

Scilab Textbook Companion for
Elementary Numerical Analysis: An
Algorithmic Approach
by S. D. Conte And C. de Boor ¹

Created by
Pravalika
B.Tech (pursuing)
Electronics Engineering
Visvesvaraya National Institute Of Technology
College Teacher
M. Devakar, VNIT Nagpur
Cross-Checked by
K. Suryanarayan and Prashant Dave, IITB

July 31, 2019

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: Elementary Numerical Analysis: An Algorithmic Approach

Author: S. D. Conte And C. de Boor

Publisher: McGraw - Hill Companies

Edition: 3

Year: 1980

ISBN: 70124477

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 Number systems and Errors	5
2 Interpolation by polynomials	6
3 The solution of nonlinear equations	11
4 Matrices and Systems of linear Equations	21
5 systems of equations and unconstrained optimization	28
6 Approximation	30
7 differentiation and integration	38
8 THE SOLUTION OF DIFFERENTIAL EQUATIONS	42

List of Scilab Codes

Exa 1.1	number system	5
Exa 2.1	shifted power form	6
Exa 2.2	second degree interpolating polynomial	7
Exa 2.3	determine polynomial by newton formula	7
Exa 2.5	Newtons formula	8
Exa 2.7	find N	9
Exa 3.1	Root finding	11
Exa 3.2.a	finding roots	12
Exa 3.3	Fixed point iteration	13
Exa 3.4	Fixed point iteration	14
Exa 3.5	Secant method	14
Exa 3.8.a	Roots of a polynomial equation	15
Exa 3.8	Polynomial equations real roots	16
Exa 3.10	Roots of polynomial equation	17
Exa 3.11	Roots of polynomial equation	17
Exa 3.12	Roots of polynomial equation	18
Exa 3.14	Roots of a polynomial equation	18
Exa 3.15	Roots of a polynomial equation	18
Exa 3.16	Roots of polynomial equation	19
Exa 3.17.a	Roots of a polynomial equation	19
Exa 4.1	matrix multiplication	21
Exa 4.2	matrix multiplication	21
Exa 4.3	properties of matrices	22
Exa 4.4	matrix	23
Exa 4.6	determinant	23
Exa 4.7	matrix	23
Exa 4.8	Backward substitution	24
Exa 4.14	norm	25

Exa 4.16	determinant	25
Exa 4.17	determinant	25
Exa 4.18	determinant	26
Exa 4.19	Eigen values	26
Exa 4.20	Eigen values	26
Exa 4.21	determinant	27
Exa 4.22	eigen values	27
Exa 5.1	gradient	28
Exa 5.2	Steep descent	29
Exa 6.1	uniform aproximation	30
Exa 6.2	distance at infinity	30
Exa 6.3	aproximation	31
Exa 6.5	approximate	33
Exa 6.11	polynomial of degree less than 3	34
Exa 6.12	Least squares approximation	35
Exa 7.1	integration	38
Exa 7.8	adaptive quadrature	38
Exa 8.1	Taylor series	42
Exa 8.5	Adamsbashforth3	43
Exa 8.6	Modified euler method	43

Chapter 1

Number systems and Errors

Scilab code Exa 1.1 number system

```
1
2 //Example (pg no.20)
3
4 //the number %pi/4 is the value of infinite
   series
5 //sum(((−1)^i)/(2*i+1))=1−sum(2/(16*(j)^2)−1)
6 // The sequence alpha1 ,alpha2 ,..... is monotone−
   decreasing
7 // to its limit %pi/4
8 // 0< =(alpha)n − %pi/4 <= (1/(4*n+3))    n
   =1 ,2 ,.....
9 // To calculate %pi/4 correct to within 10^(−6)
   using this sequence
10 // we would need 10^(6)<=4*n+3
11 n=(10^(6)−3)/4
12 // roughly n=250,000
```

Chapter 2

Interpolation by polynomials

Scilab code Exa 2.1 shifted power form

```
1 //Example 2.1
2
3 // Given  $p(6000)=1/3$  ,  $p(6001)=-2/3$ 
4 // From  $p(x)=a_0+a_1*x$  , by substituting  $x=6000$  &  $x$ 
   =6001
5 // we get equations  $a_0+a_1*(6000)=1/3$  &  $a_0+a_1*(6001)$ 
   =-2/3
6 //solving the above equations we get
7 a0=6000.3
8 a1=-1
9 def ('[y]=f(x)', 'y=6000.3-x')
10 f(6000)
11 f(6001)
12 //  $y=6000.3-x$  , equation recovers only only the
   first digit of the
13 // given function values ,a loss of four decimal
   digits
14 // remedy for such loss of significance is the use
   of SHIFTED POWER FORM
15 // $p(x)=a_0 + a_1*(x-c) + a_2*(x-c)^2 + \dots + a_n*(x-$ 
    $c)^n$ 
```



```

16 //if we choose the center c to be 6000
17
18 def(' [y]=p(x) ', 'y=0.33333-(x-6000) ')
19 p(6000)
20 p(6001)

```

Scilab code Exa 2.2 second degree interpolating polynomial

```

1 //Example 2.2
2
3 K(1)=1.5709
4 K(4)=1.5727
5 K(6)=1.5751
6 l0(3.5)=[(3.5-4)*(3.5-6)]/[(1-4)*(1-6)]
7 l1(3.5)=[(3.5-1)*(3.5-6)]/[(4-1)*(4-6)]
8 l2(3.5)=[(3.5-1)*(3.5-4)]/[(6-1)*(6-4)]
9 K(3.5)=l0(3.5)*K(1)+l1(3.5)*K(4)+l2(3.5)*K(6);
10 K(3.5)

