Scilab Textbook Companion for An Introduction To Numerical Analysis by K. E. Atkinson[1]

Created by
Warsha B
B.Tech (pursuing)
Electronics Engineering
VNIT
College Teacher
G.NAGA RAJU
Cross-Checked by
Prashant Dave, IITB

August 10, 2013

[1]Funded 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: An Introduction To Numerical Analysis

Author: K. E. Atkinson

Publisher: John Wiley And Sons

Edition: 2

Year: 2001

ISBN: 8126518502

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

Chapter 1

Error Its sources Propagation and Analysis

Scilab code Exa 1.1 Taylor series

```
        //    PG (6)

//    Taylor series for e^(-x^2) upto first four
    terms

deff('[y]=f(x)','y=exp(-x^2)')
funcprot(0)
deff('[y]=fp(x)','y=-2*x*exp(-x^2)')
funcprot(0)
deff('[y]=fpp(x)','y=(1-2*x^2)*(-2*exp(-2*x^2))')
funcprot(0)
deff('[y]=g(x)','y=4*x*exp(-x^2)*(3-2*x^2)')
funcprot(0)
deff('[y]=gp(x)','y=(32*x^4*exp(-x^2))+(-72*x^2*exp
    (-x^2))+12*exp(-x^2)')
funcprot(0)
x0=0;
x=poly(0,"x");
T = f(x0) + (x-x0)*fp(x0)/factorial(1) + (x-x0)^2 *
```

```
    fpp(x0)/factorial(2) + (x-x0)^3 * g(x0)/factorial
    (3) + (x-x0)^4 * gp(x0)/factorial(4)



//      Similarily Taylor series for inv(tan(x))
```

Scilab code Exa 1.3 Vector norms

```
          //    PG (11)

A = [1 -1;3 2]
x = [1;2]
y = A*x
norm(A,'inf')
norm(x,'inf')
norm(y,'inf')

x = [1;1]
y = A*x
norm(y,'inf')
norm(A,'inf')*norm(x,'inf')

//    norm(y,'inf') = norm(A,'inf') * norm(x,'inf')
```

Scilab code Exa 1.4 Conversion to decimal

```
          //    PG (12)

//    11011.01 is a binary number. Its decimal
    equivalent is:

```

```
1*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0 + 0*2^(-1) +
    1*2^(-2)

//     56C.F is a hexadecimal number. Its decimal
    equivalent is:

5*16^2 + 6*16^1 + 12*16^0 + 15*16^(-1)
```

Scilab code Exa 1.5 Error and relative error

```
            //     PG (17)

xT = exp(1)
xA = 19/7

//     Error(xA)

xT - xA

//     Relative error, Rel(xA)

(xT-xA)/xT
```

Scilab code Exa 1.6 Errors

```
            //     PG (18)

xT = 1/3
xA = 0.333
abs(xT-xA)     //     Error

//——————————————————————————————————

```

```
xT = 23.496
xA = 23.494
abs(xT-xA)    //    Error

//———————————————————————————————————

xT = 0.02138
xA = 0.02144
abs(xT-xA)    //    Error
```

Scilab code Exa 1.7 Taylor series

```
        //    PG (20)

//    Taylor series for the first two terms

deff('[y]=f(x)','y=sqrt(1+x)')
funcprot(0)
deff('[y]=fp(x)','y=0.5*(1+x)^(-1/2)')
funcprot(0)
x0=0;
x=poly(0,"x");
T = f(x0) + (x-x0)*fp(x0)/factorial(1)
```

Scilab code Exa 1.8 Graph of polynomial

```
        //    PG (21)

deff('[y]=f(x)','y = x^3-3*x^2+3*x-1')
xset('window',0);
x=-0:.01:2;                                              //
   defining the range of x.
```

```
y=feval(x,f);

a=gca();

a.y_location = "origin";

a.x_location = "origin";
plot(x,y)
    // instruction to plot the graph

title(' y = x^3-3*x^2+3*x-1')
```

Scilab code Exa 1.9 Error and Relative error

```
        //    PG (24)

xT = %pi
xA = 3.1416
yT = 22/7
yA = 3.1429
xT - xA              //    Error
(xT - xA)/xT         //    Relative Error
yT - yA              //    Error
(yT - yA)/yT         //    Relative Error

(xT - yT) - (xA - yA)
((xT - yT) - (xA - yA))/(xT - yT)

//    Although the error in xA - yA is quite small,
//    the relative error in xA - yA is much larger
    than that in xA or yA alone.
```

Scilab code Exa 1.10 Loss of significance errors

```
        //     PG (25)

//     Consider solving ax^2 + b*x + c =


//     Consider a polynomial y = x^2 - 26*x + 1 = 0

x = poly(0,"x");
y = x^2 - 26*x + 1
p = roots(y)
ra1 = p(2,1)
ra2 = p(1,1)

//     Using the standard quadratic formula for
   finding roots,

rt1 = (-(-26)+sqrt((-26)^2 - 4*1*1))/(2*1)
rt2 = (-(-26)-sqrt((-26)^2 - 4*1*1))/(2*1)

//     Relative error

rel1 = (ra1-rt1)/ra1
rel2 = (ra2-rt2)/ra2

//     The significant errors have been lost in the
   subtraction ra2 = xa - ya.
//     The accuracy in ra2 is much less.
//     To calculate ra2 accurately, we use:

rt2 = ((13-sqrt(168))*(13+sqrt(168)))/(1*(13+sqrt
   (168)))
//     Now, rt2 is nearly equal to ra2. So, by exact
   calculations, we will now get a much better rel2.
```

Scilab code Exa 1.11 Loss of significance errors

```
        //    PG (26)

x = poly(0,"x");
x = 0;
deff('[y]=f(t)','y=exp(x*t)')
integrate('exp(x*t)','t',0,1)

//    So, for x = 0, f(0) = 1
//    f(x) is continuous at x = 0.

//    To see that there is a loss of significance
   problem when x is small,
//    we evaluate f(x) at 1.4*10^(-9)

x = 1.4*10^(-9)
integrate('exp(x*t)','t',0,1)
//    When we use a ten-digit hand calculator, the
   result is 1.000000001

//    To avoid the loss of significance error, we
   may use a quadratic Taylor approximation to exp(x
   ) and then simplify f(x).
```

Chapter 2

Rootfinding for Nonlinear equations

check Appendix AP 13 for dependency:

```
bisection1.sce
```

Scilab code Exa 2.1 Bisection method

```
    //    EXAMPLE (PG 57)
    //    To find largest root, alpha, of x^6 - x - 1 = 0
    //    using bisection method
    //    The graph of this function can also be observed here.

deff('[y]=f(x)','y=x^6-x-1')
                        //    It is straightforward to show that 1<alpha<2, and
                        //we will use this as our initial interval [a,b]


xset('window',0);
```

```
x=-5:.01:5;                                                          //
    defining the range of x.
y=feval(x,f);

a=gca();

a.y_location = "origin";

a.x_location = "origin";
plot(x,y)
    // instruction to plot the graph

title(' y = x^6-x-1')

// execution of the user defined function so as to
    use it in program to find the approximate
    solution.

// we call a user-defined function 'bisection' so as
     to find the approximate
// root of the equation with a defined permissible
    error.

bisection(1,2,f)
```

check Appendix AP 11 for dependency:

`newton.sce`

Scilab code Exa 2.2 Newton method

```
     //     EXAMPLE (PG 60)
     //     To find largest root, alpha, of f(x) = x^6
         - x - 1 = 0
```

```
    //    using newton's method


deff('[y]=f(x)','y=x^6-x-1')
deff('[y]=fp(x)','y=6*x^5-1')                    //
      Derivative of f(x)
x=(1+2)/2                       //    Initial
   appoximation

//we call a user-defined function 'newton' so as to
   find the approximate
// root of the equation with a defined permissible
   error.


newton(x,f,fp)
```

check Appendix AP 10 for dependency:

`secant.sce`

Scilab code Exa 2.3 Secant method

```
    //    EXAMPLE ( PG 66)
    //    To find largest root, alpha, of f(x) = x^6
        - x - 1 = 0
    //    using secant method

deff('[y]=f(x)','y=x^6-x-1')
a=1
b=2        //    Initial approximations


// we call a user-defined function 'secant' so as to
    find the approximate
```

```
// root of the equation with a defined permissible
    error.

secant(a,b,f)
```

check Appendix AP 9 for dependency:

`muller.sce`

Scilab code Exa 2.4 Muller method

```
    //    EXAMPLE1 (PG 76)
    //    f(x) = x^20 - 1
    //    solving using Muller's method


xset('window',1);
x=-2:.01:4;                                          //
     defining the range of x.
deff('[y]=f(x)','y=x^20-1');                       //
    defining the cunction.
y=feval(x,f);

a=gca();

a.y_location = "origin";

a.x_location = "origin";
plot(x,y)                                          //
    instruction to plot the graph
title(' y = x^20-1')

// from the above plot we can infre that the
    function has roots between
// the intervals (0,1),(2,3).

