Scilab Textbook Companion for Numerical Methods For Scientific And Engineering Computation by M. K. Jain, S. R. K. Iyengar And R. K. Jain[1]

Created by
M Vamsi Krishan
B.Tech (pursuing)
Electronics Engineering
Visvesvaraya National Institute of Technology
College Teacher
Mr. Nagarajun
Cross-Checked by
Santosh Kumar, IITB

July 31, 2019

[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: Numerical Methods For Scientific And Engineering Computation

Author: M. K. Jain, S. R. K. Iyengar And R. K. Jain

Publisher: New Age International (P) Limited

Edition: 5

Year: 2007

ISBN: 8122420012

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 2

TRANSCENDENTAL AND POLINOMIAL EQUATIONS

Scilab code Exa 2.1 intervals containing the roots of the equation

```
// The equation 8*x^3-12*x^2-2*x+3==0 has three real roots.
// the graph of this function can be observed here.
xset('window',0);
x=-1:.01:2.5;                                        // defining the range of x.
deff('[y]=f(x)','y=8*x^3-12*x^2-2*x+3');     //defining the cunction
y=feval(x,f);

```

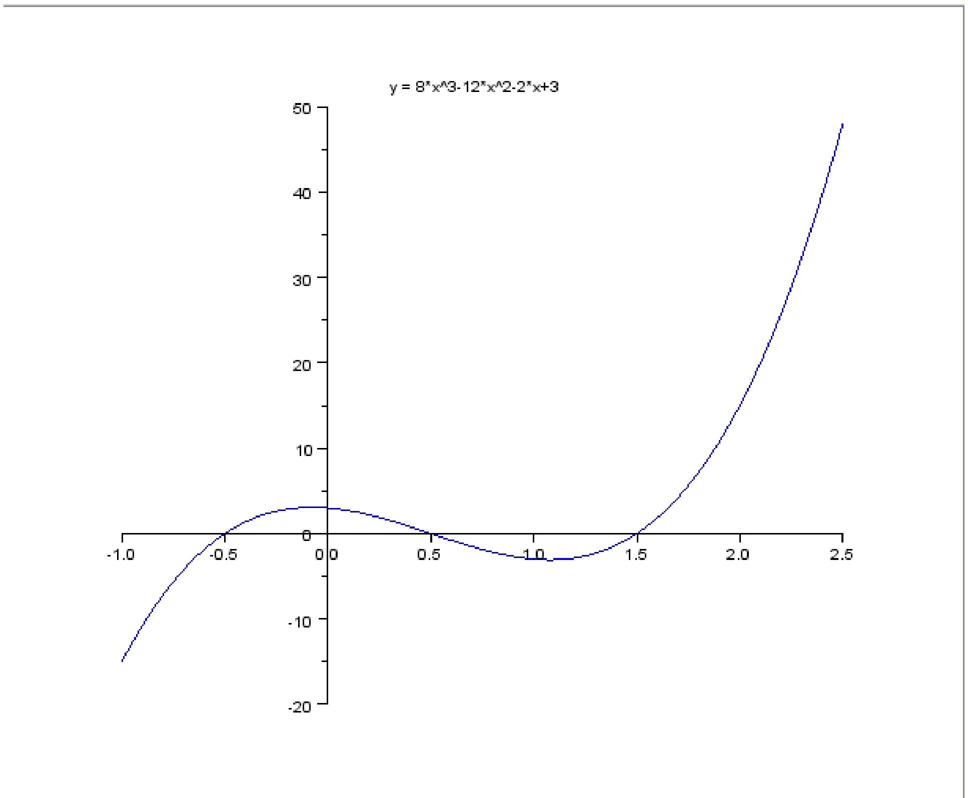

Figure 2.1: intervals containing the roots of the equation

```
a=gca();

a.y_location = "origin";

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

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

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

Scilab code Exa 2.2 `interval containing the roots`

```
// The equation cos(x)-x*%e^x==0 has  real roots.
// the graph of this function can be observed here.
xset('window',1);
x=0:.01:2;
    // defining the range of x.
deff('[y]=f(x)','y=cos(x)-x*%e^x');
    //defining the cunction.
y=feval(x,f);

```

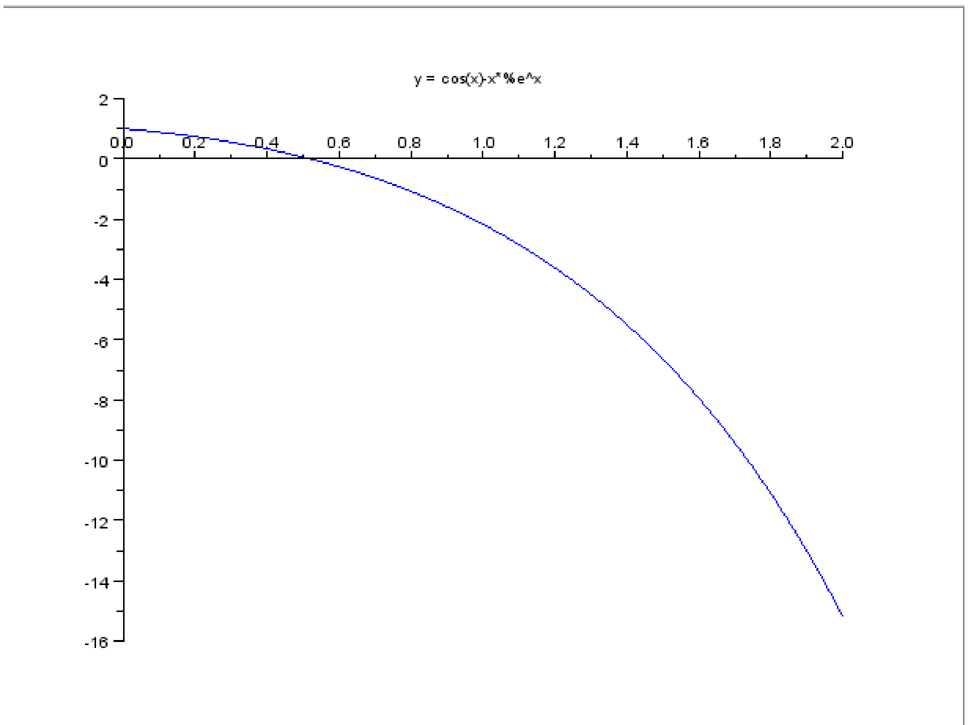

Figure 2.2: interval containing the roots

```
a=gca();

a.y_location = "origin";

a.x_location = "origin";
plot(x,y)
    // instruction to plot the graph
title(' y = cos(x)-x*%e^x')

// from the above plot we can infre that the
    function has root between
// the interval (0,1)
```

check Appendix AP 49 for dependency:

`Vbisection.sce`

check Appendix AP 48 for dependency:

`Vbisection5.sce`

Scilab code Exa 2.3 solution to the eq by bisection method

```
                                    // The equation x
                                       ^3-5*x+1==0
                                       has real
                                       roots.
                                    // the graph of
                                       this function
                                       can be
                                       observed here.
xset('window',2);
x=-2:.01:4;
                                                //
    defining the range of x.
```

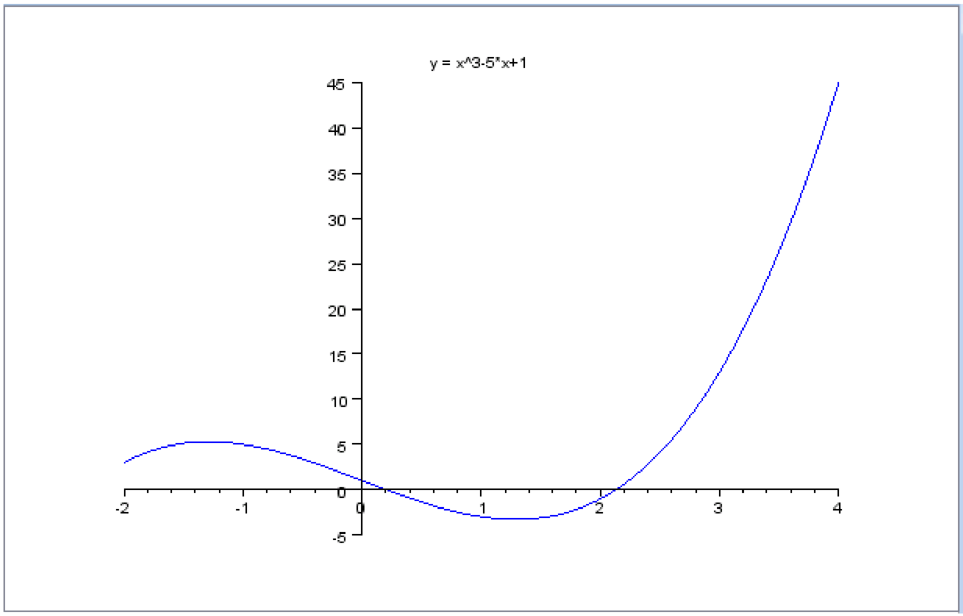

Figure 2.3: solution to the eq by bisection method

```
deff('[y]=f(x)','y=x^3-5*x+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^3-5*x+1')

// from the above plot we can infre that the function has roots between
// the intervals (0,1),(2,3).
// since we have been asked for the smallest
```

```
positive root of the equation ,
// we are intrested on the interval (0,1)
// a=0;b=1,

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

bisection(0,1,f)

// since in the example 2.3 we have been asked to perform 5 itterations
// the approximate  root after 5 iterations can be observed.



bisection5(0,1,f)


// hence the approximate root after 5 iterations is 0.203125 witin the permissible error of 10^-4,
```

check Appendix AP 49 for dependency:

`Vbisection.sce`

check Appendix AP 48 for dependency:

`Vbisection5.sce`

Scilab code Exa 2.4 `solution to the eq by bisection method`

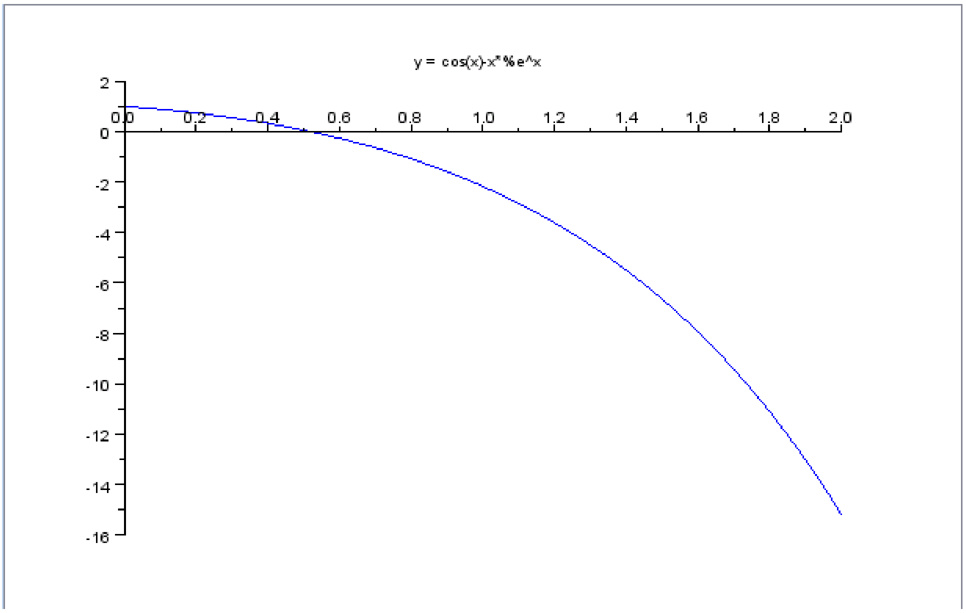

Figure 2.4: solution to the eq by bisection method

```
// The equation cos(x)-x*%e^x ==0 has  real roots.
// the graph of this function can be observed here.
xset('window',3);
x=0:.01:2;                                    // defining the range of x.
deff('[y]=f(x)','y=cos(x)-x*%e^x');           //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 = cos(x)-x*%e^x')

// from the above plot we can infre that the function has root between
// the interval (0,1)


// a=0;b=1,

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

bisection(0,1,f)

// since in the example 2.4 we have been asked to perform 5 itterations ,

bisection5(0,1,f)


// hence the approximate root after 5 iterations is 0.515625 witin the permissible error of 10^-4,
```

check Appendix AP 46 for dependency:

`regulafalsi4.sce`

check Appendix AP 47 for dependency:

`secant4.sce`

Scilab code Exa 2.5 `solution to the given equation`

```
// The equation x^3-5*x+1==0 has real roots.
// the graph of this function can be observed here.
xset('window',4);
x=-2:.01:4; // defining the range of x.
deff('[y]=f(x)','y=x^3-5*x+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^3-5*x+1')

// from the above plot we can infre that the function has roots between
// the intervals (0,1),(2,3).
// since we have been given the interval  to be considered as (0,1)
// a=0;b=1,


// Solution by secant method
```

```
// since in the example 2.5 we have been asked to perform 4 itterations ,
secant4(0,1,f)            // we call a user−defined function 'bisection' so as to find the approximate
// root of the equation with a defined permissible error.



// hence the approximate root occured in secant method after 4 iterations is 0.201640 witin the permissible error of 10^−4,



                                  // solution by regular falsi  method


// since in the example 2.5 we have been asked to perform 4 itterations ,

regulafalsi4(0,1,f)            // we call a user−defined function 'regularfalsi4' so as to find the approximate
// root of the equation with a defined permissible error.



// hence the approximate root occured in regularfalsi method after 4 iterations is
```

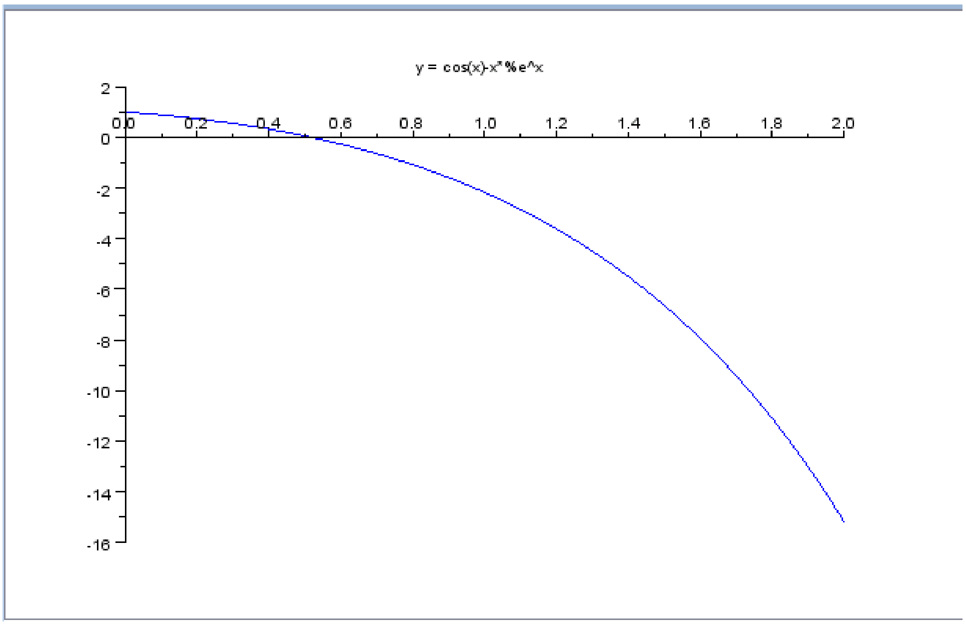

Figure 2.5: solution by secant and regula falsi

```
  0.201640 witin the permissible error of 10^-4,
```

---

check Appendix AP 44 for dependency:

`Vsecant.sce`

check Appendix AP 45 for dependency:

`regulafalsi.sce`

Scilab code Exa 2.6 `solution by secant and regula falsi`

```
                                        // The equation
                                           cos(x)-x*%e^x
                                           ==0 has  real
                                            roots.
```

```
// the graph of this function can be observed here.
xset('window',3);
x=0:.01:2;                                                    // defining the range of x.
deff('[y]=f(x)','y=cos(x)-x*%e^x');                  //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 = cos(x)-x*%e^x')

// from the above plot we can infre that the function has root between
// the interval (0,1)


// a=0;b=1,


                                        // Solution by secant method





// since in the example 2.6 we have no specification
```

```
     of the no. of itterations ,
// we define a function 'secant' and execute it.



secant(0,1,f)              // we call a user-defined
    function 'secant' so as to find the approximate
// root of the equation with a defined permissible
    error.



