(i);
19 while abs(bisec(c(i)))>(0.5*(10^-k))
20     i = i+1;
21     N(i)=i;
22     c(i) = (a(i)*bisec(b(i))-b(i)*bisec(a(i)))/(
        bisec(b(i))-bisec(a(i)));
23     bs(i) = bisec(c(i));
24     if (bisec(b(i))*bisec(c(i))<0)
25         a(i+1)=c(i);
26         b(i+1)=b(i);
27     else
28         b(i+1)=c(i);
```

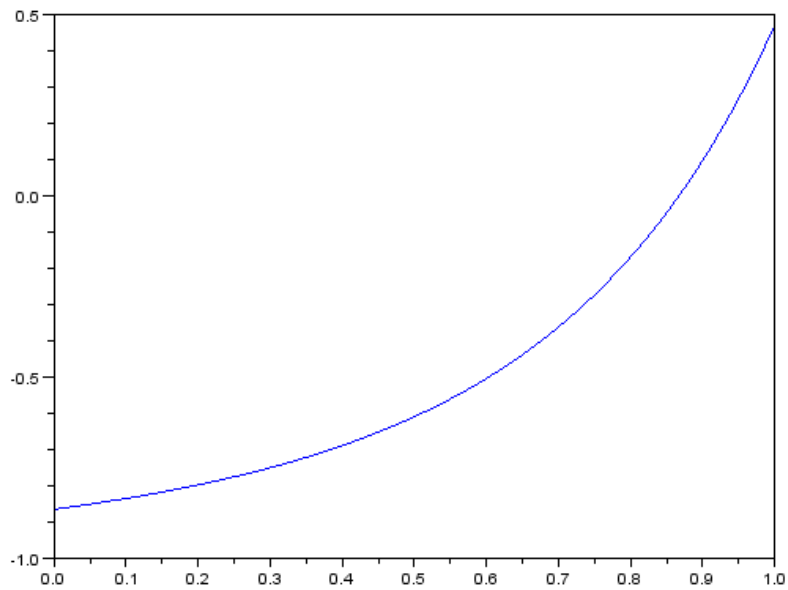


Figure 3.2: False positioning method

```

29         a(i+1)=a(i);
30     end
31 end
32 a(10)=[];b(10)=[];
33 ann = [N a b c bs];
34 disp(ann , 'The Table : ');
35 disp('The root of the function is bracketed by
      [0.647116 1] ');

```

Scilab code Exa 3.3 fixed point iteration method

```
1 //We have quadratic equation  $x^2-2x-8=0$  with roots
   -2 and 4
2 //for solving it we use fixed point iteration method
   for that we rearrange it in 3 ways.
3 //first way  $x=(2x+8)^{(1/2)}$ 
4 //here x0 is chosen arbitrarily
5
6 clear;
7 clc;
8 close();
9 format('v',5)
10 funcprot(0);
11 deff('[fixed_point]=fx(x)', 'fixed_point=(2*x+8)^0.5'
   )
12 x0=5;
13 while abs(x0-fx(x0))>0.5*10^(-2)
14     x0=fx(x0);
15 end
16 disp(x0, 'root is :')
17
18 //second way  $x=(2x+8)/x$ 
19
20 format('v',5)
21 funcprot(0);
22 deff('[fixed_point]=fx(x)', 'fixed_point=(2*x+8)/x')
23 x0=5;
24 while abs(x0-fx(x0))>0.5*10^(-2)
25     x0=fx(x0);
26 end
27 disp(x0, 'root is :')
28
29 //third way  $x=(x^2-8)/2$ 
30
31 format('v',10)
32 funcprot(0);
33 deff('[fixed_point]=fx(x)', 'fixed_point=(x^2-8)/2')
```

```

34 x0=5;
35 for i=1:5
36     x0=fx(x0);
37     disp(x0,'value is :')
38 end
39 disp(x0,'As you can see that the root is not
    converging.So this method is not applicable.')
```

Scilab code Exa 3.4 Type of convergence

```

1 //checking for the convergence and divergence of
  different functions we are getting after
  rearrangement of the given quadratic equation  $x^2-2*x-8=0$ .
2 //after first type of arrangement we get a function
   $gx=(2*x+8)^{(1/2)}$ .for this we have..
3
4 clear;
5 clc;
6 close();
7 alpha=4;
8 I=alpha-1:alpha+1;//required interval
9 deff(' [f1]=gx(x) ','f1=(2*x+8)^(1/2) ');
10 deff(' [f2]=diffgx(x) ','f2=(2*x+8)^(-0.5) ');
11 x=linspace(3,5);
12 subplot(2,1,1);
13 plot(x,(2*x+8)^(1/2))
14 plot(x,x)
15 x0=5;
16 if diffgx(I)>0
17     disp('Errors in two consecutive iterates are of
    same sign so convergence is monotonic')
18 end
```

```

19 if abs(diffgx(x0))<1
20     disp('So this method converges')
21 end
22
23 //after second type of arrangement we get a function
    gx=(2*x+8)/x.for this we have..
24
25 deff(' [f1]=gx(x) ', 'f1=(2*x+8)/x ');
26 deff(' [f2]=diffgx(x) ', 'f2=(-8)/(x^2) ');
27 x=linspace(1,5);
28 for i=1:100
29     y(1,i)=2+8/x(1,i);
30 end
31 subplot(2,1,2);
32 plot(x,y)
33 plot(x,x)
34 x0=5;
35 if diffgx(I)<0
36     disp('Errors in two consecutive iterates are of
        opposite sign so convergence is oscillatory')
37 end
38 if abs(diffgx(x0))<1
39     disp('So this method converges')
40 end

```

Scilab code Exa 3.5 Newton Method

```

1 //Newton's Method
2 //the first few iteration converges quickly in
    negative root as compared to positive root
3 clc;

```

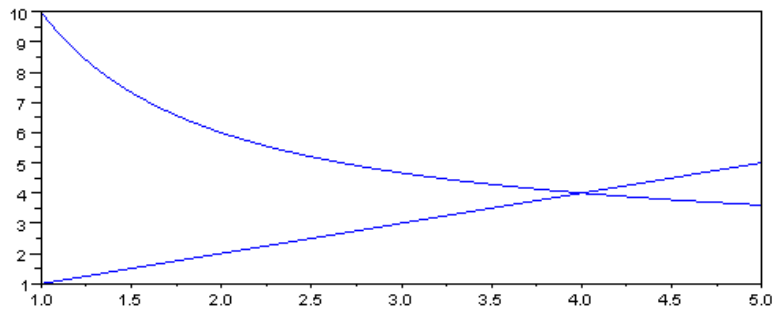
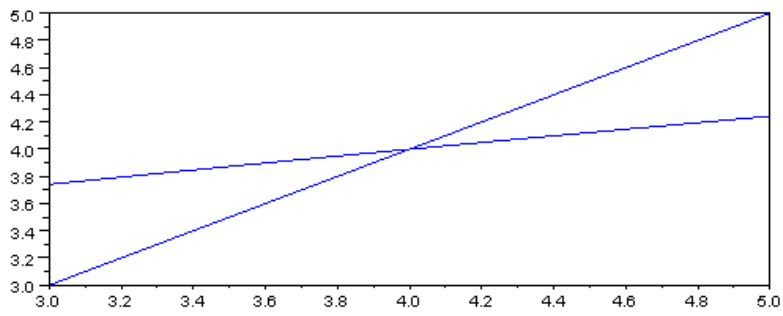


Figure 3.3: Type of convergence

```

4 clear;
5 close();
6 funcprot(0);
7 format('v',9);
8 deff('[Newton]=fx(x)', 'Newton=exp(x)-x-2');
9 deff('[diff]=gx(x)', 'diff=exp(x)-1');
10 x = linspace(-2.5,1.5);
11 plot(x,exp(x)-x-2)
12 //from the graph the function has 2 roots
13 //considering the initial negative root -10
14 x1 = -10;
15 x2 = x1-fx(x1)/gx(x1);
16 i=0;
17 while abs(x1-x2)>(0.5*10^-7)
18     x1=x2;
19     x2 = x1-fx(x1)/gx(x1);
20     i=i+1;
21 end
22 disp(i,'Number of iterations : ')
23 disp(x2,'The negative root of the function is : ')
24
25
26 //considering the initial positive root 10
27 x1 = 10;
28 x2 = x1-fx(x1)/gx(x1);
29 i=0;
30 while abs(x1-x2)>(0.5*10^-7)
31     x1=x2;
32     x2 = x1-fx(x1)/gx(x1);
33     i=i+1;
34 end
35 disp(i,'Number of iteration : ')
36 disp(x2,'The positive root of the function is : ')
37 //number of iterations showing fast and slow
    convergent

```

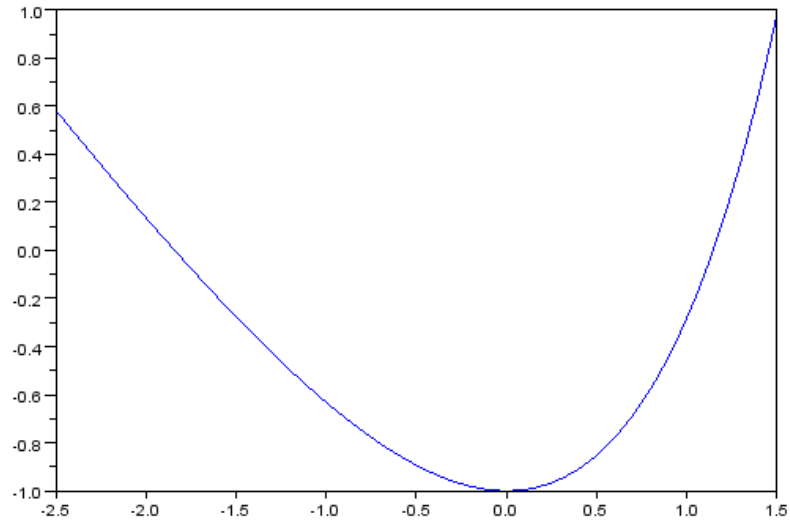


Figure 3.4: Newton Method

Scilab code Exa 3.6 Secant Method

```

1 //Secant Method
2 //the first few iteration converges quickly in
   negative root as compared to positive root
3 clc;
4 clear;
5 close();
6 funcprot(0);
7 format('v',9);
8 deff('[Secant]=f(x)', 'Secant=exp(x)-x-2');

```

```

 9 x = linspace(0,1.5);
10 subplot(2,1,1);
11 plot(x,exp(x)-x-2);
12 plot(x,0);
13 //from the graph the function has 2 roots
14 //considering the initial negative root -10
15 x0 = -10
16 x1 = -9;
17 x2 = (x0*f(x1)-x1*f(x0))/(f(x1)-f(x0));
18 i=0;
19 while abs(x1-x2)>(0.5*10^-7)
20     x0=x1;
21     x1=x2;
22     x2 = (x0*f(x1)-x1*f(x0))/(f(x1)-f(x0));
23     i=i+1;
24 end
25 disp(i,'Number of iterations : ')
26 disp(x2,'The negative root of the function is : ')
27
28
29 //considering the initial positive root 10
30 subplot(2,1,2);
31 x = linspace(-2.5,0);
32 plot(x,exp(x)-x-2);
33 plot(x,0);
34 x0 = 10
35 x1 = 9;
36 x2 = (x0*f(x1)-x1*f(x0))/(f(x1)-f(x0));
37 i=0;
38 while abs(x1-x2)>(0.5*10^-7)
39     x0=x1;
40     x1=x2;
41     x2 = (x0*f(x1)-x1*f(x0))/(f(x1)-f(x0));
42     i=i+1;
43 end
44 disp(i,'Number of iteration : ')
45 disp(x2,'The positive root of the function is : ')
46 //number of iterations showing fast and slow

```

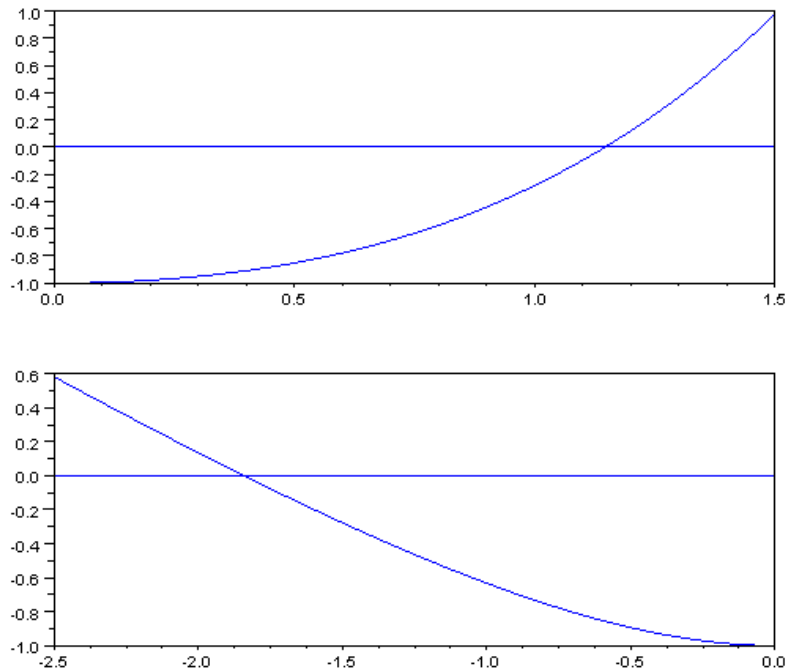


Figure 3.5: Secant Method

```

47     convergent
48     format('v',6)
49     //Order of secant method (p)
50     p = log(31.52439)/log(8.54952);
51     disp(p,'Order of Secant Method : ')

```

Scilab code Exa 3.7 System of Non Linear Equations

