

Scilab Textbook Companion for
Numerical Methods For Scientists And
Engineers
by K. S. Rao¹

Created by
Viswanath Pasumarthi
Numerical Methods
Chemical Engineering
IIT Guwahati
College Teacher
Dr. Prakash Kotecha
Cross-Checked by

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 Methods For Scientists And Engineers

Author: K. S. Rao

Publisher: PHI Learning Pvt. Ltd., New Delhi

Edition: 2

Year: 2004

ISBN: 81-203-2395-5

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 Basics in Computing	8
2 Solution of Algebraic and Transcendental Equations	10
3 Solution of Linear System of Equations and Matrix Inversion	24
4 Eigenvalue Problems	36
5 Curve Fitting	42
6 Interpolation	50
7 Numerical Differentiation and Integration	73
8 Ordinary Differential Equations	93
9 Parabolic Partial Differential Equations	103
10 Elliptical Partial Differential Equations	113
11 Hyperbolic Partial Differential Equations	120

List of Scilab Codes

Exa 1.1	Conversion of Decimal to Binary	8
Exa 1.2	Conversion of Binary to Decimal	8
Exa 1.3	Conversion of Decimal to Binary and Octal	9
Exa 2.1	Root using Bisection Method	10
Exa 2.2	Root using Regula Falsi Method	12
Exa 2.3	Root using Regula Falsi Method	14
Exa 2.4	Root using Regula Falsi Method	15
Exa 2.5	Root using Method of Iteration	17
Exa 2.6	Root using Method of Iteration	18
Exa 2.7	Root using Newton Raphson Method	19
Exa 2.8	Root using Newton Raphson Method	20
Exa 2.9	Newton Scheme of Iteration	20
Exa 2.10	Newton Formula	21
Exa 2.11	Newton Raphson Extended Formula	21
Exa 2.12	Root using Muller Method	21
Exa 2.13	Graeffe Root Squaring Method	22
Exa 3.1	Gauss Elimination Method	24
Exa 3.2	Gauss Elimination Method with Partial Pivoting	25
Exa 3.3	Gauss Elimination Method with Partial Pivoting	26
Exa 3.4	Gauss Jordan Method	27
Exa 3.5	Crout Reduction Method	28
Exa 3.6	Jacobi Iterative Method	29
Exa 3.7	Gauss Seidel Method	31
Exa 3.8	Relaxation Method	32
Exa 3.9	Matrix Inverse using Gauss Elimination Method	33
Exa 3.10	Matrix Inverse using Gauss Jordan Method	34
Exa 4.1	Eigenvalues and Eigenvectors	36
Exa 4.2	Eigenvalues and Eigenvectors using Jacobi Method	37

Exa 4.3	Eigenvalues using Jacobi Method	38
Exa 4.4	Eigenvalues and Eigenvectors using Jacobi Method . .	40
Exa 5.1	Method of Group Averages	42
Exa 5.2	Method of Group Averages	43
Exa 5.3	Method of Group Averages	43
Exa 5.4	Method of Least Squares	44
Exa 5.5	Method of Least Squares	45
Exa 5.6	Method of Least Squares	45
Exa 5.7	Method of Least Squares	46
Exa 5.8	Principle of Least Squares	47
Exa 5.9	Method of Moments	48
Exa 5.10	Method of Moments	48
Exa 6.1	Forward Difference Table	50
Exa 6.2	Expression for Finite Difference Elements	51
Exa 6.3	Expression for Finite Difference Elements	51
Exa 6.4	Expression for Finite Difference Elements	51
Exa 6.5	Proof of Relation	51
Exa 6.6	Proofs of given Relations	52
Exa 6.7	Proof for Commutation of given Operations	52
Exa 6.8	Newton Forward Difference Interpolation Formula . .	52
Exa 6.9	Newton Forward Difference Interpolation Formula . .	53
Exa 6.10	Newton Forward Difference Interpolation Formula . .	54
Exa 6.11	Newton Forward Difference Interpolation Formula . .	55
Exa 6.12	Newton Backward Difference Interpolation Formula . .	56
Exa 6.13	Newton Backward Difference Interpolation Formula . .	57
Exa 6.14	Lagrange Interpolation Formula	58
Exa 6.15	Lagrange Interpolation Formula	59
Exa 6.16	Lagrange and Newton Divided Difference Interpolation Formulae	60
Exa 6.17	Newton Divided Difference Interpolation Formulae . .	61
Exa 6.18	Identity Proof for Newton and Lagrange Interpolation Formulae	62
Exa 6.19	Interpolation in Two Dimensions	62
Exa 6.20	Cubic Spline Curve	65
Exa 6.21	Cubic Spline Curve	67
Exa 6.22	Minima of a Tabulated Function	68
Exa 6.23	Maxima of a Tabulated Function	70
Exa 6.24	Determination of Function Value	71

Exa 7.1	Determination of Differential Function Value	73
Exa 7.2	Determination of Differential Function Value	74
Exa 7.3	Determination of Differential Function Value	75
Exa 7.4	Determination of Differential Function Value	76
Exa 7.5	Richardson Extrapolation Limit	78
Exa 7.6	Integral using Trapezoidal and Simpson One Third Rule	79
Exa 7.7	Integral using Simpson One Third Rule	80
Exa 7.8	Integral using Trapezoidal and Simpson One Third Rule	82
Exa 7.9	Integral using Simpson One Third Rule	83
Exa 7.10	Integral using Simpson One Third Rule	85
Exa 7.11	Romberg Integration Method	85
Exa 7.12	Double Integral using Trapezoidal Rule	87
Exa 7.13	Double Integral using Trapezoidal Rule	88
Exa 7.14	One Point Gauss Legendre Quadrature Formula	89
Exa 7.15	Two Point Gauss Legendre Quadrature Formula	90
Exa 7.16	Four Point Gauss Legendre Quadrature Formula	91
Exa 8.1	Initial Value Problem using Taylor Series Method	93
Exa 8.2	Initial Value Problem using Euler Method	94
Exa 8.3	Initial Value Problem using Modified Euler Method	95
Exa 8.4	Initial Value Problem using Second Order Runge Kutta Method	96
Exa 8.5	Initial Value Problem using Fourth Order Runge Kutta Method	97
Exa 8.6	Van Der Pol Equation using Fourth Order Runge Kutta Equation	98
Exa 8.7	Milne Predictor Corrector Method	99
Exa 8.8	Milne Predictor Corrector Method	100
Exa 8.9	Adam Moulton Predictor Corrector Method	101
Exa 9.1	Taylor Series Expansion	103
Exa 9.2	Initial Boundary Value Problem using Explicit Finite Difference Method	103
Exa 9.3	Initial Boundary Value Problem using Explicit Finite Difference Method	105
Exa 9.4	Crank Nicolson Finite Difference Method	106
Exa 9.5	Crank Nicolson Finite Difference Method	108
Exa 9.6	Crank Nicolson Scheme for Diffusion Equation	109
Exa 9.7	Alternate Direction Implicit Method	109
Exa 10.1	Laplace Equation using Five Point Formulae	113

Exa 10.2	Temperature in Two Dimensional Geometry	114
Exa 10.3	Laplace Equation in Two Dimension using Five Point Formulae	114
Exa 10.4	Poisson Equation using Liebmann Iterative Method . .	116
Exa 10.5	Laplace Equation using Liebmann Over Relaxation Method	117
Exa 11.1	Initial Value Problem using Wave Equation	120
Exa 11.2	Initial Value Problem using Wave Equation	121

Chapter 1

Basics in Computing

Scilab code Exa 1.1 Conversion of Decimal to Binary

```
1 //Example 1.1
2 clc
3 clear
4
5 dec_N = 47;
6 bin_N = dec2bin(dec_N)
7 disp(bin_N)
```

Scilab code Exa 1.2 Conversion of Binary to Decimal

```
1 //Example 1.2
2 clc
3 clear
4
5 dec = 0.7625;
6 iter = 1;
7 while(1)
8     dec = 2 * dec;
```

```
9     p(iter) = int(dec);
10    dec = dec - int(dec);
11    if iter == 8 then
12        break
13    end
14    iter = iter + 1;
15 end
16 a = strcat(string(p));
17 bin = strcat(['0.',a])
18 disp(bin)
```

Scilab code Exa 1.3 Conversion of Decimal to Binary and Octal

```
1 //Example 1.3
2 clc
3 clear
4
5 dec_N = 59;
6 bin_N = dec2bin(dec_N)
7 oct_N = dec2oct(dec_N)
8 disp(bin_N," Binary:")
9 disp(oct_N," Octal:")
```

Chapter 2

Solution of Algebraic and Transcendental Equations

Scilab code Exa 2.1 Root using Bisection Method

```
1 //Example 2.1
2 clc
3 clear
4
5 function [root] = Bisection(fun,x,tol,maxit)
6 // Bisection: Computes roots of the function in the
   given range using Bisection Method
7 //// Input: Bisection(fun,x,tol,maxit)
8 // fun = function handle
9 // x = range in between sign change is evident
10 // tol = Maximum error between iterations that can
   be tolerated
11 // maxit = Maximum number of iterations
12 //// Output: [root]
13 // Root: Root of the given function in defined range
14
15 if fun(x(1)) > 0 then
16     xu = x(1);    x1 = x(2);
17 else
```

```

18     xu = x(2);     xl = x(1);
19 end
20
21 Ea = 1;
22 iter = 1;
23
24 while(1)
25     xr(iter) = (xl(iter) + xu(iter)) / 2;
26     if fun(xr(iter)) > 0 then
27         xu(iter+1) = xr(iter);
28         xl(iter+1) = xl(iter);
29     elseif fun(xr(iter)) < 0 then
30         xl(iter+1) = xr(iter);
31         xu(iter+1) = xu(iter);
32     else
33         break
34     end
35
36     if iter>1 then
37         Ea(iter) = 100 * abs((xr(iter) - xr(iter-1))
38             / xr(iter));
39
40     if Ea(iter) < tol | iter == maxit then
41         break
42     end
43     iter = iter + 1;
44 end
45 root = xr(iter);
46 endfunction
47
48 function f = fun1(x)
49     f = x.^3 -9*x + 1;
50 endfunction
51
52 x = [2 4];
53 tol = 1e-4;
54 maxit = 5;

```

```
55 root = Bisection(fun1,x,tol,maxit);
56 disp(root,"root = ")
```

Scilab code Exa 2.2 Root using Regula Falsi Method

```
1 //Example 2.2
2 clc
3 clear
4
5 function [root] = FalsePosition(fun,x,tol,maxit)
6 // FalsePosition: Computes roots of the function in
7 // the given range using False Position Method
8 // Input: FalsePosition(fun,x,tol,maxit)
9 // fun = function handle
10 // x = range in between sign change is evident
11 // tol = Maximum error between iterations that can
12 // be tolerated
13 // maxit = Maximum number of iterations
14 // Output: [root]
15 // Root: Root of the given function in defined range
16
17 if fun(x(1)) > 0 then
18     xu = x(1);    x1 = x(2);
19 else
20     xu = x(2);    x1 = x(1);
21 end
22
23 Ea = 1;
24 iter = 1;
25 while(1)
26     xr(iter) = x1(iter) - ((xu(iter)-x1(iter)) / (
27         fun(xu(iter))-fun(x1(iter))) * fun(x1(iter)))
28     );
29     if fun(xr(iter)) > 0 then
```

```

27         xu(iter+1) = xr(iter);
28         xl(iter+1) = xl(iter);
29     elseif fun(xr(iter)) < 0 then
30         xl(iter+1) = xr(iter);
31         xu(iter+1) = xu(iter);
32     else
33         break
34     end
35
36     if iter>1 then
37         Ea(iter) = 100 * abs((xr(iter) - xr(iter-1))
38             / xr(iter));
39
40     if Ea(iter) < tol | iter == maxit then
41         break
42     end
43     iter = iter + 1;
44 end
45 root = xr(iter);
46 endfunction
47
48 function f = fun1(x)
49     f = x.^3 -9*x + 1;
50 endfunction
51
52 x = [2 4; 2 3];
53 tol = 1e-4;
54 maxit = 3;
55 for i = 1:2
56     root = FalsePosition(fun1,x(i,:),tol,maxit);
57     root = round(root*10^5)/10^5;
58     disp(strcat(["root(",string(i),") = ",string(
59         root)]))

```

Scilab code Exa 2.3 Root using Regula Falsi Method

```
1 //Example 2.3
2 clc
3 clear
4
5 function [root] = FalsePosition(fun,x,tol,maxit)
6 // FalsePosition: Computes roots of the function in
7 // the given range using False Position Method
8 // Input: FalsePosition(fun,x,tol,maxit)
9 // fun = function handle
10 // x = range in between sign change is evident
11 // tol = Maximum error between iterations that can
12 // be tolerated
13 // maxit = Maximum number of iterations
14 // Output: [root]
15 // Root: Root of the given function in defined range
16
17 if fun(x(1)) > 0 then
18     xu = x(1);    xl = x(2);
19 else
20     xu = x(2);    xl = x(1);
21 end
22
23 Ea = 1;
24 iter = 1;
25 while(1)
26     xr(iter) = xl(iter) - ((xu(iter)-xl(iter)) / (
27         fun(xu(iter))-fun(xl(iter))) * fun(xl(iter)))
28     ;
29     if fun(xr(iter)) > 0 then
30         xu(iter+1) = xr(iter);
31         xl(iter+1) = xl(iter);
```

```

29     elseif fun(xr(iter)) < 0 then
30         xl(iter+1) = xr(iter);
31         xu(iter+1) = xu(iter);
32     else
33         break
34     end
35
36     if iter>1 then
37         Ea(iter) = 100 * abs((xr(iter) - xr(iter-1))
38             / xr(iter));
39
40     if Ea(iter) < tol | iter == maxit then
41         break
42     end
43     iter = iter + 1;
44 end
45 root = xr(iter);
46 endfunction
47
48 function f = fun3(x)
49     f = log(x) - cos(x);
50 endfunction
51
52 x = [1 2];
53 tol = 1e-4;
54 maxit = 5;
55 root = FalsePosition(fun3,x,tol,maxit);
56 disp(round(root*10^4)/10^4,"root = ")

```

Scilab code Exa 2.4 Root using Regula Falsi Method

```

1 //Example 2.4
2 clc
3 clear

```



```

4
5 function [root] = FalsePosition(fun,x,tol,maxit)
6 // FalsePosition: Computes roots of the function in
   the given range using False Position Method
7 //// Input: FalsePosition(fun,x,tol,maxit)
8 // fun = function handle
9 // x = range in between sign change is evident
10 // tol = Maximum error between iterations that can
   be tolerated
11 // maxit = Maximum number of iterations
12 //// Output: [root]
13 // Root: Root of the given function in defined range
14
15 if fun(x(1)) > 0 then
16     xu = x(1);    xl = x(2);
17 else
18     xu = x(2);    xl = x(1);
19 end
20
21 Ea = 1;
22 iter = 1;
23
24 while(1)
25     xr(iter) = xl(iter) - ((xu(iter)-xl(iter)) / (
        fun(xu(iter))-fun(xl(iter))) * fun(xl(iter)))
        ;
26     if fun(xr(iter)) > 0 then
27         xu(iter+1) = xr(iter);
28         xl(iter+1) = xl(iter);
29     elseif fun(xr(iter)) < 0 then
30         xl(iter+1) = xr(iter);
31         xu(iter+1) = xu(iter);
32     else
33         break
34     end
35
36     if iter>1 then
37         Ea(iter) = 100 * abs((xr(iter) - xr(iter-1))