// hence the approximate root occured in secant
    method witin the permissible error of 10^-5 is   ,



                                  // solution by regular
                                      falsi  method





// since in the example 2.6 we have no specification
     of the no. of itterations ,


regulafalsi(0,1,f)              // we call a user-
    defined function 'regularfalsi' so as to find the
     approximate
// root of the equation with a defined permissible
    error.
```

check Appendix AP 43 for dependency:

`Vnewton4.sce`

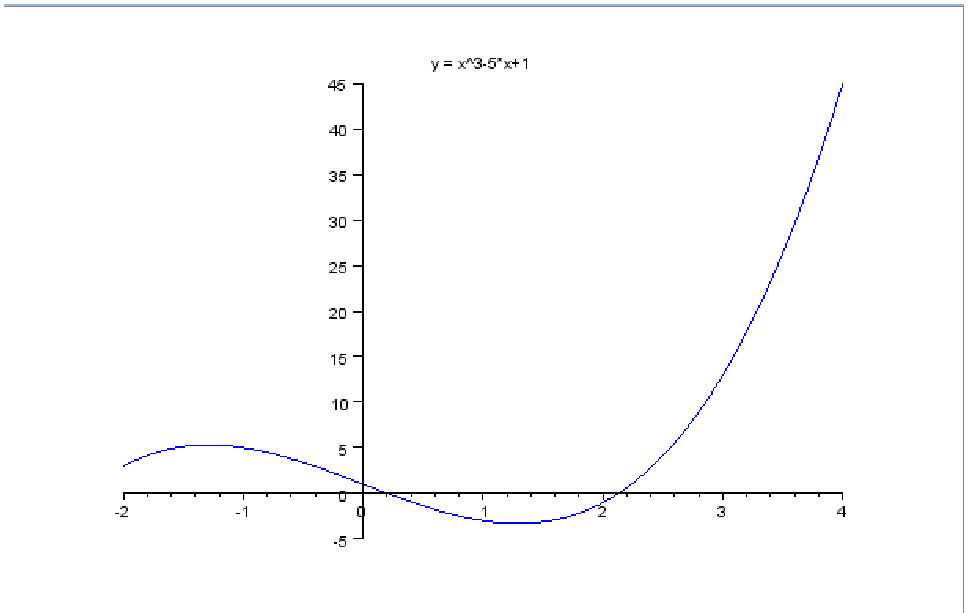

Figure 2.6: solution to the equation by newton raphson method

Scilab code Exa 2.7 `solution to the equation by newton raphson method`

```
// The equation x^3-5*x+1==0 has real roots.
// the graph of this function can be observed here.
xset('window',6);
x=-2:.01:4;
//
```

```
defining the range of x.
deff('[y]=f(x)','y=x^3-5*x+1');                              // defining the function.
deff('[y]=fp(x)','y=3*x^2-5');
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^3-5*x+1')

// from the above plot we can infre that the function has roots between
// the intervals (0,1),(2,3).
// since we have been asked for the smallest positive root of the equation,
// we are intrested on the interval (0,1)
// a=0;b=1,

// since in the example 2.7 we have been asked to perform 4 itterations ,
// the approximate  root after 4 iterations can be observed.


newton4(0.5,f,fp)


// hence the approximate root after 4 iterations is 0.201640 witin the permissible error of 10^-15,
```

check Appendix AP 43 for dependency:

`Vnewton4.sce`

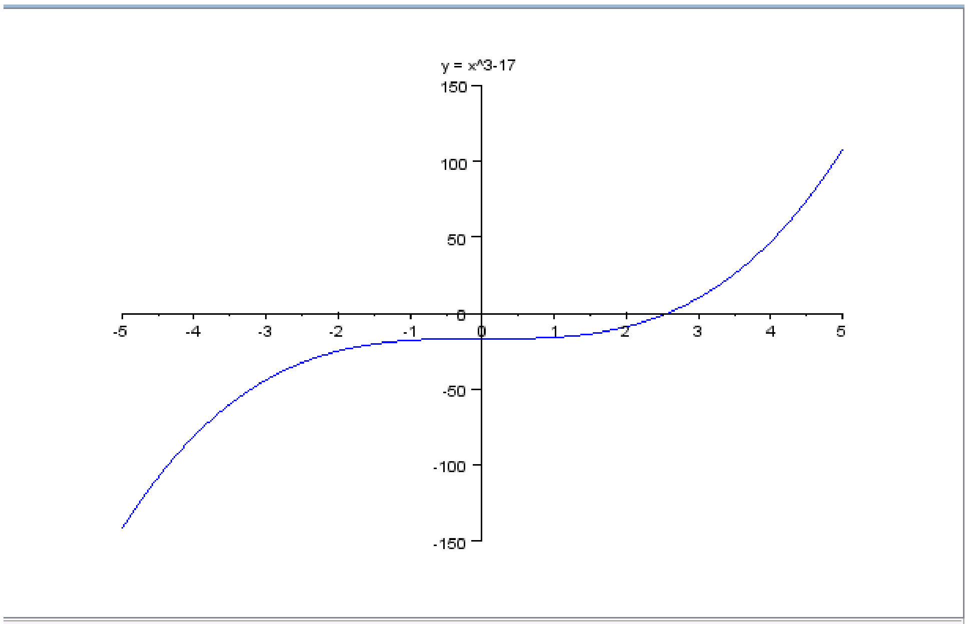

Figure 2.7: solution to the equation by newton raphson method

Scilab code Exa 2.8 `solution to the equation by newton raphson method`

```
// The equation x^3-17==0 has three real roots.
// the graph of this function can be observed here.
xset('window',7);
x=-5:.001:5;
//
```

```
defining the range of x.
deff('[y]=f(x)','y=x^3-17');                    //defining the cunction.
deff('[y]=fp(x)','y=3*x^2');
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^3-17')

// from the above plot we can infre that the function has root between
// the interval (2,3).



             //solution by newton raphson's method



 // since in example no.2.8 we have been asked to perform 4 iterations ,we define a fuction newton4'' which does newton raphson's method of finding approximate root upto 4 iterations ,



 newton4(2,f,fp)                    //calling the pre-defined function 'newton4'.
```

check Appendix AP 42 for dependency:

`Vnewton.sce`

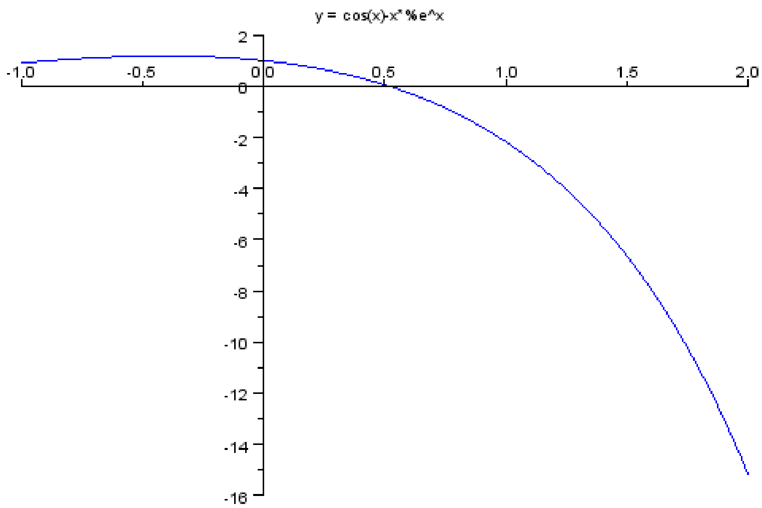

Figure 2.8: solution to the equation by newton raphson method

Scilab code Exa 2.9 `solution to the equation by newton raphson method`

```
// The equation cos(x)-x*%e^x==0 has real roots.
// the graph of this function can be observed here.
xset('window',8);
x=-1:.001:2; //
```

```
defining the range of x.
deff('[y]=f(x)','y=cos(x)-x*%e^x'); //defining the cunction.
deff('[y]=fp(x)','y=-sin(x)-x*%e^x-%e^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 = cos(x)-x*%e^x')

// from the above plot we can infre that the function has root between
// the interval (0,1)


// a=0;b=1,



            // solution by newton raphson's method with a permissible error of 10^-8.


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

newton(1,f,fp)
```

```


// hence the approximate root witin the permissible error of 10^-8 is 0.5177574.
```

check Appendix AP 61 for dependency:

muller3.sce

Scilab code Exa 2.11 solution to the given equation by muller method

```
                                        // The equation x^3-5*x+1==0 has real roots.
                                  // the graph of this function can be observed here.
xset('window',10);
x=-2:.01:4;                                  // defining the range of x.
deff('[y]=f(x)','y=x^3-5*x+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^3-5*x+1')

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

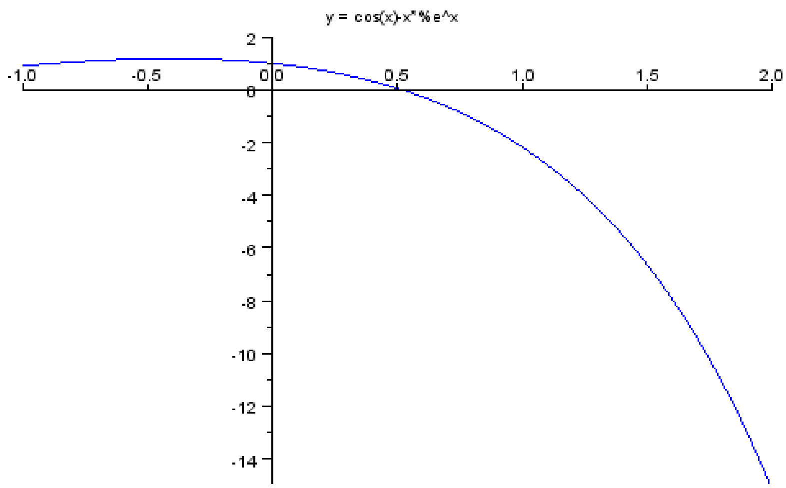

Figure 2.9: solution by five itrations of muller method

```
// since we have been asked for the smallest
    positive root of the equation,
// we are intrested on the interval (0,1)


          //sollution by muller method to 3 iterations
              .

muller3(0,.5,1,f)
```

check Appendix AP 60 for dependency:

`muller5.sce`

Scilab code Exa 2.12 `solution by five itrations of muller method`

```
// The equation cos(x)-x*%e^x ==0 has  real roots.
// the graph of this function can be observed here.
xset('window',8);
x=-1:.001:2;                                                    // defining the range of x.
deff('[y]=f(x)','y=cos(x)-x*%e^x');                  //defining the cunction.
deff('[y]=fp(x)','y=-sin(x)-x*%e^x-%e^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 = cos(x)-x*%e^x')

// from the above plot we can infre that the function has root between
// the interval (0,1)


          //sollution by muller method to 5 iterations.


muller5(-1,0,1,f)
```

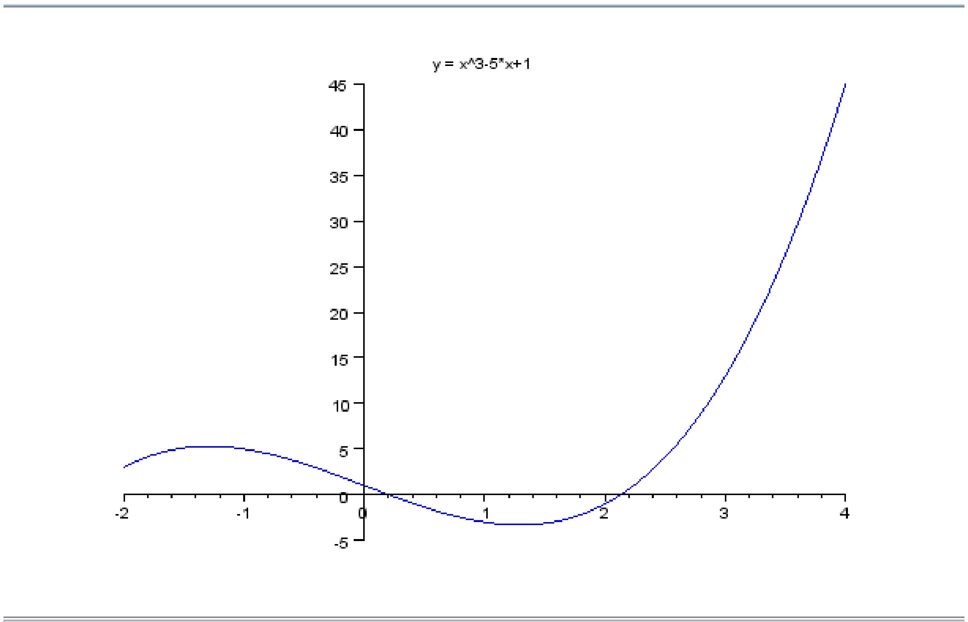

Figure 2.10: solution by chebeshev method

check Appendix AP 59 for dependency:

`chebyshev.sce`

Scilab code Exa 2.13 `solution by chebeshev method`

```
// The equation x^3-5*x+1==0 has real roots.
// the graph of this function can be observed here.
xset('window',12);
```

```
x=-2:.01:4;                                                    // defining the range of x.
deff('[y]=f(x)','y=x^3-5*x+1');                                // defining the function.
deff('[y]=fp(x)','y=3*x^2-5');
deff('[y]=fpp(x)','y=6*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^3-5*x+1')

// from the above plot we can infre that the function has roots between
// the intervals (0,1),(2,3).
// since we have been asked for the smallest positive root of the equation,
// we are intrested on the interval (0,1)
// a=0;b=1,


//                  solution by chebyshev method

// the approximate  root after 4 iterations can be observed.


chebyshev(0.5,f,fp)


// hence the approximate root  witin the permissible
```

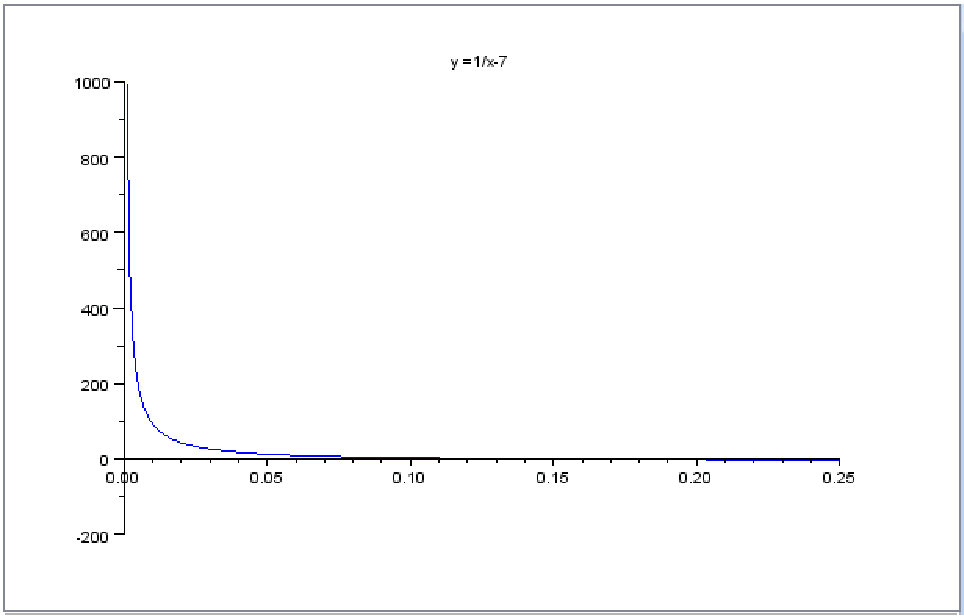

Figure 2.11: solution by chebeshev method

```
error of 10^-15 is .2016402,
```

---

check Appendix AP 59 for dependency:

`chebyshev.sce`

Scilab code Exa 2.14 `solution by chebeshev method`

```


                                        // The equation 1/x-7==0 has a real root.
                                        // the graph of this function
```

```
                                        can be observed here.
xset('window',13);
x=0.001:.001:.25;                                       // defining the range of x.
deff('[y]=f(x)','y=1/x-7');                          // defining the function.
deff('[y]=fp(x)','y=-1/x^2');
y=feval(x,f);

a=gca();

a.y_location = "origin";

a.x_location = "origin";
plot(x,y)                                               // instruction to plot the graph
title(' y =1/x-7')

// from the above plot we can infre that the function has roots between
// the interval (0,2/7)


            //solution by chebyshev method


 chebyshev(0.1,f,fp)                    //calling the pre-defined function 'chebyshev' to find the approximate root in the range of (0,2/7).
```

check Appendix AP 59 for dependency:

`chebyshev.sce`

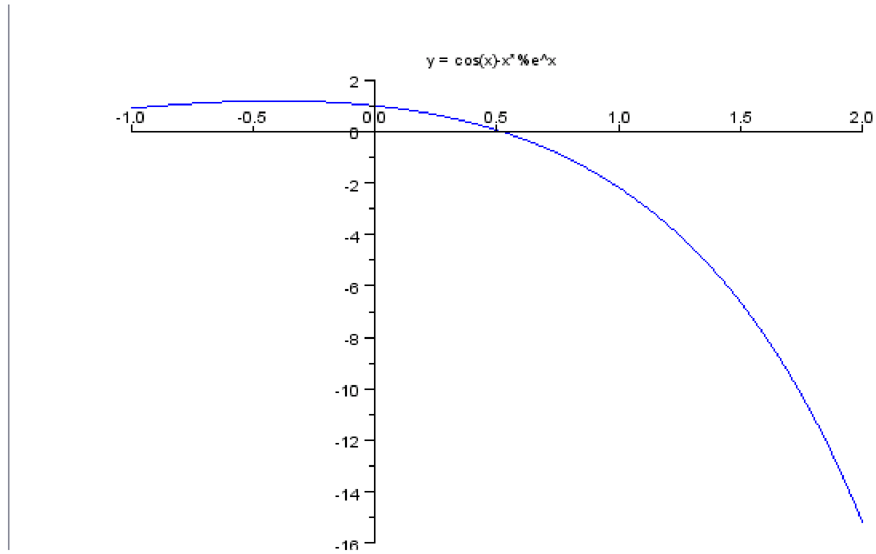

Figure 2.12: solution by chebeshev method

Scilab code Exa 2.15 `solution by chebeshev method`

```
                                               // The equation
   cos(x)-x*%e^x
   ==0 has  real
   roots.
                                               // the graph of
   this function
   can be
   observed here.
xset('window',8);
x=-1:.001:2;
                                                         //
   defining the range of x.
deff('[y]=f(x)','y=cos(x)-x*%e^x');
                    //defining the cunction.
deff('[y]=fp(x)','y=-sin(x)-x*%e^x-%e^x');
```

```
deff('[y]=fpp(x)','y=-cos(x)-x*%e^x-2*%e^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 = cos(x)-x*%e^x')

// from the above plot we can infre that the function has root between
// the interval (0,1)


// a=0;b=1,



                // solution by chebyshev with a permissible error of 10^-15.

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

chebyshev(1,f,fp)



// hence the approximate root witin the permissible error of 10^-15 is
```

check Appendix AP 58 for dependency:

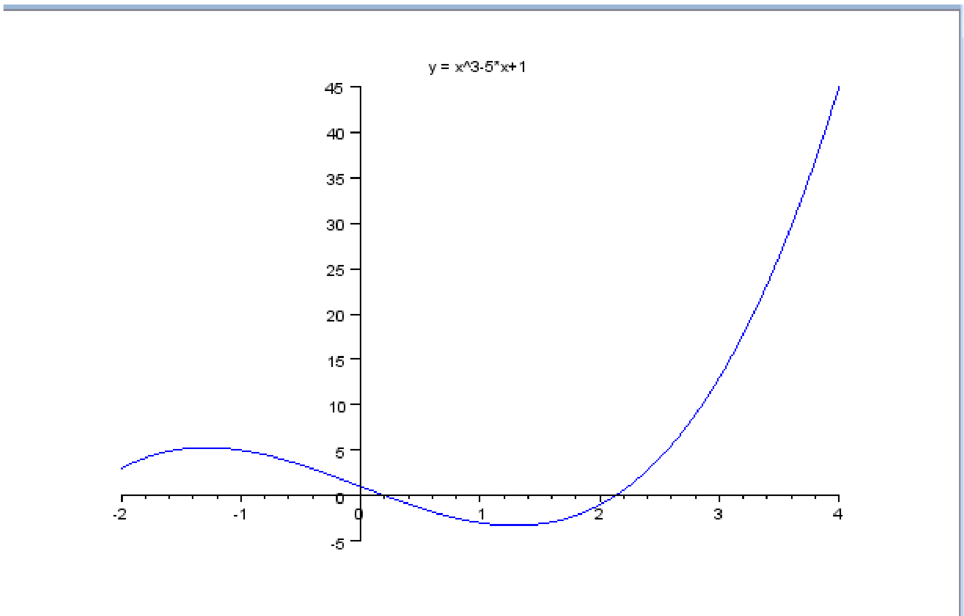

Figure 2.13: multipoint iteration

`multipoint_iteration31.sce`

check Appendix AP 57 for dependency:

`multipoint_iteration33.sce`

Scilab code Exa 2.16 `multipoint iteration`

```
// The equation x^3-5*x+1==0 has real roots.
// the graph of this function can be observed here.
```

```
xset('window',15);
x=-2:.01:4;                                                    // defining the range of x.
deff('[y]=f(x)','y=x^3-5*x+1');                                // defining the function.
deff('[y]=fp(x)','y=3*x^2-5');
deff('[y]=fpp(x)','y=6*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^3-5*x+1')

// from the above plot we can infre that the function has roots between
// the intervals (0,1),(2,3).
// since we have been asked for the smallest positive root of the equation,
// we are intrested on the interval (0,1)
// a=0;b=1,


//                  solution by multipoint iteration method

// the approximate  root after 3 iterations can be observed.


multipoint_iteration31(0.5,f,fp)

```

```
// hence the approximate root  witin the permissible
     error of 10^-15 is .201640,



multipoint_iteration33(0.5,f,fp)

// hence the approximate root  witin the permissible
     error of 10^-15 is .201640,
```

check Appendix AP 57 for dependency:

multipoint_iteration33.sce

Scilab code Exa 2.17 multipoint iteration

```
                                            // The equation
                                               cos(x)-x*%e^x
                                               ==0 has  real
                                               roots.
                                            // the graph of
                                               this function
                                               can be
                                               observed here.
xset('window',8);
x=-1:.001:2;
                                                           //
    defining the range of x.
deff('[y]=f(x)','y=cos(x)-x*%e^x');
                          //defining the function.
deff('[y]=fp(x)','y=-sin(x)-x*%e^x-%e^x');
deff('[y]=fpp(x)','y=-cos(x)-x*%e^x-2*%e^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 = cos(x)-x*%e^x')

// from the above plot we can infre that the
    function has root between
// the interval (0,1)


// a=0;b=1,



                // solution by multipoint_iteration
                    method using the formula given in
                    equation no.2.33.

// we call a user-defined function '
    multipoint_iteration33' so as to find the
    approximate
// root of the equation within the defined
    permissible error limit.

multipoint_iteration33(1,f,fp)


// hence the approximate root witin the permissible
    error of 10^-5 is 0.5177574.
```

check Appendix AP 56 for dependency:

`generaliteration.sce`

Scilab code Exa 2.23 general iteration

```
                                    // The equation 3*x^3+4*x^2+4*x+1==0 has three real roots.
                                    // the graph of this function can be observed here.
xset('window',22);
x=-1.5:.001:1.5;                                            // defining the range of x.
deff('[y]=f(x)','y=3*x^3+4*x^2+4*x+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 =3*x^3+4*x^2+4*x+1')

// from the above plot we can infre that the function has root between
// the interval (-1,0),

x0=-.5;                   // initial approximation


// let the iterative function g(x) be x+A*(3*x^3+4*x
```

```
^2+4*x+1) =g(x);

// gp(x)=(1+A*(9*x^2+8*x+4) )
// we need to choose a value for A , which makes abs(gp(x0))<1

// hence abs(gp(x0))=abs(1+9*A/4)

A=-1:.1:1;

abs(1+9*A/4)                // tryin to check the values of abs(gp(x0)) for different values of A.


// from the above values of 'A' and the values of 'abs(gp(x0))',
// we can infer that for the vales of 'A 'in the range (-.8,0) g(x ) will be giving a converging solution ,

// hence deliberatele we choose a to be -0.5,

A=-0.5;

deff('[y]=g(x)','y= x-0.5*(3*x^3+4*x^2+4*x+1)');
deff('[y]=gp(x)','y= 1-0.5*(9*x^2+8*x+4)');        // hence defining g(x) and gp(x),
generaliteration(x0,g,gp)
```

check Appendix AP 55 for dependency:

`aitken.sce`

Scilab code Exa 2.24.1 `solution by general iteration and aitken method`

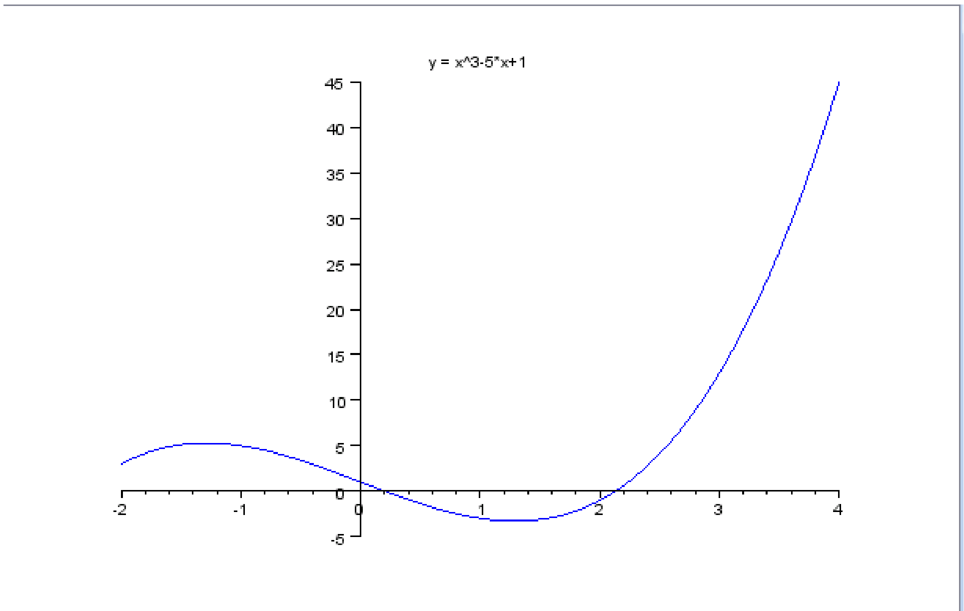

Figure 2.14: solution by general iteration and aitken method

```
// The equation x^3-5*x+1==0 has real roots.
// the graph of this function can be observed here.
xset('window',2);
x=-2:.01:4;                                        // defining the range of x.
deff('[y]=f(x)','y=x^3-5*x+1');                    // defining the function.
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^3-5*x+1')

// from the above plot we can infre that the
   function has roots between
// the intervals (0,1),(2,3).
// since we have been asked for the smallest
   positive root of the equation,
// we are intrested on the interval (0,1)

x0=.5;

         //solution using linear iteration method
            for the first two iterations and aitken'
            s process two times for the third
            iteration.

         deff('[y]=g(x)','y=1/5*(x^3+1)');
          deff('[y]=gp(x)','y=1/5*(3*x^2)');


generaliteration2(x0,g,gp)


// from the above iterations performed we can infer
   that-
x1=0.225;
x2=0.202278;




aitken(x0,x1,x2,g)          // calling the aitken
   method for one iteration
```

check Appendix AP 54 for dependency:

generaliteration2.sce

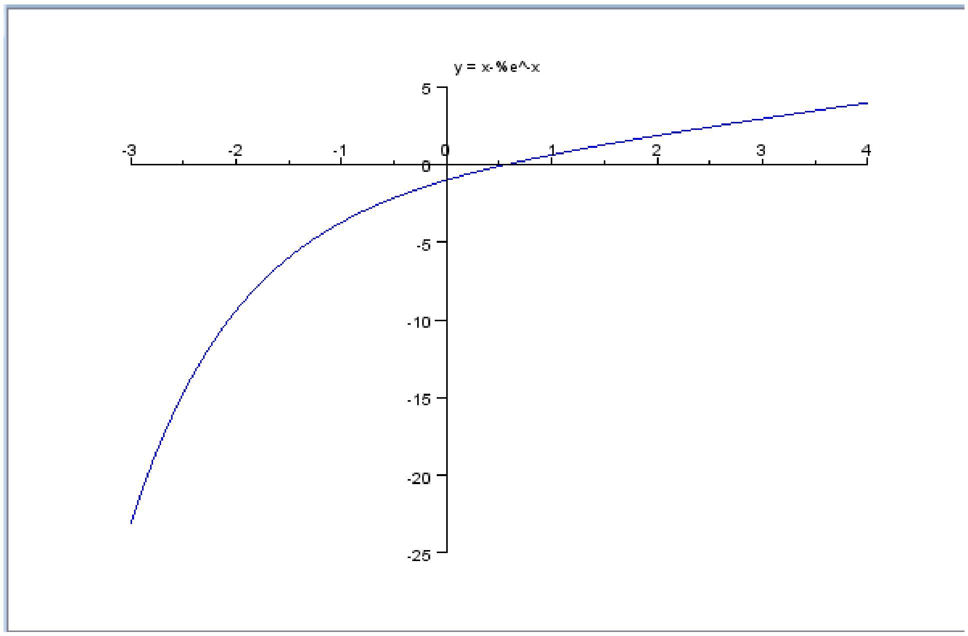

Figure 2.15: solution by general iteration and aitken method

check Appendix AP 55 for dependency:

aitken.sce

Scilab code Exa 2.24.2 solution by general iteration and aitken method

```
// The equation x-%e^-x ==0 has real roots.
// the graph of this function can be observed here.
xset('window',24);
```

```
x=-3:.01:4;                                                   //defining the range of x.
deff('[y]=f(x)','y=x-%e^-x');                                 //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-%e^-x')

// from the above plot we can infre that the function has root between
// the interval (0,1)

x0=1;

          //solution using linear iteration method for the first two iterations and aitken's process two times for the third iteration.



          deff('[y]=g(x)','y=%e^-x');
          deff('[y]=gp(x)','y=-%e^-x');


generaliteration2(x0,g,gp)


// from the above iterations performed we can infer
```

```
   that−
x1=0.367879;
x2=0.692201;




aitken(x0,x1,x2,g)            // calling the aitken
   method for one iteration
```

check Appendix AP 54 for dependency:

`generaliteration2.sce`

check Appendix AP 55 for dependency:

`aitken.sce`

check Appendix AP 54 for dependency:

`generaliteration2.sce`

Scilab code Exa 2.25 `solution by general iteration and aitken method`

```
                                                        // The equation
                                                           cos(x)−x∗%eˆx
                                                           ==0 has real
                                                           roots.
                                                        // the graph of
                                                           this function
                                                           can be
                                                           observed here.
xset('window',25);
x=0:.01:2;
                                                                   //
   defining the range of x.
```

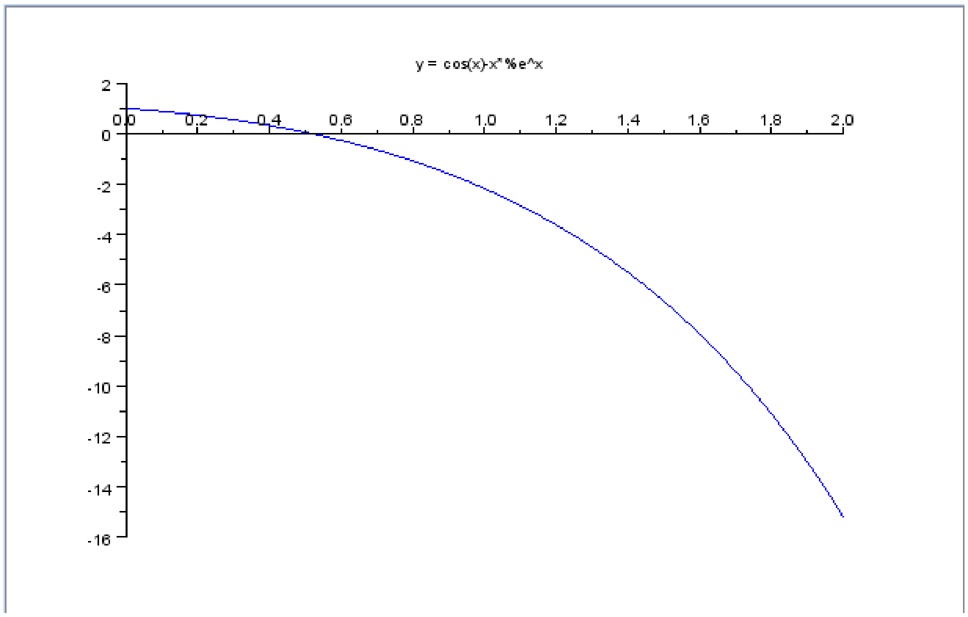

Figure 2.16: solution by general iteration and aitken method

```
deff('[y]=f(x)','y=cos(x)-x*%e^x'); //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 = cos(x)-x*%e^x')

// from the above plot we can infre that the function has root between
// the interval (0,1)

```

```
x0=0;

            //solution using linear iteration method
               for the first two iterations and aitken'
               s process two times for the third
               iteration.

            deff('[y]=g(x)','y=x+1/2*(cos(x)-x*%e^x)');
            deff('[y]=gp(x)','y=1+1/2*(-sin(x)-x*%e^x-
               %e^x)');


generaliteration2(x0,g,gp)


// from the above iterations performed we can infer
   that-
x1=0.50000000;
x2=0.5266110;



aitken(x0,x1,x2,g)          // calling the aitken
   method for one iteration
```

check Appendix AP 42 for dependency:

`Vnewton.sce`

check Appendix AP 53 for dependency:

`modified_newton.sce`

Scilab code Exa 2.26 `solution to the eq with multiple roots`

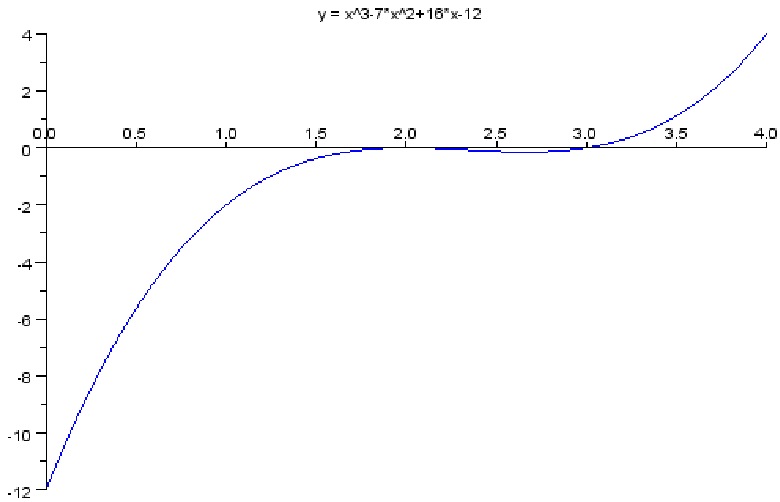

Figure 2.17: solution to the eq with multiple roots

```
// The equation x^3-7*x^2+16*x-12==0 has real roots.
// the graph of this function can be observed here.
xset('window',25);
x=0:.001:4;                    // defining the range of x.
deff('[y]=f(x)','y=x^3-7*x^2+16*x-12');          //defining the cunction.
deff('[y]=fp(x)','y=3*x^2-14*x+16');
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^3-7*x^2+16*x-12')




// given that the equation has double roots at x=2
    hence m=2;

m=2;

                  // solution by newton raphson
                     method


newton(1,f,fp)                    // calling the user
    defined function




                        //solution by modified
                           newton raphsons mathod



modified_newton(1,f,fp)
```

check Appendix AP 42 for dependency:

`Vnewton.sce`

check Appendix AP 43 for dependency:

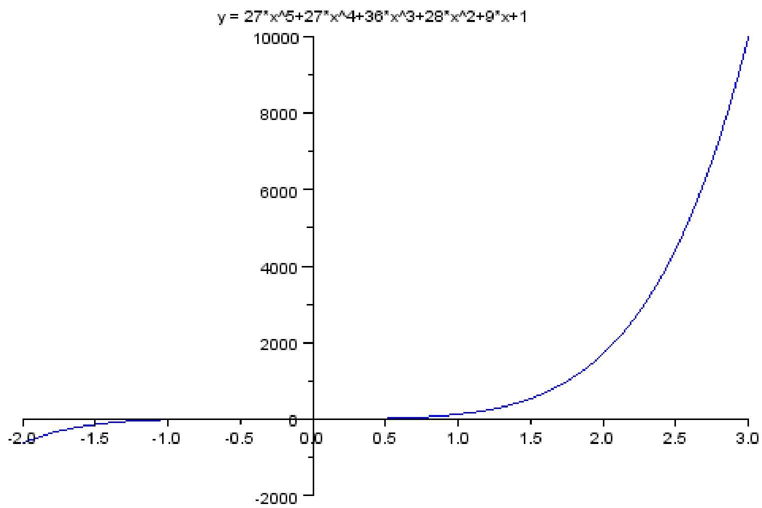

Figure 2.18: solution to the given transcendental equation

`Vnewton4.sce`

check Appendix AP 50 for dependency:

`newton63.sce`

check Appendix AP 51 for dependency:

`secant64.sce`

check Appendix AP 52 for dependency:

`secant65.sce`

Scilab code Exa 2.27 `solution to the given transcendental equation`

```
// The equation 27*x^5+27*x^4+36*x^3+28*x^2+9*x+1==0 has real roots.
// the graph of this function can be observed here.
xset('window',26);
x=-2:.001:3; // defining the range of x.
deff('[y]=f(x)','y=27*x^5+27*x^4+36*x^3+28*x^2+9*x+1'); //defining the cunction.
deff('[y]=fp(x)','y=27*5*x^4+27*4*x^3+36*3*x^2+28*2*x+9');
deff('[y]=fpp(x)','y=27*5*4*x^3+27*4*3*x^2+36*3*2*x+28*2');
y=feval(x,f);

a=gca();

a.y_location = "origin";

a.x_location = "origin";
plot(x,y) // instruction to plot the graph
title(' y = 27*x^5+27*x^4+36*x^3+28*x^2+9*x+1')




// solution by newton raphson method as per the equation no. 2.14
```

```


newton(-1,f,fp)                         // calling the user
    defined function


newton4(-1,f,fp)


                              // solution by newton
                                 raphson method as per the
                                  equation no.  2.63

newton63(-1,f,fp,fpp)                        // calling
   the user defined function
                                 // solution by the secant
                                    method defined to
                                   satisfy the equation
                                   no.2.64.


secant64(0,-1,f,fp)






                                 // solution by the secant
                                    method defined to
                                   satisfy the equation
                                   no.2.65.


secant65(0,-.5,f)
```

Chapter 3

SYSTEM OF LINEAR ALGEBRIC EQUATIONS AND EIGENVALUE PROBLEMS

Scilab code Exa 3.1 determinent

```
// example 3.1
//Positive definite

A=[12 4 -1;4 7 1;-1 1 6];
//check if the determinant of the leading minors is
    a positive value-

A1=A(1);
det(A1)
A2=A(1:2,1:2);
det(A2)
A3=A(1:3,1:3);
det(A3)

//we observe that the determinant of the leading
```

```
   minors is a positive ,hence the given matrix A is
    a positive definite.
```

Scilab code Exa 3.2 property A in the book

```
//example no.3.2
// show if a given matrix 'A' possesses 'property A'

A=[2 -1 0 -1;-1 2 -1 0;0 -1 2 0;0 0 -1 2 ]

P=[0 0 0 1;0 1 0 0;0 0 1 0;1 0 0 0]              //
   let us take the pemutation matrix as P

// then we find that

P*A*P'

// is of the form (3.2) . hence the matrix 'A' has
   property A
```

Scilab code Exa 3.4 solution to the system of equations

```
// example no.3.4
// solve the system of equations

//(a) . by using cramer's rule,

A=[1 2 -1;3 6 1;3 3 2]

B1=[2 2 -1;1 6 1;3 3 2]

B2=[1 2 -1;3 1 1;3 3 2]

```

```
B3=[1 2 2;3 6 1; 3 3 3]


//we know;

X1=det(B1)/det(A)
X2=det(B2)/det(A)
X3=det(B3)/det(A)


//(b). by detemining the inverse of the coefficient
    matrix

A=[1 2 -1;3 6 1;3 3 2]

b=[2 ;1 ;3]

//we know;

X=inv(A)*b
```

check Appendix AP 34 for dependency:

`Vgausselim.sce`

Scilab code Exa 3.5 `solution by gauss elimination method`

```
//example-3.5
//caption-solution by gauss elimibnation method

A=[10 -1 2;1 10 -1;2 3 20]                    //matrices
    A and b from the above
                                                     //
                                                       system

                                                       of
```

```
                                                         equations
b=[4;3;7]

gausselim(A,b)            //call gauss elimination
   function to solve the
                                  //matrices A and b
```

check Appendix AP 33 for dependency:

`pivotgausselim.sci`

Scilab code Exa 3.6 `solution by pivoted gauss elimination method`

```
//example-3.6
//caption-solution by gauss elimibnation method

A=[1 1 1;3 3 4;2 1 3]               //matrices A and
    b from the above
                                                     //
                                                       system

                                                       of

                                                       equations

b=[6;20;13]

pivotgausselim(A,b)           //call gauss
   elimination function to solve the
                                  //matrices A and b
```

check Appendix AP 33 for dependency:

`pivotgausselim.sci`

Scilab code Exa 3.8 `solution by pivoted gauss elimination method`

```
//example no. 3.8
// solving the matrix equation with partial pivoting
    in gauss elimination

A=[2 1 1 -2;4 0 2 1;3 2 2 0;1 3 2 -1]

b=[-10;8;7;-5]

pivotgausselim(A,b)
```

check Appendix AP 32 for dependency:

`jordan.sce`

Scilab code Exa 3.9 `solution using the inverse of the matrix`

```
//example no.3.9
//solving the system using inverse of the cofficient
    matrix

A=[1 1 1;4 3 -1;3 5 3]

I=[1 0 0;0 1 0;0 0 1]

b=[1 ;6 ;4]

M=jorden(A,I)

IA=M(1:3,4:6)

X=IA*b
```

check Appendix AP 41 for dependency:

`LandU.sce`

check Appendix AP 38 for dependency:

`back.sce`

check Appendix AP 40 for dependency:

`fore.sce`

Scilab code Exa 3.10 `decomposition method`

```
//example no. 3.10
//solve system by decomposition method

A=[1 1 1;4 3 -1;3 5 3]
n=3;

b=[1;6;4]

[U,L]=LandU(A,3)

Z=fore(L,b)

X=back(U,Z)
```

check Appendix AP 41 for dependency:

`LandU.sce`

Scilab code Exa 3.11 `inverse using LU decoposition`

```
//example no.3.11
//caption: Inverse using LU decomposition

A=[3 2 1;2 3 2;1 2 2]

[U,L]=LandU(A,3)        // call LandU function to
    evaluate U,L of A,

//since  A=L*U ,
// inv(A)=inv(U)*inv(L)
// let inv(A)=AI

AI=U^-1*L^-1
```

check Appendix AP 41 for dependency:

`LandU.sce`

check Appendix AP 38 for dependency:

`back.sce`

check Appendix AP 40 for dependency:

`fore.sce`

Scilab code Exa 3.12 solution by decomposition method

```
//example no. 3.12
//solve system by decomposition method

A=[1 1 -1;2 2 5;3 2 -3]

b=[2;-3;6]



```

```
        // hence we can observe that LU decomposition method fails to solve this system since the pivot L(2,2)=0;


        //we note that the coefficient matrix is not a positive definite matrix and hence its LU decomposition is not guaranteed,


        //if we interchange the rows of A as shown below the LU decomposition would work,

        A=[3 2 -3;2 2 5;1 1 -1]

        b=[6;-3;2]

        [U,L]=LandU(A,3)         // call LandU function to evaluate U,L of A,

n=3;
Z=fore(L,b);

X=back(U,Z)
```

check Appendix AP 41 for dependency:

`LandU.sce`

check Appendix AP 38 for dependency:

`back.sce`

check Appendix AP 40 for dependency:

`fore.sce`

Scilab code Exa 3.13 `LU decomposition`

```
//example no. 3.13
//solve system by LU decomposition method

A=[2 1 1 -2;4 0 2 1;3 2 2 0;1 3 2 -1]

b=[-10;8;7;-5]

[U,L]=LandU(A,4)
n=4;
Z=fore(L,b);

X=back(U,Z)

//since  A=L*U ,
// inv(A)=inv(U)*inv(L)
// let inv(A)=AI

AI=U^-1*L^-1
```

check Appendix AP 38 for dependency:

`back.sce`

check Appendix AP 39 for dependency:

`cholesky.sce`

check Appendix AP 12 for dependency:

`fact.sci`

Scilab code Exa 3.14 `cholesky method`

```
//example no. 3.14
//solve system by cholesky method

A=[1 2 3;2 8 22;3 22 82]
```

```

b=[5;6;-10]

L=cholesky (A,3)  //call cholesky function to
    evaluate the root of the system
n=3;
Z=fore(L,b);

X=back(L',Z)
```

check Appendix AP 38 for dependency:

`back.sce`

check Appendix AP 39 for dependency:

`cholesky.sce`

check Appendix AP 40 for dependency:

`fore.sce`

Scilab code Exa 3.15 `cholesky method`

```
//example no. 3.15
//solve system by cholesky method

A=[4 -1 0 0;-1 4 -1 0;0 -1 4 -1;0 0 -1 4]

b=[1;0;0;0]

L=cholesky (A,4)  //call cholesky function to
    evaluate the root of the system

n=4;
Z=fore(L,b);

```

```
X=back(L',Z)

//since  A=L*L' ,
// inv(A)=inv(L')*inv(L)
// let inv(A)=AI

AI=L'^-1*L^-1
```

check Appendix AP 37 for dependency:

`jacobiiteration.sce`

Scilab code Exa 3.21 `jacobi iteration method`

```
//example no. 3.21
//solve the system by jacobi iteration method

A=[4 1 1;1 5 2;1 2 3]

b=[2 ;-6;-4]

N=3;          //no. of ierations
n=3;          // order of the matrix is n*n

X=[.5;-.5;-.5]                 //initial approximation


jacobiiteration(A,n,N,X,b)                //call the
    function which performs jacobi iteration method
    to solve the system
```

check Appendix AP 36 for dependency:

`Vgaussseidel.sce`

Scilab code Exa 3.22 `solution by gauss siedal method`

```
//example no. 3.22
//solve the system by gauss seidel method

A=[2 -1 0;-1 2 -1;0 -1 2]

b=[7;1;1]

N=3;          //no. of ierations
n=3;          // order of the matrix is n*n

X=[0;0;0]                //initial approximation


gaussseidel(A,n,N,X,b)                  //call the
   function which performs gauss seidel method to
   solve the system
```

check Appendix AP 35 for dependency:

`geigenvectors.sci`

Scilab code Exa 3.27 `eigen vale and eigen vector`

```
// example 3.27
// a) find eigenvalue and eigen vector;
// b) verify inv(S)*A*S is a diagonal matrix;

// 1)
A=[1 2 -2 ;1 1 1;1 3 -1];

B=[1 0 0;0 1 0; 0 0 1];

 [x,lam] = geigenvectors(A,B);

```

```
 inv(x)*A*x

 // 2)
A=[3 2 2;2 5 2;2 2 3];

B=[1 0 0;0 1 0; 0 0 1];


 [x,lam] = geigenvectors(A,B);

 inv(x)*A*x
```

Chapter 4

INTERPOLATION AND APPROXIMATION

check Appendix AP 29 for dependency:

`NDDinterpol.sci`

check Appendix AP 28 for dependency:

`aitkeninterpol.sci`

check Appendix AP 25 for dependency:

`legrangeinterpol.sci`

Scilab code Exa 4.3 `linear interpolation polinomial`

```
// example 4.3
// find the linear interpolation polinomial
// using

disp('f(2)=4');
disp('f(2.5)=5.5');
// 1)lagrange interpolation,

```

```
P1=legrangeinterpol (2,2.5,4,5.5)
// 2)aitken's iterated interpolation ,

P1=aitkeninterpol (2,2.5,4,5.5)

// 3) newton devided differance interpolation ,

P1=NDDinterpol (2,2.5,4,5.5)

// hence approximate value of f(2.2)= 4.6;
```

check Appendix AP 25 for dependency:

`legrangeinterpol.sci`

Scilab code Exa 4.4 `linear interpolation polinomial`

```
// example 4.4
// find the linear interpolation polinomial
// using lagrange interpolation ,

disp('sin(.1)=.09983; sin(.2)=.19867');


P1=legrangeinterpol (.1,.2,.09983,.19867)

// hence;
disp('P(.15)=.00099+.9884*.15')
disp('P(0.15)=0.14925');
```

check Appendix AP 25 for dependency:

`legrangeinterpol.sci`

Scilab code Exa 4.6 `legrange linear interpolation polinomial`

```
//  example :4.6
//  caption : obtain the legrange linear interpolating polinomial


// 1)  obtain the legrange linear interpolating polinomial in the interval [1,3] and obtain approximate value of f(1.5),f(2.5);
x0=1;x1=2;x2=3;f0=.8415;f1=.9093;f2=.1411;

P13=legrangeinterpol (x0,x2,f0,f2)                //in the range [1  ,3]


P12=legrangeinterpol (x0,x1,f0,f1)                  // in the range [1,2]

P23=legrangeinterpol (x1,x2,f1,f2)                // inthe range [2,3]

// from P23 we find that ; where as exact value is sin(2.5)=0.5985;
disp('P(1.5)=0.8754');
disp('exact value of sin(1.5)=.9975')
disp('P(2.5)=0.5252');
```

check Appendix AP 24 for dependency:

`lagrangefundamentalpoly.sci`

Scilab code Exa 4.7 `polynomial of degree two`

```
 // example 4.7
 // polinomial of degree 2;
```

```
// f(0)=1;f(1)=3;f(3)=55;

// using legrange fundamental polinomial rule,

x=[0 1 3];                          // arrainging the inputs of the function as elements of a row,
f=[1 3 55];                         // arrainging the outputs of the function as elements of a row,
n=2;                                // degree of the polinomial;


P2=lagrangefundamentalpoly(x,f,n)
```

check Appendix AP 24 for dependency:

`lagrangefundamentalpoly.sci`

Scilab code Exa 4.8 solution by quadratic interpolation

```

// example 4.8
// caption: solution by quadratic interpolation;

// x-degrees:[10 20 30]
// hence x in radians is
x=[3.14/18 3.14/9 3.14/6];
f=[1.1585 1.2817 1.3660];
n=2;


P2=lagrangefundamentalpoly(x,f,n)

// hence from P2 ,the exact value of f(3.14/12) is 1.2246;
```

```
// where as exact value is 1.2247;
```

check Appendix AP 23 for dependency:

NDDinterpol2.sci

check Appendix AP 22 for dependency:

iteratedinterpol.sci

Scilab code Exa 4.9 polinomial of degree two

```
// example 4.9
// caption :obtain the polinomial of degree 2

x=[0 1 3];
f=[1 3 55];
n=2;

// 1) iterated interpolation;


[L012,L02,L01]=iteratedinterpol (x,f,n)

//  2) newton divided diffrences interpolation;


 P2=NDDinterpol2 (x,f)
```

check Appendix AP 31 for dependency:

NBDP.sci

Scilab code Exa 4.15 forward and backward difference polynomial

```
// example 4.15:
// obtain the interpolate using backward differences
    polinomial


xL=[.1 .2 .3 .4 .5 ]'

f=[1.4 1.56 1.76 2 2.28]'
n=2;


// hence;
disp('P=1.4+(x-.1)*(.16/.1)+(x-.5)*(x-.4)*(.04/.02)'
    )
disp('P=2x^2+x+1.28');

// 1) obtain the interpolate at x=0.25;
x=0.25;
[P]=NBDP(x,n,xL,f);
P
disp('f(.25)=1.655');


// 2) obtain the interpolate at x=0.35;
x=0.35;
[P]=NBDP(x,n,xL,f);
P
disp('f(.35)=1.875');
```

check Appendix AP 30 for dependency:

`hermiteinterpol.sci`

Scilab code Exa 4.20 `hermite interpolation`

```
// example 4.20;
```

```
// hermite interpolation:

x=[-1 0 1];

f=[1  1 3];

fp=[-5 1 7];

 P= hermiteinterpol(x,f,fp);

// hence;
disp('f(-0.5)=3/8');
disp('f(0.5)=11/8');
```

check Appendix AP 25 for dependency:

`legrangeinterpol.sci`

Scilab code Exa 4.21 piecewise linear interpolating polinomial

```
// example: 4.21;
// piecewise linear interpolating polinomials:

x1=1;x2=2; x3=4;x4=8;
f1=3; f2=7; f3=21; f4=73;
//  we need to apply legranges interpolation in sub-
    ranges [1,2];[2,4],[4,8];

 x=poly(0,"x");

P1=legrangeinterpol (x1,x2,f1,f2);          // in the
     range [1,2]
P1

P1=legrangeinterpol (x2,x3,f2,f3);          // in the
     range [2,4]
```

```
P1

P1=legrangeinterpol (x3,x4,f3,f4);          // in the
    range [4,8]
P1
```

check Appendix AP 24 for dependency:

`lagrangefundamentalpoly.sci`

Scilab code Exa 4.22 `piecewise quadratic interpolating polinomial`

```
// example: 4.22;
// piecewise quadratic interpolating polinomials:

X=[-3 -2 -1 1 3 6 7];
F=[369 222 171 165 207 990 1779];
//  we need to apply legranges interpolation in sub-
   ranges [-3 ,-1];[-1,3],[3,7];

 x=poly(0,"x");

  // 1) in the range [-3,-1]
  x=[-3 -2 -1];
  f=[369 222 171];
  n=2;
P2=lagrangefundamentalpoly(x,f,n);

 // 2) in the range [-1,3]
 x=[-1 1 3];
 f=[171 165 207];
n=2;
P2=lagrangefundamentalpoly(x,f,n)

 // 3) in the range [3,7]
  x=[3 6 7];
```

```
 f=[207 990 1779];
n=2;
P2=lagrangefundamentalpoly(x,f,n)



// hence, we obtain the values of f(-2.5)=48;  f
   (6.5)=1351.5;
```

check Appendix AP 24 for dependency:

`lagrangefundamentalpoly.sci`

Scilab code Exa 4.23 `piecewise cubical interpolating polinomial`

```
// example: 4.23;
// piecewise cubical interpolating polinomials:

X=[-3 -2 -1 1 3 6 7];
F=[369 222 171 165 207 990 1779];
//  we need to apply legranges interpolation in sub-
   ranges [-3  ,1];[1 ,7];


 x=poly(0,"x");

  // 1) in the range [-3,1]
  x=[-3 -2 -1 1];
  f=[369 222 171 165];
  n=3;
P2=lagrangefundamentalpoly(x,f,n);

 // 2) in the range [1,7]
 x=[1 3 6 7];
 f=[165 207 990 1779];
n=3;
```

```
P2=lagrangefundamentalpoly(x,f,n)



 // hence ,
 disp('f(6.5)=1339.25');
```

Scilab code Exa 4.31 `linear approximation`

```
// example 4.31
// obtain the linear polinomial approximation to the
    function f(x)=x^3

// let P(x)=a0*x+a1

// hence I(a0,a1)= integral (x^3-(a0*x+a1))^2 in the
    interval [0,1]

printf('I=1/7-2*(a0/5+a1/4)+a0^2/3+a0*a1+a1^2')
printf('dI/da0 = -2/5+2/3*a0+a1=0')

printf('dI/da1 = -1/2+a0+2*a1=0')

// hence

printf('[2/3 1;1 2]*[a0 ;a1]=[2/5; 1/2]')

// solving for a0 and a1;

a0=9/10;
a1=-1/5;
// hence considering the polinomial with intercept
   P1(x)=(9*x-2)/10;

// considering the polinomial approximation through
```

```
origin P2(x)=3*x/5;
```

