COMPUTER

EGE UNIVERSITY SCHOOL OF ENGINEERING

DEPARTMENT OF MECHANICAL ENGINEERING

INSTRUCTOR: Dr. M. Turhan Çoban

2017-2018 Fall semester

1.0 SUM OF NUMBERS FROM 1 TO N

Java example version 1 (for loop)

```
import javax.swing.JOptionPane;
public class sum
{
public static double sum(int m)
{  int total=0;
   for(int i=1;i<=m;i++) {total=total+i;}
   return total;
}

public static void main(String arg[])
{
int n=Integer.parseInt(JOptionPane.showInputDialog("n = "));
System.out.println("total = "+sum(n));
}}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" sum
total = 5050.0
> Terminated with exit code 0.
```

Java example version 2 (while loop)

```
import javax.swing.JOptionPane;
public class sum1
{
public static double sum1(int m)
{  int total=0;
   int i=1;
   while(i<=m) {total=total+i;i++;}
   return total;
}

public static void main(String arg[])
{
int n=Integer.parseInt(JOptionPane.showInputDialog("n = "));
System.out.println("total = "+sum1(n));
}}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" sum1
total = 5050.0
> Terminated with exit code 0.
```

C++ example version 1 (for loop)

```
#include <stdio.h>
#include <iostream>
#include <math.h>
using namespace std;

double sum(int m)
{  int total=0;
   for(int i=1;i<=m;i++) {total=total+i;}
   return total;
}

int main()
{ int n=0;
cout<<"n = ";
cin>>n;
cout<<"total = "<<sum(n);
}
```

```
---------- Capture Output ----------
> "D:\okul\SCO1\run.bat" sum

D:\okul\SCO1>sum
n = 100
total = 5050
```

```
> Terminated with exit code 0.
```

C++ example version 2 (while loop)

```
#include <stdio.h>
#include <iostream>
#include <math.h>
using namespace std;

double sum1(int m)
{  int total=0;
   int i=1;
   while(i<=m) {total=total+i;i++;}
   return total;
}

int main()
{ int n=0;
cout<<"n = ";
cin>>n;
cout<<"total = "<<sum1(n);
}
```

Python example version 1 (for loop)

```
from math import *

def sum(n):
    total=0;
    for i in range(1,(n+1)):
        total=total+i;
    return total;

n=int(input("n = "));
r=sum(n);
print("total=",r);
```

```
---------- Capture Output ----------
> "D:\co\python\pythonw.exe" func3.py
n = 100
total= 5050

> Terminated with exit code 0.
```

Python example version 1 (for loop)

```
from math import *

def sum1(n):
    total=0;
    i=1;
    while i<=n :
        total=total+i;
        i=i+1;
    return total;

n=int(input("n = "));
r=sum1(n);
print("total=",r);
```

```
---------- Capture Output ----------
> "D:\co\python\pythonw.exe" sum1.py
n = 100
total= 5050

> Terminated with exit code 0.
```

Octave(Matlab) version 1

```
>> sum(1:100)
ans =  5050
>>
```

Octave(Matlab) version 2 (for loop)

```
function total = sum1 (n)
  total=0;
               for i=1:n
   total=total+i;
    endfor
endfunction
```

```
>> sum1(100)
ans =  5050
>>
```

Octave(Matlab) version 3(while loop)

```
function total = sum2 (n)
  total=0;
  i=1;
  while (i <=n)
     total=total+i;
     i=i+1;
   endwhile
endfunction
```

```
>> sum2(100)
ans =  5050
>>
```

2.0 ROOTS OF THE SECOND DEGREE POLYNOMIALS

The roots of quadratic polynomials are well known.

$$f(x)=ax^2+bx+c=0$$

$$\Delta=b^2-4\text{ac}$$

$$x_0=\frac{-b}{2\text{a}}+\frac{\sqrt{(\Delta)}}{2\text{a}}$$

$$x_1=\frac{-b}{2\text{a}}-\frac{\sqrt{(\Delta)}}{2\text{a}}$$

Of course if $\Delta=b^2-4\text{ac}$ is negative roots will be complex numbers

Java version program

```
public class square_roots
{
public static double[][] square_roots(double d[])
{
   double x[][]=new double[2][2];
   double a=d[2];
   double b=d[1];
   double c=d[0];
   double delta=b*b-4*a*c;
   if(delta>=0)
    {
   x[0][0]=(-b+Math.sqrt(delta))/(2*a);
   x[1][0]=(-b-Math.sqrt(delta))/(2*a);
    }
    else
    {x[0][0]=-b/(2*a);
     x[1][0]=-b/(2*a);
     x[0][1]=Math.sqrt(-delta)/(2*a);
     x[1][1]=Math.sqrt(-delta)/(2*a);
    }
   return x;
}
public static void main(String arg[])
{
double d[]={1,-2,1};
double x[][]=square_roots(d);
String s="";
for(int i=0;i<x.length;i++)
{ s+="x["+i+"] = "+x[i][0]+" + "+x[i][1]+"*i\n";}
System.out.println(s);
```

```
}}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" square_roots
x[0] = 1.0 + 0.0*i
x[1] = 1.0 + 0.0*i


> Terminated with exit code 0.
```

Python version

```
from math import *

def square_roots(d):
   x= [0 for i in range(2)];
   a=d[2];
   b=d[1];
   c=d[0];
   delta=b*b-4*a*c;
   x[0]=(-b+sqrt(delta))/(2*a);
   x[1]=(-b-sqrt(delta))/(2*a);
   return x;

a=[1.0,-2.0,1.0]  # y(x)=x^2-2*x++1
print("Roots of the polynomial : ",square_roots(a))
```

```
---------- Capture Output ----------
> "D:\co\python\pythonw.exe" square_roots.py
Roots of the polynomial :  [1.0, 1.0]

> Terminated with exit code 0.
```

OpenOffice Calc (Excel) version

Quadratic polynomial $f(x)=ax^2+bx+c=0$

enter coefficients

a	b	c
1	-4	1

Δ	12	
	real	imag
xo	3,7320508076	0
x1	0,2679491924	0

Octave (Matlab) version

```
>> x=[1,-4,1]
x =

   1  -4   1

>> roots(x)
ans =

   3.732050807568877
   0.267949192431123

>>
```

Note that we do not have to write any program in Octave environment to solve the problem.

3.0 ROOTS OF THE CUBIC POLYNOMIALS

Similar to quadratic polynomials, third degree polynomials can also be solved by using analytical formulations. The basic difference is that the roots should be calculated as complex variables.

Assuming a third degree polynomial of

$$f(x)=d_3x^3+d_2x^2+d_1x+d_0=0$$

$$a=\frac{d_2}{d_3}\quad b=\frac{d_1}{d_3}\quad c=\frac{d_0}{d_3}$$

$$f(x)=x^3+ax^2+bx+c=0$$

$$Q=\frac{a^2-3b}{9}\quad y_1=2a^3\quad y_2=-9ab\quad y_3=27c\quad y_4=y_1+y_2+y_3$$

$$R=\frac{2a^3-9ab+27c}{54}=\frac{y_4}{54}\quad \theta=\arccos\left(\frac{R}{\sqrt{(Q^3)}}\right)$$

If $(R^2<Q^3)$ three real root existed

$$x_0=(-2\text{sqrt}(Q))\cos\left(\frac{\theta}{3}\right)-\frac{a}{3}$$

$$x_1=(-2\text{sqrt}(Q))\cos\left(\frac{\theta-2\pi}{3}\right)-\frac{a}{3}$$

$$x_2=(-2\text{sqrt}(Q))\cos\left(\frac{\theta+2\pi}{3}\right)-\frac{a}{3}$$

(This equation first appears in Chapter VI of François Viete's treatise "De emendatione," published in 1615!)

If $(R^2>Q^3)\quad A=(R+\sqrt{(R^2-Q^3)})^{1/3}$

If A=0 then B=0

$$B=\frac{Q}{A}$$

if A<>0

$$x_0=(A+B)-\frac{a}{3}$$

$$x_1=\left[-\left(\frac{A+B}{2}\right)-\frac{a}{3}\right]+\left[\frac{\sqrt{(3)}(A-B)}{2}\right]i$$

$$x_2=\left[-\left(\frac{A+B}{2}\right)-\frac{a}{3}\right]-\left[\frac{\sqrt{(3)}(A-B)}{2}\right]i$$

Java version program

```
 import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import java.text.*;
import java.util.Locale;
//import Numeric;

class NA13D
{
public static double[][] cubic_roots(double d[])
{
// roots of a degree 3 polynomial
// P(x)=a[3]*x^3+a[2]*x^2+a[1]*x+a[0]=0;
// assuming all a is all real;
// if more than 4 values are entered the remaining ones will be ignored
double x[][]=new double[2][3];
double a,b,c;
a=d[2]/d[3];
```

```
b=d[1]/d[3];
c=d[0]/d[3];
double Q,R,theta;
Q=(a*a-3.0*b)/9.0;
double y1,y2,y3,y4;
y1=2.0*a*a*a;
y2=-9*a*b;
y3=27*c;
y4=y1+y2+y3;
R=y4/54.0;
double Q3=Q*Q*Q;
double R2=R*R;
double qq;
double A,B;
if(R2<Q3)
{
qq=-2.0*Math.sqrt(Q);
theta=Math.acos(R/Math.sqrt(Q3));
x[0][0]=qq*Math.cos(theta/3.0)-a/3.0;
x[0][1]=qq*Math.cos((theta-2.0*Math.PI)/3.0)-a/3.0;
x[0][2]=qq*Math.cos((theta+2.0*Math.PI)/3.0)-a/3.0;
}
else
{
A=-Math.pow((R+Math.sqrt(R2-Q3)),(1.0/3.0));
if(A==0) B=0;
else     B=Q/A;
x[0][0]=(A+B)-a/3.0;
x[0][1]=-0.5*(A+B)-a/3;
x[1][1]=Math.sqrt(3.0)/2.0*(A-B);
x[0][2]=-0.5*(A+B)-a/3;
x[1][2]=-Math.sqrt(3.0)/2.0*(A-B);
}
return x;
}

public static c[] cubic_rootsC(double c[])
{
// roots of a degree 3 polynomial
//return 3 complex roots
double a[][]=new double[2][3];
a=cubic_roots(c);
c e[]=new c[3];
for(int i=0;i<3;i++)
  e[i]=new c(a[0][i],a[1][i]);
return e;
}

    public static double[] enterdata(String s)
    {
    String s1=JOptionPane.showInputDialog(s);
    StringTokenizer token=new StringTokenizer(s1);
    int n=token.countTokens()-1;
    int m=n+1;
    double a[]=new double[m];
    int j=0;
    while(token.hasMoreTokens())
    {
    Double ax=new Double(token.nextToken());
    a[j++]=ax.doubleValue();
    }
    return a;
    }

  public static void main(String args[]) throws IOException
  {
  String s="enter coefficients of the third degree polynomial a[0]..a[3] y=a[0]+a[1]*x+,a[2]*x^2+a[3}*x^3: ";
  double [] x0=enterdata(s);
  c z[]=cubic_rootsC(x0);
  String s1="";
  for(int i=0;i<3;i++)
    {s1+=c.toString(z[i])+"\n";}
  System.out.println(s1);
  }
}
```

```
---------- Capture Output ----------
```

```
> "D:\co\java\bin\javaw.exe" NA13D
(   1.0000000000 +    0.0000000000*i )
(   1.0000000000 +   -0.0000000000*i )
(   1.0000000000 +    0.0000000000*i )

> Terminated with exit code 0.
```

Python version

```
from math import *

def cubic_roots(d):
  x=[[0 for i in range(3)] for j in range(2)]
  a=d[2]/d[3]
  b=d[1]/d[3]
  c=d[0]/d[3]
  Q=(a*a-3.0*b)/9.0
  y1=2.0*a*a*a
  y2=-9*a*b
  y3=27*c
  y4=y1+y2+y3
  R=y4/54.0
  Q3=Q*Q*Q
  R2=R*R
  if R2<Q3:
    qq=-2.0*sqrt(Q)
    theta=acos(R/Math.sqrt(Q3))
    x[0][0]=qq*cos(theta/3.0)-a/3.0
    x[0][1]=qq*cos((theta-2.0*pi)/3.0)-a/3.0
    x[0][2]=qq*cos((theta+2.0*pi)/3.0)-a/3.0
  else:
    A=-pow((R+sqrt(R2-Q3)),(1.0/3.0))
    if A==0:  B=0
    else   :  B=Q/A
    x[0][0]=(A+B)-a/3.0
    x[0][1]=-0.5*(A+B)-a/3
    x[1][1]=sqrt(3.0)/2.0*(A-B)
    x[0][2]=-0.5*(A+B)-a/3
    x[1][2]=-sqrt(3.0)/2.0*(A-B)
  return x

a=[1.0,-3.0,3.0,-1.0]  # y(x)=x^3+3*x^2+3*x+1=(x+1)^3
print("Roots of the polynomial : ",cubic_roots(a))
```

```
---------- Capture Output ----------
> "D:\co\python\pythonw.exe" NA13D.py
Roots of the polynomial :  [[1.0, 1.0, 1.0], [0, -0.0, 0.0]]

> Terminated with exit code 0.
```

Octave (Matlab) version

```
>> x=[1,-3,3,-1]
x =

   1  -3   3  -1

>> roots(x)
ans =

   1.000006571943641 + 0.000000000000000i
   0.999996714028179 + 0.000005691454547i
   0.999996714028179 - 0.000005691454547i

>>
```

Note that we do not have to write any program in Octave environment to solve the problem.