```

Scilab code Exa 2.3 determine polynomial by newton formula

```

1 //Example 2.3
2 //Using Newton formula
3
4 x0=1
5 x1=4
6 x2=6
7 P2(1)=1.5709
8 P2(4)=1.5727
9 P2(6)=1.5751
10 K1=[P2(1)-P2(4)]/(1-4)
11 K2=[P2(4)-P2(6)]/(4-6)
12 K3={K1-K2}/(1-6)

```

```

13          // Where as K1 = K[1,4] , K2 =[4,6] ,K3 =
           [1,4,6]
14          def f(' [y]= f(x) ', 'y=P2(x0)+K1*(x-x0)+K3*(
           x-x0)*(x-x1) ')
15          funcprot(0)
16  x=poly(0,"x")
17  y=P2(x0)+K1*(x-x0)+K3*(x-x0)*(x-x1)
18  x=3.5
19  f

```

Scilab code Exa 2.5 Newtons formula

```

1 //Example 2.5
2
3  x0=1
4  x1=4
5  x2=6
6  x3=0
7  x4=3.5
8  K(1)=1.5709
9  K(4)=1.5727
10 K(6)=1.5751
11  P2(1)=1.5709
12  P2(4)=1.5727
13  P2(6)=1.5751
14
15  p0=K(1)
16  U0=1 //U0=U0(x')
17  K1=[P2(1)-P2(4)]/(1-4)
18  // Where as K1 = K[1,4]
19  U1=(x4-x0)*U0 //U1=U1(x')
20  p1=p0+U1*K1 //p1=p1(x')
21
22 //adding the point x2=6
23  K2=[P2(4)-P2(6)]/(4-6)

```

```

24         // Where as K2 =K[4,6]
25
26     K3={K1-K2}/(1-6)
27         // Where as K1 = K[1,4] , K2 =K[4,6] ,K3 = K
           [1,4,6]
28     U2=(x4-x0)*(x4-x1) //U2=U2(x')
29     p2=p1+U2*K3 //p2=p2(x')
30
31     // to check error approximation for k(3.5) we add
           point x3=0
32     // K(0)=1.5708=a
33     // p2(0)=1.5708=K(0)
34     a=1.5708
35     K4=[P2(6)-a]/(6-0)
36     //K4=K[x2,x3]=[6,0]
37     K5=-0.000001
38     //K5=K[x0,x1,x2,x3]
39     U3=(x4-x2)*(x4-x1)*(x4-x0) //U3=U3(x')
40
41     p3=p2+U3*K5
42     //p3=p3(x')

```

Scilab code Exa 2.7 find N

```

1
2 //Example 2.7
3
4 def f(' [y]=f(x)', 'y=(x)^(1/2)')
5 x0=1
6 x1=2
7
8 //abs(f(x')-p2(x')) <= ((x1-x0)^2)*M/8
9 // abs(f(x')-p2(x')) <= 2*((h)^3)/(3*(3)^(1/3))
10
11 h=(5*((10)^(-8))*24*((3)^(1/2)))^(1/3)

```

```
12
13 //h is approximately 0.0128
14 //h=(x1-x0)/N
15
16 N={ (x1-x0)/h}
17 //N is approximately 79
```

Chapter 3

The solution of nonlinear equations

check Appendix [AP 4](#) for dependency:

```
bisect.sce
```

check Appendix [AP 6](#) for dependency:

```
newt.sce
```

check Appendix [AP 7](#) for dependency:

```
regulfalsi.sce
```

check Appendix [AP 3](#) for dependency:

```
secantm.sce
```

Scilab code Exa 3.1 Root finding

```
1 // Example (3.1)
2
3 // finding roots using bisection method
4
5 def f(' [y]=f(x) ', 'y=x-0.2*sin(x)-0.5')
```

```

6         bisection(0.5,1.0,f)
7
8
9 //regula falsi method
10
11 def f(' [y]=f(x) ', 'y=x-0.2*sin(x)-0.5 ')
12 regularfalsi(0.5,1.0,f)
13
14 //secant method
15
16 def f(' [y]=f(x) ', 'y=x-0.2*sin(x)-0.5 ')
17 secant(0.5,1.0,f)
18
19
20 //newton rapson method
21
22
23 x=(0.5+1)/2
24 def f(' [y]=f(x) ', 'y=x-0.2*sin(x)-0.5 ')
25 def g(' [y]=g(x) ', 'y=1-0.2*cos(x) ')
26 x=newton(x,f,g)

```

check Appendix [AP 4](#) for dependency:

bisect.sce

Scilab code Exa 3.2.a finding roots

```

1 // // Example(3.2 a)
2 //
3 //// finding roots using bisection method
4 //
5 //
6 //         def f(' [y]=f(x) ', 'y=x^3-x-1')
7 //         bisection(0,10,f);
8 //

```

```

9 //
10 /// regula falsi method
11 //
12 // deff ( '[y]=f(x) ', 'y=x^3-x-1' )
13 // regularfalsi (0,10, f);
14 //
15 //
16 //
17 // newton rapson method
18 //
19 // x=(0+10)/2
20 // deff ( '[y]=f(x) ', 'x^3-x-1' )
21 // deff ( '[y]=g(x) ', 'y=2*x^2 -1' )
22 // x=newton(x, f, g);

```

check Appendix [AP 5](#) for dependency:

`fixedp.sce`

check Appendix [AP 6](#) for dependency:

`newt.sce`

check Appendix [AP 7](#) for dependency:

`regulfalsi.sce`

Scilab code Exa 3.3 Fixed point iteration

```

1 //example (3.3)
2
3 //here f(x)=e^(-x)-sin(x) , according
  to fixed point iteration we take
  g(x)=x+e^(-x)-sin(x);
4 // so , xn=g(xn)
5 deff ( '[y]=g(x) ', 'y=x+(2.718)^(-x)-sin(x) ' )
6 x=0.6

```

```

7 for n=1:1:18
8   g(x);
9   x=g(x)
10 end

```

Scilab code Exa 3.4 Fixed point iteration

```

1 //example(3.4)
2
3 //here f(x)=1.5*x-tan(x)-0.1=0, according to
  //fixed point iteration we get x=(0.1+tan(x))
  //1.5
4 //where g(x)=x=(0.1+tan(x))/1.5
5 //& xn=g(xn)
6 defm(' [y]=g(x) ', 'y=(0.1+tan(x))/1.5 ')
7 x=0
8
9 for n=1:1:10
10   g(x);
11   x=g(x)
12 end

```

check Appendix [AP 3](#) for dependency:

secantm.sce

Scilab code Exa 3.5 Secant method

```

1 //example(3.5)
2
3 defm(' [y]=f(x) ', 'y=x^3-x-1 ')
4 secant(1.0,1.5,f)

```

Scilab code Exa 3.8.a Roots of a polynomial equation

```
1 //example(pg no.111)
2
3
4 //a,b & f are the modulus coeff of x^0,x^1,x^5
5 c=[-6.8 10.8 -10.8 7.4 -3.7 1]
6 a=6.8;
7 b=10.8;
8 f=1;
9 n=5
10 p5=poly(c, 'x', 'coeff')
11 p=n*a/b
12 q=a/f^(1/n)
13 roots(p4)
14
15 xset('window',0);
16 x=-2:.01:2.5;
//
// defining the range of x.
17 def(' [y]=f(x)', 'y=x^5-3.7*x^4+7.4*x^3-10.8*x
//defining the
// cunction
18 y=feval(x,f);
19
20 a=gca();
21
22 a.y_location = "origin";
23
24 a.x_location = "origin";
25 plot(x,y)
// instruction to plot the graph
26
```

```
27 title(' y = 8*x^3-12*x^2-2*x+3')
```

Scilab code Exa 3.8 Polynomial equations real roots

```
1 //example(3.8)
2
3
4
5 //a,b & e are the modulus coeff of x^0,x^1,x^4
6 c=[-1 1 -1 -1 1]
7 a=1;
8 b=1;
9 e=1;
10 n=4
11 p4=poly(c, 'x', 'coeff')
12 p=n*a/b
13 q=(a/e)^(1/n)
14 roots(p4)
15 //from here we found that only 2 real roots,
16 //other two are complex roots
17 xset('window',0);
18 x=-2:0.1:3;
19
20 //
21 //defining the range of x.
22 defff(' [y]=f(x)', 'y=x^4-x^3-x^2+x-1');
23 //defining the function
24 y=feval(x,f);
25
26 a=gca();
27
28 a.y_location = "origin";
29
30 a.x_location = "origin";
31 plot(x,y)
```

```

    // instruction to plot the graph
27
28 title(' y =x^4-x^3-x^2+x-1')

```

Scilab code Exa 3.10 Roots of polynomial equation

```

1 //example(3.10)
2
3 c=[-3 1 0 1]
4 p3=poly(c, 'x', 'coeff')
5 roots(p3)
6 //here
7 xset('window',0);
8 x=-2:.01:2.5;
                                     //
    defining the range of x.
9 deff(' [y]=f(x)', 'y=x^3+x-3');
                                     //
    defining the cunction
10 y=feval(x,f);
11
12 a=gca();
13
14 a.y_location = "origin";
15
16 a.x_location = "origin";
17 plot(x,y)
                                     // instruction to plot the graph
18
19 title(' y =x^3+x-3')

```

Scilab code Exa 3.11 Roots of polynomial equation

```
1 //example(3.11)
2 c=[-6.8 10.8 -10.8 7.4 -3.7 1]
3 p5=poly(c, 'y', 'coeff')
4 roots(p5)
```

Scilab code Exa 3.12 Roots of polynomial equation

```
1 //example(3.12)
2
3 c=[-5040 13068 -13132 6769 -1960 322 -28 1]
4 p7=poly(c, 'y', 'coeff')
5 roots(p7)
```

Scilab code Exa 3.14 Roots of a polynomial equation

```
1 //example(3.14)
2
3 c=[-3 1 0 1]
4 p3=poly(c, 'y', 'coeff')
5 roots(p3)
```

Scilab code Exa 3.15 Roots of a polynomial equation

```
1 //example(3.15)
2
3 c=[-6.8 10.8 -10.8 7.4 -3.7 1]
4 p5=poly(c, 'x', 'coeff')
5 roots(p5)
```

Scilab code Exa 3.16 Roots of polynomial equation

```
1 //example(3.16)
2
3 c=[-5040 13068 -13132 6769 -1960 322 -28 1]
4 p7=poly(c, 'x', 'coeff')
5 roots(p7)
```

Scilab code Exa 3.17.a Roots of a polynomial equation

```
1 //example(3.17)
2
3 c=[51200 0 -39712 0 7392 0 -170 0 1 ]
4 p8=poly(c, 'x', 'coeff')
5 roots(p8)
6
7 xset('window',0);
8 x=-11:01:11;
//
    defining the range of x.
9 def(' [y]=f(x)', 'y=x^8-170*x^6+7392*x^4-39712*x
    ^2+51200'); //defining the
    cunction
10 y=feval(x,f);
11
12 a=gca();
13
14 a.y_location = "origin";
15
16 a.x_location = "origin";
17 plot(x,y)
```

```
18 // instruction to plot the graph
19 title(' y =x^8-170*x^6+7392*x^4-39712*x^2+51200')
```