Scilab code Exa 4.32 `linear polinomial approximation`

```
// example 4.32
// obtain the linear polinomial approximation to the
   function f(x)=x^1/2

// let P(x)=a0*x+a1


// for n=1;
// hence I(c0,c1)= integral (x^1/2-(c1*x+c0))^2 in
   the interval [0,1]


printf('dI/dc0 = -2*(2/3-c0-c1/2)=0')

printf('dI/dc1 =-2*(2/5-c0/2-c1/3) =0')

// hence

printf('[1 1/2;1/2 1/3]*[c0 ;c1]=[-4/3; -4/5]')

//  hence solving for c0 and c1;


// the first degree square approximation P(x)
   =4*(1+3*x)/15;

// for n=2;

// hence I(c0,c1,c2)= integral (x^1/2-(c2*x^2+c1*x+
   c0))^2 in the interval [0,1]

```

```
printf('dI/dc0 = (2/3-c0-c1/2-c2/2)=0')

printf('dI/dc1 =(2/5-c0/2-c1/3-c2/4) =0')

printf('dI/dc2 =(2/7-c0/3-c1/4-c2/5) =0')


// hence

printf('[1 1/2 1/2;1/2 1/3 1/4;1/3 1/4 1/5]*[c0 ;c1;
    c2]=[-2/3; -2/5;-2/7]')

//  hence solving for c0,c1 and c2;


// the first degree square approximation P(x)=(6+48*
    x-20*x^2)/35;
```

check Appendix AP 27 for dependency:

`straightlineapprox.sce`

Scilab code Exa 4.34 `least square straight fit`

```
// example 4.34

// obtain least square straight line fit

x=[.2 .4 .6 .8 1];
f=[.447 .632 .775 .894 1];


[P]=straightlineapprox(x,f)          // call of the
    function to get the desired solution
```

check Appendix AP 26 for dependency:

quadraticapprox.sci

Scilab code Exa 4.35 least square approximation

```
// example 4.35

// obtain least square approximation of second
   degree;
x=[-2 -1 0 1 2];
f=[15 1 1 3 19];

[P]=quadraticapprox(x,f)        // call of the
   function to get the desired solution
```

Scilab code Exa 4.36 least square fit

```
// example 4.36
// method of least squares to fit the data to the
   curve P(x)=c0*X+c1/squrt(X)

x=[.2 .3 .5 1 2];
f=[16 14 11 6 3];

// I(c0,c1)= summation of (f(x)-(c0*X+c1/sqrt(X)))^2

// hence on parcially derivating the summation,

n=length(x);m=length(f);
if m<>n then
   error('linreg - Vectors x and f are not of the
      same length.');
```

```
   abort;
end;

s1=0;                    // s1= summation of x(i
   )*f(i)
s2=0;                    // s2= summation of f(i
   )/sqrt(x(i))
s3=0;
for i=1:n
    s1=s1+x(i)*f(i);
    s2=s2+f(i)/sqrt(x(i));
    s3=s3+1/x(i);
end

c0=det([s1 sum(sqrt(x));s2 s3])/det([sum(x^2) sum(
   sqrt(x));sum(sqrt(x)) s3])

c1=det([sum(x^2) s1;sum(sqrt(x)) s2])/det([sum(x^2)
   sum(sqrt(x));sum(sqrt(x)) s3])
X=poly(0,"X");
P=c0*X+c1/X^1/2
// hence considering the polinomial P(x)=7.5961*X
   ^1/2-1.1836*X
```

Scilab code Exa 4.37 least square fit

```
// example 4.37
// method of least squares to fit the data to the
   curve P(x)=a*%e^(-3*t)+b*%e^(-2*t);

t=[.1 .2 .3 .4];
f=[.76 .58 .44 .35];

// I(c0,c1)= summation of (f(x)-a*%e^(-3*t)+b*%e
   ^(-2*t))
```

```
// hence on parcially derivating the summation,

n=length(t);m=length(f);
if m<>n then
   error('linreg - Vectors t and f are not of the same length.');
   abort;
end;

s1=0;                          // s1= summation of f(i)*%e^(-3*t(i));
s2=0;                          // s2= summation of f(i)*%e^(-2*t(i));

for i=1:n
    s1=s1+f(i)*%e^(-3*t(i));
    s2=s2+f(i)*%e^(-2*t(i));

end

a=det([s1 sum(%e^(-5*t)); s2 sum(%e^(-4*t))])/det([sum(%e^(-6*t)) sum(%e^(-5*t)); sum(%e^(-5*t)) sum(%e^(-4*t))])

b=det([sum(%e^(-6*t)) s1; sum(%e^(-5*t)) s2])/det([sum(%e^(-6*t)) sum(%e^(-5*t)); sum(%e^(-5*t)) sum(%e^(-4*t))])

// hence  considering the polinomial  P(t)=.06853*%e^(-3*t)+0.3058*%e^(-2*t)
```

Scilab code Exa 4.38 `gram schmidt orthogonalisation`

```
// example 4.38
```

```
// gram schmidt orthogonalisation

W=1;
 x=poly(0,"x");
    P0=1;
    phi0=P0;
      a10=integrate('W*x*phi0','x',0,1)/integrate('W*1*phi0','x',0,1)
    P1=x-a10*phi0
    phi1=P1;

      a20=integrate('W*x^2*phi0','x',0,1)/integrate('W*1*phi0','x',0,1)

    a21=integrate('(x^2)*(x-1/2)','x',0,1)/integrate('(x-1/2)^2','x',0,1)

    P2=x^2-a20*x-a21*phi1

// since ,I= intgral [x^(1/2)-c0*P0-c1*P1-c2*P2]^2 inthe range [0,1]

// hence partially derivating I

c0=integrate('x^(1/2)','x',0,1)/integrate('1','x',0,1)
c1=integrate('(x^(1/2))*(x-(1/2))','x',0,1)/integrate('(x-(1/2))^2','x',0,1)
c1=integrate('(x^(1/2))*(x^2-4*x/3+1/2)','x',0,1)/integrate('(x^2-4*x/3+1/2)^2','x',0,1)
```

Scilab code Exa 4.39 gram schmidt orthogonalisation

```
// example 4.39
// gram schmidt orthogonalisation
```

```
// 1)
 W=1;
 x=poly(0,"x");
    P0=1
    phi0=P0;
      a10=0;
    P1=x-a10*phi0
    phi1=P1;

     a20=integrate('x^2','x',-1,1)/integrate('W*1*
        phi0','x',-1,1);

    a21=integrate('(x^3)','x',-1,1)/integrate('(x)^2
        ','x',-1,1);

    P2=x^2-a20*x-a21*phi1


// 2)
disp(' W=1/(1-x^2)^(1/2)');
 x=poly(0,"x");
    P0=1
    phi0=P0;
      a10=0;
    P1=x-a10*phi0
    phi1=P1;

     a20=integrate('x^2/(1-x^2)^(1/2)','x',-1,1)/
        integrate('1/(1-x^2)^(1/2)','x',-1,1);

    a21=0;                         // since x^3 is
       an odd function;

    P2=x^2-a20*x-a21*phi1
```

Scilab code Exa 4.41 chebishev polinomial

```
// example 4.41
// using chebyshev polinomials obtain least squares approximation of second degree;

// the chebeshev polinomials;
x=poly(0,"x");
T0=1;
T1=x;
T2=2*x^2-1;


// I=integrate ('1/(1-x^2)^(1/2)*(x^4-c0*T0-c1*T1-c2*T2)^2','x',-1,1)

// since;
c0=integrate('(1/3.14)*(x^4)/(1-x^2)^(1/2)','x',-1,1)

c1=integrate('(2/3.14)*(x^5)/(1-x^2)^(1/2)','x',-1,1)

c2=integrate('(2/3.14)*(x^4)*(2*x^2-1)/(1-x^2)^(1/2)','x',-1,1)

 f=(3/8)*T0+(1/2)*T2;
```

Chapter 5

DIFFERENTIATION AND INTEGRATION

check Appendix AP 21 for dependency:

`linearinterpol.sci`

Scilab code Exa 5.1 `linear interpolation`

```
// example: 5.1
// linear and quadratic interpolation:

// f(x)=ln x;

xL=[2 2.2 2.6];
f=[.69315 .78846 .95551];

// 1) fp(2) with linear interpolation;

fp=linearinterpol(xL,f);
disp(fp);
```

Scilab code Exa 5.2 `quadratic interpolation`

```
// example 5.2
// evaluate fp(.8) and fpp(.8) with quadratic
   interpolation;

xL=[.4 .6 .8];
f=[.0256 .1296 .4096];
h=.2;

fp=(1/2*h)*(f(1)-4*f(2)+3*f(3))
fpp=(1/h^2)*(f(1)-2*f(2)+f(3))
                              // from equation 5.22c
   and 5.24c in the book;
```

check Appendix AP 20 for dependency:

`jacobianmat.sci`

Scilab code Exa 5.10 `jacobian matrix of the given system`

```
// example 5.10;
// find the jacobian matrix;


// given two functions in x,y;
// and the point at which the jacobian has to be
   found out;

deff('[w]=f1(x,y)','w=x^2+y^2-x');

deff('[q]=f2(x,y)','q=x^2-y^2-y');

h=1;k=1;

J= jacobianmat (f1,f2,h,k);
```

```
disp(J);
```

check Appendix AP 18 for dependency:

simpson.sci

check Appendix AP 19 for dependency:

trapezoidal.sci

Scilab code Exa 5.11 solution by trapizoidal and simpsons

```
// example : 5.11
// solve the definite integral by 1) trapezoidal
   rule , 2)simpsons rule
// exact value of the integral is ln 2= 0.693147,

deff('[y]=F(x)','y=1/(1+x)')

//1)trapezoidal rule ,

a=0;
b=1;
I =trapezoidal(0,1,F)
disp(error =.75-.693147)

// simpson's rule

I=simpson(a,b,F)

disp(error =.694444-.693147)
```

Scilab code Exa 5.12 integral approximation by mid point and two point

```
// example 5.12
// caption: solve the integral by 1)mid-point rule
   ,2)two-point open type rule


// let integration of f(x)=sin(x)/(x) in the range
   [0,1]  is equal to I1 and I2
// 1)mid -point rule;
a=0;b=1;
h=(b-a)/2;

x=0:h:1;
deff('[y]=f(x)','y=sin(x)/x')
I1=2*h*f(x(1)+h)


//2)  two-point open type rule
h=(b-a)/3;
I2=(3/2)*h*(f(x(1)+h)+f(x(1)+2*h))
```

check Appendix AP 17 for dependency:

`simpson38.sci`

Scilab code Exa 5.13 `integral approximation by simpson three eight rule`

```
// example 5.13
// caption: simpson 3-8 rule


// let integration of f(x)=1/(1+x) in the range
   [0,1] by simpson 3-8 rule is equal to I

x=0:1/3:1;
deff('[y]=f(x)','y=1/(1+x)')

```

```
[I] = simpson38(x,f)
```

Scilab code Exa 5.15 `quadrature formula`

```
// example :5.15
// find the quadrature formula of
// integral of f(x)*(1/sqrt(x(1+x))) in the range [0,1]= a1*f(0)+a2*f(1/2)+a3*f(1)=I
// hence find integral 1/sqrt(x-x^3) in the range [0,1]

// making the method exact for polinomials of degree upto 2,
// I=I1=a1+a2+a3
// I=I2=(1/2)*a2+a3
// I=I3=(1/4)*a2+a3

// A=[a1 a2 a3]'

I1=integrate('1/sqrt(x*(1-x))','x',0,1)
I2=integrate('x/sqrt(x*(1-x))','x',0,1)
I3=integrate('x^2/sqrt(x*(1-x))','x',0,1)

//hence
//  [1 1 1;0 1/2 1 ;0 1/4 1]*A=[I1 I2 I3]'

A=inv([1 1 1;0 1/2 1 ;0 1/4 1])*[I1 I2 I3]'
// I=(3.14/4)*(f(0)+2*f(1/2)+f(1));

// hence,   for solving the integral 1/sqrt(x-x^3) in the range [0,1]=I

deff('[y]=f(x)','y=1/sqrt(1+x)');
I=(3.14/4)*[1+2*sqrt(2/3)+sqrt(2)/2]
```

Scilab code Exa 5.16 `gauss legendary three point method`

```
// example 5.16
// caption: gauss-legendre three point method
// I= integral 1/(1+x) in the range [0,1];
// first we need ti transform the interval [0,1 ] to
    [-1,1], since gauss-legendre three point method
   is applicable in the range[-1,1],

// let t=ax+b;
// solving for a,b from the two ranges, we get a=2;
   b=-1; t=2x-1;

// hence I=integral 1/(1+x) in the range [0,1]=
   integral 1/(t+3) in the range [-1,1];


deff('[y]=f(t)','y=1/(t+3)');
// since , from gauss legendre three point rule(n=2)
   ;
I=(1/9)*(5*f(-sqrt(3/5))+8*f(0)+5*f(sqrt(3/5)))

// we know , exact solution is ln 2=0.693147;
```

Scilab code Exa 5.17 `gauss legendary method`

```
// example 5.17
// caption: gauss-legendre method
// I= integral 2*x/(1+x^4) in the range [1,2];
// first we need ti transform the interval [1,2 ] to
    [-1,1], since gauss-legendre three point method
   is applicable in the range[-1,1],
```

```

// let t=ax+b;
// solving for a,b from the two ranges, we get a =1/2; b=3/2; x=(t+3)/2;

// hence I=integral 2*x/(1+x^4) in the range [0,1]= integral 8*(t+3)/16+(t+3)^4 in the range [-1,1];


deff('[y]=f(t)','y=8*(t+3)/(16+(t+3)^4) ');

// 1) since , from gauss legendre one point rule;
I1=2*f(0)

// 2) since , from gauss legendre two point rule;
I2=f(-1/sqrt(3))+f(1/sqrt(3))

// 3) since , from gauss legendre three point rule;
I=(1/9)*(5*f(-sqrt(3/5))+8*f(0)+5*f(sqrt(3/5)))


// we know , exact solution is 0.5404;
```

Scilab code Exa 5.18 integral approximation by gauss chebishev

```
// example 5.18
// caption: gauss-chebyshev method

// we write the integral as I=integral f(x)/sqrt(1-x^2) in the range [-1,1];
// where f(x)=(1-x^2)^2*cos(x)

deff('[y]=f(x)','y=(1-x^2)^2*cos(x)');

// 1) since , from gauss chebyshev one point rule;
```

```
I1=(3.14)*f(0)

// 2) since , from gauss chebyshev two point rule;
I2=(3.14/2)*f(-1/sqrt(2))+f(1/sqrt(2))

// 3) since , from gauss chebyshev three point rule;
I=(3.14/3)*(f(-sqrt(3)/2)+f(0)+f(sqrt(3)/2))


// and 4) since , from gauss legendre three point
    rule;
I=(1/9)*(5*f(-sqrt(3/5))+8*f(0)+5*f(sqrt(3/5)))
```

Scilab code Exa 5.20 `integral approximation by gauss legurre method`

```
// example 5.20
// caption: gauss-leguerre method
// I= integral e^-x/(1+x^2) in the range [0,~];

// observing the integral we can inffer that f(x)
    =1/(1+x^2)

deff('[y]=f(x)','y=1/(1+x^2) ');


// 1) since , from gauss leguerre two point rule;
I2=(1/4)*[(2+sqrt(2))*f(2-sqrt(2))+(2-sqrt(2))*f(2+
    sqrt(2))]

// 3) since , from gauss leguerre  three point rule;
I=(0.71109*f(0.41577)+0.27852*f(2.29428)+0.01039*f
    (6.28995))
```

Scilab code Exa 5.21 `integral approximation by gauss legurre method`

```
// example 5.21
// caption: gauss-leguerre method
// I= integral e^-x*(3*x^3-5*x+1) in the range
    [0,~];

// observing the integral we can inffer that f(x)
    =(3*x^3-5*x+1)

deff('[y]=f(x)','y=(3*x^3-5*x+1) ');


// 1) since , from gauss leguerre two point rule;
I2=(1/4)*[(2+sqrt(2))*f(2-sqrt(2))+(2-sqrt(2))*f(2+
    sqrt(2))]

// 3) since , from gauss leguerre  three point rule;
I3=(0.71109*f(0.41577)+0.27852*f(2.29428)+0.01039*f
    (6.28995))
```

Scilab code Exa 5.22 `integral approximation by gauss legurre method`

```
// example 5.22
// caption: gauss-leguerre method
// I= integral 1/(x^2+2*x+2) in the range [0,~];

// since in the gauss-leguerre method the integral
    would be of the form e^x*f(x);

// observing the integral we can inffer that f(x)=%e
    ^x/(x^2+2*x+2)
deff('[y]=f(x)','y=%e^x/(x^2+2*x+2) ');


```

```
// 1) since , from gauss leguerre two point rule;
I2=(1/4)*[(2+sqrt(2))*f(2-sqrt(2))+(2-sqrt(2))*f(2+
    sqrt(2))]

// 3) since , from gauss leguerre  three point rule;
I=(0.71109*f(0.41577)+0.27852*f(2.29428)+0.01039*f
    (6.28995))


// the exact solution is given by,

I=integrate('1/((x+1)^2+1)','x',0,1000)      // 1000
    ~infinite;
```

check Appendix AP 15 for dependency:

`comp_trapezoidal.sci`

check Appendix AP 16 for dependency:

`simpson13.sci`

Scilab code Exa 5.26 `composite trapizoidal and composite simpson`

```
// Example 5.26
// caption: 1) composite trapizoidal rule, 2)
    composite simpsons rule with 2,4 ,8 equal sub-
    intervals,

// I=integral 1/(1+x) in the range [0,1]

deff('[y]=f(x)','y=1/(1+x)')

// when N=2;
// 1)composite trapizoidal rule
h=1/2;
x=0:h:1;
```

```
 IT=comptrapezoidal(x,h,f)

 // 2)composite simpsons rule

  [I] = simpson13(x,h,f)


  // when N=4
 // 1)composite trapizoidal rule
h=1/4;
x=0:h:1;

 IT=comptrapezoidal(x,h,f)

 // 2)composite simpsons rule

  [I] = simpson13(x,h,f)



    // when N=8
 // 1)composite trapizoidal rule
h=1/8;
x=0:h:1;

 IT=comptrapezoidal(x,h,f)

 // 2)composite simpsons rule

  [I] = simpson13(x,h,f)
```