```

1 //Non-Linear Equation
2 clc;
3 clear;
4 close();
5 funcprot(0);
6 format('v',9);
7 i = 1;
8 deff(' [func1]=f(x,y)', 'func1=x^2+y^2-4');
9 deff(' [func2]=g(x,y)', 'func2=2*x-y^2');
10 deff(' [difffx]=fx(x)', 'difffx=2*x');
11 deff(' [diffgy]=fy(y)', 'diffgy=2*y');
12 deff(' [diffgx]=gx(x)', 'diffgx=2');
13 deff(' [diffgy]=gy(y)', 'diffgy=-2*y');
14 x1(i)=1;y1(i)=1;
15 J = [fx(x1(i)),fy(y1(i));gx(x1(i)),gy(y1(i))];
16 n=[x1(i);y1(i)]-inv(J)*[f(x1(i),y1(i));g(x1(i),y1(i)
    )]);
17 x2(i)=n(1,1);y2(i)=n(2,1);
18 N(1)=i-1;
19 while (abs(x2(i)-x1(i))>0.5*10^-7) | (abs(y2(i)-y1(i)
    ))>0.5*10^-7)
20     i=i+1;
21     N(i)=i-1;
22     x1(i)=x2(i-1);
23     y1(i)=y2(i-1);
24     J = [fx(x1(i)),fy(y1(i));gx(x1(i)),gy(y1(i))];
25     n=[x1(i);y1(i)]-inv(J)*[f(x1(i),y1(i));g(x1(i),
        y1(i))];
26     x2(i)=n(1,1);y2(i)=n(2,1);
27 end
28 N(i+1)=i;
29 x1(i+1) = x2(i);
30 y1(i+1) = y2(i);
31 n = [N x1 y1];
32 disp(n,'The value of n x and y :')

```

Scilab code Exa 3.8 System of Non Linear Equations

```
1 //Non-Linear Equation
2 clc;
3 clear;
4 close();
5 funcprot(0);
6 format('v',9);
7 deff(' [ func1]=f(x1 ,x2) ', ' func1 = -2.0625*x1+2*x2
    -0.0625*x1^3+0.5 ');
8 deff(' [ func2]=g(x1 ,x2 ,x3) ', ' func2=x3-2*x2+x1 -0.0625*
    x2^3+0.125*x2*(x3-x1) ');
9 deff(' [ func3]=h(x2 ,x3 ,x4) ', ' func3=x4-2*x3+x2 -0.0625*
    x3^3+0.125*x3*(x4-x2) ');
10 deff(' [ func4]=k(x3 ,x4) ', ' func4 = -1.9375*x4+x3 -0.0625*
    x4^3-0.125*x3*x4+0.5 ');
11 //define the corresponding partial differenciatio
    of the function = 16
12 deff(' [ difffx1]=fx1(x1) ', ' difffx1 = -2.0625-3*0.0625*
    x1^2 ');
13 deff(' [ difffx2]=fx2(x2) ', ' difffx2=2 ');
14
15 deff(' [ diffgx1]=gx1(x2) ', ' diffgx1=1-0.125*x2 ');
16 deff(' [ diffgx2]=gx2(x1 ,x2 ,x3) ', ' diffgx2 = -2-3*0.0625*
    x2^2+0.125*(x3-x1) ');
17 deff(' [ diffgx3]=gx3(x2) ', ' diffgx3=1+0.125*x2 ');
18
19 deff(' [ diffhx2]=hx2(x3) ', ' diffhx2=1-0.125*x3 ');
20 deff(' [ diffhx3]=hx3(x3 ,x4) ', ' diffhx3 = -2-0.0625*3*x3
    ^2+0.125*x4 ');
21 deff(' [ diffhx4]=hx4(x3) ', ' diffhx4 = 1+0.125*x3 ');
22
```

```

23 deff(' [ diffkx3]=kx3(x4) ', ' diffkx3=1-0.125*x4 ');
24 deff(' [ diffkx4]=kx4(x3, x4) ', ' diffkx4
    =-1.9375-3*0.0625*x4^2-0.125*x3 ');
25
26 x = [1.5 1.25 1.0 0.75]';
27 for i=1:6
28     N(i)=i-1;
29     x1(i) = x(1);x2(i)=x(2);x3(i) = x(3);x4(i)=x(4);
30     J = [fx1(x(1)),fx2(x(2)),0,0;gx1(x(2)),gx2(x(1),
        x(2),x(3)),gx3(x(2)),0;0,hx2(x(3)),hx3(x(3),x
        (4)),hx4(x(3));0,0,kx3(x(4)),kx4(x(3),x(4))];
31     x = x - inv(J)*[f(x(1),x(2));g(x(1),x(2),x(3));h
        (x(2),x(3),x(4));k(x(3),x(4))];
32 end
33 n = [N x1 x2 x3 x4];
34 disp(n, 'The values of N x1 x2 x3 x4 respectively : '
    );

```

Chapter 4

Eigenvalues and eigenvectors

Scilab code Exa 4.1 Power Method of finding largest Eigen value

```
1 //The Power Method of finding largest Eigen value of
   given matrix
2 clear;
3 clc;
4 close();
5 A=[3 0 1;2 2 2;4 2 5]; //Given Matrix
6 u0=[1 1 1]'; //Intial vector
7 v=A*u0;
8 a=max(u0);
9 while abs(max(v)-a)>0.05 //for accuracy
10     a=max(v);
11     u0=v/max(v);
12     v=A*u0;
13 end
14 format('v',4);
15 disp(max(v),'Eigen value :')
16 format('v',5);
17 disp(u0,'Eigen vector :')
```

Scilab code Exa 4.2 Power Method of finding largest Eigen value

```
1 //The Power Method of finding largest Eigen value of
   given matrix
2 clear;
3 clc;
4 close();
5 A=[3 0 1;2 2 2;4 2 5];
6 new_A=A-7*eye(3,3); //Given Matrix
7 u0=[1 1 1]'; //Intial vector
8 v=new_A*u0;
9 a=max(abs(u0));
10 while abs(max(abs(v))-a)>0.005 //for accuracy
11     a=max(abs(v));
12     u0=v/max(abs(v));
13     v=new_A*u0;
14 end
15 format('v',5);
16 disp(max(v),'Eigen value :')
17 format('v',5);
18 disp(u0,'Eigen vector :')
```

Scilab code Exa 4.3 Convergence of Inverse Iteration

```
1 //Convergence of Inverse Iteration
2 clc;
3 clear;
4 close();
```



```

5 format('v',4);
6 A = [3 0 1;2 2 2; 4 2 5];
7 e1 = 7.00;
8 e2 = 1.02;
9 p = sum(diag(A))-e1-e2;
10 disp(A, 'A = ');
11 A = A - p*eye(3,3);
12 disp(A, 'A-1.98I = ');
13 L = [1 0 0; 0.50 1 0; 0.26 0.52 1];
14 U = [4 2 3.02; 0 -.98 0.49; 0 0 -.03];
15 disp(L,U, 'The decomposition of A - 1.98I (L,U): ');
16 u = [1,1,1]';
17 I = inv(U)*inv(L);
18 for i = 1:3
19     v = inv(U)*inv(L)*u;
20     disp(max(v),v,u,i-1, 'The values of s u(s) v(s+1)
        and max(v(s+1)) : ');
21     u = v./max(v);
22 end
23 disp(u, 'The Eigen Vector : ');
24 ev = p+1/max(v);
25 disp(ev, 'The approx eigen value : ');

```

Scilab code Exa 4.4 Deflation

```

1 //Deflation
2 clc;
3 clear;
4 close();
5 A = [10 -6 -4; -6 11 2; -4 2 6];
6 P = [1 0 0;-1 1 0;-0.5 0 1];
7 disp(P,A, 'The A and the P(transformation matrix) are
        : ');

```

```

8 B = inv(P)*A*P;
9 disp(B, 'Hence B = ');
10 C = B;
11 C(1,:) = [];
12 C(:,1) = [];
13 disp(C, 'The deflated matrix : ');
14 Y = spec(C);
15 disp(Y, 'The matrix A therefore has eigen values : ');
16 e1 = [1/3, 1, -1/2]';
17 e2 = [2/3, 1, 1]';
18 disp(e1, e2, 'The eigen values of B are : ');
19 x1 = P*e1;
20 x2 = P*e2;
21 disp(3/2.*x1, 3/2.*x2, 'The eigen vextors of the
    original matrix A : ')

```

Scilab code Exa 4.5 Threshold serial Jacobi Method

```

1 //Threshold serial Jacobi Method
2 //taking threshold values 0.5 and 0.05
3 clc;
4 clear;
5 close();
6 format('v',9);
7 A = [3 0.4 5;0.4 4 0.1;5 0.1 -2];
8 //for first cycle |0.4|<0.5 tranformation is
    omitted
9 //|5|>0.5 a zero is created at (1,3)
10 //by taking the rotation matrix P1=[c 0 s; 0 1 0;-s
    0 c]; where c=cos and s=sin
11 //O is theta
12 p=1;q=3;

```

```

13 0 = 0.5*atan(2*A(p,q)/(A(q,q)-A(p,p)));
14 P1 = [cos(0) 0 sin(0);0 1 0;-sin(0) 0 cos(0)];
15 A1 = A;
16 A2 = inv(P1)*A*P1;
17 //as all the off-diagonals < 0.5 the first cycle is
    complete
18 disp(diag(A2),'The eigen values for case 1 : ')
19
20 //second cycle for 0.05
21 count =0;
22 EV = P1;
23 for i=1:3
24     for j=i+1:3
25         if A2(i,j)>0.05 then
26             p=i;q=j;
27             0 = 0.5*atan(2*A2(p,q)/(A2(q,q)-A2(p,p))
                );
28             c = cos(0);
29             s = sin(0);
30             P = eye(3,3);
31             P(p,p)=c;
32             P(q,q)=c;
33             P(p,q)=s;
34             P(q,p)=-s;
35             A = inv(P)*A2*P;
36             disp(EV,'value of P*')
37             EV = EV * P;
38             count = count+1;
39         end
40     end
41 end
42 //eigen values are the diagonal elements of A and
    the column of P gives eigen vectors
43 disp(diag(A),'Eigen values : ')
44 disp(EV,'Corresponding eigen vectors : ')

```

Scilab code Exa 4.6 The Gerchgorin circle

```
1 //The Gerchgorin circle
2 clc;
3 clear;
4 close();
5 format('v',9);
6 x = [0:.1:14];
7 plot2d(0,0,-1,"031", " ", [0,-5,14,5]);
8 plot(x,0);
9 A = [5 1 0;-1 3 1;-2 1 10];
10 disp(A, 'A = ');
11 for i=1:3
12     disp(A(i,i), 'Centers are : ');
13     radius = 0;
14     for j=1:3
15         if j~=i then
16             radius = radius + abs(A(i,j));
17         end
18     end
19     disp(radius, 'Radius : ');
20     xarc(A(i,i)-radius, radius, 2*radius, 2*radius
21         ,0,360*64);
21 end
22 disp('The figure indicates that 2 of the eigenvalues
23     of A lie inside the intersected region of 2
24     circles, and the remaining eigen value in the
25     other circle.');
```

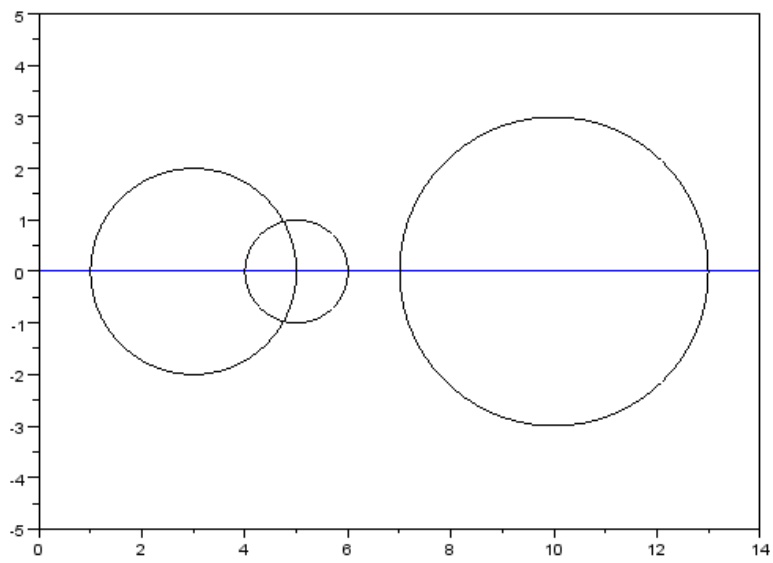


Figure 4.1: The Gerchgorin circle

Scilab code Exa 4.7 Sturm sequence property

```
1 //Sturm sequence property
2 clc;
3 clear;
4 close();
5 C=[2,4,0,0;4,10,3,0;0,3,9,-1;0,0,-1,5];
6 //find the eigen vClues lying (0,5)
7 p=0;
8
9 f(1)=1;
10 f(2)=C(1,1)-p;
11 count = 0;
12 if f(1)*f(2)>=0 then
13     count = 1;
14 end
15 for r=3:5
16     br=C(r-2,r-1);
17     f(r)=-br^2*f(r-2)+(C(r-1,r-1)-p)*f(r-1);
18     if f(r)*f(r-1)>=0 then
19         count = count+1;
20     end
21 end
22 disp(f,'Sturm sequences')
23 disp(count,'Number of eigen values strickly greater
    than 0 : ')
24
25 p=5;
26 f(1)=1;
27 f(2)=C(1,1)-p;
28 count1 = 0;
29 if f(1)*f(2)>=0 then
```

