Lab Report

Course Title: Numerical Methods Laboratory

Course Code: CSE 206

2nd Year 1st Semester Examination 2021

Date of Submission: August 10, 2022

Submitted to-

Dr. Md. Golam Moazzam

Professor

Department of Computer Science and Engineering

Jahangirnagar University

Savar, Dhaka-1342

Class Roll	Exam Roll	Name
406		Md. Nafees Zaman

<u>Contents</u>

Experiment No.: 01

Name of the Experiment:

Determining the root of a non-linear equation using Bisection Method.

Objectives:

- Getting introduced with Bisection Method.
- Determining the roots of non-linear equations in C++.
- Determining the roots of non-linear equations in Microsoft Excel.
- Making comparison of experimental results in C++ and in Microsoft Excel.

Theory:

The bisection method is one of the simplest and most reliable of iterative methods for the solution of nonlinear equations. This method is also known as binary chopping or half interval method. It relies on the fact that if $f(x)$ is real and continuous in the interval $a < x < b$, and $f(a)$ and $f(b)$ are of opposite signs, that is,

$$f(a) \cdot f(b) < 0$$

Then there is at least one real root in the interval between a and b. That is,

$$x_0=(x_1+x_2)/2$$

Now there exist following three conditions:

1. If $f(x_0) = 0$, we have a root at x_0.
2. If $f(x_0) f(x_1) < 0$, there is a root between x_0 and x_1
3. If $f(x_0) f(x_2) < 0$, there is a root between x_0 and x_2

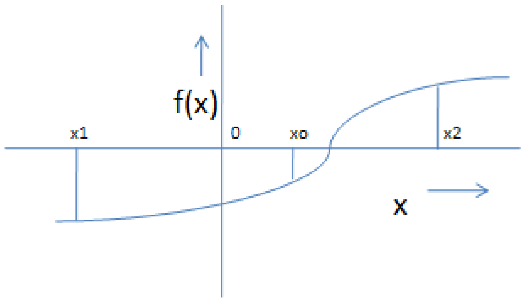


Figure: Illustration of Bisection Method

Algorithm for Bisection Method:

1. Decide initial values for x_1 and x_2 and stopping criterion, E.
2. Computing f1=f(x1) and f2=f(x2)
3. If $f_1 * f_2 > 0$, x_1 and x_2 do not bracket any root and go to step 7.
4. Compute $x_0=(x_1+x_2)/2$ and compute $f_0=f(x_0)$

5. If $f_1*f_0 <0$ then

    set $x_2=x_0$

  else

    set $x_1=x_0$

    set $f_1=f_0$

6. If absolute value of $(x_2 - x_1)/x_2$ is less than error E, then

    root= $(x_1 + x_2)/2$

    write the value of root,

    go to step 7

  else

    go to step 4

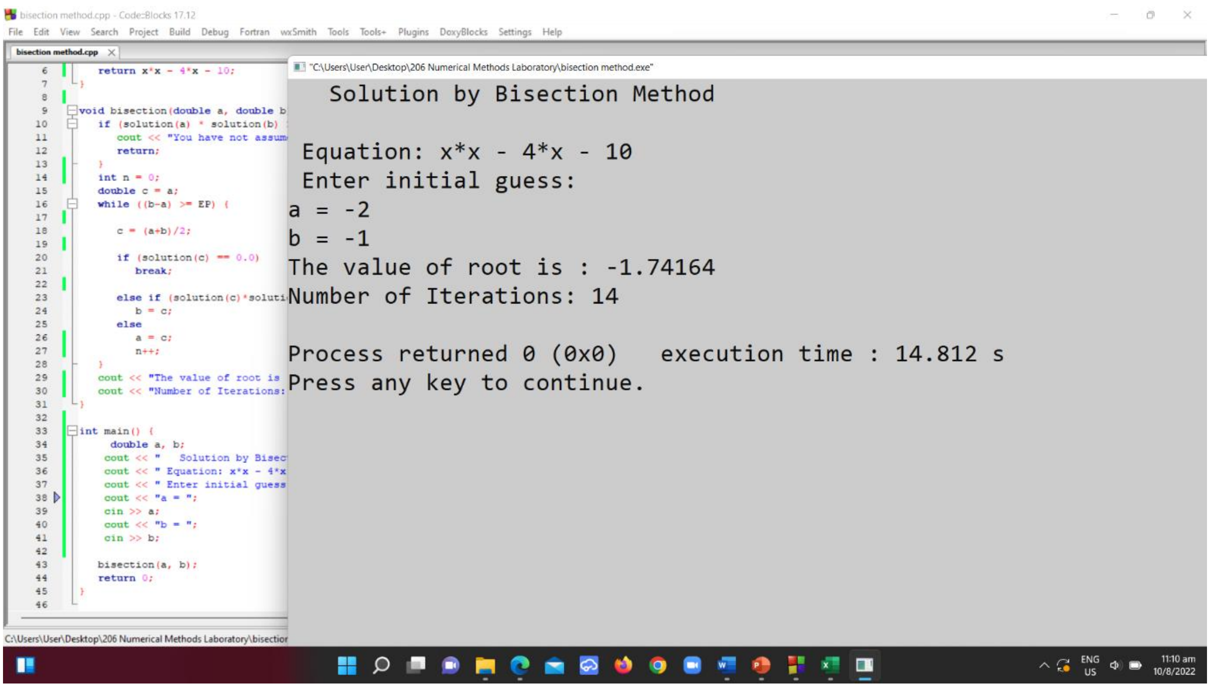
7. Stop.

C++ code of Bisection Method:

```
/* C++ program to find out a real root of the following non-linear equation using Bisection
method:
          x^2 - 4x - 10 = 0
    Done by: Md. Nafees Zaman, Class Roll: 406, Exam Roll: ----
    Date: 10/8/22
*/
#include <iostream>
using namespace std;
#define EP 0.0001

double solution(double x) {
  return x*x - 4*x - 10;
}

void bisection(double a, double b) {
  if (solution(a) * solution(b) >= 0) {
    cout << "You have not assumed right a and b\n";
    return;
  }
  int n = 0;
  double c = a;
  while ((b-a) >= EP) {

    c = (a+b)/2;

    if (solution(c) == 0.0)
      break;

    else if (solution(c)*solution(a) < 0)
      b = c;
    else
      a = c;
```

```
        n++;
    }
    cout << "The value of root is : " << c << endl;
    cout << "Number of Iterations: " << n << endl;
}

int main() {
     double a, b;
    cout << "   Solution by Bisection Method\n\n";
    cout << " Equation: x*x - 4*x - 10\n";
    cout << " Enter initial guess:\n";
    cout << "a = ";
    cin >> a;
    cout << "b = ";
    cin >> b;

   bisection(a, b);
   return 0;
}
```

Output:



Bisection Method in Microsoft Excel:

Experiment Name: Find the root of the following equation using Bisection Method:

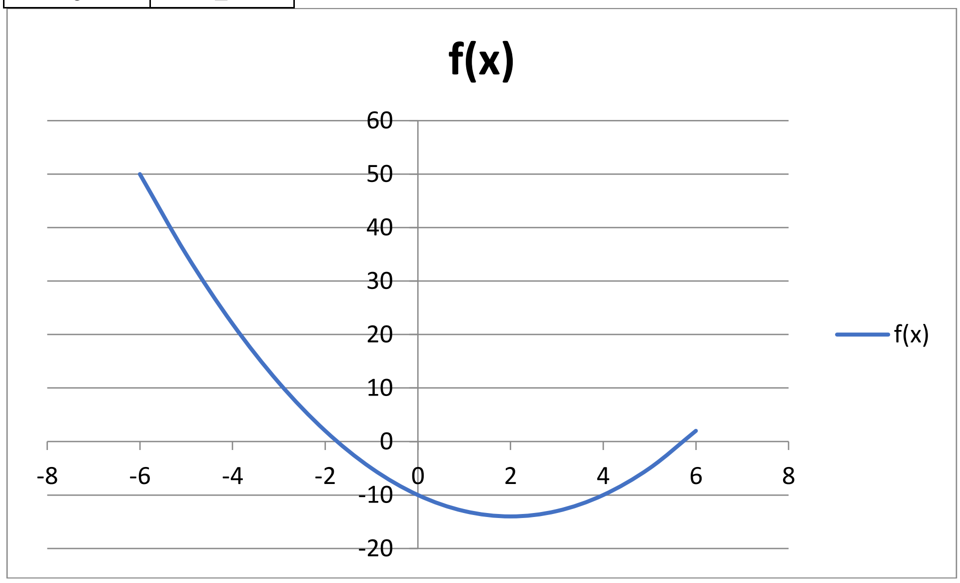
$$f(x) = x^2 - 4x - 10$$

$$Therefore, range\,of\;X = \sqrt{\left(\frac{a_{n-1}}{a_n}\right)^2 - 2\left(\frac{a_{n-2}}{a_n}\right)}$$

$$= 6$$

Ploting the function:

x	f(x)
-6	50
-5	35
-4	22
-3	11
-2	2
-1	-5
0	-10
1	-13
2	-14
3	-13
4	-10
5	-5
6	2



x1	x2	x0	f(x1)	f(x2)	f(x0)	f(x1)f(x0)	f(x2)f(x0)
-2	-1	-1.5	2	-5	-1.75	-3.5	8.75
-2	-1.5	-1.75	2	-1.75	0.0625	0.125	-0.109375
-1.75	-1.5	-1.625	0.0625	-1.75	-0.859375	-0.0537109	1.50390625
-1.75	-1.625	-1.6875	0.0625	-0.859375	-0.4023438	-0.0251465	0.34576416
-1.75	-1.6875	-1.71875	0.0625	-0.402344	-0.1708984	-0.0106812	0.06875992
-1.75	-1.71875	-1.734375	0.0625	-0.170898	-0.0544434	-0.0034027	0.00930429
-1.75	-1.734375	-1.742188	0.0625	-0.054443	0.0039673	0.00024796	-0.000216
-1.742188	-1.734375	-1.738281	0.0039673	-0.054443	-0.0252533	-0.0001002	0.00137487
-1.742188	-1.7382813	-1.740234	0.0039673	-0.025253	-0.0106468	-4.224E-05	0.00026887
-1.742188	-1.7402344	-1.741211	0.0039673	-0.010647	-0.0033407	-1.325E-05	3.5568E-05
-1.742188	-1.7412109	-1.741699	0.0039673	-0.003341	0.000313	1.2419E-06	-1.046E-06
-1.741699	-1.7412109	-1.741455	0.000313	-0.003341	-0.0015139	-4.739E-07	5.0575E-06
-1.741699	-1.7414551	-1.741577	0.000313	-0.001514	-0.0006004	-1.88E-07	9.0901E-07
-1.741699	-1.7415771	-1.741638	0.000313	-0.0006	-0.0001437	-4.499E-08	8.6285E-08
-1.741699	-1.7416382	-1.741669	0.000313	-0.000144	8.467E-05	2.6505E-08	-1.217E-08
-1.741669	-1.7416382	-1.741653	8.467E-05	-0.000144	-2.952E-05	-2.499E-09	4.2417E-09

Result:

After 1st iteration the root is -1.5
After 2nd iteration the root is -1.75
After 3rd iteration the root is -1.625
After 5th iteration the root is -1.71875
After 10th iteration the root is -1.74121
After 15th iteration the root is -1.74167
Approximately the root is -1.74166

Discussion:

The root is not totally accurate. The root has been taken when the interval between x1 and x2 is equal to 1.91E-06. After 20th iteration the difference is 1.91E-06. This is the error of this calculation. The amount of error is too little that it can be avoided. So, -1.74166 can be considered as the root of the equation $x^2 - 4x - 10 = 0$.

Experiment No.: 02

Name of the Experiment:

Determining the root of a non-linear equation using False Position Method.

Objectives:

- Getting introduced with False Position Method.
- Determining the roots of non-linear equations in C++.
- Determining the roots of non-linear equations in Microsoft Excel.
- Making comparison of experimental results in C++ and in Microsoft Excel.

Theory:

We know that equation of the line joining the points (x1, f(x1)) and (x2, f(x2)) is given by

$$\frac{f(x_2) - f(x_1)}{x_2 - x_1} = \frac{y - f(x_1)}{x - x_1}$$

Since the line intersects the x-axis at x0, when x = x0, y = 0, we have

$$\frac{f(x_2) - f(x_1)}{x_2 - x_1} = \frac{0 - f(x_1)}{x_0 - x_1}$$

$$\Rightarrow x_0 - x_1 = -\frac{f(x_1)(x_2 - x_1)}{f(x_2) - f(x_1)}$$

Therefore,

$$x_0 = x_1 - \frac{f(x_1)(x_2 - x_1)}{f(x_2) - f(x_1)}$$

This equation is known as the false position formula.

