RAJSHAHI UNIVERSITY OF ENGINEERING AND TECHNOLOGY

LAB REPORT - 01
TOPIC: NEWTON'S FORWARD AND BACKWARD DIFFERENCE INTERPOLATION FORMULA

COURSE NAME: SESSIONAL BASED ON CSE-2103
COURSE CODE: CSE-2104

SUBMITTED TO-
MD. FARUKUZZAMAN FARUK
LECTURER, DEPT. OF CSE,RUET

SUBMITTED BY-
MRITTIKA ROY
ROLL-1803175
SECTION - C
DEPT. - CSE

SUBMISSION DATE - 21 NOVEMBER, 2020

Title 01: Implementation of Newton's forward difference interpolation formula.

1.1 Objective:

- Implementing Newton's forward difference interpolation formula in c++.

1.2 Methodology:

- Take x and y values as input.
- Generate the forward difference table and display it.
- Calculate the p value
- Calculate the interpolating value using the formula:
  $y(x) = \Delta y_0 + p\Delta y_0 + (p(p-1)\Delta^2 y_0)/2! + \ldots.$

1.3 Implementation:

I have implemented Newton's forward difference interpolation formula according to the above methodology. I have used c++ as a programming language and Codeblocks 20.03 IDE.

Problem:

x	y
1	24
3	120
5	336
7	720

y(8) = ?

1.3.1 Source Code:

```
#include<bits/stdc++.h>
using namespace std;

int n=4;
double x[4];
double y[4][4];

string buffer;
vector<string>tmp;

void Input()
{
    ifstream f1;
    f1.open("newton's.txt");

    while(! f1.eof() )
```

```
        {
            f1>>buffer;
            tmp.push_back(buffer);
            buffer.clear();
        }


    for(int i=0,j=0;i<tmp.size();i+=2,j++)
    x[j] = stod(tmp.at(i));



    for(int i=1,j=0;i<tmp.size();i+=2,j++)
    y[j][0] = stod(tmp.at(i));

    }

    void Forward_difference_table()
    {
        for(int i=1;i<n;i++)
        {
            for(int j=0;j<n-i;j++)
            {
                y[j][i] = y[j+1][i-1] - y[j][i-1];
            }
        }
    }

    void Draw_difference_table()
    {
        for(int i=0;i<n;i++){

            cout<<x[i]<<"\t";

            for(int j=0;j<n-i;j++)
            cout<<y[i][j]<<"\t";

            cout<<endl;

        }
    }


    int factorial(int n)
    {
        if(n==1)
            return 1;
        else
            return n*factorial(n-1);

    }

    double p_val(int n, double p)
    {
```

```
    double p_original = p;

    for(int i=1; i<n;i++)
        p_original = p_original*(p-i);

    return p_original;

}

double Interpolation(double val)
{
    double result = y[0][0];
    double h = x[1]-x[0];
    double p = (val-x[0])/h;


    for(int i=1;i<n;i++)
    {
        result = result + ( p_val(i,p)*y[0][i] )/factorial(i);
    }

    return result;
}


int main()
{

   Input();


   Forward_difference_table();

   Draw_difference_table();

   double val;

   while(true)
   {
       cout<<"Enter an interpolating value: "<<endl;
       cin>>val;

       if(val==0)
           break;

       double missing_value = Interpolation(val);
       cout<<"Missing value for "<<val<<" is "<<missing_value<<endl;

   }


}
```

1.4 Output:

```
1       24      96      120     48
3       120     216     168
5       336     384
7       720
Enter an interpolating value:
8
Missing value for 8 is 990
Enter an interpolating value:
```

1.5 Conclusion and Observation:

The performance of the above code depends on the value of x for which the value of y will be calculated. If the value of x is near to the starting of the difference table, the value of y will be more appropriate than the value of x situated near to the ending of the difference table.

Title 02: Implementation of Newton's backward difference interpolation formula.

1.1 Objective:

- Implementing Newton's backward difference interpolation formula in c++.

1.2 Methodology:

- Take x and y values as input.
- Generate the forward difference table and display it.
- Calculate the p value
- Calculate the interpolating value using the formula:
  $y(x) = \Delta y_n + p\Delta y_n + (p(p+1)\Delta 2y_n)/2! + \ldots.$

1.3 Implementation:

I have implemented Newton's forward difference interpolation formula according to the above methodology. I have used c++ as a programming language and Codeblocks 20.03 IDE.

Problem:

x	y
15	0.2588190
20	0.3420201
25	0.4226183
30	0.5
35	0.5735764
40	0.6427876

y(38) = ?