Chapter 4

Matrices and Systems of linear Equations

Scilab code Exa 4.1 matrix multiplication

```
1
2 //Example (pg no.130)
3
4     A=[3 0 2;1 2 0;0 1 1]
5     B=[2 1;0 1;1 0]
6     A*B
```

Scilab code Exa 4.2 matrix multiplication

```
1 //Example (pg no.130)
2
3
4     A=[2 1;1 3]
5     B=[2 1;0 1]
6     A*B
7     B*A
```

```
8 //matrix multiplication is not commutative
9 //so A*B!=B*A
```

Scilab code Exa 4.3 properties of matrices

```
1 //Example (pg no.133)
2
3     A=[1 1;0 1]
4     inv(A)
5     B=[1 0;1 1]
6     inv(B)
7     A*B
8     inv(A*B)
9     inv(A)*inv(B)
10
11     inv(B)*inv(A)
12
13     I=eye(3,3)
14     C=(A*B)*(inv(A)*inv(B))
15
16
```

//inv(A*B)=
inv(B)*inv(A)

//Hence inv(A)*inv(B) = inv(A)*inv(B)

//C!=I
//so, inv(A)*inv(B) cannot be the inverse of (A*B)

Scilab code Exa 4.4 matrix

```
1
2 //Example (pg no.136)
3
4 // x1 + 2(x2) = 3
5 //2(x1) + 4(x2) = 6
6
7     A=[1 2;2 4]
8         //coefficient matrix of above equations
9     b=[3 6] '
10    x=A\b
11    //for corresponding homogenous system
12        // x1 + 2(x2) = 0
13        //2(x1) + 4(x2) = 0
14    A=[1 2;2 4]
15        //coefficient matrix of above equations
16    b=[0 0] '
17    x=A\b
```

Scilab code Exa 4.6 determinant

```
1 //Example (pg no.140)
2
3     A=[1 2;2 4]
4     det(A)
5
6         // Here A is a singular
7         // matrix i.e, det(A)=0
8         //inv(A)=(adj(A))/det(A)
9         //so A is not invertible
```

Scilab code Exa 4.7 matrix

```

1 //Example (pg no.144)
2
3 A = [1 2 3;4 5 6;7 8 9]
4 [L,U,E] = lu(A)
5     // L is lower triangular matrix(mxn)
6     // U is upper triangular matrix(mxmin(m,n))
7     // E is permutation matrix(min(m,n)xn)
8 A*E

```

Scilab code Exa 4.8 Backward substitution

```

1 //example 4.1 (pg 149)
2
3     //2x1 + 3x2 - x3 = 5
4     //-2x2 - x3 = -7
5     //-5x3 = -15
6
7 A = [2 3 -1;0 -2 -1;0 0 -5]
8 b = [5 -7 -15]'
9 a=[A b]
10 [nA ,mA]=size(A)
11 n=nA
12
13     //Backward substitution
14
15 x(3) = a(n,n+1)/a(n,n);
16
17 for i = n-1:-1:1
18     sumk=0;
19     for k=i+1:n
20         sumk=sumk+a(i,k)*x(k);
21     end
22     x(i)=(a(i,n+1)-sumk)/a(i,i);
23 end
24 disp(x)

```

Scilab code Exa 4.14 norm

```
1 //example (3.14)
2
3 c=[-3 1 0 1]
4 p3=poly(c, 'y', 'coeff')
5 roots(p3)
```

Scilab code Exa 4.16 determinant

```
1 //Example (pg no.186)
2
3
4 //(1)
5     A=[1 4;2 3]
6     det(A)
7
8 //(2)
9     A=[4 1;3 2]
10    det(A)
```

Scilab code Exa 4.17 determinant

```
1 //Example (pg no.186)
2
3     A=[3.1 4;3.2 3]
4     det(A)
```

Scilab code Exa 4.18 determinant

```
1 //Example (pg no.186)
2
3     A=[1 2;2 2]
4     B=[1 2;1 1]
5     det(A)+det(B)
6     C=[1 2; 3 3]
7     det(C)
8     //det(A)+det(B)=det(C)
```

Scilab code Exa 4.19 Eigen values

```
1 //Example(4.11) (pg no.191)
2
3     B=[1 2 0;2 1 0;0 0 -1]
4     lam = spec(B)
```

Scilab code Exa 4.20 Eigen values

```
1 //Example(4.14) (pg no.201)
2
3
4 B=[1 2 0;2 1 0;0 0 -1]
5 lam = spec(B)
6 norm(B)
7     //Each eigen value of the matrix must have
8     // absolute value
9     // no bigger than the norm of that
10    matrix
```

Scilab code Exa 4.21 determinant

```
1 //Example(4.15) (pgno 202)
2
3 B=[1 2 0;2 1 0;0 0 -1]
4 I=[1 0 0;0 1 0;0 0 1]
5 //here we are taking lamda=a
6 //det(B-a)*I=0 is characteristic equation
  to get lamda
7 deff ('[y]=pa', 'pa=det(B-a)*I')
8 pa=0
```

Scilab code Exa 4.22 eigen values

```
1 //Example(4.16) (page no.203)
2
3 A=[4 -1 -1 -1;-1 4 -1 -1;-1 -1 4 -1;-1 -1 -1 4]
4 spec(A)
```

