Lab Report

Title: NUMERICAL METHODS
Course code: CSE-206
2nd Year 1st Semester

Date of Submission: **20.07.2022**

Submitted to-

Dr. Md. Golam Moazzem
Professor
Department of Computer Science and Engineering
Jahangirnagar University
Savar, Dhaka-1342

Sl	Class Roll	Registration Number	Name
01	388	20200650758	Md.Tanvir Hossain Saon

Experiment Number	**Name of the Experiment**
01	Bisection Method
02	False Position Method
03	Newton Raphson Method
04	Gauss Elimination Method
05	Lagrange Interpolation Method
06	Trapezoidal Rule
07	Simpson’s ⅓ Rule

Experiment No.01

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

OBJECTIVES :

1.Getting introduced with Bisection Method.

2.Determining the roots of non-linear equations in C++.

3.Determining the roots of nonlinear equations in Microsoft Excel.

4.Making comparison of experimental results in C++ and in Microsoft Excel.

ALGORITHM :

1. Decide initial values for x1 and x2 and stopping criterion E.

2. Compute f1 = f (x1) and f2 = f (x2).

3. If f1 * f2> 0, x1 and x2 do not bracket any root and go to step 1.

4. Compute x0 = (x1 + x2) / 2 and compute f0 = f (x0).

5. If f0 = 0 then x0 is the root of the equation, print the root

6. If f1 * f0< 0 then set x2 = x0 else set x1 = x0.

7. If |(x2-x1)/x2|< E then root = (x1 + x2) / 2, print the root and go to step 8

Else go to step 4

8. Stop.

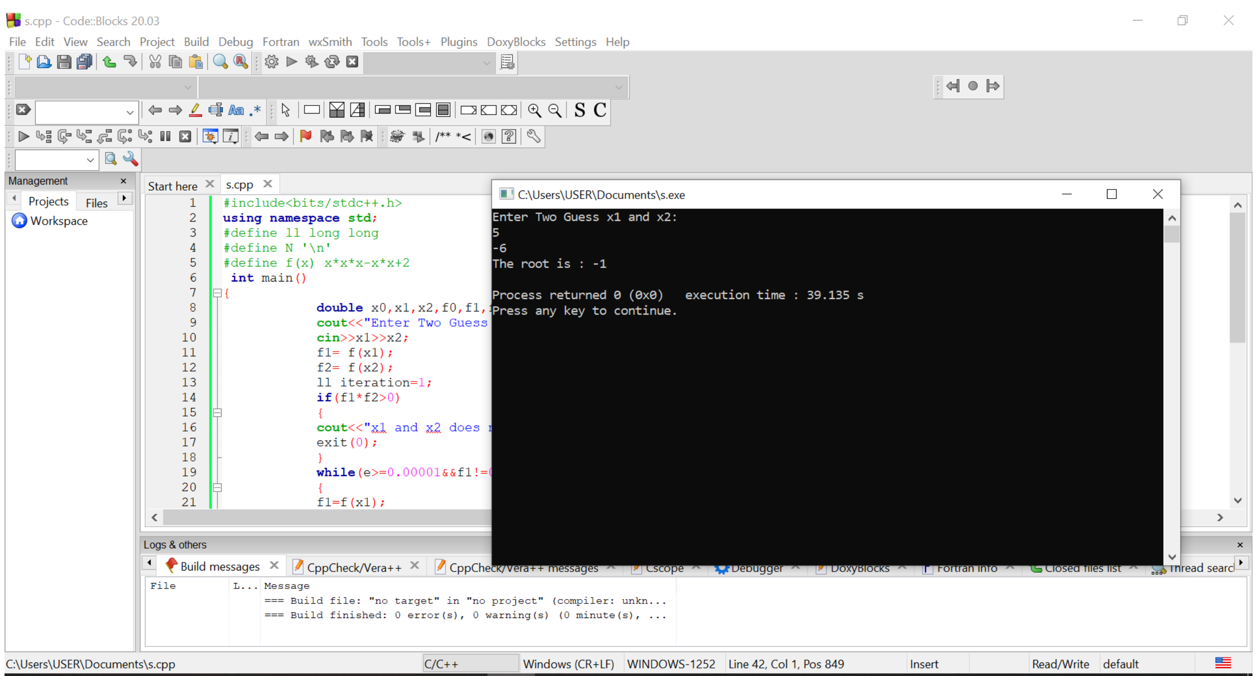
Source Code:

```
#include<bits/stdc++.h>

using namespace std;

#define ll long long

#define N '\n'

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

```
int main()
{
        double x0,x1,x2,f0,f1,f2,e=1;
        cout<<"Enter Two Guess x1 and x2: "<<endl;
        cin>>x1>>x2;
        f1= f(x1);
        f2= f(x2);
        ll iteration=1;
        if(f1*f2>0)
        {
                cout<<"x1 and x2 does not bound any root"<<endl;
                exit(0);
        }
        while(e>=0.00001&&f1!=0&&iteration!=100)
        {
                f1=f(x1);
                f2=f(x2);
                x0=(x1+x2)/2;
                f0=f(x0);
                e=fabs((x2-x1)/x2);
                if(f0==0.0)
                {
                break;
                }
                if(f1*f0<0)
                {
```

