NUMERICAL ANALYSIS WEEK I 2015

Each problem will be solved by using graphics, bisection and false position methods by hand, and then by computer program (java or Mathlab-Octave again by bisection and false position methods)

PROBLEM 1

Determine the real root of $f(x) = 4x^3 - 6x^2 + 7x - 2.3$:

(a) Graphically.

(b) Using bisection to locate the root. Employ initial guesses of $xl = 0$ and $xu = 1$ and iterate until the estimated error εa falls below a level of $\varepsilon s = 10\%$.

(c) Using false position method to locate the root. Employ initial guesses of $xl = 0$ and $xu = 1$ and iterate until the estimated error εa falls below a level of $\varepsilon s = 10\%$.

PROBLEM 2

Determine the real root of $f(x) = -26 + 85x - 91x2 + 44x3 - 8x4 + x5$:

(a) Graphically.

(b) Using bisection to determine the root Employ initial guesses of $xl = 0.5$ and $xu = 1.0$.

(c) Using falseposition n to determine the root Employ initial guesses of $xl = 0.5$ and $xu = 1.0$.

PROBLEM 3

The velocity v of a falling parachutist is given by

$$v = \frac{gm}{c}(1 - e^{-(c/m)t}$$

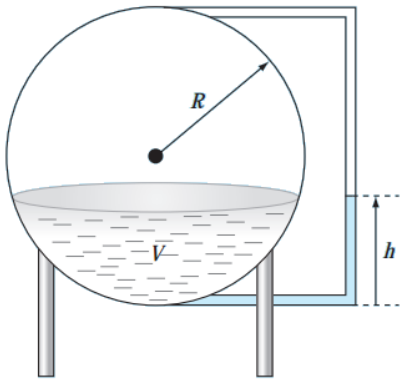
where $g = 9.8$ m/s2. For a parachutist with a drag coefficient $c = 15$ kg/s, compute the mass m so that the velocity is $v = 35$ m/s at $t = 9$ s. Use the false-position method to determine m to a level of $\varepsilon s = 0.1\%$.

PROBLEM 4

You are designing a spherical tank (Fig. P5.17) to hold water for a small village in a developing country. The volume of liquid itcan hold can be computed as

$$V = \pi h^2 \frac{[3R - h]}{3}$$

where V = volume [m3], h = depth of water in tank [m], and R is the tank radius [m]. If $R = 3$ m, to what depth must the tank be filled so that it holds 30 m^3? Use three iterations of the false-position method to determine your answer. Determine the approximate relative error after each iteration. Employ initial guesses of 0 and R.



Bisection method

```
 import static java.lang.Math.*;
import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;

public class NA1
{

public static double bisection(if_x f,double xl,double xu)
{
//bisection root finding method
double test;
double xr=0;
double es,ea;
double fxl,fxr,fxu;
int maxit=100,iter=0;
es=0.00000001;
ea=1.1*es;
String s="";
fxl= f.func(xl);
fxu= f.func(xu);

if(fxl*fxu>0) System.out.println("xl="+xu+"xr="+xr+"fxl="+fxl+"fxu="+fxu+"error");
while((ea>es)&&(iter<maxit))
{
   xr=(xl+xu)/2.0;
   iter++;
   if((xl+xu)!=0)
      { ea=Math.abs((xu-xl)/(xu+xl))*100;}
   fxl= f.func(xl);
   fxr= f.func(xr);
   s+="xr="+xr+"fxr="+fxr+"iter="+iter+"\n";
   test= fxl*fxr;
   if(test==0.0)      ea=0;
   else if(test<0.0)  xu=xr;
   else               {xl=xr;}
}
if(iter>=maxit) JOptionPane.showMessageDialog(null,"Maximum number of iteration is exceeded
\n"+
  " result might not be valid","MAKSİMUM NUMBER OF ITERATION
WARNING",JOptionPane.WARNING_MESSAGE);
//JOptionPane.showMessageDialog(null,s,
//     "bisection root finding method : ",JOptionPane.PLAIN_MESSAGE);
  return xr;
}
    public static void main (String args[]) throws IOException
    {
    double a,b;
    a=Double.parseDouble(JOptionPane.showInputDialog(" INPUT LOWER LİMİT OF SEARCH
AREA a : "));
    b=Double.parseDouble(JOptionPane.showInputDialog(" INPUT UPPER LIMIT OF SEARCH
```

```
AREA b : "));
    double r;
    if_x f1=x->x*x-2.0;
    r= bisection(f1,a,b);
    JOptionPane.showMessageDialog(null," ROOT  : "+r+"\nFUNCTION VALUE : "+f1.func(r),
    "bisection root finding method : ",JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```