```

30     count1 = 1;
31 end
32 for r=3:5
33     br=C(r-2,r-1);
34     f(r)=-br^2*f(r-2)+(C(r-1,r-1)-p)*f(r-1);
35     if f(r)*f(r-1)>=0 then
36         count1 = count1+1;
37     end
38 end
39 disp(f, 'Sturm sequences ')
40 disp(count1, 'Number of eigen values strickly greater
    than 5 : ')
41 disp(count-count1, 'Number of eigen values between 0
    and 5 : ')

```

Scilab code Exa 4.8 Gerschgorins first theorem

```

1 //Gerschgorin 's first theorem
2 clc;
3 clear;
4 close();
5 //find the eigen values lying [0,4] with an error of
    0.25
6 //taking p at mid point of the interval
7 C=[2,-1,0;-1,2,-1;0,-1,1];
8 p=2;
9
10 f(1)=1;
11 f(2)=C(1,1)-p;
12 count = 0;
13 if f(1)*f(2)>0 then
14     count = 1;
15 end

```

```

16 for r=3:4
17     br=C(r-2,r-1);
18     f(r)=-br^2*f(r-2)+(C(r-1,r-1)-p)*f(r-1);
19     if f(r)*f(r-1)>0 then
20         count = count+1;
21 //     elseif f(r-1)==0 && f(r-1)*          ??????
22         check for sign when f(r)=zero
23     end
24 end
25 disp(f, 'Sturm sequences')
26 disp(count, 'Number of eigen values strickly greater
27     than 2 : ')
28
29 p=1;
30 f(1)=1;
31 f(2)=C(1,1)-p;
32 count1 = 0;
33 if f(1)*f(2)>0 then
34     count1 = 1;
35 end
36 for r=3:4
37     br=C(r-2,r-1);
38     f(r)=-br^2*f(r-2)+(C(r-1,r-1)-p)*f(r-1);
39     if f(r)*f(r-1)>0 then
40         count1 = count1+1;
41     end
42 end
43 disp(f, 'Sturm sequences')
44 disp(count1, 'Number of eigen values strickly greater
45     than 1 : ')
46
47 p=1.5;
48 f(1)=1;
49 f(2)=C(1,1)-p;
50 count2 = 0;
51 if f(1)*f(2)>0 then
52     count2 = 1;
53 end

```



```

51 for r=3:4
52     br=C(r-2,r-1);
53     f(r)=-br^2*f(r-2)+(C(r-1,r-1)-p)*f(r-1);
54     if f(r)*f(r-1)>0 then
55         count2 = count2+1;
56     end
57 end
58 disp(f, 'Sturm sequences')
59 disp(count2, 'Number of eigen values strickly greater
    than 1.5 : ')
60 disp(p+0.25, 'Eigen value lying between [1.5,2] ie
    with an error of 0.25 is : ')

```

Scilab code Exa 4.9 Givens Method

```

1 //Given's Method
2 //reduce A1 to tridiagonal form
3 clc;
4 clear;
5 close();
6 format('v',7);
7 A1 = [2 -1 1 4;-1 3 1 2;1 1 5 -3;4 2 -3 6];
8 disp(A1, 'A = ')
9 // zero is created at (1,3)
10 //by taking the rotation matrix X1=[c 0 s; 0 1 0;-s
    0 c]; where c=cos and s=sin
11 //O is theta
12
13 count =0;
14 for i=1:(4-2)
15     for j=i+2:4
16         if abs(A1(i,j))>0 then
17             p=i+1;q=j;

```

```

18         0 = -atan(A1(p-1,q)/(A1(p-1,p)));
19         c = cos(0);
20         s = sin(0);
21         X = eye(4,4);
22         X(p,p)=c;
23         X(q,q)=c;
24         X(p,q)=s;
25         X(q,p)=-s;
26
27         A1 = X'*A1*X;
28         disp(A1, 'Ai = ');
29         disp(X, 'X = ');
30         disp(0, 'Theta = ');
31         count = count+1;
32     end
33 end
34 end
35 disp(A1, 'Reduced A1 to trigonal matrix is : ')

```

Scilab code Exa 4.10 Householder Matrix

```

1 //Householder Matrix
2 clc;
3 clear;
4 close();
5 format('v',7);
6 e = [1;0;0];
7 x = [-1;1;4];
8 disp(e, 'e = ');
9 disp(x, 'x = ');
10 //considering the positive k according to sign
    convention
11 k = sqrt(x'*x);

```

```

12 disp(k, 'k = ');
13 u = x - k*e;
14 disp(u, 'u = ');
15 Q = eye(3,3) - 2*u*u'/(u'*u);
16 disp(Q, 'Householder Matrix : ')

```

Scilab code Exa 4.11 Householder methods

```

1 //Householder methods
2 clc;
3 clear;
4 close();
5 format('v',7);
6 A = [2 -1 1 4;-1 3 1 2;1 1 5 -3;4 2 -3 6];
7 disp(A, 'A = ');
8 n=4;
9 for r=1:n-2
10     x = A(r+1:n,r);
11     f = eye(n-r,n-r);
12     e = f(:,1)
13     I = eye(r,r);
14     O(1:n-r,r) = 0;
15     //calculating Q
16     k = sqrt(x'*x);
17     u = x - k*e;
18     Q = eye(n-r,n-r) - 2*u*u'/(u'*u);
19     //substituting in P
20     P(1:r,1:r)= I;
21     P(r+1:n,1:r)=0;
22     P(1:r,r+1:n)=0;
23     P(r+1:n,r+1:n)=Q;
24     A = P*A*P;
25     disp(A,Q,P, 'The P Q and A matrix are ; ')

```

```

26 end
27 C = A;
28 disp(C, 'The tridiagonal matrix by householder method
      is : ')

```

Scilab code Exa 4.12 stable LR method

```

1 //stable LR method
2 clc;
3 clear;
4 close();
5 format('v',7);
6 A = [2 1 3 1;-1 2 2 1;1 0 1 0;-1 -1 -1 1];
7 disp(A, 'A = ');
8 for i = 1:6
9     [L,R,P]= lu(A);
10    A = R*P*L;
11    disp(A,R,L, 'The L R and A matrix are : ');
12 end
13 disp(A, 'The (1,1) and (4,4) elements have converged
      to real eigenvalues ')
14 X = [A(2,2) A(2,3);A(3,2) A(3,3)];
15 E = spec(X);
16 disp(E, 'Although submatrix themselves are not
      converging their eigen values converges. ')

```

Scilab code Exa 4.13 Orthogonal decomposition QR method

```

1 //Orthogonal decomposition – QR method

```

```

2 //reduce A to tridiagonal form
3 clc;
4 clear;
5 close();
6 format('v',7);
7 A1 = [1 4 2;-1 2 0;1 3 -1];
8 disp(A1, 'A = ');
9 // zero is created in lower triangle
10 //by taking the rotation matrix X1=[c s 0;-s c 0;0 0
    1]; where c=cos and s=sin
11 //O is theta
12
13 Q = eye(3,3);
14 for i=2:3
15     for j=1:i-1
16         p=i;q=j;
17         O = -atan(A1(p,q)/(A1(q,q)));
18         c = cos(O);
19         s = sin(O);
20         X = eye(3,3);
21         X(p,p)=c;
22         X(q,q)=c;
23         X(p,q)=-s;
24         X(q,p)=s;
25         A1 = X'*A1;
26         Q = Q*X;
27         disp(A1,X, 'The X and A matrix : ');
28     end
29 end
30 R = A1;
31 disp(R,Q, 'Hence the original matrix can be
    decomposed as : ')

```

Scilab code Exa 4.14 Reduction to upper Hessenberg form

```
1 //Redduction to upper Hessenberg form
2 clc;
3 clear;
4 close();
5 format('v',7);
6 A1 = [4 2 1 -3;2 4 1 -3;3 2 2 -3;1 2 1 0];
7 disp(A1, 'A = ');
8 //the element with largest modulus below diagonal in
   first column need to be at the top and then
   similarly for column 2
9 A1=gsort(A1, 'lr');
10 temp = A1(:,3);
11 A1(:,3) = A1(:,2);
12 A1(:,2) = temp;
13 M1 = eye(4,4);
14 M1(3,2) = A1(3,1)/A1(2,1);
15 M1(4,2) = A1(4,1)/A1(2,1);
16 A2 = inv(M1)*A1*M1;
17 disp(A2,M1, 'M1 and A2 : ')
18 A2=gsort(A2, 'lr');
19 temp = A2(:,3);
20 A2(:,3) = A2(:,4);
21 A2(:,4) = temp;
22 M2 = eye(4,4);
23 M2(4,3) = A2(4,2)/A2(3,2);
24 A3 = inv(M2)*A2*M2;
25 disp(M2, 'M2 = ');
26 disp(A3, 'Upper Hessenberg Matrix :')
27
28
29 //for i=2:n
30 //     M=eye(4,4);
31 //     for j=i+1:n
32 //         M(j,i) = A(j,);
33 //     end
34 //end
```

Scilab code Exa 4.15 Redduction to upper Hessenberg form and calculating eigen values

```
1 //Redduction to upper Hessenberg form and
   calculating eigen values
2 clc;
3 clear;
4 close();
5 format('v',7);
6 A1 = [4 2 1 -3;2 4 1 -3;3 2 2 -3;1 2 1 0];
7 //the element with largest modulus below diagonal in
   first column need to be at the top and then
   similarly for column 2
8 A1=gsort(A1,'lr');
9 temp = A1(:,3);
10 A1(:,3) = A1(:,2);
11 A1(:,2) = temp;
12 M1 = eye(4,4);
13 M1(3,2) = A1(3,1)/A1(2,1);
14 M1(4,2) = A1(4,1)/A1(2,1);
15 A2 = inv(M1)*A1*M1;
16
17 A2=gsort(A2,'lr');
18 temp = A2(:,3);
19 A2(:,3) = A2(:,4);
20 A2(:,4) = temp;
21 M2 = eye(4,4);
22 M2(4,3) = A2(4,2)/A2(3,2);
23 A3 = inv(M2)*A2*M2;
24 H = A3;
25 disp(H,'Upper Hessenberg Matrix :')
26 l =0;
```

```
27 for i=4:-1:1
28     K =H(1:i,1:i);
29     while abs(K(i,i)-1)>0.005
30         l=K(i,i);
31         [Q,R]=qr(K-K(i,i)*eye(i,i));
32         K = R*Q + K(i,i)*eye(i,i);
33     end
34     l = 0;
35     EV(i) = K(i,i);
36 end
37 disp(EV,'Eigen Values : ')
```

Chapter 5

Methods of approximation theory

Scilab code Exa 5.1 Lagranges Method of interpolation

```
1 //Construction of the quadratic interpolating
   polynomial to the function f(x)=ln(x) by using
   Lagrange's Method of interpolation.
2
3 close();
4 clear;
5 clc;
6 xi = linspace(2,3,3);
7 format('v',10);
8 y = [0.69315 0.91629 1.09861];
9 x = poly(0, 'x');
10
11 //Following are the polynomials
12
13 L0 = (x-xi(2))*(x-xi(3))/((xi(1)-xi(2))*(xi(1)-xi(3)
   ));
14 L1 = (x-xi(1))*(x-xi(3))/((xi(2)-xi(1))*(xi(2)-xi(3)
   ));
15 L2 = (x-xi(1))*(x-xi(2))/((xi(3)-xi(1))*(xi(3)-xi(2))
```

```

    ));
16 p2 = L0*y(1) + L1*y(2) + L2*y(3);
17 disp(p2 , 'The Required Polynomial : ')
18
19 //Showing the difference between actual and obtained
    value
20 format('v',8);
21 disp(log(2.7), 'Actual Value of Polynomial at x=2.7')
22 disp(horner(p2,2.7), 'Obtained Value of Polynomial at
    x=2.7')
23
24 err = log(2.7)-horner(p2,2.7);
25 disp(err , 'Error in approximation : ')

```

Scilab code Exa 5.2 Theoretical bound on error

```

1 //Theoretical bound on error
2 //it needs Symbolic Toolbox
3 //cd ~\Desktop\maxima_symbolic;
4 //exec 'symbolic.sce'
5 clc;
6 clear;
7 close();
8 syms x;
9 fx = log(x);
10 n = 2;
11 x0 = 2;
12 x1 = 2.5;
13 x2 = 3;
14 diff1_fx = diff(fx,x);
15 diff2_fx = diff(diff1_fx,x);
16 diff3_fx = diff(diff2_fx,x);
17 //so fx satisfies the continuity conditions on [2,3]

```

```

18 x= poly(0, 'x');
19 eta = linspace(2,3,100);
20 //fx-p2x is equal to
21 func = (x-2)*(x-2.5)*(x-3)*2/(factorial(3)*eta^3);
22 min_func = (x-2)*(x-2.5)*(x-3)*2/(factorial(3)*min(
    eta)^3);
23 disp(min_func , 'func will be less than or equal to'
    );
24 x = 2.7;
25 max_error = abs(horner(min_func,x));
26 disp(max_error , 'Error does not exceed :');

```

Scilab code Exa 5.3 Divided difference

```

1 //Divided difference for the functin = ln(x)
2 clc;
3 clear;
4 close();
5 format('v',9);
6 x = [1 1.5 1.75 2];
7 fx = [0 0.40547 0.55962 0.69315];
8 fab(1) = (fx(2)-fx(1))/(x(2)-x(1));
9 fab(2) = (fx(3)-fx(2))/(x(3)-x(2));
10 fab(3) = (fx(4)-fx(3))/(x(4)-x(3));
11 fabc(1)= (fab(2)-fab(1))/(x(3)-x(1));
12 fabc(2)= (fab(3)-fab(2))/(x(4)-x(2));
13 fabcd(1)= (fabc(2)-fabc(1))/(x(4)-x(1));
14 disp(fx',fab,fabc,fabcd,'Divided difference columns
    : ')
15
16 //We can redraw the table, the existing entries does
    not change
17 x(5)=1.1;

```