Chapter 5

systems of equations and unconstrained optimization

Scilab code Exa 5.1 gradient

```
1
2 //Example 5.1
3
4 def ('y=f(x)', 'y=((x1)^3)+((x2)^3)-2*((x1)^2)+3*((x2)
   )^2)-8')
5 funcprot(0)
6 def ('y=g(x)', 'y=3*((x1)^2)-4*(x1)+3*((x2)^2)+6*(x2)
   ')
7           // f1=(df/dx1)(x)=0 , f2=(df/dx2)(x)=0
8   def ('y=fp(x)', 'y=3*((x1)^2)-4*(x1)')
9   def ('y=fpp(x)', 'y=3*((x2)^2)+6*(x2)')
10      x1=poly(0,"x1")
11   fp=(3*((x1)^2)-4*(x1))
12   p=roots(fp)
13
14   x2=poly(0,"x2")
15   fpp=3*((x2)^2)+6*(x2)
16   p=roots(fpp)
```

Scilab code Exa 5.2 Steep descent

```
1
2 //Example 5.2
3
4 deff ('[y]=f(x1,x2)', 'y=((x1)^3)+((x2)^3)-2*((x1)^2)
      +3*((x2)^2)-8')
5     //x1=1 , x2=-1
6     //(del)f(X(0))=[3*((x1)^2)-4*x1,3*((x2)^2)+6*x2
      ]'=[-1,-3]'
7     //Thus , in the first step of steep descent ,
8     // we look for a minimum of the function
9     funcprot(0)
10 deff ('[y]= g(t)', 'y=((1+t)^3)+((-1+3*t)^3)-2*((1+t)
      ^2)+3*((-1+3*t)^2)-8')
11     //g'(t)=3*((1+t)^2)+3*3*((-1+3*t)^2)-4*(1+t)
      +3*2*(-1+3*t)
12     t=poly(0,"t")
13 y=3*((1+t)^2)+3*3*((-1+3*t)^2)-4*(1+t)+3*2*3*(-1+3*t)
      )
14 p=roots(y)
15     // We choose the positive root t=1/3
16     t=1/3;
17     x1=1+t
18     x2=-1+3*t
19     a=3*((x1)^2)-4*x1
20     b=3*((x2)^2)+6*x2
21     funcprot(0)
22 deff ('[y]=fp(x1)', 'y=(3*((x1)^2)-4*(x1))')
23
24     x1=poly(0,"x1")
25     fp=(3*((x1)^2)-4*(x1))
26     p=roots(fp)
```

Chapter 6

Approximation

Scilab code Exa 6.1 uniform approximation

```
1 //Example 6.1
2
3 def f(' [y]=f(x) ', 'y=exp(x) ')
4 x0=-1
5 x1=0
6 x2=1
7 // F=f(x0, x1, x2)=f(-1, 0, 1)
8 F=f(x0)/[(x1-x0)*(x2-x0)]+f(x1)/[(x0-x1)*(x2-x1)]+f(
    x2)/[(x0-x2)*(x1-x2)]
9 // W(-1,0,1)=2 and so, for a<= -1, 1 <=b
10 // |f[-1,0,1]|/2 <= dist(at infinity)(f, pi1)*****
11 // dist(exp(x), pi1) >= 0.27154
```

Scilab code Exa 6.2 distance at infinity

```
1 //Example 6.2
2
3 def f(' [y]=f(x) ', 'y =tan((%pi/4)*x) ')
```



```

4
5 // on std interval -1 <= x <= 1 from pi3
6 // this is an odd function f(-x)=f(x)
7 n=4
8 p= (1/(2*(n+1)))*(f(1)-2*f(cos(%pi/(n+1)))+2*f(cos
      (2*%pi/(n+1)))-2*f(cos(3*%pi/(n+1)))+2*f(cos(4*
      %pi/(n+1)))-f(-1))
9 // 0.00203 <= dist(at infinity)(f,pi4)=p=0.0041

```

Scilab code Exa 6.3 aproximation

```

1 //Example 6.3
2
3 deff(' [y]=f(x) ', 'y=exp(x) ')
4 xset('window',0);
5 x=-1:.01:1; // defining the range of
      x.
6 y=feval(x,f);
7
8 a=gca();
9
10 a.y_location = "origin";
11
12 a.x_location = "origin";
13 plot(x,y) // instruction to plot the
      graph
14
15
16
17 // possible approximation
18 // y = q1(x)
19
20 // Let e(x) = exp(x) - [a0+a1*x]
21 // q1(x) & exp(x) must be equal at two points in
      [-1,1], say at x1 & x2

```

```

22 //      sigma1 = max(abs(e(x)))
23 //      e(x1) = e(x2) = 0.
24 //      By another argument based on shifting the
graph of y = q1(x),
25 //      we conclude that the maximum error sigma1 is
attained at exactly 3 points.
26 //      e(-1) = sigma1
27 //      e(1) = sigma1
28 //      e(x3) = -sigma1
29 //      x1 < x3 < x2
30 //      Since e(x) has a relative minimum at x3, we
have e'(x) = 0
31 //      Combining these 4 equations, we have..
32 //      exp(-1) - [a0-a1] = sigma1 -----(
i)
33 //      exp(1) - [a0+a1] = p1 -----(
ii)
34 //      exp(x3) - [a0+a1*x3] = -sigma1 -----(
iii)
35 //      exp(x3) - a1 = 0 -----(
iv)
36
37 //      These have the solution
38
39 a1 = (exp(1) - exp(-1))/2
40 x3 = log(a1)
41 sigma1 = 0.5*exp(-1) + x3*(exp(1) - exp(-1))/4
42 a0 = sigma1 + (1-x3)*a1
43
44 x = poly(0,"x");
45 //      Thus,
46 q1 = a0 + a1*x
47
48 deff(' [y1]=f(x) ', 'y1=1.2643+1.1752*x ')
49
50 xset('window',0);
51 x=-1:.01:1;          // defining the range of
x.

