**Heaven’s Light is Our Guide**



**Rajshahi University of Engineering and Technology**

**Department of Computer Science and Engineering**

**Course No:** CSE.2104

**Course Title:** Sessional based on CSE.2104 ( Numerical Methods )

**Lab Report On:** Newton’s Forward & Backward Interpolation Formula and Numerical Integration.

**Submitted To**

Sadia Zaman Mishu, Assistant Professor, Dept . of CSE RUET

Md. Farukuzzaman Faruk, Lecturer, Dept. of CSE, RUET

**Submitted By**

Md. Ariful Islam  
Roll No: 1803046  
Section: A  
Department: CSE

Date : 20/11/2020

Chapter

1

**Title:** Implementation of Newton’s Forward & Backward Interpolation Formula.

## **Objective**

* Gathering knowledge about Newton’s Forward and Backward interpolation formula.
* Implementing the Formula’s in C++

## **Methodology**

* Initialize the value of n
* Load n tabulated pointes
* Calculate missing values of y(x) by Newton’s Interpolation
  + - * + Generate difference table
        + Find the value of p
        + Build a Factorial function
        + Find y(x) by the interpolating formula

## **Implementation**

I have implemented Newton’s Forward & Backward Interpolation formula according the above Pseudocode. I have taken the tabulated values from a text file. The tools I used here are :

