

Department of Computer Science

SPRING 20’

**NUMERICAL COMPUTATION PROJECT**

INSTRUCTOR

SIR JAMEEL USMANI

GROUP MEMBERS

18k-0353 (WAQAR ALI)

18K-1268 (HASAN SHAFIQ)

18k-1148 (MIRZA UZAIR ALI)

18k-0398 (SHARJEEL GHAURI)

**INTRODUCTION**

**Overview**

The main aim of the project was to create a program which can compute numerical computation problems by just enter some initial conditions for the question and getting the solution (in tabular form where necessary).

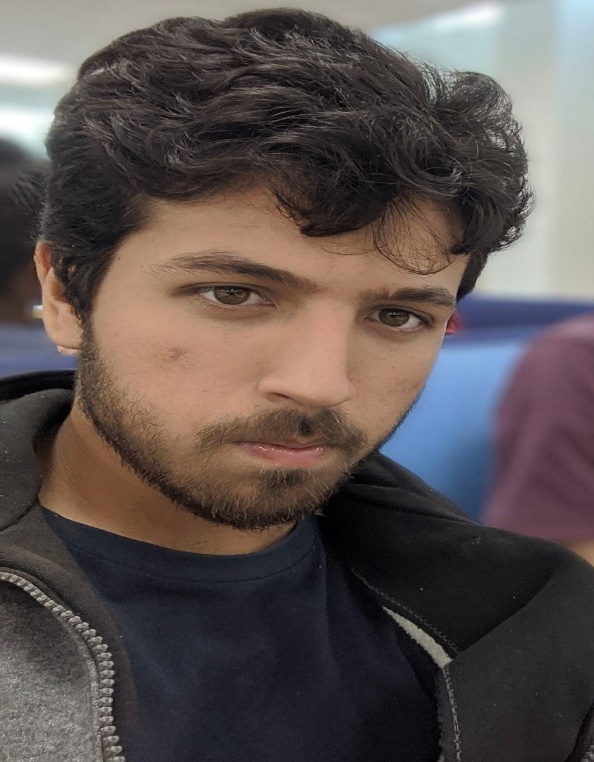
**Project Scope**

The main scope of the project is to solve difficult problems on this program and getting the solution in a few secs.

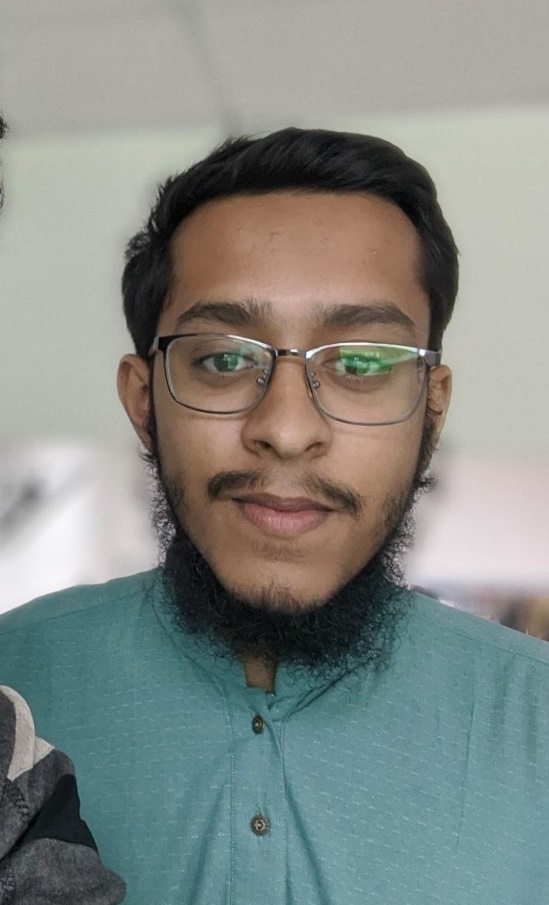
**ABOUT**

This project is purely developed on C++ language and can compute the solutions to the problem very easily. The program takes the initial conditions for the question such as value of upper bound, lower bound etc. and take the input of the function and then computes the answer and displays it on the output screen. This program is very efficient and gives the solution nearest to the approximation.