Scilab code Exa 5.27 `integral approximation by gauss legurre method`

```
// example 5.27
```

```
// caption: gauss-legendre three point method
// I= integral 1/(1+x) in the range [0,1];

// we are asked to subdivide the range into two,
// first we need to sub-divide the interval [0,1 ]
   to [0,1/2] and [1/2,1] and then transform both to
    [-1,1], since gauss-legendre three point method
   is applicable in the range[-1,1],

// t=4x-1 and y=4x-3;

// hence I=integral 1/(1+x) in the range [0,1]=
   integral 1/(t+5) in the range [-1,1]+ integral
   1/(t+7) in the range [-1,1]


deff('[y1]=f1(t)','y1=1/(t+5)');
// since , from gauss legendre three point rule(n=2)
   ;
I1=(1/9)*(5*f1(-sqrt(3/5))+8*f1(0)+5*f1(sqrt(3/5)))

deff('[y2]=f2(t)','y2=1/(t+7)');
// since , from gauss legendre three point rule(n=2)
   ;
I2=(1/9)*(5*f2(-sqrt(3/5))+8*f2(0)+5*f2(sqrt(3/5)))

I=I1+I2

// we know , exact solution is .693147;
```

Scilab code Exa 5.29 double integral using simpson rule

```
// example 5.29
// evaluate the given double integral using the
   simpsons rule;
```

```
// I= double integral f(x)=1/(x+y) in the range x
    =[1,2],y=[1,1.5];

h=.5;
k=.25;
deff('[w]=f(x,y)','w=1/(x+y)')

I=(.125/9)*[{f(1,1)+f(2,1)+f(1,1.5)+f(2,1.5)}+4*{f
    (1.5,1)+f(1,1.25)+f(1.5,1.5)+f(2,1.25)}+16*f
    (1.5,1.25)];
disp(I);
```

Scilab code Exa 5.30 double integral using simpson rule

```
// example 5.30
// evaluate the given double integral using the
    simpsons rule;

// I= double integral f(x)=1/(x+y) in the range x
    =[1,2],y=[1,2];
// 1)
h=.5;
k=.5;
deff('[w]=f(x,y)','w=1/(x+y)')

I=(1/16)*[{f(1,1)+f(2,1)+f(1,2)+f(2,2)}+2*{f(1.5,1)+
    f(1,1.5)+f(2,1.5)+f(1.5,2)}+4*f(1.5,1.5)]

// 2)
h=.25;
k=.25;
deff('[w]=f(x,y)','w=1/(x+y)')

I=(1/64)*[{f(1,1)+f(2,1)+f(1,2)+f(2,2)}+2*{f(5/4,1)+
```

```
f(3/2,1)+f(7/4,1)+f(1,5/4)+f(1,3/2)+f(1,7/4)+f
(2,5/4)+f(2,3/4)+f(2,7/4)+f(5/4,2)+f(3/2,2)+f
(7/4,2)}+4*{f(5/4,5/4)+f(5/4,3/2)+f(5/4,7/4)+f
(3/2,5/4)+f(3/2,3/2)+f(3/2,7/4)+f(7/4,5/4)+f
(7/4,3/2)+f(7/4,7/4)}]
```

Chapter 6

ORDINARY DIFFERENTIAL EQUATIONS INNITIAL VALUE PROBLEMS

check Appendix AP 3 for dependency:

`eigenvectors.sci`

Scilab code Exa 6.3 `solution to the system of equations`

```
// example 6.3
// solution to the given IVP

disp('du/dt= A*u');
// u=[u1 u2]';
A=[-3 4 ;-2 3];                                  // given
B=[1 0;0 1];                                      // identity
   matrix;




[x,lam] = geigenvectors(A,B);
```

```

// hence;
disp('u=c1*%e^t*x(:,1)+c2*%e^-t*x(:,2)');
disp('u1=c1*%e^t+c2*%e^-t*2')
disp('u2=c1*%e^t+c2*%e^-t')
```

check Appendix AP 3 for dependency:

`eigenvectors.sci`

Scilab code Exa 6.4 `solution ti the IVP`

```
// example 6.4
// solution to the given IVP

disp('du/dt= A*u');
// u=[u1 u2]';
A=[-2 1;1 -20];                         // given
B=[1 0;0 1];                            // identity matrix;





[x,lam] = geigenvectors(A,B);

// hence;
disp('u=c1*%e^(lam(1)*t)*x(:,1)+c2*%e^-(lam(2)*t)*x(:,2)');
```

check Appendix AP 2 for dependency:

`Euler1.sce`

Scilab code Exa 6.9 `euler method to solve the IVP`

```
// example 6.9
// solve the IVP by euler method,
// with h=0.2, 0.1, 0.05;
// u'=f(t,u)
// u'=-2tu^2
deff('[z]=f(t,u)','z=-2*t*u^2');


[u,t] = Euler1(1,0,1,.2,f)       // h=0.2;

[u,t] = Euler1(1,0,1,0.1,f)       // h=0.1;


[u,t] = Euler1(1,0,1,0.05,f)       // h=0.05;
```

check Appendix AP 14 for dependency:

`backeuler.sci`

Scilab code Exa 6.12 `solution ti IVP by back euler method`

```
// example 6.12,
// caption: solve the IVP by backward euler method,
// with h=0.2,
// u'=f(t,u)
deff('[z]=f(t,u)','z=-2*t*u^2');


[u] = backeuler(1,0,0.4,.2,f)      // h=0.2;
```

check Appendix AP 13 for dependency:

`eulermidpoint.sci`

Scilab code Exa 6.13 `solution ti IVP by euler mid point method`

```
// example 6.13,
// caption: solve the IVP by euler midpoint method,
// with h=0.2,
// u'=f(t,u)
deff('[z]=f(t,u)','z=-2*t*u^2');
deff('[w]=fp(t,u)','w=-2*u^2-4*u*t');



[u] =  eulermidpoint(1,0,1,.2,f,fp)    // h=0.2;
```

check Appendix AP 12 for dependency:

`fact.sci`

check Appendix AP 11 for dependency:

`taylor.sci`

Scilab code Exa 6.15 `solution ti IVP by taylor expansion`

```
// example 6.15
//caption: solving ODE by tailor series method
// u'=t^2+ u^2,  u0=0;

t=0;U=0;                    //at t=0, the value of u
   is 0
U1=0;                    // u1 is the 1st derivatove
   of the funtion u
U2=2*t+2*U*U1                // U2 ---- 2nd
   derivative
```

```
U3=2+2*(U*U2+U1^2)
U4=2*(U*U3+3*U1*U2)
U5=2*(U*U4+4*U1*U3+3*U2^2)
U6=2*(U*U5+5*U1*U4+10*U2*U3)
U7=2*(U*U6+6*U1*U5+15*U2*U4+10*U3^2)
U8=0;
U9=0;
U10=0;
U11=2*(U*U10+10*U1*U9+45*U2*U8+120*U3*U7+210*U4*U6
    +126*U5^2)
                                        // U11 is the 11th
                                           derivative of u


taylor(1)
```

check Appendix AP 10 for dependency:

`heun.sci`

check Appendix AP 9 for dependency:

`modifiedeuler.sci`

Scilab code Exa 6.17 `solution ti IVP by modified euler cauchy and heun`

```
// example 6.17,
// caption: solve by 1)modified euler cauchy, 2)heun
    method

// h=0.2
// 1) modified euler cauchy method,

// u'=f(t,u)
// u'=-2tu^2
deff('[z]=f(t,u)','z=-2*t*u^2');

```

```
modifiedeuler(1,0,.4,.2,f)        // calling the
    function,

// 2) heun method,
deff('[z]=f(t,u)','z=-2*t*u^2');


heun(1,0,.4,.2,f)        // calling the function,
```

check Appendix AP 8 for dependency:

`RK4.sci`

Scilab code Exa 6.18 `solution ti IVP by fourth order range kutta method`

```
// example 6.18,
// caption: use of 4th order runge kutta method,

// u'=f(t,u)
// u'=-2tu^2
deff('[z]=f(t,u)','z=-2*t*u^2');

RK4(1,0,.4,.2,f)        // calling the function,
```

check Appendix AP 6 for dependency:

`Vsim_eulercauchy.sce`

check Appendix AP 7 for dependency:

`simRK4.sci`

Scilab code Exa 6.20 `solution to the IVP systems`

```
// example no. 6.20,
// caption: solve the system of equations

//1) eulercauchy  method solving simultanious ODE

deff('[z]=f1(t,u,v)','z=-3*u+2*v');
deff('[w]=f2(t,u,v)','w=3*u-4*v');


[u,v,t] = simeulercauchy(0,.5,0,.4,.2,f1,f2)

// 2)  RK4 method solving simultanious ODE



[u,v,t]=simRK4(0,.5,0,.4,.2,f1,f2)
```

check Appendix AP 5 for dependency:

`newtonrap.sce`

Scilab code Exa 6.21 solution ti IVP by second order range kutta method

```
// example  no. 6.21,
// caption: solving the IVP by implicit RK2 method

// u'=f(t,u)
// u'=-2tu^2
//u(0)=1,h=0.2;
t0=0;h=0.2;tn=.4;u0=1;
deff('[z]=f(t,u)','z=-2*t*u^2');
umaxAllowed = 1e+100;

t = [t0:h:tn]; u = zeros(t); n = length(u); u(1) =
    u0;

```

```
for j = 1:n-1
   // k1=h*f(t(j)+h/2,u(j)+k1/2);
    // conidering the IVP we can infer that the
       above expression in non linear in k1,
// hence we use newton rapson method to solve for k1
   ;
deff('[w]=F(u2)','w=k1+h*(2*t(j)+h)*(u(j)+k1/2)^2')
      // u2=u(2)
deff('[x]=Fp(u2)','x=1+h*(2*t(j)+h)*(u(j)+k1/2)')

k1=h*f(t(j),u(j));

newton(k1,F,Fp);
   u(j+1) = u(j) +k1
   disp(u(j+1))

end;
```

check Appendix AP 4 for dependency:

`adamsbashforth3.sci`

Scilab code Exa 6.25 `solution ti IVP by third order adamsbashfort meth`

```
// example 6.25
// caption: solving the IVP by adams-bashforth 3rd
   order method.
// u'=f(t,u)
// u'=-2tu^2
//u(0)=1,h=0.2;
deff('[z]=f(t,u)','z=-2*t*u^2');

adamsbashforth3(1,0,1,.2,f)      // calling the
   function,
```

Scilab code Exa 6.27 solution ti IVP by third order adams moult method

```
// example 6.27
// solving IVP by 3rd order adams moulton
// u'=t^2+u^2,    u(1)=2,
// h=0.1,        [1,1.2]
deff('[z]=f(t,u)','z=t^2+u^2');
t0=1;u0=2;h=0.1;tn=1.2;
// third order adams moulton method,
// u(j+2)=u(j+1)+(h/12)*(5*f(t(j+2),u(j+2))+8*f(t(j
   +1),u(j+1))-f(t(j),u(j)));-  is the expression
   for adamsbas-moulton3


// on observing the IVP we can  inffer that this
   would be a non linear equation,
// u(j+2)=u(j+1)+(h/12)*(5*((t(j+2))^2+(u(j+2))^2)
   +8*((t(j+1))^2+(u(j+1))^2)-((t(j))^2+(u(j))^2))

t = [t0:h:tn]; u = zeros(t); n = length(u); u(1) =
   u0;
for j = 1:n-2
    if j==1 then
         k1=h*f(t(j),u(j));
    k2=h*f(t(j)+h,u(j)+k1);
    u(j+1) = u(j) + (k2+k1)/2;
    disp(u(j+1))
    end;
end;

// hence the third order adams moulton expression
   turns to be,
// u(2)= 0.041667*(u(2))^2+3.194629
// let us use newton raphsom method to solve this,
```

```
deff('[w]=F(u2)','w=-u2+ 0.041667*(u2)^2+3.194629')      // u2=u(2)
deff('[x]=Fp(u2)','x=-1+ 0.041667*2*u2')

// let us assume the initial guess of u(2)=u(1);

newton(2.633333,F,Fp)
```

Scilab code Exa 6.32 solution by numerov method

```
// example 6.32
// caption: solving the IVP by numerov method
// u''=(1+t^2)*u
// u(0)=1, u'(0)=0  ,[0,1]
// h=0.2,

// expression for numerov method is
//u(j+1)-2*u(j)+u(j-1)=(h^2/12)*(u''(j+1)+10*u''(j)+u''(j-1));

// observing the IVP we can reduce the  numerov method to
//u(2)=2*u(1)-u(0)+(.2^2/12)*(1.16*u(2)+10.4*u(1)+1);      for j=1
// u(3)=2*u(2)-u(1)+(.2^2/12)*(1.36*u(3)+11.6*u(2)+1.04*u(1));      for j=2
// u(4)=2*u(3)-u(2)+(.2^2/12)*(1.64*u(4)+13.6*u(3)+1.16*u(2));      for j=3
// u(5)=2*u(4)-u(3)+(.2^2/12)*(2*u(5)+16.4*u(4)+1.36*u(3));      for j=4

// from taylor series expansion we observe that
u1=1.0202;u0=1;
//u2-(.2^2/12)*(1.16*u2)=2*u1-u0+(.2^2/12)*(10.4*u1+1);
```

```
u2=(1/.9961333)*2*u1-u0+(.2^2/12)*(10.4*u1+1)

u3=(1/.995467)*(2.038667*u2-.996533*u1)

u4=(1/.994533)*(2.045333*u3-.996133*u2)

u5=(1/.993333)*(2.054667*u4-.995467*u3)
```

Chapter 7

ORDINARY DIFFERENTIAL EQUATIONS BOUNDARY VALUE PROBLEM

check Appendix AP 1 for dependency:

shooting.sci

Scilab code Exa 7.1 solution to the BVP by shooting method

```
// example 7.1
// solve by shooting method;

//    u''=u+1;
// u(0)=0;   u(1)=%e-1;

// let ->    U1(x)=du/dx;
// U2(x)=d2u/dx2;

// U(x)=[U1(x);U2(x)]

// hence ;
// dU/dx=f(x,U);
```

```
deff('[w]=f(x,U)','w=[U(2); U(1)+1]')

h=0.25;
x=[0:h:1];
ub=[0,%e-1];
up=[0:1:10];


[U] = shooting(ub,up,x,f);

// the solution obtained would show the values of u
   and their derivatives at various x taken in
   regular intervals of h;
```

check Appendix AP 1 for dependency:

`shooting.sci`

Scilab code Exa 7.3 `solution to the BVP by shooting method`

```
// example 7.3
// solve by shooting method;

//   u''=2*u*u';
// u(0)=0.5;  u(1)=1;

// let ->   U1(x)=du/dx;
// U2(x)=d2u/dx2;

// U(x)=[U1(x);U2(x)]

// hence ;
// dU/dx=f(x,U);
```

```
h=.25;

ub=[.5,1];

up=[0:.1:1];

x=0:h:1;

deff('[w]=f(x,U)','w=[U(2); 2*U(1)*U(2)]')



[U] = shooting(ub,up,x,f);

// the solution obtained would show the values of u
   in the first collumn and their corresponding
   derivatives in the second collumn ;
```

check Appendix AP 1 for dependency:

`shooting.sci`

Scilab code Exa 7.4 `solution to the BVP by shooting method`

```
// example 7.4
// solve by shooting method;

//    u''=2*u*u';
// u(0)=0.5;  u(1)=1;

// let ->    U1(x)=du/dx;
// U2(x)=d2u/dx2;

// U(x)=[U1(x);U2(x)]

```

```
// hence ;
// dU/dx=f(x,U);

h=.25;

ub=[.5,1];

up=[0:.1:1];

x=0:h:1;

deff('[w]=f(x,U)','w=[U(2); 2*U(1)*U(2)]')


[U] = shooting(ub,up,x,f);

// the solution obtained would show the values of u
   in the first collumn and their corresponding
   derivatives in the second collumn ;
```

Scilab code Exa 7.5 solution to the BVP

```
// example 7.5
// solve the boundary value problem        u''=u+x;
// u(x=0)=u(0)=0;    u(x=1)=u(4)=0;              h=1/4;


// we know;       u''=(u(j-1)-2*u(j)+u(j+1))/h^2;

// 1) second order method;
 x=0:1/4:1;
 u0=0;
 u4=0;
u1_3 = rand(1,3)
 u=[u0 u1_3 u4];
```

```
// hence;
disp('(u(j-1)-2*u(j)+u(j+1))/h^2=u(j)+x(j)')
                // for j=1,2,3;

disp('for j=1             -16*u0+33*u1-16*u2=-.25')

disp('for j=2             -16*u1+33*u2-16*u3=-.50')

disp('for j=3             -16*u2+33*u3-16*u4=-.75')

// hence solving for u1,u2,u3)    ,
u1=-.034885;
u2=-.056326;
u3=-.050037;

disp(x);
disp(u);

// 2) numerov method;
 x=0:1/4:1;
 u0=0;
 u4=0;
 u=[u0 u1 u2 u3 u4];
// since according to numerov method we get the
   following system of equations;
disp('(191*u(j-1)-394*u(j)+191*u(j+1)=x(j-1)+10*x(j)
   +x(j+1)')             // for j=1,2,3;

disp('for j=1             191*u0-394*u1+191*u2=3')

disp('for j=2             191*u1-394*u2+191*u3=6')

disp('for j=3             191*u2-394*u3+191*u4=9')

// hence solving for u1,u2,u3    ,
u1=-.034885
u2=-.056326
u3=-.050037
```

```


disp(x);
disp(u);
```