```

```
        //sollution by muller method to 3 iterations.

muller(0,.5,1,f)
```

check Appendix AP 9 for dependency:

`muller.sce`

Scilab code Exa 2.6 Muller method

```
    //     EXAMPLE3 (PG 76)
    //    f(x) = x^6- 12 * x^5 + 63 * x^4 - 216* x^3 + 567 * x^2 - 972 * x + 729
    //     or f(x) = (x^2+9)*(x-3)^4
    //     solving using Muller's method

deff('[y]=f(x)','y=(x^2+9)*(x-3)^4')

xset('window',2);
x=-10:.1:10;                                                      // defining the range of x.
y=feval(x,f);

a=gca();

a.y_location = "origin";

a.x_location = "origin";
plot(x,y)                                         // instruction to plot the graph

title(' y = (x^2+9)*(x-3)^4')

```

```

muller(0,.5,1,f)
```

Scilab code Exa 2.7 One point iteration method

```
    //      EXAMPLE (PG 77)
    //      x^2-a = 0

    //      The graph for x^2-3 can also be observed here.

deff('[y]=f(x)','y=x*x-3')
funcprot(0)
xset('window',3);
x=-2:.01:10;                                                    // defining the range of x.
y=feval(x,f);

a=gca();

a.y_location = "origin";

a.x_location = "origin";
plot(x,y)                                                       // instruction to plot the graph

title(' y = x^2-3')
    //          CASE 1

//We have f(x) = x^2-a.
//So, we assume g(x) = x^2+x-a and the value of a = 3

```

```
deff('[y]=g(x)','y=x^2+x-3')
funcprot(0)
x=2
for n=0:1:3
 g(x);
   x=g(x)
end

//          CASE 2

//We have f(x) = x^2-a.
//So, we assume g(x) = a/x and the value of a = 3

deff('[y]=g(x)','y=3/x')
funcprot(0)
x=2
for n=0:1:3
 g(x);
   x=g(x)
end

//          CASE 3

//We have f(x) = x^2-a.
//So, we assume g(x) = 0.5*(x+(a/x)) and the value
   of a = 3

deff('[y]=g(x)','y=0.5*(x+(3/x))')
funcprot(0)
x=2
for n=0:1:3
 g(x);
   x=g(x)
end
```

Scilab code Exa 2.8 One point Iteration method

```
    //    EXAMPLE (PG 81)

    //Assume alpha is a solution of x = g(x)

alpha=sqrt(3);

//    case 1


deff('[y]=g(x)','y=x^2+x-3')
deff('[z]=gp(x)','z=2*x+1')          //    Derivative
   of g(x)
gp(alpha)

//    case 2

deff('[y]=g(x)','y=3/x')
funcprot(0)
deff('[z]=gp(x)','z=3/x')          //    Derivative of
    g(x)
gp(alpha)

//    case 3

deff('[y]=g(x)','y=0.5*(x+(3/x))')
funcprot(0)
deff('[z]=gp(x)','z=0.5*(1-(3/(x^2)))')          //
      Derivative of g(x)
gp(alpha)
```

check Appendix AP 12 for dependency:

`aitken1.sce`

Scilab code Exa 2.10 Aitken

```
    //      EXAMPLE (PG 85)

    //      x(n+1) = 6.28 + sin(x(n))
    //      True root is alpha = 6.01550307297

    deff ('[y]=f(x)','f(x)=6.28+sin(x(n))')
    //      k=6.01550307297

//x=6.01550307297

deff('[y]=g(x)','y=cos(x)')


// we call a user-defined function 'aitken' so as to
   find the approximate
// root of the equation with a defined permissible
   error.


aitken(0.2,0.5,1,g)
```

Scilab code Exa 2.11 Multiple roots

```
    //      EXAMPLE (PG 87)

        //      f(x) = (x-1.1)^3 * (x-2.1)

c = [2.7951 -8.954 10.56 -5.4 1]
p4=poly(c,'x','coeff')
roots(p4)
deff('[y]=f(x)','y=(x-1.1)^3*(x-2.1)')
xset('window',0);
```

```
x=0:.01:3;                                          //
   defining the range of x.
y=feval(x,f);

a=gca();

a.y_location = "origin";

a.x_location = "origin";
plot(x,y)
   // instruction to plot the graph

title(' y = (x-1.1)^3*(x-2.1)')
```

Chapter 3

Interpolation Theory

check Appendix AP 8 for dependency:

`lagrange.sce`

Scilab code Exa 3.1 Lagrange formula

```
            //      PG (134)

X = [0, -1, 1]
Y = [1, 2, 3]
lagrange(X,Y)
```

check Appendix AP 8 for dependency:

`lagrange.sce`

Scilab code Exa 3.2 Lagrange Formula

```
            //      PG (136)


```

```
X=[0]
Y=[1]
deff('[y]=f(x)','y=log10(x)')
p=lagrange(X,Y)
```

check Appendix AP 8 for dependency:

`lagrange.sce`

Scilab code Exa 3.3 Lagrange formula

```
            //      PG (137)

X=[0,-1]
Y=[1,2]
deff('[y]=f(x)','y=log(x)')
deff('[y]=fp(x)','y=1/x')
deff('[y]=fpp(x)','y=-1/(x)^2')
p = lagrange(X,Y)
//      E = f(x)-p
e = 0.00005       //      for a four-place logarithmic
    table
```

Scilab code Exa 3.4 Divided differences

```
            //      PG (140)

X = [2.0,2.1,2.2,2.3,2.4]
X1 = X(1,1)
X2 = X(1,2)
X3 = X(1,3)
X4 = X(1,4)
```

```
X5 = X(1,5)
deff('[y]=f(x)','y=sqrt(x)')
Y = [f(X1) f(X2) f(X3) f(X4) f(X5)]
Y1 = Y(1,1)
Y2 = Y(1,2)
Y3 = Y(1,3)
Y4 = Y(1,4)
Y5 = Y(1,5)

//     Difference

//     f[X1,X2]
(f(X2) - f(X1))*10
//     f[X2,X3]
(f(X3) - f(X2))*10
//     f[X3,X4]
(f(X4) - f(X3))*10
//     f[X4,X5]
(f(X5) - f(X4))*10

//     D^2 * f[Xi]

((f(X3)-f(X2)) - (f(X2)-f(X1))) * 50
((f(X4)-f(X3)) - (f(X3)-f(X2))) * 50
((f(X5)-f(X4)) - (f(X4)-f(X3))) * 50
```

Scilab code Exa 3.6 Bessel Function

```
          //     PG (142)

//     Values of Bessel Function Jo(x)

//          x                    Jo(x)

//          2.0          0.2238907791
```

```
//          2.1          0.1666069803
//          2.2          0.1103622669
//          2.3          0.0555397844
//          2.4          0.0025076832
//          2.5         −0.0483837764
//          2.6         −0.0968049544
//          2.7         −0.1424493700
//          2.8         −0.1850360334
//          2.9         −0.2243115458

//     Calculate the value of x for which Jo(x) = 0.1
```

Scilab code Exa 3.7 Divided differences

```
          //     PG (144)

deff('[y]=f(x)','y=sqrt(x)')
funcprot(0)
deff('[y]=fp(x)','y=0.5/sqrt(x)')
funcprot(0)
deff('[y]=fpp(x)','y=−0.25*x^(−3/2)')
funcprot(0)
deff('[y]=fppp(x)','y=3*x^(−2.5)/8')
deff('[y]=fpppp(x)','y=−15*x^(−7/2)/16')

//     f[2.0,2.1,.....2.4] = −0.002084

fpppp(2.3103)/factorial(4)
```

Scilab code Exa 3.8 Newton forward difference

```
          //     PG (150)

```

```
X = [2.0,2.1,2.2,2.3,2.4]
X1 = X(1,1)
X2 = X(1,2)
X3 = X(1,3)
X4 = X(1,4)
X5 = X(1,5)
deff('[y]=f(x)','y=sqrt(x)')
Y = [f(X1) f(X2) f(X3) f(X4) f(X5)]
Y1 = Y(1,1)
Y2 = Y(1,2)
Y3 = Y(1,3)
Y4 = Y(1,4)
Y5 = Y(1,5)

//    Difference

//    f[X1,X2]
(f(X2) - f(X1))
//    f[X2,X3]
(f(X3) - f(X2))
//    f[X3,X4]
(f(X4) - f(X3))
//    f[X4,X5]
(f(X5) - f(X4))

//    D^2 * f[Xi]

((f(X3)-f(X2)) - (f(X2)-f(X1)))
((f(X4)-f(X3)) - (f(X3)-f(X2)))
((f(X5)-f(X4)) - (f(X4)-f(X3)))

//    D^3 * f[Xi]

((f(X4)-f(X3)) - (f(X3)-f(X2))) - ((f(X3)-f(X2)) - (
   f(X2)-f(X1)))
((f(X5)-f(X4)) - (f(X4)-f(X3))) - ((f(X4)-f(X3)) - (
   f(X3)-f(X2)))

```

```
//     D^4 * f[Xi]

(((f(X5)-f(X4)) - (f(X4)-f(X3))) - ((f(X4)-f(X3)) - (f(X3)-f(X2)))) - (((f(X4)-f(X3)) - (f(X3)-f(X2))) - ((f(X3)-f(X2)) - (f(X2)-f(X1))))

mu = 1.5;
x = 2.15;

p1 = f(X1) + mu * (f(X2) - f(X1))
p2 = p1 + mu*(mu-1)*((f(X3)-f(X2)) - (f(X2)-f(X1)))/2

//     Similarly, p3 = 1.466288
//                p4 = 1.466288
```

Chapter 4

Approximation of functions

Scilab code Exa 4.1 Error of approximating exponent of x

```
        //      PG (199)

x = poly(0,"x");
p3 = 1 + x + (1/2)*x^(2) + (1/6)*x^3
deff('[y]=f(x)','y=exp(x)')
funcprot(0)
x = -1:0.01:1;
f(x) - p3
```