Algorithm for False Position Method:

1. Decide initial values for x_1 and x_2 and stopping criterion, E.
2. Computing f1=f(x1) and f2=f(x2)
3. If $f_1 * f_2 > 0$, x_1 and x_2 do not bracket any root and go to step 7.
4. Compute

$$x_0 = x_1 - \frac{f(x_1)(x_2 - x_1)}{f(x_2) - f(x_1)}$$

and f0 = f(x0)

5. If $f_1*f_0 > 0$ then

set $x_1=x_0$

else

set $x_2=x_0$

set $f_1=f_0$

6. If f(x0) is less than error E, then

root= f(x0)

write the value of root,

go to step 7

else

go to step 4

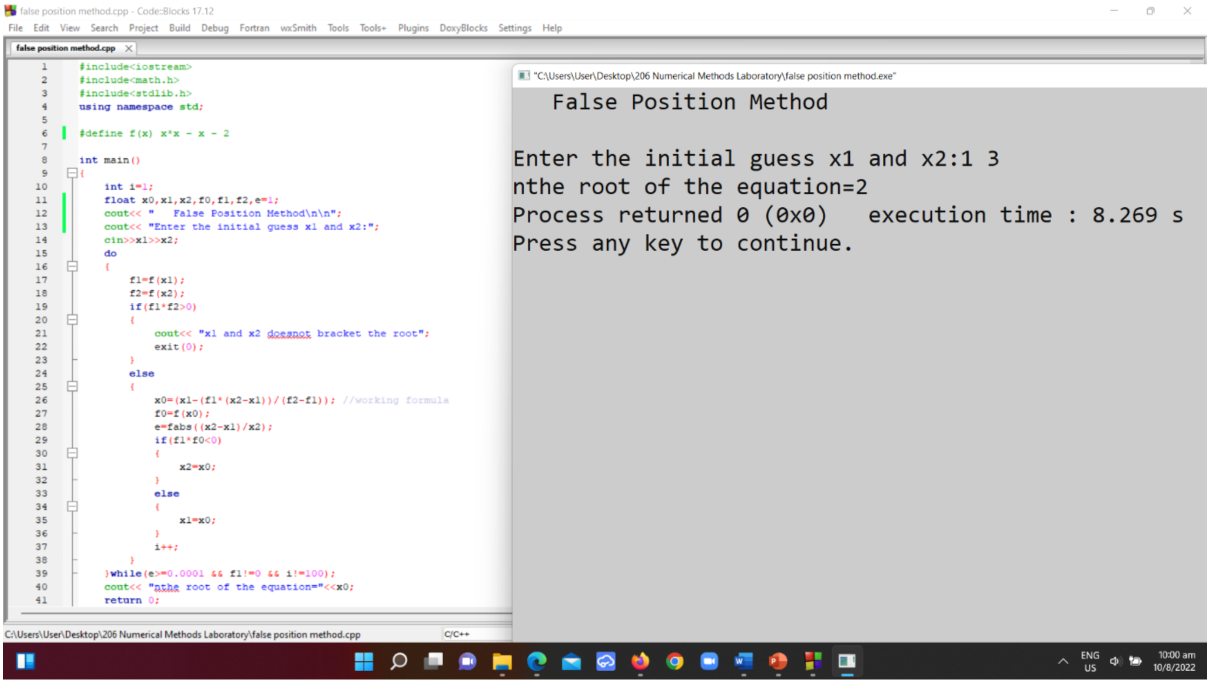
7. Stop.

C++ code of False Position Method:

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

#define f(x) x*x - x - 2

int main()
{
   int i=1;
   float x0,x1,x2,f0,f1,f2,e=1;
   cout<< "   False Position Method\n\n";
   cout<< "Enter the initial guess x1 and x2:";
   cin>>x1>>x2;
   do
   {
      f1=f(x1);
      f2=f(x2);
      if(f1*f2>0)
      {
```