```
import static java.lang.Math.*;
import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;
  // single function single independent variable
  // example f=x*x
  // includes full set of derivatives
  // Reference :"Generation of Finite Difference Formulas on Arbitrary Spaced Grids",
  // Bength Fornberg, Mathematics of Computation, Volume 51, Number 184, October 1988
  // pages 699-706
  //interface version
@FunctionalInterface
interface if_x
{
public double func(double x);

default void Plot(double a,double b)
{   Plot pp=new Plot(this,a,b);
            pp.plot();
}
default void vplot(double a,double b,int n)
{   vplot.plot(this,a,b,n);}
default void vplot(double a,double b)
{   vplot.plot(this,a,b,400);}
default void Plot(double a,double b,int n)
{   Plot pp=new Plot(this,a,b,n);
            pp.plot();
}
// nth order derivative

default double dfunc(double x,int N,int Mi,double hi)
{// order of the maximum derivative
 // N order of derivative
 // M degree of difference formula
double c[][][];
double alpha[];
double h;
int M;
double a[]=new double[0];
h=0.01;
double x0=0;
M=20;
double alphai[]={0,1,-1,2,-2,3,-3,4,-4,5,-5,6,-6,7,-7,8,-8,9,-9,10,-10,11,-11,12,-12,13,-13,14,-14,15,
```

```
-15,16,-16,17,-17,18,-18,19,-19,20,-20,21,-21,22,-22,23,-23,24,-24,25,-25,26,-26,27,-27,28,-28,29,-
29,30,-30,
31,-31,32,-32,33,-33,34,-34,35,-35,36,-36,37,-37,38,-38,39,-39,40,-40,41,-41,42,-42,43,-43,44,-
44,45,-45,46,
-46,47,-47,48,-48,49,-49,50,-50,51,-51,52,-52,53,-53,54,-54,55,-55,56,-56,57,-57,58,-58,59,-59,60,-
60,
-61,61,-62,62,-63,63,-64,64,-65,65,-66,66,-67,67,-68,68,-70,70,-71,71,-72,72,-73,73,-74,74,-75,75,
-76,76,-77,77,-78,78,-79,79,-80,80,-81,81,-82,82,-83,83,-84,84,-85,85,-86,86,-87,87,
-88,88,-89,89,-90,90,-91,91,-92,92,-93,93,-94,94,-95,95,-96,96,-97,97,-98,98,-99,99,-100,100};
alpha=alphai;
int N1=alpha.length-1;
// M   degree of highest derivative
// N+1 number of coefficients
double delta[][][]=new double[N1+1][N1+1][M+1];
double c1,c2,c3;
delta[0][0][0]=1.0;
c1=1.0;
for(int n=1;n<=N1;n++)
{  c2=1;
   for(int nu=0;nu<=(n-1);nu++)
   {c3=alpha[n]-alpha[nu];
    c2=c2*c3;
    if(n<=M) delta[n-1][nu][n]=0.0;
    for(int m=0;m<=Math.min(n,M);m++)
    {
       if(m==0)
       { delta[n][nu][m]=((alpha[n]-x0)*delta[n-1][nu][m])/c3;}
       else
       {delta[n][nu][m]=((alpha[n]-x0)*delta[n-1][nu][m]-m*delta[n-1][nu][m-1])/c3;}
    }//next m
   }//next nu
   for(int m=0;m<=Math.min(n,M);m++)
   {   if(m==0)
               {delta[n][n][m]=c1/c2*(-(alpha[n-1]-x0)*delta[n-1][n-1][m]);}
               else
               {delta[n][n][m]=c1/c2*(m*delta[n-1][n-1][m-1]-(alpha[n-1]-x0)*delta[n-1][n-1][m]);}
   }//next m
   c1=c2;
}//next n
c=delta;
  if(Mi<N) M=N;
  else M=Mi;
  h=hi;
  double deriv=0;
  double h1=1/h;
  double h2=1;
  for(int j=0;j<N;j++)
  {h2*=h1;}
  for(int i=0;i<c[0].length;i++)
  {   deriv+=c[M][i][N]*func(x+alpha[i]*h);}
  return deriv*h2;
}