OpenOffice Calc (Excel) version

Root of third degree polynomial (analytical formula) Excel version

$f(x) = d_3x^3 + d_2x^2 + d_1x + d_0 = 0$

$f(x) = x^3 + ax^2 + bx + c = 0$

Enter poynomial coefficients

d3	d2	d1	d0
1	-3	3	-1

	a	b	c
1	-3	3	-1

Q	0
y1	-54
y2	81
y3	-27
y4	0
R	0
θ	1,5707963

true/false
0

true if R^2<Q^3

x0	x1	x2
0	0	0

true if R^2>=Q^3

A	0	
B	0	

true/false
1

xo real	xo imaginary
1	0

x1 real	x1 imaginary
1	0

x2 real	x2 imaginary
1	-0

4.0 TAYLOR SERIES AND EXPONENTIAL FUNCTION SERIES SOLUTION

Taylor equation can be considered the roots of many numerical analysis methods. It will give us serial. **Taylor series** is a representation of a function as an infinite sum of terms calculated from the values of its derivatives at a single point. It is named after the English mathematician Brook Taylor. If the series is centered at zero, the series is also called a **Maclaurin series**, named after the Scottish mathematician Colin Maclaurin. It is common practice to use a finite number of terms of the series to approximate a function. Taylor series around a point a can be written as

$$f(x)=f(a)+\frac{f'(a)}{1!}(x-a)+\frac{f''(a)}{2!}(x-a)^2+\frac{f^{(3)}(a)}{3!}(x-a)^3+...$$

Or if it is written in summation notation

$$f(x)=\frac{\sum_{n=0}^{\infty} f^{(n)}(a)}{n!}(x-a)^n$$

Or if a is replaces with x^k and x is replaced with x^{k+1}

$$f(x^{k+1})=\frac{\sum_{n=0}^{\infty} f^{(n)}(x^k)}{n!}(x^{k+1}-x^k)^n$$

Computers could not calculate anything more complex than addition, the rest usually accomplish through Taylor series approximations. For example function $f(x)=e^x$ can be

approximated as $$f(x)=e^x=1+\frac{x}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}+\frac{x^4}{4!}+\frac{x^5}{5!}+..$$

Java Version

```
public class func
{
  public static double exp(double x)
  {double total=0;
   double factorial=1;
   double power=1;
   for(int i=1;i<=200;i++)
   {total=total+power/factorial;
    power=power*x;
    factorial=factorial*i;
   }
   return total;
  }
  public static void main(String arg[])
  {System.out.println(exp(1));}
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" func
2.7182818284590455

> Terminated with exit code 0.
```

C++ Version

```
#include <stdio.h>
#include <iostream>
#include <math.h>
using namespace std;

  double exp(double x)
  {double total=0;
   double factorial=1;
   double power=1;
   for(int i=1;i<=200;i++)
   {total=total+power/factorial;
    power=power*x;
    factorial=factorial*i;
   }
   return total;
  }
  int main()
  { double r=exp(1.0);
    printf("r=%20.15f\n",r);
    cout<<"r="<<r;
    return 0;
  }
```

```
---------- Capture Output ----------
> "D:\okul\SCO1\run.bat" func

D:\okul\SCO1>func
r=   2.718281828459046
r=2.71828
> Terminated with exit code 0.
```

Python version

```
from math import *

def exp(x):
    total=0;
    factorial=1;
    power=1;
    for i in range(1,100):
      total=total+power/factorial;
      power=power*x;
      factorial=factorial*i;
    return total;

r=exp(1.0);
print("r=",r);
```

```
---------- Capture Output ----------
> "D:\co\python\pythonw.exe" func.py
r= 2.7182818284590455

> Terminated with exit code 0.
```

Octave (Matlab) version

```
function total=exp1(x)
    total=0;
    factorial=1;
    power=1;
    for i=1:100
      total=total+power/factorial;
      power=power*x;
      factorial=factorial*i;
    endfor
endfunction
```

```
>> exp1(1)
ans =  2.71828182845905
>>
```

Note: the function name exp is not used due to predefined function is already existed in Octave

OpenOffice Calc (excel) version

	x=	1	
n	x^n	n!	total
0	1	1	1
1	1	1	2
2	1	2	2,5
3	1	6	2,66666667
4	1	24	2,70833333
5	1	120	2,71666667
6	1	720	2,71805556
7	1	5040	2,71825397
8	1	40320	2,71827877
9	1	362880	2,71828153
10	1	3628800	2,7182818
11	1	39916800	2,71828183
12	1	479001600	2,71828183
13	1	6227020800	2,71828183
14	1	8,718E+010	2,71828183

5.0 TAYLOR SERIES AND LOGARITHMIC FUNCTION SERIES SOLUTION

Taylor equation can be considered the roots of many numerical analysis methods. It will give us serial. **Taylor series** is a representation of a function as an infinite sum of terms calculated from the values of its derivatives at a single point. It is named after the English mathematician Brook Taylor. If the series is centered at zero, the series is also called a **Maclaurin series**, named after the Scottish mathematician Colin Maclaurin. It is common practice to use a finite number of terms of the series to approximate a function. Taylor series around a point a can be written as

$$f(x)=f(a)+\frac{f'(a)}{1!}(x-a)+\frac{f''(a)}{2!}(x-a)^2+\frac{f^{(3)}(a)}{3!}(x-a)^3+...$$

Or if it is written in summation notation

$$f(x)=\frac{\sum_{n=0}^{\infty} f^{(n)}(a)}{n!}(x-a)^n$$

Or if a is replaces with x^k and x is replaced with x^{k+1}

$$f(x^{k+1}) = \frac{\sum_{n=0}^{\infty} f^{(n)}(x^k)}{n!}(x^{k+1}-x^k)^n$$

Computers could not calculate anything more complex than addition, the rest usually accomplish through Taylor series approximations. For example function f(x)=ln(x) can be approximated as

$$f(x) = \ln(x) = \log_e(x) = 2\sum_{k=1}^{\infty} \frac{1}{2k-1}\left(\frac{x-1}{x+1}\right)^{2k-1} \quad [x>0]$$

Java Version

```
public class func2
{
  public static double ln(double x)
  {double total=0;
   double y=(x-1.0)/(x+1.0);
   double power=y;
   for(int k=1;k<=200;k++)
   {  total=total+power/(2.0*k-1.0);
     power=power*y*y;
   }
   return 2.0*total;
  }
  public static void main(String arg[])
  {   double r=2.7182818284590455;
              System.out.println(ln(r));
  }
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" func2
0.9999999999999999

> Terminated with exit code 0.
```

C++ Version

```
#include <stdio.h>
#include <iostream>
#include <math.h>
using namespace std;
double ln(double x)
  {double total=0;
   double y=(x-1.0)/(x+1.0);
   double power=y;
   for(int k=1;k<=200;k++)
   {  total=total+power/(2.0*k-1.0);
     power=power*y*y;
   }
   return 2.0*total;
  }
  int main()
  {   double r=2.7182818284590455;
              cout<<ln(r);
  }
```

```
---------- Capture Output ----------
> "D:\okul\SCO1\run.bat" func2

D:\okul\SCO1>func2
1
> Terminated with exit code 0.
```

Python version

```
from math import *

def ln(x):
   total=0;
   y=(x-1.0)/(x+1.0);
   power=y;
   for k in range(1,100):
```

```
      total=total+power/(2.0*k-1.0);
      power=power*y*y;
   return 2*total;
e=r=2.7182818284590455;
r=ln(e);
print("r=",r);
```

```
---------- Capture Output ----------
> "D:\co\python\pythonw.exe" func2.py
r= 0.9999999999999999

> Terminated with exit code 0.
```

Octave (Matlab) version

```
function total=ln(x)
    total=0;
    y=(x-1.0)/(x+1.0);
    power=y;
    for k=1:100
      total=total+power/(2.0*k-1.0);
      power=power*y*y;
    endfor
    total=total*2;
endfunction
```

```
>> ln(e)
ans =  1.000000000000000
>>
```

Note: the function name exp is not used due to predefined function is already existed in Octave

OpenOffice Calc (excel) version

ln(x)

k	x=	2,7182818285
	y=(x-1)/(x+1)	0,4621171573
	power	total
0	0,4621171573	0
1	0,0986861666	0,9242343145
2	0,0210746546	0,9900250922
3	0,0045005403	0,9984549541
4	0,0009611006	0,9997408227
5	0,0002052452	0,9999544006
6	4,38306E-005	0,9999917179
7	9,36012E-006	0,9999984611
8	1,99887E-006	0,9999997091
9	4,26864E-007	0,9999999443
10	9,11578E-008	0,9999999892
11	1,94670E-008	0,9999999979
12	4,15721E-009	0,9999999996
13	8,87782E-010	0,9999999999
14	1,89588E-010	1
15	4,04869E-011	1
16	8,64608E-012	1
17	1,84639E-012	1
18	3,94301E-013	1
19	8,42038E-014	1
20	1,79819E-014	1

6.0 TAYLOR SERIES AND POWER FUNCTION SERIES SOLUTION

Taylor equation can be considered the roots of many numerical analysis methods. It will give us serial. **Taylor series** is a representation of a function as an infinite sum of terms calculated from the values of its derivatives at a single point. It is named after the English mathematician Brook Taylor. If the series is centered at zero, the series is also called a **Maclaurin series**, named after the Scottish mathematician Colin Maclaurin. It is common practice to use a finite number of terms of the series to approximate a function. Taylor series around a point a can be written as

$$f(x)=f(a)+\frac{f'(a)}{1!}(x-a)+\frac{f''(a)}{2!}(x-a)^2+\frac{f^{(3)}(a)}{3!}(x-a)^3+...$$

Or if it is written in summation notation

$$f(x)=\frac{\sum_{n=0}^{\infty} f^{(n)}(a)}{n!}(x-a)^n$$

Or if a is replaces with x^k and x is replaced with x^{k+1}

$$f(x^{k+1})=\frac{\sum_{n=0}^{\infty} f^{(n)}(x^k)}{n!}(x^{k+1}-x^k)^n$$

Computers could not calculate anything more complex than addition, the rest usually accomplish through Taylor series approximations. For example function f(x)=pow(a,x) can be approximated as

$$f(x)=e^x=1+\frac{x}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}+\frac{x^4}{4!}+\frac{x^5}{5!}+..$$

$$f(x)=\ln(x)=\log_e(x)=2\sum_{k=1}^{\infty}\frac{1}{2k-1}\left(\frac{x-1}{x+1}\right)^{2k-1} \qquad [x>0]$$

$$f(x)=a^x=e^{(x*\ln(a))}$$

Java Version

```
public class func3
{
  public static double ln(double x)
  {double total=0;
   double y=(x-1.0)/(x+1.0);
   double power=y;
   for(int k=1;k<=200;k++)
   {   total=total+power/(2.0*k-1.0);
       power=power*y*y;
   }
   return 2.0*total;
  }
  public static double exp(double x)
  {double total=0;
   double factorial=1;
   double power=1;
   for(int i=1;i<=200;i++)
   {total=total+power/factorial;
    power=power*x;
    factorial=factorial*i;
   }
   return total;
  }
  public static double pow(double a,double x)
  {return exp(x*ln(a));}

  public static void main(String arg[])
  {   System.out.println(pow(2.0,3.0));}
}
```

```
---------- Capture Output ----------
```

```
> "D:\co\java\bin\javaw.exe" func3
7.999999999999997

> Terminated with exit code 0.
```