Scilab code Exa 4.2 Minimax Approximation problem

```
        //      PG (200)

deff('[y]=f(x)','y=exp(x)')

xset('window',0);
x=-1:.01:1;                  // defining the range of
   x.
```

```
y=feval(x,f);

a=gca();

a.y_location = "origin";

a.x_location = "origin";
plot(x,y)                    // instruction to plot the graph



//     possible approximation
//         y = q1(x)

//     Let e(x) = exp(x) - [a0+a1*x]
//     q1(x) & exp(x) must be equal at two points in [-1,1], say at x1 & x2
//     sigma1 = max(abs(e(x)))
//     e(x1) = e(x2) = 0.
//     By another argument based on shifting the graph of y = q1(x),
//     we conclude that the maximum error sigma1 is attained at exactly 3 points.
//     e(-1) = sigma1
//     e(1) = sigma1
//     e(x3) = -sigma1
//     x1 < x3 < x2
//     Since e(x) has a relative minimum at x3, we have e'(x) = 0
//     Combining these 4 equations, we have..
//     exp(-1) - [a0-a1] = sigma1 --------------------(i)
//     exp(1) - [a0+a1] = p1 -------------------------(ii)
//     exp(x3) - [a0+a1*x3] = -sigma1 ---------------(iii)
```

```
//     exp(x3) - a1 = 0 -----------------------------(iv)

//     These have the solution

a1 = (exp(1) - exp(-1))/2
x3 = log(a1)
sigma1 = 0.5*exp(-1) + x3*(exp(1) - exp(-1))/4
a0 = sigma1 + (1-x3)*a1

x = poly(0,"x");
//     Thus,
q1 = a0 + a1*x

deff('[y1]=f(x)','y1=1.2643+1.1752*x')

xset('window',0);
x=-1:.01:1;                    // defining the range of x.
y=feval(x,f);

a=gca();

a.y_location = "origin";

a.x_location = "origin";
plot(x,y)                      // instruction to plot the graph
```

Scilab code Exa 4.3 Least squares approximation problem

```
          //     PG (205)

deff('[y]=f(x)','y=exp(x)')

```

```
x=-1:.01:1;                    // defining the range of x

//     Let r1(x) = b0 + b1(x)
//     Minimize
//          ||f-r1||^2 = integrate('(exp(x)-b0-b1*x)^2','x',-1,1) = F(b0,b1)
//     F = integrate('exp(2*x) + b0^2 + (b1^2)*(x^2) - 2*b0*x*exp(x) + 2*b0*b1*x','x',b0,b1)
//     To find a minimum, we set

//          df/db0 = 0
//          df/db1 = 0------------necessary conditions at a minimal point
//     On solving, we get the values of b0 & b1

b0 = 0.5*integrate('exp(x)','x',-1,1)
b1 = 1.5*integrate('x*exp(x)','x',-1,1)
r1 = b0+b1*x;
norm(exp(x)-r1,'inf')     //     least squares approximation

r3 = 0.996294 + 0.997955*x + 0.536722*x^2 + 0.176139*x^3
norm(exp(x)-r3,'inf')     //     cubic least squares approximation
```

Scilab code Exa 4.4 Weight functions

```
          //     PG (206)

//     The following are the weight functions of most interest in the
//     developments of this text:

```

```
//          w(x)=1                      a < = x < = b
//          w(x)=1/sqrt(1-x^2)          -1 < = x < = 1
//          w(x)=exp(-x)             0 < = x < infinity
//          w(x)=exp(-x^2)             -infinity < x <
    infinity
```

Scilab code Exa 4.5 Formulae

```
            //      PG (215)
//      for laguerre polynomials ,
//      L(n+1)=1*[2*n+1-x]*L(n)/(n+1) - n*L(n-1)/(n+1)
//      for Legendre polynomials ,
//      P(n+1)= (2*n +1)*x*P(n)/(n+1) - n*P(n-1)/(n+1)
```

Scilab code Exa 4.6 Formulae for laguerre and legendre polynomials

```
            //      PG (215)
//      for laguerre polynomials ,
//      L(n+1)=1*[2*n+1-x]*L(n)/(n+1) - n*L(n-1)/(n+1)
//      for Legendre polynomials ,
//      P(n+1)= (2*n +1)*x*P(n)/(n+1) - n*P(n-1)/(n+1)
```

Scilab code Exa 4.7 Average error in approximation

```
          //      PG (219)

deff('[y]=f(x)','y=exp(x)')

x=-1:.01:1;                         // defining the range of x

//      Let r1(x) = b0 + b1(x)
//      Minimize
//           ||f-r1||^2 = integrate('(exp(x)-b0-b1*x)^2','x',-1,1) = F(b0,b1)
//      F = integrate('exp(2*x) + b0^2 + (b1^2)*(x^2) - 2*b0*x*exp(x) + 2*b0*b1*x','x',b0,b1)
//      To find a minimum, we set

//           df/db0 = 0
//           df/db1 = 0-----------necessary conditions at a minimal point
//      On solving, we get the values of b0 & b1

b0 = 0.5*integrate('exp(x)','x',-1,1);
b1 = 1.5*integrate('x*exp(x)','x',-1,1);
r1 = b0+b1*x;
norm(exp(x)-r1,'inf');      //      least squares approximation

r3 = 0.996294 + 0.997955*x + 0.536722*x^2 + 0.176139*x^3;
norm(exp(x)-r3,'inf');      //      cubic least squares approximation

//      average error E

E = norm(exp(x)-r3,2)/sqrt(2)
```

Scilab code Exa 4.8 Chebyshev expansion coefficients

```
          //    PG (220)

deff('[y]=f(x)','y=exp(x)')

//    Chebyshev expansion coefficients for exp(x)
//    j = 0
C0=2*(integrate('exp(cos(x))','x',0,3.14))/(3.14)

//    j = 1
C1=2*(integrate('exp(cos(x))*cos(x)','x',0,3.14))/(3.14)

//    j = 2
C2=2*(integrate('exp(cos(x))*cos(2*x)','x',0,3.14))/(3.14)

//    j = 3
C3=2*(integrate('exp(cos(x))*cos(3*x)','x',0,3.14))/(3.14)

//    j = 4
C4=2*(integrate('exp(cos(x))*cos(4*x)','x',0,3.14))/(3.14)

//    j = 5
C5=2*(integrate('exp(cos(x))*cos(5*x)','x',0,3.14))/(3.14)

//    we obtain
c1=1.266+1.130*x;
c3=0.994571+0.997308*x+0.542991*x^2+0.177347*x^3;
norm(exp(x)-c1,'inf')
```

```
norm(exp(x)-c3,'inf')
```

Scilab code Exa 4.9 Max errors in cubic chebyshev least squares approx

```
          //    PG (223)

deff('[y]=f(x)','y=exp(x)')

x=[-1.0 -0.6919 0.0310 0.7229 1.0];
   // defining x

r3 = 0.996294 + 0.997955*x + 0.536722*x^2 +
   0.176139*x^3;
norm(exp(x)-r3,'inf');    //    cubic least squares
   approximation
deff('[y]=g(x)','y=0.994571+0.997308*x+0.542991*x
   ^2+0.177347*x^3')
//    c3=g(x);
x1=x(1,1);
(exp(x1)-g(x1))
x2=x(1,2);
(exp(x2)-g(x2))
x3=x(1,3);
(exp(x3)-g(x3))
x4=x(1,4);
(exp(x4)-g(x4))
x5=x(1,5);
(exp(x5)-g(x5))
```

Scilab code Exa 4.10 Near minimax approximation

```
          //    PG (227)

```

```
deff('[y]=f(x)','y=exp(x)')
c3=0.994571+0.997308*x+0.542991*x^2+0.177347*x^3;
norm(exp(x)-c3,'inf')

//    as obtained in the example 6, c4 = 0.00547, T4
    (x) = (-1)
//    c4*T4(x) = 0.00547 * (-1)
//    norm(exp(x)-q3,'inf') = 0.00553
```

Scilab code Exa 4.11 Forced oscillation of error

```
        //    PG (234)

deff('[y]=f(x)','y=exp(x)')
x = -1:0.01:1;
//    For
n = 1;
x = [-1 0 1];
E1 = 0.272;
F1 = 1.2715 + 1.1752*x;

//    Relative errors

x = -1.0;
exp(x) - F1;
r1 = ans(1,1)
x = 0.1614;
exp(x) - F1;
r2 = ans(1,2)
x = 1.0;
exp(x) - F1;
r3 = ans(1,3)

F3 = 0.994526 + 0.995682*x + 0.543981*x*x +
    0.179519*x*x*x;
```

```
x = [-1.0 -0.6832 0.0493 0.7324 1.0]
exp(x) - F3     //    relative errors
```

Chapter 5

Numerical Integration

Scilab code Exa 5.1 Integration

```
            //      PG (250)

deff('[y]=f(x)','y=(exp(x)-1)/x')
x0=0;
x1=1;
integrate('(exp(x)-1)/x','x',x0,x1)
```

Scilab code Exa 5.2 Trapezoidal rule for integration

```
            //      PG (254)

deff('[y]=f(x)','y=exp(x)*cos(x)')
deff('[y]=fp(x)','y=exp(x)*(cos(x)-sin(x))')
deff('[y]=fpp(x)','y=-2*exp(x)*sin(x)')
x0=0;
x1=%pi;


```

```
//     True value
integrate('exp(x)*cos(x)','x',x0,x1)

//     Using Trapezoidal rule

n=2;
h=(x1-x0)/n;
I1 = (x1-x0) * (f(x0)+f(x1)) /4
E1 = -h^2 * (fp(x1)-fp(x0)) /12

n=4;
h=(x1-x0)/n;
I2 = (x1-x0) * (f(x0)+f(x1)) /4
E2 = -h^2 * (fp(x1)-fp(x0)) /12

n=8;
h=(x1-x0)/n;
I3 = (x1-x0) * (f(x0)+f(x1)) /4
E3 = -h^2 * (fp(x1)-fp(x0)) /12

n=16;
h=(x1-x0)/n;
I4 = (x1-x0) * (f(x0)+f(x1)) /4
E4 = -h^2 * (fp(x1)-fp(x0)) /12

n=32;
h=(x1-x0)/n;
I5 = (x1-x0) * (f(x0)+f(x1)) /4
E5 = -h^2 * (fp(x1)-fp(x0)) /12

n=64;
h=(x1-x0)/n;
I6 = (x1-x0) * (f(x0)+f(x1)) /4
E6 = -h^2 * (fp(x1)-fp(x0)) /12

n=128;
h=(x1-x0)/n;
I7 = (x1-x0) * (f(x0)+f(x1)) /4
```

```
E7 = -h^2 * (fp(x1)-fp(x0)) /12
```