```

```

        / xr(iter));
38     end
39
40     if Ea(iter) < tol | iter == maxit then
41         break
42     end
43     iter = iter + 1;
44 end
45 root = xr(iter);
46 endfunction
47
48 function f = fun4(x)
49     f = x.*log10(x) - 1.2;
50 endfunction
51
52 clc
53 x = [2 3];
54 tol = 1e-4;
55 maxit = 2;
56 root = FalsePosition(fun4,x,tol,maxit);
57 disp(round(root*10^4)/10^4,"root = ")

```

Scilab code Exa 2.5 Root using Method of Iteration

```

1 //Example 2.5
2 clc
3 clear
4
5 function f = fun5(x)
6     f = exp(-x)/10;
7 endfunction
8
9 clc
10 tol = 1e-4;
11 maxit = 4;

```

```

12 xold = 0;
13 iter = 1;
14 while(1)
15     xnew = fun5(xold);
16     EA = abs(xnew - xold);
17     if EA < tol | iter > maxit then
18         break
19     end
20     xold = xnew;
21     iter = iter + 1;
22 end
23 root = round(xnew*10^4) / 10^4;           //rounded to 4
      decimal places
24 disp(root,"root = ")

```

Scilab code Exa 2.6 Root using Method of Iteration

```

1 //Example 2.6
2 clc
3 clear
4
5 function f = fun6(x)
6     f = 1./ sqrt(x+1);
7 endfunction
8
9 tol = 1e-4;
10 maxit = 6;
11 xold = 1;
12 iter = 1;
13 while(1)
14     xnew = fun6(xold);
15     EA = abs(xnew - xold);
16     if EA < tol | iter > maxit then
17         break
18     end

```

```
19     xold = xnew;
20     iter = iter + 1;
21 end
22 root = round(xnew*10^4) / 10^4;           //rounded to 4
      decimal places
23 disp(root,"root = ")
```

Scilab code Exa 2.7 Root using Newton Raphson Method

```
1 //Example 2.7
2 clc
3 clear
4
5 function [f,df] = fun7(x)
6     f = x.*exp(x) - 2;
7     df = x.*exp(x) + exp(x);
8 endfunction
9
10 xold = 1;
11 maxit = 2;
12 iter = 1;
13
14 while (1)
15     [fx,dfx] = fun7(xold);
16     xnew = xold - fx/dfx;
17     if iter == maxit then
18         break
19     end
20     xold = xnew;
21     iter = iter + 1;
22 end
23 root = round(xnew*10^3) / 10^3;
24 disp(root,"root = ")
```

Scilab code Exa 2.8 Root using Newton Raphson Method

```
1 //Example 2.8
2 clc
3 clear
4
5 function [f,df] = fun8(x)
6     f = x.^3 - x - 1;
7     df = 3*x.^2 - 1;
8 endfunction
9
10 xold = 1;
11 maxit = 5;
12 iter = 1;
13
14 while (1)
15     [fx,dfx] = fun8(xold);
16     xnew = xold - fx/dfx;
17     if iter == maxit then
18         break
19     end
20     xold = xnew;
21     iter = iter + 1;
22 end
23 root = round(xnew*10^4) / 10^4;
24 disp(root,"root = ")
```

Scilab code Exa 2.9 Newton Scheme of Iteration

```
1 // Example 2.9
2 // This is an analytical problem and need not be
   coded.
```

Scilab code Exa 2.10 Newton Formula

```
1 //Example 2.10
2 clc
3 clear
4
5 N = 12;
6 xold = 3.5;
7 iter = 1;
8 maxit = 3;
9
10 while (1)
11     xnew = (xold + N/xold) / 2;
12     if iter == maxit then
13         break
14     end
15     xold = xnew;
16     iter = iter + 1;
17 end
18 root = round(xnew*10^4) / 10^4;
19 disp(root,"root = ")
```

Scilab code Exa 2.11 Newton Raphson Extended Formula

```
1 // Example 2.11
2 // This is an analytical problem and need not be
   coded.
```

Scilab code Exa 2.12 Root using Muller Method

```

1 //Example 2.12
2 clc
3 clear
4
5 function [f] = fun12(x)
6     f = x.^3 - x - 1;
7 endfunction
8
9 x = [0 1 2];
10 h = [x(2)-x(1) x(3)-x(2)];
11 lamdai = h(2)/h(1);
12 deli = 1 + lamdai;
13 f = fun12(x);
14
15 g = f(1)*lamdai^2 - f(2)*deli^2 + f(3)*(lamdai +
    deli);
16 lamda = -2*f(3)*deli / (g + sqrt(g^2 - 4*f(3)*deli*(
    f(1)*lamdai - f(2)*deli + f(3))));
17 xnew = x(3) + lamda*h(2);
18 xnew = round(xnew*10^5) / 10^5;
19 disp(xnew,"root = ")

```

Scilab code Exa 2.13 Graeffe Root Squaring Method

```

1 //Example 2.13
2 clc
3 clear
4
5 a = [-6 11 -6 1];
6 maxit = 3;
7 for iter = 1:maxit
8     a = [a(4)^2 -(a(3)^2 -2*a(2)*a(4)) (a(2)^2 - 2*a
        (1)*a(3)) -a(1)^2];
9     root = abs([a(4)/a(3) a(3)/a(2) a(2)/a(1)])
        ^(1/(2^iter));

```

```
10 end
11 root = round(root*10^5) / 10^5;
12 disp(root,"Estimated roots for the polynomial are: ")
    )
```

Chapter 3

Solution of Linear System of Equations and Matrix Inversion

Scilab code Exa 3.1 Gauss Elimination Method

```
1 //Example 3.1
2 clc
3 clear
4
5 A = [2 3 -1; 4 4 -3; -2 3 -1]; //Coefficient Matrix
6 B = [5; 3; 1]; //Constant Matrix
7
8 n = length(B);
9 Aug = [A,B];
10
11 // Forward Elimination
12 for j = 1:n-1
13     for i = j+1:n
14         Aug(i,j:n+1) = Aug(i,j:n+1) - Aug(i,j) / Aug
            (j,j) * Aug(j,j:n+1);
15     end
16 end
17
18 // Backward Substitution
```

```

19 x = zeros(n,1);
20 x(n) = Aug(n,n+1) / Aug(n,n);
21 for i = n-1:-1:1
22     x(i) = (Aug(i,n+1)-Aug(i,i+1:n)*x(i+1:n))/Aug(i,
        i);
23 end
24 disp(strcat(["x = ",string(x(1))]))
25 disp(strcat(["y = ",string(x(2))]))
26 disp(strcat(["z = ",string(x(3))]))

```

Scilab code Exa 3.2 Gauss Elimination Method with Partial Pivoting

```

1 //Example 3.2
2 clc
3 clear
4
5 A = [1 1 1; 3 3 4; 2 1 3]; //Coefficient Matrix
6 B = [7; 24; 16]; //Constant Matrix
7
8 n = length(B);
9 Aug = [A,B];
10
11 // Forward Elimination
12 for j = 1:n-1
13     // Partial Pivoting
14     [dummy,t] = max(abs(Aug(j:n,j)));
15     lrow = t(1)+j-1;
16     Aug([j,lrow],:) = Aug([lrow,j],:);
17
18     for i = j+1:n
19         Aug(i,j:n+1) = Aug(i,j:n+1) - Aug(i,j) / Aug
                (j,j) * Aug(j,j:n+1);
20     end
21 end
22

```

```

23 // Backward Substitution
24 x = zeros(n,1);
25 x(n) = Aug(n,n+1) / Aug(n,n);
26 for i = n-1:-1:1
27     x(i) = (Aug(i,n+1)-Aug(i,i+1:n)*x(i+1:n))/Aug(i,
           i);
28 end
29 disp(strcat(["x = ",string(x(1))]))
30 disp(strcat(["y = ",string(x(2))]))
31 disp(strcat(["z = ",string(x(3))]))

```

Scilab code Exa 3.3 Gauss Elimination Method with Partial Pivoting

```

1 //Example 3.3
2 clc
3 clear
4
5 A = [0 4 2 8; 4 10 5 4; 4 5 6.5 2; 9 4 4 0]; //
   Coefficient Matrix
6 B = [24; 32; 26; 21]; //Constant Matrix
7
8 n = length(B);
9 Aug = [A,B];
10
11 // Forward Elimination
12 for j = 1:n-1
13     // Partial Pivoting
14     [dummy,t] = max(abs(Aug(j:n,j)));
15     lrow = t(1)+j-1;
16     Aug([j,lrow],:) = Aug([lrow,j],:);
17
18     for i = j+1:n
19         Aug(i,j:n+1) = Aug(i,j:n+1) - Aug(i,j) / Aug
           (j,j) * Aug(j,j:n+1);
20     end

```

```

21 end
22
23 // Backward Substitution
24 x = zeros(n,1);
25 x(n) = Aug(n,n+1) / Aug(n,n);
26 for i = n-1:-1:1
27     x(i) = (Aug(i,n+1)-Aug(i,i+1:n)*x(i+1:n))/Aug(i,
           i);
28 end
29
30 disp(strcat(["x1 = ",string(x(1))]))
31 disp(strcat(["x2 = ",string(x(2))]))
32 disp(strcat(["x3 = ",string(x(3))]))
33 disp(strcat(["x4 = ",string(x(4))]))

```

Scilab code Exa 3.4 Gauss Jordan Method

```

1 //Example 3.4
2 clc
3 clear
4
5 A = [1 2 1; 2 3 4; 4 3 2];
6 B = [8; 20; 16];
7 n = length(B);
8 Aug = [A,B];
9
10 // Forward Elimination
11 for j = 1:n-1
12     for i = j+1:n
13         Aug(i,j:n+1) = Aug(i,j:n+1) - Aug(i,j) / Aug
           (j,j) * Aug(j,j:n+1);
14     end
15 end
16
17 // Backward Elimination

```

```

18 for j = n:-1:2
19     Aug(1:j-1,:) = Aug(1:j-1,:) - Aug(1:j-1,j) / Aug
        (j,j) * Aug(j,:);
20 end
21
22 // Diagonal Normalization
23 for j=1:n
24     Aug(j,:) = Aug(j,:) / Aug(j,j);
25 end
26 x = Aug(:,n+1);
27 disp(strcat(["x = ",string(x(1))]))
28 disp(strcat(["y = ",string(x(2))]))
29 disp(strcat(["z = ",string(x(3))]))

```

Scilab code Exa 3.5 Crout Reduction Method

```

1 //Example 3.5
2 clc
3 clear
4
5 A = [5 -2 1; 7 1 -5; 3 7 4];
6 B = [4; 8; 10];
7
8 n = length (B);
9 L = zeros(n,n); // L = Lower Triangular
    Matrix Initiation
10 U = eye(n,n); // U = Upper Triangular
    Matrix Initiation
11
12 // LU Decomposition
13 for i = 1:n
14     sum1 = zeros(n-i+1,1);
15     for k = 1:i-1
16         sum1 = sum1 + L(i:n,k) * U(k,i);
17     end

```

```

18     L(i:n,i) = A(i:n,i) - sum1;
19
20     sum2 = zeros(1,n-i);
21     for k = 1:i-1
22         sum2 = sum2 + L(i,k) * U(k,i+1:n);
23     end
24     U(i,i+1:n) = (A(i,i+1:n) - sum2) / L(i,i);
25 end
26
27 // Forward Substitution
28 D = ones(n,1);
29 for i = 1:n
30     sum3 = 0;
31     for k = 1:i-1
32         sum3 = sum3 + L(i,k) * D(k);
33     end
34     D(i) = (B(i) - sum3) / L(i,i);
35 end
36
37 // Back Substitution
38 x = ones(n,1);
39 for i = n:-1:1
40     sum4 = 0;
41     for k = i+1:n
42         sum4 = sum4 + U(i,k) * x(k);
43     end
44     x(i) = D(i) - sum4;
45 end
46
47 disp(strcat(["x1 = ",string(x(1))]))
48 disp(strcat(["x2 = ",string(x(2))]))
49 disp(strcat(["x3 = ",string(x(3))]))

```

Scilab code Exa 3.6 Jacobi Iterative Method

```

1 //Example 3.6
2 clc
3 clear
4
5 A = [83 11 -4; 7 52 13; 3 8 29];
6 B = [95; 104; 71];
7
8 n = length (B);
9 tol = 1e-4;
10 iter = 1;
11 maxit = 5;
12
13 x = zeros(n,1);           //Intial guess
14 E = ones(n,1);           //Assuming to avoid
    variable size error
15 S = diag(diag(A));
16
17
18 while (1)
19     x(:,iter+1) = S\(B + (S-A)*(x(:,iter)));
20     E(:,iter+1) = (x(:,iter+1)-x(:,iter))./x(:,iter
        +1)*100;
21     if x(:,iter) == 0
22         Error = 1;
23     else
24         Error = sqrt((sum((E(:,iter+1)).^2))/n);
25     end
26
27     if Error <= tol | iter == maxit
28         break
29     end
30     iter = iter+1;
31 end
32 xact = x(:,iter);
33 x = round(x*10^4)/10^4;
34 x(:,1) = [];
35 mprintf('%s %3s %9s %9s', 'Iter No.', 'x', 'y', 'z');
36 disp([(1:iter)' x']);

```

Scilab code Exa 3.7 Gauss Seidel Method

```
1 //Example 3.7
2 clc
3 clear
4
5 A = [1 -1/4 -1/4 0; -1/4 1 0 -1/4; -1/4 0 1 -1/4; 0
      -1/4 -1/4 1];
6 B = [1/2; 1/2; 1/4; 1/4];
7
8 n = length (B);
9 tol = 1e-4;
10 iter = 1;
11 maxit = 5;
12
13 x = zeros(n,1);           //Intial guess
14 E = ones(n,1);           //Assuming to avoid
      variable size error
15 S = diag(diag(A));
16 T = S-A;
17 xold = x;
18
19 while (1)
20     for i = 1:n
21         x(i,iter+1) = (B(i) + T(i,:) * xold) / A(i,i
22             );
23         E(i,iter+1) = (x(i,iter+1)-xold(i))/x(i,iter
24             +1)*100;
25         xold(i) = x(i,iter+1);
26     end
27
28     if x(:,iter) == 0
29         E = 1;
30     else
```



```

29         E = sqrt((sum((E(:,iter+1)).^2))/n);
30     end
31
32     if E <= tol | iter == maxit
33         break
34     end
35     iter = iter + 1;
36 end
37 X = x(:,iter);
38 x = round(x*10^5)/10^5;
39 x(:,1) = [];
40 mprintf('%s %3s %11s %10s %10s', 'Iter No. ', 'x1', 'x2'
        , 'x3', 'x4');
41 disp([(1:iter)' x']);