C++ Version

```
#include <stdio.h>
#include <iostream>
#include <math.h>
using namespace std;

   double exp(double x)
   {double total=0;
    double factorial=1;
    double power=1;
    for(int i=1;i<=200;i++)
    {total=total+power/factorial;
     power=power*x;
     factorial=factorial*i;
    }
    return total;
   }
   double ln(double x)
   {double total=0;
    double y=(x-1.0)/(x+1.0);
    double power=y;
    for(int k=1;k<=200;k++)
    {  total=total+power/(2.0*k-1.0);
       power=power*y*y;
    }
    return 2.0*total;
   }
   double pow(double a,double x)
   {return exp(x*ln(a));}
   int main()
   { double r=pow(2.0,3.0);
     printf("r=%20.15f\n",r);
     cout<<"r="<<r;
     return 0;
   }
```

```
---------- Capture Output ----------
> "D:\okul\SCO1\run.bat" func3

D:\okul\SCO1>func3
r=   7.999999999999997
r=8
> Terminated with exit code 0.
```

Python version

```
from math import *

def ln(x):
    total=0;
    y=(x-1.0)/(x+1.0);
    power=y;
    for k in range(1,100):
        total=total+power/(2.0*k-1.0);
        power=power*y*y;
    return 2*total;
def exp(x):
      total=0;
      factorial=1;
      power=1;
      for i in range(1,100):
        total=total+power/factorial;
        power=power*x;
        factorial=factorial*i;
      return total;
def pow(a,x):
    return exp(x*log(a));

r=pow(2.0,3.0);
print("r=",r);
```

```
---------- Capture Output ----------
> "D:\co\python\pythonw.exe" func3.py
r= 7.999999999999999

> Terminated with exit code 0.
```

Octave (Matlab) version

```
function total=pow(a,x)
    total=exp1(x*ln(a));
endfunction
```

```
>> pow(2,3)
ans =  8.00000000000000
>>
```

Note: the function name exp is not used due to predefined function is already existed in Octave

OpenOffice Calc (excel) version

ln(a)

k	a=	2
	x=	3
	y=(x-1)/(x+1)	0,3333333333

	power	total
0	0,3333333333	0
1	0,037037037	0,6666666667
2	0,0041152263	0,6913580247
3	0,0004572474	0,6930041152
4	5,08053E-005	0,6931347573
5	0,000005645	0,6931460474
6	6,27225E-007	0,6931470738
7	6,96917E-008	0,6931471703
8	7,74352E-009	0,6931471795
9	8,60392E-010	0,6931471805
10	9,55991E-011	0,6931471805
11	1,06221E-011	0,6931471806
12	1,18024E-012	0,6931471806
13	1,31137E-013	0,6931471806
14	1,45708E-014	0,6931471806
15	1,61898E-015	0,6931471806
16	1,79887E-016	0,6931471806
17	1,99874E-017	0,6931471806
18	2,22082E-018	0,6931471806
19	2,46758E-019	0,6931471806
20	2,74175E-020	0,6931471806

exp(z) =	exp(x*ln(a))
z=x*ln(a)	2,0794415417

power	factorial	total
1	1	1
2,0794415417	1	3,0794415417
4,3240771253	2	5,2414801043
8,9916656037	6	6,7400910383
18,697642985	24	7,519159496
38,880655555	120	7,8431649589
80,850050329	720	7,9554566955
168,1229533	5040	7,9888144243
349,6018532	40320	7,9974851052
726,9766166	362880	7,9994884579
1511,7053764	3628800	7,9999050434
3143,5029584	39916800	7,9999837947
6536,7306382	479001600	7,9999974413
13592,749236	6227020800	7,9999996242
28265,327427	87178291200	7,9999999484
58776,09604	1,3077E+012	7,9999999934
122221,45576	2,0923E+013	7,9999999992
254152,3724	3,5569E+014	7,9999999999
528495,00108	6,4024E+015	8
1098974,4598	1,2165E+017	8
2285253,145	2,4329E+018	8

7.0 RUSSIAN PEASANT MULTIPLICATION

Take two integer for example 21 and 37 then double the first one(left hand side) and divide by two the second one(right side) ignore remaining in division process and stop when the rigth hand side reached to 1 and then cross each line where the right side number is even add up remaining left hand side numbers for example
21 37
42 18 cross out (18 is even)
84 9
168 4 cross out(4 is even)
336 2 cross out(2 is even)
672 1

then 21+84+672=777

21*37=777

Java version of Russian multiplication

```
public  class russian
{
public static int multiply(int x1,int x2)
{ int y2=x2;
  int y1=x1;
  int total=0;
  while(y2>=1)
  {  if(y2%2 != 0) total+=y1;
                System.out.println("y1 = "+y1+" y2 = "+y2);
    y1=y1*2;y2=y2/2;
  }
  return total;
}
public static void main(String arg[])
{System.out.println(multiply(21,37));}
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" russian
y1 = 21 y2 = 37
y1 = 42 y2 = 18
y1 = 84 y2 = 9
y1 = 168 y2 = 4
y1 = 336 y2 = 2
y1 = 672 y2 = 1
777

> Terminated with exit code 0.
```

C++ version of Russian multiplication

```
#include <stdio.h>
#include <iostream>
#include <math.h>
using namespace std;

int multiply(int x1,int x2)
{ int y2=x2;
  int y1=x1;
  int total=0;
  while(y2>=1)
  {  if(y2%2 != 0) total+=y1;
                cout<<"y1 = "<<y1<<" y2 = "<<y2<<endl;
    y1*=2;y2/=2;
  }
  return total;
}

int main()
{ cout << "multiplication = "<<multiply(21,37);
  return 0;
}
```

```
---------- Capture Output ----------
> "D:\okul\SCO1\run.bat" russian

D:\okul\SCO1>russian
y1 = 21 y2 = 37
y1 = 42 y2 = 18
y1 = 84 y2 = 9
y1 = 168 y2 = 4
y1 = 336 y2 = 2
y1 = 672 y2 = 1
multiplication = 777
> Terminated with exit code 0.
```

Python version of Russian Multiplication

```
from math import *
```

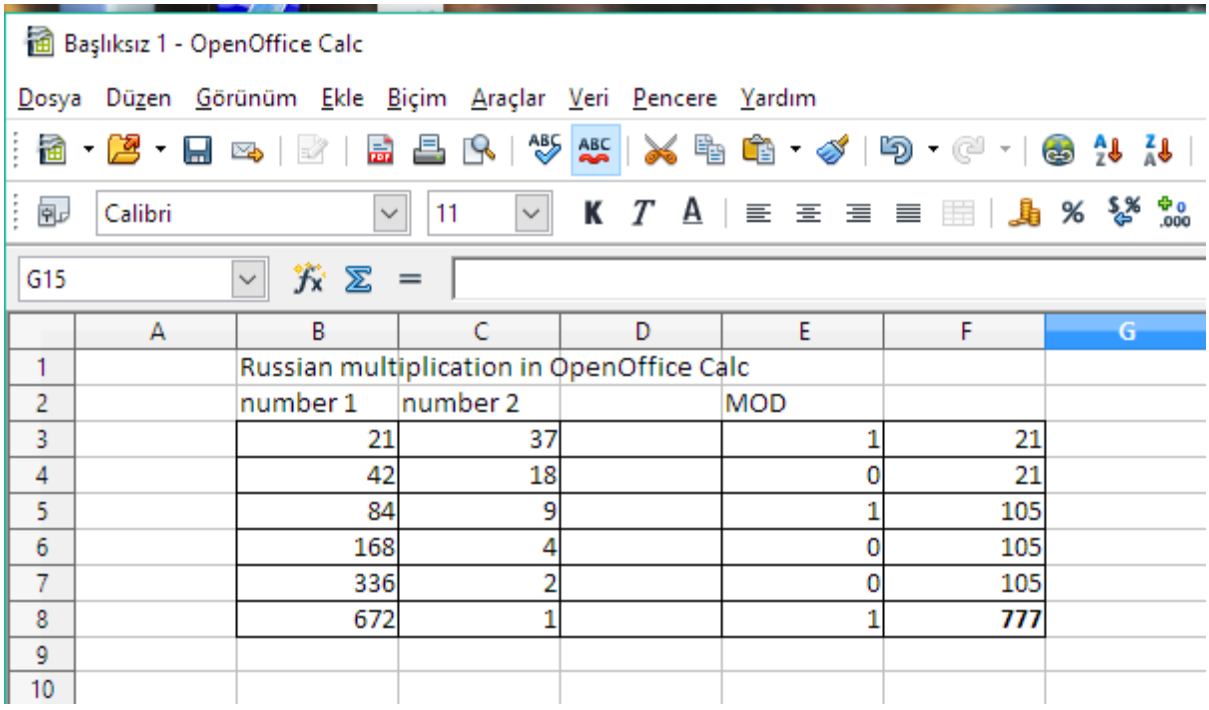
```
def multiply(x1,x2):
  y2=x2;
  y1=x1;
  b2=0;
  total=0;
  while y2>=1:
    if y2 %2 != 0 :
      total=total+y1;
    print("total=",total,"y1=",y1,"y2=",y2);
    y1=y1*2;
    y2=(int)(y2/2);
  return total;

x1=21;
x2=37;
x=multiply(x1,x2);
print("Russian multiplication :\nx = ",x)
```

```
---------- Capture Output ----------
> "D:\co\python\pythonw.exe" russian.py
total= 21 y1= 21 y2= 37
total= 21 y1= 42 y2= 18
total= 105 y1= 84 y2= 9
total= 105 y1= 168 y2= 4
total= 105 y1= 336 y2= 2
total= 777 y1= 672 y2= 1
Russian multiplication :
x =  777

> Terminated with exit code 0.
```

OpenOffice calc (excel) version of russian multiplication



	A	B	C	D	E	F	G
1		Russian multiplication in OpenOffice Calc					
2		number 1	number 2		MOD		
3		21	37		1	21	
4		42	18		0	21	
5		84	9		1	105	
6		168	4		0	105	
7		336	2		0	105	
8		672	1		1	**777**	
9							
10							

8.0 ROOT FINDING: BISECTION METHOD

In the bisection method root of the functions is searched in a given region aIf $a \geq x \geq b$ a single root is existed in the given region equation f(a)*f(b)<0 (2.1.1) will be satisfied. If this condition is not satisfied, it can be assumed that there is no root in the given region. If the root existed region is divide to two equal parts as

$$p = \frac{a+b}{2} \quad (2.1.2)$$

and function is evaluated in the middle. If

f(a)*f(b)<0 (2.1.3)

root should be located in a-p else it is located in p-b region. Rarely it is also possible that

f(a)*f(b)=0 (2.1.4)

can happen, that will indicate the root. The new root region is used in further iterations. In order to use iterative process an exceptible criteria for error limit can be assumed $\frac{(b-a)}{(b-a)} < \varepsilon$ (2.1.5)

Figure 2.1 Convergence of bisection Method

An example code is created for the Bisection method. (Java language)

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

class f1 extends f_x
{
public double func(double x)
{ return x*x-2;}          }

public class NA1
{

public static double bisection(f_x f,double xl,double xu)
{
//bisection root finding method
double test;
double xr=0;
double es,ea;
double fxl,fxr,fxu;
double xold=0;
int maxit=100,iter=0;
es=0.0000001;
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))
{   xold=xr;
    xr=(xl+xu)/2.0;
    fxr= f.func(xr);
    iter++;
    if(xr!=0)
        { ea=Math.abs((xr-xold)/xr)*100;}
    test= fxl*fxr;
    if(test==0.0)      ea=0;
    else if(test<0.0) {xu=xr;fxu=fxr;}
    else if(test>0)   {xl=xr;fxl=fxr;}
    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);
//JOptionPane.showMessageDialog(null,s,
//     "bisection root finding method : ",JOptionPane.PLAIN_MESSAGE);
   return xr;
}
     public static void main (String args[]) throws IOException
     {
     double r;
     f1 f=new f1();
     r= bisection(f,0.0,3.0);
     System.out.println(" ROOT  : "+r+"\nFUNCTION VALUE : "+f.func(r));
     System.exit(0);
     }
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" NA1
```

```
 ROOT  : 1.414213560987264
FUNCTION VALUE : -3.919722324496888E-9

> Terminated with exit code 0.
```

C++ language code:

```
#include <stdio.h>
#include <iostream>
#include <math.h>
using namespace std;

double func(double x)
{ return x*x-2.0;}

double bisection(double xl,double xu)
{
//bisection root finding method
double test;
double xr=0;
double es,ea;
double fxl,fxr;
int maxit=100;
int iter=0;
es=0.00000001;
ea=1.1*es;
while((ea>es)&&(iter<maxit))
{
   xr=(xl+xu)/2.0;
   iter++;
   if(xr!=0)
       { ea=fabs((xu-xl)/(xu+xl))*100.0;}
   fxl= func(xl);
   fxr= func(xr);
   test= fxl*fxr;
   if(test==0.0)      ea=0;
   else if(test<0.0)  xu=xr;
   else               {xl=xr;}
}
if(iter>=maxit) cout <<("Maximum number of iteration is exceeded \n result might not be valid");
return xr;
}
int main()
{  double a,b,r;
  cout << " INPUT LOWER LIMIT OF SEARCH AREA a : ";
  cin >>a;
  cout << " INPUT UPPER LIMIT OF SEARCH AREA b : ";
  cin >>b;
  r=bisection(a,b);
  printf("r=%20.15f\n",r);
  cout<<"r="<<r;
  return 0;
}
```

```
---------- Capture Output ----------
> "D:\okul\SCO1\run.bat" bisection

D:\okul\SCO1>bisection
 INPUT LOWER LIMIT OF SEARCH AREA a : 1
 INPUT UPPER LIMIT OF SEARCH AREA b : 3
r=   1.414213562267833
r=1.41421
> Terminated with exit code 0.
```

Python language code:

```
from math import *

def f1(x):
  return x*x-2.0

def bisection(xl,xu):
  # bisection root finding method
  maxit=100
  iter=0
```

```
  es=0.0000001
  ea=1.1*es
  s=""
  fxl= f1(xl)
  fxu= f1(xu)
  xr=xl
  while ea>es and iter<maxit:
    xold=xr
    xr=(xl+xu)/2.0;
    fxr= f1(xr);
    iter=iter+1;
    if xr!=0:
      ea=abs((xr-xold)/xr)*100
    test= fxl*fxr;
    if test==0.0 :      ea=0;
    elif test<0.0: xu=xr;fxu=fxr;
    elif test>0:    xl=xr;fxl=fxr;
    else :          ea=0;
    if iter>=maxit: print("Maximum number of iteration is exceeded \n result might not be valid")
  return xr

a=float(input("enter lower limit : "))
b=float(input("enter upper limit : "))
r=bisection(a,b);
print(" ROOT  : ",r,"\nFUNCTION VALUE : ",f1(r))
```

```
---------- Capture Output ----------
> "D:\co\python\pythonw.exe" NA1.py
enter lower limit : 1
enter upper limit : 3
 ROOT  :  1.4142135614529252
FUNCTION VALUE :  -2.6026334420947705e-09

> Terminated with exit code 0.
```

Octave(Matlab) language code

```
function xr=bisection(f,xl,xu)
% bisection root finding method
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('Maximum number of iteration is exceeded result might not be valid","MAKSİMUM NUMBER OF ITERATION
WARNING');
end
end
```

```
>> f=@(x)x*x-2
f =

@(x) x * x - 2

>> bisection(f,0,3)
ans =  1.4142
>>
```

OpenOffice calculate(excel) language code

function f(x)= x^2-2

xl	xu	xr	fxl	fxu	fxr	fxr*fxl
0	5	2,5	-2	23	4,25	-8,5
0	2,5	1,25	-2	4,25	-0,4375	0,875
1,25	2,5	1,875	-0,4375	4,25	1,515625	-0,66308594
1,25	1,875	1,5625	-0,4375	1,515625	0,44140625	-0,19311523
1,25	1,5625	1,40625	-0,4375	0,44140625	-0,02246094	0,00982666
1,40625	1,5625	1,484375	-0,02246094	0,44140625	0,203369141	-0,00456786
1,40625	1,484375	1,4453125	-0,02246094	0,203369141	0,088928223	-0,00199741
1,40625	1,4453125	1,42578125	-0,02246094	0,088928223	0,032852173	-0,00073789
1,40625	1,42578125	1,416015625	-0,02246094	0,032852173	0,00510025	-0,00011456
1,40625	1,416015625	1,411132813	-0,02246094	0,00510025	-0,00870419	0,000195504
1,411132813	1,416015625	1,413574219	-0,00870419	0,00510025	-0,00180793	1,5737E-005
1,413574219	1,416015625	1,414794922	-0,00180793	0,00510025	0,001644671	-2,9734E-006
1,413574219	1,414794922	1,41418457	-0,00180793	0,001644671	-8,2001E-005	1,4825E-007
1,41418457	1,414794922	1,414489746	-8,2001E-005	0,001644671	0,000781242	-6,4063E-008
1,41418457	1,414489746	1,414337158	-8,2001E-005	0,000781242	0,000349597	-2,8667E-008
1,41418457	1,414337158	1,414260864	-8,2001E-005	0,000349597	0,000133792	-1,0971E-008
1,41418457	1,414260864	1,414222717	-8,2001E-005	0,000133792	2,5894E-005	-2,1233E-009
1,41418457	1,414222717	1,414203644	-8,2001E-005	2,5894E-005	-2,8054E-005	2,3004E-009
1,414203644	1,414222717	1,414213181	-2,8054E-005	2,5894E-005	-0,00000108	3,0298E-011
1,414213181	1,414222717	1,414217949	-0,00000108	2,5894E-005	0,000012407	-1,3399E-011

9.0 ROOT FINDING: NEWTON-RAPHSON METHOD

Taylor Formula can be written as

$$f\left(x^{k+1}\right)=\frac{\sum_{n=0}^{\infty} f^{(n)}\left(x^{k}\right)}{n!}\left(x^{k+1}-x^{k}\right)^{n} \quad (2.4.1)$$

We would like to find $f(x_{n+1}) = 0$. If We substitude it into the Taylor Formula, and also cancel out terms above the first degree , The equation becomes

$$0=f\left(x_{n}\right)+f'\left(x_{n}\right)\left(x_{n+1}-x_{n}\right) \quad (2.4.2)$$

If x_{n+1} taken to left hand side Newton-Raphson Formula is obtained

$$x_{n+1}=x_{n}-f\frac{\left(x_{n}\right)}{f'\left(x_{n}\right)} \quad (2.4.3)$$

We can find the root iteratively starting from a first guess. Newton Formula aproached to the root quickly if the first guess is close to the actual root. But if the first gues is far away from the root, it might take longer or it might fail to reach to the root. Another difficulty in Newton-Raphson Formula is the requirement of knowing the actual derivative of the function

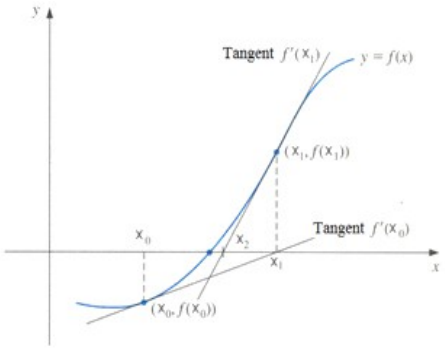


Figure 2.5 Convergence of the Newton-Raphson Method

An example code is created for the Newton-Raphson method. (Java language)

```
// Newton-Raphson root finding
//Numerical Analysis
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
//============== The function ================

class fb extends f_x
{  public double func(double x) { return x*x-2;}
}
//============== Derivative of the function ================
class dfb extends f_x
{
   public double func (double x)
   { return 2.0*x;}
}
//============================================================

public class NA5
{
public static double newton(f_x f,f_x df,double x)
{
int nmax=500;
double tolerance=1.0e-8;
double fx,dfx;
for(int i=0;i<nmax;i++)
{
  fx=f.func(x);
  dfx=df.func(x);
  x-=fx/dfx;
  System.out.println("i="+i+"x="+x+"fx="+fx+"dfx = "+dfx);
  if(Math.abs(fx)<tolerance) { return x;}
}
JOptionPane.showMessageDialog(null,"Warning : MAximum number of iteration is exceeded \n"+
   " çözüm geçerli olmiyabilir","MAXİMUM ITERATION NUMBER WARNING",JOptionPane.WARNING_MESSAGE);
return x;
}

    public static void main (String args[])
    {
              fb f=new fb();
              dfb df=new dfb();
    double x0;
    x0=Double.parseDouble(JOptionPane.showInputDialog(" First guess for the root : "));
    double r= newton(f,df,x0);
    System.out.println("r = "+r);
    }
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" NA5
i=0x=1.5fx=-1.0dfx = 2.0
i=1x=1.4166666666666667fx=0.25dfx = 3.0
i=2x=1.4142156862745099fx=0.006944444444444642dfx = 2.8333333333333335
i=3x=1.4142135623746899fx=6.007304882871267E-6dfx = 2.8284313725490198
i=4x=1.4142135623730951fx=4.510614104447086E-12dfx = 2.8284271247493797
r = 1.4142135623730951
> Terminated with exit code 0.
```

C++ language code:

```
#include <stdio.h>
#include <iostream>
#include <math.h>
using namespace std;

double func(double x)   { return x*x-2.0;}
double dfunc (double x) { return 2.0*x;}

double newton(double x)
{
int nmax=10;
double tolerance=1.0e-8;
```

```
double fx,dfx;
for(int i=0;i<nmax;i++)
{ fx=func(x);
  dfx=dfunc(x);
  x-=fx/dfx;
  cout<<"i="<<i<<"x="<<x<<"fx="<<fx<<"dfx = "<<dfx<<endl;
  if(fabs(fx)<tolerance) { return x;}
}
cout<<"Warning : Maximum number of iteration is exceeded \n";
return x;
}

    int main ()
    {
    double x0=1.0;
    double r= newton(x0);
    cout<<"r = "<<r;
    }
```

```
---------- Capture Output ----------
> "D:\okul\SCO1\run.bat" NA5

D:\okul\SCO1>NA5
i=0x=1.5fx=-1dfx = 2
i=1x=1.41667fx=0.25dfx = 3
i=2x=1.41422fx=0.00694444dfx = 2.83333
i=3x=1.41421fx=6.0073e-006dfx = 2.82843
i=4x=1.41421fx=4.51061e-012dfx = 2.82843
r = 1.41421
> Terminated with exit code 0.
> Terminated with exit code 0.
```

Python language code:

```
from math import *

def f1(x):
   return x*x-2.0

def df1(x):
   return 2.0*x

def newton(x):
   # Newton-Raphson root finding method
  nmax=100
  tolerance=1.0e-10
  for i in range(0,nmax):
      fx= f1(x);
      dfx=df1(x)
      x=x-fx/dfx
      print("fx",fx,"dfx",dfx,"x=",x)
      if abs(fx)<tolerance: return x
  return x

a=float(input("enter initial guess : "))
r=newton(a)
print(" ROOT  : ",r,"\nFUNCTION VALUE : ",f1(r))
```

```
---------- Capture Output ----------
> "D:\co\python\pythonw.exe" NA5.py
enter initial guess : 1
fx -1.0 dfx 2.0 x= 1.5
fx 0.25 dfx 3.0 x= 1.4166666666666667
fx 0.006944444444444642 dfx 2.8333333333333335 x= 1.4142156862745099
fx 6.007304882871267e-06 dfx 2.8284313725490198 x= 1.4142135623746899
fx 4.510614104447086e-12 dfx 2.8284271247493797 x= 1.4142135623730951
 ROOT  :  1.4142135623730951
FUNCTION VALUE :  4.440892098500626e-16

> Terminated with exit code 0.
```

Octave(Matlab) language code

```
function p=newton_raphson(f,df,x,tolerance,nmax)
% secant method for root finding
```

```
if nargin<5 nmax=100;end
if nargin<4 tolerance=1e-10;end
i=0;
fx=f(x);
while (abs(fx) > tolerance) && i<nmax
   fx=f(x);
   dfx=df(x);
   x=x-fx/dfx;
   i=i+1;
end
p=x;
if i>=nmax fprintf('Maximum number of iterations is exceeded the result may not be valid');end
endfunction
```