```

```

52 y=feval(x,f);
53
54 a=gca();
55
56 a.y_location = "origin";
57
58 a.x_location = "origin";
59 plot(x,y) // instruction to plot the
    graph

```

Scilab code Exa 6.5 approximate

```

1 //Example 6.5
2
3 //xn=10+(n-1)/5
4 //Accordingly we choose
5 //phi1(x)=1 , phi2(x)=x ,phi3(x)=(x)^3
6 A=[6 63 662.2; 63 662.2 6967.8; 662.2 6967.8
    73393.5664]
7 norm(A,'inf')
8 x=[10.07 -2 0.099]'
9 A*x
10 norm(A*x,'inf')
11 norm(A*x)
12 a=(norm(x))/norm((A)^(-1))
13
14 //norm(A*x) >=norm(x)/norm((A)^(-1))
15 // norm(A^(-1),'inf') >= 7.8
16
17 cond(A)
18
19 //the condition number of A is much larger than
    10^5, so we take
20 deff(' [y]=f(x) ', 'y=10-2*x+(((x)^2))/2 ')
21 //f(x) is a polynomial of degree 2 ,F*(x) should be

```

```

    f(x) itself
22
23 c1=10
24 c2=-2
25 c3=0.1
26
27 // by using elimination algorithm (Gauss elimination
    ), we get
28 c1=9.99999997437
29 c2=-1.999999951
30 c3=0.099999976
31 // by 14-decimal digit floating point arith metic
    method
32 c1=6.035
33 c2=-1.243
34 c3=0.0639
35 // calculation carried out in seven decimal digit
    floating point arithmetic
36 c1=8.492
37 c2=-1.712
38 c3=0.0863

```

Scilab code Exa 6.11 polynomial of degree less than 3

```

1
2 //Example 6.11
3
4 x0=-1
5 x1=18
6 //pi=<f, pi>
7 p0=integrate('exp(x)', 'x', x0, x1)
8
9 p1=integrate('x*exp(x)', 'x', x0, x1)
10
11 p2=integrate('(exp(x))*((x^2)-(1/3))', 'x', x0, x1)

```

```

12
13 p3=integrate(' (exp(x)) * ((x^3)-3*x/5)', 'x', x0, x1)
14
15 //for legendre polynomials one can show
16 //si= <pi, pi> = 2/(2*i+1)
17 s0=2/(2*0+1)
18 s1=2/(2+1)
19 s2=2/(2*2+1)
20 s3=2/(2*3+1)
21
22 //di*=<f, pi>/si
23 //p*(x)=y=d0*1+d1*x+d2*(3/2)*((x^2)-(1/3))+d3*((x^3)
    -3*x/5)*(5/2)
24 //p*(x)=y=(p0/s0)*1+(p1/s1)*x+(p2/s2)*(3/2)*((x^2)
    -(1/3))+((p3/s3)*((x^3)-3*x/5)*(5/2)
25 poly(0, "x")
26 y=1.17552011*(1)+(1.103638324)*(x)+(0.3578143506)
    *(3/2)*((x^2)-(1/3))+((0.07045563367)*((x^3)-3*x
    /5)*(5/2)
27 //On (-1,1), this polynomial a maximum deviation
    from exp(x) of about 0.01

```

Scilab code Exa 6.12 Least squares approximation

```

1 //Example 6.12
2
3 //Least squares approximation
4
5 def f(' [y]=f(x)', 'y=10-2*x+((x^2)/10)')
6 //we seek the polynomial of degree <= 2 which
    minimizes
7 //sum(n=1 to 9) [fn-p(xn)]^2
8 //we are dealing with scalar product with w(x)=1
9 P0x=1
10 //hence

```

```

11 s0=0;
12 B=0;
13 A1=0;
14 s1=0;
15 for n=1:1:6
16
17 s0=s0+1
18 B=[10+(n-1)/5]+B
19
20 A1=[10+[n-1]/5]*{[(n-1)/5]-0.5]^2}+A1
21
22 s1={[(n-1)/5]-0.5]^2}+s1
23
24 end
25 B0=B/s0
26
27 B1=A1/s1
28 C1=s1/s0
29
30 x=poly(0,"x")
31 y1=x-B0
32 x=poly(0,"x")
33 y2=((x-B0)^2)-0.1166667
34 //similarly calculate s2
35 s2=0.05973332
36 //p*(x)=(d0*)*P0x+(d1*)*P1(x)+(d2*)*P2(x)
37 //d0*=d0,d1*=d1,d2*=d2 are least squares
    approximation
38 //d0*=d0=sigma(n=1 to 6)[fn/6]           where fn=f(xn)
39
40 d0=0.03666667
41 d1=0.1
42 d2=0.09999999
43
44 x=poly(0,"x")
45 p=d0+d1*(x-B0)+d2*{[(x-B0)^2]-C1}
46 //c1=c1*,c2=c2*,c3=c3*
47 c1=9.99998