* + - * + C++
        + Text File
        + Editor: CodeBlocks

## **Code**

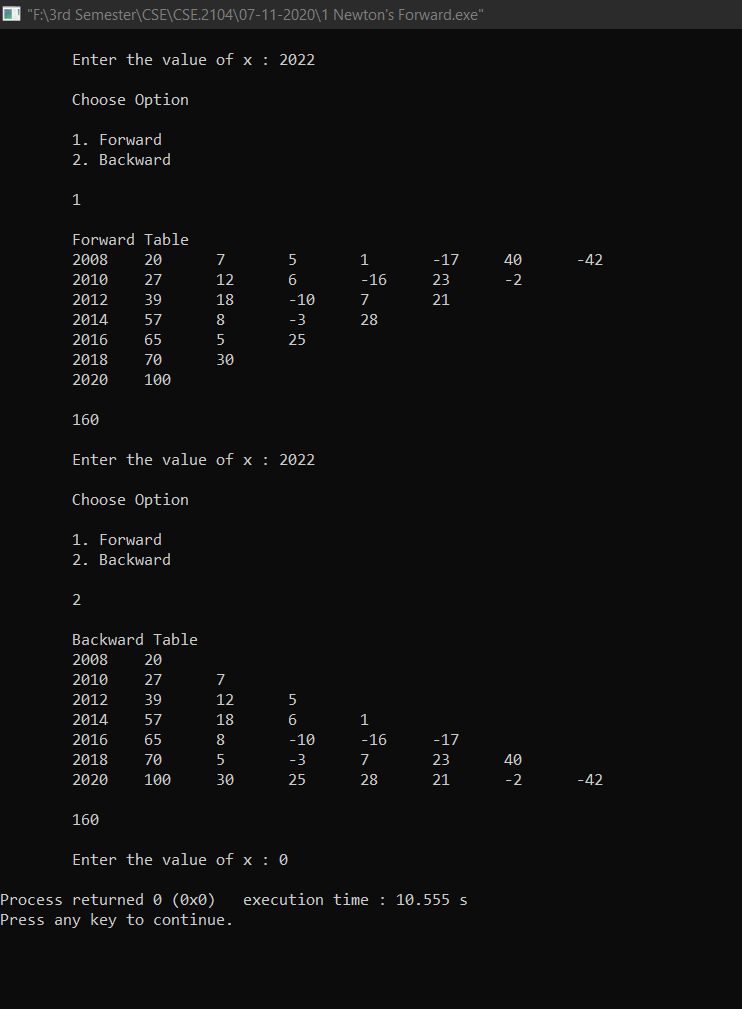
|  |
| --- |
| #include<bits/stdc++.h>  using namespace std;  int n;  double x[101];  double y[101][101];  string buffer;  vector<string>tmp;  void input(){  ifstream f1;  f1.open("Newton's.txt");  while(! f1.eof()){  f1>>buffer;  tmp.push\_back(buffer);  buffer.clear();  }  for(int i=0,j=0;i<tmp.size();i+=2,j++){  x[j]=stod(tmp.at(i));  y[j][0]=stod(tmp.at(i+1));  n=j+1;  }  }  void NFD(){  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 NBD(){  for(int i=1;i<n;i++){  for(int j=n-1;j>=i;j--){  y[j][i]=y[j][i-1]-y[j-1][i-1];  }  }  }  void FDT(){  cout<<"\n\tForward Table"<<endl;  for(int i=0;i<n;i++){  cout<<"\t"<<x[i]<<"\t";  for(int j=0;j<n-i;j++){  cout<<y[i][j]<<"\t";  }  cout<<endl;  }  cout<<endl;  }  void BDT(){  cout<<"\n\tBackward Table"<<endl;  for(int i=0;i<n;i++){  cout<<"\t"<<x[i]<<"\t";  for(int j=0;j<=i;j++){  cout<<y[i][j]<<"\t";  }  cout<<endl;  }  cout<<endl;  }  int fact(int n){  if(n==1)  return 1;  else  return n\*fact(n-1);  }  double Fp\_val(int n,double p){  double p\_o = p;  for(int i=1;i<n;i++){  p\_o\*=(p-i);  }  return p\_o;  }  double Bp\_val(int n,double p){  double p\_o = p;  for(int i=1;i<n;i++){  p\_o\*=(p+i);  }  return p\_o;  }  double FINT(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+=(Fp\_val(i,p)\*y[0][i])/fact(i);  }  return result;  }  double BINT(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+=(Bp\_val(i,p)\*y[n-1][i])/fact(i);  }  return result;  }  void menu(){  cout<<"\n\tChoose Option \n"<<endl;  cout<<"\t1. Forward\n\t2. Backward\n"<<endl;  }  int main(){  /// Step 1: Input  input();  double vx;  while(true){  cout<<"\n\tEnter the value of x : ";  cin>>vx;  if(!vx){  break;  }  menu();  int a;  cout<<"\t";  cin>>a;  switch(a){  case 1: NFD(); /// Newton's Forward Diff.  FDT(); /// Forward Diff. Table  cout<<"\t"<<FINT(vx)<<endl; /// Forward Interpolation  break;  case 2: NBD(); /// Newton's Backward Diff.  BDT(); /// Backward Diff. Table  cout<<"\t"<<BINT(vx)<<endl; /// Backward Interpolation  break;  default:  cout<<"Invalid Input\n"<<endl;  break;  }  }  return 0;  } |

## **Output**

I had used the following dataset in the implementation:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Year** | 2008 | 2010 | 2012 | 2014 | 2016 | 2018 | 2020 |
| **Sell** | 20 | 27 | 39 | 57 | 65 | 70 | 100 |

And my output was like below:



Chapter

2

**Title 1:** Implementation of Numerical Integration to Find Volume (Ex. 6.9 S.S. Sastry).

### **Objective**

* Gathering knowledge about Numerical Integration to Find Volume.
* Implementing the Knowledge in C++.

### **Methodology**

* Load coordinates from .txt file.
* Find n, the number of coordinates.
* Calculate volume by one of Numerical Integration Formula-
  + - * + Generate x ~ y**2** table from coordinates.
        + Find the value of h.
        + Calculate integrated value.
        + Calculate Volume by multiplying integrated value with Pi.