```
>> f=@(x)x*x-2;
>> df=@(x)2*x;
>> x=1;
>> x=newton_raphson(f,df,x)
x =  1.41421356237310
>>
```

OpenOffice calculate(excel) language code

Newton-Raphson root finding f(x)=x*x-a

a	2	
x	f(x)=x*x-a	fx/dx=2*x
1,00000000000000000	-1	2
1,50000000000000000	0,25	3
1,41666666666667000	0,0069444444	2,8333333333
1,41421568627451000	6,007304883E-006	2,8284313725
1,41421356237469000	4,510614104E-012	2,8284271247
1,41421356237310000	0	2,8284271247
1,41421356237310000	0	2,8284271247
1,41421356237310000	0	2,8284271247
1,41421356237310000	0	2,8284271247
1,41421356237310000	0	2,8284271247
1,41421356237310000	0	2,8284271247
1,41421356237310000	0	2,8284271247
1,41421356237310000	0	2,8284271247
1,41421356237310000	0	2,8284271247
1,41421356237310000	0	2,8284271247
1,41421356237310000	0	2,8284271247
1,41421356237310000	0	2,8284271247

10. ROOT FINDING: SECANT METHOD

One of the difficulty of the Newton-Raphson method is the requirement to find the second derivative. If difference equation (Numerical derivative) is used instead of derivative (tangent to the function), a secant to the function is drawn to the function, so the method is called secant method. If derivative is approximated with a first degree difference equation

$$f'(x_n) \approx \frac{(f(x_n) - f(x_{n-1}))}{(x_n - x_{n-1})} \quad (2.5.1)$$

Newton –Raphson Formula is converted to

$$x_{n+1} \approx x_n - \frac{f(x_n)(x_n - x_{n-1})}{(f(x_n) - f(x_{n-1}))} \quad \textbf{(2.5.2)}$$

Two first estimation is needed to start iteration by using this formula x_n and x_{-1}

If we would like to establish a one estimation secant Formula, the previous difference equation can be written for a small as (central difference Formula)
If we would like to establish a one estimation secant Formula, the previous difference equation can be written for a small Δx as (central difference Formula)

$$f'(x_n) \approx \frac{(f(x_n+\Delta x)-f(x_n-\Delta x))}{2\Delta x} \quad (2.5.3)$$

Second equation becomes:

$$x_{n+1} \approx x_n - \frac{2f(x_n)\Delta x}{(f(x_n+\Delta x)-f(x_n-\Delta x))} \quad (2.5.4)$$

In order to minimize error Δx should be relatively small, to have a further approximation a second or higher order derivative (difference) Formulas can also be used

$$f'(x_n) \approx \frac{(-f(x_n+2\Delta x)+8f(x_n+\Delta x)-8f(x_n-\Delta x)+f(x_n-2\Delta x))}{12\Delta x} \quad (2.5.5)$$

$$f'(x_n) \approx \frac{(-f(x_n+3\Delta x)-9f(x_n+2\Delta x)+45f(x_n+\Delta x)-45f(x_n-\Delta x)+9f(x_n-2\Delta x)-f(x_n-3\Delta x))}{60\Delta x} \quad (2.5.6)$$

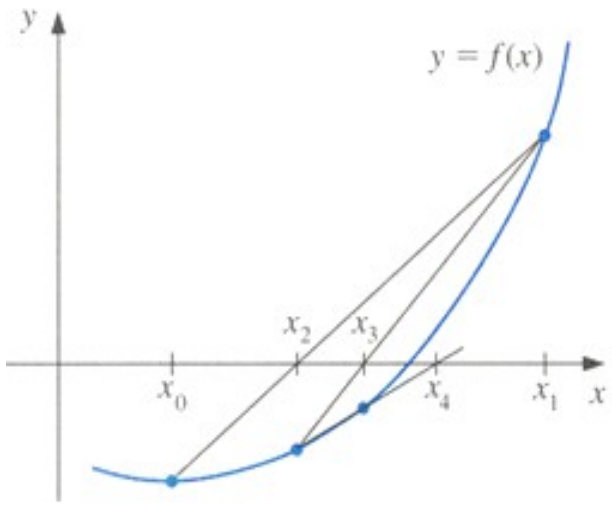


Figure 2.6 : Second method approximation with linear difference

An example code is created for the secant method. (Java language)

```
///Numerical Analysis
//
//
// Secant method with one root estimation
// Linear difference formula
//

import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

//============= function definition ================
class fb
{  public double func(double x) { return x*x-2;}
   public double dfunc(double x)
  {double h=1.0e-5;
   double f=0;
```

```
   f=(-func(x+2*h)+8.0*func(x+h)-8.0*func(x-h)+func(x+2.0*h))/(12.0*h);
   return f;
  }

}

public class NA8
{

public static double secant(fb f,double x)
{
int nmax=100;
double tolerance=1.0e-10;
double fx,dfx;
double dx=0.01;
for(int i=0;i<nmax;i++)
{
  fx=f.func(x);
  dfx=f.dfunc(x);
  x-=fx/dfx;
  System.out.println("fx="+fx+"dfx="+dfx+"x="+x);
  if(Math.abs(fx)<tolerance) { return x;}
}
JOptionPane.showMessageDialog(null,"Warning : Maximum number of iteration is exceeded \n"+
   " çözüm geçerli olmıyabilir","MAXİMUM ITERATION NUMBER WARNING",JOptionPane.WARNING_MESSAGE);
return x;
}

    public static void main (String args[])
    {
                fb f=new fb();
    double x0;
    x0=1.0;
    double r= secant(f,x0);
    System.out.println(" the root : "+r+"\nFunction value : "+f.func(r));
    }
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" NA8
fx=-1.0dfx=2.6666666666656327x=1.3750000000001454
fx=-0.1093749999996001dfx=3.6666666666906873x=1.4048295454543864
fx=-0.026453948218422063dfx=3.746212121239859x=1.4118910644934275
fx=-0.006563622003616265dfx=3.765042838669664x=1.4136343705552346
fx=-0.0016378663849057062dfx=3.769691654846122x=1.4140688534081756
fx=-4.0927782088773235E-4dfx=3.770850275778651x=1.4141773906851702
fx=-1.023076748833951E-4dfx=3.771139708508286x=1.4142045197980622
fx=-2.55761827323564E-5dfx=3.7712120528314106x=1.414211301751019
fx=-6.393999688381058E-6dfx=3.771230138024013x=1.414212997218931
fx=-1.598497047838876E-6dfx=3.771234659281456x=1.4142134210846387
fx=-3.996240824921671E-7dfx=3.7712357895773128x=1.4142135270509861
fx=-9.990600968734498E-8dfx=3.77123607216608x=1.4142135535425682
fx=-2.497650175570243E-8dfx=3.7712361428058703x=1.4142135601654633
fx=-6.244125438925607E-9dfx=3.771236160480621x=1.414213561821187
fx=-1.5610315262648555E-9dfx=3.771236164877104x=1.414213562235118
fx=-3.902580480996676E-10dfx=3.771236165972524x=1.4142135623386007
fx=-9.756462304721936E-11dfx=3.7712361662537806x=1.4142135623644714
 the root : 1.4142135623644714
Function value : -2.4391377806409764E-11

> Terminated with exit code 0.
```

C++ language code:

```
///Numerical Analysis
//
//
// Secant method with one root estimation
// Linear difference formula
#include <stdio.h>
#include <iostream>
#include <math.h>
using namespace std;

double func(double x) { return x*x-2;}
double dfunc(double x)
```

```
  {double h=1.0e-5;
   double f=0;
   f=(-func(x+2*h)+8.0*func(x+h)-8.0*func(x-h)+func(x+2.0*h))/(12.0*h);
   return f;
  }
double secant(double x)
{
int nmax=100;
double tolerance=1.0e-10;
double fx,dfx;
double dx=0.01;
for(int i=0;i<nmax;i++)
{
  fx=func(x);
  dfx=dfunc(x);
  x-=fx/dfx;
  cout<<"fx="<<fx<<"dfx="<<dfx<<"x="<<x<<endl;
  if(fabs(fx)<tolerance) { return x;}
}
cout<<"Warning : Maximum number of iteration is exceeded \n";
return x;
}

    int main ()
    {
    double x0=1.0;
    double r= secant(x0);
    cout<<" the root : "<<r<<"\nFunction value : "<<func(r);
    }
```

```
---------- Capture Output ----------
> "D:\okul\SCO1\run.bat" NA8

D:\okul\SCO1>NA8
fx=-1dfx=2.66667x=1.375
fx=-0.109375dfx=3.66667x=1.40483
fx=-0.0264539dfx=3.74621x=1.41189
fx=-0.00656362dfx=3.76504x=1.41363
fx=-0.00163787dfx=3.76969x=1.41407
fx=-0.000409278dfx=3.77085x=1.41418
fx=-0.000102308dfx=3.77114x=1.4142
fx=-2.55762e-005dfx=3.77121x=1.41421
fx=-6.394e-006dfx=3.77123x=1.41421
fx=-1.5985e-006dfx=3.77123x=1.41421
fx=-3.99624e-007dfx=3.77124x=1.41421
fx=-9.9906e-008dfx=3.77124x=1.41421
fx=-2.49765e-008dfx=3.77124x=1.41421
fx=-6.24413e-009dfx=3.77124x=1.41421
fx=-1.56103e-009dfx=3.77124x=1.41421
fx=-3.90258e-010dfx=3.77124x=1.41421
fx=-9.75646e-011dfx=3.77124x=1.41421
 the root : 1.41421
Function value : -2.43914e-011
> Terminated with exit code 0.
```

Python language code:

```
 from math import *


def f1(x):

   return x*x+2.0


def df1(x):

   dx=0.001

   return (f1(x+dx)-f1(x-dx))/(2.0*dx)


def secant(x):

   # secant root finding method

  nmax=100
```

```
 tolerance=1.0e-10
 for i in range(0,nmax):
     fx= f1(x);
     dfx=df1(x)
     x=x-fx/dfx
     print("fx",fx,"dfx",dfx,"x=",x)
     if abs(fx)<tolerance: return x
 return x


a=complex(1,1)
r=secant(a)
print(" ROOT  : ",r,"\nFUNCTION VALUE : ",f1(r))
```

```
---------- Capture Output ----------
> "D:\co\python\pythonw.exe" NA8.py
fx (2+2j) dfx (2.0000000000000018+1.9999999999998908j) x= (-2.6867397195928788e-14+0.9999999999999722j)
fx (1.0000000000000555-5.373479439185609e-14j) dfx (-1.1102230246251565e-13+1.9999999999999445j) x= (2.7755575615631993e-14+1.5000000000000138j)
fx (-0.2500000000000413+8.326672684689675e-14j) dfx 3.0000000000000275j x= (-3.1554436208840472e-30+1.4166666666666674j)
fx (-0.006944444444446418-8.940423592504805e-30j) dfx 2.833333333333335j x= 1.4142156862745099j
fx (-6.007304882871267e-06+0j) dfx 2.8284313725490198j x= 1.4142135623746899j
fx (-4.510614104447086e-12+0j) dfx 2.8284271247493797j x= 1.4142135623730951j
 ROOT  :  1.4142135623730951j
FUNCTION VALUE :  (-4.440892098500626e-16+0j)


> Terminated with exit code 0.
```

Octave(Matlab) language code

```
function p=df(f,x)
% derivative of a function
dx=0.001;
p = ( -3*f(x+4*dx)+32*f(x+3*dx)-168*f(x+2*dx)+672*f(x+dx)-672*f(x-dx) + 168*f(x-2*dx)-32*f(x-3*dx)+3*f(x-4*dx) )/(dx*840);
end
```

```
function p=secant(f,x,tolerance,nmax)
% secant method for root finding
if nargin<4 nmax=100;end
if nargin<3 tolerance=1e-10;end
 dx=0.0001;
i=0;
fx=f(x);
dfx=df(f,x);
while (abs(fx) > tolerance) && i<nmax
   fx=f(x);
   dfx=df(f,x);
   x=x-fx/dfx;
   i=i+1;
end
p=x;
if i>=nmax fprintf('Maximum number of iterations is exceeded the result may not be valid');end
endfunction
```