```

48 $c_2 = -1.9999998$

49 $c_3 = 0.0999999$

Chapter 7

differentiation and integration

Scilab code Exa 7.1 integration

```
1 //I=integral(exp^(-x^2) dx)
2
3 def f('y=f(x)', 'y=exp(-(x^2))')
4 a=0, b=1
5 c=(a+b)/2
6 def g('y=g(x)', 'y=-2*x*exp(-(x^2))')
7 f(a)
8 f(b)
9 f(c)
10 g(a)
11 g(b)
12 g(c)
13 R=(b-a)*f(a)
14 M=(b-a)*f(c)
15 T=(b-a)*[f(a)+f(b)]/2
16 S=(b-a)*{f(a)+4*f(c)+f(b)}/6
17 CT=[(b-a)/2]*[f(a)+f(b)]+[(b-a^2)/12]*[g(a)-g(b)]
```

Scilab code Exa 7.8 adaptive quadrature


```

1 //Example 7.8
2
3 // True value of the integral
4 x0=0
5 x1=1
6 I=integrate('sqrt(x)', 'x', 0, 1)
7
8 //using adaptive quadrature based on simpsons rule
9
10 deff(' [y]=f(x)', 'y=[(x)^(1/2)] ')
11 x=1:1:10
12 plot(x, f)
13
14 x2=(x0+x1)/2;
15 h=1/2
16 //considering the interval [x2,x1]=[1/2,1]
17
18 s=h/6.*{f(x2)+4*f((x2)+h/2)+f(x1)}
19 p=h/12.*{f(x2)+4*f((x2)+h/4)+2*f((x2)+h/2)+4*f(x2+3*h
    /4)+f(x1)}
20 E=(1/15)*(p-s)
21 ////Error criterion is clearly satisfied , hence we
    put value of p to SUM register to obtain partial
    approximation
22 //considering the interval [x2,x1]=[0,1/2]
23
24 s1=h/6.*{f(x0)+4*f((x0)+h/2)+f(x2)}
25 p1=h/12.*{f(x0)+4*f((x0)+h/4)+2*f((x0)+h/2)+4*f(x0
    +3*h/4)+f(x2)}
26 E1=1/15.*[p1-s1]
27
28 // Here since error is not less than 0.00025 we have
    to divide interval[0,1/2] to [0,1/4]& [1/4,1/2]
29 h=1/4
30 //considering interval [1/4,1/2]
31
32 x3=1/4
33

```

```

34 s2=h/6.*{f(x3)+4*f((x3)+h/2)+f(x2)}
35 p2=h/12.*{f(x3)+4*f((x3)+h/4)+2*f((x3)+h/2)+4*f(x3
    +3*h/4)+f(x2)}
36 E2=1/15.*[p2-s2]
37
38 // E2 < (0.0005)/4
39 //Error criterion is clearly satisfied , hence we
    add value of p2 to SUM register to obtain partial
    approximation
40 sum=p+p2
41 funcprot(0)
42 //Applying again above basic formulas
43
44 //with h=1/4 , in interval [0,1/4]
45 // we get
46
47 s3=0.07975890
48 p3=0.08206578
49 E3=0.0001537922
50 // Here since error is not less than 0.000125 we
    have to divide interval
51 // [0,1/4] in to [0,1/8]& [1/8,1/4] with h=1/8
52 h=1/8
53
54 // for interval [1/8,1/4]
55
56 s4=0.05386675
57 p4=0.05387027
58 E4=0.0000002346
59
60
61 // E4 < (0.0005)*h =(0.0005)/8 =0.0000625
62 //Error criterion is clearly satisfied , hence we
    add value of p4 to
63 //SUM register to obtain partial approximation
64 sum=p+p2+p4
65
66

```

```
67 // consider interval [0,1/8]
68
69 s5=0.02819903
70 p5=0.02901464
71 E5=0.00005437
72
73 // E5 < 0.0000625
74
75 //Since the error test is passed on both intervals ,
    we can add these values in to sUM register to
    get
76 sum=p+p2+p4+p5
77
78 // since the exact value of I is 0.666666666
79 abs(sum-I) <0.0005 // over the interval [0,1]
```

Chapter 8

THE SOLUTION OF DIFFERENTIAL EQUATIONS

Scilab code Exa 8.1 Taylor series

```
1 // Example 8.1
2
3 deff ('[v]=f(x,y)', 'v=1-(y/x)')
4 funcprot(0)
5 deff ('[v]=fp(x,y)', 'v=-(f(x)/x)+(y/(x^2))')
6 funcprot(0)
7 deff ('[v]=fpp(x,y)', 'v=-(fp(x)/x)+2*(f(x)/(x^2))-2*(
      y/(x^3))')
8 funcprot(0)
9 deff ('[v]=g(x,y)', 'v=-(fpp(x)/x)+3*(fp(x)/(x^2))-6*(
      f(x)/(x^3))+6*(y/(x^4))')
10 funcprot(0)
11 x1=2
12 y1=2
13 x=2.1
14 y2=y1+(x-2)*f(x1,y1)+((x-2)^2)*fp(x1,y1)/factorial
      (2)+((x-2)^3)*fpp(x1,y1)/factorial(3)+((x-2)^4)*g
```

```
(x1,y1)/factorial(4)
```

check Appendix [AP 2](#) for dependency:

```
adamsbash.sce
```

Scilab code Exa 8.5 Adamsbashforth3

```
1 // Example 8.5
2
3 def f(' [v]=f(x,y) ', 'v=x+y ')
4
5 [y,x]= adamsbashforth3(0,0,1,1/32,f)
```

check Appendix [AP 1](#) for dependency:

```
modifiedEuler.sce
```

Scilab code Exa 8.6 Modified euler method

```
1 // Example 8.6
2
3 // Modified Eulers method
4
5 def f(' [v]=f(x,y) ', 'v=x-(1/y) ')
6
7 [y,x] = modifiedeuler(1,0,0.2,0.1,f)
```