```

18 fx(5)=0.09531;
19 fab(4) = (fx(5)-fx(4))/(x(5)-x(4));
20 fabc(3)= (fab(4)-fab(3))/(x(5)-x(3));
21 fabcd(2)= (fabc(3)-fabc(2))/(x(5)-x(2));
22 fabcde(1)=(fabcd(2)-fabcd(1))/(x(5)-x(1));
23 disp('fx',fab,fabc,fabcd,fabcde,'Divided difference
      columns after addition of an entry : ')

```

Scilab code Exa 5.4 Polynomial Interpolation Divided Differnce form

```

1 //Polynomial Interpolation: Divided Differnce form
2 clc;
3 clear;
4 close();
5 format('v',8);
6 x = [1 1.5 1.75 2];
7 fx = [0 0.40547 0.55962 0.69315];
8 fab(1) = (fx(2)-fx(1))/(x(2)-x(1));
9 fab(2) = (fx(3)-fx(2))/(x(3)-x(2));
10 fab(3) = (fx(4)-fx(3))/(x(4)-x(3));
11 fabc(1)= (fab(2)-fab(1))/(x(3)-x(1));
12 fabc(2)= (fab(3)-fab(2))/(x(4)-x(2));
13 fabcd(1)= (fabc(2)-fabc(1))/(x(4)-x(1));
14
15 x(5)=1.1;
16 fx(5)=0.09531;
17 fab(4) = (fx(5)-fx(4))/(x(5)-x(4));
18 fabc(3)= (fab(4)-fab(3))/(x(5)-x(3));
19 fabcd(2)= (fabc(3)-fabc(2))/(x(5)-x(2));
20 fabcde(1)=(fabcd(2)-fabcd(1))/(x(5)-x(1));
21 disp(fabcde,fabcd,fabc,fab,fx,'Divided difference
      columns : ')
22

```

```

23 x1 = poly(0, 'x1');
24 p3x = fx(1)+fab(1)*(x1-x(1))+fab(1)*(x1-x(1))*(x1-x
      (2))+fabcd(1)*(x1-x(1))*(x1-x(2))*(x1-x(3));
25 p3=horner(p3x,1.3);
26 disp(p3,'The interpolated value at 1.3 using p3(x)
      is : ')
27
28 p4x = p3x + fabcde(1)*(x1-x(1))*(x1-x(2))*(x1-x(3))
      *(x1-x(4));
29 p4=horner(p4x,1.3);
30 disp(p4,'The interpolated value at 1.3 using p4(x)
      is : ')

```

Scilab code Exa 5.5 Construction of Forward Difference Table

```

1 //Construction of Forward Difference Table
2 close();
3 clear;
4 clc;
5 x = poly(0, 'x');
6 fx = (x-1)*(x+5)/((x+2)*(x+1));
7 xi = linspace(0.0,0.8,9);
8 x0 = 0;
9 h = 0.1;
10 format('v',9);
11 // values of function at different xi's
12 fi = horner(fx, xi);
13 // First order difference
14 for j = 1:8
15     delta1_fi(j) = fi(j+1) - fi(j);
16 end
17 // Second order difference
18 for j = 1:7

```

```

19  delta2_fi(j) = delta1_fi(j+1) - delta1_fi(j);
20  end
21  // Third order difference
22  for j = 1:6
23    delta3_fi(j) = delta2_fi(j+1) - delta2_fi(j);
24  end
25  // Fourth order difference
26  for j = 1:5
27    delta4_fi(j) = delta3_fi(j+1) - delta3_fi(j);
28  end
29
30  disp(fi , 'Values of f(x) : ')
31  disp(delta1_fi , 'First Order Difference :')
32  disp(delta2_fi , 'Second Order Difference :')
33  disp(delta3_fi , 'Third Order Difference :')
34  disp(delta4_fi , 'Fourth Order Difference :')

```

Scilab code Exa 5.6 Illustration of Newtons Forward Difference Formula

```

1  //Illustration of Newton's Forward Difference
   Formula
2  close();
3  clear;
4  clc;
5  x = poly(0, 'x');
6  fx = (x-1)*(x+5)/((x+2)*(x+1));
7  xi = linspace(0.0,0.8,9);
8  x0 = 0;
9  h = 0.1;
10 format('v',9);
11 // values of function at different xi's
12 f0 = horner(fx , xi);
13 // First order difference

```

```

14 for j = 1:8
15     delta1_f0(j) = f0(j+1) - f0(j);
16 end
17 // Second order difference
18 for j = 1:7
19     delta2_f0(j) = delta1_f0(j+1) - delta1_f0(j);
20 end
21 // Third order difference
22 for j = 1:6
23     delta3_f0(j) = delta2_f0(j+1) - delta2_f0(j);
24 end
25 // Fourth order difference
26 for j = 1:5
27     delta4_f0(j) = delta3_f0(j+1) - delta3_f0(j);
28 end
29 // Calculating p4(0.12)
30 // x0+s*h=0.12
31 s = (0.12-x0)/h;
32 p4 = f0(1) + delta1_f0(1)*s + delta2_f0(1)*s*(s-1)/
        factorial(2) + delta3_f0(1)*s*(s-1)*(s-2)/
        factorial(3) + delta4_f0(1)*s*(s-1)*(s-2)*(s-3)/
        factorial(4);
33 disp(p4 , 'Value of p4(0.12) ');
34 // exact value of f(0.12) is -1.897574 so error
35 err = p4--1.897574;
36 disp(err , 'Error in estimation ');

```

Scilab code Exa 5.7 Illustration of Central Difference Formula

```

1 // Illustration of Central Difference Formula
2 close();
3 clear;
4 clc;

```

```

5 xi = 0:0.2:1.2;
6 fi = sin(xi);
7 x0 = 0;
8 h = 0.2;
9 format('v',8);
10 // First order difference
11 delta1_fi = diff(fi);
12 // Second order difference
13 delta2_fi = diff(delta1_fi);
14 // Third order difference
15 delta3_fi = diff(delta2_fi);
16 // Fourth order difference
17 delta4_fi = diff(delta3_fi);
18 //Fifth order difference
19 delta5_fi = diff(delta4_fi);
20 //Sixth order difference
21 delta6_fi = diff(delta5_fi);
22 disp(fi , 'Values of f(x) : ');
23 disp(delta1_fi , 'First Order Difference :');
24 disp(delta2_fi , 'Second Order Difference :');
25 disp(delta3_fi , 'Third Order Difference :');
26 disp(delta4_fi , 'Fourth Order Difference :');
27 disp(delta5_fi , 'Fifth Order Difference :');
28 disp(delta6_fi , 'Sixth Order Difference :');
29 //Calculating p2(0.67)
30 xm = 0.6;
31 x = 0.67;
32 s = (x-xm)/0.2;
33 p2 = fi(4) + {s*(delta1_fi(3)+delta1_fi(4))/2} + s*s
      *(delta2_fi(3))/2;
34 disp(p2 , 'Value of p2(0.67) : ');
35 //Calculating p4(0.67)
36 p4 = p2 + s*(s*s-1)*(delta3_fi(3)+delta3_fi(2))/12 +
      s*s*(s*s-1)*delta4_fi(2)/24;
37 disp(p4 , 'Value of p4(0.67) : ');
38 //Exact value of sin(0.67) is 0.62099 so error in
      estimation
39 err = 0.62099-0.62098;

```



```
40 disp(err , 'Error in estimation : ');
```

Scilab code Exa 5.8 Hermite Interpolation

```
1 //Hermite Interpolation
2 clc;
3 clear;
4 close();
5 format('v',9);
6 funcprot(0);
7 deff(' [LL0]=L0(x)', 'LL0= 2*x^2-11*x+15');
8 deff(' [LL1]=L1(x)', 'LL1= -4*x^2+20*x-24');
9 deff(' [LL2]=L2(x)', 'LL2= 2*x^2-9*x+10');
10 deff(' [LL0d]=L0d(x)', 'LL0d= 4*x-11');
11 deff(' [LL1d]=L1d(x)', 'LL1d= -8*x+20');
12 deff(' [LL2d]=L2d(x)', 'LL2d= 4*x-9');
13
14 disp('In this case n = 2. The legranges polynomial
      and their derivative . ');
15 disp('L0(x)=2*x^2-11*x+15  L1(x)= -4*x^2+20x-24  L2(
      x)=2x^2-9x+10');
16 disp('L0d(x)=4*x-11  L1d(x)= -8*x+20  L2d(x)=4*x-9')
      ;
17
18 disp('ri(x) = [1-2(x-xi)Lid(xi)][Li(x)]^2  si(x) =
      (x-xi)[Li(x)]^2');
19
20 deff(' [rr0]=r0(x)', 'rr0=(1-2*(x-2)*L0d(2))*(L0(x))^2
      ');
21 deff(' [rr1]=r1(x)', 'rr1=(1-2*(x-2.5)*L1d(2.5))*(L1(x)
      )^2');
22 deff(' [rr2]=r2(x)', 'rr2=(1-2*(x-3)*L2d(3))*(L2(x))^2
      ');
```

```

23
24 deff ( ' [ ss0]=s0 (x) ', ' ss0=(x-2)*L0(x)^2 ');
25 deff ( ' [ ss1]=s1 (x) ', ' ss1=(x-2.5)*L1(x)^2 ');
26 deff ( ' [ ss2]=s2 (x) ', ' ss2=(x-3)*L2(x)^2 ');
27
28 y = [log(2) log(2.5) log(3)];
29 yd = [0.500000 0.400000 0.333333];
30
31 deff ( ' [H5]=H(x) ', ' H5=r0(x)*y(1)+r1(x)*y(2)+r2(x)*y
    (3)+s0(x)*yd(1)+s1(x)*yd(2)+s2(x)*yd(3) ');
32 y2 = H(2.7);
33 disp(y2, 'The calculated value of y(2.7): ');
34 act = log(2.7);
35 disp(act, 'The exact value is of y(2.7): ');
36 err = act - y2;
37 disp(err, 'The error is : ');

```

Scilab code Exa 5.9 Hermite cubic Interpolation

```

1 //Hermite cubic Interpolation
2 clc;
3 clear;
4 close();
5 format('v',9);
6 funcprot(0);
7
8 x0 = -2;x1 = 0;x2 = 1;
9 y0 = 3;y1 = 1;y2 = -2;
10 y0d = -1;y1d = 0;y1d = 1;
11 h0 = 2;
12 h1 = 1;
13
14 deff ( ' [ H3_0]=H30(x) ', ' H3_0=y0*((x-x1)^2/h0^2+2*(x-x0

```

```

    )*(x-x1)^2/h0^3)+y1*((x-x0)^2/h0^2-2*(x-x1)*(x-x0)
    )^2/h0^3)+y0d*(x-x0)*(x-x1)^2/h0^2+y1d*((x-x1)*(x
    -x0)^2)/h0^2');
15 def ('[H3_1]=H31(x)', 'H3_1=y1*((x-x2)^2/h1^2+2*(x-x1)
    )*(x-x2)^2/h1^3)+y2*((x-x1)^2/h1^2-2*(x-x2)*(x-x1)
    )^2/h1^3)+y1d*(x-x1)*(x-x2)^2/h1^2+y2d*((x-x2)*(x
    -x1)^2)/h1^2');
16
17 disp ('H(x) = x^3/4+x^2+1      on  -2<=x<=0');
18 disp ('          7*x^3-10*x^2+1  on  0<=x<=1');

```

Scilab code Exa 5.10 Illustration cubic spline interpolation with equal difference

```

1 //Illustration cubic spline interpolation with equal
  difference
2 //It needs Symbolic Toolbox
3 clc;
4 clear;
5 close();
6 x = -1:1;
7 fx = x^4;
8 y = fx;
9 function y = myfunction(x)
10 y = x^4;
11 endfunction
12 diff_y = derivative(myfunction, x');
13 diff_y0 = diff_y(1);
14 diff_y2 = diff_y(9);
15 //cd ~/Desktop/maxima_symbolic
16 //exec symbolic.sce
17 syms a0 b0 c0 d0;
18 x = poly(0, 'x');

```

```

19 s0x = a0+b0*x+c0*x^2+d0*x^3;
20 syms a1 b1 c1 d1;
21 s1x = a1+b1*x+c1*x^2+d1*x^3;
22 diff1_s0x = diff(s0x,x);
23 diff2_s0x = diff(diff1_s0x,x);
24 diff1_s1x = diff(s1x,x);
25 diff2_s1x = diff(diff1_s1x,x);
26 //from condition(ii)
27 x = -1;
28 eval(s0x,x);
29 //it gives equation a0-b0+c0-d0=1
30 x=1;
31 eval(s1x,x);
32 //it gives equation a1+b1+c1+d1=1
33 x = 0;
34 eval(s0x,x);
35 //it gives equation a0=0
36 eval(s1x,x);
37 //it gives equation a1=0
38 //from condition(iii)
39 x=0;
40 eval(diff1_s0x,x);
41 eval(diff1_s1x,x);
42 //it gives b0=b1;
43 //from condition(iv)
44 eval(diff2_s0x);
45 eval(diff2_s1x);
46 //it gives 2*c0=2*c1
47 //Applying boundary conditions
48 x=-1;
49 eval(diff1_s0x);
50 //it gives b0-2*c0+3*d0=-4
51 x=1;
52 eval(diff1_s1x);
53 //it gives b1+2*c1+3*d1=4
54 //Matrix form for the equations
55 A=[1 -1 1 -1 0 0 0 0;
56 1 0 0 0 0 0 0 0;

```