Scilab code Exa 5.3 Corrected trapezoidal rule

```
        //    PG (255)

deff('[y]=f(x)','y=exp(x)*cos(x)')
deff('[y]=fp(x)','y=exp(x)*(cos(x)-sin(x))')
deff('[y]=fpp(x)','y=-2*exp(x)*sin(x)')
x0=0;
x1=%pi;


//    True value
integrate('exp(x)*cos(x)','x',x0,x1)

//    Using Corrected Trapezoidal rule

n=2;
h=(x1-x0)/n;
I1 = ((x1-x0)/2) * (f(x0)+f(x1)) /2
E1 = -h^2 * (fp(x1)-fp(x0)) /12
C1 = I1 + E1

n=4;
h=(x1-x0)/n;
I2 = ((x1-x0)/2) * (f(x0)+f(x1)) /2
E2 = -h^2 * (fp(x1)-fp(x0)) /12
C2 = I2 + E2

n=8;
h=(x1-x0)/n;
I3 = ((x1-x0)/2) * (f(x0)+f(x1)) /2
E3 = -h^2 * (fp(x1)-fp(x0)) /12
C3 = I3 + E3
```

```
n=16;
h=(x1-x0)/n;
I4 = ((x1-x0)/2) * (f(x0)+f(x1)) /2
E4 = -h^2 * (fp(x1)-fp(x0)) /12
C4 = I4 + E4

n=32;
h=(x1-x0)/n;
I5 = ((x1-x0)/2) * (f(x0)+f(x1)) /2
E5 = -h^2 * (fp(x1)-fp(x0)) /12
C5 = I5 + E5

n=64;
h=(x1-x0)/n;
I6 = ((x1-x0)/2) * (f(x0)+f(x1)) /2
E6 = -h^2 * (fp(x1)-fp(x0)) /12
C6 = I6 + E6
```

Scilab code Exa 5.4 Simpson s rule for integration

```
        //      PG (258)

deff('[y]=f(x)','y=exp(x)*cos(x)')
x0=0;
xn=%pi;
x=x0:xn;

//      True value

I = integrate('exp(x)*cos(x)','x',x0,x1)

//      Using Simpson's rule

N=2;
```

```
h=(xn-x0)/N;
x1=x0+h;
x2=x0+2*h;
    I1 = h*(f(x0)+4*f(x1)+f(x2))/3

N=4;
h=(xn-x0)/N;
x1=x0+h;
x2=x0+2*h;
x3=x0+3*h;
x4=x0+4*h;
    I2 = h*(f(x0)+4*f(x1)+2*f(x2)+4*f(x3)+f(x4))/3

N=8;
h=(xn-x0)/N;
x1=x0+h;
x2=x0+2*h;
x3=x0+3*h;
x4=x0+4*h;
x5=x0+5*h;
x6=x0+6*h;
x7=x0+7*h;
x8=x0+8*h;
    I3 = h*(f(x0)+4*f(x1)+2*f(x2)+4*f(x3)+2*f(x4)+4*
       f(x5)+2*f(x6)+4*f(x7)+f(x8))/3
```

Scilab code Exa 5.5 Trapezoidal and simpson integration

```
        //    PG (261)

//     Example 1

deff('[y]=f(x)','y=x^(7/2)')
deff('[y]=fp(x)','y=3.5*x^(5/2)')
deff('[y]=fpp(x)','y=8.75*x^(3/2)')
```

```
deff('[y]=fppp(x)','y=(105*sqrt(x))/8')
deff('[y]=fpppp(x)','y=(105*x^(-0.5))/16')

x0=0;
x1=1;
x=x0:x1;

//    True value
I = integrate('x^(7/2)','x',x0,x1)

//    Using Trapezoidal rule

n=2;
h=(xn-x0)/n;
I1 = (xn-x0) * (f(x0)+f(xn)) /4;
E1 = -h^2 * (fp(xn)-fp(x0)) /12          //    Error

n=4;
h=(xn-x0)/n;
I2 = (xn-x0) * (f(x0)+f(xn)) /4;
E2 = -h^2 * (fp(xn)-fp(x0)) /12          //    Error

//    Using Simpson's rule

N=2;
h=(xn-x0)/N;
x1=x0+h;
x2=x0+2*h;
    I1 = h*(f(x0)+4*f(x1)+f(x2))/3
    E1 = -h^4*(xn-x0)*fpppp(0.5)/180

N=4;
h=(xn-x0)/N;
x1=x0+h;
x2=x0+2*h;
x3=x0+3*h;
x4=x0+4*h;
    I2 = h*(f(x0)+4*f(x1)+2*f(x2)+4*f(x3)+f(x4))/3
```

```
    E2 = -h^4*(xn-x0)*fpppp(0.5)/180
```

Scilab code Exa 5.6 Newton Cotes formulae

```
          //      PG (266)

//      Commonly used Newton Cotes formulae:-

//      n=1

//      h/2 * [f(a)+f(b)] - (h^3)*f''(e)/12---------Trapezoidal rule

//      n=2

//      h/3 * [f(a)+4*f((a+b)/2)+f(b)] - (h^5)*f^(4)(e)/90------Simpson's rule

//      n=3

//      3*h/8 * [f(a)+3*f(a+h)+3*f(b-h)+f(b)] - (3*h^5)*f^(4)(e)/80

//      n=4

//      2*h/45 * [7*f(a)+32*f(a+h)+12*f((a+b)/2)+32*f(b-h)+7*f(b)] - (8*h^7)*f^(7)(e)/945
```

check Appendix AP 6 for dependency:

`legendrepol.sce`

Scilab code Exa 5.7 Gaussian Quadrature

```
            //      PG (277)

deff('[y]=f(x)','y=exp(x)*cos(x)')
x0=0;
x1=%pi;


//      True value
I = integrate('exp(x)*cos(x)','x',x0,x1)

//      Using Gaussian Quadrature

//      For n=2, w=1

n=2;
p  = legendrepol(n,'x')
xr = roots(p);
A  = [];

for j = 1:2
    pd = derivat(p)
    A = [A 2/((1-xr(j)^2)*(horner(pd,xr(j)))^2)]
end

tr = ((x1-x0)/2.*xr)+((x1+x0)/2)
```

check Appendix AP 6 for dependency:

`legendrepol.sce`

Scilab code Exa 5.8 Gaussian Legendre Quadrature

```
            //      PG (278)

deff('[y]=f(x)','y=exp(-x^2)')
x0=0;
```

```
x1=1;


//     True value
I = integrate('exp(-x^2)','x',x0,x1)

//     Using Gaussian Quadrature

//     For n=2, w=1

n=2;
p  = legendrepol(n,'x')
xr = roots(p);
A  = [];

for j = 1:2
    pd = derivat(p)
    A = [A 2/((1-xr(j)^2)*(horner(pd,xr(j)))^2)]
end

tr = ((x1-x0)/2.*xr)+((x1+x0)/2);

s = ((x1-x0)/2)*f(tr)
I = s*A
```

Scilab code Exa 5.9 Integration

```
        //     PG (280)

I1 = integrate('sqrt(x)','x',0,1)

I2 = integrate('1/(1+(x-%pi)^2)','x',0,5)

I3 = integrate('exp(-x)*sin(50*x)','x',0,2*%pi)
```

Scilab code Exa 5.10 Simpson Integration error

```
        //    PG (292)

deff('[y]=f(x)','y=x^(3/2)')
x0=0;
xn=1;
x=x0:xn;

//    True value

I = integrate('x^(3/2)','x',0,1)

//    Using Simpson's rule

N=2;
h=(xn-x0)/N;
x1=x0+h;
x2=x0+2*h;
    I1 = h*(f(x0)+4*f(x1)+f(x2))/3
    I-I1

N=4;
h=(xn-x0)/N;
x1=x0+h;
x2=x0+2*h;
x3=x0+3*h;
x4=x0+4*h;
    I2 = h*(f(x0)+4*f(x1)+2*f(x2)+4*f(x3)+f(x4))/3
    I-I2

N=8;
h=(xn-x0)/N;
x1=x0+h;
```

```
x2=x0+2*h;
x3=x0+3*h;
x4=x0+4*h;
x5=x0+5*h;
x6=x0+6*h;
x7=x0+7*h;
x8=x0+8*h;
    I3 = h*(f(x0)+4*f(x1)+2*f(x2)+4*f(x3)+2*f(x4)+4*
       f(x5)+2*f(x6)+4*f(x7)+f(x8))/3
    I-I3
```

check Appendix AP 7 for dependency:

`romberg.sce`

Scilab code Exa 5.11 Romberg Integration

```
        //     PG (297)

deff('[y]=f(x)','y=exp(x)*cos(x)')
a=0;
b=%pi;
h=1;

//     True value

I = integrate('exp(x)*cos(x)','x',a,b)

//     Using Romberg integration

Romberg(a,b,f,h)
```

Scilab code Exa 5.12 Adaptive simpson

```
            //      PG (302)

deff('[y]=f(x)','y=sqrt(x)')
funcprot(0)
a=0;
b=1;

//      True value

I = integrate('sqrt(x)','x',a,b)
```