```
                x2=x0;
                }
                else
                {
                x1=x0;
                }
                iteration++;
        }
        cout<<"The root is : "<<x0<<N;
}
```

Output:



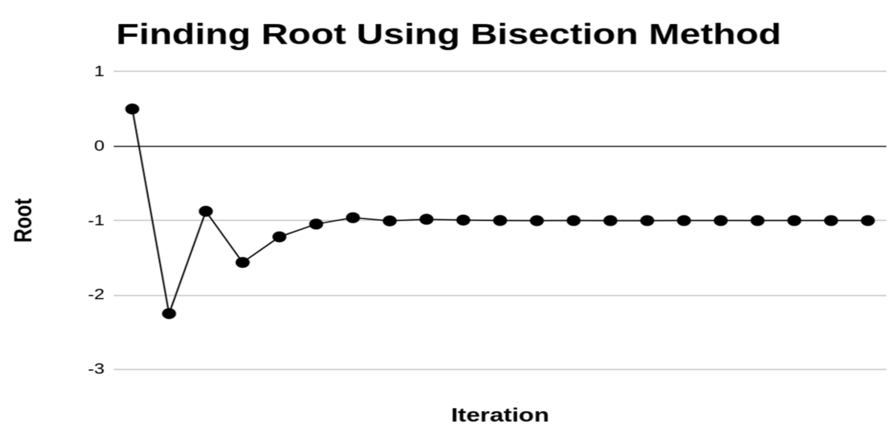
Bisection Method in Microsoft Excel:

Find the root of the following equation using Bisection Method:

F(x)=x^3-x^2+2

x1	x2	x0	F(x1)	F(x2)	F(x0)
-5	6	0.5	-148	182	1.875
-5	0.5	-2.25	-148	1.875	-14.4531
-2.25	0.5	-0.875	-14.4531	1.875	0.564453
-2.25	-0.875	-1.5625	-14.4531	0.564453	-4.2561
-1.5625	-0.875	-1.21875	-4.2561	0.564453	-1.29562
-1.21875	-0.875	-1.04688	-1.29562	0.564453	-0.24327
-1.04688	-0.875	-0.96094	-0.24327	0.564453	0.189269
-1.04688	-0.96094	-1.00391	-0.24327	0.189269	-0.01959
-1.00391	-0.96094	-0.98242	-0.01959	0.189269	0.08666
-1.00391	-0.98242	-0.99316	-0.01959	0.08666	0.033993
-1.00391	-0.99316	-0.99854	-0.01959	0.033993	0.007316
-1.00391	-0.99854	-1.00122	-0.01959	0.007316	-0.00611
-1.00122	-0.99854	-0.99988	-0.00611	0.007316	0.00061
-1.00122	-0.99988	-1.00055	-0.00611	0.00061	-0.00275
-1.00055	-0.99988	-1.00021	-0.00275	0.00061	-0.00107
-1.00021	-0.99988	-1.00005	-0.00107	0.00061	-0.00023
-1.00005	-0.99988	-0.99996	-0.00023	0.00061	0.000191
-1.00005	-0.99996	-1	-0.00023	0.000191	-1.9E-05
-1	-0.99996	-0.99998	-1.9E-05	0.000191	8.58E-05
-1	-0.99998	-0.99999	-1.9E-05	8.58E-05	3.34E-05
-1	-0.99999	-1	-1.9E-05	3.34E-05	7.15E-06

Graph:



Result:

After 1 st iteration the root is 0.5

After 2 nd iteration the root is -2.25

After 3 rd iteration the root is -1.5625

After 4 th iteration the root is -1.21875

After 5 th iteration the root is -1.04688

After 6 th iteration the root is -0.96094

Approximately the root is -1

Discussion:

The root is not totally accurate. The root has been taken when the interval between x1 and x2 . The amount of error is so little that it can be avoided. So, -1 can be considered as the root of the equation.

Experiment No.02

Experiment Name: Finding root using False Position Method.

OBJECTIVES: To find the roots of nonlinear equations using the False Position method.

ALGORITHM:

1. Decide initial values for x1 and x2 and stopping criterion E.

2. Compute f1 = f (x1) and f2 = f (x2).

3. If f1 * f2> 0, x1 and x2 do not bracket any root and go to step 1.

4. Compute x0 = x1- (f(x1) (x2-x1)) / (f (x2) – f (x1)) and compute f0 = f (x0).

5. If f0 = 0 then x0 is the root of the equation, print the root

6. If f1 * f0< 0 then set x2 = x0 else set x1 = x0.

7. If |(x2-x1)/x2|< E then root = (x1 + x2) / 2, print the root and go to step 8

Else go to step 4

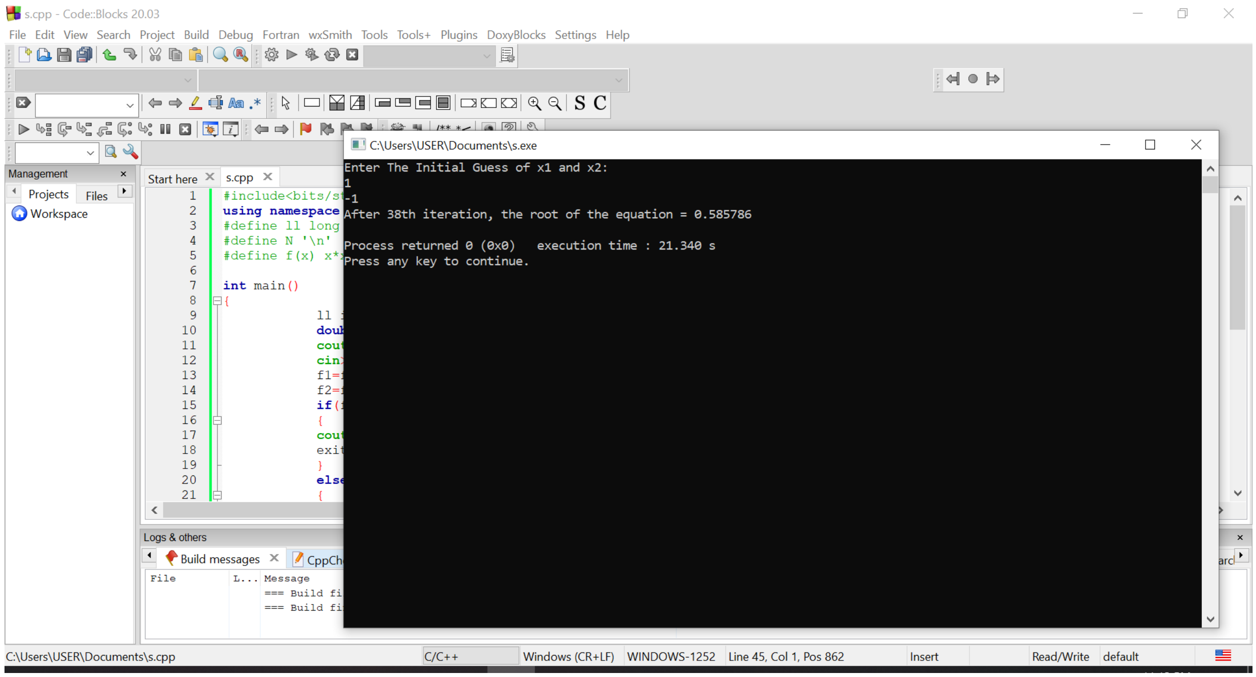
8. Stop.

Source Code:

```
#include<bits/stdc++.h>

using namespace std;

#define ll long long

#define N '\n'

#define f(x) x*x-4*x+2


int main()

{

        ll i=1;

        double x0,x1,x2,f0,f1,f2,e=1;

        cout<<"Enter The Initial Guess of x1 and x2: "<<endl;

        cin>>x1>>x2;

        f1=f(x1);

        f2=f(x2);

        if(f1*f2>0)

        {

                cout<<"x1 and x2 does not bracket the root"<<endl;

                exit(0);

        }

        else

        {

                while(e>=0.0001&&f1!=0&&i!=100)
```

```
{
f1=f(x1);
f2=f(x2);
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++;
}
cout<<"After "<<i<<"th iteration,"<<" the root of the equation = "<<x0<<N;

}
}
```

Output:



False Position Method Using Microsoft Excel:

Find the root of the following equation using false position Method

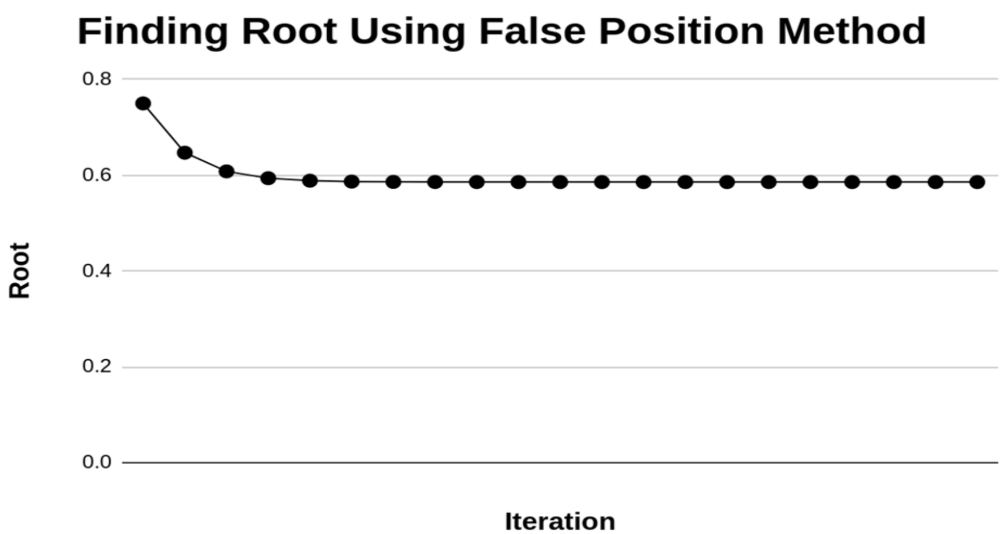
F(x)=x^2-4x+2

x1	x2	x0	F(x1)	F(x2)	F(x0)
-1	1	0.75	7	-1	-0.4375
-1	0.75	0.6470588235	7	-0.4375	-0.169550173
-1	0.6470588235	0.6081081081	7	-0.169550173	-0.06263696129
-1	0.6081081081	0.5938461538	7	-0.06263696129	-0.02273136095

-1	0.5938461538	0.5886871508	7	-0.02273136095	-0.00819604179
-1	0.5886871508	0.5868291911	7	-0.00819604179	-0.002948264793
-1	0.5868291911	0.5861611306	7	-0.002948264793	-0.001059651489
-1	0.5861611306	0.5859210558	7	-0.001059651489	-0.000380739682
-1	0.5859210558	0.5858348001	7	-0.000380739682	-0.0001367873672
-1	0.5858348001	0.5858038118	7	-0.0001367873672	-0.00004914132538
-1	0.5858038118	0.5857926793	7	-0.00004914132538	-0.00001765393986
-1	0.5857926793	0.5857886799	7	-0.00001765393986	-0.000006342116705
-1	0.5857886799	0.5857872432	7	-0.000006342116705	-0.000002278379315
-1	0.5857872432	0.585786727	7	-0.000002278379315	-0.0000008184978557
-1	0.585786727	0.5857865416	7	-0.0000008184978557	-0.0000002940417243
-1	0.5857865416	0.585786475	7	-0.0000002940417243	-0.0000001056331769
-1	0.585786475	0.585786451	7	-0.0000001056331769	-0.00000003794824632
-1	0.585786451	0.5857864424	7	-0.00000003794824632	-0.00000001363273672
-1	0.5857864424	0.5857864394	7	-0.00000001363273672	-0.000000004897500183

-1	0.5857864394	0.5857864382	7	-0.000000004897500183	-0.000000001759404622
-1	0.5857864382	0.5857864379	7	-0.000000001759404622	-0.0000000006320575174

Graph:



Result: The approximate Root is 0.5857864379.

Discussion:

The root is not totally accurate . The amount of error is so little that it can be avoided. So, 0.5857864379 can be considered as the root of the equation.

Experiment No.03

Experiment Name: Newton-Raphson Method.

OBJECTIVES : Find the roots of nonlinear equations using Newton-Raphson method.

ALGORITHM:

1. Assign an initial value for x, say x0 and stopping criterion E.

2. Compute f (x0) and f ‘(x0).

3. Find the improved estimate of x0

x1 = x0 – f (x0) / f ’(x0)

4. Check for accuracy of the latest estimate.

If | (x1-x0)/x1| < E then stop; otherwise continue.

5. Replace x0 by x1 and repeat steps 3 and 4.

Source Code:

```
#include<bits/stdc++.h>

using namespace std;

#define f(x) x*x+4*x+4

#define g(x) 2*x+4

int main()

{

    float x0, x1, f0, f1, g0, e;

    int step = 1, N;

            cout<< setprecision(6)<< fixed;

    cout<<"Enter initial guess: ";

    cin>>x0;

    cout<<"Enter tolerable error: ";

    cin>>e;

    cout<<"Enter maximum iteration: ";

    cin>>N;
```

```
cout<< endl<<"********************"<< endl;

cout<<"Newton Raphson Method"<< endl;

cout<<"********************"<< endl;

do

{

        g0 = g(x0);

        f0 = f(x0);

        if(g0 == 0.0)

        {

                cout<<"Mathematical Error.";

                exit(0);

        }

        x1 = x0 - f0/g0;

        cout<<"Iteration\t"<< step<<":\t x = "<< setw(10)<< x1<<" and f(x) = "<< setw(10)<< f(x1)<<
endl;

        x0 = x1;


        step = step+1;


        if(step > N)

        {

                cout<<"Not Convergent.";

                 exit(0);

        }

        f1 = f(x1);

}while(fabs(f1)>e);
```

```
    cout<< endl<<"Root is: "<< x1;

    return 0;

}
```

Output:

```
Enter tolerable error: .000001
Enter maximum iteration: 100

*********************
Newton Raphson Method
*********************
Iteration       1:       x =   1.500000 and f(x) =  12.250000
Iteration       2:       x =  -0.250000 and f(x) =   3.062500
Iteration       3:       x =  -1.125000 and f(x) =   0.765625
Iteration       4:       x =  -1.562500 and f(x) =   0.191406
Iteration       5:       x =  -1.781250 and f(x) =   0.047852
Iteration       6:       x =  -1.890625 and f(x) =   0.011963
Iteration       7:       x =  -1.945312 and f(x) =   0.002991
Iteration       8:       x =  -1.972656 and f(x) =   0.000748
Iteration       9:       x =  -1.986328 and f(x) =   0.000187
Iteration       10:      x =  -1.993164 and f(x) =   0.000047
Iteration       11:      x =  -1.996582 and f(x) =   0.000012
Iteration       12:      x =  -1.998291 and f(x) =   0.000003
Iteration       13:      x =  -1.999128 and f(x) =   0.000001