```

57 0 0 0 0 1 0 0 0;
58 0 0 0 0 1 1 1 1;
59 0 1 0 0 0 -1 0 0;
60 0 0 1 0 0 0 -1 0;
61 0 1 -2 3 0 0 0 0;
62 0 0 0 0 0 1 2 3];
63 C=[1 0 0 1 0 0 -4 4];
64 B = inv(A)*C';
65 //it implies
66 a0=0;b0=0;c0=-1;d0=-2;a1=0;b1=0;c1=-1;d1=2;
67 //for -1<=x<=0
68 x=poly(0,'x');
69 sx = eval(s0x);
70 disp(sx , 'for -1<=x<=0 sx =' );
71 //for 0<=x<=1
72 sx = eval(s1x);
73 disp(sx , 'for 0<=x<=1 sx =' );

```

Scilab code Exa 5.11 Illustration cubic spline interpolation with unequal difference

```

1 //Illustration cubic spline interpolation with
   unequal difference
2 clc;
3 clear;
4 close();
5 //with free boundary conditions
6 xi = [0 1 3 3.5 5];
7 yi = [1.00000 0.54030 -0.98999 -0.93646 0.28366];
8 n = 4;
9 h0 = xi(2)-xi(1);
10 h1 = xi(3)-xi(2);
11 h2 = xi(4)-xi(3);

```

```

12 h3 = xi(5)-xi(4);
13 //After imposing free boundary conditions the matrix
    we get
14 A = [2 1 0 0 0;
15 1 3 1/2 0 0;
16 0 1/2 5 2 0;
17 0 0 2 16/3 2/3;
18 0 0 0 2/3 4/3];
19 C = [-1.37910 ; -2.52682 ; -0.50536 ; 2.26919 ;
    1.62683] ;
20 format('v',8);
21 B = inv(A)*C;
22 //it gives
23 diff1_y0 = -0.33966;
24 diff1_y1 = -0.69978;
25 diff1_y2 = -0.17566;
26 diff1_y3 = 0.36142;
27 diff1_y4 = 1.03941;
28 //cubic polynomial for 3<=x<=3.5
29 x = poly(0, 'x')
30 s2x = yi(3)*[{(x-3.5)*(x-3.5)/(0.5*0.5)}+{2*(x-3)*(x
    -3.5)*(x-3.5)/(0.5*0.5*0.5)}] + yi(4)*[{(x-3)*(x
    -3)/(0.5*0.5)}-{2*(x-3.5)*(x-3)*(x-3)
    /(0.5*0.5*0.5)}] + diff1_y2*{(x-3)*(x-3.5)*(x
    -3.5)/(0.5*0.5)} + diff1_y3*{(x-3.5)*(x-3)*(x-3)
    /(0.5*0.5)};
31 x = 3.14159;
32 disp(horner(s2x,x) , 'value of s2x at 3.14159 : ');
33 //with clamped boundary conditions
34 diff1_y0 = -sin(0);
35 diff1_y4 = -sin(5);
36 //matrix form
37 A = [3 0.5 0;0.5 5 2 ; 0 2 16/3];
38 C = [-2.52682 ; -0.50536 ; 1.62991];
39 B = inv(A)*C;
40 //it gives
41 diff1_y1 = -0.81446;
42 diff1_y2 = -0.16691;

```

```

43 diff1_y3 = 0.36820;
44 s2x = yi(3)*[{(x-3.5)*(x-3.5)/(0.5*0.5)}+{2*(x-3)*(x
    -3.5)*(x-3.5)/(0.5*0.5*0.5)}] + yi(4)*[{(x-3)*(x
    -3)/(0.5*0.5)}-{2*(x-3.5)*(x-3)*(x-3)
    /(0.5*0.5*0.5)}] + diff1_y2*{(x-3)*(x-3.5)*(x
    -3.5)/(0.5*0.5)} + diff1_y3*{(x-3.5)*(x-3)*(x-3)
    /(0.5*0.5)};
45 x = 3.14159;
46 disp(horner(s2x,x) , 'value of s2x at 3.14159 : ');

```

Scilab code Exa 5.12 Alternating way of constructing cubic splines

```

1 //Alternating way of constructing cubic splines
2 clc;
3 clear;
4 close();
5 //from example 5.11
6 xi = [0 1 3 3.5 5];
7 yi = [1.00000 0.54030 -0.98999 -0.93646 0.28366];
8 //free boundary conditions
9 //matrix form
10 format('v',8);
11 A = [6 2 0; 2 5 1/2; 0 1/2 4];
12 B = 6*[-0.30545 ; 0.87221 ; 0.70635];
13 C = inv(A)*B;
14 c1 = C(1);
15 c2 = C(2);
16 c3 = C(3);
17 x = poly(0, 'x');
18 s2x = c2*(3.5-x)*(3.5-x)*(3.5-x)/(6*0.5) + c3*(x-3)
    *(x-3)*(x-3)/(6*0.5) + {yi(3)/0.5+0.5*c2/6}*(3.5-
    x) + {yi(4)/0.5 + 0.5*c3/6}*(x-3);
19 x = 3.14159;

```

```

20 val = horner(s2x,x)*(-1.00271)/(-0.90705);
21 disp(val , 'value of s2x at 3.14159 : ');
22 //clamped boundary conditions
23 A = [2 1 0 0 0;
24 1 6 2 0 0;
25 0 2 5 1/2 0;
26 0 0 1/2 4 3/2;
27 0 0 0 3/2 3];
28 B = 6*[-0.45970; -0.30545 ; 0.87221 ; 0.70635;
        0.14551];
29 C = inv(A)*B;
30 c0 = C(1);
31 c1 = C(2);
32 c2 = C(3);
33 c3 = C(4);
34 c4 = C(5);
35 s2x = c2*(3.5-x)*(3.5-x)*(3.5-x)/(6*0.5) + c3*(x-3)
        *(x-3)*(x-3)/(6*0.5) + {yi(3)/0.5+0.5*c2/6}*(3.5-
        x) + {yi(4)/0.5 + 0.5*c3/6}*(x-3);
36 x = 3.14159;
37 val = horner(s2x,x)*(-1.00227)/(-0.91030);
38 disp(val , 'value of s2x at 3.14159 : ');

```

Scilab code Exa 5.13 Linear Least square approximation method

```

1 //Linear Least square approximation method
2 clc;
3 clear;
4 close();
5 xi = [-5 -3 1 3 4 6 8];
6 yi = [18 7 0 7 16 50 67];
7 wi = [1 1 1 1 20 1 1];
8 format('v',7);

```



```

9 //Representation of equation in matrix form
10 W = [sum(wi) sum(wi.*xi); sum(wi.*xi) sum(wi.*xi.*xi
    )];
11 Y = [sum(wi.*yi); sum(wi.*yi.*xi)];
12 A = inv(W)*Y;
13 a0 = A(1);
14 a1 = A(2);
15 x = poly(0, 'x');
16 p1x = a1*x + a0;
17 disp(p1x, 'The approximating polynomial is :');
18 x = linspace(-5,8,1000);
19 p1x = a1*x + a0;
20 subplot(2,1,1);
21 plot(x,p1x);
22 plot(xi,yi, 'o');
23
24 wi = [1 1 1 1 1 1 1];
25 //Representation of equation in matrix form
26 W = [sum(wi) sum(wi.*xi); sum(wi.*xi) sum(wi.*xi.*xi
    )];
27 Y = [sum(wi.*yi); sum(wi.*yi.*xi)];
28 A = inv(W)*Y;
29 a0 = A(1);
30 a1 = A(2);
31 x = poly(0, 'x');
32 p1x = a1*x + a0;
33 disp(p1x, 'The approximating polynomial is :')
34 x = linspace(-5,8,1000);
35 p1x = a1*x + a0;
36 subplot(2,1,2);
37 plot(x,p1x);
38 plot(xi,yi, 'o');

```

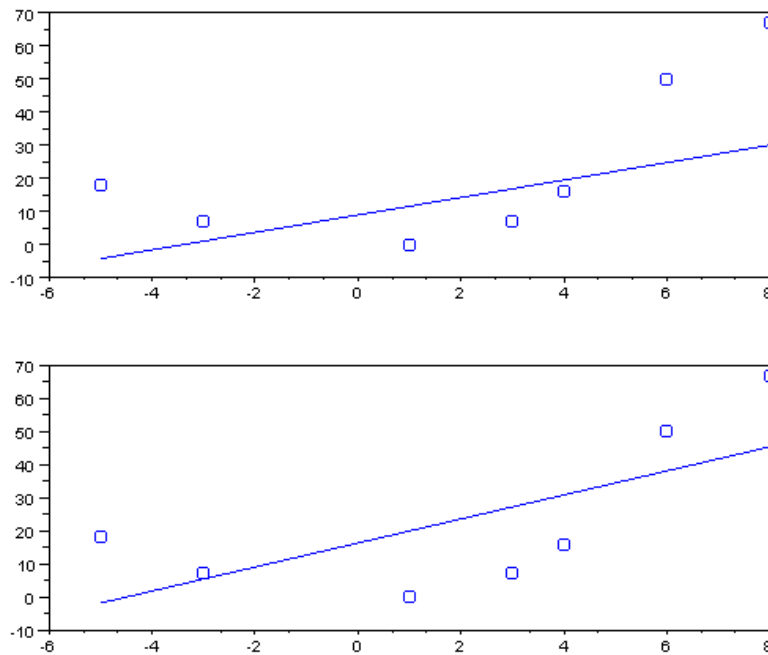


Figure 5.1: Linear Least square approximation method

Scilab code Exa 5.14 Quadratic Least square approximation method

```
1 //Quadratic Least square approximation method
2 clc;
3 clear;
4 close();
5 xi = [-5 -3 1 3 4 6 8];
6 yi = [18 7 0 7 16 50 67];
7 wi = [1 1 1 1 20 1 1];
8 format('v',7);
9 //Representation of equation in matrix form
10 W = [sum(wi) sum(wi.*xi) sum(wi.*xi.*xi); sum(wi.*xi
      ) sum(wi.*xi.*xi) sum(wi.*xi.*xi.*xi); sum(wi.*xi
      .*xi) sum(wi.*xi.*xi.*xi) sum(wi.*xi.*xi.*xi.*xi)
      ];
11 Y = [sum(wi.*yi); sum(wi.*yi.*xi); sum(wi.*xi.*xi.*
      yi)];
12 A = inv(W)*Y;
13 a0 = A(1);
14 a1 = A(2);
15 a2 = A(3);
16 x = poly(0, 'x');
17 p1x = a2*x^2 + a1*x + a0;
18 disp(p1x, 'The approximating polynomial is :');
19 x = linspace(-5,8,1000);
20 p1x = a2*x^2 + a1*x + a0;
21 plot(x,p1x);
22 plot(xi,yi, 'o');
```

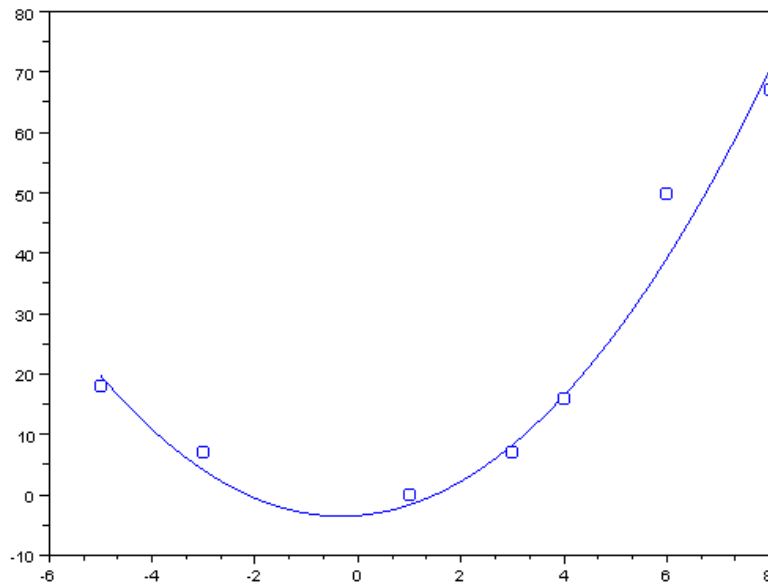


Figure 5.2: Quadratic Least square approximation method

Scilab code Exa 5.15 Least square approximation method with exponential functions

```

1 //Least square approximation method with exponential
  functions
2 clc;
3 clear;
4 close();
5 xi = [0 0.25 0.4 0.5];
6 yi = [9.532 7.983 4.826 5.503];
7 wi = ones(1,4);
8 //data corresponding to linearised problem
9 Xi = [0 0.25 0.4 0.5];
10 Yi = [2.255 2.077 1.574 1.705];
11 wi = ones(1,4);
12 format('v',6);
13 //Representation of equation in matrix form
14 W = [sum(wi) sum(wi.*xi); sum(wi.*xi) sum(wi.*xi.*xi
    )];
15 Y = [sum(wi.*Yi); sum(wi.*Yi.*Xi)];
16 C = inv(W)*Y;
17 A = C(1);
18 B = C(2);
19 a = exp(2.281);
20 b = B;
21 disp(a, 'a = ');
22 disp(b, 'b = ');
23 //So the non linear system becomes
24 disp('9.532 - a + 7.983*exp(0.25*b) - a*exp(0.5*b) + 4.826*
    exp(0.4*b) - a*exp(0.8*b) + 5.503*exp(0.5*b) - a*exp(b)
    = 0');
25 disp('1.996*a*exp(0.25*b) - 0.25*a*a*exp(0.5*b) + 1.930*

```

```

    a*exp(0.4*b) - 0.4*a*a*exp(0.8*b) + 2.752*a*exp(0.5*b
    ) - 0.5*a*a*exp(b) = 0');
26 //Applying Newtons Method we get
27 a = 9.731;
28 b = -1.265;
29 disp(a , 'a = ');
30 disp(b , ' b = ');

```

Scilab code Exa 5.16 Least square approximation to continuous functions