Scilab code Exa 5.13 Integration

```
            //      PG (307)

deff('[y]=f(x)','y=sqrt(-log(x))')
funcprot(0)
a=0;
b=1;

//      True value

I = integrate('sqrt(-log(x))','x',a,b)
```

Scilab code Exa 5.14 Integration

```
            //      PG (313)

deff('[y]=f(x)','y=(log(x))/(x+2)')
funcprot(0)
a=0;
b=1;

```

```
//     True value

I = integrate('(log(x))/(x+2)','x',a,b)
```

Chapter 6

Numerical methods for ordinary differential equations

Scilab code Exa 6.1 1st order linear differential equation

```
        //    PG (334)

//    dy/dt=-y
function ydot=f(y,t),ydot=-y,
endfunction
y0=0;t0=0;t=0:1:%pi;
y=ode(y0,t0,t,f)
plot(t,y)
```

Scilab code Exa 6.4 Stability of solution

```
        //    PG (339)

//    dy/dx=100y  -101*(%e)^(-x)
function ydox=f(x,y),ydox=100*y -101*(%e)^(-x),
   endfunction
```

```
funcprot(0)
y0=1;
x0=0;
x=0:5;
y=ode(y0,x0,x,f)

//     Solution will be Y(x) = exp(-x)

//     For the perturbed problem, dy/dx = 100*y - 101*exp(-x), y(0) = 1+e
//     Solution will be Y(x;e) = exp(-x) + e*exp(100*x)
//     This rapidly departs from the true solution.
```

check Appendix AP 2 for dependency:

euler.sce

Scilab code Exa 6.5 Euler method

```
          //     PG (344)

//     dy/dx = y

// y'=f(x,t)
deff('[z]=f(x,y)','z=y');

// execute the function euler1 , so as to call it to evaluate the value of y,


[y,x] = Euler1(0.40,1,2.00,0.2,f)      // h=0.2;

[y,x] = Euler1(0.40,1,2.00,0.1,f)      // h=0.1;

[y,x] = Euler1(0.40,1,2.00,0.05,f)      // h=0.05;
```

```

//     True solution is
Y = exp(x)


//     dy/dx = (1/(1+x^2))-(2*y^2)

// y'=f(x,t)
deff('[z]=f(x,y)','z=(1/(1+x^2))-(2*y^2)');

// execute the function euler1 , so as to call it to
   evaluate the value of y,



[y,x] = Euler1(0,0,2,0.2,f)      // h=0.2;

[y,x] = Euler1(0,0,2,0.1,f)      // h=0.1;

[y,x] = Euler1(0,0,2,0.05,f)      // h=0.05;

//     True solution is
Y = x/(1+x^2)
```

check Appendix AP 2 for dependency:

euler.sce

Scilab code Exa 6.6 Euler

```
          //     PG (351)

//     dy/dx = -y + 2 * cos(x)

deff('[y]=g(x,y)','y=-y+2*cos(x)')
y0=1;
```

```
x0=0;
xn=5;

// execute the function euler1 , so as to call it to
    evaluate the value of y,

[y,x] = Euler1(y0,x0,xn,0.04,g)    //    h = 0.04

[y,x] = Euler1(y0,x0,xn,0.02,g)    //    h = 0.02

[y,x] = Euler1(y0,x0,xn,0.01,g)    //    h = 0.01
```

Scilab code Exa 6.7 Asymptotic error analysis

```
          //    PG (354)

//    dy/dx = -y

deff('[z]=f(x,y)','z=-y')
y0=1;

//    True solution is
Y = exp(-x)
//    The equation for D(x) is
//        D'(x) = -D(x) + 0.5*exp(-x)
//        D(0) = 0
//    The solution is
//        D(x) = 0.5*x*exp(-x)
```

Scilab code Exa 6.9 Midpoint and trapezoidal method

```
          //    PG (357)

```

```
//      1. The mid-point method is defined by

//      y(n+1) = y(n-1) + 2*h*f(xn,yn)---------n>=1

//      It is an explicit two-step method.


//      The trapezoidal method is defined by

//      y(n+1) = yn + h*[f(xn,yn) + f(x(n+1),y(n+1))
    ]-------n>=0

//      It is an implicit one-step method.
```

check Appendix AP 2 for dependency:

`euler.sce`

Scilab code Exa 6.10 Euler

```
          //      PG (365)

deff('[z]=g(x,y)','z=-y')
[y,x] = Euler1(0.25,1,2.25,0.25,g)

//------------------------------------------

deff('[z]=f(x,y)','z=x-y^2')
[y,x] = Euler1(0.25,0,3.25,0.25,f)
```

check Appendix AP 5 for dependency:

`trapezoidal.sce`

Scilab code Exa 6.11 Trapezoidal method

```
          //     PG (372)

deff('[y]=f(x,y)','y=-y^2')
[x,y] = trapezoidal(1,1,5,1,f)
```

Scilab code Exa 6.16 Adams Moulton method

```
          //     PG (389)

//     Using Adams-Moulton Formula

deff('[z]=f(x,y)','z=(1/(1+x^2))-2*y^2')
y0 = 0;

//     Solution is Y(x) = x/(1+x^2)

function [y,x] = adamsmoulton4(y0,x0,xn,h,f)

//adamsmoulton4 4th order method solving ODE
//  dy/dx = f(y,x), with initial
//conditions y=y0 at x=x0.  The
//solution is obtained for x = [x0:h:xn]
//and returned in y

umaxAllowed = 1e+100;

x = [x0:h:xn]; y = zeros(x); n = length(y); y(1) =
   y0;
for j = 1:n-1
if j<3 then
      k1=h*f(x(j),y(j));
    k2=h*f(x(j)+h,y(j)+k1);
    y(j+1) = y(j) + (k2+k1)/2;
```

```
end;

if j>=2 then
        y(j+2) = y(j+1) + (h/12)*(23*f(x(j+1),y(j+1)
           )-16*f(x(j),y(j))+5*f(x(j-1),y(j-1)));
end;
end;
endfunction

adamsmoulton4(0,2.0,10.0,2.0,f)
```

check Appendix AP 2 for dependency:

`euler.sce`

Scilab code Exa 6.21 Euler method

```
        //      PG (405)

deff('[y]=f(x,y)','y=lamda*y+(1-lamda)*cos(x)-(1+
   lamda)*sin(x)')
lamda = -1;
[x,y]=Euler1(1,1,5,0.5,f)
lamda = -10;
[x,y]=Euler1(1,1,5,0.1,f)
lamda = -50;
[x,y]=Euler1(1,1,5,0.01,f)
```

check Appendix AP 5 for dependency:

`trapezoidal.sce`

Scilab code Exa 6.24 Trapezoidal method

```
            //      PG (409)

deff('[y]=f(x,y)','y=lamda*y+(1-lamda)*cos(x)-(1+
   lamda)*sin(x)')
lamda = -1;
[x,y]=trapezoidal(1,1,5,0.5,f)
lamda = -10;
[x,y]=trapezoidal(1,1,5,0.5,f)
lamda = -50;
[x,y]=trapezoidal(1,1,5,0.5,f)
```

check Appendix AP 4 for dependency:

`bvpeigen.sce`

check Appendix AP 3 for dependency:

`eigenvectors.sce`

Scilab code Exa 6.31 Boundary value problem

```
            //      PG (434)

//     2-point linear Boundary value problem


//     Boundary value problems with eigenvalues -
   case: d^y/dx^2 + lam*y = 0
//     subject to y(0) = 0, y(1) = 0, where lam is
   unknown.
//     The finite-difference approximation is:
//     (y(i-1)-2*y(i)+y(i+1))=-lam*Dx^2*y(i), i =
   2,3,...,n-1


[x,y,lam] = BVPeigen1(1,5)
```

Chapter 7

Linear Algebra

Scilab code Exa 7.1 Orthonomal basis

```
        //    PG (470)

u1 = [1/2,sqrt(3)/2]
u2 = [-sqrt(3)/2,1/2]

//    For a given vector x = (x1,x2), it can be
   written as
//                     x = alpha1*u1 + alpha2*u2
//    alpha1 = (x1+x2*sqrt(3))/2
//    alpha2 = (x2-x1*sqrt(3))/2

//    (1,0) = (1/2)*u1 - (sqrt(3)/2)*u2
```

Scilab code Exa 7.2 Canonical forms

```
        //    PG (476)

A = [0.2 0.6 0;1.6 -0.2 0;-1.6 1.2 3.0]
```

```
U = [0.6 0 -0.8;0.8 0 0.6;0 1.0 0]
Ustar = inv(U)
T = Ustar*A*U
trace(A)
lam =spec(A)'
lam1 = lam(1,1)
lam2 = lam(1,2)
lam3 = lam(1,3)
lam1 + lam2 + lam3

    //    trace(A) = lam1 + lam2 + lam3

det(A)
lam1*lam2*lam3

    //    det(A) = lam1 * lam2 * lam3
```

Scilab code Exa 7.3 Orthonomal eigen vectors

```
        //    PG (477)

A = [2 1 0;1 3 1;0 1 2]
lam = spec(A)'
lam1 = lam(1,1)
lam2 = lam(1,2)
lam3 = lam(1,3)
//    Orthonomal Eigen vectors

u1 = (1/sqrt(3))*[1;-1;1]
u2 = (1/sqrt(2))*[1;0;-1]
u3 = (1/sqrt(6))*[1;2;1]
```

Scilab code Exa 7.4 Vector and matrix norms

```
        //    PG (481)

x = [1,0,-1,2]
    //    1-norm
norm(x,1)
    //    2-norm
norm(x,2)
    //    infinity norm
norm(x,'inf')
```