Scilab code Exa 7.6 solution to the BVP by finite differences

```
// example 7.6
// solve the boundary value problem      u''=u*x;
// u(0)+u'(0)=1;    u(x=1)=0;          h=1/3;


// we know;      u''=(u(j-1)-2*u(j)+u(j+1))/h^2;

// 1) second order method;
 x=0:1/3:1;
 u1_2 = rand(1,3)
 u3=1;
 u=[u1_2 u3];
// hence;
disp('(u(j-1)-2*u(j)+u(j+1))/h^2=u(j)*x(j)')     // for j=0,1,2,3;

disp('for j=0              u1!-2*u0+u1=0')   // u1!=u(-1)

disp('for j=1              u0-2*u1+u2=(1/27)u1')

disp('for j=2              u1-2*u2+u3=(2/27)u2')

// we know;       u'=(u(j+1)-u(j-1))/2h
// hence eliminating u1!
// solving for u0,u1,u2,u3 ,
u0=-.9879518;
u1=-.3253012;
```

```
u2=-.3253012;

disp(x);
disp(u);
```

Scilab code Exa 7.11 `solution to the BVP by finite differences`

```
// example 7.11
// solve the boundary value problem       u''=u'+1;
// u(0)=1;    u(x=1)=2(%e-1);             h=1/3;


// we know;       u''=(u(j-1)-2*u(j)+u(j+1))/h^2;
// we know;        u'=(u(j+1)-u(j-1))/2h;

// 1) second order method;
 x=0:1/3:1;

 u= rand(1,4);;
// hence;
disp('(u(j-1)-2*u(j)+u(j+1))/h^2=((u(j+1)-u(j-1))/2h
   )+1')                // for j=1,2;


disp('for j=1                (7/6)*u0-2*u1+(5/6)*u2
   =(1/9)')

disp('for j=2                (7/6)*u1-2*u2+(5/6)*u3
   =(1/9)')


// hence eliminating u1!
// solving for u1,u2,
u0=1;
u3=2*(%e-1);
```

```
u1=1.454869;
u2=2.225019;

disp(x);
disp(u);
```

Appendix

Scilab code AP 1 `shooting method for solving BVP`

```
function [U] = shooting(ub,up,x,f)

//Shooting method for a second order
//boundary value problem
//ub = [u0 u1] -> boundary conditions
//x = a vector showing the range of x
//f = function defining ODE, i.e.,
//     du/dx = f(x,u), u = [u(1);u(2)].
//up = vector with range of du/dx at x=x0
//xuTable = table for interpolating derivatives
//uderiv  = derivative boundary condition

n  = length(up);
m  = length(x);
y1 = zeros(up);

for j = 1:n
    u0    = [ub(1);up(j)];
    uu    = ode(u0,x(1),x,f);
    u1(j) = uu(1,m);
end;

xuTable = [u1';up];
uderiv  = interpln(xuTable,ub(2));
u0      = [ub(1);uderiv];
u       = ode(u0,x(1),x,f);
```

```
U=u';

endfunction
```

Scilab code AP 2 `euler method`

```
function [u,t] = Euler1(u0,t0,tn,h,f)

//Euler 1st order method solving ODE
//  du/dt = f(u,t), with initial
//conditions u=u0 at t=t0.  The
//solution is obtained for t = [t0:h:tn]
//and returned in u

umaxAllowed = 1e+100;

t = [t0:h:tn]; u = zeros(t); n = length(u); u(1) =
   u0;

for j = 1:n-1
    u(j+1) = u(j) + h*f(t(j),u(j));
    if u(j+1) > umaxAllowed then
            disp('Euler 1 - WARNING: underflow or
               overflow');
        disp('Solution sought in the following range:
           ');
            disp([t0 h tn]);
        disp('Solution evaluated in the following
           range:');
        disp([t0 h t(j)]);
            n = j; t = t(1,1:n); u = u(1,1:n);
        break;
    end;
end;

endfunction
```

Scilab code AP 3 `eigen vectors and eigen values`

```
function [x,lam] = geigenvectors(A,B)

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

[nA,mA] = size(A);
[nB,mB] = size(B);

if (mA<>nA | mB<>nB) then
    error('geigenvectors - matrix A or B not square');
    abort;
end;

if nA<>nB then
    error('geigenvectors - matrix A and B have different dimensions');
    abort;
end;

lam = poly(0,'lam');          //Define variable "lam"
chPoly = det(A-B*lam);            //Characteristic polynomial
lam = roots(chPoly)';             //Eigenvalues of matrix A

x = []; n = nA;

for k = 1:n
    BB = A - lam(k)*B;       //Characteristic matrix
        CC = BB(1:n-1,1:n-1);    //Coeff. matrix for reduced system
    bb = -BB(1:n-1,n);       //RHS vector for reduced system
    y = CC\bb;       //Solution for reduced system
```

```
    y = [y;1];          //Complete eigenvector

    x = [x y];          //Add eigenvector to matrix
end;

endfunction
```

Scilab code AP 4 adams bashforth third order method

```
function [u,t] = adamsbashforth3(u0,t0,tn,h,f)

//adamsbashforth3 3rd order method solving ODE
//  du/dt = f(u,t), with initial
//conditions u=u0 at t=t0.  The
//solution is obtained for t = [t0:h:tn]
//and returned in u

umaxAllowed = 1e+100;

t = [t0:h:tn]; u = zeros(t); n = length(u); u(1) =
    u0;
for j = 1:n-2
if j<3 then
      k1=h*f(t(j),u(j));
    k2=h*f(t(j)+h,u(j)+k1);
    u(j+1) = u(j) + (k2+k1)/2;
end;

if j>=2 then
        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)));
end;
end;
endfunction
```

Scilab code AP 5 newton raphson method

```
function x=newton(x,f,fp)
    R=5;
    PE=10^-15;
    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 6 euler cauchy solution to the simultanioys equations

```
function [u,v,t] = simeulercauchy(u0,v0,t0,tn,h,f1,
   f2)


//  du/dt = f1(t,u,v), dv/dt = f2(t,u,v) with
   initial
//conditions u=u0,v=v0 at t=t0.  The
//solution is obtained for t = [t0:h:tn]
//and returned in u,v


umaxAllowed = 1e+100;

t = [t0:h:tn]; u = zeros(t);v= zeros(t); n = length(
   u); u(1) = u0;v(1)=v0;

for j = 1:n-1
```

```
    k11=h*f1(t(j),u(j),v(j));
    k21=h*f2(t(j),u(j),v(j));
    k12=h*f1(t(j)+h,u(j)+k11,v(j)+k21);
    k22=h*f2(t(j)+h,u(j)+k11,v(j)+k21);
    u(j+1) = u(j) + (k11+k12)/2;
    v(j+1) = v(j) + (k21+k22)/2;

end;

endfunction
```

Scilab code AP 7 simultaneous fourth order range kutta

```
function [u,v,t] = simRK4(u0,v0,t0,tn,h,f1,f2)

//  RK4 method solving simultanious ODE
//  du/dt = f1(t,u,v), dv/dt = f2(t,u,v) with
    initial
//conditions u=u0,v=v0 at t=t0.  The
//solution is obtained for t = [t0:h:tn]
//and returned in u,v

umaxAllowed = 1e+100;

t = [t0:h:tn]; u = zeros(t);v=zeros(t) ;n = length(u
    ); u(1) = u0;v(1)=v0

for j = 1:n-1
    k11=h*f1(t(j),u(j),v(j));
    k21=h*f2(t(j),u(j),v(j));
    k12=h*f1(t(j)+h/2,u(j)+k11/2,v(j)+k21/2);
    k22=h*f2(t(j)+h/2,u(j)+k11/2,v(j)+k21/2);
    k13=h*f1(t(j)+h/2,u(j)+k12/2,v(j)+k22/2);
    k23=h*f2(t(j)+h/2,u(j)+k12/2,v(j)+k22/2);
    k14=h*f1(t(j)+h,u(j)+k13,v(j)+k23);
    k24=h*f2(t(j)+h,u(j)+k13,v(j)+k23);
    u(j+1) = u(j) + (1/6)*(k11+2*k12+2*k13+k14);
    v(j+1) = v(j) + (1/6)*(k21+2*k22+2*k23+k24);
```

```
end;

endfunction
```

Scilab code AP 8 `fourth order range kutta method`

```
function [u,t] = RK4(u0,t0,tn,h,f)

//  RK4 method solving ODE
//  du/dt = f(u,t), with initial
//conditions u=u0 at t=t0.  The
//solution is obtained for t = [t0:h:tn]
//and returned in u

umaxAllowed = 1e+100;

t = [t0:h:tn]; u = zeros(t); n = length(u); u(1) =
   u0;

for j = 1:n-1
    k1=h*f(t(j),u(j));
    k2=h*f(t(j)+h/2,u(j)+k1/2);
    k3=h*f(t(j)+h/2,u(j)+k2/2);
    k4=h*f(t(j)+h,u(j)+k3);
    u(j+1) = u(j) + (1/6)*(k1+2*k2+2*k3+k4);
    if u(j+1) > umaxAllowed then
           disp('Euler 1 - WARNING: underflow or
              overflow');
        disp('Solution sought in the following range:
           ');
           disp([t0 h tn]);
        disp('Solution evaluated in the following
           range:');
        disp([t0 h t(j)]);
           n = j; t = t(1,1:n); u = u(1,1:n);
        break;
    end;
```

```
end;

endfunction
```

Scilab code AP 9 `modified euler method`

```
function [u,t] = modifiedeuler(u0,t0,tn,h,f)

//modifiedeuler 1st order method solving ODE
//  du/dt = f(u,t), with initial
//conditions u=u0 at t=t0.  The
//solution is obtained for t = [t0:h:tn]
//and returned in u

umaxAllowed = 1e+100;

t = [t0:h:tn]; u = zeros(t); n = length(u); u(1) =
   u0;

for j = 1:n-1
    k1=h*f(t(j),u(j));
    k2=h*f(t(j)+h/2,u(j)+k1/2);
    u(j+1) = u(j) + k2;
    if u(j+1) > umaxAllowed then
          disp('Euler 1 - WARNING: underflow or
             overflow');
       disp('Solution sought in the following range:
          ');
          disp([t0 h tn]);
       disp('Solution evaluated in the following
          range:');
       disp([t0 h t(j)]);
          n = j; t = t(1,1:n); u = u(1,1:n);
       break;
    end;
end;

endfunction
```

Scilab code AP 10 `euler cauchy or heun`

```
function [u,t] = heun(u0,t0,tn,h,f)

//heun method solving ODE
//  du/dt = f(u,t), with initial
//conditions u=u0 at t=t0.  The
//solution is obtained for t = [t0:h:tn]
//and returned in u

umaxAllowed = 1e+100;

t = [t0:h:tn]; u = zeros(t); n = length(u); u(1) = u0;

for j = 1:n-1
    k1=h*f(t(j),u(j));
    k2=h*f(t(j)+h,u(j)+k1);
    u(j+1) = u(j) + (k2+k1)/2;
    if u(j+1) > umaxAllowed then
            disp('Euler 1 - WARNING: underflow or overflow');
        disp('Solution sought in the following range:');
            disp([t0 h tn]);
        disp('Solution evaluated in the following range:');
        disp([t0 h t(j)]);
            n = j; t = t(1,1:n); u = u(1,1:n);
        break;
    end;
end;

endfunction
```

Scilab code AP 11 `taylor series`

```
function u=taylor(t)
  u=(t^1*U1)/fact(1)+(t^2*U2)/fact(2)+(t^3*U3)/fact
    (3)+(t^4*U4)/fact(4)+(t^5*U5)/fact(5)+(t^6*U6)
    /fact(6)+(t^7*U7)/fact(7)+(t^8*U8)/fact(8)+(t
    ^9*U9)/fact(9)+(t^10*U10)/fact(10)+(t^11*U11)/
    fact(11)
endfunction
```

Scilab code AP 12 factorial

```
function x=fact(n)
    x=1;
    for i=2:1:n
         x=x*i;
    end;
endfunction
```

Scilab code AP 13 mid point nmethod

```
function [u] =  eulermidpoint(u0,t0,tn,h,f,fp)

//midpoint 1st order method solving ODE
//  du/dt = f(u,t), with initial
//conditions u=u0 at t=t0.  The
//solution is obtained for t = [t0:h:tn]
//and returned in u

umaxAllowed = 1e+100;

t = [t0:h:tn]; u = zeros(t); n = length(u); u(1) =
   u0;
u(2)=u(1)+h*f(t(1),u(1))+(h^2/2)*fp(t(1),u(1));
for j = 2:n-1
    u(j+1) = u(j-1) + 2*h*f(t(j),u(j));
    if u(j+1) > umaxAllowed then
           disp('Euler 1 - WARNING: underflow or
              overflow');
```

```
        disp('Solution sought in the following range:
           ');
            disp([t0 h tn]);
        disp('Solution evaluated in the following
           range:');
        disp([t0 h t(j)]);
            n = j; t = t(1,1:n); u = u(1,1:n);
        break;
     end;
end;

endfunction
```

Scilab code AP 14 back euler method

```
function [u] = backeuler(u0,t0,tn,h,f)

//backeuler 1st order method solving ODE
//   du/dt = f(u,t), with initial
//conditions u=u0 at t=t0.  The
//solution is obtained for t = [t0:h:tn]
//and returned in u

umaxAllowed = 1e+100;

 t = [t0:h:tn]; u = zeros(t); n = length(u); u(1) =
    u0;

for j=1:n-1
    u(j+1)=u(j);
for i = 0:5
    u(j+1) = u(j) + h*f(t(j+1),u(j+1));
    i=i+1;
end;
end;


endfunction
```

Scilab code AP 15 `composite trapizoidal rule`

```
function I=comptrapezoidal(x,h,f)
    //This function calculates the numerical
        integration of f(x)dx
//between limits x(1) and x(n) using composite
    trapezoidal rule
//Check that x and y have the same size (which must
    be an odd number)
//Also, the values of x must be equally spaced with
    spacing h
y=feval(x,f);
[nrx,ncx]=size(x)
[nrf,ncf]=size(y)
if ((nrx<>1)|(nrf<>1)) then
    error('x or f, or both, not column vector(s)');
    abort;
end;
if ((ncx<>ncf)) then
    error('x and f are not of the same length');
    abort;
end;
//check that the size of the lists xL and f is odd
if (modulo(ncx,2)==0) then
    disp(ncx,"list size =")
    error('list size must be an odd number');
    abort
end;
n = ncx;

I = f(x(1)) + f(x(n));
for j = 2:n-1
    if(modulo(j,2)==0) then
        I = I + 2*f(x(j));
    else
        I = I + 2*f(x(j));
```

```
    end;
end;
I = (h/2.0)*I
endfunction
```

Scilab code AP 16 simpson rule

```
function [I] = simpson13(x,h,f)
//This function calculates the numerical integration
    of f(x)dx
//between limits x(1) and x(n) using Simpson's 1/3
    rule
//Check that x and y have the same size (which must
    be an odd number)
//Also, the values of x must be equally spaced with
    spacing h
y=feval(x,f);
[nrx,ncx]=size(x)
[nrf,ncf]=size(y)
if ((nrx<>1)|(nrf<>1)) then
    error('x or f, or both, not column vector(s)');
    abort;
end;
if ((ncx<>ncf)) then
    error('x and f are not of the same length');
    abort;
end;
//check that the size of the lists xL and f is odd
if (modulo(ncx,2)==0) then
    disp(ncx,"list size =")
    error('list size must be an odd number');
    abort
end;
n = ncx;

I = f(x(1)) + f(x(n));
for j = 2:n-1
    if(modulo(j,2)==0) then
```

```
        I = I + 4*f(x(j));
    else
        I = I + 2*f(x(j));
    end;
end;
I = (h/3.0)*I
endfunction
```

Scilab code AP 17 `simpson rule`

```
function [I] = simpson38(x,f)
//This function calculates the numerical integration of f(x)dx
//between limits x(1) and x(n) using Simpson's 3/8 rule
//Check that x and f have the same size (which must be of the form 3*i+1,
//where i is an integer number)
//Also, the values of x must be equally spaced with spacing h

y=feval(x,f);
[nrx,ncx]=size(x)
[nrf,ncf]=size(y)
if ((nrx<>1)|(nrf<>1)) then
    error('x or f, or both, not column vector(s)');
    abort;
end;
if ((ncx<>ncf)) then
    error('x and f are not of the same length');
    abort;
end;
//check that the size of the lists xL and f is odd
if (modulo(ncx-1,3)<>0) then
    disp(ncx,"list size =")
    error('list size must be of the form 3*i+1, where i=integer');
    abort
```

```
end;
n = ncx;
xdiff =  mtlb_diff(x);
h = xdiff(1,1);
I = f(x(1)) + f(x(n));
for j = 2:n-1
    if(modulo(j-1,3)==0) then
        I = I + 2*f(x(j));
    else
        I = I + 3*f(x(j));
    end;
end;
I = (3.0/8.0)*h*I
endfunction
```

Scilab code AP 18 approximation to the integral by simpson method

```
function I=simpson(a,b,f)
    I=((b-a)/6)*(f(a)+4*f((a+b)/2)+f(b));
endfunction
```

Scilab code AP 19 integration by trapizoidal method

```
// solves the definite integral by the trapezoidal rule,
// given the limits a,b and the function f,
// returns the integral value I

function I =trapezoidal(a,b,f)
    I=((b-a)/2)*(f(a)+f(b));
endfunction
```

Scilab code AP 20 jacobian of a given matrix

```
function J= jacobianmat (f1,f2,h,k)
    J=zeros(2,2);
J(1,1)=(f1(1+h,1)-f1(1,1))/2*h;
```

```
J(1,2)=(f1(1,1+k)-f1(1,1))/2*k;
J(2,1)=(f2(1+h,1)-f2(1,1))/2*h;
J(2,2)=(f2(1,1+k)-f2(1,1))/2*k;
endfunction
```

Scilab code AP 21 linear interpolating polinomial

```
function fp=linearinterpol(xL,f)
    fp=(f(2)-f(1))/(xL(2)-xL(1));
endfunction;
```

Scilab code AP 22 iterated interpolation

```
function [L012,L02,L01]=iteratedinterpol (x,f,n)
    X=poly(0,"X");
    L01=(1/(x(2)-x(1)))*det([f(1) x(1)-X;f(2) x(2)-X
       ]);
    L02=(1/(x(3)-x(1)))*det([f(1) x(1)-X;f(3) x(3)-X
       ]);
    L012=(1/(x(3)-x(2)))*det([L01 x(2)-X;L02 x(3)-X
       ]);

endfunction
```

Scilab code AP 23 newton divided differences interpolation order two