**GROUP MEMBERS DETAILS**

18k-1148 (MIRZA UZAIR ALI) 18k-0353 (WAQAR ALI)

18K-1268 (HASAN SHAFIQ) 18k-0398 (SHARJEEL GHAURI)

WORK DISTRIBUTION

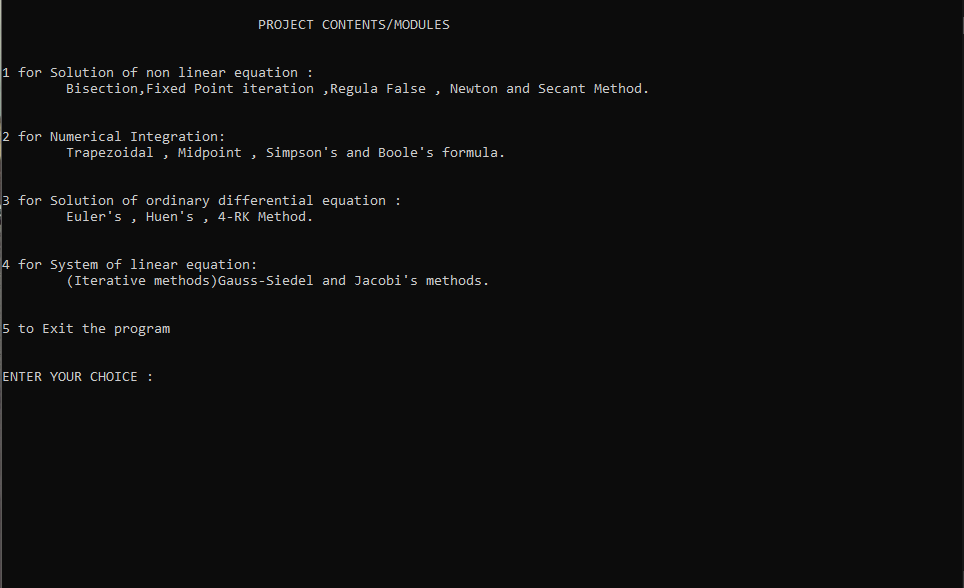
WORK WAS EQUALLY DIVIDED AMONGST THE GROUP MEMBERS 1 MODULE FOR EACH MEMBER.

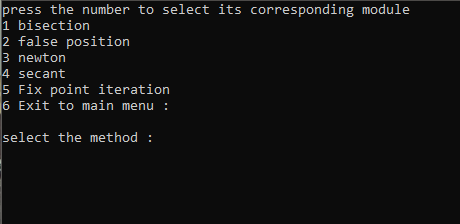
**ALGORITHMS**

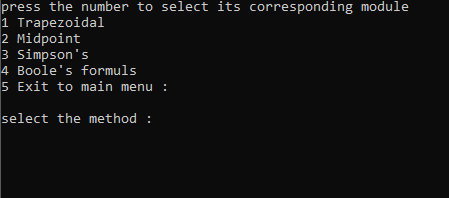
ALGORITHMS OF ALL THE DIFFERENT MODULES AND PROGRAMS WERE TAKEN FROM NUMERICAL COMPUTING BOOK BY Richard L. Burden and J. Douglas Faires & NUMERICAL METHODS BY John H.Mathews, Kurtis D.FINK WERE USED AND CODING OF THE PROGRAMS WAS DONE IN C++ LANGUAGE IN CODE BLOCKS IDE.