Appendix

Scilab code AP 1 Modified euler method

```
1 function [u,t] = modifiedeuler(u0,t0,tn,h,f)
2
3 //modifiedeuler 1st order method solving ODE
4 // du/dt = f(u,t), with initial
5 //conditions u=u0 at t=t0. The
6 //solution is obtained for t = [t0:h:tn]
7 //and returned in u
8
9 umaxAllowed = 1e+100;
10
11 t = [t0:h:tn]; u = zeros(t); n = length(u); u(1) =
    u0;
12
13 for j = 1:n-1
14     k1=h*f(t(j),u(j));
15     k2=h*f(t(j)+h/2,u(j)+k1/2);
16     u(j+1) = u(j) + k2;
17     if u(j+1) > umaxAllowed then
18         disp('Euler 1 - WARNING: underflow or
                overflow ');
19         disp('Solution sought in the following
                range: ');
20         disp([t0 h tn]);
21         disp('Solution evaluated in the following
                range: ');
22         disp([t0 h t(j)]);
```

```

23         n = j; t = t(1,1:n); u = u(1,1:n);
24     break;
25     end;
26 end;
27
28 endfunction

```

Scilab code AP 2 adamsbashforth

```

1
2 function [u,t] = adamsbashforth3(u0,t0,tn,h,f)
3
4 //adamsbashforth3 3rd order method solving ODE
5 // du/dt = f(u,t), with initial
6 //conditions u=u0 at t=t0. The
7 //solution is obtained for t = [t0:h:tn]
8 //and returned in u
9
10 umaxAllowed = 1e+100;
11
12 t = [t0:h:tn]; u = zeros(t); n = length(u); u(1) =
    u0;
13 for j = 1:n-1
14     if j<3 then
15         k1=h*f(t(j),u(j));
16         k2=h*f(t(j)+h,u(j)+k1);
17         u(j+1) = u(j) + (k2+k1)/2;
18     end;
19
20     if j>=2 then
21         u(j+2) = u(j+1) + (h/12)*(23*f(t(j+1),u(j+1))
            -16*f(t(j),u(j))+5*f(t(j-1),u(j-1)));
22     end;
23 end;
24 endfunction

```

Scilab code AP 3 secant mehod

```

1 function [x]=secant(a,b,f)
2     N=100;                // define max. no. iterations
                          // to be performed
3     PE=10^-4             // define tolerance for
                          // convergence
4     for n=1:1:N          // initiating for loop
5         x=a-(a-b)*f(a)/(f(a)-f(b));
6         disp(x)
7         if abs(f(x))<=PE then break; //checking for
                          // the required condition
8         else a=b;
9             b=x;
10        end
11    end
12    disp(n," no. of iterations =") //
13 endfunction

```

Scilab code AP 4 bisection method

```

1 function x=bisection(a,b,f)
2     N=100;                //
                          // define max. number of iterations
3     PE=10^-4             //
                          // define tolerance
4     if (f(a)*f(b) > 0) then
5         error('no root possible f(a)*f(b) > 0')
6         // checking if the decided range is
7         // containing a root
8         abort;
9     end;
10    if(abs(f(a)) < PE) then
11        error('solution at a') //
12        // seeing if there is an approximate root
13        // at a,
14        abort;
15    end;
16    if(abs(f(b)) < PE) then
17        // seeing if there is an approximate root at b,

```



```

13     error('solution at b')
14     abort;
15     end;
16     x=(a+b)/2
17     for n=1:1:N                               //
18         initialising 'for' loop,
19         p=f(a)*f(x)
20         if p<0 then b=x ,x=(a+x)/2;
21             //checking for the required conditions( f
22             (x)*f(a)<0),
23         else
24             a=x
25             x=(x+b)/2;
26         end
27         if abs(f(x))<=PE then break
28             // instruction to come out of the loop
29             after the required condition is achieved,
30         end
31     end
32     disp(n," no. of iterations =")
33         // display the no. of iterations took to
34         achive required condition,
35 endfunction

```

Scilab code AP 5 fixed point

```

1 function [x]=fixedp(x0,f)
2 //fixed-point iteration
3 N = 100; eps = 1.e-5; // define max. no. iterations
4 and error
5 maxval = 10000.0; // define value for divergence
6 a = x0;
7 while (N>0)
8     xn = f(a);
9     if(abs(xn-a)<eps) then
10         x=xn
11         disp(100-N);
12         return(x);

```

```

12     end;
13     if (abs(f(x))>maxval) then
14         disp(100-N);
15         error('Solution diverges');
16         abort;
17     end;
18     N = N - 1;
19     xx = xn;
20 end;
21 error('No convergence');
22 abort;
23 //end function

```

Scilab code AP 6 newton method

```

1 function x=newton(x,f,fp)
2     R=100;
3     PE=10^-8;
4     maxval=10^4;
5
6     for n=1:1:R
7         x=x-f(x)/fp(x);
8         if abs(f(x))<=PE then break
9         end
10        if (abs(f(x))>maxval) then error('Solution
11            diverges');
12            abort
13            break
14        end
15        disp(n," no. of iterations =")
16 endfunction

```

Scilab code AP 7 Regular falsi method

```

1 function [x]=regularfalsi(a,b,f)
2     N=100;
3     PE=10^-5;

```

```
4     for n=2:1:N
5         x=a-(a-b)*f(a)/(f(a)-f(b));
6         disp(x)
7         if abs(f(x))<=PE then break;
8         elseif (f(a)*f(x)<0) then b=x;
9             else a=x;
10        end
11    end
12    disp(n," no. of ittirations =")
13 endfunction
```