```

Scilab code Exa 3.8 Relaxation Method

```

1 //Example 3.8
2 clc
3 clear
4
5 A = [6 -3 1; 2 1 -8;1 -7 1];
6 b = [11; -15; 10];
7
8 n = length (b);
9 tol = 1e-4;
10 iter = 1;
11 maxit = 9;
12
13 x = zeros(1,n); //Intial guess
14 absA = abs(A);
15 [dummy,index] = max(absA(1,:),absA(2,:),absA(3,:));
16 if length(unique(index)) == n
17     nu_T = diag(diag(A(index,:))) - A(index,:);
18     if abs(diag(A(index,:))) - (sum(abs(nu_T),2)) >

```

```

0
19         A(index,:) = A;
20         b(index,:) = b;
21     end
22 end
23
24 for iter = 1:maxit
25     R(iter,:) = b' - x(iter,:) * A';
26     [mx,i] = max(abs(R(iter,:)));
27     Rmax(iter) = R(iter,i);
28     dx(iter) = Rmax(iter) ./ A(i,i);
29     x(iter+1,:) = x(iter,:);
30     x(iter+1,i) = x(iter,i) + dx(iter);
31 end
32 R = round(R*10^4)/10^4;
33 Rmax = round(Rmax*10^4)/10^4;
34 dx = round(dx*10^4)/10^4;
35 x = round(x*10^4)/10^4;
36 mprintf('%s %3s %9s %9s %12s %10s %6s %9s %9s', 'Iter
        No. ', 'R1', 'R2', 'R3', 'Max Ri', 'Diff dxi', 'x1', 'x2
        ', 'x3');
37 disp([(1:maxit)' R Rmax dx x(1:maxit,:)])

```

Scilab code Exa 3.9 Matrix Inverse using Gauss Elimination Method

```

1 //Example 3.9
2 clc
3 clear
4
5 A = [1 1 1; 4 3 -1; 3 5 3];
6 n = length(A(1,:));
7 Aug = [A, eye(n,n)];
8
9 // Forward Elimination
10 for j = 1:n-1

```

```

11     for i = j+1:n
12         Aug(i,j:2*n) = Aug(i,j:2*n) - Aug(i,j) / Aug
            (j,j) * Aug(j,j:2*n);
13     end
14 end
15
16 // Backward Elimination
17 for j = n:-1:2
18     Aug(1:j-1,:) = Aug(1:j-1,:) - Aug(1:j-1,j) / Aug
            (j,j) * Aug(j,:);
19 end
20
21 // Diagonal Normalization
22 for j=1:n
23     Aug(j,:) = Aug(j,:) / Aug(j,j);
24 end
25 Inv_A = Aug(:,n+1:2*n);
26 disp(Inv_A,"Inverse of A (A-1) = ")

```

Scilab code Exa 3.10 Matrix Inverse using Gauss Jordan Method

```

1 //Example 3.10
2 clc
3 clear
4
5 A = [1 1 1; 4 3 -1; 3 5 3];
6 n = length (A(1,:));
7 Aug = [A, eye(n,n)];
8
9 N = 1:n;
10 for i = 1:n
11     dummy1 = N;
12     dummy1(i) = [];
13     index(i,:) = dummy1;
14 end

```

```

15
16 // Forward Elimination
17 for j = 1:n
18     [dummy2,t] = max(abs(Aug(j:n,j)));
19     lrow = t+j-1;
20     Aug([j,lrow],:) = Aug([lrow,j],:);
21     Aug(j,:) = Aug(j,:) / Aug(j,j);
22     for i = index(j,:)
23         Aug(i,:) = Aug(i,:) - Aug(i,j) / Aug(j,j) *
                Aug(j,:);
24     end
25 end
26 Inv_A = Aug(:,n+1:2*n);
27 disp(Inv_A,"Inverse of A (A-1) = ")

```

Chapter 4

Eigenvalue Problems

Scilab code Exa 4.1 Eigenvalues and Eigenvectors

```
1 //Example 4.1
2 clc
3 clear
4
5 A = [2 3 2; 4 3 5; 3 2 9];
6 v = [1; 1; 1];
7 iter = 1;
8 maxit = 5;
9
10 while(1)
11     u(:,iter) = A * v(:,iter);
12     q(iter) = max(u(:,iter));
13     v(:,iter+1) = u(:,iter) / q(iter);
14     if iter == maxit then
15         break
16     end
17     iter = iter + 1;
18 end
19 X = round(v(:,iter)*10^2) / 10^2;
20 disp(X,"Eigen Vector:")
```

Scilab code Exa 4.2 Eigenvalues and Eigenvectors using Jacobi Method

```
1 //Example 4.2
2 clc
3 clear
4
5 rt2 = sqrt(2);
6 A = [1 rt2 2; rt2 3 rt2; 2 rt2 1];
7 [n,n] = size(A);
8 iter = 1;
9 maxit = 3;
10 D = A;
11 S = 1;
12
13 while(1)
14     D_offdiag = D - diag(diag(D));
15     [mx,index1] = max(abs(D_offdiag));
16     i = index1(1);
17     j = index1(2);
18     if (D(i,i)-D(j,j)) == 0 then
19         theta = %pi/4;
20     else
21         theta = atan(2*D(i,j)/(D(i,i)-D(j,j))) / 2;
22     end
23     S1 = eye(n,n);
24     S1(i,i) = cos(theta);
25     S1(i,j) = -sin(theta);
26     S1(j,i) = sin(theta);
27     S1(j,j) = cos(theta);
28
29     D1 = inv(S1) * D * S1;
30     for j = 1:n
31         for i = 1:n
32             if abs(D1(i,j)) < 1D-10 then
```

```

33             D1(i,j) = 0;
34         end
35     end
36 end
37 S = S * S1;
38
39 if D1 - diag(diag(D1)) == zeros(n,n) | iter ==
    maxit then
40     eigval = diag(D1);
41     disp('Eigen Values:')
42     disp(eigval)
43
44     disp('Eigen Vectors:')
45     disp(S(:,1))
46     disp(S(:,2))
47     disp(S(:,3))
48     break
49 end
50
51     iter = iter + 1;
52     D = D1;
53 end

```

Scilab code Exa 4.3 Eigenvalues using Jacobi Method

```

1 //Example 4.3
2 clc
3 clear
4
5 A = [2 -1 0; -1 2 -1; 0 -1 2];
6 [n,n] = size(A);
7 iter = 1;
8 maxit = 3;
9 //Note: Diagonal form may be achieved at iter = 9.
    Modify maxit to greater than 9 for exact result.

```

```

10
11 D = A;
12 S = 1;
13
14 while(1)
15     D_offdiag = D - diag(diag(D));
16     [mx,index1] = max(abs(D_offdiag));
17     i = index1(1);
18     j = index1(2);
19     if (D(i,i)-D(j,j)) == 0 then
20         theta = %pi/4;
21     else
22         theta = atan(2*D(i,j)/(D(i,i)-D(j,j))) / 2;
23     end
24     S1 = eye(n,n);
25     S1(i,i) = cos(theta);
26     S1(i,j) = -sin(theta);
27     S1(j,i) = sin(theta);
28     S1(j,j) = cos(theta);
29
30     D1 = inv(S1) * D * S1;
31     for j = 1:n
32         for i = 1:n
33             if abs(D1(i,j)) < 1D-10 then
34                 D1(i,j) = 0;
35             end
36         end
37     end
38     S = S * S1;
39
40     if D1 - diag(diag(D1)) == zeros(n,n) | iter ==
maxit then
41         eigval = diag(D1);
42         eigval = round(eigval*10^3)/10^3;
43         disp('Eigen Values:')
44         disp(eigval)
45         break
46     end

```



```

47
48     iter = iter + 1;
49     D = D1;
50 end

```

Scilab code Exa 4.4 Eigenvalues and Eigenvectors using Jacobi Method

```

1 //Example 4.4
2 clc
3 clear
4
5 A = [5 0 1; 0 -2 0; 1 0 5];
6 [n,n] = size(A);
7 iter = 1;
8 maxit = 3;
9 D = A;
10 S = 1;
11
12 while(1)
13     D_offdiag = D - diag(diag(D));
14     [mx,index1] = max(abs(D_offdiag));
15     i = index1(1);
16     j = index1(2);
17     if (D(i,i)-D(j,j)) == 0 then
18         theta = %pi/4;
19     else
20         theta = atan(2*D(i,j)/(D(i,i)-D(j,j))) / 2;
21     end
22     S1 = eye(n,n);
23     S1(i,i) = cos(theta);
24     S1(i,j) = -sin(theta);
25     S1(j,i) = sin(theta);
26     S1(j,j) = cos(theta);
27
28     D1 = inv(S1) * D * S1;

```

```

29     for j = 1:n
30         for i = 1:n
31             if abs(D1(i,j)) < 1D-10 then
32                 D1(i,j) = 0;
33             end
34         end
35     end
36     S = S * S1;
37
38     if D1 - diag(diag(D1)) == zeros(n,n) | iter ==
39         maxit then
40         eigval = diag(D1);
41         disp('Eigen Values:')
42         disp(eigval)
43
44         disp('Eigen Vectors:')
45         disp(S(:,1))
46         disp(S(:,2))
47         disp(S(:,3))
48         break
49     end
50     iter = iter + 1;
51     D = D1;
52 end

```

Chapter 5

Curve Fitting

Scilab code Exa 5.1 Method of Group Averages

```
1 //Example 5.1
2 clc
3 clear
4
5 x = 10:10:80;
6 y = [1.06 1.33 1.52 1.68 1.81 1.91 2.01 2.11];
7
8 X = log(x);
9 Y = log(y);
10
11 n = length(Y);
12 M1 = [sum(Y); sum(X.*Y)];
13 M2 = [n sum(X); sum(X) sum(X.^2)];
14
15 A = M2\M1;
16
17 m = exp(A(1));
18 n = A(2);
19
20 disp(round(m*10^4)/10^4, "m =")
21 disp(round(n*10^4)/10^4, "n =")
```

Scilab code Exa 5.2 Method of Group Averages

```
1 //Example 5.2
2 clc
3 clear
4
5 x = [20 30 35 40 45 50];
6 y = [10 11 11.8 12.4 13.5 14.4];
7
8 X = x.^2;
9 Y = y;
10
11 n = length(Y);
12 M1 = [sum(Y); sum(X.*Y)];
13 M2 = [n sum(X); sum(X) sum(X.^2)];
14
15 A = M2\M1;
16
17 a = A(1);
18 b = A(2);
19
20 disp(round(a*10^4)/10^4, "a =")
21 disp(round(b*10^4)/10^4, "b =")
```

Scilab code Exa 5.3 Method of Group Averages

```
1 //Example 5.3
2
3 clc
4 clear
5
```

```

6 x = [8 10 15 20 30 40];
7 y = [13 14 15.4 16.3 17.2 17.8];
8
9 X = 1 ./x;
10 Y = 1 ./y;
11
12 n = length(Y);
13 M1 = [sum(Y); sum(X.*Y)];
14 M2 = [n sum(X); sum(X) sum(X.^2)];
15
16 A = M2\M1;
17
18 b = A(1);
19 a = A(2);
20
21 disp(round(a*10^4)/10^4, "a =")
22 disp(round(b*10^4)/10^4, "b =")

```

Scilab code Exa 5.4 Method of Least Squares

```

1 //Example 5.4
2
3 clc
4 clear
5
6 X = 0.5:0.5:3;
7 Y = [15 17 19 14 10 7];
8
9 n = length(Y);
10 M1 = [sum(Y); sum(X.*Y)];
11 M2 = [n sum(X); sum(X) sum(X.^2)];
12
13 A = M2\M1;
14
15 b = A(1);

```

```
16 a = A(2);
17
18 disp(round(a*10^4)/10^4, "a =")
19 disp(round(b*10^4)/10^4, "b =")
```

Scilab code Exa 5.5 Method of Least Squares

```
1 //Example 5.5
2
3 clc
4 clear
5
6 x = 1:6;
7 y = [2.6 5.4 8.7 12.1 16 20.2];
8
9 X = x;
10 Y = y ./x;
11
12 n = length(Y);
13 M1 = [sum(Y); sum(X.*Y)];
14 M2 = [n sum(X); sum(X) sum(X.^2)];
15
16 A = M2\M1;
17
18 a = A(1);
19 b = A(2);
20
21 disp(round(a*10^5)/10^5, "a =")
22 disp(round(b*10^5)/10^5, "b =")
```

Scilab code Exa 5.6 Method of Least Squares

```
1 //Example 5.6
```

```

2
3 clc
4 clear
5
6 X = 1:0.2:2;
7 Y = [0.98 1.4 1.86 2.55 2.28 3.2];
8
9 n = length(Y);
10 M1 = [sum(X.^4) sum(X.^3) sum(X.^2); sum(X.^3) sum(X
    .^2) sum(X); sum(X.^2) sum(X) n];
11 M2 = [sum(X.^2 .* Y); sum(X.*Y); sum(Y)];
12 A = M1\M2;
13
14 a = A(1);
15 b = A(2);
16 c = A(3);
17
18 disp(round(a*10^4)/10^4, "a =")
19 disp(round(b*10^4)/10^4, "b =")
20 disp(round(c*10^4)/10^4, "c =")

```

Scilab code Exa 5.7 Method of Least Squares

```

1 //Example 5.7
2
3 clc
4 clear
5
6 x = 2:5;
7 y = [27.8 62.1 110 161];
8
9 X = log(x);
10 Y = log(y);
11
12 n = length(Y);

```

```

13 M1 = [sum(X.^2) sum(X); sum(X) n];
14 M2 = [sum(X.*Y); sum(Y)];
15 M = M1\M2;
16
17 b = M(1);
18 A = M(2);
19 a = exp(A);
20
21 disp(round(a*10^4)/10^4, "a =")
22 disp(round(b*10^4)/10^4, "b =")

```

Scilab code Exa 5.8 Principle of Least Squares

```

1 //Example 5.8
2
3 clc
4 clear
5
6 x = 1:4;
7 y = [1.65 2.7 4.5 7.35];
8
9 X = x;
10 Y = log10(y);
11
12 n = length(Y);
13 M1 = [sum(X.^2) sum(X); sum(X) n];
14 M2 = [sum(X.*Y); sum(Y)];
15 M = M1\M2;
16
17 B = M(1);
18 A = M(2);
19 a = 10^A;
20 b = B/log10(%e);
21
22 disp(round(a), "a =")

```



```
23 disp(round(b*10^4)/10^4, "b =")
```

Scilab code Exa 5.9 Method of Moments

```
1 //Example 5.9
2
3 clc
4 clear
5
6 x = 2:5;
7 y = [27 40 55 68];
8
9 delx = x(2) - x(1);
10 mu1 = delx * sum(y);
11 mu2 = delx * sum(x.*y);
12
13 n = length(y);
14 l = x(1) - delx/2;
15 u = x(n) + delx/2;
16
17 M1 = [integrate("x", 'x', l, u) u-1; integrate("x^2", 'x
    ', l, u) integrate("x", 'x', l, u)];
18 M2 = [mu1; mu2];
19 M = M1\M2;
20
21 a = M(1);
22 b = M(2);
23
24 disp(round(a*10^4)/10^4, "a =")
25 disp(round(b*10^4)/10^4, "b =")
```