Root is: -1.999128
Process returned 0 (0x0)   execution time : 9.854 s
Press ENTER to continue.
```

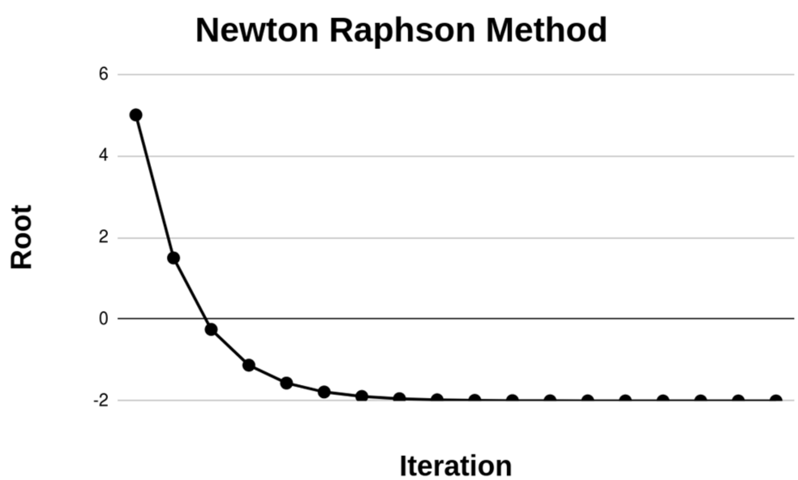
Newton Raphson Method Using Microsoft Excel:

x1	x2	f(x1)	f'(x1)
5	1.5	49	14
1.5	-0.25	12.25	7

-0.25	**-1.125**	**3.0625**	**3.5**
-1.125	**-1.5625**	**0.765625**	**1.75**
-1.5625	**-1.78125**	**0.19140625**	**0.875**
-1.78125	**-1.890625**	**0.0478515625**	**0.4375**
-1.890625	**-1.9453125**	**0.01196289063**	**0.21875**
-1.9453125	**-1.97265625**	**0.002990722656**	**0.109375**
-1.97265625	**-1.986328125**	**0.0007476806641**	**0.0546875**
-1.986328125	**-1.993164063**	**0.000186920166**	**0.02734375**
-1.993164063	**-1.996582031**	**0.0000467300415**	**0.013671875**
-1.996582031	**-1.998291016**	**0.00001168251038**	**0.0068359375**
-1.998291016	**-1.999145508**	**0.000002920627594**	**0.00341796875**
-1.999145508	**-1.999572754**	**0.0000007301568985**	**0.001708984375**
-1.999572754	**-1.999786377**	**0.0000001825392246**	**0.0008544921875**

-1.999786377	**-1.999893188**	**0.00000004563480616**	**0.0004272460938**
-1.999893188	**-1.999946594**	**0.00000001140870154**	**0.0002136230469**
-1.999946594	**-1.999973297**	**0.000000002852175385**	**0.0001068115234**

Graph:



Result: The approximate root is -1.999946594.

Discussion:

The root is not totally accurate . The amount of error is so little that it can be avoided. So, -1.999946594 can be considered as the root of the equation.

Experiment No.04

Experiment Name: Gauss Elimination.

Objective: To solve the system of linear equations using the Basic Gauss Elimination method.

Algorithm:

1. Arrange equations such that a11 ≠ 0
2. Eliminate x1 from all but the first equation. This is done as follows:
   i. Normalize the first equation by dividing it by a11.
   ii. Subtract from the second equation a21 times the normalized first equation.
   iii. Similarly, subtract from the third equation a31 times the normalized first equation.
3. Eliminate x2 from the third to the last equation in the new set. We assume that a’22≠ 0.
   i. Subtract from the third equation a’32 times the normalized first equation.
   ii. Subtract from the fourth equation a’42 times the normalized first equation and so on.
4. Obtain solution by back substitution.

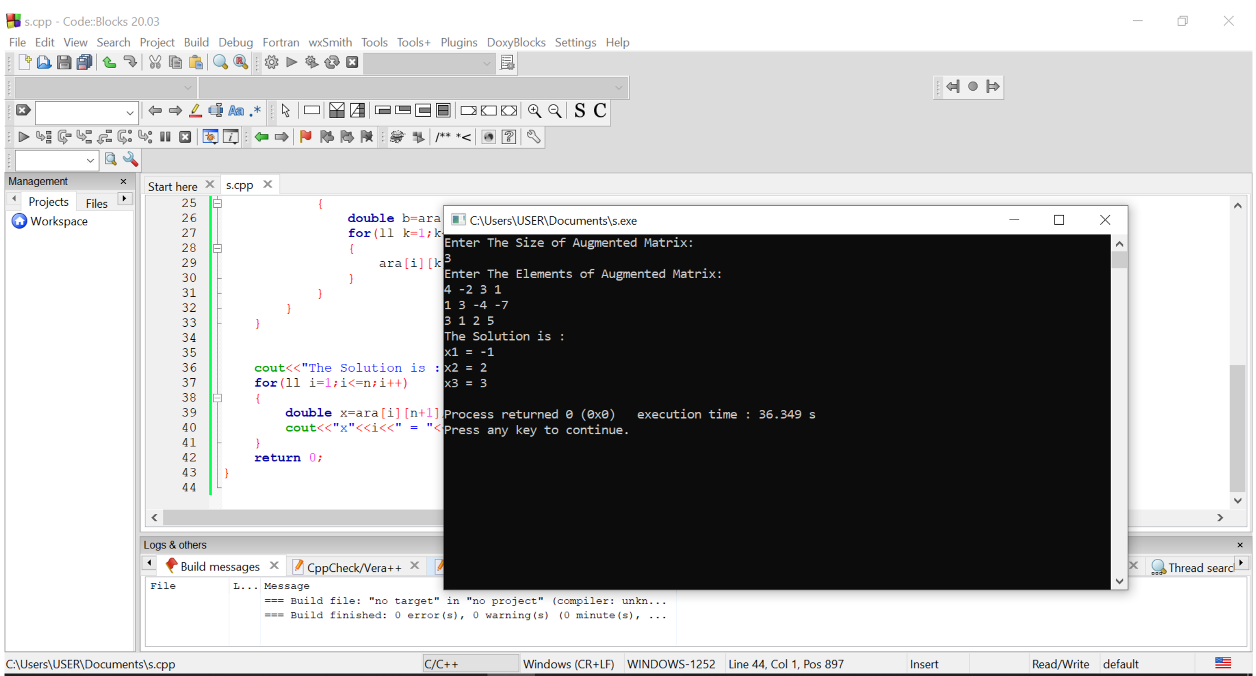
Source Code:

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
#define N '\n'
int main()
{
    cout<<"Enter The Size of Augmented Matrix: "<<endl;
```

```
ll n;
cin>>n;
cout<<"Enter The Elements of Augmented Matrix: "<<endl;
double ara[n+10][n+10];
for(ll i=1;i<=n;i++)
{
    for(ll j=1;j<=n+1;j++)
    {
        cin>>ara[i][j];
    }

}
for(ll j=1;j<=n;j++)
{
    for(ll i=1;i<=n;i++)
    {
        if(i!=j)
        {
            double b=ara[i][j]/ara[j][j];
            for(ll k=1;k<=n+1;k++)
            {
                ara[i][k]=ara[i][k]-b*ara[j][k];
            }
        }
    }
}
```

```
    cout<<"The Solution is : "<<endl;
    for(ll i=1;i<=n;i++)
    {
        double x=ara[i][n+1]/ara[i][i];
        cout<<"x"<<i<<" = "<<x<<N;
    }
    return 0;
}
```