```
function P2=NDDinterpol2 (x,f)
    X=poly(0,"X");
    f01=(f(2)-f(1))/(x(2)-x(1));
    f13=(f(3)-f(2))/(x(3)-x(2));
    f013=(f13-f01)/(x(3)-x(1));
    P2=f(1)+(X-x(1))*f01+(X-x(1))*(X-x(2))*f013;
endfunction
```

Scilab code AP 24 legrange fundamental polynomial

```
function P2=lagrangefundamentalpoly(x,f,n)
    [nrx,ncx]=size(x)
    [nrf,ncf]=size(f)
if ((nrx<>1)|(nrf<>1)) then
    error('x or f, or both, not column vector(s)');
    abort;
end;
if ((ncx<>ncf)) then
    error('x and f are not of the same length');
    abort;
end;

 X=poly(0,"X");
L=zeros(n);

P2=0;
for i=1:n+1
    L(i)=1;
   for j=1:n+1
       if  i~=j then
            L(i)=L(i)*(X-x(j))/(x(i)-x(j))
       end;
end;
P2=P2+L(i)*f(i);
end;


endfunction
```

Scilab code AP 25 `legrange interpolation`

```
function P1=legrangeinterpol (x0,x1,f0,f1)
    x=poly(0,"x");
    L0=(x-x1)/(x0-x1);
    L1=(x-x0)/(x1-x0);
    P1=L0*f0+L1*f1;
endfunction
```

Scilab code AP 26 quadratic approximation

```
function [P]=quadraticapprox(x,f)

n=length(x);m=length(f);
if m<>n then
   error('linreg - Vectors x and f are not of the
      same length.');
   abort;
end;
s1=0;
s2=0;
for i=1:n
    s1=s1+x(i)*f(i);
    s2=s2+x(i)^2*f(i);
end
c0=det([sum(f) sum(x) sum(x^2);s1 sum(x^2) sum(x^3);
    s2 sum(x^3) sum(x^4)])/det([n sum(x) sum(x^2);
   sum(x) sum(x^2) sum(x^3); sum(x^2) sum(x^3) sum(x
   ^4)]);

c1=det([n sum(f) sum(x^2);sum(x) s1 sum(x^3); sum(x
   ^2) s2 sum(x^4)])/det([n sum(x) sum(x^2);sum(x)
   sum(x^2) sum(x^3); sum(x^2) sum(x^3) sum(x^4)]);

c2=det([n sum(x) sum(f);sum(x) sum(x^2) s1; sum(x^2)
    sum(x^3) s2])/det([n sum(x) sum(x^2);sum(x) sum(
   x^2) sum(x^3); sum(x^2) sum(x^3) sum(x^4)]);

X=poly(0,"X");
P=c2*X^2+c1*X+c0;
endfunction
```

Scilab code AP 27 straight line approximation

```
function [P]=straightlineapprox(x,f)

n=length(x);m=length(f);
```

```
if m<>n then
    error('linreg - Vectors x and f are not of the
       same length.');
    abort;
end;
s=0;
for i=1:n
     s=s+x(i)*f(i);
end
c0=det([sum(f) sum(x); s sum(x^2)])/det([n sum(x);
    sum(x) sum(x^2)]);
c1=det([ n sum(f); sum(x) s])/det([n sum(x);sum(x)
    sum(x^2)]);
X=poly(0,"X");
P=c1*X+c0;
endfunction
```

Scilab code AP 28 aitken interpolation

```
function P1=aitkeninterpol (x0,x1,f0,f1)
    x=poly(0,"x");
    P1=(1/(x1-x0))*det([f0 x0-x;f1 x1-x]);
endfunction
```

Scilab code AP 29 newton divided differences interpolation

```
function P1=NDDinterpol (x0,x1,f0,f1)
    x=poly(0,"x");
    f01=(f1-f0)/(x1-x0);
    P1=f0+(x-x0)*f01;
endfunction
```

Scilab code AP 30 hermite interpolation

```
function P= hermiteinterpol(x,f,fp)
    X=poly(0,"X");
     function f0=L0(X)
```

```

  f0=(X-x(2))*(X-x(3))/((x(1)-x(2))*(x(1)-x(3)))
   endfunction;
 a0=[1-2*(X-x(1))*numdiff(L0,x(1))];
L0=(X-x(2))*(X-x(3))/((x(1)-x(2))*(x(1)-x(3)));
   A0=a0*L0*L0;
   disp(A0)
    B0=(X-x(1))*L0^2;

     X=poly(0,"X");

   function f1=L1(X)

  f1=(X-x(1))*(X-x(3))/((x(2)-x(1))*(x(2)-x(3)))
   endfunction;
a1=[1-2*(X-x(2))*0];
L1=(X-x(1))*(X-x(3))/((x(2)-x(1))*(x(2)-x(3)));
A1=a1 *L1*L1;
disp(A1)
 B1=(X-x(2))*L1^2;
   function f2=L2(X)

  f2=(X-x(1))*(X-x(2))/((x(3)-x(1))*(x(3)-x(2)))
   endfunction;
a2=[1-2*(X-x(3))*numdiff(L2,x(3))];
L2=(X-x(1))*(X-x(2))/((x(3)-x(1))*(x(3)-x(2)));
A2=a2 *L2*L2;
disp(A2)
  B2=(X-x(3))*L2^2;



    P=A0*f(1)+A1*f(2)+A2*f(3)+B0*fp(1)+B1*fp(2)+B2*
       fp(3);
endfunction
```

Scilab code AP 31 `newton backward differences polinomial`

```
function [P]=NBDP(x,n,xL,f)
//This function calculates a Newton Forward-Difference Polynomial of
//order n, evaluated at x, using column vectors xL, f as the reference
//table.  The first value of xL and of f, represent, respectively,
//xo and fo in the equation for the polynomial.
[m,nc]=size(f)
//check that it is indeed a column vector
if (nc<>1) then
    error('f is not a column vector.');
    abort
end;
//check the difference order
if (n >= m) then
    disp(n,"n=");
    disp(m,"m=");
    error('n must be less than or equal to m-1');
    abort
end;
//
xo = xL(m,1);
delx = mtlb_diff(xL);
h = delx(1,1);
s = (x-xo)/h;
P = f(m,1);
delf = f;
disp(delf);
for i = 1:n
    delf = mtlb_diff(delf);
    [m,nc] = size(delf);
    disp(delf);
    P = P + Binomial(s+i-1,i)*delf(m,1)
end;
endfunction

function[C]=Binomial(s,i)
```

```
    C = 1.0;
    for k = 0:i-1
        C = C*(s-k);
    end;
    C = C/factorial(i)
endfunction
function[fact]=factorial(nn)
    fact = 1.0
    for k = nn:-1:1
        fact=fact*k
    end;
endfunction
```

Scilab code AP 32 `gauss jorden`

```
function [M]=jorden(A,b)
    M=[A b];
    [ra,ca]=size(A);
    [rb,cb]=size(b);
    n=ra;
    for p=1:1:n
        for k=(p+1):1:n
            if abs(M(k,p))>abs(M(p,p)) then
                M({p,k},:)=M({k,p},:);
            end
        end
        M(p,:)=M(p,:)/M(p,p);
        for i=1:1:p-1
            M(i,:)=M(i,:)-M(p,:)*(M(i,p)/M(p,p));
        end
         for i=p+1:1:n
            M(i,:)=M(i,:)-M(p,:)*(M(i,p)/M(p,p));
        end
    end
endfunction
```

Scilab code AP 33 `gauss elimination with pivoting`

```
function [x]=pivotgausselim(A,b)
    M=[A b];
    [ra,ca]=size(A);
    [rb,cb]=size(b);
    n=ra;
    for p=1:1:n
        for k=(p+1):1:n
            if abs(M(k,p))>abs(M(p,p)) then
                M({p,k},:)=M({k,p},:);
            end
        end
        for i=p+1:1:n
            m(i,p)=M(i,p)/M(p,p);
             M(i,:)=M(i,:)-M(p,:)*m(i,p);

        end
    end
    a=M(1:n,1:n);
    b=M(:,n+1);
    for i = n:-1:1
    sumj=0
    for j=n:-1:i+1
        sumj = sumj + a(i,j)*x(j);
    end;
    x(i)=(b(i)-sumj)/a(i,i);
    end
endfunction
```

Scilab code AP 34 gauss elimination

```
function [x] = gausselim(A,b)

//This function obtains the solution to the system
   of
//linear equations A*x = b, given the matrix of
   coefficients A
//and the right-hand side vector, b

```

```
[nA,mA] = size(A)
[nb,mb] = size(b)

if nA<>mA then
    error('gausselim - Matrix A must be square');
    abort;
elseif mA<>nb then
        error('gausselim - incompatible dimensions
           between A and b');
    abort;
end;

a = [A b];

//Forward elimination

n = nA;
for k=1:n-1
    for i=k+1:n
    for j=k+1:n+1
        a(i,j)=a(i,j)-a(k,j)*a(i,k)/a(k,k);
    end;
    end;
end;

//Backward substitution

x(n) = a(n,n+1)/a(n,n);

for i = n-1:-1:1
    sumk=0
    for k=i+1:n
        sumk=sumk+a(i,k)*x(k);
    end;
    x(i)=(a(i,n+1)-sumk)/a(i,i);
end;

endfunction
```

Scilab code AP 35 eigen vector and eigen value

```
function [x,lam] = geigenvectors(A,B)

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

[nA,mA] = size(A);
[nB,mB] = size(B);

if (mA<>nA | mB<>nB) then
    error('geigenvectors - matrix A or B not square'
       );
    abort;
end;

if nA<>nB then
    error('geigenvectors - matrix A and B have
       different dimensions');
    abort;
end;

lam = poly(0,'lam');           //Define variable "lam
   "
chPoly = det(A-B*lam);              //Characteristic
   polynomial
lam = roots(chPoly)';               //Eigenvalues of
   matrix A

x = []; n = nA;

for k = 1:n
    BB = A - lam(k)*B;        //Characteristic matrix
        CC = BB(1:n-1,1:n-1);     //Coeff. matrix for
```

```
        reduced system
    bb = -BB(1:n-1,n);         //RHS vector for reduced system
    y = CC\bb;         //Solution for reduced system
    y = [y;1];         //Complete eigenvector

    x = [x y];         //Add eigenvector to matrix
end;

endfunction
```

Scilab code AP 36 gauss siedel method

```
function [X]=gaussseidel(A,n,N,X,b)
    L=A;
    U=A;
    D=A;
    for i=1:1:n
        for j=1:1:n
            if j>i then L(i,j)=0;
                D(i,j)=0;
            end
            if i>j then U(i,j)=0;
                D(i,j)=0;
            end
            if i==j  then L(i,j)=0;
                U(i,j)=0;
            end
        end

    end
    for k=1:1:N
        X=(D+L)^-1*(-U*X+b);
        disp(X)
    end

endfunction
```

Scilab code AP 37 jacobi iteration method

```
function [X]=jacobiiteration(A,n,N,X,b)
    L=A;
    U=A;
    D=A;
    for i=1:1:n
        for j=1:1:n
            if j>i then L(i,j)=0;
                D(i,j)=0;
            end
            if i>j then U(i,j)=0;
                D(i,j)=0;
            end
            if i==j  then L(i,j)=0;
                U(i,j)=0;
            end
        end

    end
    for k=1:1:N
        X=-D^-1*(L+U)*X+D^-1*(b);
    end

endfunction
```

Scilab code AP 38 back substitution

```
function [x] = back(U,Z)

x=zeros(1,n);
for i = n:-1:1
    sumk=0
    for j=i+1:n
        sumk=sumk+U(i,j)*x(j);
    end;
    x(i)=(Z(i)-sumk)/U(i,i);
end;
```

```
endfunction
```

Scilab code AP 39 `cholesky method`

```
function L=cholesky (A,n)
    L=zeros(n,n);
    for k=1:1:n
        S=0;
        P=0;
        for j=1:1:k-1
            S=S+(L(k,j)^2);
            P=P+L(i,j)*L(k,j)
        end
        L(k,k)=sqrt(A(k,k)-S);
        for i=k+1:1:n
            L(i,k)=(A(i,k)-P)/L(k,k);
        end
    end

endfunction
```

Scilab code AP 40 `forward substitution`

```
function x=fore(L,b)

for i = 1:1:n
    sumk=0
    for j=1:i-1
        sumk=sumk+L(i,j)*x(j);
    end;
    x(i)=(b(i)-sumk)/L(i,i);
end;

endfunction
```

Scilab code AP 41 L and U matrices

```
function [U,L]=LandU(A,n)
    U=A
    L=eye(n,n)
    for p=1:1:n-1
        for i=p+1:1:n
            m=A(i,p)/A(p,p);
            L(i,p)=m;
            A(i,:)=A(i,:)-m*A(p,:);
            U=A;
        end
    end
endfunction
```

Scilab code AP 42 newton raphson method

```

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 43 four itterations of newton raphson method

```
function x=newton4(x,f,fp)
    R=4;
    PE=10^-15;
    maxval=10^4;
    for n=1:1:R
        if fp(x)==0 then disp("select another
           initial root x0")
        end
        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 44 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 45 `regula falsi method`

```
function [x]=regulafalsi(a,b,f)
    N=100;
    PE=10^-5;
    for n=2:1:N
        x=a-(a-b)*f(a)/(f(a)-f(b));
        if abs(f(x))<=PE then break;
        elseif (f(a)*f(x)<0) then b=x;
            else a=x;
        end
    end
    disp(n," no. of iterations =")
endfunction
```

Scilab code AP 46 `four iterations of regula falsi method`

```
function [x]=regulafalsi4(a,b,f)
    N=100;
    PE=10^-5;
    for n=2:1:N
        x=a-(a-b)*f(a)/(f(a)-f(b));
        if abs(f(x))<=PE then break;
        elseif (f(a)*f(x)<0) then b=x;
            else a=x;
        end
    end
    disp(n," no. of iterations =")
endfunction
```

Scilab code AP 47 `four iterations of secant method`

```
function [x]=secant4(a,b,f)
    N=4;                // 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 48 five itterations by bisection method

```
function x=bisection5(a,b,f)
    N=5;                                          //
       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
```

Scilab code AP 49 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
```

Scilab code AP 50 solution by newton method given in equation 2.63

```

function x=newton63(x,f,fp,fpp)
    R=100;
    PE=10^-15;
    maxval=10^4;

    for n=1:1:R
        x=x-(f(x)*fp(x))/(fp(x)^2-f(x)*fpp(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 51 solution by secant method given in equation 2.64

```
function [x]=secant64(a,b,f,fp)
    N=100;                  // define max. no. iterations to be performed
    PE=10^-15                // define tolerance for convergence
     for n=1:1:N          // initiating for loop
        x=(b*f(a)*fp(b)-a*f(b)*fp(a))/(f(a)*fp(b)-f(b)*fp(a));
        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 52 solution by secant method given in equation 2.65

```
function [x]=secant65(a,b,f)
    deff('[y]=g(x)','y=-f(x)^2/(f(x-f(x))-f(x))');
    N=4;                  // define max. no. iterations to be performed
    PE=10^-15                // define tolerance for convergence
     for n=1:1:N          // initiating for loop
```

```
        x=a-(b-a)*g(a)/(g(b)-g(a));
        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 53 solution to the equation having multiple roots

```

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

    for n=1:1:R
        x=x-m*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 54 solution by two iterations of general iteration

```

function x=generaliteration2(x,g,gp)
    R=2;
    PE=10^-8;
    maxval=10^4;
```

```
    A=[0 0];
    k=gp(x);
    if abs(k)>1 then error('function chosen does not
        converge')
        abort;
    end
    for n=1:1:R
        x=g(x);
        disp(x);
        if abs(g(x))<=PE then break
        end
        if (abs(g(x))>maxval) then error('Solution
            diverges');
            abort
            break
        end
    end
    disp(n," no. of iterations =")
endfunction
```

Scilab code AP 55 solution by aitken method

```
// 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 56 solution by general iteration

```

function x=generaliteration(x,g,gp)
    R=5;
```

```
    PE=10^-8;
    maxval=10^4;
    k=gp(x);
    if abs(k)>1 then error('function chosen does not
        converge')
        abort;
    end
    for n=1:1:R
        x=g(x);
        disp(x);
        if abs(g(x))<=PE then break
        end
        if (abs(g(x))>maxval) then error('Solution
           diverges');
            abort
            break
        end
    end
    disp(n," no. of iterations =")
endfunction
```

Scilab code AP 57 solution by multipoint iteration given in equation 33

```
function x=multipoint_iteration33(x,f,fp,R)
    R=3;
    PE=10^-5;
    maxval=10^4;
    for n=1:1:R
        x=x-f(x)/fp(x)-f(x-(f(x)/fp(x)))/fp(x);
        if abs(f(x))<=PE then break;
        end
        if (abs(f(x))>maxval) then error('Solution
           diverges');
            break
        end
     end
    disp(n," no. of iterations =")
endfunction
```

Scilab code AP 58 solution by multipoint iteration given in equation 31

```
function x=multipoint_iteration31(x,f,fp,R)
    R=3;
    PE=10^-5;
    maxval=10^4;
    for n=1:1:R
        x=x-f(x)/fp(x-(1/2)*(f(x)/fp(x)));
        if abs(f(x))<=PE then break;
        end
        if (abs(f(x))>maxval) then error('Solution
           diverges');
            break
        end
     end
    disp(n," no. of iterations =")
endfunction
```

Scilab code AP 59 solution by chebeshev method

```
function x=chebyshev(x,f,fp,fpp)
    R=100;
    PE=10^-5;
    maxval=10^4;
        if fp(x)==0 then disp("select another
           initial root x0");
            break;
        end
    for n=1:1:R
        x=x-f(x)/fp(x)-(1/2)*(f(x)/fp(x))^2 *(fpp(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 60 solution by five iterations of muller method

```
function x=muller5(x0,x1,x2,f)
    R=5;
    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 61 solution by three iterations of muller method

```
function x=muller3(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
```