1.3.1 Source Code:

```
#include<bits/stdc++.h>
using namespace std;
int n=6;
double x[7];
double y[7][7];
string buffer;
vector<string>tmp;

void Input()
{
    ifstream f1;
    f1.open("newton's_backward.txt");

    while(! f1.eof())
    {
```

```
            f1>>buffer;
            tmp.push_back(buffer);
            buffer.clear();
        }



    for(int i=0,j=0;i<tmp.size();i+=2,j++)
    {
        x[j] = stod(tmp.at(i));
    }


    for(int i=1,j=0;i<tmp.size();i+=2,j++)
    {
        y[j][0] = stod(tmp.at(i));
    }
}

void backward_difference_table()
{
    for(int i=1;i<n;i++)
    {
        for(int j=0;j<n-i;j++)
        {
            y[j][i] = y[j+1][i-1] -y[j][i-1];
        }
    }
}

void draw_difference_table()
{
    for(int i=0;i<n;i++)
    {
        cout<<x[i]<<"\t";
        for(int j=0;j<n-i;j++)
            cout<<i<<" "<<j<<" "<<setw(4)<<y[i][j]<<"\t";
        cout<<endl;
    }
}

int factorial(int n)
{
    if(n==1)
        return 1;
    else
        return n*factorial(n-1);
}

double p_val(int n, double p)
{
    double p_original = p;

    for(int i=1;i<n;i++)
    {
        p_original = p_original*(p+i);
```

```
    }
    return p_original;
}

double interpolation(double val)
{
    double result = y[n-1][0];
    double h=x[1]-x[0];
    double p=(val-x[n-1])/h;


    for(int i=1;i<n;i++)
    {
        result = result + (p_val(i,p)*y[n-i-1][i]/factorial(i));
    }
    return result;
}

int main()
{
    Input();
    backward_difference_table();

    draw_difference_table();

   double val;

    while(true)
    {
        cout<<"Enter an interpolating value: "<<endl;
        cin>>val;

        if(val==0)
            break;

        double missing_value = interpolation(val);
        cout<<"Missing value for "<<val<<" is "<<missing_value<<endl;
    }
    return 0;

}
```

1.4 Output:

```
15       0 0 0.258819    0 1 0.0832011    0 2 -0.0026029   0 3 -0.0006136   0 4
2.48e-05     0 5 4.1e-06
20       1 0 0.34202     1 1 0.0805982    1 2 -0.0032165   1 3 -0.0005888   1 4
2.89e-05
25       2 0 0.422618    2 1 0.0773817    2 2 -0.0038053   2 3 -0.0005599
30       3 0  0.5        3 1 0.0735764    3 2 -0.0043652
35       4 0 0.573576    4 1 0.0692112
40       5 0 0.642788
Enter an interpolating value:
```

```
38
Missing value for 38 is 0.615661
Enter an interpolating value:
```

1.5 Conclusion and Observation:

The performance of the above code depends on the value of x for which the value of y will be calculated. If the value of x is near to the ending of the difference table, the value of y will be more appropriate than the value of x situated near to the starting of the difference table.

RAJSHAHI UNIVERSITY OF ENGINEERING AND TECHNOLOGY

LAB REPORT - 02
TOPIC: NUMERICAL INTEGRATION

COURSE NAME: SESSIONAL BASED ON CSE-2103
COURSE CODE: CSE-2104

SUBMITTED TO-
MD. FARUKUZZAMAN FARUK
LECTURER, DEPT. OF CSE,RUET

SUBMITTED BY-
MRITTIKA ROY
ROLL-1803175
SECTION - C
DEPT. - CSE

SUBMISSION DATE - 21 NOVEMBER, 2020

Title 1: Implementation of numerical integration (Trapezoidal rule and Simpson's ⅓ rule)

1.1 Objectives:

- Implementing of Trapezoidal rule
- Implementing Simpson's ⅓ rule

1.2 Methodology:

- Take x and y values as input.
- Draw the table.
- Calculate the integrated value using the following formulas:
  Trapezoidal rule: Integration of y with respect to x within limit x_0 to x_n = $h/2[y_0 + 2(y_1 + y_2 + \ldots..+y_{n-1}) + y_n]$.
  Simpson's ⅓ rule: Integration of y with respect to x within limit x_0 to x_n = $h/3[\, y_0 + 4(y_1 + y_3 + y_5 + \ldots. +y_{n-1}) + 2(y_2 + y_4 + y_6 + \ldots.. + y_{n-2}) + y_n]$.

1.3 Implementation:

I have implemented Trapezoidal rule and Simpson's ⅓ rule according to the above methodology. I have used c++ as a programming language and Codeblocks 20.03 IDE.

1.3.1 Example 6.9:

A solid of revolution is formed by rotating about the x- axis the area between the x-axis, the lines x=0 and x=1, and a curve through the points with the following coordinates:

x	y
0.00	1.0000
0.25	0.9896
0.50	0.9589
0.75	0.9089
1.00	0.8415

Estimate the volume of the solid formed.