```
f=@(x) x*x-2;
```

```
>> secant(f,1)
ans =1.414213562373095
```

OpenOffice calculate(excel) language code

Secant 2 f(x)=x*x-a

a	2					
x	f(x)=x*x-a	x+Δx	x-Δx	f(x+Δx)	f(x-Δx)	f'(x)
Δx	0,1					
1	-1	1,1	0,9	-0,79	-1,19	2
1,5	0,25	1,6	1,4	0,56	-0,04	3
1,4166666667	0,006944	1,516667	1,316667	0,300278	-0,266389	2,833333
1,4142156863	6,01E-006	1,514216	1,314216	0,292849	-0,272837	2,828431
1,4142135624	4,51E-012	1,514214	1,314214	0,292843	-0,272843	2,828427
1,4142135624	0	1,514214	1,314214	0,292843	-0,272843	2,828427
1,4142135624	0	1,514214	1,314214	0,292843	-0,272843	2,828427
1,4142135624	0	1,514214	1,314214	0,292843	-0,272843	2,828427
1,4142135624	0	1,514214	1,314214	0,292843	-0,272843	2,828427
1,4142135624	0	1,514214	1,314214	0,292843	-0,272843	2,828427
1,4142135624	0	1,514214	1,314214	0,292843	-0,272843	2,828427
1,4142135624	0	1,514214	1,314214	0,292843	-0,272843	2,828427
1,4142135624	0	1,514214	1,314214	0,292843	-0,272843	2,828427
1,4142135624	0	1,514214	1,314214	0,292843	-0,272843	2,828427
1,4142135624	0	1,514214	1,314214	0,292843	-0,272843	2,828427

11. ROOT FINDING: MULLER METHOD

In false position method roots were found by using the root of the linear line passing through the two interval points. The same concept can be use to fit a quadratic curve and find the roots of the quadratic function. But quadratic function fitting is required 3 estimation point. The method consist of deriving the coefficients of a quadratic curve(parabola) that goes through three points. If function evaluation of these 3 points are $f(x_0)$, $f(x_1)$ and $f(x_2)$, enough information will be eveilable to make a quadratic curve root estimation. Assuming quadratic function defined b the following equation :

$$f(x)=a(x-x_2)^2+b(x-x_2)+c \quad (2.6.1)$$

Evaluation of the function in 3 given points will be

$$f(x_0)=a(x_0-x_2)^2+b(x_0-x_2)+c \quad (2.6.2)$$

$$f(x_1)=a(x_1-x_2)^2+b(x_1-x_2)+c \quad (2.6.3)$$

$$f(x_2)=a(x_2-x_2)^2+b(x_2-x_2)+c=c \quad (2.6.4)$$

Considering the difference of the functions:

$$f(x_0)-f(x_2)=a(x_0-x_2)^2+b(x_0-x_2) \quad (2.6.5)$$

$$f(x_1)-f(x_2)=a(x_1-x_2)^2+b(x_1-x_2) \quad (2.6.6)$$

$$h_0=x_1-x_0 \quad (2.6.7)$$

$$h_1=x_2-x_0 \quad (2.6.8)$$

$$d_0=\frac{f(x_1)-f(x_0)}{h_0} \quad (2.6.9)$$

$$d_1 = \frac{f(x_2) - f(x_0)}{h_1} \quad (2.6.10)$$

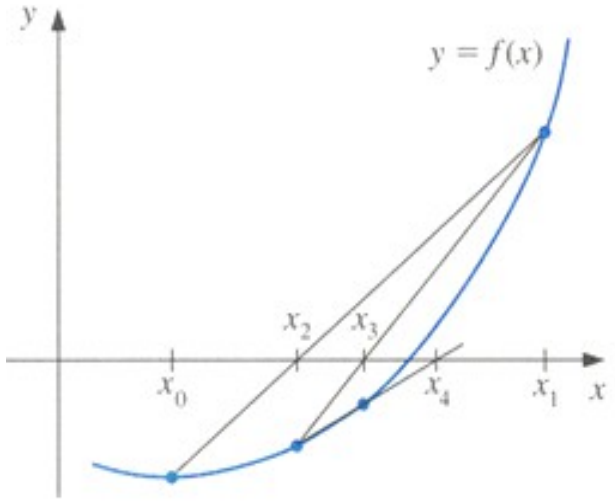


Figure 2.7 : Graphic representation of Muller root finding method.

When these are substituted back to the main equation

$$(h_0 + h_1)b - (h_0 + h_1)^2 a = h_0 d_0 + h_1 d_1 \quad (2.6.11)$$

,

$$h_1 b - h_1^2 a = h_1 d_1 \quad (2.6.12)$$

is obtained. a and b can be solved from here.

$$a = \frac{d_1 - d_0}{h_1 + h_0} \quad (2.6.13)$$

$$b = a h_1 + d_1 \quad (2.6.14)$$

$$c = f(x_2) \quad (2.6.15)$$

to find the root:

$$\Delta = \sqrt{b^2 - 4ac} \quad (2.6.16)$$

if

$$|b + \Delta| > |b - \Delta| \quad set \quad e = b + \Delta \quad (2.6.17)$$

else

$$|b + \Delta| < |b - \Delta| \quad set \quad e = b - \Delta \quad (2.6.18)$$

$$x_r = x_2 - \frac{2c}{e} \quad (2.6.19)$$

$$x_0 = x_1 \quad x_1 = x_2 \quad x_2 = x_r \quad (2.6.20)$$

x_2 will be substituted with x_r and iteration continues. It should be noted that in a quadratic formula complex roots can be located as well as the real roots, therefore this method can be used to find complex roots as well. The program NA10 is developed to calculate real roots by using java language

An example code is created for the Muller method. (Java language)

```
//Numerical Analysis
// Muller method for root finding
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
//============== Function definition ================

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


public class NA10
{
public static double muller(f_x f,double x0,double x1,double x2)
{
//Finding real roots by using Muller method
//orta xr noktasından +/- h kadar giderek diğer 2 noktaya ulaşıyoruz.
int maxit=5;
double xr=x0;
int iter=0;
double es1=0.001;
double es=es1;
double ea=1.1*es;
double h0,h1;
double d0,d1;
double a,b,c;
double fx0,fx1,fx2;
double determinant;
double den;
double dxr;
while(iter<maxit && ea>es)
{
   fx0=f.func(x0);
   fx1=f.func(x1);
   fx2=f.func(x2);
   h0=x1-x0;
   h1=x2-x1;
   d0=(fx1-fx0)/h0;
   d1=(fx2-fx1)/h1;
   a=(d1-d0)/(h1+h0);
   b=a*h1+d1;
   c=fx2;
   determinant=b*b-4.0*a*c;
   if(determinant<=0) determinant=0;//we do not want to calculate complex roots here
   else determinant=Math.sqrt(determinant);
   if(Math.abs(b+determinant)>Math.abs(b-determinant)) {den=b+determinant;;}
   else {den=b-determinant;}
   dxr=-2*c/den;
   xr=x2+dxr;
   System.out.println("a="+a+"b="+b+"c="+c+"disc="+determinant+"xr="+xr+"iter="+iter);
   ea=Math.abs((xr-x2)/xr)*100;
   iter++;
   x0=x1;
   x1=x2;
   x2=xr;
}
if(iter>=maxit) JOptionPane.showMessageDialog(null,"Warning : MAximum number of iteration is exceeded \n"+
  " çözüm geçerli olmıyabilir","MAXİMUM ITERATION NUMBER WARNING",JOptionPane.WARNING_MESSAGE);
return xr;
}

    public static void main (String args[])
    {
            fb f=new fb();
    double x0,x1,x2;
    x0=Double.parseDouble(JOptionPane.showInputDialog(" root estimation x0 : "));
    x1=Double.parseDouble(JOptionPane.showInputDialog(" root estimation x1 : "));
    x2=Double.parseDouble(JOptionPane.showInputDialog(" root estimation x2 : "));
    double r;
    r= muller(f,x0,x1,x2);
    System.out.println(" Value of the root  : "+r+"\nFunction value : "+f.func(r));
```

```
}}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" NA10
a=1.0b=6.0c=7.0disc=2.8284271247461903xr=1.414213562373095iter=0
a=1.0b=2.82842712474619c=-4.440892098500626E-16disc=2.8284271247461903xr=1.4142135623730951iter=1
 Value of the root  : 1.4142135623730951
Function value : 4.440892098500626E-16
> Terminated with exit code 0.
```

C++ language code:

```
/#include <stdio.h>
#include <iostream>
#include <math.h>
using namespace std;

double func(double x) { return x*x-2;}
double muller(double x0,double x1,double x2)
{
//Finding real roots by using Muller method
//orta xr noktasından +/- h kadar giderek diğer 2 noktaya ulaşıyoruz.
int maxit=5;
double xr=x0;
int iter=0;
double es1=0.001;
double es=es1;
double ea=1.1*es;
double h0,h1;
double d0,d1;
double a,b,c;
double fx0,fx1,fx2;
double determinant;
double den;
double dxr;
while(iter<maxit && ea>es)
{
   fx0=func(x0);
   fx1=func(x1);
   fx2=func(x2);
   h0=x1-x0;
   h1=x2-x1;
   d0=(fx1-fx0)/h0;
   d1=(fx2-fx1)/h1;
   a=(d1-d0)/(h1+h0);
   b=a*h1+d1;
   c=fx2;
   determinant=b*b-4.0*a*c;
   if(determinant<=0) determinant=0;//we do not want to calculate complex roots here
   else determinant=sqrt(determinant);
   if(fabs(b+determinant)>fabs(b-determinant)) {den=b+determinant;;}
   else {den=b-determinant;}
   dxr=-2*c/den;
   xr=x2+dxr;
   cout<<"a="<<a<<"b="<<b<<"c="<<c<<"disc="<<determinant<<"xr="<<xr<<"iter="<<iter<<"\n";
   ea=fabs((xr-x2)/xr)*100;
   iter++;
   x0=x1;
   x1=x2;
   x2=xr;
}
if(iter>=maxit) cout<<"Warning : Maximum number of iteration is exceeded \n";
return xr;
}
```

```
---------- Capture Output ----------
> "D:\okul\SCO1\run.bat" NA10

D:\okul\SCO1>NA10
 root estimation x0 : 0
 root estimation x1 : 1
 root estimation x2 : 3
a=1b=6c=7disc=2.82843xr=1.41421iter=0
a=1b=2.82843c=-4.44089e-016disc=2.82843xr=1.41421iter=1
 Value of the root  : 1.41421
```

```
Function value : 4.44089e-016
> Terminated with exit code 0.
```

Python language code:

```
# Muller method for root finding
# python version

from math import *
def f1(x):
  return x*x-2

def muller(x0,x1,x2):
 # Finding real roots by using Muller method
 iter=0;
 es1=0.001;
 es=es1;
 ea=1.1*es;
 maxit = 100;
 xr=x0;
 for iter in range(0,maxit):
  fx0=f1(x0);
  fx1=f1(x1);
  fx2=f1(x2);
  h0=x1-x0;
  h1=x2-x1;
  d0=(fx1-fx0)/h0;
  d1=(fx2-fx1)/h1;
  a=(d1-d0)/(h1+h0);
  b=a*h1+d1;
  c=fx2;
  determinant=b*b-4.0*a*c;
  if determinant<=0:  determinant=0;
  else: determinant=sqrt(determinant);
  if abs(b+determinant)>  abs(b-determinant): den=b+determinant;
  else: den=b-determinant;
  dxr=-2*c/den;
  xr=x2+dxr;
  ea=abs((xr-x2)/xr)*100;
  x0=x1;
  x1=x2;
  x2=xr;
  if ea<es: return xr
 if iter>=maxit:  print('Warning : Maximum number of iteration is exceeded result may not be valis');
 return xr;



x0=float(input("enter first number : "));
x1=float(input("enter second number : "));
x2=float(input("enter third number : "));
r=muller(x0,x1,x2);
print("muller method root  : ",r)
```

```
---------- Capture Output ----------
> "D:\co\python\pythonw.exe" muller.py
enter first number : 0
enter second number : 1
enter third number : 3
muller method root  :  1.4142135623730951

> Terminated with exit code 0.
```

Octave(Matlab) language code

```
function xr=muller(f,x0,x1,x2)
% Finding real roots by using Muller method
%
iter=0;
es1=0.001;
es=es1;
ea=1.1*es;
maxit = 100;
xr=x0;
while (iter<maxit) && (ea>es)
  fx0=f(x0);
```

```
   fx1=f(x1);
   fx2=f(x2);
   h0=x1-x0;
   h1=x2-x1;
   d0=(fx1-fx0)/h0;
   d1=(fx2-fx1)/h1;
   a=(d1-d0)/(h1+h0);
   b=a*h1+d1;
   c=fx2;
   determinant=b*b-4.0*a*c;
   if determinant<=0  determinant=0;
   else determinant=sqrt(determinant);end
   if abs(b+determinant)>  abs(b-determinant) den=b+determinant;
   else den=b-determinant;end
   dxr=-2*c/den;
   xr=x2+dxr;
   ea=abs((xr-x2)/xr)*100;
   iter=iter+1;
   x0=x1;
   x1=x2;
   x2=xr;
end
if iter>=maxit  fprintf('Warning : Maximum number of iteration is exceeded result may not be valis');end
end
```