### **Implementation**

I have implemented one of Numerical Integration Formula (Simpson’s 1/3 Rule) to find the Volume according to the above Pseudocode. I have taken the coordinates from a text file. The tools I used here are :

* + - * + C++
        + Text File
        + Editor: CodeBlocks

### **Code**

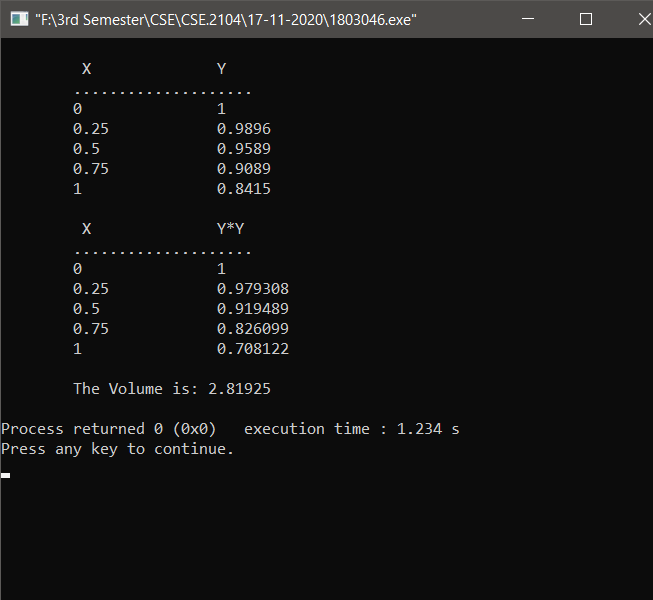
|  |
| --- |
| //This code is the Implementation of the Example 6.9 ( S.S. Sastry )  #include<bits/stdc++.h>  using namespace std;  int n;  double x[101],y[101];  string buf;  vector<string>temp;  #define pi 3.1416  void input(){  ifstream f1;  f1.open("6.9\_p.txt");  while(! f1.eof()){  f1>>buf;  temp.push\_back(buf);  buf.clear();  }  for(int i=0,j=0;i<temp.size();i+=2,j++){  x[j]=stod(temp.at(i));  y[j]=stod(temp.at(i+1));  n=j+1;  }  }  void show(){  for(int i=0;i<n;i++){  cout<<"\t"<<x[i]<<"\t\t"<<y[i]<<endl;  }  }  void double\_y(){  for(int i=0;i<n;i++){  y[i]\*=y[i];  }  }  double simpson(){  double h,res,val=0;  h=x[1]-x[0];  res=y[0]+y[n-1];  for(int i=1;i<n-1;i++){  if(i%2){  res+=(4.0\*y[i]);  }  else{  res+=(2.0\*y[i]);  }  }  val=(h\*res)/3.0;  return val;  }  void volume(){  input();  cout<<"\n\t X\t\tY\n\t...................."<<endl;  show();  double\_y();  cout<<"\n\t X\t\tY\*Y\n\t...................."<<endl;  show();  cout<<"\n\tThe Volume is: "<<pi\*simpson()<<endl;  }  int main(){  volume();  return 0;  } |

### **Output**

I had used the following dataset in the implementation:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **X** | 0.00 | 0.25 | 0.50 | 0.75 | 1.00 |
| **Y** | 1.0000 | 0.9896 | 0.9589 | 0.9089 | 0.8415 |

And my output was like below:



**Title 2:** Implementation of Numerical Integration (Ex. 6.10 S.S. Sastry).

### **2.2.1 Objective**

* Gathering knowledge about Numerical Integration.
* Implementing the Knowledge in C++.

### **2.2.2 Methodology**

* Select the value of h.
* Generate x **~**y table from x=0 to 1.
* Calculate Integrated value by Numerical Integration Formula-
  + - * + At first follow Trapezoidal Rule.
        + Then follow Simspon’s 1/3 Rule.

## **2.2.3 Implementation**

I have implemented both Trapezoidal Rule and Simpson’s 1/3 Rule to find the Integrated Value according to the above Pseudocode. I have generated the x **~**y table for each value of h. The tools I used here are :