Scilab code Exa 7.5 Frobenious norm

```
        //    PG (484)

//    A be n * n
//    norm(A*x,2)
//    norm(A*x,2) <= norm(A,'fro') * norm(x,2)
//    norm(A*B,'fro') = norm(A,'fro') * norm(B,'fro
   ')
```

Scilab code Exa 7.6 Norm

```
        //    PG (489)

A = [1 -2;-3 4]
norm(A,1)
norm(A,2)
norm(A,'inf')
lam = spec(A)
r = max(abs(lam))
    //r < = norm(A,2)
```

Scilab code Exa 7.7 Inverse exists

```
        //    PG (494)

A = [4 1 0 0;1 4 1 0;0 1 4 1;0 0 1 4]
B = A/4 - eye()
norm(B,'inf')
    //    Let (I+B = C)
C = eye() + B
inv(C)
//    Inverse of (I + B) exists
norm(C,'inf')
//    Inverse of A exists.
```

Chapter 8

Numerical solution of systems of linear equations

Scilab code Exa 8.2 LU decomposition

```
    //    EXAMPLE (PG 512)

A = [1 2 1;2 2 3;-1 -3 0]                //    Coefficient matrix
b = [0 3 2]'                             //    Right hand matrix
[l,u] = lu(A)
    //    l is lower triangular matrix & u is upper triangular matrix
l*u
if(A==l*u)
    disp('A = LU is verified')
end
det(A)
det(u)
if(det(A)==det(u))
    disp('Determinant of A is equal to that of its upper triangular matrix')

```

```
//     Product rule of determinants is verified
```

Scilab code Exa 8.4 LU decomposition

```
//     EXAMPLE (PG 518)

//     Row interchanges on A can be represented by premultiplication of A
//     by an appropriate matrix E, to get EA.
//     Then, Gaussian Elimination leads to LU = PA

A = [0.729 0.81 0.9;1 1 1;1.331 1.21 1.1]      // Coefficient Matrix
b = [0.6867 0.8338 1.000]'                     // Right Hand Matrix
[L,U,E] = lu(A)
//     L is lower triangular matrix(mxn)
//     U is upper triangular matrix(mxmin(m,n))
//     E is permutation matrix(min(m,n)xn)
Z=L*U

disp("LU = EA")
E

//     The result EA is the matrix A with first, rows 1 & 3 interchanged,
//     and then rows 2 & 3 interchanged.

//     NOTE:-According to the book, P is replaced by E here.
```

Scilab code Exa 8.5 Choleski Decomposition

```
    //    EXAMPLE (PG 526)

disp("Consider Hilbert matrix of order three")

n=3;         //    Order of the matrix
A=zeros(n,n);//    a symmetric positive definite real or complex matrix.
for i=1:n    //    Initializing 'for' loop
    for j=1:n
        A(i,j)=1/(i+j-1);
    end
end         //End of 'for' loop
A
chol(A)                      //    Choleski Decomposition
L=[chol(A)]'                 //    Lower Triangular Matrix

    //    The square roots obtained here can be avoided using a slight modification.
    //    We find a diagonal matrix D & a lower triangular matrix (L^~),
    //    with 1s on the diagonal such that A = (L^~) * D * (L^~)'


    //    chol(A) uses only the diagonal and upper triangle of A.
    //    The lower triangular is assumed to be the
    //    (complex conjugate) transpose of the upper.
```

Scilab code Exa 8.6 LU decomposition

```
    //    EXAMPLE (PG 529)
```

```
//    Consider the coefficient matrix for spline
      interpolation


A = [2 1 0 0;1 4 1 0;0 1 4 1;0 0 1 2]
[l,u] = lu(A);    //    LU Decomposition
U = l'            //    Lower Triangular matrix
L = u'            //    Upper triangular matrix
```

Scilab code Exa 8.7 Error analysis

```
//    EXAMPLE (PG 531)

//    Consider the linear system

//    7*x1 + 10*x2 = b1
//    5*x1 + 7*x2 = b2

A = [7 10;5 7]              //    Coefficient matrix
inv(A)                      //    Inverse matrix

//    cond(A)1              //    Condition matrix

norm(A,1)*norm(inv(A),1)

//    cond(A)2              //    Condition matrix

norm(A,2)*norm(inv(A),2)

//    These condition numbers all suggest that
     the above system
//    may be sensitive to changes in the right
     side b.

```

```
    //    Consider the particular case

b = [1 0.7]';          //    Right hand matrix
x = A\b;              //    Solution matrix

    //    Solution matrix

x1 = x(1,:)
x2 = x(2,:)

    //    For the perturbed system, we solve for:

b = [1.01 0.69]';          //    Right hand matrix
x = A\b;                  //    Solution matrix

    //    Solution matrix

x1 = x(1,:)
x2 = x(2,:)

    //    The relative changes in x are quite large when compared with
    //    the size of the relative changes in the right side b.
```

Scilab code Exa 8.8 Residual correction method

```
    //    EXAMPLE (PG 541)

    //    Consider a Hilbert matrix of order 3

n=3;               //    Order of the matrix
A=zeros(n,n);      //    a symmetric positive definite real or complex matrix.
for i=1:n          //    Initializing 'for' loop
```

```
    for j=1:n
        A(i,j)=1/(i+j-1);
    end
end          //    End of 'for' loop
A

    //    Rounding off to 4 decimal places
A = A*10^4;
A = int(A);
A = A*10^(-4);
disp(A)          //    Final Solution

H = A          //    Here H denoted H bar as denoted
   in the text

b = [1 0 0]'
x = H\b

    //    Rounding off to 3 decimal places
x = x*10^3;
x = int(x);
x = x*10^(-3);
disp(x)          //    Final Solution

//Now, using elimination with Partial Pivoting, we
   get the following answers

x0 = [8.968 -35.77 29.77]'

    //    ro is Residual correction

r0 = b - A*x0

    //    A*e0 = r0

e0 = inv(A)*r0

x1 = x0 + e0
```

```
    //          Repeating the above operations , we can
        get the values of r1 , x2 , e1 ...
    //          The vector x2 is accurate to 4 decimal
        digits .
    //          Note that x1 − x0 = e0 is an accurate
        predictor of the error e0 in x0 .
```

Scilab code Exa 8.9 Residual correction method

```
//EXAMPLE (PG 544)

//A(e) = A0 + eB

A0=[2 1 0;1 2 1;0 1 2]
B=[0 1 1;-1 0 1;-1 -1 0]
//inv(A(e)) = C = inv(A0)
C=inv(A0)
b=[0 1 2]'
x=A0\b
r=b-A0*x
```

Scilab code Exa 8.10 Gauss Jacobi method

```
    //     EXAMPLE (PG 547)

    //     Gauss Jacobi Method

A = [10 3 1;2 -10 3;1 3 10]          //
    Coefficient Matrix
b = [14 -5 14]'                      //          Right
    hand matrix

```

```
x = [0 0 0]'                          //       Initial
    Gauss
d = diag(A)                           //
   Diagonal elements of matrix A
a11 = d(1,1)
a22 = d(2,1)
a33 = d(3,1)
D = [a11 0 0;0 a22 0;0 0 a33]         //
   Diagonal matrix of A
[L,U] = lu(A)  //    L is lower triangular matrix, U
    is upper triangular matrix
H = -inv(D)*(L+U)
C = inv(D)*b

for(m=0:6)    //    Initialising 'for' loop for
   setting no of iterations to 6
    x = H*x+C;
    disp(x)
    m=m+1;
    x;             //    Solution
    //    Rounding off to 4 decimal places
    x = x*10^4;
    x = int(x);
    x = x*10^(-4);
    disp(x)         //    Final Solution

end
```

check Appendix AP 1 for dependency:

`gaussseidel.sce`

Scilab code Exa 8.11 Gauss seidel mathod

```
    //EXAMPLE (PG 549)

```

```
    //Gauss Seidel Method

exec gaussseidel.sce
A = [10 3 1;2 -10 3;1 3 10]     //    Coefficient
   matrix
b = [14 -5 14]'                 //    Right hand
   matrix
x0 = [0 0 0]'                   //    Initial Gauss
gaussseidel(A,b,x0)             //    Calling
   function

        //        End the problem
```

Scilab code Exa 8.13 Conjugate gradient method

```
    //    EXAMPLE (PG 568)

A= [5 4 3 2 1;4 5 4 3 2;3 4 5 4 3;2 3 4 5 4;1 2 3 4
   5]    //    Matrix of order 5
    //    Getting the eigenvalues

lam = spec(A)                 //    lamda = spectral
   radius of matrix A

max(lam)                      //    Largest eigenvalue
min(lam)                      //    Smallest eigen
   value

    //        For the error bound given earlier on
       Pg 567

c = min(lam)/max(lam)

(1-sqrt(c))/(1+sqrt(c))

```

```
    //    For linear system, choose the following
       values of b

b = [7.9380 12.9763 17.3057 19.4332 18.4196]'

x = A\b;    //    Solution matrix

    //    Rounding off to 4 decimal places
x = x*10^4;
x = int(x);
x = x*10^(-4)
disp(x)          //    Final Solution
```

Chapter 9

The Matrix Eigenvalue Problem

Scilab code Exa 9.1 Eigenvalues

```
        //    EXAMPLE 590

A = [4 1 0;1 0 -1;1 1 -4]
[n,m] = size(A);

if m<>n then
    error('eigenvectors - matrix A is not square');
    abort;
end;

lam = spec(A)'                    //Eigenvalues of matrix A
```