```

1 //Least square approximation to continuous functions
2 clc;
3 clear;
4 close();
5 format('v',8);
6 funcprot(0);
7 deff(' [g]=f(x,y)', 'g= -y^2/(1+x)');
8 disp('approximation of e^x on [0,1] with a uniform
    weight w(x)=1')
9 a11 = integrate('1', 'x', 0, 1);
10 a12 = integrate('x', 'x', 0, 1);
11 a13 = integrate('x*x', 'x', 0, 1);
12 a14 = integrate('x^3', 'x', 0, 1);
13 a21 = integrate('x', 'x', 0, 1);
14 a22 = integrate('x^2', 'x', 0, 1);
15 a23 = integrate('x^3', 'x', 0, 1);
16 a24 = integrate('x^4', 'x', 0, 1);
17 a31 = integrate('x^2', 'x', 0, 1);
18 a32 = integrate('x^3', 'x', 0, 1);
19 a33 = integrate('x^4', 'x', 0, 1);
20 a34 = integrate('x^5', 'x', 0, 1);
21 a41 = integrate('x^3', 'x', 0, 1);
22 a42 = integrate('x^4', 'x', 0, 1);

```

```

23 a43 = integrate('x^5','x',0,1);
24 a44 = integrate('x^6','x',0,1);
25
26 c1 = integrate('exp(x)','x',0,1);
27 c2 = integrate('x*exp(x)','x',0,1);
28 c3 = integrate('x^2*exp(x)','x',0,1);
29 c4 = integrate('x^3*exp(x)','x',0,1);
30
31 A = [a11 a12 a13 a14;a21 a22 a23 a24;a31 a32 a33 a34
      ;a41 a42 a43 a44];
32 C = [c1;c2;c3;c4];
33 ann = inv(A)*C;
34 disp(ann, 'The coefficients a0,a1,a2,a3 are
      respectively : ');
35
36 deff('[px]=p3(x)', 'px=ann(4)*x^3+ann(3)*x^2+ann(2)*x
      +ann(1)');
37 x = [0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0]';
38 e = exp(x);
39 p = p3(x);
40 err = e-p;
41 ann = [x e p err];
42
43 disp(ann, 'Displaying the value of x exp(x) p3(x) exp
      (x)-p3(x) :');
44 plot(x,err);
45 plot(x,0);

```

Scilab code Exa 5.17 Gram Schmidt process for finding orthogonal functions

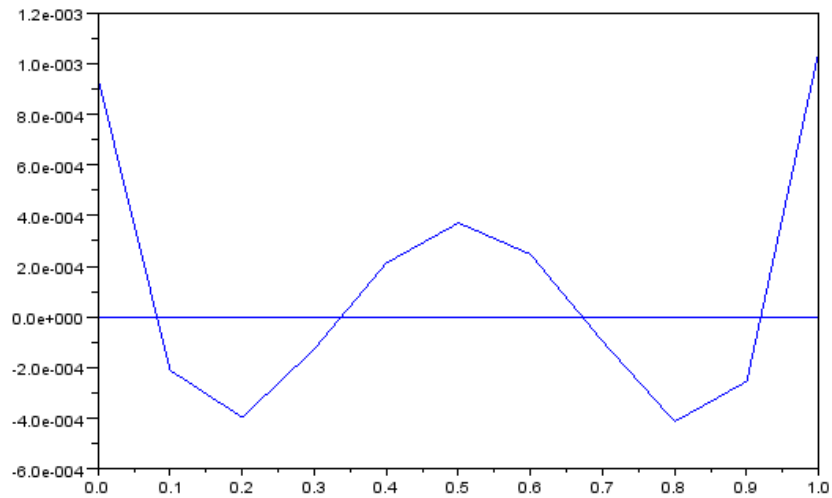


Figure 5.3: Least square approximation to continuous functions

```

1 //Gram – Schmidt process for finding orthogonal
  functions
2 clc;
3 clear;
4 close();
5 format('v',8);
6 funcprot(0);
7
8 disp('The orthogonal functions : ')
9 x = poly(0,'x');
10 ph0 = 1;
11
12 disp(ph0 , 'phi0(x) = ');
13 K1_0 = -integrate('x','x',0,1)/integrate('ph0^2','x',
  ,0,1);
14 ph1 = x + K1_0*ph0;
15 disp(ph1 , 'phi1(x) = ');
16
17 K2_0 = -integrate('x^2*ph0','x',0,1)/integrate('ph0

```



```

    ^2', 'x', 0, 1);
18 disp(K2_0 , 'K(2,0) = ');
19 K2_1 = -integrate('x^2*(x-.5)', 'x', 0, 1)/integrate('(
    x-.5)^2', 'x', 0, 1);
20 disp(K2_1 , 'K(2,1) = ');
21 ph2 = x^2 + K2_0*ph0 + K2_1*ph1;
22 disp(ph2 , 'phi2(x) = ');
23
24 K3_0 = -integrate('x^3*ph0', 'x', 0, 1)/integrate('ph0
    ^2', 'x', 0, 1);
25 disp(K3_0 , 'K(3,0) = ');
26 K3_1 = -integrate('x^3*(x-.5)', 'x', 0, 1)/integrate('(
    x-.5)^2', 'x', 0, 1);
27 disp(K3_1 , 'K(3,1) = ');
28 K3_2 = -integrate('x^3*(x^2-x+1/6)', 'x', 0, 1)/
    integrate('(x^2-x+1/6)^2', 'x', 0, 1);
29 disp(K3_2 , 'K(3,2) = ');
30 ph3 = x^3 + K3_0*ph0 + K3_1*ph1 + K3_2*ph2;
31 disp(ph3 , 'phi3(x) = ');

```

Scilab code Exa 5.18 Gram Schmidt process for cubic polynomial least squares approx

```

1 //Gram - Schmidt process for cubic polynomial least
    squares approx
2 clc;
3 clear;
4 close();
5 format('v', 8);
6 funcprot(0);
7
8 disp('The orthogonal functions : ')
9 x = poly(0, 'x');

```

```

10 ph0 = 1;
11
12 disp(ph0 , 'phi0(x) = ');
13 K1_0 = -integrate('x', 'x', 0, 1) / integrate('ph0^2', 'x',
    , 0, 1);
14 ph1 = x + K1_0*ph0;
15 disp(ph1 , 'phi1(x) = ');
16
17 K2_0 = -integrate('x^2*ph0', 'x', 0, 1) / integrate('ph0
    ^2', 'x', 0, 1);
18 disp(K2_0 , 'K(2,0) = ');
19 K2_1 = -integrate('x^2*(x-.5)', 'x', 0, 1) / integrate('(
    x-.5)^2', 'x', 0, 1);
20 disp(K2_1 , 'K(2,1) = ');
21 ph2 = x^2 + K2_0*ph0 + K2_1*ph1;
22 disp(ph2 , 'phi2(x) = ');
23
24 K3_0 = -integrate('x^3*ph0', 'x', 0, 1) / integrate('ph0
    ^2', 'x', 0, 1);
25 disp(K3_0 , 'K(3,0) = ');
26 K3_1 = -integrate('x^3*(x-.5)', 'x', 0, 1) / integrate('(
    x-.5)^2', 'x', 0, 1);
27 disp(K3_1 , 'K(3,1) = ');
28 K3_2 = -integrate('x^3*(x^2-x+1/6)', 'x', 0, 1) /
    integrate('(x^2-x+1/6)^2', 'x', 0, 1);
29 disp(K3_2 , 'K(3,2) = ');
30 ph3 = x^3 + K3_0*ph0 + K3_1*ph1 + K3_2*ph2;
31 disp(ph3 , 'phi3(x) = ');
32
33 deff(' [y]=f(x)', 'y= exp(x)');
34 deff(' [phi0]=ph_0(x)', 'phi0= horner(ph0, x)');
35 deff(' [phi1]=ph_1(x)', 'phi1= horner(ph1, x)');
36 deff(' [phi2]=ph_2(x)', 'phi2= horner(ph2, x)');
37 deff(' [phi3]=ph_3(x)', 'phi3= horner(ph3, x)');
38 a0 = integrate('f(x)*ph_0(x)', 'x', 0, 1) / integrate('
    ph_0(x)^2', 'x', 0, 1);
39 disp(a0, 'a0 = ');
40 a1 = integrate('f(x)*ph_1(x)', 'x', 0, 1) / integrate('

```

```
    ph_1(x)^2', 'x', 0, 1);
41 disp(a1, 'a1 = ');
42 a2 = integrate('f(x)*ph_2(x)', 'x', 0, 1)/integrate('
    ph_2(x)^2', 'x', 0, 1);
43 disp(a2, 'a2 = ');
44 a3 = integrate('f(x)*ph_3(x)', 'x', 0, 1)/integrate('
    ph_3(x)^2', 'x', 0, 1);
45 disp(a3, 'a3 = ');
46
47 p3 = a0*ph0 + a1*ph1 + a2*ph2 + a3*ph3;
48 disp(p3 , 'p3(x)');
```

Chapter 6

Numerical Differentiation and Integration

Scilab code Exa 6.1 Numerical Differentiation

```
1 //Numerical Differentiation
2 clc;
3 clear;
4 close();
5 format('v',9);
6 deff(' [y]=f(x) ', 'y=exp(-x) ');
7
8 x0 = ones(:,8);
9 h = [1 .2 .1 .02 .01 .002 .001 .0002];
10 x1 = 1+h;
11 f0 = f(x0);
12 f1 = f(x1);
13 dif = (f1-f0)./h;
14 max_trun_err = exp(-1).*h/2;
15 act_err = abs(-exp(-1)-dif);
16 answer = [h' f0' f1' dif' max_trun_err' act_err'];
17 disp(answer, '      h          f0          f1          f1
      -f0/h      he^-1          | Actual Error | ');
18 x = (0:.0002:3);
```

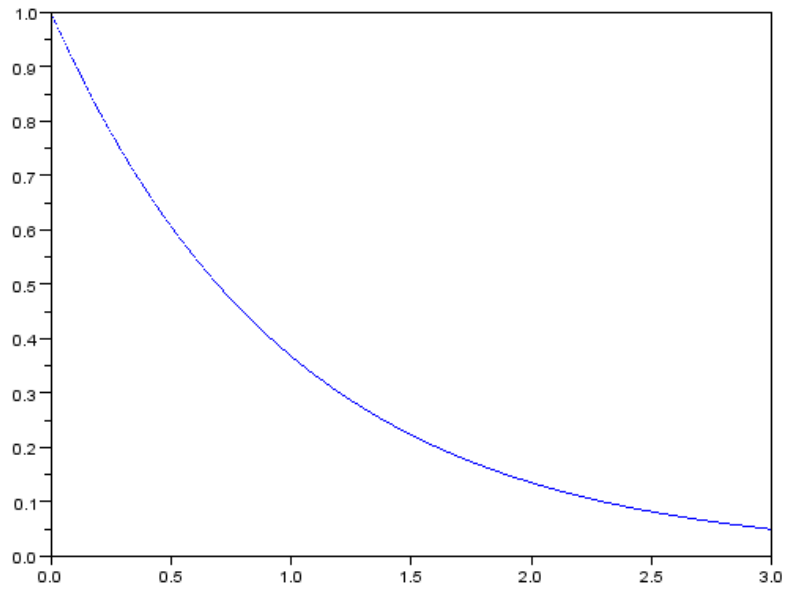


Figure 6.1: Numerical Differentiation

```
19 plot(x, f(x));
```

Scilab code Exa 6.2 Numerical Differentiation

```
1 //Numerical Differentiation
2 clc;
3 clear;
4 close();
5 format('v',9);
```

```

6 deff(' [y]=f(x) ', 'y=exp(-x) ');
7 h = [1 .2 .1 .02 .01 .002 .001 .0002];
8 x0 = 1 - h;
9 x1 = ones(:,8);
10 x2 = 1+h;
11 f0 = f(x0);
12 f1 = f(x1);
13 f2 = f(x2);
14 dif = (f2-f0)/(2*h);
15 max_trun_err = exp(h-1).*h^2/6;
16 act_err = abs(-exp(-1)-dif);
17 answer = [h' f0' f2' dif' max_trun_err' act_err'];
18 disp(answer, '      h      f0      f2      f2-
      f0/2h  h^2*exp(h-1)/6 |Actual Error| ');
19 disp('truncation error does not exceed h^2*exp(h-1)
      /6 ');
20 x = (0:.0002:3);
21 plot(x,f(x));

```

Scilab code Exa 6.3 Numerical Integration

```

1 //Numerical Integration
2 clc;
3 clear;
4 close();
5 format('v',9);
6 funcprot(0);
7 deff(' [y]=f(x) ', 'y=x*cos(x) ');
8
9 rec = %pi * f(0)/4;
10 disp(rec, 'Retangular Rule : ');

```

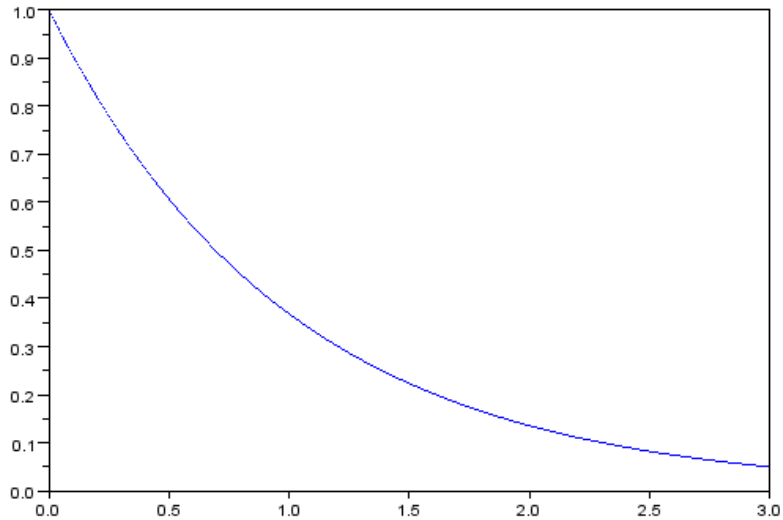


Figure 6.2: Numerical Differentiation