Scilab code Exa 5.10 Method of Moments

```

1 //Example 5.10
2
3 clc
4 clear
5
6 x = 3:7;
7 y = [31.9 34.6 33.8 27 31.6];
8
9 delx = x(2) - x(1);
10 mu1 = delx * sum(y);
11 mu2 = delx * sum(x.*y);
12 mu3 = delx * sum(x^2 .*y);
13
14 n = length(y);
15 l = x(1) - delx/2;
16 u = x(n) + delx/2;
17
18 t0 = u-1;
19 t1 = integrate("x", 'x', l, u);
20 t2 = integrate("x^2", 'x', l, u);
21 t3 = integrate("x^3", 'x', l, u);
22 t4 = integrate("x^4", 'x', l, u);
23
24 M1 = [t2 t1 t0; t3 t2 t1; t4 t3 t2];
25 M2 = [mu1; mu2; mu3];
26 M1 = round(M1*10^2)/10^2;
27 M = M1\M2;
28
29 c = M(1);
30 b = M(2);
31 a = M(3);
32
33 disp(round(a*10^4)/10^4, "a =")
34 disp(round(b*10^4)/10^4, "b =")
35 disp(round(c*10^4)/10^4, "c =")

```

Chapter 6

Interpolation

Scilab code Exa 6.1 Forward Difference Table

```
1 //Example 6.1
2
3 clc
4 clear
5
6 x = 0.1:0.2:1.3;
7 y = [0.003 0.067 0.148 0.248 0.37 0.518 0.697];
8
9 n = length(x);
10 del = %nan*ones(n,6);
11 del(:,1) = y';
12 for j = 2:6
13     for i = 1:n-j+1
14         del(i,j) = del(i+1,j-1) - del(i,j-1);
15     end
16 end
17 del = [x' del];
18 del = round(del*10^3)/10^3;
19 mprintf("%5s %7s %8s %9s %8s %8s %8s", 'x', 'y', 'dy', '
    d2y', 'd3y', 'd4y', 'd5y')
20 disp(del)
```

Scilab code Exa 6.2 Expression for Finite Difference Elements

```
1 // Example 6.2
2 // This is an analytical problem and need not be
  coded.
```

Scilab code Exa 6.3 Expression for Finite Difference Elements

```
1 // Example 6.3
2 // This is an analytical problem and need not be
  coded.
```

Scilab code Exa 6.4 Expression for Finite Difference Elements

```
1 // Example 6.4
2 // This is an analytical problem and need not be
  coded.
```

Scilab code Exa 6.5 Proof of Relation

```
1 // Example 6.5
2 // This is an analytical problem and need not be
  coded.
```

Scilab code Exa 6.6 Proofs of given Relations

```
1 // Example 6.6
2 // This is an analytical problem and need not be
  coded.
```

Scilab code Exa 6.7 Proof for Commutation of given Operations

```
1 // Example 6.7
2 // This is an analytical problem and need not be
  coded.
```

Scilab code Exa 6.8 Newton Forward Difference Interpolation Formula

```
1 //Example 6.8
2
3 clc
4 clear
5
6 x = 10:10:50;
7 y = [46 66 81 93 101];
8
9 n = length(x);
10 del = %nan*ones(n,5);
11 del(:,1) = y';
12 for j = 2:5
13     for i = 1:n-j+1
14         del(i,j) = del(i+1,j-1) - del(i,j-1);
15     end
16 end
17 del(:,1) = [];
18
19 X = 15; //input
```

```

20 for i = 1:n
21     if X>x(i) then
22         h = x(i+1) - x(i);
23         p = (X-x(i)) / h;
24         x0 = x(i);
25         y0 = y(i);
26         dely0 = del(i,:);
27         dely0(isnan(y0)) = [];
28     end
29 end
30
31 Y = y0;
32
33 for i = 1:length(dely0)
34     t = 1;
35     for j = 1:i
36         t = t * (p-j+1);
37     end
38     Y = Y + t*dely0(i)/factorial(i);
39 end
40 Y = round(Y*10^4)/10^4;
41 disp(Y,"f(15) = ")

```

Scilab code Exa 6.9 Newton Forward Difference Interpolation Formula

```

1 //Example 6.9
2
3 clc
4 clear
5
6 x = 0.1:0.1:0.5;
7 y = [1.4 1.56 1.76 2 2.28];
8
9 n = length(x);
10 del = %nan*ones(n,5);

```

```

11 del(:,1) = y';
12 for j = 2:5
13     for i = 1:n-j+1
14         del(i,j) = del(i+1,j-1) - del(i,j-1);
15     end
16 end
17 del(:,1) = [];
18
19 X = poly(0, "X");
20 h = x(2) - x(1);
21 p = (X-x(1)) / h;
22 x0 = x(1);
23 y0 = y(1);
24 dely0 = del(1,:);
25
26 Y = y0;
27
28 for i = 1:length(dely0)
29     t = 1;
30     for j = 1:i
31         t = t * (p-j+1);
32     end
33     Y = Y + t*dely0(i)/factorial(i);
34 end
35 Y = round(Y*10^2)/10^2;
36 disp(Y,"Required Newton''s Interpolating Polynomial:
    ")

```

Scilab code Exa 6.10 Newton Forward Difference Interpolation Formula

```

1 //Example 6.10
2
3 clc
4 clear
5

```

```

6 x = 1:5;
7 Y = poly(0, "Y");
8 y = [2 5 7 Y 32];
9
10 n = length(x);
11 del = %nan*ones(n,5);
12 del(:,1) = y';
13 for j = 2:5
14     for i = 1:n-j+1
15         del(i,j) = del(i+1,j-1) - del(i,j-1);
16     end
17 end
18 del(:,1) = [];
19
20 // del4 = 0
21
22 y0 = del(:,4);
23 y0(isnan(y0)) = [];
24 Y = roots(y0)
25 disp(Y,"Missing value f(x3) = ")

```

Scilab code Exa 6.11 Newton Forward Difference Interpolation Formula

```

1 //Example 6.11
2
3 clc
4 clear
5
6 x = 0:5;
7 y = [-3 3 11 27 57 107];
8
9 n = length(x);
10 del = %nan*ones(n,4);
11 del(:,1) = y';
12 for j = 2:4

```



```

13     for i = 1:n-j+1
14         del(i,j) = del(i+1,j-1) - del(i,j-1);
15     end
16 end
17 del(:,1) = [];
18
19 X = poly(0, "x");
20 h = x(2) - x(1);
21 p = (X-x(1)) / h;
22 x0 = x(1);
23 y0 = y(1);
24 dely0 = del(1,:);
25
26 Y = y0;
27
28 for i = 1:length(dely0)
29     t = 1;
30     for j = 1:i
31         t = t * (p-j+1);
32     end
33     Y = Y + t*dely0(i)/factorial(i);
34 end
35 disp(Y,"Required cubic polynomial:")

```

Scilab code Exa 6.12 Newton Backward Difference Interpolation Formula

```

1 //Example 6.12
2
3 clc
4 clear
5
6 x = 1:8;
7 y = x^3;
8
9 n = length(x);

```

```

10 del = %nan*ones(n,4);
11 del(:,1) = y';
12 for j = 2:4
13     for i = 1:n-j+1
14         del(i+j-1,j) = del(i+j-1,j-1) - del(i+j-2,j
            -1);
15     end
16 end
17
18 X = 7.5;
19 h = x(2) - x(1);
20 p = (X-x(n)) / h;
21 xn = x(n);
22 yn = y(n);
23 delyn = del(n,:);
24
25 Y = 0;
26
27 for i = 0:length(delyn)-1
28     t = 1;
29     for j = 1:i
30         t = t * (p+j-1);
31     end
32     Y = Y + t*delyn(i+1)/factorial(i);
33 end
34 disp(Y,"y(7.5) = ")

```

Scilab code Exa 6.13 Newton Backward Difference Interpolation Formula

```

1 //Example 6.13
2
3 clc
4 clear
5
6 x = 1974:2:1982;

```

```

7 y = [40 43 48 52 57];
8
9 n = length(x);
10 del = %nan*ones(n,5);
11 del(:,1) = y';
12 for j = 2:5
13     for i = 1:n-j+1
14         del(i+j-1,j) = del(i+j-1,j-1) - del(i+j-2,j
15             -1);
16     end
17 end
18 X = 1979;
19 h = x(2) - x(1);
20 p = (X-x(n)) / h;
21 xn = x(n);
22 yn = y(n);
23 delyn = del(n,:);
24
25 Y = 0;
26
27 for i = 0:length(delyn)-1
28     t = 1;
29     for j = 1:i
30         t = t * (p+j-1);
31     end
32     Y = Y + t*delyn(i+1)/factorial(i);
33 end
34 Y = round(Y*10^4)/10^4;
35 disp(Y,"Estimated sales for the year 1979: ")

```

Scilab code Exa 6.14 Lagrange Interpolation Formula

```

1 //Example 6.14
2

```

```

3  clc
4  clear
5
6  x = [1 3 4 6];
7  y = [-3 0 30 132];
8
9  n = length(x);
10 Y = 0;
11 X = poly(0, "X");
12 //X = 5;
13 for i = 1:n
14     t = x;
15     t(i) = [];
16     p = 1;
17     for j = 1:length(t)
18         p = p * (X-t(j))/(x(i)-t(j));
19     end
20     Y = Y + p*y(i);
21 end
22 Y5 = horner(Y,5);
23 disp(Y5,"y(5) = ")

```

Scilab code Exa 6.15 Lagrange Interpolation Formula

```

1  //Example 6.15
2
3  clc
4  clear
5
6  x = [1 2 5];
7  y = [1 4 10];
8
9  n = length(x);
10 Y = 0;
11 X = poly(0, "X");

```

```

12 //X = 5;
13 for i = 1:n
14     t = x;
15     t(i) = [];
16     p = 1;
17     for j = 1:length(t)
18         p = p * (X-t(j))/(x(i)-t(j));
19     end
20     Y = Y + p*y(i);
21 end
22 Y5 = horner(Y,3);
23 disp(Y5," f(3) = ")

```

Scilab code Exa 6.16 Lagrange and Newton Divided Difference Interpolation Formulae

```

1 //Example 6.16
2
3 clc
4 clear
5
6 x = [0 1 2 4];
7 y = [1 1 2 5];
8
9 n = length(x);
10 del = %nan*ones(n,4);
11 del(:,1) = y';
12 for j = 2:4
13     for i = 1:n-j+1
14         del(i,j) = (del(i+1,j-1) - del(i,j-1)) / (x(
15             i+j-1) - x(i));
16     end
17 end
18 del(:,1) = [];
19

```

```

19 Y = 0;
20 X = poly(0, "X");
21 for i = 1:n
22     t = x;
23     t(i) = [];
24     p = 1;
25     for j = 1:length(t)
26         p = p * (X-t(j))/(x(i)-t(j));
27     end
28     Y = Y + p*y(i);
29 end
30 disp(round(Y*10^4)/10^4,"Interpolating polynomial:")

```

Scilab code Exa 6.17 Newton Divided Difference Interpolation Formulae

```

1 //Example 6.17
2
3 clc
4 clear
5
6 x = [0 1 4];
7 y = [2 1 4];
8
9 n = length(x);
10 del = %nan*ones(n,3);
11 del(:,1) = y';
12 for j = 2:3
13     for i = 1:n-j+1
14         del(i,j) = (del(i+1,j-1) - del(i,j-1)) / (x(
15             i+j-1) - x(i));
16     end
17 end
18 del(:,1) = [];
19 Y = 0;

```

```

20 X = 2;
21 for i = 1:n
22     t = x;
23     t(i) = [];
24     p = 1;
25     for j = 1:length(t)
26         p = p * (X-t(j))/(x(i)-t(j));
27     end
28     Y = Y + p*y(i);
29 end
30 disp(Y,"y(2) = ")

```

Scilab code Exa 6.18 Identity Proof for Newton and Lagrange Interpolation Formulae

```

1 // Example 6.18
2 // This is an analytical problem and need not be
   coded.

```

Scilab code Exa 6.19 Interpolation in Two Dimensions

```

1 //Example 6.19
2
3 clc
4 clear
5
6 x = 0:4;
7 n = length(x);
8 f = "X^2 + Y^2 - Y";
9 tab = %nan*ones(n,5);
10
11 for j = 0:4
12     fj = strsubst(f, 'Y', 'j ');

```

```

13     for i = 1:n
14         tab(i,j+1) = eval(strsubst(fj, 'X', 'x(i)'));
15     end
16 end
17 //tab(:,1) = [];
18 mprintf("%4s %6s %6s %6s %6s %6s", 'x', 'y=0', 'y=1', 'y
    =2', 'y=3', 'y=4')
19 disp([(0:4)' tab])
20 tab2 = tab(2:4,2:4)';
21 n1 = length(tab2(:,1));
22 y = 2:4;
23
24 del1 = %nan*ones(n1,3);
25 del1(:,1) = tab2(:,1);
26 for j = 2:4
27     for i = 1:n1-j+1
28         del1(i,j) = del1(i+1,j-1) - del1(i,j-1);
29     end
30 end
31
32 del2 = %nan*ones(n1,3);
33 del2(:,1) = tab2(:,2);
34 for j = 2:4
35     for i = 1:n1-j+1
36         del2(i,j) = del2(i+1,j-1) - del2(i,j-1);
37     end
38 end
39
40 del3 = %nan*ones(n1,3);
41 del3(:,1) = tab2(:,3);
42 for j = 2:4
43     for i = 1:n1-j+1
44         del3(i,j) = del3(i+1,j-1) - del3(i,j-1);
45     end
46 end
47
48 y0 = y(1);
49 Y = 3.5;

```



```

50 hy = y(2) - y(1);
51 py = (Y-y0)/hy;
52
53 f1y = 0;
54 del1y0 = del1(1,:);
55 for i = 0:length(del1y0)-1
56     t = 1;
57     for j = 1:i
58         t = t * (py-j+1);
59     end
60     f1y = f1y + t*del1y0(i+1)/factorial(i);
61 end
62
63 f2y = 0;
64 del2y0 = del2(1,:);
65 for i = 0:length(del2y0)-1
66     t = 1;
67     for j = 1:i
68         t = t * (py-j+1);
69     end
70     f2y = f2y + t*del2y0(i+1)/factorial(i);
71 end
72
73 f3y = 0;
74 del3y0 = del3(1,:);
75 for i = 0:length(del3y0)-1
76     t = 1;
77     for j = 1:i
78         t = t * (py-j+1);
79     end
80     f3y = f3y + t*del3y0(i+1)/factorial(i);
81 end
82
83 del = %nan*ones(n1,3);
84 del(:,1) = [f1y; f2y; f3y];
85 for j = 2:4
86     for i = 1:n1-j+1
87         del(i,j) = del(i+1,j-1) - del(i,j-1);

```

```

88     end
89 end
90
91 f = 0;
92 X = 2.5;
93 x0 = x(2);
94 hx = x(2) - x(1);
95 px = (X-x0)/hx;
96 del0 = del(1,:);
97 for i = 0:length(del0)-1
98     t = 1;
99     for j = 1:i
100         t = t * (px-j+1);
101     end
102     f = f + t*del0(i+1)/factorial(i);
103 end
104 disp(f,"f(2.5,3.5) = ")