```
>> f=@(x)x*x-2;
>> y=muller(f,0,1,3)
y =  1.41421356237310
>>
```

OpenOffice calculate(excel) language code

i	x0	x1	x2	f(x0)	f(x1)	f(x2)	h0	h1	d0	d1	a	b	c	Δ	b+Δ	b-Δ	e	xr
0	0	1	3	-2	-1	7	1	2	1	4	1	6	7	2,828	8,828	3,172	8,828	1,41421
1	0	1	1,414	-2	-1	0	1	0,414	1	2,414	1	2,828	0	2,828	5,657	0	5,657	1,41421

12. INTEGRATION : GAUSS-LEGENDRE INTEGRATION FORMULA

QUADRATURES: GAUSS-LEGENDRE INTEGRATION FORMULA

A problem known from antique times, is to create a rectangle with the same area with a polynomial. IT is called a quadrature problem.

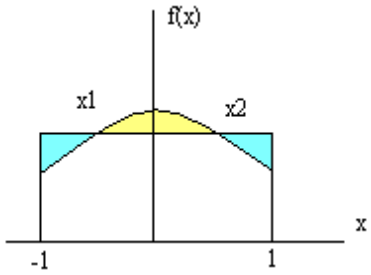


Figure 7.3-1a) a rectangle with the same area with a polynomial

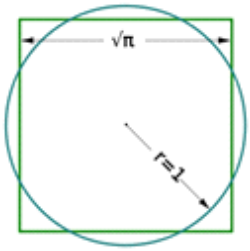


Figure 7.3-1b) A rectangle with the same area with a circle

If point x1 and x2 points intersecting the polynomial and rectangle is known, area calculation of the polynomial can be interchange with the area calculation of the rectangle. Or in more general means, instead of integration process can be substituted by summation process. As a general definition:

$$I_w(-1,1)\simeq\int_{-1}^{1} f(x)w(x)dx\simeq\sum_{k=1}^{n} c_k f(x_k)$$

Can be written. In this equation w(x) is the weight factor. In the first of a such formulation, Gauss-Legendre integral formulation can be used. For Gauss-Legendre integral formulation weight factor can be taken as 1.

$$I_w(-1,1)\simeq\int_{-1}^{1} f(x)dx\simeq\sum_{k=1}^{n} c_k f(x_k)$$

If this integration is solved for a general polynamial equation. Following relation is found:

$$\int_{-1}^{1} x^k dx\simeq\frac{1-(-1)^{(k+1)}}{k+1}$$

This equation can be written in open form as:

$$c_1 x_1^k+c_{\dot{c}}k+...+c_n x_n^k=\frac{1-(-1)^{(k+1)}}{k+1},\ 0\leqslant k\leqslant 2n$$

For a special case of n=2 equation becomes :

$f(x)=a_0+a_1x+a_2x^2+a_3x^3$

$$I_w(-1,1)\simeq\int_{-1}^{1} f(x)dx\simeq c_1 f(x_1)+c_2 f(x_2)=\int_{-1}^{1}(a_0+a_1x+a_2x^2+a_3x^3)dx$$

$$c_1(a_0+a_1x_1+a_2x_1+a_3x_1^3)+c_2(a_0+a_1x_2+a_2x_2+a_3x_2^3)=a_0\int_{-1}^{1}dx+a_1\int_{-1}^{1}xdx+a_2\int_{-1}^{1}x^2dx+a_3\int_{-1}^{1}X^3dx$$

$c_1+c_2=2$

$c_1 x_1+c_2 x_2=0$

$c_1 x_1^2+c_2 x_2^2=2/3$

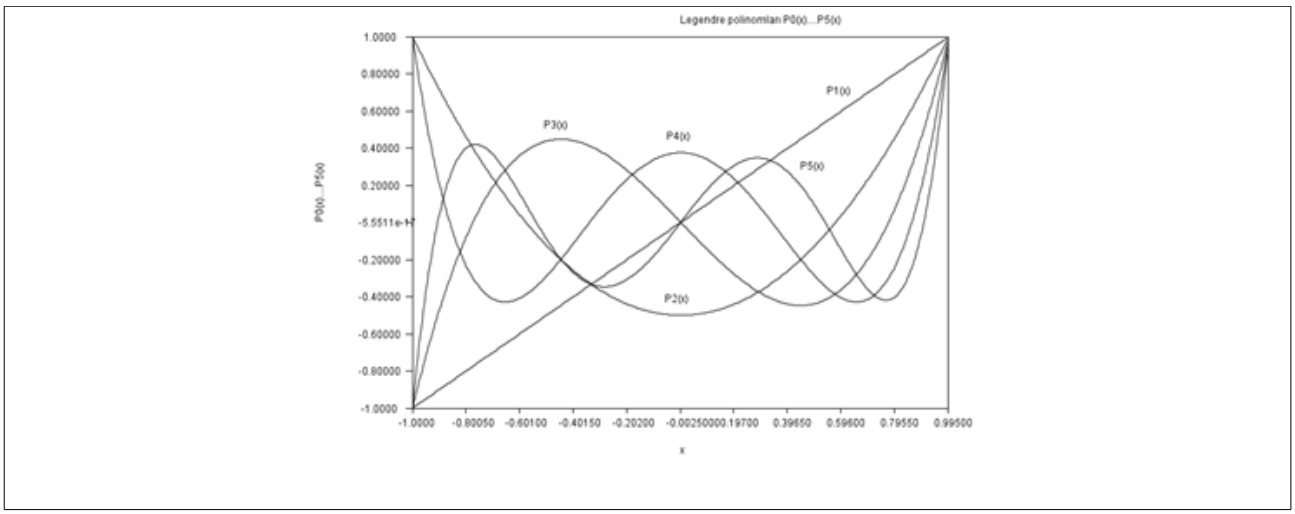
$c_1 x_1^3+c_2 x_2^3=$

If this system of equation is solved roots and coefficients can be found as:

Roots: and Coefficients : $x_{1,2}=\pm\sqrt{3}$ and $c_{1,2}=1$.

The general solution of this problem can be defined with the Legendre polynomials. Legendre polynomials has a general definition as:

$$(k+1)P_k-1(x)=(2k+1)xP_k(x)-kP_k-1(x)\,k\geqslant 1\wedge P_0(x)=1,\ P_1(x)=x$$



The first 6 Legendre polynomials are giving in graphic form.
The roots of Legendre Polynomials are the roots of the Gauss-Legendre integration Formula. For example if $P_2(x)$ value is calculated from the above general form:

$P_2(x)=(3x^2-1)/2$. The root of this is equal to $x_{1,2}=\pm\sqrt{3}$. In general roots can be calculated from

$$\theta_{n,k}=\frac{n-k+3/4}{n+1/2}\pi$$

$$x_{n,k}=[1-\frac{1}{8n^2}+\frac{1}{8n^3}-\frac{1}{384n^4}(39-\frac{28}{\sin^2(\theta_{n,k})})]\cos(\theta_{n,k})+E(n^{-5})$$

This equation contains an error level of $\|n^{-5}\|$ In order to decrease the error Newton-Raphson root finding method can be usefull. After finding the roots coeficients can be calculated as

$$c_k=\frac{2(1-x_k^2)}{[nP_{n-1}(x_k)]^2}$$

As a last concept, consider that Gauss-Legendre integration limits are -1 and 1
Region x=[-1,1] can be converted to z=[a,b] by changing the variables

$$z=(\frac{b-a}{2})x+(\frac{b+a}{2})=\alpha x+\beta$$

In this case, integration Formula will became :

$$I_w(a,b)\simeq\int_a^b f(z)dz=\alpha\sum_{k=1}^{n}c_k(\alpha x_k+\beta)$$

First 10 Gauss-Legendre integration formulas are given in the table

Table 8.4-1 Gauss – Legendre integration roots and coefficients

N	x k	ck
2	-0.577350269189625	1.000000000000000
	0.577350269189625	1.000000000000000
3	-0.774596669241483	0.555555555555552
	0.000000000000000	0.888888888888888
	0.774596669241483	0.555555555555552
4	-0.861136311594052	0.347854845137447
	-0.339981043584856	0.652145154862546
	0.339981043584856	0.652145154862546
	0.861136311594052	0.347854845137447
5	-0.906179845938664	0.236926885056181
	-0.538469310105683	0.478628670499366
	0.000000000000000	0.568888888888888
	0.538469310105683	0.478628670499366
	0.906179845938664	0.236926885056181
6	-0.932469514203152	0.171324492379162

	-0.661209386466264	0.360761573048138
	-0.238619186083196	0.467913934572691
	0.238619186083196	0.467913934572691
	0.661209386466264	0.360761573048138
	0.932469514203152	0.171324492379162
7	-0.949107912342758	0.129484966168862
	-0.741531185599394	0.279705391489276
	-0.405845151377397	0.381830050505119
	0.000000000000000	0.417959183673469
	0.405845151377397	0.381830050505119
	0.741531185599394	0.279705391489276
	0.949107912342758	0.129484966168862
8	-0.960289856497536	0.101228536290369
	-0.796666477413626	0.222381034453374
	-0.525532409916329	0.313706645877887
	-0.183434642495649	0.362683783378362
	0.183434642495649	0.362683783378362
	0.525532409916329	0.313706645877887
	0.796666477413626	0.222381034453374
	0.960289856497536	0.101228536290369
9	-0.968160239507626	0.081274388361569
	-0.836031107326635	0.180648160694857
	-0.613371432700590	0.260610696402935
	-0.324253423403808	0.312347077040002
	0.000000000000000	0.330239355001259
	0.324253423403808	0.312347077040002
	0.613371432700590	0.260610696402935
	0.836031107326635	0.180648160694857
	0.968160239507626	0.081274388361569
10	-0.973906528517171	0.066671344308684
	-0.865063366688984	0.149451349150580
	-0.679409568299024	0.219086362515982
	-0.433395394129247	0.269266719309996
	-0.148874338981631	0.295524224714752
	0.148874338981631	0.295524224714752
	0.433395394129247	0.269266719309996
	0.679409568299024	0.219086362515982
	0.865063366688984	0.149451349150580
	0.973906528517171	0.066671344308684

EXAMPLE (hand calculation) : Calculate integral of function $f(x)=\frac{\pi}{4}\sqrt{1*x^2}$ between limits 0 and 1 with three points Gauss-Legendre integration formula by hand and by computer program

For n=3 α=(1-0)/2=0.5 β=(1+0)/2=0.5

I=0.5*(0.555555555555553 *f(0.5*-0.774596669241483+0.5)+
0.888888888888889*f(0.5*0+0.5)+ 0.555555555555553 *f(0.5*0.774596669241483+0.5))
I=1.004609

Java Example

```
//=======================================================
// Numerical Analysis package in java
// example to show utilisation of integration (integral)
// and differentiation (derivative) functions
```

```
// Gauss-Legendre 10 points integration formula
// Dr. Turhan Coban
// ============================================================
import java.io.*;


class f1 extends f_x
{
  public double func(double x)
  {
  return 4.0/Math.PI*Math.sqrt(1.0-x*x);
  }
}

class NA60
{
public static double integral(f_x f_xnt,double a,double b)
{
//integral f(x)dx
//integral of a function by using gauss-legendre quadrature
//coefficients are pre-calculated for 60 terms for [-1,1]
//band then utilises variable transform
 double r[],c[];
  r=new double[10];
 c=new double[10];
 r[0]=-0.973906528517171;
 r[1]=-0.865063366688984;
 r[2]=-0.679409568299024;
 r[3]=-0.433395394129247;
 r[4]=-0.148874338981631;
 r[5]=0.148874338981631;
 r[6]=0.433395394129247;
 r[7]=0.679409568299024;
 r[8]=0.865063366688984;
 r[9]=0.973906528517171;

 c[0]=0.066671344308684;
 c[1]=0.149451349150580;
 c[2]=0.219086362515982;
 c[3]=0.269266719309996;
 c[4]=0.295524224714752;
 c[5]=0.295524224714752;
 c[6]=0.269266719309996;
 c[7]=0.219086362515982;
 c[8]=0.149451349150580;
 c[9]=0.066671344308684;

int n=10;

double z=0;
double x,y;
double k1=(b-a)/2.0;
double k2=(b+a)/2.0;
double y1=0;
for(int i=0;i<n;i++)
{
x=k2+k1*r[i];
y=f_xnt.func(x);
y1=c[i]*y;
z+=y1;
}
return k1*z;
}

  public static void main(String args[]) throws IOException
  {
  //Gauss-Legendre integration formula with n=10
  f1 b1=new f1();
  System.out.println("integral of class         f1 : "+integral(b1,0.0,1.0));
}
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" NA60
integral of class         f1 : 1.0001612291825637