1.3.11 Source Code:

```
#include<bits/stdc++.h>
using namespace std;
int n;
double x[100],y[100];
double up, low, h;

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

```
        {
            cin>>x[i]>>y[i];
            y[i] = y[i]*y[i];
        }
}

void draw_table()
{
    cout<<"x"<<"\t"<<"y^2"<<endl;
    for(int i=0;i<n;i++)
    {
        cout<<x[i]<<"\t\t"<<y[i]<<endl;
    }
}

double trapezoidal()
{
    double h=x[1]-x[0];
    cout<<"Difference: "<<h<<endl;
    double result1 =y[0]+y[n-1];
    for(int i=1;i<n-1;i++)
        result1 += 2*(y[i]);
    double result2 = result1*(h/2);
    return result2;
}

double simpson_1by3()
{
    double h=x[1]-x[0];
    cout<<"Difference: "<<h<<endl;
    double result1 = y[0] + y[n-1];
    for(int i=1;i<n-1;i++)
    {
        if(i/2==0)
            result1 += 4*y[i];
        else
            result1 += 2*y[i];
    }
    double result2 = result1*(h/3);
    return result2;
}

int main()
{
    cout<<"Enter the number of terms:";
    cin>>n;
    cout<<"Enter the tabular values:"<<endl<<"X"<<"\t"<<"Y"<<endl;
    input();
    cout<<"Displaying the table"<<endl;
    draw_table();
    double result_T=0,result_S=0;
    result_T = 3.1416*trapezoidal();
    cout<<"The volume using Trapezoidal rule is:"<<result_T<<endl;
    result_S = 3.1416*simpson_1by3();
    cout<<"The volume using Simpson's 1/3 rule is:"<<result_S;
    return 0;
}
```

1.3.12 Output:

```
Enter the number of terms:5
Enter the tabular values:
X       Y
0.00    1.0000
0.25    0.9896
0.50    0.9589
0.75    0.9089
1.00    0.8415
Displaying the table
x       y^2
0               1
0.25            0.979308
0.5             0.919489
0.75            0.826099
1               0.708122
Difference: 0.25
The volume using Trapezoidal rule is:2.81092
Difference: 0.25
The volume using Simpson's 1/3 rule is:2.38671
```

1.3.2 Example 6.10:

Evaluate the integration of [1/(1+x)] within the limit of 0 to 1.

1.3.21 Source Code:

```
#include<bits/stdc++.h>
using namespace std;
int n;
double up, low, h;
double x[100]={0.0},y[100]={0.0};

void cal_x()
{
    for(int i=1;i<=n;i++)
    {
       x[i] = x[i-1] + h;
    }

}

void cal_y()
{
      for(int i=0;i<=n;i++)
      {
          y[i] = 1/(1+x[i]);
      }
}
```

```
void table()
{
    for(int i=0;i<=n;i++)
    {
        cout<<x[i]<<"\t"<<y[i]<<endl;
    }
}

double trapezoidal()
{
    double result1 =y[0]+y[n];
    for(int i=1;i<n;i++)
        result1 += 2*(y[i]);
    double result2 = result1*(h/2);
    return result2;
}

double simpson_1by3()
{
    double result1 = y[0] + y[n];
    for(int i=1;i<=n-1;i++)
    {
        if(i/2==0)
            result1 += 2*y[i];
        else
            result1 += 4*y[i];
    }
    double result2 = result1*(h/3);
    return result2;
}

int main()
{
    cout<<"Enter upper limit: ";
    cin>>up;
    cout<<"Enter lower limit: ";
    cin>>low;
    cout<<"Enter difference: ";
    cin>>h;
    n = (int)(((up-low)/h));
    cout<<"step intervals:"<<n+1<<endl;
    x[0] = low;
    cal_x();
    cal_y();
    cout<<"x"<<"\t"<<"y"<<endl;
    table();
    double result_T=0,result_S=0;
    result_T = trapezoidal();
    cout<<"The area using Trapezoidal rule is:"<<result_T<<endl;
    result_S = simpson_1by3();
    cout<<"The area using Simpson's 1/3 rule is:"<<result_S;

}
```

1.3.22 Output:

When h = 0.5

```
Enter upper limit: 1
Enter lower limit: 0
Enter difference: 0.5
step intervals:3
x       y
0       1
0.5     0.666667
1       0.5
The area using Trapezoidal rule is:0.708333
The area using Simpson's 1/3 rule is:0.472222
```

When h = 0.25

```
Enter upper limit: 1
Enter lower limit: 0
Enter difference: 0.25
step intervals:5
x       y
0       1
0.25    0.8
0.5     0.666667
0.75    0.571429
1       0.5
The area using Trapezoidal rule is:0.697024
The area using Simpson's 1/3 rule is:0.671032
```

When h = 0.125

```
Enter upper limit: 1
Enter lower limit: 0
Enter difference: 0.125
step intervals:9
x       y
0       1
0.125   0.888889
0.25    0.8
0.375   0.727273
0.5     0.666667
0.625   0.615385
0.75    0.571429
0.875   0.533333
1       0.5
The area using Trapezoidal rule is:0.694122
The area using Simpson's 1/3 rule is:0.788922
```

1.4 Conclusion and Observation:

From the above code we get to know that in numerical integration the Trapezoidal rule is more accurate than the Simpson's ⅓ rule but it takes more time. On the other hand in numerical integration Simpson's ⅓ rule gives an optimistic result compared to the other rules.