

# 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

<b>Experiment Number</b>	<b>Name of the Experiment</b>
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 $\frac{1}{3}$ 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  $x_1$  and  $x_2$  and stopping criterion  $E$ .
2. Compute  $f_1 = f(x_1)$  and  $f_2 = f(x_2)$ .
3. If  $f_1 * f_2 > 0$ ,  $x_1$  and  $x_2$  do not bracket any root and go to step 1.
4. Compute  $x_0 = (x_1 + x_2) / 2$  and compute  $f_0 = f(x_0)$ .
5. If  $f_0 = 0$  then  $x_0$  is the root of the equation, print the root
6. If  $f_1 * f_0 < 0$  then set  $x_2 = x_0$  else set  $x_1 = x_0$ .
7. If  $|(x_2 - x_1) / x_2| < E$  then root =  $(x_1 + x_2) / 2$ , 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:

The screenshot displays the Code::Blocks IDE interface. The main editor window shows a C++ file named `s.cpp` with the following code:

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 #define ll long long
4 #define N '\n'
5 #define f(x) x*x*x-x*x+2
6 int main()
7 {
8     double x0,x1,x2,f0,f1;
9     cout<<"Enter Two Guess
10     cin>>x1>>x2;
11     f1= f(x1);
12     f2= f(x2);
13     ll iteration=1;
14     if(f1*f2>0)
15     {
16         cout<<"x1 and x2 does
17         exit(0);
18     }
19     while (e>=0.00001&&f1!=
20     {
21         f1=f(x1);

```

Overlaid on the IDE is a terminal window titled `CAUsers\USER\Documents\s.exe`. It shows the program's execution output:

```

Enter Two Guess x1 and x2:
5
-6
The root is : -1
Process returned 0 (0x0)   execution time : 39.135 s
Press any key to continue.

```

At the bottom of the IDE, the 'Logs & others' panel shows build messages:

```

=== Build file: "no target" in "no project" (compiler: unkn...
=== Build finished: 0 error(s), 0 warning(s) (0 minute(s), ...

```

The status bar at the bottom indicates the file path `C:\Users\USER\Documents\s.cpp`, the language `C/C++`, and the current line and column: `Line 42, Col 1, Pos 849`.

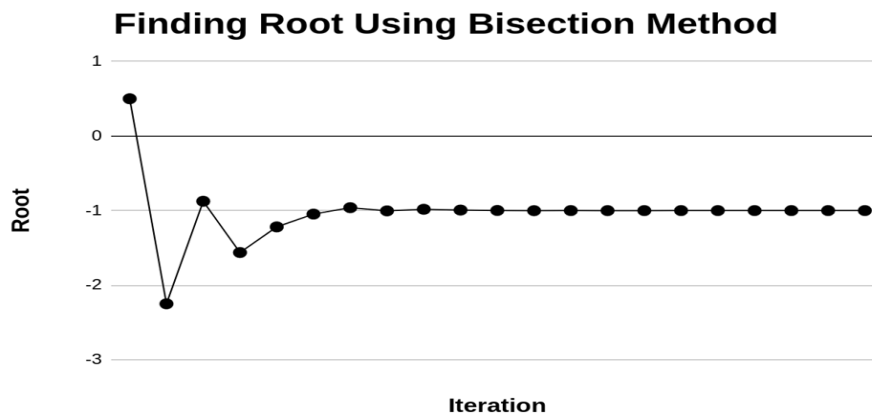
## Bisection Method in Microsoft Excel:

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  $x_1$  and  $x_2$ . 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  $x_1$  and  $x_2$  and stopping criterion  $E$ .
2. Compute  $f_1 = f(x_1)$  and  $f_2 = f(x_2)$ .
3. If  $f_1 * f_2 > 0$ ,  $x_1$  and  $x_2$  do not bracket any root and go to step 1.
4. Compute  $x_0 = x_1 - (f(x_1) (x_2 - x_1)) / (f(x_2) - f(x_1))$  and compute  $f_0 = f(x_0)$ .
5. If  $f_0 = 0$  then  $x_0$  is the root of the equation, print the root
6. If  $f_1 * f_0 < 0$  then set  $x_2 = x_0$  else set  $x_1 = x_0$ .
7. If  $|(x_2 - x_1) / x_2| < E$  then root =  $(x_1 + x_2) / 2$ , 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++;
    }

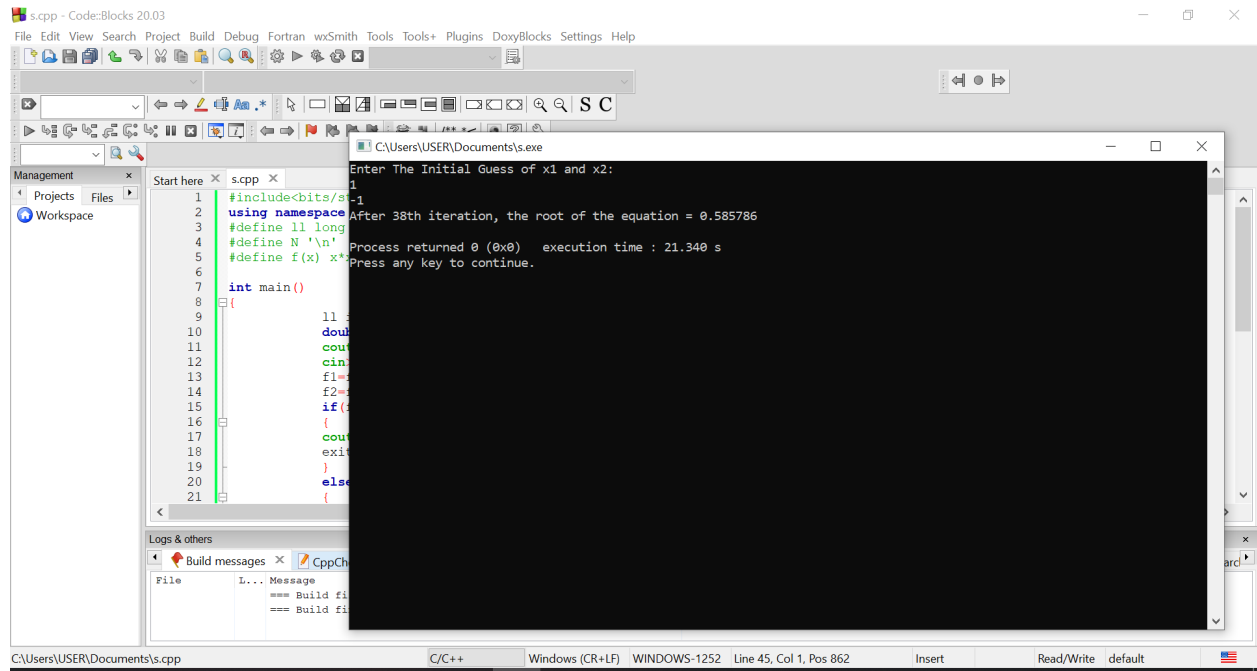
    cout<<"After "<<i<<"th iteration,"<<" the root of the equation = "<<x0<<N;

}

}

```

## Output:



The screenshot shows the Code::Blocks IDE with a C++ file named `s.cpp` open. The code implements a false position method to find the root of the equation  $F(x) = x^2 - 4x + 2$ . The program uses `long` for iterations and `double` for values. It prompts the user for initial guesses `x1` and `x2`. The output window shows the results of the iterations, including the root value `0.585786` after 38 iterations and the execution time `21.340 s`.

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 #define ll long long
4 #define N '\n'
5 #define f(x) x*x-4*x+2
6
7 int main()
8 {
9     ll x1, x2;
10    double x0, f1, f2, f0;
11    cin >> x1 >> x2;
12    f1 = f(x1);
13    f2 = f(x2);
14    if (f1 * f2 < 0)
15    {
16        cout << "Root exists between " << x1 << " and " << x2 << endl;
17        while (f(x0) > 0)
18        {
19            x0 = (x1 + x2) / 2;
20            f0 = f(x0);
21            if (f0 < 0)
22                x2 = x0;
23            else
24                x1 = x0;
25        }
26        cout << "Root of the equation is " << x0 << endl;
27    }
28    else
29    {
30        cout << "Root does not exist between " << x1 << " and " << x2 << endl;
31    }
32    return 0;
33 }
```

Output:

```
Enter The Initial Guess of x1 and x2:
After 38th iteration, the root of the equation = 0.585786
Process returned 0 (0x0)   execution time : 21.340 s
Press any key to continue.
```

## False Position Method Using Microsoft Excel:

Find the root of the following equation using false position Method

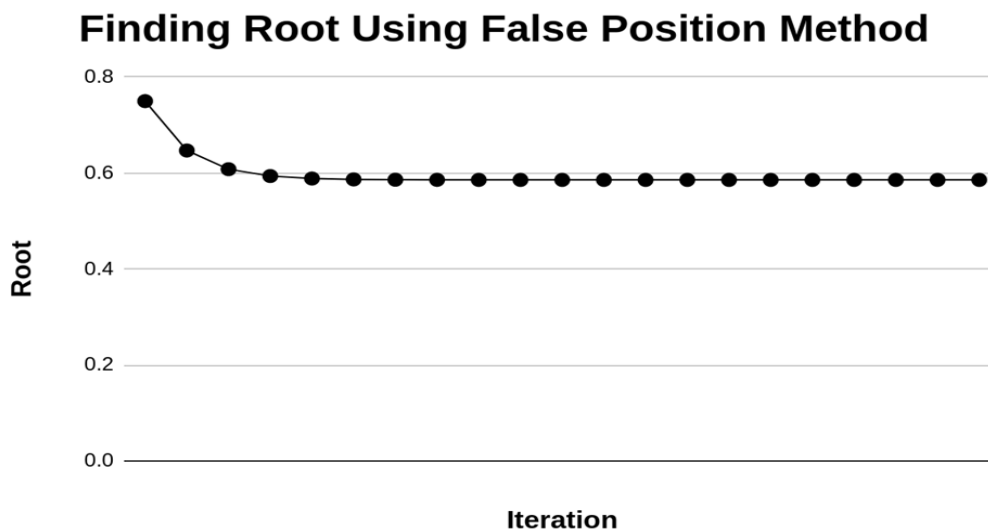
$$F(x)=x^2-4x+2$$

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

$$x_1 = x_0 - f(x_0) / f'(x_0)$$

4. Check for accuracy of the latest estimate.

If  $|(x_1 - x_0)/x_1| < E$  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:

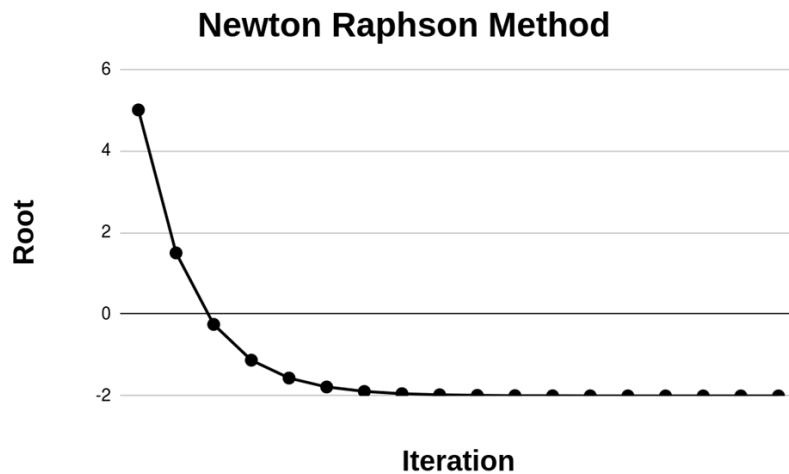
<b>x1</b>	<b>x2</b>	<b>f(x1)</b>	<b>f'(x1)</b>
<b>5</b>	<b>1.5</b>	<b>49</b>	<b>14</b>
<b>1.5</b>	<b>-0.25</b>	<b>12.25</b>	<b>7</b>

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



-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  $a_{11} \neq 0$
2. Eliminate  $x_1$  from all but the first equation. This is done as follows:
  - i. Normalize the first equation by dividing it by  $a_{11}$ .
  - ii. Subtract from the second equation  $a_{21}$  times the normalized first equation.
  - iii. Similarly, subtract from the third equation  $a_{31}$  times the normalized first equation.
3. Eliminate  $x_2$  from the third to the last equation in the new set. We assume that  $a'_{22} \neq 0$ .
  - 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:

The screenshot shows the Code::Blocks IDE with a C++ file named s.cpp. The code implements a Gauss elimination method for solving a system of linear equations. The console window displays the following output:

```

Enter The Size of Augmented Matrix:
3
Enter The Elements of Augmented Matrix:
4 -2 3 1
1 3 -4 -7
3 1 2 5
The Solution is :
x1 = -1
x2 = 2
x3 = 3
Process returned 0 (0x0)   execution time : 36.349 s
Press any key to continue.

```

The status bar at the bottom of the IDE shows the execution time as 36.349 s.

## 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  $x_1, x_2, x_3$ .

## 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, yi 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  $p = p \times (x - x_j) / (x_i - x_j)$
- end for
8.  $y = y + y_i \times p$
- 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:

The screenshot shows the Code::Blocks IDE with a C++ project named 's.cpp'. The code implements a Lagrange interpolation algorithm. It prompts the user to enter the number of data points (10) and then the data points themselves. The data points are stored in arrays x and y. The program then prompts for an interpolation point (8) and calculates the interpolated value, which is 8. The output window shows the execution details, including the interpolated value and the execution time (47.290 s).