```

11
12 trap = %pi*(f(0)+f(%pi/4))/8;
13 disp(trap, 'Trapezoidal Rule : ');
14
15 sip = %pi*(f(0)+4*f(%pi/8)+f(%pi/4))/(3*8);
16 disp(sip, 'Simpson''s Rule : ');
17
18 sip38 = %pi*3*(f(0)+3*f(%pi/12)+3*f(%pi/6)+f(%pi/4))
          /(12*8);
19 disp(sip38, 'Simpson''s 3/8 Rule : ');
20
21 exact = integrate('x*cos(x)', 'x', 0, %pi/4);
22 disp(exact, 'The exact value of intergation is :');
23 err = exact - rec;
24 err(2) = exact - trap;
25 err(3) = exact - sip;
26 err(4) = exact - sip38;
27 disp(err, 'thus corresponding errors are : ');

```

Scilab code Exa 6.4 Numerical Integration

```
1 //Newton Cotes formula
2 clc;
3 clear;
4 close();
5 format('v',9);
6 funcprot(0);
7 disp('Integral 0 to PI/4 x*cos dx');
8 disp('based on open Newton-Cotes formulas ');
9
10 deff('[y]=f(x)', 'y=x*cos(x)');
11
12 k = [0 1 2 3]
13
14 a = 0;
15 b = %pi/4;
16 h = (ones(:,4)*(b-a))./(k+2);
17 x0 = a+h;
18 xk = b-h;
19
20 k(1) = 2*h(1)*f(h(1));
21 disp(k(1), 'k=0');
22
23 k(2) = 3*h(2)*(f(h(2))+f(2*h(2)))/2;
24 disp(k(2), 'k=1');
25
26 k(3) = 4*h(3)*(2*f(h(3))-f(2*h(3))+2*f(3*h(3)))/3;
27 disp(k(3), 'k=2');
28
29 k(4) = 5*h(4)*(11*f(h(4))+f(2*h(4))+f(3*h(4))+11*f
    (4*h(4)))/24;
```



```

30 disp(k(4), 'k=3');
31
32 exact = integrate('x*cos(x)', 'x', 0, %pi/4);
33 disp(exact, 'The exact value of intergation is :');
34 exact = ones(:,4)*exact;
35 err = exact-k;
36 disp(err, 'thus corresponding errors are : ');

```

Scilab code Exa 6.5 Trapezoidal Rule

```

1 //Trapezoidal Rule
2 clc;
3 clear;
4 close();
5 format('v',10);
6 funcprot(0);
7 disp('Integral 0 to 2 e^x dx');
8 disp('based on trapezoidal rule ');
9
10 deff('[y]=f(x)', 'y=exp(x)');
11
12 n = [1 2 4 8];
13
14 a = 0;
15 b = 2;
16 h = (ones(:,4)*(b-a))./n;
17
18 t(1) = h(1)*(f(a)+f(b))/2;
19 disp(t(1), 'n=1');
20
21 t(2) = h(2)*(f(a)+f(b)+2*f(h(2)))/2;
22 disp(t(2), 'n=2');
23

```

```

24 t(3) = h(3)*(f(a)+f(b)+2*(f(h(3))+f(2*h(3))+f(3*h(3)
    ))) / 2;
25 disp(t(3), 'n=4');
26
27 t(4) = h(4)*(f(a)+f(b)+2*(f(h(4))+f(2*h(4))+f(3*h(4)
    )+f(4*h(4))+f(5*h(4))+f(6*h(4))+f(7*h(4)))) / 2;
28 disp(t(4), 'n=8');
29
30 exact = integrate('exp(x)', 'x', 0, 2);
31 disp(exact, 'The exact value of intergration is :');
32 exact = ones(4)*exact;
33 err = exact-t;
34 disp(err, 'thus corresponding errors are : ');

```

Scilab code Exa 6.6 Simpson Rule

```

1 //Simpson Rule
2 clc;
3 clear;
4 close();
5 format('v', 10);
6 funcprot(0);
7
8 deff(' [y]=f(x)', 'y=exp(x)');
9
10 n = [1 2 4];
11
12 a = 0;
13 b = 2;
14 h = (ones(:, 3)*(b-a)) ./ (2*n);
15
16 s(1) = h(1)*(f(a)+f(b)+4*f(h(1))) / 3;
17 disp(s(1), 'n=1');

```

```

18
19 s(2) = h(2)*(f(a)+f(b)+2*f(2*h(2))+4*(f(h(2))+f(3*h
    (2))))/3;
20 disp(s(2), 'n=2');
21
22 s(3) = h(3)*(f(a)+f(b)+2*(f(2*h(3))+f(4*h(3))+f(6*h
    (3)))+4*(f(h(3))+f(3*h(3))+f(5*h(3))+f(7*h(3))))
    /3;
23 disp(s(3), 'n=4');
24
25 exact = integrate('exp(x)', 'x', 0, 2);
26 disp(exact, 'The exact value of intergation is :');
27 exact = ones(3)*exact;
28 err = exact-s;
29 disp(err, 'thus corresponding errors are : ');

```

Scilab code Exa 6.7 Rombergs Interpolation

```

1 //Romberg's Interpolation
2 clc;
3 clear;
4 close();
5 exec('C:\Users\Pragya\Desktop\scilab\trap.sci', -1);
6 format('v', 10);
7 funcprot(0);
8 deff('[y]=f(x)', 'y=exp(x)');
9 a = 0;
10 b = 2;
11
12 t(1,1)=trap(f, a, b, 0, 0);
13 disp(t(1,1), 'T(0,0) : ');
14
15 t(2,1)=trap(f, a, b, 1, 0);

```

```

16 disp(t(2,1), 'T(1,0) : ');
17
18 t(3,1)=trap(f,a,b,2,0);
19 disp(t(3,1), 'T(2,0) : ');
20
21 t(4,1)=trap(f,a,b,3,0);
22 disp(t(4,1), 'T(3,0) : ');
23
24 t(2,2)=trap(f,a,b,1,1);
25 disp(t(2,2), 'T(1,1) : ');
26
27 t(3,2)=trap(f,a,b,2,1);
28 disp(t(3,2), 'T(2,1) : ');
29
30 t(4,2)=trap(f,a,b,3,1);
31 disp(t(4,2), 'T(3,1) : ');
32
33 t(3,3)=trap(f,a,b,2,2);
34 disp(t(3,3), 'T(2,2) : ');
35
36 t(4,3)=trap(f,a,b,3,2);
37 disp(t(4,3), 'T(3,2) : ');
38
39 t(4,4)=trap(f,a,b,3,3);
40 disp(t(4,4), 'T(3,3) : ');
41
42 disp(t, 'The corresponding Romberg Table is : ');

```

Scilab code Exa 6.8 Rombergs Method

```

1 //Romberg's Method
2 clc;
3 clear;

```

```

4 close();
5 exec('C:\Users\Pragya\Desktop\scilab\trap.sci', -1);
6 format('v',10);
7 funcprot(0);
8 deff('[y]=f(x)', 'y=exp(x)');
9 a = 0;
10 b = 2;
11
12 t(1,1)=trap(f,a,b,0,0);
13 disp(t(1,1), 'T(0,0) : ');
14
15 t(2,1)=(t(1,1)+2*1*f(1))/2;
16 disp(t(2,1), 'T(1,0) : ');
17
18 t(3,1)=(t(2,1)+f(1/2)+f(3/2))/2;
19 disp(t(3,1), 'T(2,0) : ');
20
21 t(4,1)=(t(3,1)+.5*(f(1/4)+f(3/4)+f(5/4)+f(7/4)))/2;
22 disp(t(4,1), 'T(3,0) : ');

```

Scilab code Exa 6.9 Simpsons Adaptive Quatrature

```

1 //Simpson's Adaptive Quatrature
2 clc;
3 clear;
4 close();
5 format('v',10);
6 funcprot(0);
7 deff('[y]=f(x)', 'y=exp(x)');
8 a = 0.5;
9 b = 1;
10 h = (b-a)/2;
11 S1 = h*(f(a)+4*f((a+b)/2)+f(b))/3;

```

```

12 disp(S1, 'S1 : ');
13
14 S2 = h*(f(a)+4*f((3*a+b)/4)+2*f((a+b)/2)+4*f((a+3*b)
    /4)+f(b))/6;
15 disp(S2, 'S2 : ');
16
17 err = abs(S2-S1)/15;
18 disp(err, 'An estimate of the error in S2 is : ');
19
20 act = integrate('exp(x)', 'x', .5, 1)
21 act_err = abs(act-S2);
22 disp(act_err, 'The Actual error in S2 is : ');

```

Scilab code Exa 6.10 Simpsons Adaptive Quatrature

```

1 //Simpson's Adaptive Quatrature
2 clc;
3 clear;
4 close();
5 format('v', 7);
6 funcprot(0);
7 deff('[y]=f(x)', 'y=exp(-3*x)*sin(3*x)');
8 e = 0.0005;
9 a = 0;
10 b = %pi;
11 h = (b-a)/2;
12
13 S1 = h*(f(a)+4*f((a+b)/2)+f(b))/3;
14 disp(S1, 'S1 : ');
15 S2 = h*(f(a)+4*f((3*a+b)/4)+2*f((a+b)/2)+4*f((a+3*b)
    /4)+f(b))/6;
16 disp(S2, 'S2 : ');
17

```

```

18 err = abs(S2-S1)/15;
19 disp(err, '|S2-S1|>15e so [0.%pi] must be subdivided
    ');
20
21 a = (a+b)/2;
22 h = (b-a)/2;
23 S1 = h*(f(a)+4*f((a+b)/2)+f(b))/3;
24 disp(S1, 'S1 : ');
25 S2 = h*(f(a)+4*f((3*a+b)/4)+2*f((a+b)/2)+4*f((a+3*b)
    /4)+f(b))/6;
26 disp(S2, 'S2 : ');
27 s = S2;
28 disp(abs(S2-S1), '|S2-S1|<15e/2 ');
29
30 b = a;
31 a = 0;
32 h = (b-a)/2;
33
34 S1 = h*(f(a)+4*f((a+b)/2)+f(b))/3;
35 disp(S1, 'S1 : ');
36 S2 = h*(f(a)+4*f((3*a+b)/4)+2*f((a+b)/2)+4*f((a+3*b)
    /4)+f(b))/6;
37 disp(S2, 'S2 : ');
38
39 err = abs(S2-S1)/15;
40 disp(err, '|S2-S1|>15e so interval must be subdivided
    ');
41
42 a = (a+b)/2;
43 h = (b-a)/2;
44 S1 = h*(f(a)+4*f((a+b)/2)+f(b))/3;
45 disp(S1, 'S1 : ');
46 S2 = h*(f(a)+4*f((3*a+b)/4)+2*f((a+b)/2)+4*f((a+3*b)
    /4)+f(b))/6;
47 disp(S2, 'S2 : ');
48 s = s+S2;
49 disp(abs(S2-S1), '|S2-S1|<15e/4 ');
50

```

```

51 b = a;
52 a = 0;
53 h = (b-a)/2;
54
55 S1 = h*(f(a)+4*f((a+b)/2)+f(b))/3;
56 disp(S1, 'S1 : ');
57 S2 = h*(f(a)+4*f((3*a+b)/4)+2*f((a+b)/2)+4*f((a+3*b)
    /4)+f(b))/6;
58 disp(S2, 'S2 : ');
59
60 err = abs(S2-S1)/15;
61 disp(err, '|S2-S1|>15e so interval must be subdivided
    ');
62
63 a = (a+b)/2;
64 h = (b-a)/2;
65 S1 = h*(f(a)+4*f((a+b)/2)+f(b))/3;
66 disp(S1, 'S1 : ');
67 S2 = h*(f(a)+4*f((3*a+b)/4)+2*f((a+b)/2)+4*f((a+3*b)
    /4)+f(b))/6;
68 disp(S2, 'S2 : ');
69 s = s+S2;
70 disp(abs(S2-S1), '|S2-S1|<15e/8 ');
71
72 b = a;
73 a = 0;
74 h = (b-a)/2;
75
76 S1 = h*(f(a)+4*f((a+b)/2)+f(b))/3;
77 disp(S1, 'S1 : ');
78 S2 = h*(f(a)+4*f((3*a+b)/4)+2*f((a+b)/2)+4*f((a+3*b)
    /4)+f(b))/6;
79 disp(S2, 'S2 : ');
80 disp(abs(S2-S1), '|S2-S1|<15e/8 ');
81 s = s+S2;
82 disp(s);

```

Scilab code Exa 6.11 Gaussian Quadrature Rule

```
1 //Gaussian Quadrature Rule
2 clc;
3 clear;
4 close();
5 format('v',10);
6 funcprot(0);
7 disp('Integral 0 to 1 f(x)dx');
8 b = 1;
9 a = 0;
10 x = poly(0, 'x');
11 p = x^2-x+1/6;
12 x1 = roots(p);
13 A = [1 1;x1'];
14 //X = [c0;c1];
15 B = [(b-a);(b^2-a^2)/2];
16 X = inv(A)*B;
17 disp(X,'Are the c1,c2 constants : ');
18 disp(x1,'Are the corresponding roots (x1,x2) : ');
19 disp('c0*f(x0)+c1*f(x1)');
```

Scilab code Exa 6.12 Gaussian Quadrature Rule