```
        cout<< "x1 and x2 doesnot bracket the root";
        exit(0);
    }
    else
    {
        x0=(x1-(f1*(x2-x1))/(f2-f1));
        f0=f(x0);
        e=fabs((x2-x1)/x2);
        if(f1*f0<0)
        {
            x2=x0;
        }
        else
        {
            x1=x0;
        }
        i++;
    }
  }while(e>=0.0001 && f1!=0 && i!=100);
  cout<< "nthe root of the equation="<<x0;
  return 0;
}
```

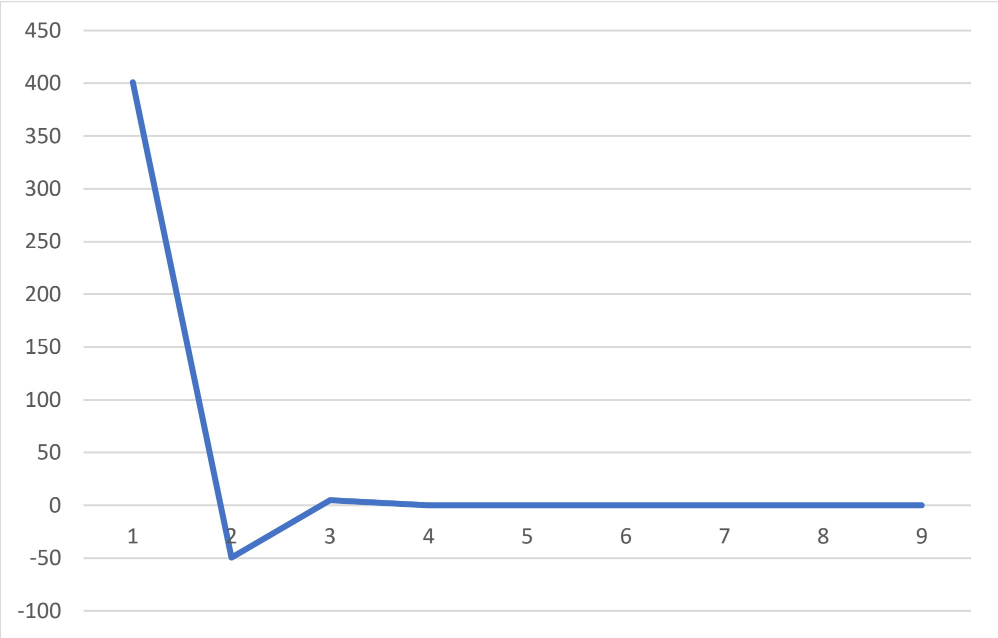
Output:



False Position Method in Microsoft Excel:

	Roots using False Position Method					Equation: x*x- x -2
i	x1	x2	x0	f(x1)	f(x2)	f(x0)
1	1	3	1.005540166	-2	720	400.996923
2	1	1.005540166	-1.222906404	400	400.996923	-49.53628152
3	1	-1.222906404	-0.977955058	400	-49.53628152	4.845027545
4	-0.977955058	-1.222906404	-0.999778659	4.845027545	-49.53628152	0.04869451
5	-0.999778659	-1.222906404	-0.99999778	0.04869451	-49.53628152	0.000488439
6	-0.99999778	-1.222906404	-0.999999978	0.000488439	-49.53628152	4.89928E-06
7	-0.999999978	-1.222906404	-1	4.89928E-06	-49.53628152	4.91421E-08

8	-1	-1.222906404	-1	4.91421E-08	-49.53628152	4.92918E-10
9	-1	-1.222906404	-1	4.92918E-10	-49.53628152	4.94538E-12



Result:

After 1st iteration the root is 1.005
After 2nd iteration the root is -1.222
After 3rd iteration the root is -0.9771
After 7th iteration the root is -1

Root = -1

Discussion:

The root is not totally accurate. The root has been taken when the interval between x1 and x2 is equal to 1.91E-06. After 7th iteration the difference is 1.91E-06. This is the error of this calculation. The amount of error is too little that it can be avoided. So, -1 can be considered as the root of the equation $x^2 - x - 2 = 0$.

Experiment No.: 03

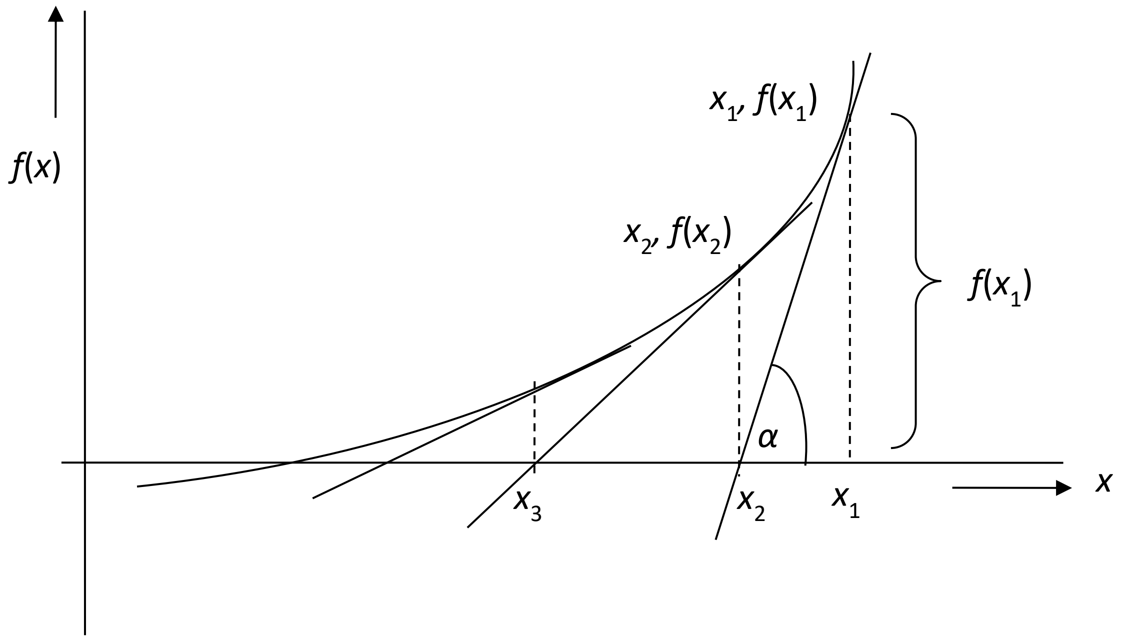
Name of the Experiment:

Determining the root of a non-linear equation using Newton-Raphson Method.

Objectives:

- Getting introduced with Newton-Raphson Method.
- Determining the roots of non-linear equations in C.
- Determining the roots of non-linear equations in Microsoft Excel.
- Making comparison of experimental results in C and in Microsoft Excel.

Theory:



The point of intersection of this tangent with the x-axis gives the second approximation to the root. Let the point of intersection be x2. The slope of the tangent is given by

$$\tan\alpha = \frac{f(x_1)}{x_1 - x_2} = f^{/}(x_1)$$

where f/(x1) is the slope of f(x) at x = x1.

Solving for x2 we obtain

$$x_2 = x_1 - \frac{f(x_1)}{f^{/}(x_1)}$$

This is called the Newton-Raphson formula.

Algorithm for Newton-Raphson Method:

1. Decide initial value for x_0 .
2. Compute f(x1) and =f `(x1)
3. If f(x1) = 0 then

   root= x1

   write the value of root,

   go to step 5

else, Compute

$$x_2 = x_1 - \frac{f(x_1)}{f^{/}(x_1)}$$

4.

   set x_1=x_2

   go to step 3

5. Stop.

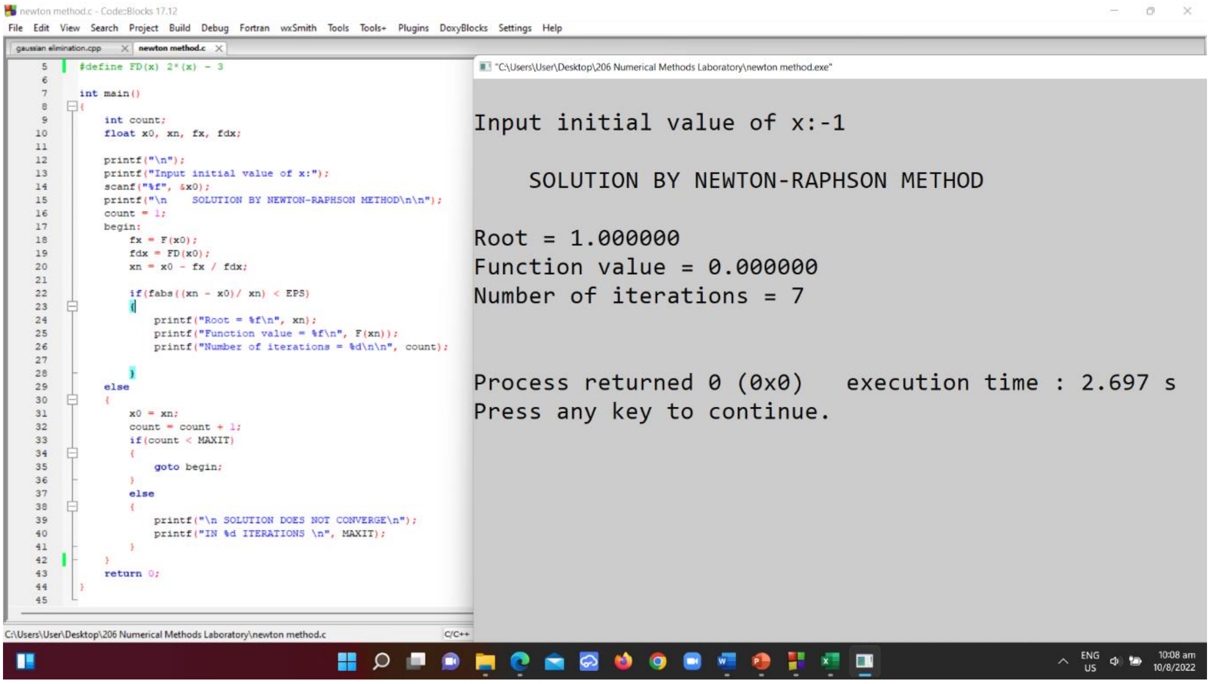
C code of Newton-Raphson Method:

```
#include<math.h>

#define EPS 0.000001

#define MAXIT 20

#define F(x) (x)*(x) - 3*x + 2

#define FD(x) 2*(x) - 3


int main()
```

```
{
  int count;
  float x0, xn, fx, fdx;

  printf("\n");
  printf("Input initial value of x:");
  scanf("%f", &x0);
  printf("\n    SOLUTION BY NEWTON-RAPHSON METHOD\n\n");
  count = 1;
  begin:
    fx = F(x0);
    fdx = FD(x0);
    xn = x0 - fx / fdx;

    if(fabs((xn - x0)/ xn) < EPS)
    {
      printf("Root = %f\n", xn);
      printf("Function value = %f\n", F(xn));
      printf("Number of iterations = %d\n\n", count);

    }
  else
  {
    x0 = xn;
    count = count + 1;
    if(count < MAXIT)
    {
      goto begin;
    }
    else
    {
```

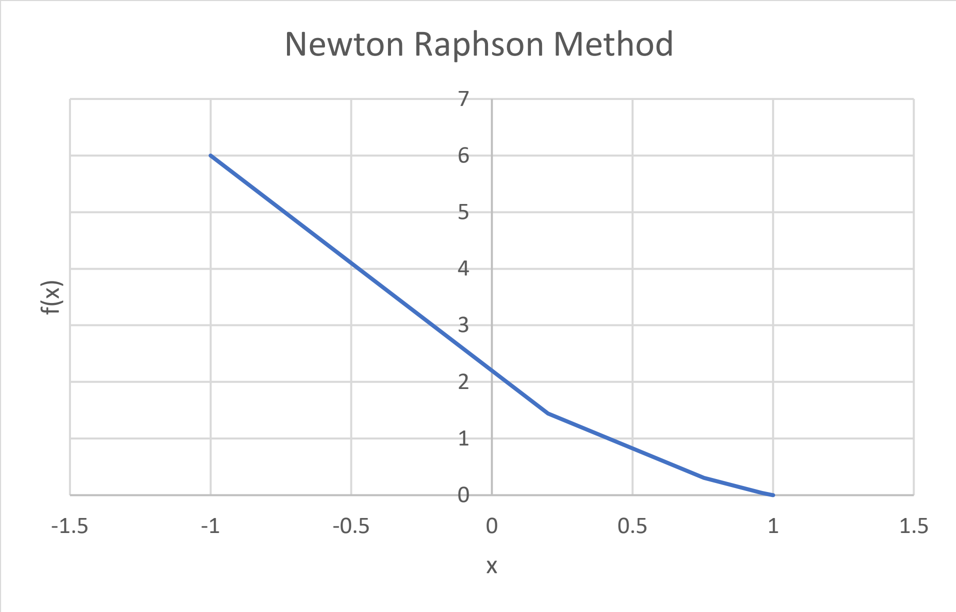
```
            printf("\n SOLUTION DOES NOT CONVERGE\n");
            printf("IN %d ITERATIONS \n", MAXIT);
        }
    }
    return 0;
}
```

Output:



Newton-Raphson Method in Microsoft Excel:

Plot	
x	f(x)
-1	6
0.2	1.44
0.753846	0.306746
0.959397	0.042251
0.998475	0.001527
0.999998	2.32E-06
1	0



Iterations	x0	f(x0)	f'(x0)	xn
1	-1	6	-5	0.2
2	0.2	1.44	-2.6	0.753846
3	0.753846	0.306746	-1.49231	0.959397
4	0.959397	0.042251	-1.08121	0.998475
5	0.998475	0.001527	-1.00305	0.999998
6	0.999998	2.32E-06	-1	1
7	1	0	-1	1

Total number of Iterations: 7

Root = 1

Result:

After 1st iteration the root is 0.2
After 2nd iteration the root is 0.753846
After 3rd iteration the root is 0.959
After 7th iteration the root is 1

Discussion:

The calculations are near accurate. Result have been computed upto six digits after decimal.

Experiment No.: 04

Name of the Experiment:

Solving system of linear equations using Gauss Elimination Method.

Objectives:

- Getting introduced with Gauss Elimination Method.
- Solving system of linear equations in C.
- Solving System of linear equations in Microsoft Excel.
- Making comparison of experimental results in C and in Microsoft Excel.

Theory:

Gauss elimination method proposes a systematic strategy for reducing the system of equations to the upper triangular form using the forward elimination approach and then for obtaining values of unknowns using the back substitution process.

The strategy consists of two phases:

Forward elimination phase: This phase is concerned with the manipulation of equations in order to eliminate some unknowns from the equations and produce an upper triangular system.

Back substitution phase: This phase is concerned with the actual solution of the equations and uses the back substation process on the reduced upper triangular system.

Algorithm for Gauss Elimination Method:

1.Search and locate the largest absolute value among the coefficients in the first column.

2.Exchange the first row with the row containing that element.

3.Then eliminate the first variable in the second equation.

4.When the second row becomes the pivot row, search for the coefficients in the second column from the second row to the nth row and locate the largest coefficient. Exchange the second row with the row containing the large coefficient.

5.Continue this procedure till (n – 1) unknowns are eliminated.

This process is referred to as partial pivoting.

6. Print the values of the roots.

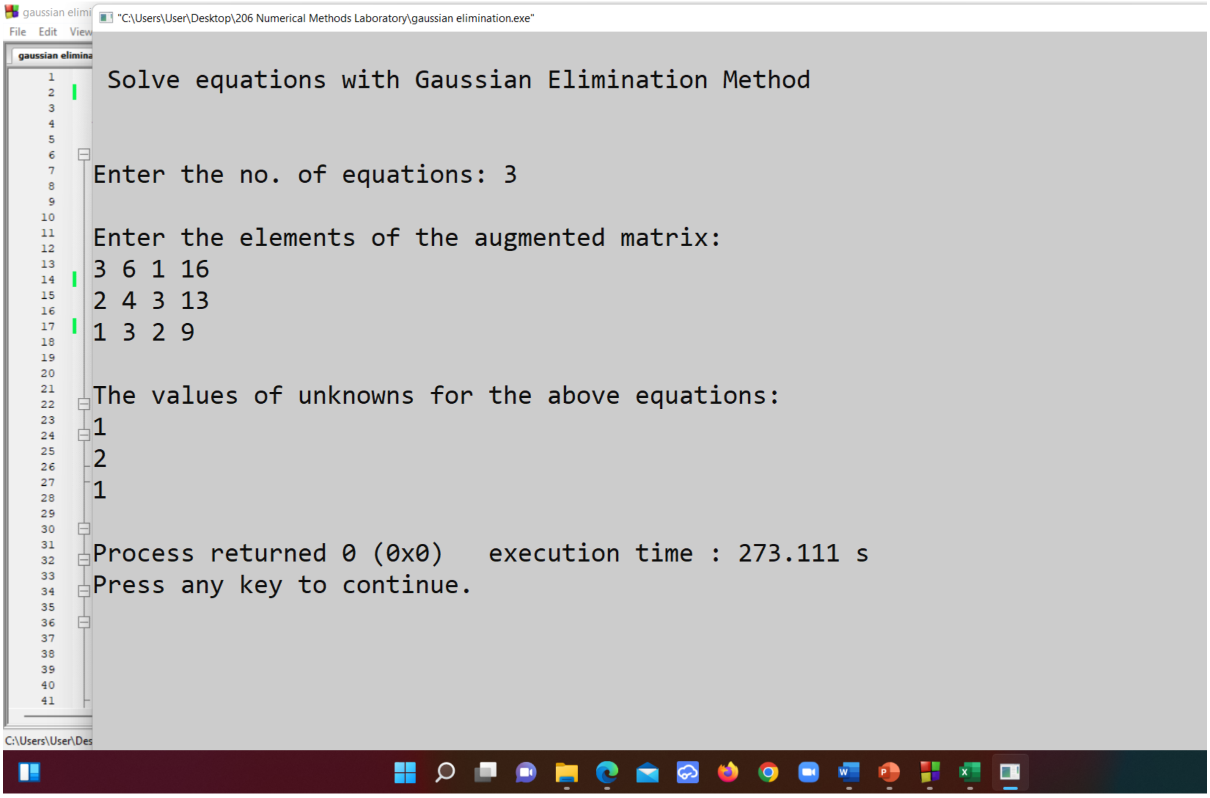
C++ code of Gaussian Elimination:

```
#include<iostream>
#include<bits/stdc++.h>
using namespace std;
int main()
{
   int i,j,k,n;

   cout << "\n Solve equations with Gaussian Elimination Method\n\n";

   cout<<"\nEnter the no. of equations: ";
   cin>>n;



   float mat[n][n+1];



   float res[n];

   cout<<"\nEnter the elements of the augmented matrix: \n";
   for(i=0;i<n;i++)
   {
     for(j=0;j<n+1;j++)
   {
     cin>>mat[i][j];
   }
   }

   for(i=0;i<n;i++)
   {
```

```
    for(j=i+1;j<n;j++)
    {
        if(abs(mat[i][i]) < abs(mat[j][i]))
        {
            for(k=0;k<n+1;k++)
            {
                /* swapping mat[i][k] and mat[j][k] */
    mat[i][k]=mat[i][k]+mat[j][k];
                mat[j][k]=mat[i][k]-mat[j][k];
                mat[i][k]=mat[i][k]-mat[j][k];
            }
        }
  }
}

 /* performing Gaussian elimination */
for(i=0;i<n-1;i++)
{
    for(j=i+1;j<n;j++)
    {
        float f=mat[j][i]/mat[i][i];
        for(k=0;k<n+1;k++)
        {
          mat[j][k]=mat[j][k]-f*mat[i][k];
  }
    }
}

/* Backward substitution for discovering values of unknowns */
for(i=n-1;i>=0;i--)
{
```

```
        res[i]=mat[i][n];

        for(j=i+1;j<n;j++)
        {
         if(i!=j)
         {
            res[i]=res[i]-mat[i][j]*res[j];
    }
  }
 res[i]=res[i]/mat[i][i];
    }

   cout<<"\nThe values of unknowns for the above equations:\n";
   for(i=0;i<n;i++)
   {
     cout<<res[i]<<"\n";
   }
   return 0;
}
```

Output:



Result:

The roots are

x1 = 1

x2 = 2

x3 = 1

Discussion:

The roots are near accurate value, as there was no fractional error in this process.

Experiment No.: 05

Name of the Experiment:

Determining Interpolated function value by Lagrange Interpolation Polynomial.

Objectives:

- Getting introduced with Curve fitting and Lagrange Interpolation.
- Determining interpolated function value in C++.
- Determining interpolated function value in Microsoft Excel.
- Making comparison of experimental results in C++ and in Microsoft Excel.

Theory:

Let x0, x1, ..., xn denote n distinct real numbers and let f0, f1, ..., fn be arbitrary real numbers. The points (x0, f0), (x1, f1), (x2, f2), ..., (xn, fn) can be imagined to be data values connected by a curve. Any function p(x) satisfying the conditions p(xk) = fk for k = 0, 1, …., n is called interpolation function. An interpolation function is, therefore, a curve that passes through the data points as pointed out.

For example, considering a second order polynomial of the form

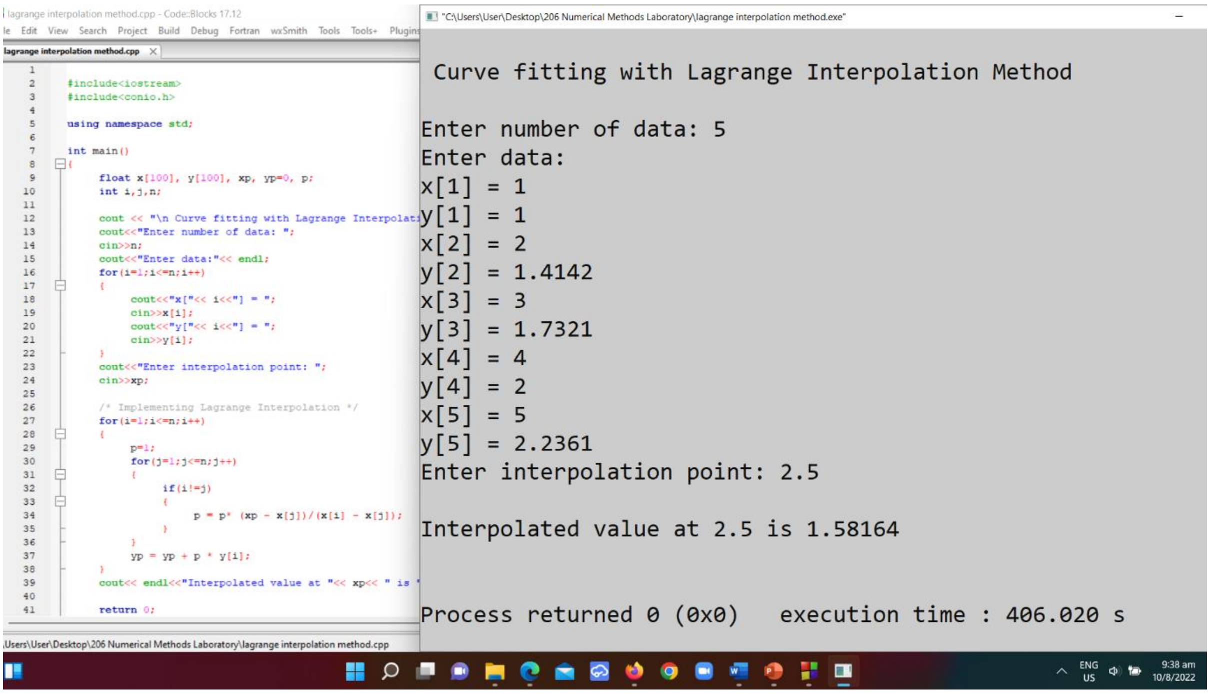
$$p_2(x) = b_1(x - x_0)(x - x_1) + b_2(x - x_1)(x - x_2) + b_3(x - x_2)(x - x_0) \text{..........(1)}$$

C++ code of Lagrange Interpolation:

```
#include<iostream>

#include<conio.h>


using namespace std;


int main()

{

        float x[100], y[100], xp, yp=0, p;

        int i,j,n;


        cout << "\n Curve fitting with Lagrange Interpolation Method\n\n";

        cout<<"Enter number of data: ";

        cin>>n;
```

```
    cout<<"Enter data:"<< endl;
    for(i=1;i<=n;i++)
    {
        cout<<"x["<< i<<"] = ";
        cin>>x[i];
        cout<<"y["<< i<<"] = ";
        cin>>y[i];
    }
    cout<<"Enter interpolation point: ";
    cin>>xp;

    /* Implementing Lagrange Interpolation */
    for(i=1;i<=n;i++)
    {
        p=1;
        for(j=1;j<=n;j++)
        {
            if(i!=j)
            {
                p = p* (xp - x[j])/(x[i] - x[j]);
            }
        }
        yp = yp + p * y[i];
    }
    cout<< endl<<"Interpolated value at "<< xp<< " is "<< yp << endl << endl;

    return 0;
}
```

Output:



Result:

The interpolated value is 1.58164

Experiment No.: 06

Name of the Experiment:

Evaluating integral using Trapezoidal Rule.

Objectives:

- Getting introduced with Trapezoidal rule.
- Solving imtegral in C++.
- Discussion on results.

Theory:

The trapezoidal rule is the first and the simplest of the Newton-Cotes formulae.

Since it is a two point formula, it uses the first order interpolation polynomial p1(x) for approximating the function f(x) and assumes x0=a and x1=b.

This is illustrated in the following figure.This is a process of measuring the area under a curve.

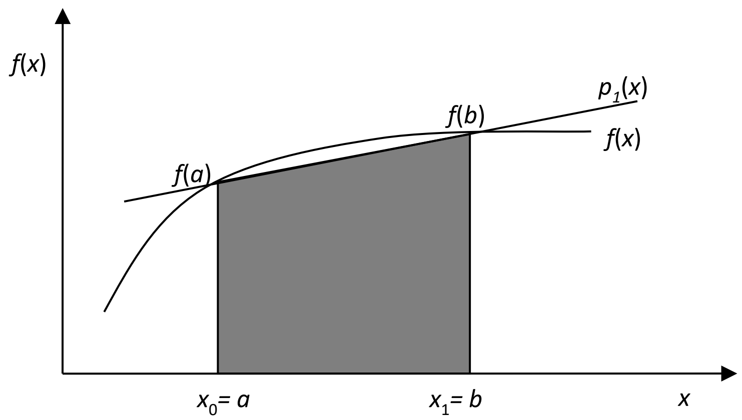


Fig.: Representation of trapezoidal

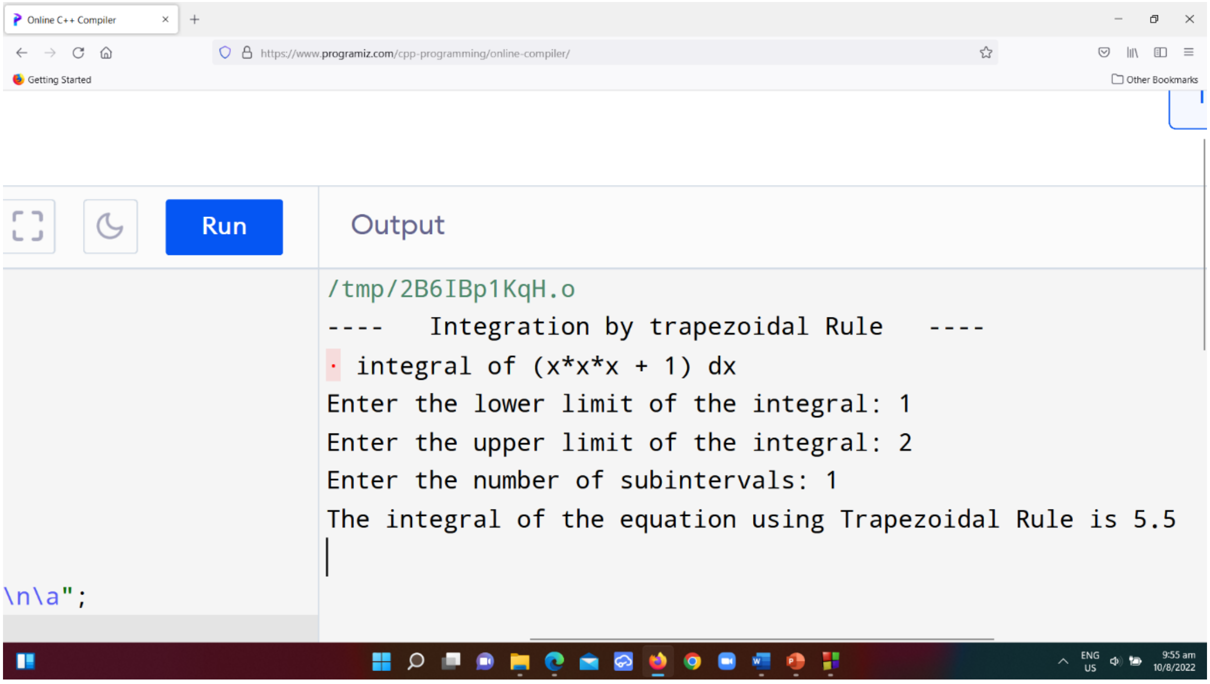
Where,

$$Area = (b-a)\frac{f(a)+f(b)}{2}$$

C++ code of Trapezoidal Rule:

```
#include <iostream>
#include <math.h>
#define f(x) (x*x*x + 1)
using namespace std;
void trapezoidalRule (){
double a, b;
int n;
cout << "\n  ----   Integration by trapezoidal Rule   ----\n\n\a";
cout << " integral of (x*x*x + 1) dx\n";
cout << "Enter the lower limit of the integral: ";
cin >> a;
cout << "Enter the upper limit of the integral: ";
cin >> b;
cout << "Enter the number of subintervals: ";
cin >> n;
double h = abs(b - a) / n;
double ifx = 0;
ifx = ifx + f(a) + f(b);
for (double i = a+h; i < b;){
ifx = ifx + (2 * f(i));
i = i + h;
}
ifx = ifx * h / 2;
cout << "\nThe integral of the equation using Trapezoidal Rule is " << ifx << endl;
}
int main (){
trapezoidalRule();
}
```

Output:



Result:

Calculated area under curve is 5.5

Discussion:

The result is obtained from Trapezoidal rule.

Experiment No.: 07

Name of the Experiment:

Evaluating integral using Simpson's 1/3 Rule.

Objectives:

- Learning to apply Simpson's 1/3 rule.
- Determining integrations in C++.
- Determining the roots of non-linear equations in Microsoft Excel.
- Comparing experimental results in C++ and in Microsoft Excel.

Theory:

Simpson's 1/3 is a popular method for numerical integrations.

The width h is given by

$$h = (b - a)/2$$

Newton-Gregory forward formula (polynomial):

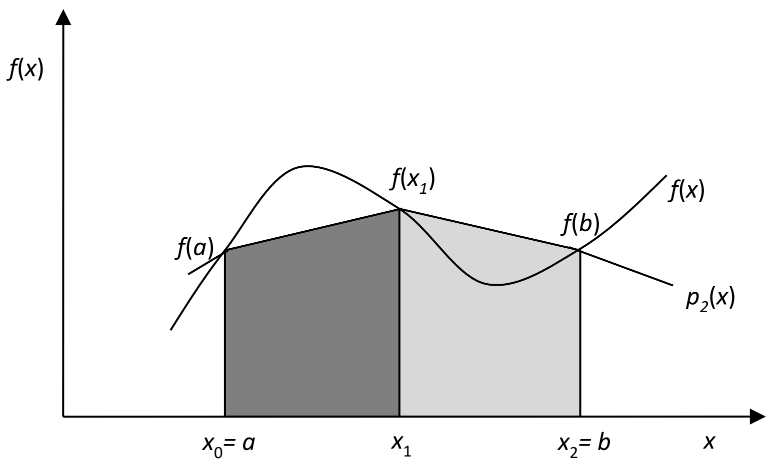


Fig.: Representation of Simpson's 1/3 rule

Where,

$$I_s = 2hf_0 + 2h\Delta f_0 + \frac{h\Delta^2 f_0}{3}$$

$$= h\left[2f_0 + 2\Delta f_0 + \frac{\Delta^2 f_0}{3}\right]$$

$$= h\left[2f_0 + 2(f_1 - f_0) + \frac{(f_2 - 2f_1 + f_0)}{3}\right]$$

$$= \frac{h}{3}\left[6f_0 + 6(f_1 - f_0) + (f_2 - 2f_1 + f_0)\right]$$

$$= \frac{h}{3}\left[f_0 + 4f_1 + f_2\right] = \frac{h}{3}\left[f(a) + 4f(x_1) + f(b)\right]$$

Therefore,

$$I_s = \frac{(b-a)}{6}\left[f_0 + 4f_1 + f_2\right] = \frac{h}{3}\left[f(a) + 4f(x_1) + f(b)\right] \quad \text{..................} \; (1)$$

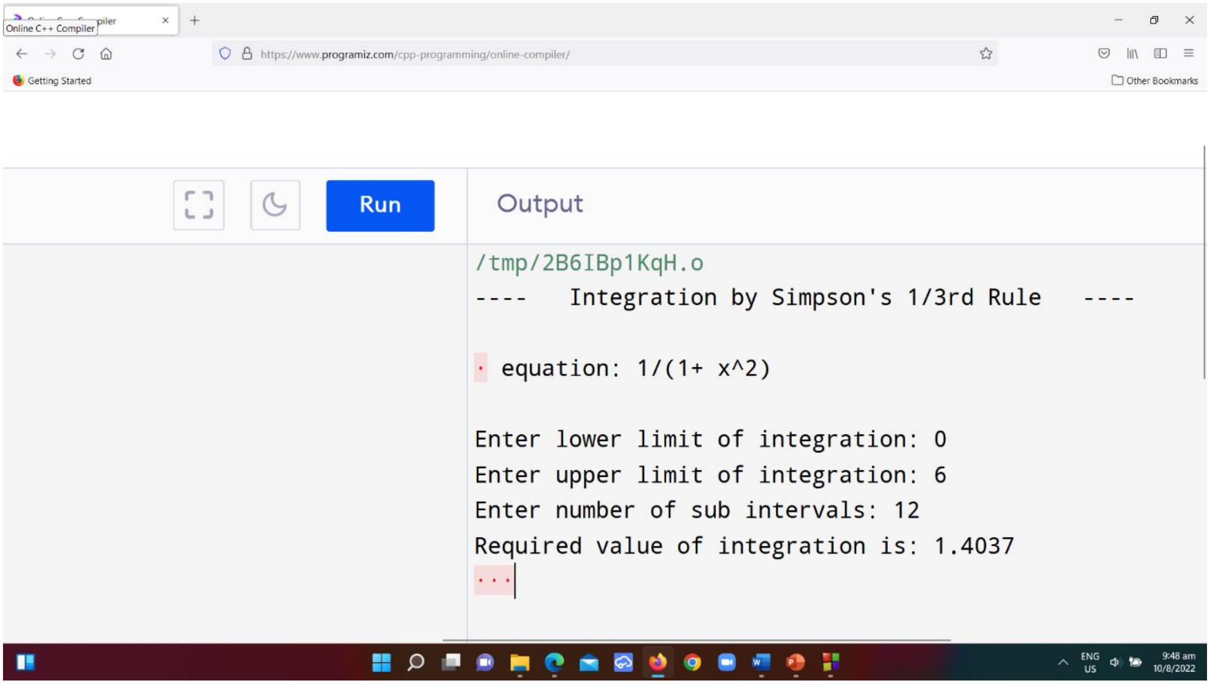
This equation is called Simpson's 1/3 rule. This shows that the area is given by the product of total width of the segments and the weighted average of heights f(a), f(x1) and f(b).

C++ code of Simpson's 1/3 Rule:

```
#include<iostream>
#include<math.h>

#define f(x) 1/(1+pow(x,2))

using namespace std;
int main()
{
 float lower, upper, integration=0.0, stepSize, k;
 int I, subInterval;
cout << "\n  ----   Integration by Simpson's 1/3rd Rule   ----\n\n\a";
cout << " equation: 1/(1+ x^2)\n" << endl;
cout<<"Enter lower limit of integration: ";
```

```
 cin>>lower;
 cout<<"Enter upper limit of integration: ";
 cin>>upper;
 cout<<"Enter number of sub intervals: ";
 cin>>subInterval;

stepSize = (upper – lower)/subInterval;

integration = f(lower) + f(upper);

 for(i=1; i<= subInterval-1; i++)
 {
  k = lower + i*stepSize;

  if(i%2==0)
  {
    integration = integration + 2 * (f(k));
  }
  else
  {
    integration = integration + 4 * (f(k));
  }

 }

 integration = integration * stepSize/3;

 cout<< endl <<"Required value of integration is: "<< integration << "\n\a\a\a";

 return 0;
}
```

Output:



Result:

Calculated Area = 1.4037.

Discussion:

The area is computed through Simpson's 1/3rd rule.

-----*-----