```

Scilab code Exa 6.20 Cubic Spline Curve

```

1 //Example 6.20
2
3 clc
4 clear
5
6 function [p] = cubicsplin(x,y)
7 // Fits point data to cubic spline fit
8
9 n = length(x);
10 a = y(1:n-1); // Spline Initials
11
12 M1 = zeros(3*(n-1));
13 M2 = zeros(3*(n-1),1);
14 // Point Substitutions
15 for i = 1:n-1

```

```

16     M1(i,i) = x(i+1) - x(i);
17     M1(i,i+n-1) = (x(i+1) - x(i))^2;
18     M1(i,i+2*(n-1)) = (x(i+1) - x(i))^3;
19     M2(i) = y(i+1) - y(i);
20 end
21
22 // Knot equations
23 for i = 1:n-2
24     // Derivative (S') continuity
25     M1(i+n-1,i) = 1;
26     M1(i+n-1,i+1) = -1;
27     M1(i+n-1,i+n-1) = 2*(x(i+1)-x(i));
28     M1(i+n-1,i+2*(n-1)) = 3*(x(i+1)-x(i))^2;
29     // S'' continuity
30     M1(i+2*n-3,i+n-1) = 2;
31     M1(i+2*n-3,i+n) = -2;
32     M1(i+2*n-3,i+2*(n-1)) = 6*(x(i+1)-x(i));
33 end
34 // Given BC
35 M1(3*n-4,n) = 1;
36 M1(3*n-3,2*n-2) = 1;
37 M1(3*n-3,3*n-3) = 3*(3-2);
38
39 var = M1\M2;
40 var = round(var);
41 b = var(1:n-1);
42 c = var(n:2*(n-1));
43 d = var(2*(n-1)+1:3*(n-1));
44 p = [d c b a(:)];
45 endfunction
46
47 x = 0:3;
48 y = [1 4 0 -2];
49 p = cubicsplin(x,y);
50 for i = 1:length(p(:,1))
51     disp(strcat(["S",string(i-1)," (x) ="]))
52     disp(poly(p(i,:), "X", ["coeff"]))
53 end

```

Scilab code Exa 6.21 Cubic Spline Curve

```
1 //Example 6.21
2
3 clc
4 clear
5
6 function [p] = cubicsplin(x,y)
7 // Fits point data to cubic spline fit
8
9 n = length(x);
10 a = y(1:n-1); // Spline Initials
11
12 M1 = zeros(3*(n-1));
13 M2 = zeros(3*(n-1),1);
14 // Point Substitutions
15 for i = 1:n-1
16     M1(i,i) = x(i+1) - x(i);
17     M1(i,i+n-1) = (x(i+1) - x(i))^2;
18     M1(i,i+2*(n-1)) = (x(i+1) - x(i))^3;
19     M2(i) = y(i+1) - y(i);
20 end
21
22 // Knot equations
23 for i = 1:n-2
24     // Derivative (S') continuity
25     M1(i+n-1,i) = 1;
26     M1(i+n-1,i+1) = -1;
27     M1(i+n-1,i+n-1) = 2*(x(i+1)-x(i));
28     M1(i+n-1,i+2*(n-1)) = 3*(x(i+1)-x(i))^2;
29     // S'' continuity
30     M1(i+2*n-3,i+n-1) = 2;
31     M1(i+2*n-3,i+n) = -2;
32     M1(i+2*n-3,i+2*(n-1)) = 6*(x(i+1)-x(i));
```

```

33 end
34 // Given BC
35 M1(3*n-4,1) = 1;
36 M1(3*n-3,n-1) = 1;
37 M1(3*n-3,2*n-2) = 2*(3-2);
38 M1(3*n-3,3*n-3) = 3*(3-2)^2;
39 M2(3*n-4) = 2;
40 M2(3*n-3) = 2;
41
42 var = M1\M2;
43 var = round(var);
44 b = var(1:n-1);
45 c = var(n:2*(n-1));
46 d = var(2*(n-1)+1:3*(n-1));
47
48 p = [a(:) b c d];
49 endfunction
50
51 x = 0:3;
52 y = [1 4 0 -2];
53 p = cubicsplin(x,y);
54 for i=1:length(p(:,1))
55     disp(strcat(["S",string(i-1)," (x) = "]))
56     disp(poly(p(i,:), "x", ["coeff"]))
57 end

```

Scilab code Exa 6.22 Minima of a Tabulated Function

```

1 //Example 6.22
2
3 clc
4 clear
5
6 x = 3:8;
7 y = [0.205 0.24 0.259 0.262 0.25 0.224];

```

```

8
9 n = length(x);
10 del = %nan*ones(n,5);
11 del(:,1) = y';
12 for j = 2:5
13     for i = 1:n-j+1
14         del(i,j) = del(i+1,j-1) - del(i,j-1);
15     end
16 end
17
18 X = poly(0, "X");
19 x0 = x(1);
20 y0 = y(1);
21 h = x(2) - x(1);
22 p = (X-x0)/h;
23 del0 = del(1,:);
24 del0 = round(del0*10^4)/10^4;
25 del0 = del0(1:find(del0==0)-1);
26
27 Y = 0;
28 for i = 0:length(del0)-1
29     t = 1;
30     for j = 1:i
31         t = t * (p-j+1);
32     end
33     Y = Y + t*del0(i+1)/factorial(i);
34 end
35 disp(Y,"y = ")
36
37 dydx = derivat(Y);
38 minx = roots(dydx);
39 miny = round(horner(Y,minx)*10^5)/10^5;
40 disp(minx,"min_x = ")
41 disp(miny,"min_y = ")
42 //min_y value is incorrectly displayed in textbook
    as 0.25425 instead of 0.26278

```

Scilab code Exa 6.23 Maxima of a Tabulated Function

```
1 //Example 6.23
2
3 clc
4 clear
5
6 x = [-1 1 2 3];
7 y = [-21 15 12 3];
8
9 n = length(x);
10 X = poly(0, "X");
11 Y = 0;
12 for i = 1:n
13     t = x;
14     t(i) = [];
15     p = 1;
16     for j = 1:length(t)
17         p = p * (X-t(j))/(x(i)-t(j));
18     end
19     Y = Y + p*y(i);
20 end
21
22 dydx = derivat(Y);
23 extx = real(roots(dydx));
24 extx = round(extx*10^4)/10^4;
25 d2ydx = derivat(dydx);
26
27 if horner(d2ydx,extx(1)) < 0 then
28     maxx = extx(1);
29     maxy = horner(Y,maxx);
30 else
31     maxx = extx(2);
32     maxy = horner(Y,maxx);
```

```
33 end
34 maxy = round(maxy*10^4)/10^4;
35 disp(maxx,"max_x = ")
36 disp(maxy,"max_y = ")
```

Scilab code Exa 6.24 Determination of Function Value

```
1 //Example 6.24
2
3 clc
4 clear
5
6 x = 1:3:10;
7 F = [500426 329240 175212 40365];
8
9 n = length(x);
10 del = %nan*ones(n,4);
11 del(:,1) = F';
12 for j = 2:4
13     for i = 1:n-j+1
14         del(i,j) = del(i+1,j-1) - del(i,j-1);
15     end
16 end
17
18 del0 = del(1,:);
19 X = 2;
20 x0 = x(1);
21 h = x(2) - x(1);
22 p = (X-x0) / h;
23 F2 = 0;
24 for i = 0:length(del0)-1
25     t = 1;
26     for j = 1:i
27         t = t * (p-j+1);
28     end
```



```
29     F2 = F2 + t*del0(i+1)/factorial(i);
30 end
31
32 f2 = F(1) - F2;
33 disp(f2," f(2) = ")
```

Chapter 7

Numerical Differentiation and Integration

Scilab code Exa 7.1 Determination of Differential Function Value

```
1 //Example 7.1
2
3 clc
4 clear
5
6 x = 0:0.2:1;
7 y = [1 1.16 3.56 13.96 41.96 101];
8
9 n = length(x);
10 del = %nan*ones(n,6);
11 del(:,1) = y';
12 for j = 2:6
13     for i = 1:n-j+1
14         del(i,j) = del(i+1,j-1) - del(i,j-1);
15     end
16 end
17 del = round(del*10^2)/10^2;
18 mprintf("%5s %6s %9s %8s %8s %8s %7s", 'x', 'y', 'dy', 'd2y', 'd3y', 'd4y', 'd5y')
```

```

19 disp([x' del])
20
21 h = x(2) - x(1);
22 del0 = del(1,:);
23 del1 = del(2,:);
24
25 df1 = (del1(2) - del1(3)/2 + del1(4)/3 - del1(5)/4)
      / h;
26 d2f0 = (del0(2) - del0(3) + del0(4)*11/12 - del0(5)
      *5/6) / h^2;
27 disp(round(d2f0*10^1)/10^1,"f''''(0) = ")
28 disp(round(df1*10)/10,"f''(0.2) = ")

```

Scilab code Exa 7.2 Determination of Differential Function Value

```

1 //Example 7.2
2
3 clc
4 clear
5
6 x = 1.4:0.2:2.2;
7 y = [4.0552 4.953 6.0496 7.3891 9.025];
8
9 n = length(x);
10 del = %nan*ones(n,5);
11 del(:,1) = y';
12 for j = 2:5
13     for i = 1:n-j+1
14         del(i+j-1,j) = del(i+j-1,j-1) - del(i+j-2,j
15             -1);
16     end
17 end
18 mprintf("%5s %6s %10s %10s %9s %9s", 'x', 'y', 'dy', '
19     d2y', 'd3y', 'd4y')
20 disp([x' del])

```

```

19
20 h = x(2) - x(1);
21 deln = del(5,:);
22
23 dfn = (deln(2) + deln(3)/2 + deln(4)/3 + deln(5)/4)
      / h;
24 d2fn = (deln(3) + deln(4) + deln(5)*11/12) / h^2;
25 dfn = round(dfn*10^4)/10^4;
26 d2fn = round(d2fn*10^4)/10^4;
27 disp(dfn,"y''(2.2) = ")
28 disp(d2fn,"y''''(2.2) = ")

```

Scilab code Exa 7.3 Determination of Differential Function Value

```

1 //Example 7.3
2
3 clc
4 clear
5
6 x = 0:4;
7 y = [6.9897 7.4036 7.7815 8.1281 8.451];
8
9 n = length(x);
10 del = %nan*ones(n,5);
11 del(:,1) = y';
12 for j = 2:6
13     for i = 1:n-j+1
14         del(i,j) = del(i+1,j-1) - del(i,j-1);
15     end
16 end
17 del(:,1) = [];
18 n0 = length(del(1,:));
19
20 X = 2;
21 i = find(x==X);

```

```

22 dowy = 0;
23
24 for j = 1:n0
25     if j==2*int(j/2) then
26         add = del(i,j);
27     else
28         add = (del(i-1,j) + del(i,j))/2;
29         i = i-1;
30         if i==0 then
31             break
32         end
33     end
34
35     if add == %nan then
36         break
37     else
38         dowy(j) = add;
39     end
40 end
41 mprintf("%5s %6s %10s %9s %9s %9s", 'x', 'y', 'dy', 'd2y
    ', 'd3y', 'd4y')
42 disp(['x' 'y' 'del'])
43
44 mu = 1;
45 h = x(2) - x(1);
46 dy2 = mu/h*(dowy(1) - 1/6*dowy(3));
47 d2y2 = mu/h^2*(dowy(2) - 1/12*dowy(4));
48 dy2 = round(dy2*10^4)/10^4;
49 d2y2 = round(d2y2*10^4)/10^4;
50
51 disp(dy2, "y''(2) = ")
52 disp(d2y2, "y''''(2) = ")

```

Scilab code Exa 7.4 Determination of Differential Function Value

```

1 //Example 7.4
2
3 clc
4 clear
5
6 x = [0.15 0.21 0.23 0.27 0.32 0.35];
7 y = [0.1761 0.3222 0.3617 0.4314 0.5051 0.5441];
8
9 n = length(x);
10 del = %nan*ones(n,6);
11 del(:,1) = y';
12 for j = 2:6
13     for i = 1:n-j+1
14         del(i,j) = (del(i+1,j-1) - del(i,j-1)) / (x(
15             i+j-1)-x(i));
16     end
17 end
18 del(:,1) = [];
19 del = round(del*10^3)/10^3;
20 mprintf("%5s %6s %10s %10s %8s %9s %9s", 'x', 'y', 'dy',
21     'd2y', 'd3y', 'd4y', 'd5y')
22 disp([x' y' del])
23 X = poly(0, "X");
24 del0 = del(1,:);
25 y0 = y(1);
26 Y = y0;
27 for i = 1:length(del0)
28     p = 1;
29     for j = 1:i
30         p = p*(X-x(j));
31     end
32     Y = Y + p*del0(i);
33 end
34 dydx = derivat(Y);
35 d2ydx = derivat(dydx);
36 XX = 0.25;

```

```

37 dy = horner(dydx,XX);
38 d2y = horner(d2ydx,XX);
39
40 disp(round(dy*10^4)/10^4,"y''(0.25) = ")
41 disp(d2ydx,"y''''(x) = ")
42 disp(d2y,"y''''(0.25) = ")
43 //The constant term in y''(x) is incorrectly
    computed to -91.7 instead of -97.42 in the text.