default double dfunc(double x,int N)
{int M=30;
```

```
 double h=0.05*N;
 return dfunc(x,N,M,h);
}}
```

False position method

```
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
//============= Function to be define ================

class fb extends f_x
{
   public double func (double x)
   { return x*x-2;}
}
//============================================================
public class NA3
{

public static double false_position(f_x f,double xa,double xu)
{
//Modified False position root finding
double test;
double p=0;
double es,ea;
double fa,fp,fu;
int maxit=100,iter=0;
es=0.000001;
ea=1.1*es;
double xold;
int ia=0,iu=0;
fa=f.func(xa);
fu=f.func(xu);
p=xu;
double xpold;
while(iter<maxit && ea>es)
{
xpold=p;
if((iter%4)==0) p=(xa+xu)/2.0;
else p=xu-fu*(xa-xu)/(fa-fu);
fp=f.func(p);
iter++;
if(p!=0) ea=Math.abs((p-xpold)/p)*100.0;
test=fa*fp;
if(test<0) {xu=p;fu=f.func(xu);iu=0;ia++;if(ia>=2) fa/=2.0;}
else if(test>0) {xa=p;fa=f.func(xa);ia=0;iu+=1;if(iu>=2)fu/=2.0;}
else ea=0;
}
if(iter>=maxit) JOptionPane.showMessageDialog(null,"Maximum number of iteration is exceeded
\n"+
   " result might not be valid","MAKSİMUM NUMBER OF ITERATION
WARNING",JOptionPane.WARNING_MESSAGE);
```

```
return p;
}

    public static void main (String args[])
    {
   double a,b;
    a=Double.parseDouble(JOptionPane.showInputDialog(" INPUT LOWER LİMİT OF SEARCH
AREA a : "));
    b=Double.parseDouble(JOptionPane.showInputDialog(" INPUT UPPER LIMIT OF SEARCH
AREA b : "));
    double r;
    fb f=new fb();
    r= false_position(f,a,b);
    JOptionPane.showMessageDialog(null," ROOT  : "+r+"\nFUNCTION VALUE : "+f.func(r),
    "false position root finding method : ",JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```

bisection.m

```
function xr=bisection(f,xl,xu)
% ikiyebölme metodu (bisection)
maxit=100;
iter=0;
es=0.0000001;
ea=1.1*es;
while((ea>es)&&(iter<maxit))
    xr=(xl+xu)/2.0;
    iter=iter+1;
    if xr~=0 ea=abs((xu-xl)/(xu+xl))*100;end
    fxl= f(xl);
    fxr= f(xr);
    test= fxl*fxr;
    if test == 0.0       ea=0;
    elseif test < 0.0  xu=xr;
    else                xl=xr;
    end
end
if(iter>=maxit) fprintf('maksimum döngü sayısı aşıldı , sonuç geçerli
olmayabilir');
end
end
```

false_position.m

```
function p=false_position(f,xa,xu)
% modified false position root finding method
maxit=100;
iter=0;
es=0.000001;
ea=1.1*es;
ia=0;
iu=0;
fa=f(xa);
fu=f(xu);
p=xu;
double xpold;
while(iter<maxit && ea>es)
xpold=p;
p=xu-fu*(xa-xu)/(fa-fu);
fp=f(p);
iter=iter+1;
```

```
if p~=0 ea=abs((p-xpold)/p)*100.0;
test=fa*fp;
if test<0
    xu=p;
    fu=f(xu);
    iu=0;ia=ia+1;
    if ia >= 2 fa=fa/2.0;end
elseif test>0  xa=p;fa=f(xa);ia=0;iu=iu+1;
        if iu>=2 fu=fu/2.0; end
elseif test==0 ea = 0; end
end
if iter>=maxit fprintf('Maximum number of iteration is exceeded result might not be
valid'); end
end
```