```
1 //Gaussian Quadrature Rule
2 clc;
3 clear;
4 close();
```

```

5 format('v',10);
6 funcprot(0);
7 disp('Integral 0 to 2 exp(x)dx');
8 deff('[y]=f(t)', 'y=exp(t+1)');
9 b = 1;
10 a = -1;
11 x = poly(0, 'x');
12 p = x^4 - 6*x^2/7+3/35;
13 x1 = roots(p);
14 A = [1 1 1 1;x1';(x1.^2)';(x1.^3)'];
15 B = [(b-a);(b^2-a^2)/2;(b^3-a^3)/3;(b^4-a^4)/4];
16 C = inv(A)*B;
17 I = C(1)*f(x1(1))+C(2)*f(x1(2))+C(3)*f(x1(3))+C(4)*f
    (x1(4));
18 disp(I, 'Calculated integration : ');
19 exact = integrate('exp(x)', 'x', 0, 2);
20 disp(exact, 'The exact value of intergation is :');
21 err = exact - I ;
22 disp(err, 'Error : ');

```

Chapter 7

Ordinary Differential Equations Initial value problem

Scilab code Exa 7.1 Eulers Method

```
1 //Euler 's Method
2 clc;
3 clear;
4 close();
5 format('v',8);
6 funcprot(0);
7 deff(' [g]=f(x,y)', 'g= -y^2/(1+x)');
8 y = 1;
9 x = 0;
10 h = 0.05;
11 while x<0.2
12     y = y - 0.05*y^2/(1+x);
13     x = x + h;
14     disp(y,x,'Value of y at x :');
15 end
16 disp(y,'The calculated value of y(0.2):');
17 x = 0.2;
18 act = 1/(1+log(1+x));
19 disp(act,'The exact value is of y(0.2):');
```

```
20 err = act - y;
21 disp(err, 'The error is :');
```

Scilab code Exa 7.2 Eulers trapezoidal predictor corrector pair

```
1 //Euler's trapezoidal predictor-corrector pair
2 clc;
3 clear;
4 close();
5 format('v',8);
6 funcprot(0);
7 deff('[g]=f(x,y)', 'g= -y^2/(1+x)');
8 y = 1;
9 x = 0;
10 h = 0.05;
11 i=0;
12 while x<0.2
13     y0 = y - 0.05*y^2/(1+x);
14     disp(y0, 'The Y0 :')
15     y1 = y - h*(y^2/(1+x)+y0^2/(1+x+h))/2;
16     disp(y1, 'The Y1 :')
17     y2 = y - h*(y^2/(1+x)+y1^2/(1+x+h))/2;
18     disp(y2, 'The Y2 :')
19     y = y2;
20     x = x + h;
21 end
22 disp(y2, 'The calculated value of y(0.2):');
23 x = 0.2;
24 act = 1/(1+log(1+x));
25 disp(act, 'The exact value is of y(0.2): ');
26 err = act - y2;
27 disp(err, 'The error is :');
```

Scilab code Exa 7.3 Mid point formula

```
1 //Mid-point formula
2 clc;
3 clear;
4 close();
5 format('v',8);
6 funcprot(0);
7 deff('[g]=f(x,y)', 'g= -y^2/(1+x)');
8 y0 = 1;
9 y1 = 0.95335;
10 x = 0.05;
11 h = 0.05;
12 i=0;
13 while x<0.2
14     y2 = y0 - 0.1*y1^2/(1+x);
15     disp(y2, 'The Y :')
16     y0 = y1;
17     y1 = y2;
18     x = x + h;
19 end
20 disp(y2, 'The calculated value of y(0.2):');
21 x = 0.2;
22 act = 1/(1+log(1+x));
23 disp(act, 'The exact value is of y(0.2): ');
24 err = act - y2;
25 disp(err, 'The error is :');
```

Scilab code Exa 7.4 Illustration of Taylor Series for approximation

```
1 //Illustration of Taylor Series for approximation
2 //It needs symbolic toolbox
3 clc;
4 clear;
5 close();
6 cd ~/Desktop/maxima_symbolic;
7 exec symbolic.sce
8 y0 = 1;
9 x0 = 0;
10 y1_0 = -y0^2/(1+x0);
11 y2_0 = (2*y0^3+y0^2)/((1+x0)^2);
12 y3_0 = -(6*y0^4 + 6*y0^3 + 2*y0^2)/((1+x0)^3);
13 //similarly
14 y4_0 = 88;
15 y5_0 = -694;
16 y6_0 = 6578;
17 y7_0 = -72792;
18 syms r h;
19 format('v',10);
20 yxr = 1 - r*h + (y2_0*(r*h)^2)/factorial(2) - (y3_0
    *(r*h)^3)/factorial(3) + (y4_0*(r*h)^4)/factorial
    (4) - (y5_0*(r*h)^5)/factorial(5) +(y6_0*(r*h)^6)
    /factorial(6) - (y7_0*(r*h)^7)/factorial(7);
21 yxr_5d = 1 - r*h + (y2_0*(r*h)^2)/factorial(2) + (
    y3_0*(r*h)^3)/factorial(3) + (y4_0*(r*h)^4)/
    factorial(4);
22 h = 0.05;
23 r = 1;
24 yx1 = eval(yxr_5d);
25 format('v',8);
26 disp(dbl(yx1), 'Value when r = 1 :');
27
28 syms r h;
29 format('v',10);
30 yxr = 1 - r*h + (y2_0*(r*h)^2)/factorial(2) - (y3_0
    *(r*h)^3)/factorial(3) + (y4_0*(r*h)^4)/factorial
```

```

        (4) - (y5_0*(r*h)^5)/factorial(5) +(y6_0*(r*h)^6)
        /factorial(6) - (y7_0*(r*h)^7)/factorial(7);
31 yxr_5d = 1 - r*h + (y2_0*(r*h)^2)/factorial(2) + (
        y3_0*(r*h)^3)/factorial(3) + (y4_0*(r*h)^4)/
        factorial(4) + (y5_0*(r*h)^5)/factorial(5) ;
32 h = 0.05;
33 r = 2;
34 yx1 = eval(yxr_5d);
35 format('v',8);
36 disp(dbl(yx1), 'Value when r = 2 :')

```

Scilab code Exa 7.5 3 Step Adams Bashforth and 2 step Adam Moulton formula

```

1 // 3-Step Adams - Bashforth and 2- step Adam-Moulton
  formula
2 clc;
3 clear;
4 close();
5 format('v',8);
6 funcprot(0);
7 disp('Integral 0 to 2 exp(x)dx');
8 deff('[yd]=f(x,y)', 'yd = -y^2/(1+x)');
9
10 y0 = 1;
11 x0 = 0;
12 h = 0.05;
13 x1 = x0+h;
14 x2 = x1+h;
15 y2 = 0.91298;
16 y1 = 0.95348;
17 for i = 1:2
18     yn = y2 + h*(23*f(x2,y2)-16*f(x1,y1)+5*f(x0,y0))

```

```

        /12;
19     disp(yn, 'yn(0) = ');
20     yn_i = yn;
21     yn_i = y2 + h*(5*f(x2+h, yn_i)+8*f(x2, y2)-f(x1, y1
        ))/12;
22     disp(yn_i , 'yn(i)');
23     yn_i = y2 + h*(5*f(x2+h, yn_i)+8*f(x2, y2)-f(x1, y1
        ))/12;
24     disp(yn_i , 'yn(i)');
25     y0 = y1; y1 = y2; y2 = yn_i;
26     x0 = x1; x1 = x2; x2 = x2+h;
27 end
28 x = 0.2 ;
29 act = 1/(1+log(1+x));
30 disp(act, 'The exact value is of y(0.2): ');
31 err = act - y2;
32 disp(err, 'The error is :');

```

Scilab code Exa 7.10 Runge Kutta Methods

```

1 // Runge- Kutta Methods
2 clc;
3 clear;
4 close();
5 format('v',8);
6 funcprot(0);
7 disp('Integral 0 to 2 exp(x)dx');
8 deff(' [t]=f(x,y)', 't=-y^2/(1+x)');
9 yn = 1;
10 xn = 0;
11 h = 0.05;
12 for i = 1:4
13     k1 = f(xn, yn);

```



```

14     k2 = f(xn+0.5*h,yn+.5*h*k1);
15     k3 = f(xn+0.5*h,yn+.5*h*k2);
16     k4 = f(xn+h,yn+h*k3);
17     yn_1 = yn + h*(k1+2*k2+2*k3+k4)/6;
18     n = i-1;
19     ann(:,i) = [n k1 k2 k3 k4 yn_1]';
20     yn = yn_1;
21     xn = xn+h;
22 end
23
24 disp(ann, 'Calculated integration : ');

```

Scilab code Exa 7.11 Eulers Methods

```

1 // Euler 's Methods
2 clc;
3 clear;
4 close();
5 format('v',8);
6 funcprot(0);
7 disp('Integral 0 to 2 exp(x)dx');
8 deff('[ud]=f(u,v)', 'ud=u^2-2*u*v');
9 deff('[vd]=g(x,u,v)', 'vd=u*x+u^2*sin(v)');
10 un = 1;
11 vn = -1;
12 xn = 0;
13 h = 0.05;
14 for i = 1:2
15     un_1 = un + h*(f(un,vn));
16     disp(un_1);
17     vn_1 = vn + h*(g(xn,un,vn));
18     disp(vn_1);
19     vn = vn_1;

```

```

20     un = un_1;
21     xn = xn + h;
22 end
23 ann = [un vn];
24 disp(ann, 'Calculated U2 n V2 values : ');

```

Scilab code Exa 7.12 Eulers trapezoidal predictor corrector pair

```

1 // Euler's trapezoidal predictor-corrector pair
2 clc;
3 clear;
4 close();
5 format('v',8);
6 funcprot(0);
7 disp('Integral 0 to 2 exp(x)dx');
8 deff('[ud]=f(u,v)', 'ud=u^2-2*u*v');
9 deff('[vd]=g(x,u,v)', 'vd=u*x+u^2*sin(v)');
10 un = 1;
11 vn = -1;
12 xn = 0;
13 h = 0.05;
14 for i = 1:2
15     un_1p = un + h*(f(un, vn));
16     disp(un_1p);
17     vn_1p = vn + h*(g(xn, un, vn));
18     disp(vn_1p);
19     un_1c = un + h*(f(un, vn)+f(un_1p, vn_1p))/2;
20     disp(un_1c);
21     vn_1c = vn + h*(g(xn, un, vn)+g(xn+h, un_1p, vn_1p))
        /2;
22     disp(vn_1c);
23     un_1cc = un + h*(f(un, vn)+f(un_1c, vn_1c))/2;
24     disp(un_1cc);

```

```

25     vn_1cc = vn + h*(g(xn,un,vn)+g(xn+h,un_1c,vn_1c)
        )/2;
26     disp(vn_1cc);
27     vn = vn_1cc;
28     un = un_1cc;
29     xn = xn + h;
30 end
31 ann = [un vn];
32 disp(ann, 'Calculated U2 n V2 values : ');

```

Scilab code Exa 7.13 4 Stage Runge Kutta method

```

1 // 4-Stage Runge-Kutta method
2 clc;
3 clear;
4 close();
5 format('v',8);
6 funcprot(0);
7 disp('Integral 0 to 2 exp(x)dx');
8 deff(' [ud]=f(u,v) ', 'ud=u^2-2*u*v');
9 deff(' [vd]=g(x,u,v) ', 'vd=u*x+u^2*sin(v)');
10 un = 1;
11 vn = -1;
12 xn = 0;
13 h = 0.05;
14 for i = 1:2
15     k1 = f(un, vn);
16     disp(k1);
17     l1 = g(xn, un, vn);
18     disp(l1);
19     k2 = f(un+.5*h*k1, vn+.5*h*l1) ;
20     disp(k2);
21     l2 = g(xn+.5*h, un+.5*h*k1, vn+.5*h*l1) ;

```

```

22     disp(l2);
23     k3 = f(un+.5*h*k2,vn+.5*h*l2) ;
24     disp(k3);
25     l3 = g(xn+.5*h,un+.5*h*k2,vn+.5*h*l2) ;
26     disp(l3);
27     k4 = f(un+h*k3,vn+h*l3);
28     disp(k4);
29     l4 = g(xn+h,un+h*k3,vn+h*l3);
30     disp(l4);
31     un_1 = un + h*(k1+2*k2+2*k3+k4)/6;
32     disp(un_1,'u(n+1) : ');
33     vn_1 = vn + h*(l1+2*l2+2*l3+l4)/6;
34     disp(vn_1,'v(n+1) : ');
35     un = un_1;
36     vn = vn_1;
37     xn = xn +h;
38 end
39 ann = [un vn];
40 disp(ann,'Calculated U2 n V2 values : ');

```

Chapter 8

Ordinary Differential Equations boundary value problem

Scilab code Exa 8.1 The finite difference method

```
1 //The finite difference method
2 clc;
3 clear;
4 close();
5 format('v',7);
6 funcprot(0);
7 disp('Integral 0 to 2 exp(x)dx');
8 deff('[pp]=p(x)', 'pp=x');
9 deff('[qq]=q(x)', 'qq=-3');
10 deff('[rr]=r(x)', 'rr=exp(x)');
11 y0 = 1;
12 yn = 2;
13 x = [.2 .4 .6 .8 1];
14 h = 0.2;
15 A = [-2-h^2*q(x(1)) 1-h*p(x(1))/2 0 0;1+h*p(x(2))/2
      -2-h^2*q(x(2)) 1-h*p(x(2))/2 0;0 1+h*p(x(3))/2
      -2-h^2*q(x(3)) 1-h*p(x(3))/2;0 0 1+h*p(x(4))/2
      -2-h^2*q(x(4))];
16 disp(A, 'A');
```

```
17 c = [h^2*r(x(1))-(1+h*p(x(1))/2)*y0;h^2*r(x(2));h^2*  
        r(x(3));h^2*r(x(4))-(1-h*p(x(4))/2)*yn];  
18 Y = inv(A)*c;  
19 disp(Y', 'The respective values of y1,y2,y3,y4 : ');
```