```

Scilab code Exa 7.5 Richardson Extrapolation Limit

```

1 //Example 7.5
2
3 clc
4 clear
5
6 function [f] = y(x)
7     f = -1/x;
8 endfunction
9
10 H = [0.0128 0.0064 0.0032];
11 n = length(H);
12 x = 0.05;
13 h = H(1);
14 Fh = (y(x+h) - y(x-h)) / (2*h);
15 Fh2 = (y(x+h/2) - y(x-h/2)) / (h);
16 Fh4 = (y(x+h/4) - y(x-h/4)) / (h/2);
17
18 F1h2 = (4*Fh2 - Fh) / (4-1);
19 F1h4 = (4*Fh4 - Fh2) / (4-1);
20 F2h4 = (4^2*F1h4 - F1h2) / (4^2-1);
21 del = %nan*ones(n,3);
22 del(:,1) = [Fh Fh2 Fh4]';
23 del(1:2,2) = [F1h2 F1h4]';
24 del(1,3) = F2h4;

```

```

25
26 disp(del(1,n),"y'(0.05) = ")
27 Exact = 1/x^2;
28 disp(Exact,"Exact Value:")

```

Scilab code Exa 7.6 Integral using Trapezoidal and Simpson One Third Rule

```

1 //Example 7.6
2
3 clc
4 clear
5
6 function [I] = trap (fun,a,b,n)
7 // Integrate the function over the interval using
   Trapezoidal Formula
8 // trap (fun,a,b,n)
9 // fun - function to be integrated
10 // a - lower limit of integration
11 // b - upper limit of integration
12 // n - No. of times trapezoidal rule needs to be
   performed
13
14 N = n + 1; // N - total no. of points
15 h = (b-a) / (N-1);
16 x = linspace(a,b,N);
17 y = fun(x);
18
19 sum1 = y(1) + 2 * sum(y(2:N-1)) + y(N);
20 I = h * sum1 / 2; // Trapezoidal
   Integral Value
21 endfunction
22
23 function [I] = simp13 (fun,a,b,n)
24 // Integrate the function over the interval using

```



```

    Simpson's 1/3rd rule
25 // simp13 (fun,a,b,n)
26 // fun - function to be integrated
27 // a - lower limit of integration
28 // b - upper limit of integration
29 // n - No. of times simpson's 1/3rd rule needs to be
    performed
30
31 N = 2 * n + 1;          // N - total no. of points
32 h = (b-a) / (N-1);
33 x = linspace(a,b,N);
34 y = fun(x);
35
36 sum1 = y(1) + 4 * sum(y(2:2:N-1)) + 2 * sum(y(3:2:N
    -2)) + y(N);
37 I = h* sum1 / 3;          // Simpson's 1/3
    rd Integral Value
38 endfunction
39
40 n = 6;
41 ntrap = n;
42 ns13 = n/2;
43 I = [trap(sin,0,%pi,ntrap); simp13(sin,0,%pi,ns13)];
44 I = round(I*10^4)/10^4;
45 true = integrate('sin(x)', 'x', 0, %pi);
46 err = abs(true - I) / true*100;
47 err = round(err*100)/100;
48
49 disp(I(1), "y_trap = ")
50 disp(I(2), "y_simp13 = ")
51 disp(err(1), "error_trap = ")
52 disp(err(2), "error_simp13 = ")

```

Scilab code Exa 7.7 Integral using Simpson One Third Rule

```

1 //Example 7.7
2
3 clc
4 clear
5
6 function [I] = simp13 (fun,a,b,n)
7 // Integrate the function over the interval using
   Simpson's 1/3rd rule
8 // simp13 (fun,a,b,n)
9 // fun – function to be integrated
10 // a – lower limit of integration
11 // b – upper limit of integration
12 // n – No. of times simpson's 1/3rd rule needs to be
   performed
13
14 N = 2 * n + 1;          // N – total no. of points
15 h = (b-a) / (N-1);
16 x = linspace(a,b,N);
17 y = fun(x);
18
19 sum1 = y(1) + 4 * sum(y(2:2:N-1)) + 2 * sum(y(3:2:N
   -2)) + y(N);
20 I = h* sum1 / 3;          // Simpson's 1/3
   rd Integral Value
21 endfunction
22
23 n = 8;
24 ns13 = n/2;
25 I = simp13(log,1,5,ns13);
26 I = round(I*10^4)/10^4;
27 deff(' [y] = true(x)', ['y = x * log(x) - x']);
28 trueVal = true(5) - true(1);
29 err = abs(trueVal - I) / trueVal*100;
30 err = round(err*100)/100;
31
32 disp(I,"y_simp13 = ")
33 disp(trueVal,"Actual Integral = ")
34 disp(err,"error_simp13 = ")

```

Scilab code Exa 7.8 Integral using Trapezoidal and Simpson One Third Rule

```
1 //Example 7.8
2
3 clc
4 clear
5
6 function [I] = trap (fun,a,b,n)
7 // Integrate the function over the interval using
   Trapezoidal Formula
8 // trap (fun,a,b,n)
9 // fun - function to be integrated
10 // a - lower limit of integration
11 // b - upper limit of integration
12 // n - No. of times trapezoidal rule needs to be
   performed
13
14 N = n + 1; // N - total no. of points
15 h = (b-a) / (N-1);
16 x = linspace(a,b,N);
17 y = fun(x);
18
19 sum1 = y(1) + 2 * sum(y(2:N-1)) + y(N);
20 I = h * sum1 / 2; // Trapezoidal
   Integral Value
21 endfunction
22
23 function [I] = simp13 (fun,a,b,n)
24 // Integrate the function over the interval using
   Simpson's 1/3rd rule
25 // simp13 (fun,a,b,n)
26 // fun - function to be integrated
27 // a - lower limit of integration
```

```

28 // b – upper limit of integration
29 // n – No. of times simpson's 1/3rd rule needs to be
    performed
30
31 N = 2 * n + 1;          // N – total no. of points
32 h = (b-a) / (N-1);
33 x = linspace(a,b,N);
34 y = fun(x);
35
36 sum1 = y(1) + 4 * sum(y(2:2:N-1)) + 2 * sum(y(3:2:N
    -2)) + y(N);
37 I = h* sum1 / 3;          // Simpson's 1/3
    rd Integral Value
38 endfunction
39
40 function [f] = fun1(x)
41     f = 1 ./ (1+x^2);
42 endfunction
43
44
45 n = 4;
46 ntrap = n;
47 ns13 = n/2;
48 I = [trap(fun1,0,1,ntrap); simp13(fun1,0,1,ns13)];
49 I = round(I*10^4)/10^4;
50 true = intg(0,1,fun1);
51
52 disp(I(1),"y_trap = ")
53 disp(I(2),"y_simp13 = ")
54 disp(I(2)*4,"Approx pi = ")

```

Scilab code Exa 7.9 Integral using Simpson One Third Rule

```

1 //Example 7.9
2

```

```

3  clc
4  clear
5
6  function [I] = simp13 (fun,a,b,n)
7  // Integrate the function over the interval using
   Simpson's 1/3rd rule
8  // simp13 (fun,a,b,n)
9  // fun - function to be integrated
10 // a - lower limit of integration
11 // b - upper limit of integration
12 // n - No. of times simpson's 1/3rd rule needs to be
   performed
13
14 N = 2 * n + 1;          // N - total no. of points
15 h = (b-a) / (N-1);
16 x = linspace(a,b,N);
17 y = fun(x);
18
19 sum1 = y(1) + 4 * sum(y(2:2:N-1)) + 2 * sum(y(3:2:N
   -2)) + y(N);
20 I = h* sum1 / 3;          // Simpson's 1/3
   rd Integral Value
21 endfunction
22
23 function [f] = fun1(x)
24     f = sqrt(2/%pi)*exp(-x^2/2);
25 endfunction
26
27 h = 0.125;
28 n = (1-0)/h;
29 ns13 = n/2;
30 I = simp13(fun1,0,1,ns13);
31 I = round(I*10^4)/10^4;
32
33 disp(I,"Integral value , I = ")

```

Scilab code Exa 7.10 Integral using Simpson One Third Rule

```
1 //Example 7.10
2
3 clc
4 clear
5
6 t = 0:10:80;
7 a = [30 31.63 33.34 35.47 37.75 40.33 43.25 46.69
      50.67];
8
9 h = t(2) - t(1);
10 n = length(t);
11
12 Is13 = a(1);
13 for i = 2:n-1
14     rem2 = i-fix(i./2).*2;
15     if rem2 == 0 then
16         Is13 = Is13 + 4*a(i);
17     else
18         Is13 = Is13 + 2*a(i);
19     end
20 end
21 Is13 = (Is13 + a(n))/10^3;
22 Is13 = round(h/3*Is13*10^4)/10^4;
23 disp(strcat(["v = ",string(Is13)," km/s"]))
```

Scilab code Exa 7.11 Romberg Integration Method

```
1 //Example 7.11
2
3 clc
```

```

4 clear
5
6 x = 1:0.1:1.8;
7 x = round(x*10)/10;
8 y = [1.543 1.669 1.811 1.971 2.151 2.352 2.577 2.828
      3.107];
9 n = length(x);
10 x0 = x(1);
11 xn = x(n);
12
13 N = [1 2 4 8]
14 for j = 1:length(N)
15     h = (xn - x0)./N(j);
16     I = y(1);
17     for xx = x0+h:h:xn-h
18         xx = round(xx*10)/10;
19         I = I + 2*y(x==xx);
20     end
21     Itrap(j) = h/2*(I + y(n));
22     IRomb(1) = Itrap(1);
23     if j~=1 then
24         IRomb(j) = (4^(j-1)*Itrap(j)-IRomb(j-1))
                /(4^(j-1)-1);
25     end
26 end
27 IRomb = round(IRomb*10^5)/10^5;
28
29 disp(Itrap(length(N)), "Integral using Trapezoidal
      rule:")
30 disp(IRomb(length(N)), "Integral using Romberg's
      formula:")
31 //In third step of computation of integral using
      Romberg's formula, author mistakenly took the
      1.7672 instead of 1.7684 which resulted in a
      difference

```

Scilab code Exa 7.12 Double Integral using Trapezoidal Rule

```
1 //Example 7.12
2
3 clc
4 clear
5
6 function [f] = fun1(x,y)
7     f = 1 / (x+y);
8 endfunction
9
10 x = 1:0.25:2;
11 y = x;
12
13 m = length(x);
14 n = length(y);
15
16 del = %nan*ones(m,n);
17 for j = 1:n
18     for i = 1:m
19         del(i,j) = fun1(x(i),y(j));
20     end
21 end
22
23 hx = x(2) - x(1);
24 for i = 1:m
25     I = del(i,1);
26     for j = 2:n-1
27         I = I + 2*del(i,j);
28     end
29     Itrap1(i) = hx/2 * (I+del(i,n));
30 end
31 Itrap1 = round(Itrap1*10^4)/10^4;
32
```



```

33 hy = y(2) - y(1);
34 Itrap2 = Itrap1(1)
35 for i = 2:n-1
36     Itrap2 = Itrap2 + 2* Itrap1(i);
37 end
38 Itrap2 = round(hy/2*(Itrap2+Itrap1(m))*10^4)/10^4;
39 disp(Itrap2,"I = ")

```

Scilab code Exa 7.13 Double Integral using Trapezoidal Rule

```

1 //Example 7.13
2
3 clc
4 clear
5
6 function [f] = fun1(x,y)
7     f = sqrt(sin(x+y));
8 endfunction
9
10 x = 0:%pi/8:%pi/2;
11 y = x;
12
13 m = length(x);
14 n = length(y);
15
16 del = %nan*ones(m,n);
17 for j = 1:n
18     for i = 1:m
19         del(i,j) = fun1(x(i),y(j));
20     end
21 end
22
23 hx = x(2) - x(1);
24 for i = 1:m
25     I = del(i,1);

```

```

26     for j = 2:n-1
27         I = I + 2*del(i,j);
28     end
29     Itrap1(i) = hx/2 * (I+del(i,n));
30 end
31 Itrap1 = round(Itrap1*10^4)/10^4;
32
33 hy = y(2) - y(1);
34 Itrap2 = Itrap1(1)
35 for i = 2:n-1
36     Itrap2 = Itrap2 + 2* Itrap1(i);
37 end
38 Itrap2 = round(hy/2*(Itrap2+Itrap1(m))*10^4)/10^4;
39 disp(Itrap2,"I = ")

```

Scilab code Exa 7.14 One Point Gauss Legendre Quadrature Formula

```

1 //Example 7.14
2
3 clc
4 clear
5
6 n = 1;
7 if n==1 then
8     M = [0 2];
9 elseif n==2
10    M = [sqrt(1/3) 1; -sqrt(1/3) 1];
11 elseif n==3
12    M = [0 8/9; -0.774597 5/9; 0.774597 5/9];
13 elseif n==4
14    M = [-0.339981 0.652145; -0.861136 0.347855;
15         0.339981 0.652145; 0.861136 0.347855];
16 elseif n==5
17    M = [-0 0.568889; -0.538469 0.467914; -0.906180
18         0.236927; 0 0.568889; 0.538469 0.467914;

```

```

        0.906180 0.236927];
17 elseif n==6
18     M = [-0.238619 0.467914; -0.661209 0.360762;
          -0.932470 0.171325; 0.238619 0.467914;
          0.661209 0.360762; 0.932470 0.171325];
19 end
20
21 X = M(:,1);
22 W = M(:,2);
23
24 disp(X,"E1 = ")
25 disp(W,"W1 = ")

```

Scilab code Exa 7.15 Two Point Gauss Legendre Quadrature Formula

```

1 //Example 7.15
2
3 clc
4 clear
5
6 n = 2;
7 if n==1 then
8     M = [0 2];
9 elseif n==2
10    M = [sqrt(1/3) 1; -sqrt(1/3) 1];
11 elseif n==3
12    M = [0 8/9; -0.774597 5/9; 0.774597 5/9];
13 elseif n==4
14    M = [-0.339981 0.652145; -0.861136 0.347855;
          0.339981 0.652145; 0.861136 0.347855];
15 elseif n==5
16    M = [-0 0.568889; -0.538469 0.467914; -0.906180
          0.236927; 0 0.568889; 0.538469 0.467914;
          0.906180 0.236927];
17 elseif n==6

```

```

18     M = [-0.238619  0.467914; -0.661209  0.360762;
          -0.932470  0.171325;  0.238619  0.467914;
           0.661209  0.360762;  0.932470  0.171325];
19 end
20
21 X = M(:,1);
22 W = M(:,2);
23
24 disp(W(1),"W1 = ")
25 disp(W(2),"W2 = ")
26 disp(X(1),"E1 = ")
27 disp(X(2),"E2 = ")

```

Scilab code Exa 7.16 Four Point Gauss Legendre Quadrature Formula

```

1 //Example 7.16
2
3 clc
4 clear
5
6 function [f] = fun1(x)
7     f = 3*x^2 + x^3;
8 endfunction
9
10 n = 4;
11 if n==1 then
12     M = [0 2];
13 elseif n==2
14     M = [sqrt(1/3) 1];
15 elseif n==3
16     M = [0 8/9; -0.774597 5/9; 0.774597 5/9];
17 elseif n==4
18     M = [-0.339981 0.652145; -0.861136 0.347855;
          0.339981 0.652145; 0.861136 0.347855];
19 elseif n==5

```

```
20     M = [-0 0.568889; -0.538469 0.467914; -0.906180
           0.236927; 0 0.568889; 0.538469 0.467914;
           0.906180 0.236927];
21 elseif n==6
22     M = [-0.238619 0.467914; -0.661209 0.360762;
           -0.932470 0.171325; 0.238619 0.467914;
           0.661209 0.360762; 0.932470 0.171325];
23 end
24
25 X = M(:,1);
26 W = M(:,2);
27 I = 0;
28 for i = 1:length(X)
29     I = I + W(i)*fun1(X(i));
30 end
31 disp(I,"I = ")
```

Chapter 8

Ordinary Differential Equations

Scilab code Exa 8.1 Initial Value Problem using Taylor Series Method

```
1 //Example 8.1
2
3 clc
4 clear
5
6 function [f] = dydt(t,y)
7     f = t+y;
8 endfunction
9
10 y0 = 0;
11 t0 = 1;
12 t = 1.2;
13 h = 0.1;
14
15 n = (t-t0)/h;
16 tt = t0;
17 y = y0;
18 den = [1 2 6 24 120];
19 for i = 1:n
20     d2ydt = 1 + dydt(tt,y);
21     d3ydt = d2ydt;
```

```

22     d4ydt = d3ydt;
23     d5ydt = d4ydt;
24     dy = [dydt(tt,y) d2ydt d3ydt d4ydt d5ydt];
25     tt = tt + h;
26     for j = 1:length(dy)
27         y = y + dy(j)*(tt-t0)^j/den(j);
28     end
29     t0 = tt;
30 end
31 disp(y,"y(1.2) = ")
32
33 function [f] = closed(t)
34     f = -t -1 + 2*exp(t-1);
35 endfunction
36 yclosed = closed(1.2);
37 yclosed = round(yclosed*10^4)/10^4;
38 disp(yclosed,"y_closed form = ")
39 disp("Comparing the results obtained numerically and
      in closed form, we observe ")
40 disp("that they agree up to four decimals")