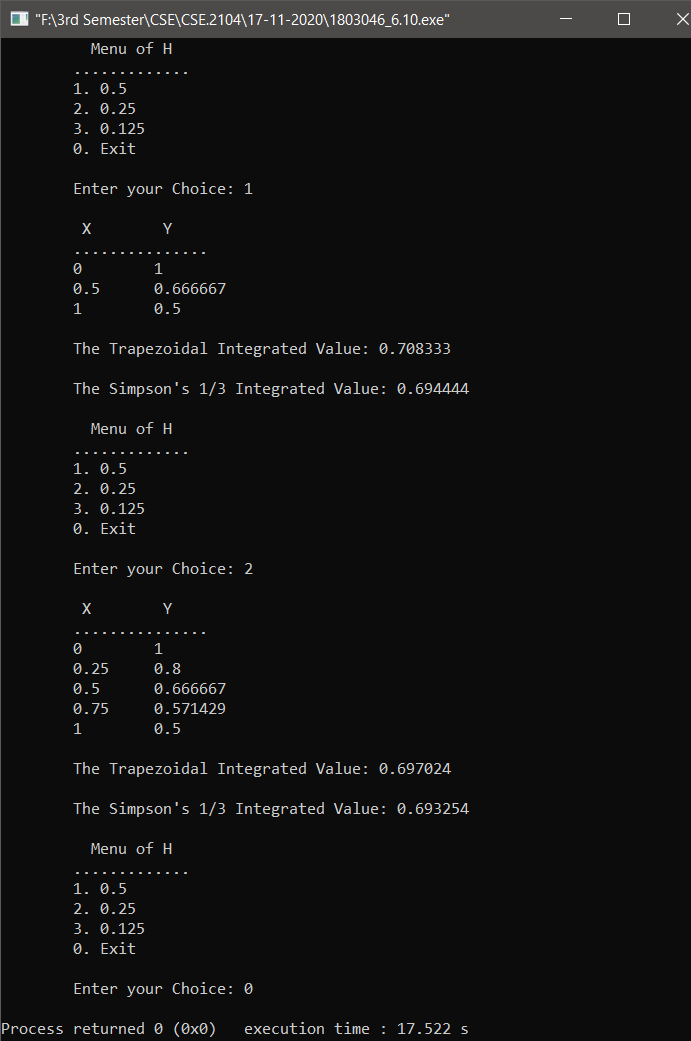
* + - * + C++
        + Editor: CodeBlocks

### **2.2.4 Code**

|  |
| --- |
| //This code is the Implementation of the Example 6.10 ( S.S. Sastry )  #include<bits/stdc++.h>  using namespace std;  double x[51],y[51];  int n;  void show\_xy(){  cout<<”\n\t X\t Y\n\t…............”<<endl;  for(int i=0;i<n;i++){  cout<<”\t”<<x[i]<<”\t “<<y[i]<<endl;  }  }  void Trapezoidal(double h){  double res=y[0]+y[n-1];  for(int i=1;i<n-1;i++){  res+=(2\*y[i]);  }  res=(h\*res)/2;  cout<<”\n\tThe Trapezoidal Integrated Value: “<<res<<endl;  }  void Simpsons(double h){  double res=y[0]+y[n-1];  for(int i=1;i<n-1;i++){  if(i%2){  res+=(4\*y[i]);  }  else{  res+=(2\*y[i]);  }  }  res=(h\*res)/3;  cout<<”\n\tThe Simpson’s 1/3 Integrated Value: “<<res<<endl;  }  void Init(double h){  n=(1.0/h)+1;  x[0]=0.0;  for(int i=0;i<n;i++){  x[i]=x[0]+(i\*h);  y[i]=1/(1+x[i]);  }  show\_xy();  Trapezoidal(h);  Simpsons(h);  }  void menu(){  cout<<”\n\t Menu of H\n\t…..........”<<endl;  cout<<”\t1. 0.5\n\t2. 0.25\n\t3. 0.125\n\t0. Exit\n”<<endl;  cout<<”\tEnter your Choice: “;  }  int main(){  int b=1;  while(b){  int a;  menu();  cin>>a;  switch(a){  case 1: Init(0.5); break;  case 2: Init(0.25); break;  case 3: Init(0.125); break;  case 0: b=0; break;  default: cout<<”Invalid Input\n”<<endl;break;  }  }  return 0;  } |

## **2.2.5 Output**

The equation of the problem was **Y = 1 / ( 1 + X )**

And my output was like below:

# End #