```

s.cpp - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help

Management
  Projects
  Files
  Workspace

Start here x s.cpp x
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37

}
cout<<"Enter interpolation point: ";
cin>>xp;
for(i=1;i<=n;i++)
{
    x[i] = 1;
    y[i] = 1;
    x[i] = 2;
    y[i] = 2;
    x[i] = 3;
    y[i] = 3;
    x[i] = 4;
    y[i] = 4;
    x[i] = 5;
    y[i] = 5;
    x[i] = 6;
    y[i] = 6;
    x[i] = 7;
    y[i] = 7;
    x[i] = 8;
    y[i] = 9;
    x[i] = 9;
    y[i] = 10;
    x[i] = 10;
    y[i] = 11;
}
cout<< endl<< "Interpolated value at " << xp << " is " << result << endl;
return 0;

Logs and others
  Build messages x CppCheck/Ver++ x
  File L... Message
  === Build file: "no target"
  === Build finished: 0 errors

C:\Users\USER\Documents\s.exe
Enter number of data: 10
Enter data:
x[1] = 1
y[1] = 1
x[2] = 2
y[2] = 2
x[3] = 3
y[3] = 3
x[4] = 4
y[4] = 4
x[5] = 5
y[5] = 5
x[6] = 6
y[6] = 6
x[7] = 7
y[7] = 7
x[8] = 8
y[8] = 9
x[9] = 9
y[9] = 10
x[10] = 10
y[10] = 11
Enter interpolation point: 8
Interpolated value at 8 is 8
Process returned 0 (0x0)   execution time : 47.290 s
Press any key to continue.

C:\Users\USER\Documents\s.cpp C/C++ Windows (CR+LF) WINDOWS-1252 Line 36, Col 2, Pos 1039 Insert Read/Write default

```

**Result:**Interpolated value at 8 is 8.

## Discussion:

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

The screenshot shows the Code::Blocks IDE with a C++ project named 's.cpp'. The code implements the Trapezoidal Rule for numerical integration. The output window displays the following text:

```

Enter Lower Limit: 2
Enter Upper Limit: 7
Enter the segment width: 1
Integration between 2 and 7 is 4.3106240773 when h = 1.0000000000
Process returned 0 (0x0)   execution time : 6.533 s
Press any key to continue.

```

## 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 upper 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  $\text{ans} = \text{sum} * (h/3)$ .

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:

The screenshot shows the Code::Blocks IDE with a C++ project named 's.cpp'. The code implements Simpson's 1/3 Rule for numerical integration. The console window displays the program's execution, showing the input values for the lower limit (2), upper limit (7), and number of segments (10). The output shows the integration result between 2 and 7 is 4.3246213365 when h = 0.5000000000.

```

s.cpp - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help

Management
  Projects
  Files
  Workspace

Start here x s.cpp x
14 cout<<"Enter Lower Limit: ";
15 cin>>a;
16 cout<<"\nEnter Upper Limit: ";
17 cin>>b;
18 cout<<"\nEnter the number of segments (Even number): ";
19 cin>>n;
20 h=(b-a)/n; m=n/2;
21 sum=0.0; x=a; f1=F(x);
22 for(i=1;i<=m;i++)
23 {
24     f2=F(x+h); f3=F(x+2*h);
25     sum+=(f1+4*f2+f3);
26     f1=f3;
27     x+=2*h;
28 }
29 ics=sum*h/3.0;
30 cout<<"\nIntegration between "<<a<<" and "<<b<<" is "<<fixed<<setprecision(10)<<ics<<"
31 when h = "<<h<<endl;
32
33

C:\Users\USER\Documents\s.exe
Program for Simpson 1/3 Rule
Enter Lower Limit: 2
Enter Upper Limit: 7
Enter the number of segments (Even number): 10
Integration between 2 and 7 is 4.3246213365 when h = 0.5000000000
Process returned 0 (0x0) execution time : 30.330 s
Press any key to continue.

Logs and others
Build messages x CppCheck/Vera++ x CppCheck
File L... Message
=== Build file: "no target" in "no p
=== Build finished: 0 error(s), 0 wa

```

## 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.