```

Scilab code Exa 8.2 Initial Value Problem using Euler Method

```

1 //Example 8.2
2
3 clc
4 clear
5
6 function [f] = dydt(t,y)
7     f = (y-t) / (y+t);
8 endfunction
9
10 y0 = 1;
11 t0 = 0;
12 t = 0.1;

```

```

13 n = 5;
14 h = (t-t0)/n;
15
16 tt = t0;
17 y = y0;
18 for i = 1:n
19     y = y +h*dydt(tt,y);
20     y = round(y*10^4)/10^4;
21     tt = tt + h;
22 end
23 disp(y,"y(t = 0.1) = ")

```

Scilab code Exa 8.3 Initial Value Problem using Modified Euler Method

```

1 //Example 8.3
2
3 clc
4 clear
5
6 function [f] = dydt(t,y)
7     f = t + sqrt(y);
8 endfunction
9
10 y0 = 1;
11 t0 = 0;
12 h = 0.2;
13 t = 0.6;
14 n = (t-t0)/h;
15
16 tt = t0;
17
18 for i = 1:n
19     y11 = y0 + h*dydt(tt,y0);
20     t1 = tt + h;
21     y1 = y0 + h/2*(dydt(tt,y0) + dydt(t1,y11));

```



```

22     y1 = round(y1*10^4)/10^4;
23
24     y(i) = y1;
25     y0 = y1;
26     tt = t1;
27 end
28 mprintf("%5s %8s", 't', 'y')
29 disp([(t0+h:h:t)' y])

```

Scilab code Exa 8.4 Initial Value Problem using Second Order Runge Kutta Method

```

1 //Example 8.4
2
3 clc
4 clear
5
6 function [f] = fun1(x,y)
7     f = (y+x) / (y-x);
8 endfunction
9
10 function [f] = rk2(x,y)
11     k1 = h*fun1(x,y);
12     k2 = h*fun1(x+3/2*h,y+3/2*k1);
13     f = y + 1/3*(2*k1+k2);
14 endfunction
15
16 x0 = 0;
17 y0 = 1;
18 h = 0.2;
19 x = 0.4;
20 n = (x-x0)/h;
21
22 for i = 1:n
23     y = rk2(x0,y0);

```

```

24     x0 = x0 + h;
25     y0 = y;
26     y = round(y*10^5)/10^5;
27 end
28
29 disp(y,"y(0.4) = ")

```

Scilab code Exa 8.5 Initial Value Problem using Fourth Order Runge Kutta Method

```

1 //Etmple 8.5
2
3 clc
4 clear
5
6 function [f] = fun1(t,y)
7     f = t+y;
8 endfunction
9
10 function [f] = rk4(t,y)
11     k1 = h*fun1(t,y);
12     k2 = h*fun1(t+1/2*h,y+1/2*k1);
13     k3 = h*fun1(t+1/2*h,y+1/2*k2);
14     k4 = h*fun1(t+h,y+k1);
15     f = y + 1/6*(k1+2*k2+2*k3+k4);
16 endfunction
17
18 t0 = 0;
19 y0 = 1;
20 h = 0.1;
21 t = 0.4;
22 n = (t-t0)/h;
23
24 for i = 1:n
25     y = rk4(t0,y0);

```

```

26     t0 = t0 + h;
27     y0 = y;
28     y = round(y*10^5)/10^5;
29 end
30
31 disp(y,"y(0.4) = ")

```

Scilab code Exa 8.6 Van Der Pol Equation using Fourth Order Runge
Kutta Equation

```

1 //Example 8.6
2
3 clc
4 clear
5
6 function [f] = f1(x,y,p)
7     f = p;
8 endfunction
9
10 function [f] = f2(x,y,p)
11     f = 0.1*(1-y^2)*p - y;
12 endfunction
13
14 x0 = 0;
15 y0 = 1;
16 p0 = 0;
17 h = 0.2;
18 x = 0.2;
19 n = (x-x0)/h;
20
21 for i = 1:n
22     k1 = h*f1(x0,y0,p0);
23     l1 = h*f2(x0,y0,p0);
24     k2 = h*f1(x0+h/2,y0+k1/2,p0+l1/2);
25     l2 = h*f2(x0+h/2,y0+k1/2,p0+l1/2);

```

```

26     k3 = h*f1(x0+h/2,y0+k2/2,p0+l2/2);
27     l3 = h*f2(x0+h/2,y0+k2/2,p0+l2/2);
28     k4 = h*f1(x0+h,y0+k3,p0+l3);
29     l4 = h*f2(x0+h,y0+k3,p0+l3);
30     y = y0 + 1/6*(k1+2*(k2+k3)+k4);
31     p = p0 + 1/6*(l1+2*(l2+l3)+l4);
32     y = round(y*10^4)/10^4;
33     p = round(p*10^4)/10^4;
34 end
35
36 disp(y,"y(0.2) = ")
37 disp(p,"y''(0.2) = ")

```

Scilab code Exa 8.7 Milne Predictor Corrector Method

```

1 //Example 8.7
2
3 clc
4 clear
5
6 function [f] = dy(t,y)
7     f = 1/2*(t+y);
8 endfunction
9
10 tt = 0:0.5:1.5;
11 yy = [2 2.636 3.595 4.968];
12
13 t0 = tt(1);
14 y0 = yy(1);
15 t = 2;
16 h = tt(2) - tt(1);
17 n = (t-t0)/h;
18 for i = 1:n
19     dydt(1) = dy(t0,yy(1));
20     dydt(2) = dy(t0+h,yy(2));

```

```

21     dydt(3) = dy(t0+2*h,yy(3));
22     dydt(4) = dy(t0+3*h,yy(4));
23
24     yP = yy(1) + 4*h/3*(2*dydt(2)-dydt(3)+2*dydt(4))
        ;
25     dydt(5) = dy(t0+4*h,yP);
26     yC = yy(3) + h/3*(dydt(3)+4*dydt(4)+dydt(5));
27 end
28 yC = round(yC*10^4)/10^4;
29 disp(yC,"y(2.0) = ")

```

Scilab code Exa 8.8 Milne Predictor Corrector Method

```

1 //Example 8.8
2
3 clc
4 clear
5
6 function [f] = dy(t,y)
7     f = t+y;
8 endfunction
9
10
11 tt = 0:0.1:0.3;
12 yy = [1 1.1103 1.2428 1.3997];
13
14 t0 = tt(1);
15 y0 = yy(1);
16 t = 2;
17 h = tt(2) - tt(1);
18 n = (t-t0)/h;
19 for i = 1:n
20     dydt(1) = dy(t0,yy(1));
21     dydt(2) = dy(t0+h,yy(2));
22     dydt(3) = dy(t0+2*h,yy(3));

```

```

23     dydt(4) = dy(t0+3*h,yy(4));
24
25     yP = yy(1) + 4*h/3*(2*dydt(2)-dydt(3)+2*dydt(4))
        ;
26     dydt(5) = dy(t0+4*h,yP);
27     yC = yy(3) + h/3*(dydt(3)+4*dydt(4)+dydt(5));
28 end
29 yC = round(yC*10^4)/10^4;
30 disp(yC,"y(0.4) = ")
31
32 t = [tt'; t0+4*h];
33 y = [yy'; yC];
34 mprintf("\n%6s %8s", 't', 'y')
35 disp([t y])

```

Scilab code Exa 8.9 Adam Moulton Predictor Corrector Method

```

1 //Example 8.9
2
3 clc
4 clear
5
6 function [f] = fun1(t,y)
7     f = y - t^2;
8 endfunction
9
10 function [f] = rk4(t,y)
11     k1 = h*fun1(t,y);
12     k2 = h*fun1(t+1/2*h,y+1/2*k1);
13     k3 = h*fun1(t+1/2*h,y+1/2*k2);
14     k4 = h*fun1(t+h,y+k1);
15     f = y + 1/6*(k1+2*k2+2*k3+k4);
16 endfunction
17
18 t0 = 0;

```

```

19 y0 = 1;
20 t = 1;
21 h = 0.2;
22 n = (t-t0)/h;
23 y = y0;
24
25 for i = 2:4
26     y(i) = rk4(t0,y0);
27     t0 = t0 + h;
28     y0 = y(i);
29 end
30
31 t0 = 0;
32 dydt(1) = fun1(t0,y(1));
33 dydt(2) = fun1(t0+h,y(2));
34 dydt(3) = fun1(t0+2*h,y(3));
35 dydt(4) = fun1(t0+3*h,y(4));
36
37 for i = 1:n-3
38     yP = y(4) + h/24*(55*dydt(4)-59*dydt(3)+37*dydt
        (2)-9*dydt(1));
39     dydt(5) = fun1(t0+(3+i)*h,yP);
40     yC = y(4) + h/24*(9*dydt(5)+19*dydt(4)-5*dydt(3)
        +dydt(2));
41     y = [y(2:4); yC];
42     dydt = [dydt(2:4); fun1(t0+(3+i)*h,yC)]
43 end
44 disp(yC,"Computed Solution: y(1.0) = ")
45
46 function [f] = true(t)
47     f = t^2 + 2*t +2 - exp(t);
48 endfunction
49 ytrue = true(1.0);
50 ytrue = round(ytrue*10^4)/10^4;
51 disp(ytrue,"Analytical Solution: y(1.0) = ")

```

Chapter 9

Parabolic Partial Differential Equations

Scilab code Exa 9.1 Taylor Series Expansion

```
1 // Example 9.1
2 // This is an analytical problem and need not be
   coded.
```

Scilab code Exa 9.2 Initial Boundary Value Problem using Explicit Finite Difference Method

```
1 //Example 9.2
2
3 clc
4 clear
5
6 delx = 0.1;
7 delt = 0.002;
8 xf = 1;
9 tf = 0.006;
```



```

10 x = 0:delx:xf;
11 t = 0:delt:tf;
12 m = length(x);
13 n = length(t);
14 lamda = delt/delx^2;
15
16 y = zeros(n,m);
17 y(1:n,1) = 0;
18 y(1:n,m) = 0;
19 y(1,1:m) = sin(%pi*x);
20 for k = 2:n
21     M1 = zeros(m-2);
22     M2 = zeros(m-2,1);
23     for i = 1:m-2
24         M1(i,i) = 1+2*lamda;
25         if i==1
26             M1(i,i+1) = -lamda;
27             M2(i) = y(k-1,i+1) + lamda*y(k,i);
28         elseif i==m-2
29             M1(i,i-1) = -lamda;
30             M2(i) = y(k-1,i+1) + lamda*y(k,i+2);
31         else
32             M1(i,i+1) = -lamda;
33             M1(i,i-1) = -lamda;
34             M2(i) = y(k-1,i+1);
35         end
36     end
37     y(k,2:m-1) = (M1\M2)';
38 end
39 y = round(y*10^4)/10^4;
40 mprintf("%4s %7s %9s %8s %9s %9s %9s %9s %9s %9s %9s\n", 'n', 't', 'x = 0.0 ', 'x = 0.1 ', 'x = 0.2 ', 'x = 0.3 ', 'x = 0.4 ', 'x = 0.5 ', 'x = 0.6 ', 'x = 0.7 ', 'x = 0.8 ', 'x = 0.9 ', 'x = 1.0 ');
41 disp([(0:n-1)' t' y])
42
43 disp("At t = 0.006:")
44 disp(y(n,1:m),"Computed Solution:")

```

```
45 Texact = exp(-%pi^2*tf)*sin(%pi*x);
46 Texact = round(Texact*10^4)/10^4;
47 disp(Texact," Analytical Solution:")
```

Scilab code Exa 9.3 Initial Boundary Value Problem using Explicit Finite Difference Method

```
1 //Example 9.3
2
3 clc
4 clear
5
6 delx = 0.1;
7 delT = 0.001;
8 xf = 0.5;
9 tf = 0.003;
10 x = 0:delx:xf;
11 t = 0:delT:tf;
12 m = length(x);
13 n = length(t);
14 r = delT/delx^2;
15
16 T = zeros(m,n);
17 T(1:m,1) = 0;
18 delTxi = 0;
19 delTxf = 1;
20
21 for j = 1:n
22     M1 = zeros(m,m);
23     M2 = zeros(m,1);
24     for i = 1:m
25         if i == 1 then
26             M1(i,i) = 1;
27             M1(i,i+1) = -1;
28             M2(i) = -delx * delTxi;
```

```

29         elseif i == m then
30             M1(i,i) = 1;
31             M1(i,i-1) = -1;
32             M2(i) = delx * delTxf;
33         else
34             M1(i,i) = 1;
35             M2(i) = r*T(i+1,j) + (1-2*r) * T(i,j) +
                    r*T(i-1,j);
36         end
37     end
38     T(1:m,j+1) = (M1\M2);
39 end
40 T = T(:,2:n+1);
41 mprintf("%4s %7s %9s %5s %7s %9s %9s %9s", 'n', 't', 'x
        = 0.0 ', 'x=0.1 ', 'x = 0.2 ', 'x = 0.3 ', 'x = 0.4 ', 'x
        = 0.5 ');
42 disp([(0:n-1)' t' T'])

```

Scilab code Exa 9.4 Crank Nicolson Finite Difference Method

```

1 //Example 9.4
2
3 clc
4 clear
5
6 delx = 0.25;
7 delt = 1/32;
8 xf = 1;
9 tf = delt;
10 x = 0:delx:xf;
11 t = 0:delt:tf;
12 m = length(x);
13 n = length(t);
14 r = delt/delx^2;
15

```

```

16
17 T = zeros(m,n);
18 T(1:m,1) = 1;
19 T(1,1:n) = 0;
20 T(m,1:n) = 0;
21
22 for j = 1:n-1
23     M1 = zeros(m-2,m-2);
24     M2 = zeros(m-2,1);
25     for i = 2:m-1
26         if i == 2 then
27             M1(i-1,i-1) = -2*(1+r);
28             M1(i-1,i) = r;
29             M2(i-1) = -(r*T(i+1,j) + 2*(1-r)*T(i
30                 ,j) + r*T(i-1,j));
31         elseif i == m-1
32             M1(i-1,i-2) = r;
33             M1(i-1,i-1) = -2*(1+r);
34             M2(i-1) = -(r*T(i+1,j) + 2*(1-r)*T(i
35                 ,j) + r*T(i-1,j));
36         else
37             M1(i-1,i-2) = r;
38             M1(i-1,i-1) = -2*(1+r);
39             M1(i-1,i) = r;
40             M2(i-1) = -(r*T(i+1,j) + 2*(1-r)*T(i
41                 ,j) + r*T(i-1,j));
42         end
43     end
44     T(2:m-1,j+1) = (M1\M2);
45 end
46 T1 = M1\M2;
47 for i = 1:length(T1)
48     disp(strcat(["T",string(i)," = ",string(T1(i))])
49         );
50 end

```

Scilab code Exa 9.5 Crank Nicolson Finite Difference Method

```
1 //Example 9.5
2
3 clc
4 clear
5
6 delx = 1;
7 delt = 1.893;
8 alpha = 0.132;
9 xf = 4;
10 tf = delt;
11 x = 0:delx:xf;
12 t = 0:delt:tf;
13 m = length(x);
14 n = length(t);
15 r = alpha*delt/delx^2;
16 r = round(r*10^2)/10^2;
17
18 T = zeros(m,n);
19 T(1:m,1) = 1000;
20
21 for j = 1:n-1
22     M1 = zeros(m,m);
23     M2 = zeros(m,1);
24     for i = 1:m
25         if i == 1 then
26             M1(i,i) = -(2+2.15*r);
27             M1(i,i+1) = 2*r;
28             M2(i) = -(2*r*T(2,j) + (2-2.15*r)*T(1,j)
                + 21*r);
29         elseif i == m then
30             M1(i,i) = -(2+1.75*r);
31             M1(i,i-1) = 2*r;
```

```

32         M2(i) = -(2*r*T(m-1,j) + (2-1.75*r)*T(m,j) -
33             35*r);
34     else
35         M1(i,i-1) = r;
36         M1(i,i)   = -2*(1+r);
37         M1(i,i+1) = r;
38         M2(i)     = -(r*T(i+1,j) + 2*(1-r)*T(i,j)
39             ) + r*T(i-1,j));
40     end
41 end
42 disp(M1," Coefficient Matrix:")
43 disp(M2," Constant Matrix:")
44 T = round(T*10^4)/10^4;
45 disp(T'," Table:")

```

Scilab code Exa 9.6 Crank Nicolson Scheme for Diffusion Equation

```

1 // Example 9.6
2 // This is an analytical problem and need not be
   coded.