Output:



Gauss Elimination Method Using Microsoft Excel:

Gauss-Jordan elimination for 3 by 3 matrices						
Starting Matrix						
	4	-2	3			1
A=	1	3	-4		B=	-7
	3	1	2			5
Step 1	(Normalize Pivot)					
	1	-0.5	0.75			0.25
	1	3	-4		B=	-7
	3	1	2			5
Step 2	(eliminate)					
	1	-0.5	0.75			0.25
	0	3.5	-4.75		B=	-7.25
	0	2.5	-0.25			4.25
Step 3	(Normalize pivot)					
	1	-0.5	0.75			0.25
	0	1	-1.35714		B=	-2.07143
	0	2.5	-0.25			4.25
Step 4	(Eliminate)					
	1	0	0.071429			-0.78571
	0	1	-1.35714		B=	-2.07143
	0	0	3.142857			9.428571

Step 5	(Normalize pivot)					
	1	0	0.071429			-0.78571
	0	1	-1.35714		B=	-2.07143
	0	0	1			3
Step 5	(Normalize pivot)					
	1	0	0			-1
	0	1	0		B=	2
	0	0	1			3
				Verification		
x1=	-1			0.227273	0.159091	-0.02273
x2=	2		A^(-1)=	-0.31818	-0.02273	0.431818
x3=	3			-0.18182	-0.22727	0.318182
	-1					
x=	2					
	3					

Result: The solutions are -1,2,3.

Discussion:

From the system of equations we find the solutions.Gauss elimination is the technique to find the solutions of x1,x2,x3.

Experiment No.05

Experiment Name: Lagrange Interpolation Method.

Objective: To find the value of y for x using Lagrange interpolation method.

Algorithm:

1. Read x, n

2. for i = 1 to (n+1) in steps of 1 do read xi, y end for

3. y=0

4. for i = 1 to (n+1) in steps of 1 do

5. p=1

6. for j = 1 to (n+1) in steps of 1 do

7. if (j≠i) then p=p ×(x - xj)/(xi - xj)

end for

8. y=y+yi ×p

end for

9. Write x,y

10. Stop

Source Code:

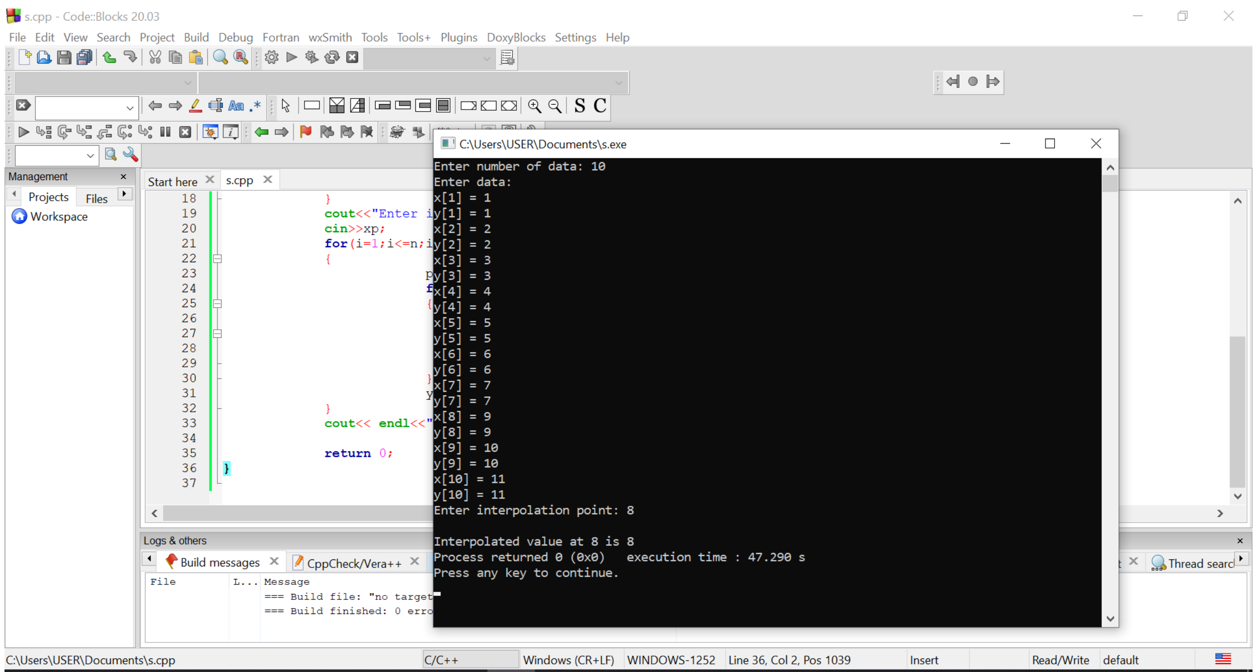
```
#include<bits/stdc++.h>

using namespace std;

 int main()

{

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

        int i,j,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;

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;
```

return 0;

}

Output:



Result: Interpolated value at 8 is 8.