Scilab code Exa 9.2 Eigen values and matrix norm

```
        // PG 591
```

```

n = 4
A = [4 1 0 0;1 4 1 0;0 1 4 1;0 0 1 4]
lam = spec(A)

//     Since A is symmetric, all eigen values are
   real.
//     The radii are all 1 or 2.
//     The centers of all the circles are 4.
//     All eigen values must all lie in the interval
   [2,6]
//     Since the eigen values of inv(A) are the
   reciprocals of those of A,
//     1/6 <= mu <= 1/2

//     Let inv(A) = B

B=inv(A);
norm(B,2)
n
i = 1:n;
j = 1:n;

    //     for j~i
    //         r = sum(abs(B(i,j)))

//     norm(B,2) = r(B) <= o.5
```

Scilab code Exa 9.3 Bounds for perturbed eigen values

```
        //     PG 593

disp("Consider Hilbert matrix of order three")

n=3;          //     Order of the matrix
```

```
A=zeros(n,n);//    a symmetric positive definite
   real or complex matrix.
for i=1:n    //    Initializing 'for' loop
    for j=1:n
        A(i,j)=1/(i+j-1);
    end
end         //End of 'for' loop
A

[n,m] = size(A)

if m<>n then
    error('eigenvectors - matrix A is not square');
    abort;
end;

lam = spec(A)'                   //Eigenvalues of
   matrix A

lam1 = lam(1,1)
lam2 = lam(1,2)
lam3 = lam(1,3)

    //    Rounding off to 4 decimal places

A = A*10^4;
    A = int(A);
    A = A*10^(-4);
    disp(A)        //    Final Solution

lamr = spec(A)'

lamr1 = lamr(1,1)
lamr2 = lamr(1,2)
lamr3 = lamr(1,3)

    //    Errors

```

```
lam - lamr

    //    Relative Errors

R1 = (lam1-lamr1)/lam1
R2 = (lam2-lamr2)/lam2
R3 = (lam3-lamr3)/lam3
```

Scilab code Exa 9.4 Eigenvalues of nonsymmetric matrix

```
        //    PG 594

A = [101 -90;110 -98]
[n,m] = size(A)

if m<>n then
    error('eigenvectors - matrix A is not square');
    abort;
end;

lam = spec(A)'                  //Eigenvalues of
   matrix A


    //    A+E = [101-e -90-e;110 -98]
    //    Let e = 0.001
e = 0.001;
    //    Let A+E = D
D = [101-e -90-e;110 -98]

[n,m] = size(D)

if m<>n then
    error('eigenvectors - matrix D is not square');
    abort;
```

```
end;
lam = spec(D)'                              //Eigenvalues of matrix A
```

Scilab code Exa 9.5 Stability of eigenvalues for nonsymmetric matrices

```
        //    PG 599

    //    e = 0.001
    //    From earlier example :
    //    eigen values of matrix A are 1 and 2. So ,..

    //    inv(P)*A*P = [1 0;0 2]

A = [101 -90;110 -98]
B = [-1 -1;0 0]
    //    From the above equation , we get:

P = [9/sqrt(181) -10/sqrt(221);10/sqrt(181) -11/sqrt(221)]
inv(P)
K = norm(P)*norm(inv(P))          //    K is condition number
u1 = P(:,1)
u2 = P(:,2)
Q = inv(P)
R = Q'
w1 = R(:,1)
w2 = R(:,2)
s1 = 1/norm(w1,2)
norm(B)

//    abs(lam1(e) - lam1) <= sqrt(2)*e/0.005 + O(e^2) = 283*e + O(e^2)
```

Scilab code Exa 9.7 Rate of convergence

```
        //    (PG 607)

A = [1 2 3;2 3 4;3 4 5]
lam = spec(A)'
lam1 = lam(1,3)
lam2 = lam(1,1)
lam3 = lam(1,2)

    //    Theoretical ratio of convergence

lam2/lam1

b = 0.5*(lam2+lam3)
B = A-b*eye(3,3)

    //    Eigen values of A-bI are:

lamb = spec(B)'
lamb1 = lamb(1,3)
lamb2 = lamb(1,2)
lamb3 = lamb(1,1)

    //    Ratio of convergence for the power method
       applied to A-bI will be:

lamb2/lamb1

    //    This is less than half the magnitude of
       the original ratio.
```

Scilab code Exa 9.8 Rate of convergence after extrapolation

```
        //    PG (608)

A = [1 2 3;2 3 4;3 4 5]
lam = spec(A)'         //    Eigen values of A
lam1 = lam(1,3)
lam2 = lam(1,1)
lam3 = lam(1,2)

//    Theoretical ratio of convergence

lam2/lam1

//    After extrapolating, we get
        lame1 = 9.6234814

//    Error:
lam1-lame1
```

Scilab code Exa 9.9 Householder matrix

```
        //    PG (610)

w = [1/3 2/3 2/3]'
w1 = w(1,1)
w2 = w(2,1)
w3 = w(3,1)

U = [1-2*abs(w1)^2 -2*w1*w2' -2*w1*w3';-2*w1'*w2
   1-2*abs(w2)^2 -2*w2*w3';-2*w1'*w3 -2*w2'*w3 1-2*
   abs(w3)^2]
U
inv(U)
//    U = inv(U)------Hence, U is Hermitian
```

```
U*U
//     U*U = I-----------Hence, U is orthogonal
```

Scilab code Exa 9.11 QR factorisation

```
          //     PG (613)

A = [4 1 1;1 4 1;1 1 4]
w1 = [0.985599 0.119573 0.119573]'
P1 = eye() - 2*w1*w1'
A2 = P1*A
w2 = [0 0.996393 0.0848572]'
P2 = eye() - 2*w2*w2'
R = P2*A2
Q = P1*P2
Q*R

//     A = Q * R

abs(det(A))
abs(det(Q)*det(R))

//     |det(A)| = |det(Q)*det(R)| = |det(R)| = 54 (
   approx)

lam = spec(A)'
lam1 = lam(1,1)
lam2 = lam(1,2)
lam3 = lam(1,3)
lam1 * lam2 * lam3

//     Product of eigen values also comes out to be
   54
```

Scilab code Exa 9.12 Tridiagonal Matrix

```
        //     PG (617)

A = [1 3 4;3 1 2;4 2 1]
w2 = [0 2/sqrt(5) 1/sqrt(5)]'
P1 = eye() - 2*w2*w2'
T = P1' * A * P1    //     Tridiagonal matrix
```

Scilab code Exa 9.13 Planner Rotation Orthogonal Matrix

```
        //     PG (619)

x = %pi/4
R = [cos(x) 0 sin(x);0 1 0;-sin(x) 0 cos(x)]

//     Planner Rotation Orthogonal Matrix
```

Scilab code Exa 9.14 Eigen values of a symmetric tridiagonal Matrix

```
        //     PG (620)

T = [2 1 0 0 0 0;1 2 1 0 0 0;0 1 2 1 0 0;0 0 1 2 1
   0;0 0 0 1 2 1;0 0 0 0 1 2]
lam = spec(T)'
lam1 = lam(1,1)
B = [2-lam1 1 0 0 0 0;1 2-lam1 1 0 0 0;0 1 2-lam1 1
   0 0;0 0 1 2-lam1 1 0;0 0 0 1 2-lam1 1;0 0 0 0 1
   2]
```

```
f0 = abs(det(B))
f1 = 2-lam1
```

Scilab code Exa 9.15 Sturm Sequence property

```
        //   PG (621)

//   For the previous example, consider the
   sequence f0, f1....f6

//   For lam = 3,

//       (f0,.....f6) = (1,−1,0,1,−1,0,1)

//   The corresponding sequence of signs is

//       (+,−,+,+,−,+,+)

//   and s(3) = 2
```

Scilab code Exa 9.16 QR Method

```
        //   PG (624)

A1 = [2 1 0;1 3 1;0 1 4]
lam = spec(A1)'
[Q1,R1] = qr(A1);
A2 = R1 * Q1
[Q2,R2] = qr(A2);
A3 = R2 * Q2
[Q3,R3] = qr(A3);
A4 = R3 * Q3
[Q4,R4] = qr(A4);
```

```
A5 = R4 * Q4
[Q5,R5] = qr(A5);
A6 = R5 * Q5
[Q6,R6] = qr(A6);
A7 = R6 * Q6
[Q7,R7] = qr(A7);
A8 = R7 * Q7
[Q8,R8] = qr(A8);
A9 = R8 * Q8
[Q9,R9] = qr(A9);
A10 = R9 * Q9
[Q10,R10] = qr(A10);
```

Scilab code Exa 9.18 Calculation of Eigen vectors and Inverse iteration

```
        //    PG (631)

A = [2 1 0;1 3 1;0 1 4]
lam = spec(A)
[L,U] = lu(A)
y1 = [1 1 1]'
w1 = [3385.2 -2477.3 908.20]'
z1 = [w1/norm(w1,'inf')]'
w2 = [20345 -14894 5451.9]'
z2 = [w2/norm(w2,'inf')]'
z3 = z2

//    The true answer is

x3 = [1 1-sqrt(3) 2-sqrt(3)]'

//    z2 equals x3 to within the limits of rounding
   error accumulations.
```

Scilab code Exa 9.19 Inverse Iteration

```
        //      PG (633)

A = [2 1 0;1 3 1;0 1 4]
lam = spec(A)
[L,U] = lu(A)
y1 = [1 1 1]'
w1 = [3385.2 -2477.3 908.20]'
z1 = [w1/norm(w1,'inf')]'
w2 = [20345 -14894 5451.9]'
z2 = [w2/norm(w2,'inf')]'
z3 = z2

//      The true answer is

x3 = [1 1-sqrt(3) 2-sqrt(3)]'

//      z2 equals x3 to within the limits of rounding
   error accumulations.

//      Consider lam = 1.2679

//      0.7321*x1 + x2 = 0
//      x1 + 1.7321*x2 + x3 = 0
//      Taking x1= 1.0, we have the approximate
   eigenvector

//                      x = [1.0000 -0.73210 0.26807]


//      Compared with the true answer obtained above,
   this is a slightly poorer
//      result obtained by inverse iteration.
```