```

Scilab code Exa 9.7 Alternate Direction Implicit Method

```

1 //Example 9.7
2
3 clc
4 clear
5
6 h = 2;
7 deltt = 4;
8 tff = 8;

```

```

 9  xf = 8;
10  yf = 6;
11  x = 0:h:xf;
12  y = 0:h:yf;
13  t = 0:delt:tf;
14  m = length(x);
15  n = length(y);
16  p = length(t);
17  r = delt/h^2;
18  r = round(r*10^2)/10^2;
19
20  T = 50*ones(n,m);
21  T0 = T;
22  T(1,1:m) = 110:-10:70;
23  T(n,1:m) = 0:10:40;
24  T(2:n-1,1) = [65; 25];
25  T(2:n-1,m) = [60; 50];
26
27  u = (m-2)*(n-2);
28  index = [repmat(2:m-1,1,n-2); gsort(repmat(2:n-1,1,m
      -2))];
29
30  M1 = zeros(u,u);
31  M2 = zeros(u,1);
32  for j = 2:m-1
33      for i = 2:n-1
34          ind = find(index(1,:) == j & index(2,:) == i);
35          if j == 2 then
36              M1(ind,ind) = 1+2*r;
37              M1(ind,ind+1) = -r;
38              M2(ind) = r*T(i,j-1) + r*T0(i-1,j) +
                  (1-2*r)*T0(i,j) + r*T0(i+1,j);
39          elseif j == m-1 then
40              M1(ind,ind-1) = -r;
41              M1(ind,ind) = 1+2*r;
42              M2(ind) = r*T(i,j+1) + r*T0(i-1,j) +
                  (1-2*r)*T0(i,j) + r*T0(i+1,j);
43          else

```

```

44         M1(ind,ind-1) = -r;
45         M1(ind,ind) = 1+2*r;
46         M1(ind,ind+1) = -r;
47         M2(ind) = r*T0(i-1,j) + (1-2*r)*T0(i,j)
           + r*T0(i+1,j);
48     end
49 end
50 end
51 value = M1\M2;
52 value = round(value*10^4)/10^4;
53 for i = 1:length(index(1,:))
54     t = index(:,i);
55     T(t(2),t(1)) = value(i);
56 end
57 disp(T,"At t = 4:")
58 T0 = T;
59
60 index = gsort(index,'lc','i');
61 M1 = zeros(u,u);
62 M2 = zeros(u,1);
63 for j = 2:m-1
64     for i = 2:n-1
65         ind = find(index(1,:) == j & index(2,:) == i);
66         if i == 2 then
67             M1(ind,ind) = 1+2*r;
68             M1(ind,ind+1) = -r;
69             M2(ind) = r*T(i-1,j) + r*T0(i,j-1) +
           (1-2*r)*T0(i,j) + r*T0(i,j+1);
70         elseif i == n-1 then
71             M1(ind,ind-1) = -r;
72             M1(ind,ind) = 1+2*r;
73             M2(ind) = r*T(i+1,j) + r*T0(i,j-1) +
           (1-2*r)*T0(i,j) + r*T0(i,j+1);
74         else
75             M1(ind,ind-1) = -r;
76             M1(ind,ind) = 1+2*r;
77             M1(ind,ind+1) = -r;
78             M2(ind) = + r*T0(i,j-1) + (1-2*r)*T0(i,j)

```



```

                                ) + r*T0(i,j+1);
79         end
80     end
81 end
82 value = M1\M2;
83 value = round(value*10^4)/10^4;
84 for i = 1:length(index(1,:))
85     t = index(:,i);
86     T(t(2),t(1)) = value(i);
87 end
88 disp(T,"At t = 8:")

```

Chapter 10

Elliptical Partial Differential Equations

Scilab code Exa 10.1 Laplace Equation using Five Point Formulae

```
1 //Example 10.1
2
3 clc
4 clear
5
6 h = 1/4;
7 xf = 1;
8 yf = 1;
9 x = 0:h:xf;
10 y = 0:h:yf;
11 m = length(y);
12 n = length(x);
13
14 u = zeros(m,n);
15 u(m,:) = 100*x;
16 u(:,n) = 100*y';
17 u0 = u;
18
19 I = ceil(m/2);
```

```

20 J = ceil(n/2);
21
22 u(J,I) = (u0(J-2,I-2) + u0(J-2,I+2) + u0(J+2,I-2) +
           u0(J+2,I+2)) / 4;
23
24 for j = [J-1 J+1]
25     for i = [I-1 I+1]
26         u(j,i) = (u(j-1,i-1) + u(j-1,i+1) + u(j+1,i
                    -1) + u(j+1,i+1)) / 4;
27     end
28 end
29
30 j1 = [J-1 J J J+1];
31 i1 = [I I-1 I+1 I];
32 for k = 1:4
33     i = i1(k);
34     j = j1(k);
35     u(j,i) = (u(j,i-1) + u(j,i+1) + u(j-1,i) + u(j
                +1,i)) / 4;
36 end
37
38 disp(u,"u:")

```

Scilab code Exa 10.2 Temperature in Two Dimensional Geometry

```

1 // Example 10.2
2 // This is an analytical problem and need not be
   coded.

```

Scilab code Exa 10.3 Laplace Equation in Two Dimension using Five Point Formulae

```

1 //Example 10.3

```

```

2
3 clc
4 clear
5
6 m = 5;
7 n = 5;
8 u = zeros(m,n);
9 u(m,:) = [50 100 100 100 50];
10 u0 = u;
11 I = ceil(m/2);
12 J = ceil(n/2);
13
14 u(J,I) = (u0(J-2,I-2) + u0(J-2,I+2) + u0(J+2,I-2) +
           u0(J+2,I+2)) / 4;
15
16 for j = [J-1 J+1]
17     for i = [I-1 I+1]
18         u(j,i) = (u(j-1,i-1) + u(j-1,i+1) + u(j+1,i
                    -1) + u(j+1,i+1)) / 4;
19     end
20 end
21
22 j1 = [J-1 J J J+1];
23 i1 = [I I-1 I+1 I];
24 for k = 1:4
25     i = i1(k);
26     j = j1(k);
27     u(j,i) = (u(j,i-1) + u(j,i+1) + u(j-1,i) + u(j
                +1,i)) / 4;
28 end
29
30 kf = 2;
31 tab = zeros(kf+1,(m-2)*(n-2));
32 row = [];
33 for j = 2:n-1
34     row = [row u(j,2:m-1)];
35 end
36 tab(1,:) = row;

```

```

37 for k = 1:kf
38     row = [];
39     for j = 2:n-1
40         for i = 2:m-1
41             u(j,i) = (u(j,i-1) + u(j,i+1) + u(j-1,i)
42                     + u(j+1,i)) / 4;
43         end
44         row = [row u(j,2:m-1)];
45     end
46     row = round(row*10^4)/10^4;
47     tab(k+1,:) = row;
48 end
49 mprintf("%4s %9s %9s %9s %9s %10s %10s %10s %10s
50         %10s", 'r', 'u11', 'u21', 'u31', 'u12', 'u22', 'u32', '
51         u13', 'u23', 'u33')
52 disp([(1:k+1)' tab])

```

Scilab code Exa 10.4 Poisson Equation using Liebmann Iterative Method

```

1 //Example 10.4
2
3 clc
4 clear
5
6 h = 1/3;
7 x = 0:h:1;
8 y = 0:h:1;
9 m = length(y);
10 n = length(x);
11 u = zeros(m,n);
12 u(m,2:n-1) = 1;
13
14 kf = 5;
15 tab = zeros(kf, (m-2)*(n-2));
16 for k = 1:kf

```

```

17     row = [];
18     for j = 2:n-1
19         for i = 2:m-1
20             constant = 10/9* (5 + 1/9*(i-1)^2 +
21                             1/9*(j-1)^2);
22             u(j,i) = (u(j,i-1) + u(j,i+1) + u(j-1,i)
23                     + u(j+1,i) + constant) / 4;
24         end
25     row = [row u(j,2:m-1)];
26 end
27 row = round(row*10^4)/10^4;
28 tab(k,:) = row;
29 disp([(1:k)' tab])

```

Scilab code Exa 10.5 Laplace Equation using Liebmann Over Relaxation Method

```

1 //Example 10.5
2
3 clc
4 clear
5
6 x = 0:4;
7 y = 0:4;
8 m = length(y);
9 n = length(x);
10 u = zeros(m,n);
11 u(m,:) = x.^3;
12 u(:,n) = 16*y';
13 u0 = u;
14
15 I = ceil(m/2);

```

```

16 J = ceil(n/2);
17
18 u(J,I) = (u0(J-2,I-2) + u0(J-2,I+2) + u0(J+2,I-2) +
           u0(J+2,I+2)) / 4;
19
20 for j = [J-1 J+1]
21     for i = [I-1 I+1]
22         u(j,i) = (u(j-1,i-1) + u(j-1,i+1) + u(j+1,i
                    -1) + u(j+1,i+1)) / 4;
23     end
24 end
25
26 j1 = [J-1 J J J+1];
27 i1 = [I I-1 I+1 I];
28 for k = 1:4
29     i = i1(k);
30     j = j1(k);
31     u(j,i) = (u(j,i-1) + u(j,i+1) + u(j-1,i) + u(j
                +1,i)) / 4;
32 end
33 disp(u,"u:")
34
35 p = m-1;
36 q = n-1;
37 c = cos(%pi/p) + cos(%pi/q);
38 w = 4/(2+sqrt(4-c^2));
39 w = round(w*10^3)/10^3;
40
41 kf = 10;
42 tab = zeros(kf+1,(m-2)*(n-2));
43 row = [];
44 for j = 2:n-1
45     row = [row u(j,2:m-1)];
46 end
47 tab(1,:) = row;
48 for k = 1:kf
49     row = [];
50     for j = 2:n-1

```

```

51         for i = 2:m-1
52             u(j,i) = (u(j,i-1) + u(j,i+1) + u(j-1,i)
                    + u(j+1,i)) *w/4 + (1-w)*u(j,i);
53         end
54         row = [row u(j,2:m-1)];
55     end
56     row = round(row*10^4)/10^4;
57     tab(k+1,:) = row;
58 end
59 mprintf("\n\n%8s %9s %10s %10s %9s %10s %10s %9s %9s
        ", 'u11 ', 'u21 ', 'u31 ', 'u12 ', 'u22 ', 'u32 ', 'u13 ', 'u23 ',
        , 'u33 ')
60 disp(tab)

```

Chapter 11

Hyperbolic Partial Differential Equations

Scilab code Exa 11.1 Initial Value Problem using Wave Equation

```
1 //Example 11.1
2
3 clc
4 clear
5
6 delx = 1/8;
7 delt = 1/8;
8 x = 0:delx:1;
9 t = 0:delt:1;
10 m = length(x);
11 n = length(t);
12 u = zeros(n,m);
13 u(1,:) = sin(%pi*x);
14 N = 1/delx;
15 r = delt/delx;
16
17 for j = 2:n
18     for i = 2:m-1
19         if j == 2 then
```

```

20         u(j,i) = (2*(1-r^2)*u(j-1,i) + r^2*(u(j
           -1,i-1) + u(j-1,i+1))) /2;
21     else
22         u(j,i) = 2*(1-r^2)*u(j-1,i) + r^2*(u(j
           -1,i-1) + u(j-1,i+1)) - u(j-2,i);
23     end
24 end
25 end
26 u = round(u*10^4)/10^4;
27 mprintf("\n\n%6s %9s %9s %8s %s %8s %10s %10s %9s
           %7s %s", 't', 'x = 0.0 ', 'x = 0.125 ', 'x = 0.25 ', 'x =
           0.375 ', 'x = 0.5 ', 'x=0.625 ', 'x = 0.75 ', 'x=0.875 ',
           'x = 1.0 ', 'n');
28 disp([(0:1/8:1)' u (0:n-1)']);
29 mprintf("\n\n");
30 t = [1/2 1];
31 for i = 1:length(t)
32     Ex(i,:) = sin(%pi*x) * cos(%pi*t(i));
33 end
34 Ex = round(Ex*10^4)/10^4;
35 disp("At t = 1/2:")
36 disp(u(find(x==1/2),:),"Computed Solution:")
37 disp(Ex(1,:), "Actual Solution:")
38
39 disp("At t = 1:")
40 disp(u(find(x==1),:),"Computed Solution:")
41 disp(Ex(2,:), "Actual Solution:")

```

Scilab code Exa 11.2 Initial Value Problem using Wave Equation

```

1 //Example 11.2
2
3 clc
4 clear
5

```

```

6 delx = 0.2;
7 delt = 0.2;
8 x = 0:delx:1;
9 t = 0:delt:0.8;
10 m = length(x);
11 n = length(t);
12 u = zeros(n,m);
13 u(1,:) = x^2;
14 u(:,m) = 1+t';
15 N = 1/delx;
16 r = delt/delx;
17
18 for j = 2:n
19     for i = 2:m-1
20         if j == 2 then
21             u(j,i) = (2*(1-r^2)*u(j-1,i) + r^2*(u(j
                -1,i-1) + u(j-1,i+1)) + 2*delt) /2;
22         else
23             u(j,i) = 2*(1-r^2)*u(j-1,i) + r^2*(u(j
                -1,i-1) + u(j-1,i+1)) - u(j-2,i);
24         end
25     end
26 end
27 u = round(u*10^4)/10^4;
28 mprintf("\n%5s %9s %7s %7s %s %6s %6s", 't', 'x = 0.0 ',
        'x = 0.2 ', 'x = 0.4 ', 'x = 0.6 ', 'x = 0.8 ', 'x = 1.0
        ');
29 disp([t' u])

```