Discussion:

From the system of equations we find the interpole point .Lagrange elimination is the technique to find the solution of the interpole point.

Experiment No.06

Experiment Name: Trapezoidal Rule.

Objective: To evaluate a definite integral by Trapezoidal Rule .

Algorithm:

1. Given a function f(x):

2. (Get user inputs) Input a,b=endpoints of interval n=number of intervals (Do the integration)

3. Set h= (b-a)/n.

4. Set sum=0.

5. Begin For i= 1 to n-1 Set x =a + h*i. Set sum=sum+2*f(x) End For

6. Set sum = sum + f(a)+f(b)

7. Set ans = sum*h/2.

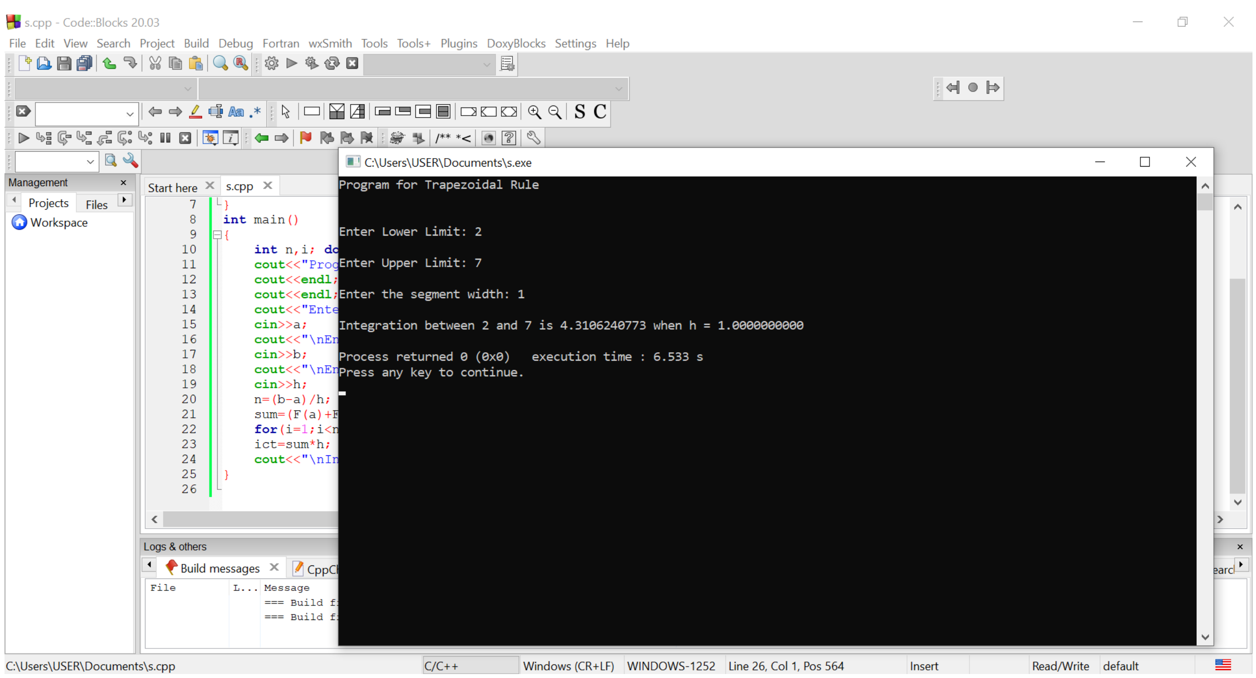
8. End

Source Code:

```
#include<bits/stdc++.h>

using namespace std;

double F(double x)

{

        double f=1.0-exp(-x/2.0);

        return f;

}

int main()

{

        int n,i; double a,b,sum,h,ict;

        cout<<"Program for Trapezoidal Rule"<<endl;

        cout<<endl;

        cout<<endl;

        cout<<"Enter Lower Limit: ";

        cin>>a;
```

```
    cout<<"\nEnter Upper Limit: ";

    cin>>b;

    cout<<"\nEnter the segment width: ";

    cin>>h;

    n=(b-a)/h;

    sum=(F(a)+F(b))/2.0;

    for(i=1;i<n;i++) sum+=F(a+i*h);

    ict=sum*h;

    cout<<"\nIntegration between "<<a<<" and "<<b<<" is "<<fixed<<setprecision(10)<<ict<<" when h = "<<h<<endl;

}
```

Output:



Trapezoidal Rule in MS excel:

Upper Limit b	Lower Limit	Segment h=1	n=(7-2)/1=5	

=7	a=2			
F=1-exp(x/2)	F(b)=0.9698026166	F(a)=0.6321205588		
I	a+i*h	F(a+i*h)	sum	Result
1	3	0.7768698399	1.577831428	1.577831428
2	4	0.8646647168	2.442496144	2.442496144
3	5	0.9179150014	3.360411146	3.360411146
4	6	0.9502129316	4.310624077	4.310624077

Result: Integration between 2 and 7 is 4.31062407.