Appendix

Scilab code AP 1 Gauss seidel method

```
function [x]=gaussseidel(A,b,x0)
[nA,mA]=size(A)
n=nA
[L,U] = lu(A)
d = diag(A)
a11 = d(1,1)
a22 = d(2,1)
a33 = d(3,1)
D = [a11 0 0;0 a22 0;0 0 a33]
H = -inv(L+D)*U
C = inv(L+D)*b
for m=0:3
            x = -inv(D)*(L+U)*x + inv(D)*b
            m=m+1
            disp(x)
end

endfunction
```

Scilab code AP 2 Euler method

```
function [x,y] = Euler1(x0,y0,xn,h,g)

//Euler 1st order method solving ODE
//  dy/dx = g(x,y), with initial
//conditions y=y0 at x = x0.  The
```

```
//solution is obtained for x = [x0:h:xn]
//and returned in y

ymaxAllowed = 1e+100

x = [x0:h:xn];
y = zeros(x);
n = length(y);
y(1) = y0;

for j = 1:n-1
    y(j+1) = y(j) + h*g(x(j),y(j));
    if y(j+1) > ymaxAllowed then
            disp('Euler 1 - WARNING: underflow or overflow');
        disp('Solution sought in the following range:');
            disp([x0 h xn]);
        disp('Solution evaluated in the following range:');
        disp([x0 h x(j)]);
            n = j;
            x = x(1,1:n); y = y(1,1:n);
        break;
    end;
end;

endfunction

//End function Euler1
```

Scilab code AP 3 Eigen vectors

```
function [x,lam] = eigenvectors(A)

//Calculates unit eigenvectors of matrix A
//returning a matrix x whose columns are
//the eigenvectors. The function also
```

```
//returns the eigenvalues of the matrix.

[n,m] = size(A);

if m<>n then
    error('eigenvectors - matrix A is not square');
    abort;
end;

lam = spec(A)';                   //Eigenvalues of matrix A

x = [];

for k = 1:n
    B = A - lam(k)*eye(n,n); //Characteristic matrix
        C = B(1:n-1,1:n-1);  //Coeff. matrix for reduced system
    b = -B(1:n-1,n);         //RHS vector for reduced system
    y = C\b;          //Solution for reduced system
    y = [y;1];        //Complete eigenvector
    y = y/norm(y);        //Make unit eigenvector
    x = [x y];        //Add eigenvector to matrix
end;

endfunction
//End of function
```

Scilab code AP 4 Boundary value problem

```
function [x,y,lam] = BVPeigen1(L,n)

Dx = L/(n-1);
x=[0:Dx:L];
a = 1/Dx^2;
k  = n-2;

```

```
A = zeros(k,k);
for j = 1:k
    A(j,j) = 2*a;
end;
for j = 1:k-1
    A(j,j+1) = -a;
    A(j+1,j) = -a;
end;

exec eigenvectors.sce

[yy,lam]=eigenvectors(A);
//disp('yy');disp(yy);

y = [zeros(1,k);yy;zeros(1,k)];
//disp('y');disp(y);


xmin=min(x);xmax=max(x);ymin=min(y);ymax=max(y);
rect = [xmin ymin xmax ymax];

if k>=5 then
   m = 5;
else
   m = k;
end


endfunction
```

Scilab code AP 5 Trapezoidal method

```
function [x,y] = trapezoidal(x0,y0,xn,h,g)

//Trapezoidal method solving ODE
//  dy/dx = g(x,y), with initial
//conditions y=y0 at x = x0.  The
//solution is obtained for x = [x0:h:xn]
```

```
//and returned in y

ymaxAllowed = 1e+100

x = [x0:h:xn];
y = zeros(x);
n = length(y);
y(1) = y0;

for j = 1:n-1
    y(j+1) = y(j) + h*(g(x(j),y(j))+g(x(j+1),y(j+1)))/2;
    if y(j+1) > ymaxAllowed then
           disp('Euler 1 - WARNING: underflow or overflow');
        disp('Solution sought in the following range:');
           disp([x0 h xn]);
        disp('Solution evaluated in the following range:');
        disp([x0 h x(j)]);
           n = j;
           x = x(1,1:n); y = y(1,1:n);
        break;
    end;
end;

endfunction

//End function trapezoidal
```

Scilab code AP 6 Legendre Polynomial

```

function [pL] = legendrepol(n,var)

//     Generates the Legendre polynomial
//     of order n in variable var
```

```

if n == 0 then
    cc = [1];
elseif n == 1 then
    cc = [0 1];
else
    if modulo(n,2) == 0 then
        M = n/2
    else
        M = (n-1)/2
    end;

    cc = zeros(1,M+1);
    for m = 0:M
        k = n-2*m;
        cc(k+1)=...
        (-1)^m*gamma(2*n-2*m+1)/(2^n*gamma(m+1)*
           gamma(n-m+1)*gamma(n-2*m+1));
    end;
end;

pL = poly(cc,var,'coeff');

//    End function legendrepol
```

Scilab code AP 7 Romberg Integration

```
function [I]=Romberg(a,b,f,h)

//    This function calculates the numerical
   integral of f(x) between
//    x = a and x = b, with intervals h.
   Intermediate results are obtained
//    by using SCILAB's own inttrap function

x=(a:h:b)
x1=x(1,1)
x2=x(1,2)
```

```
x3=x(1,3)
x4=x(1,4)
y1=f(x1)
y2=f(x2)
y3=f(x3)
y4=f(x4)
y=[y1 y2 y3 y4]
I1 = inttrap(x,y)
x=(a:h/2:b)
x1=x(1,1)
x2=x(1,2)
x3=x(1,3)
x4=x(1,4)
x5=x(1,5)
x6=x(1,6)
x7=x(1,7)
y1=f(x1)
y2=f(x2)
y3=f(x3)
y4=f(x4)
y5=f(x5)
y6=f(x6)
y7=f(x7)
y=[y1 y2 y3 y4 y5 y6 y7]
I2 = inttrap(x,y)
I = I2 + (1.0/3.0)*(I2-I1)

endfunction
//end function Romberg
```

Scilab code AP 8 Lagrange

```
function[P]=lagrange(X,Y)

    //    X nodes,Y values
    //    P is the numerical Lagrange polynomial
        interpolation
n=length(X)
```

```
    //    n is the number of nodes. (n−1) is the
       degree
x=poly(0,"x")
P=0
for i=1:n, L=1
    for j=[1:i-1,i+1:n] L=L*(x-X(j))/(X(i)-X(j))
        end
P=P+L*Y(i)
end
endfunction
```

Scilab code AP 9 Muller method

```
function x=muller(x0,x1,x2,f)
    R=3;
    PE=10^-8;
    maxval=10^4;
     for n=1:1:R

    La=(x2-x1)/(x1-x0);
    Da=1+La;
    ga=La^2*f(x0)-Da^2*f(x1)+(La+Da)*f(x2);
    Ca=La*(La*f(x0)-Da*f(x1)+f(x2));

     q=ga^2-4*Da*Ca*f(x2);
     if q<0 then q=0;
     end
     p= sqrt(q);
     if ga<0 then p=-p;
     end
        La=-2*Da*f(x2)/(ga+p);
        x=x2+(x2-x1)*La;
        if abs(f(x))<=PE then break
        end
        if (abs(f(x))>maxval) then error('Solution
           diverges');
            abort;
            break
```

```
        else
        x0=x1;
        x1=x2;
         x2=x;
        end
    end
    disp(n," no. of iterations =")
endfunction
```

Scilab code AP 10 Secant method

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

Scilab code AP 11 Newton

```
function x=newton(x,f,fp)
    R=100;
    PE=10^-8;
    maxval=10^4;

    for n=1:1:R
        x=x-f(x)/fp(x);
        if abs(f(x))<=PE then break
        end
```

```
        if (abs(f(x))>maxval) then error('Solution diverges');
            abort
            break
        end
    end
    disp(n," no. of iterations =")
endfunction
```

Scilab code AP 12 Aitken1

```
// this program is exclusively coded to perform one iteration of aitken method,

function x0aa=aitken(x0,x1,x2,g)
x0a=x0-(x1-x0)^2/(x2-2*x1+x0);
x1a=g(x0a);
x2a=g(x1a);
x0aa=x0a-(x1a-x0a)^2/(x2a-2*x1a+x0a);

endfunction
```

Scilab code AP 13 Bisection method

```
function x=bisection(a,b,f)
    N=100;                                              // define max. number of iterations
    PE=10^-4                                            // define tolerance
    if (f(a)*f(b) > 0) then
        error('no root possible f(a)*f(b) > 0')   // checking if the decided range is containing a root
        abort;
    end;
    if(abs(f(a)) <PE) then
        error('solution at a')                          // seeing if there is an approximate root
```

```
at a,
        abort;
    end;
    if(abs(f(b)) < PE) then                    //seeing if there is an approximate root at b,
    error('solution at b')
    abort;
    end;
    x=(a+b)/2
    for n=1:1:N                                //initialising 'for' loop,
        p=f(a)*f(x)
        if p<0 then b=x ,x=(a+x)/2;            //checking for the required conditions( f(x)*f(a)<0),
        else
             a=x
            x=(x+b)/2;
        end
        if abs(f(x))<=PE then break            // instruction to come out of the loop after the required condition is achived,
        end
    end
    disp(n," no. of iterations =")             // display the no. of iterations took to achive required condition,
endfunction
```