> Terminated with exit code 0.
```

Python Example

```
# Gauss - Legendre integral 10 coefficients
# python version

from math import *
def f1(x):
   return 4/pi*sqrt(1-x*x)

def integral(a,b):
  #integral f1(x)dx
  #integral of a function by using gauss-legendre quadrature
  #coefficients are pre-calculated for 60 terms for [-1,1]
  #band then utilises variable transform
  r=[-0.973906528517171,-0.865063366688984,-0.679409568299024,-0.433395394129247,-
0.148874338981631,0.148874338981631,0.433395394129247,0.679409568299024,0.865063366688984,0.973906528517171]


c=[0.066671344308684,0.149451349150580,0.219086362515982,0.269266719309996,0.295524224714752,0.295524224714752,0.2692667193099
96,0.219086362515982,0.149451349150580,0.066671344308684]
  n=10;
  z=0;
  k1=(b-a)/2.0;
  k2=(b+a)/2.0;
  for i in range(n):
    x=k2+k1*r[i];
    y=f1(x);
    z=z+k1*c[i]*y;
  return z

a=float(input("enter lower limit : "))
b=float(input("enter upper limit : "))
r=integral(a,b)
print("integral  : ",r)
```

```
---------- Capture Output ----------
> "D:\co\python\pythonw.exe" NA60.py
enter lower limit : 0
enter upper limit : 1
integral  :  1.0001612291825637

> Terminated with exit code 0.
```

C++ Example

```
//=========================================================
// Numerical Analysis package in java
// example to show utilisation of integration (integral)
// and differentiation (derivative) functions
// Gauss-Legendre 10 points integration formula
// Dr. Turhan Coban
// =========================================================
#include <stdio.h>
#include <iostream>
#include <math.h>
using namespace std;


double func(double x)
{return 4.0/M_PI*sqrt(1.0-x*x);}

double integral(double a,double b)
{
//integral f(x)dx
//integral of a function by using gauss-legendre quadrature
//coefficients are pre-calculated for 60 terms for [-1,1]
//band then utilises variable transform
  double r[]={-0.973906528517171,-0.865063366688984,-0.679409568299024,-0.433395394129247,-
0.148874338981631,0.148874338981631,0.433395394129247,0.679409568299024,0.865063366688984,0.973906528517171};
  double
c[]={0.066671344308684,0.149451349150580,0.219086362515982,0.269266719309996,0.295524224714752,0.295524224714752,0.269266719309
9996,0.219086362515982,0.149451349150580,0.066671344308684};
  int n=10;
double z=0;
double x,y;
```

```
double k1=(b-a)/2.0;
double k2=(b+a)/2.0;
double y1=0;
for(int i=0;i<n;i++)
{
x=k2+k1*r[i];
y=func(x);
y1=c[i]*y;
z+=y1;
}
return k1*z;
}

   int main()
   {
  //Gauss-Legendre integration formula with n=10
  cout<<"integral of class         f1 : "<<integral(0.0,1.0);
  return 0;
  }
```

```
---------- Capture Output ----------
> "D:\okul\SCO1\run.bat" NA60

D:\okul\SCO1>NA60
integral of class         f1 : 1.00016
> Terminated with exit code 0.
```

Octave(Mathlab) code

```
 function z=integral(f,a,b)
% Gauss- Legendre integration n=10
  r=[-0.973906528517171,-0.865063366688984,-0.679409568299024,-0.433395394129247,-
0.148874338981631,0.148874338981631,0.433395394129247,0.679409568299024,0.865063366688984,0.973906528517171];


c=[0.066671344308684,0.149451349150580,0.219086362515982,0.269266719309996,0.295524224714752,0.295524224714752,0.2692667193099
96,0.219086362515982,0.149451349150580,0.066671344308684];
  n=10;
  z=0;
  k1=(b-a)/2.0;
  k2=(b+a)/2.0;
  for i =1:10
    x=k2+k1*r(i);
    y=f(x);
    z=z+k1*c(i)*y;
   endfor
endfunction
```

```
>> f=@(x)4/pi()*sqrt(1-x*x)
f =

@(x) 4 / pi () * sqrt (1 - x * x)

>> integral(f,0,1)
ans =  1.0002
>>
```

OpenOffice calculate(Excel) code

N=10		
Gauss Integral Formula		
a		0,0000000000
b		1,0000000000
alpha	(b-a)/2	0,5000000000
beta	(b+a)/2	0,5000000000

rk	Ck	xk=a+b*rk	yk=f(xk)	z=alpha*Ck*yk
-0,9739065285	0,0666713443	0,0130467357	1,2731311764	0,0424406835
-0,8650633667	0,1494513492	0,0674683167	1,2703383629	0,0949268911
-0,6794095683	0,2190863625	0,1602952159	1,2567754345	0,1376711792
-0,4333953941	0,2692667193	0,2833023029	1,2210757598	0,1643975319
-0,148874339	0,2955242247	0,4255628305	1,1521912348	0,1702502107
0,148874339	0,2955242247	0,5744371695	1,0422085462	0,1539989363
0,4333953941	0,2692667193	0,7166976971	0,8879368571	0,1195459222
0,6794095683	0,2190863625	0,8397047841	0,6914240280	0,0757407876
0,8650633667	0,1494513492	0,9325316833	0,4597517249	0,0343552578
0,9739065285	0,0666713443	0,9869532643	0,2050004799	0,0068338288
			Integral	1,0001612292

13. INTEGRATION : GAUSS LEGENDRE INTEGRATION WITH VARIABLE POINTS

Gauss Legendre integration formula can also be carried out by integration of root calculations, so that program can be run free of polynomial degree of accuracy

Java version:

```
//=========================================================
// Numerical Analysis package in java
// example to show utilisation of integration (integral)
// and differentiation (derivative) functions
// desired Legendre polynomials coefficients are internally calculated
// Gauss-Legendre n points integration formula
// Dr. Turhan Coban
// =========================================================
import java.io.*;
class f1 extends f_x
{
  public double func(double x)
  {
  return x*x;
  }
}
class NA61
{
public static double F1(double h,double r)
 {double H=h/r;
  double F=1.0/(H*H+1);
  return F;
 }
public static double[][] gauss_legendre_coefficients(double x1,double x2,int n)
{
              //calculates legendre gauss-coefficients as coefficients of the integral
              //for n terms
              double EPS=3.0e-15;
              int m,j,i;
              double z1,z,xm,xl,pp,p3,p2,p1;
   //double x[]=new double[n];
   //double w[]=new double[n];
   double a[][]=new double[2][n];//a[0][i]=x[i]   a[1][i]=w[i]
              m=(n+1)/2;
              xm=0.5*(x2+x1);
```

```
            xl=0.5*(x2-x1);
            for (i=1;i<=m;i++)
            { z=Math.cos(Math.PI*((i-0.25)/(n+0.5)));
                        do {
                                    p1=1.0;
                                    p2=0.0;
                                    for (j=1;j<=n;j++) {
                                                p3=p2;
                                                p2=p1;
                                                p1=((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j;
                                    }
                                    pp=n*(z*p1-p2)/(z*z-1.0);
                                    z1=z;
                                    z=z1-p1/pp;
                        } while (Math.abs(z-z1) > EPS);
                        a[0][i-1]=xm-xl*z;
                        a[0][n-i]=xm+xl*z;
                        a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp);
                        a[1][n-i]=a[1][i-1];
            }
return a;
}

public static double integral(f_x f_xnt,double x1,double x2,int n)
{
//n : number of integral coefficients
// this routine first generates gauss legendre coefficients
// for [x1,x2] band
// then calculates gauss legendre integral
double a[][]=new double[2][n];
a=gauss_legendre_coefficients(x1,x2,n);
System.out.println(Matrix.toString(a));
Text.print(Text.T(a));
//System.out.println(Matrix.toString(a));
double z=0;
for(int i=0;i<n;i++)
{ z+=a[1][i]*f_xnt.func(a[0][i]);
}
return z;
}

  public static void main(String args[])
  {
  //Gauss-Legendre integral of any desired point (general)
  f1 b1=new f1();

  double F12G=integral(b1,-1.0,1.0,3);
  System.out.println("F12G = "+F12G);
}
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" NA61
      -0.774596669241483       0.000000000000000       0.774596669241483
       0.555555555555553       0.888888888888889       0.555555555555553

F12G = 0.6666666666666633
```

Python version:

```
 Gauss - Legendre integral variable coefficients
# python version

from math import *
def f1(x):
   return x*x

def gauss_legendre_coefficients(x1,x2,n):
    #calculates legendre gauss-coefficients as coefficients of the integral
            #for n terms
            eps=3e-15
            m=int((n+1.0)/2.0)
            xm=0.5*(x2+x1)
            xl=0.5*(x2-x1)
```

```
        nmax=100
        a=[[0.0 for i in range(n)] for j in range(2)]
        for i in range(1,m+1):
                z=cos(pi*((i-0.25)/(n+0.5)))
                for ii in range(nmax):
                #while abs(z-z1) > eps:
                        p1=1.0;
                        p2=0.0;
                        for j in range(1,n+1):
                           p3=p2
                           p2=p1
                           p1=(float)(((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j)
                        pp=(float)(n*(z*p1-p2)/(z*z-1.0))
                        z1=z;
                        z=z1-p1/pp
                        if abs(z-z1) < eps: break
                a[0][i-1]=xm-xl*z
                a[0][n-i]=xm+xl*z
                a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp)
                a[1][n-i]=a[1][i-1]
        return a

def integral(x1,x2,n):
  #integral f1(x)dx
  #integral of a function by using gauss-legendre quadrature
  #between x1 and x2
  a=gauss_legendre_coefficients(x1,x2,n)
  print(a)
  z=0
  for i in range(n):
    z=z+a[1][i]*f1(a[0][i])
  return z

a=-1.0#float(input("enter lower limit : "))
b=1.0#float(input("enter upper limit : "))
r=integral(a,b,3)
print("integral  : ",r)
```

```
---------- Capture Output ----------
> "D:\co\python\pythonw.exe" NA61.py
[[-0.7745966692414834, 0.0, 0.7745966692414834], [0.5555555555555527, 0.8888888888888888, 0.5555555555555527]]
integral  :  0.6666666666666633

> Terminated with exit code 0.
```

Octave (Matlab) version

```
## Copyright (C) 2017 Mustafa Turhan Çoban
## Created: 2017-08-25

function [a] = gauss_legendre_cof (x1,x2,n)
##calculates legendre gauss-coefficients as coefficients of the integral
##for n terms
        eps=3e-15;
  nmax=100;
        m=(n+1.0)/2.0;
        xm=0.5*(x2+x1);
        xl=0.5*(x2-x1);
        for i=1:m
    z=cos(pi*((i-0.25)/(n+0.5)));
                for ii=0:nmax
                   p1=1.0;
                   p2=0.0;
                   for j =1:n
                                p3=p2;
                               p2=p1;
                                p1=((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j;
       endfor
                   pp=n*(z*p1-p2)/(z*z-1.0);
                   z1=z;
                   z=z1-p1/pp;
      if abs(z-z1) < eps break; endif
    endfor
                a(1,i)=xm-xl*z;
```

```
                        a(1,n+1-i)=xm+xl*z;
                        a(2,i)=2.0*xl/((1.0-z*z)*pp*pp);
                        a(2,n+1-i)=a(2,i);
   endfor
endfunction
```

```
 function z=integGLn(f,a,b,n)
% Gauss- Legendre integration n
a=gauss_legendre_cof(a,b,n)
  z=0;
  for i =1:n
    z=z+a(2,i)*f(a(1,i));
   endfor
endfunction
```

```
>> f=@(x)x*x;
>> y=integGLn(f,-1,1,3)
a =

  -0.774596669241483   0.000000000000000   0.774596669241483
   0.555555555555553   0.888888888888889   0.555555555555553

y =  0.666666666666663
>>
```