Discussion: Trapezoidal Rule is the method to find the Area of a closed curve between uper bound and lower bound.

Experiment No.07

Experiment name:Simpson Rule.

Objective:To evaluate a definite integral by Simpson's 1/3 Rule.

Algorithm:

1. Given a function f(x):

2. (Get user inputs) Input a,b=endpoints of interval n=number of intervals(Even) (Do the integration)

3. Set h= (b-a)/n.

4. Set sum=0.

5. Begin For i= 1 to n -1 Set x =a + h*i. If i%2=0 Then Set sum=sum+2*f(x) Else Set sum=sum+4*f(x) End For

6. Set sum = sum + f(a)+f(b)

7. Set ans = sum*(h/3).

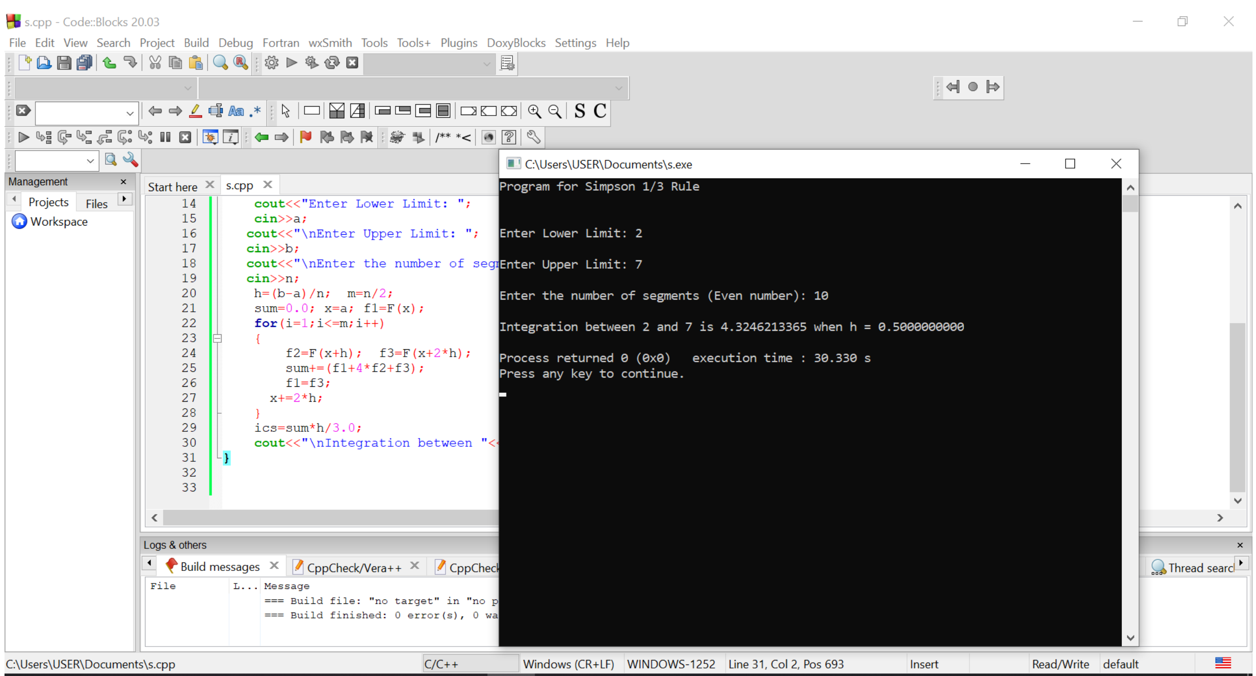
8. End

Source Code:

```
#include<bits/stdc++.h>

using namespace std;

double F(double x)

{

        double f=1.0-exp(-x/2.0);

        return f;

}

int main()

{

        int n,m,i; double a,b,sum,h,ics,x,f1,f2,f3;

        cout<<"Program for Simpson 1/3 Rule"<<endl;

        cout<<endl;

        cout<<endl;

        cout<<"Enter Lower Limit: ";

        cin>>a;

  cout<<"\nEnter Upper Limit: ";

  cin>>b;

  cout<<"\nEnter the number of segments (Even number): ";

  cin>>n;

        h=(b-a)/n;  m=n/2;

        sum=0.0; x=a; f1=F(x);
```

```
for(i=1;i<=m;i++)

{

f2=F(x+h);  f3=F(x+2*h);

sum+=(f1+4*f2+f3);

f1=f3;

  x+=2*h;

}

ics=sum*h/3.0;

cout<<"\nIntegration between "<<a<<" and "<<b<<" is "<<fixed<<setprecision(10)<<ics<<" when h = "<<h<<endl;

}
```

Output:



Simpson Rule by MS Excel:

Upper limit=7	Lower Limit=2	Segments=10	h=0.5	
F=1-exp(x/2)	F(b)=0.969802 6166	F(a)=0.632120 5588	m=10/2=5	
F1	X	F2	F3	sum
0.6321205588	2	0.8262260565	0.9502129316	9.58233349
0.9502129316	3	0.9360721388	0.9816843611	13.93189729
0.9816843611	4	0.9764822541	0.993262053	17.97494641
0.993262053	5	0.9913483048	0.9975212478	21.9907833
0.9975212478	6	0.9968172192	0.999088118	25.99660937
	7			
ict=sum*0.5/3= 4.332768228				

Result: Integration between 2 and 7 is 4.332768228

Discussion: Simpson Rule is the method to find the Area of a closed curve between upper bound and lower bound.