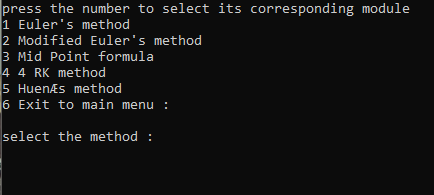
**Screenshots**

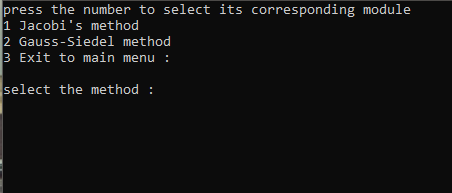
PROJECT OVERVIEW (THE MENU):



****

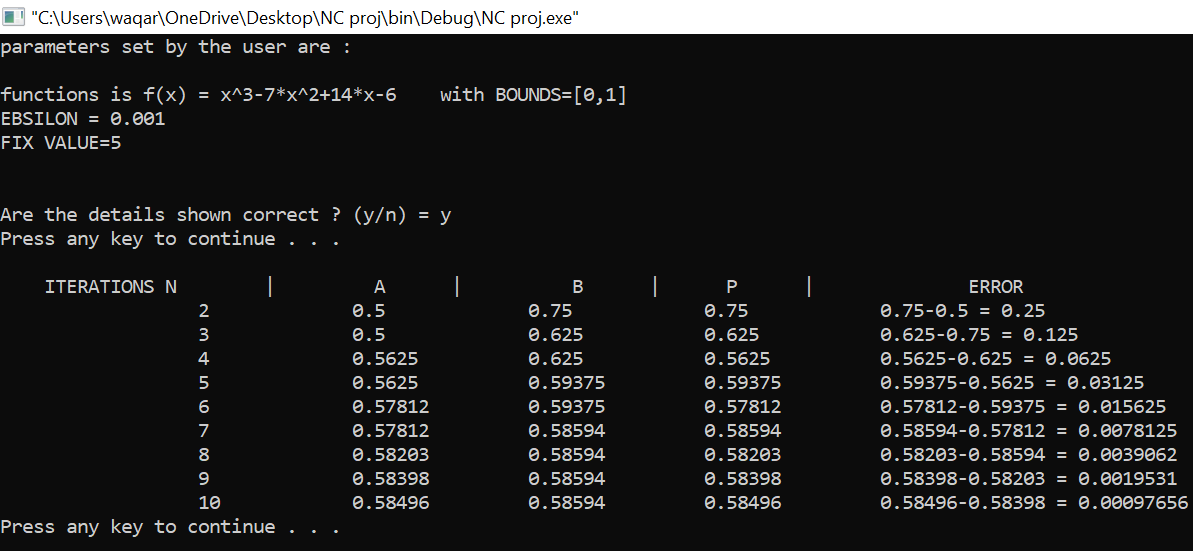
****

****

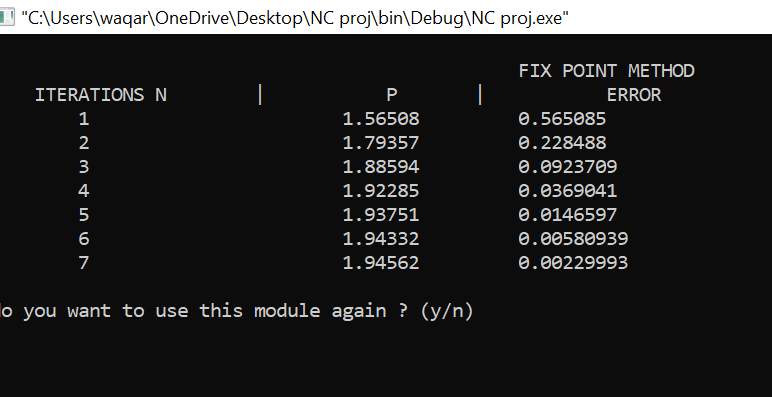
****

SOLUTION OF NON-LINEAR EQUATIONS:

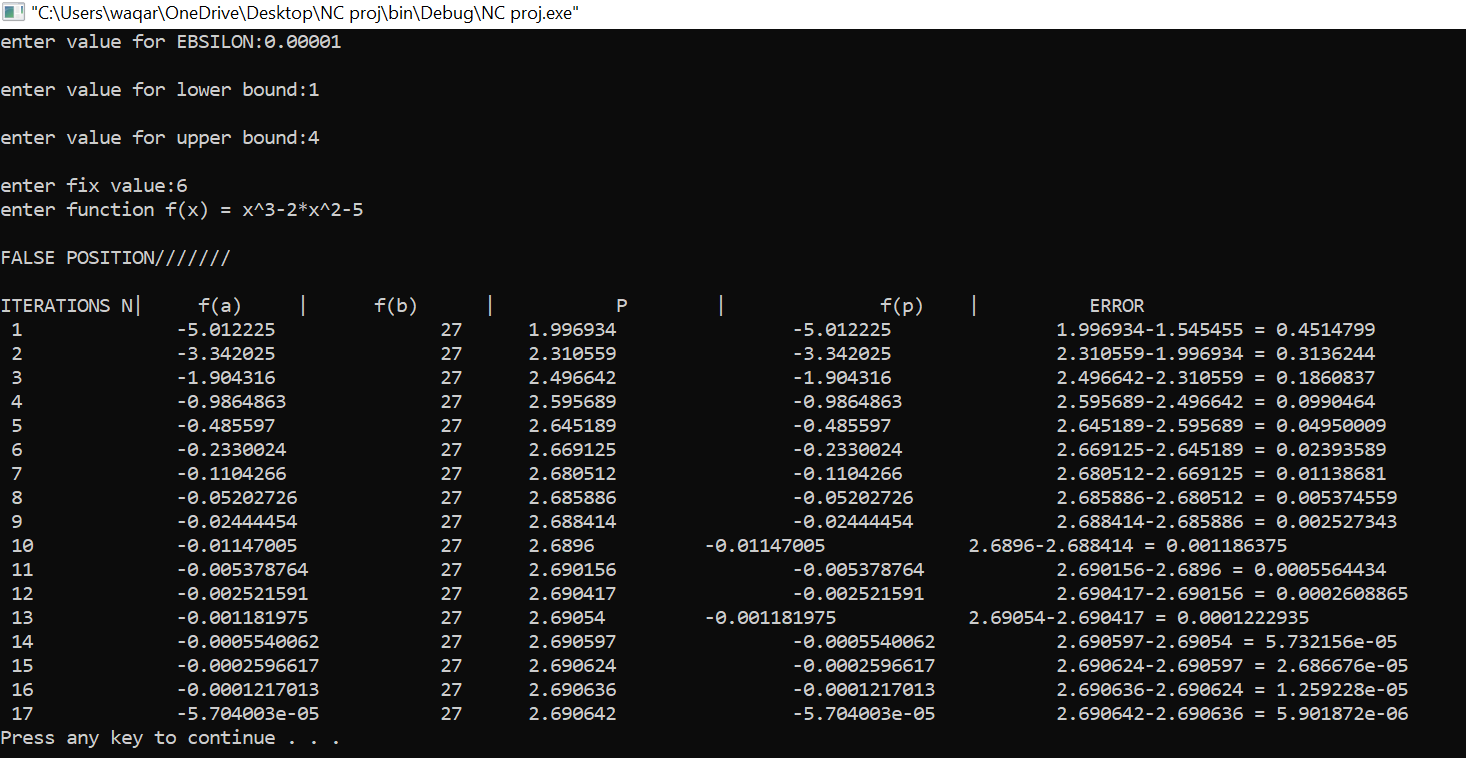
Bisection Method:



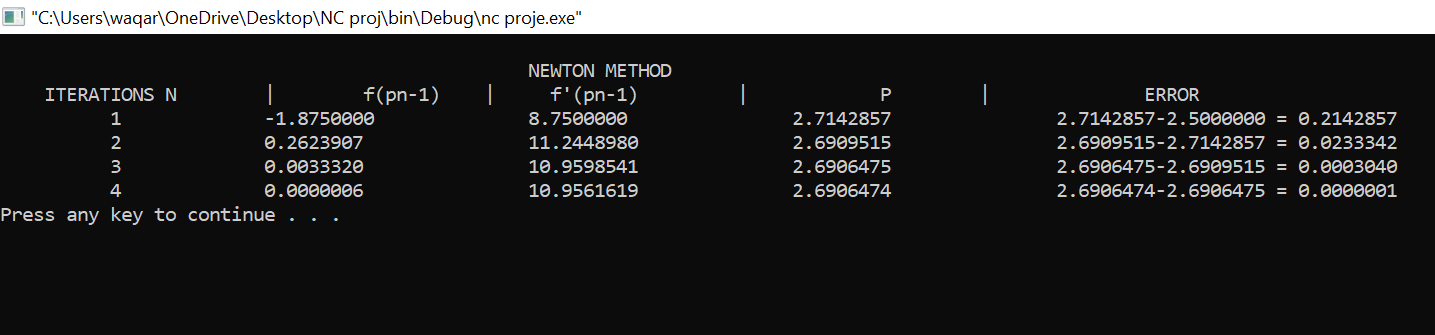
Fixed Point Iteration:



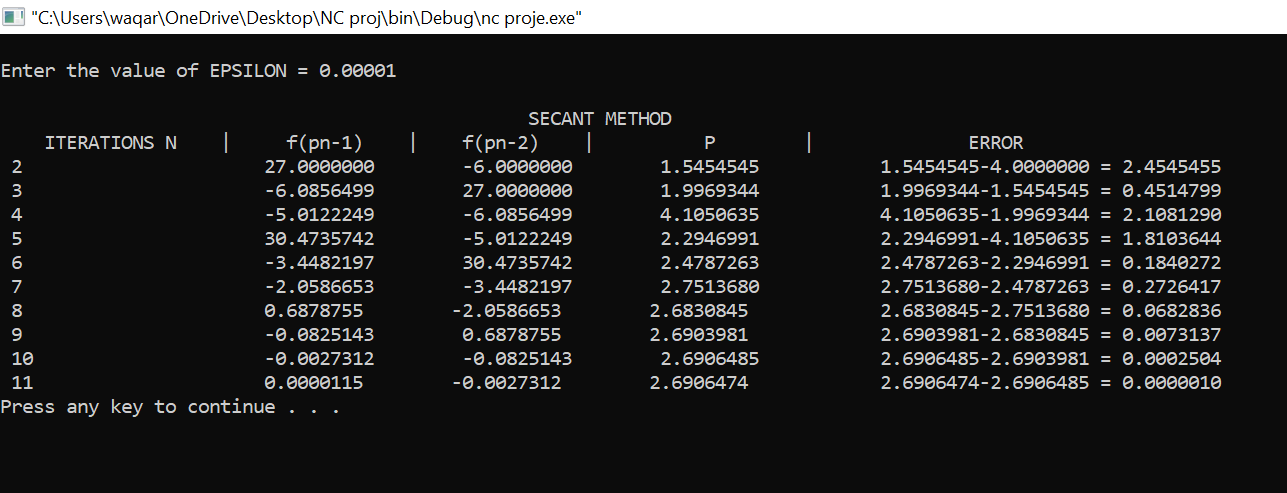
False Position:



Newton Method:

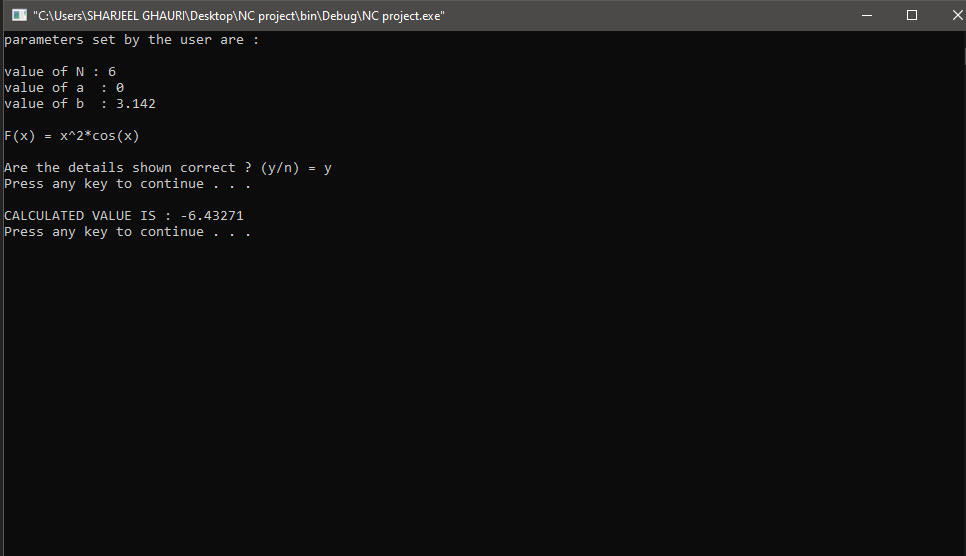


Secant Method:

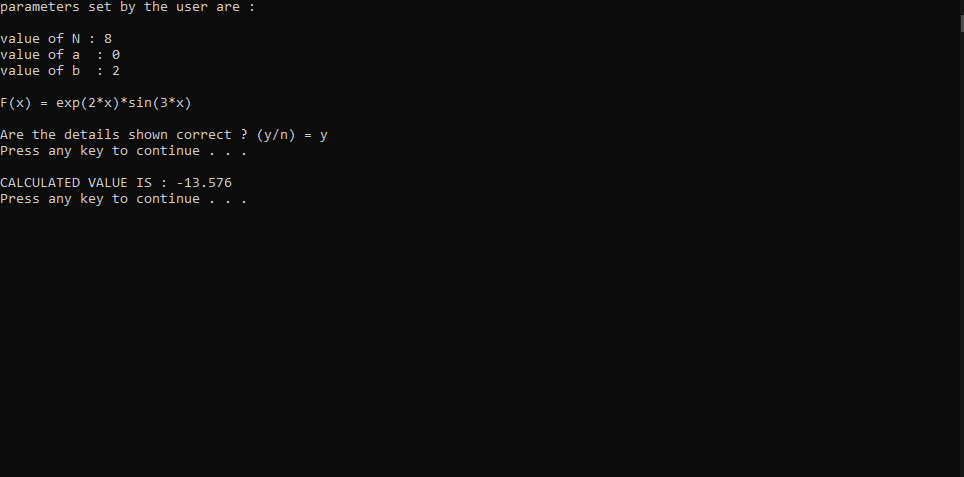


NUMERICAL INTEGRATION:

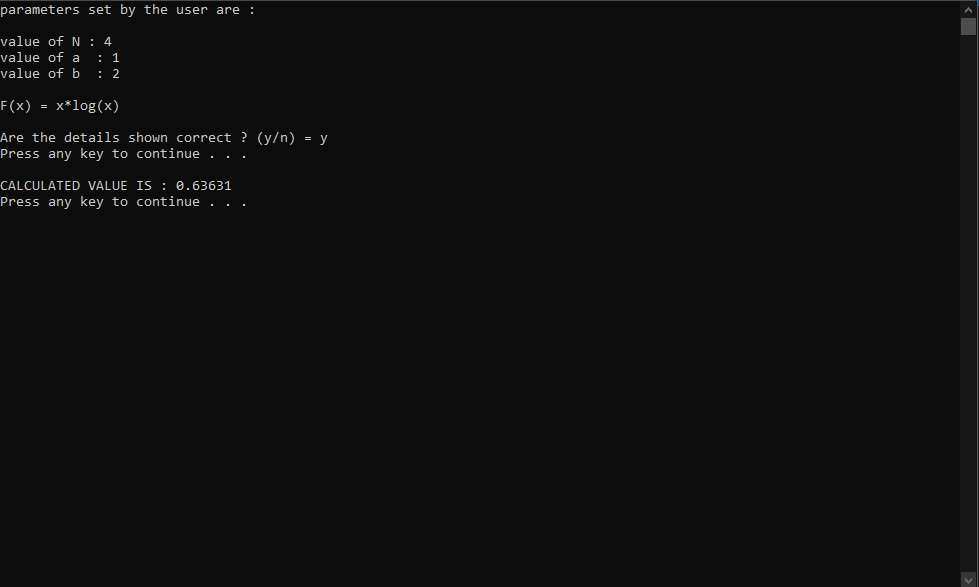
Trapezoid Method:



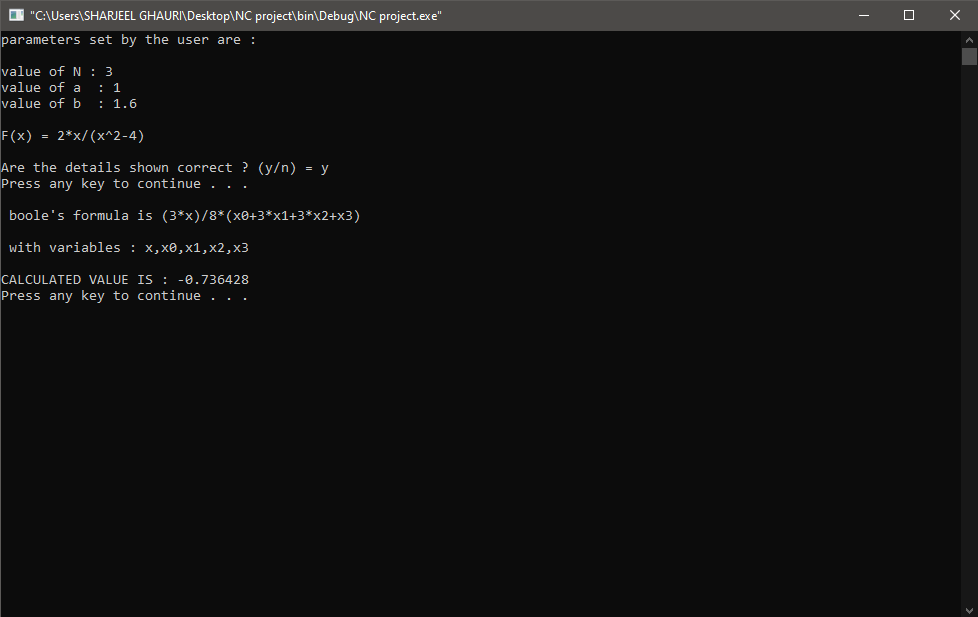
MIDPOINT METHOD:



SIMPSON’S METHOD:

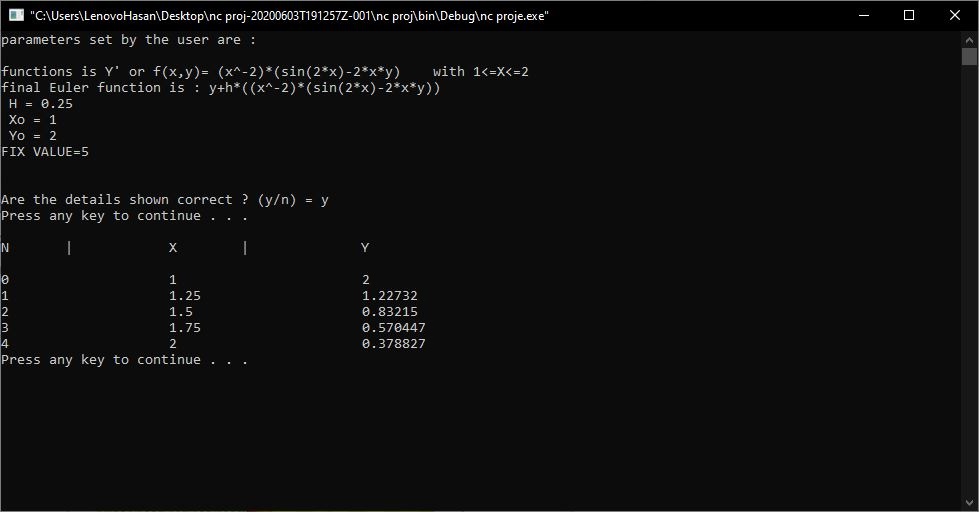


BOOLE’S METHOD:

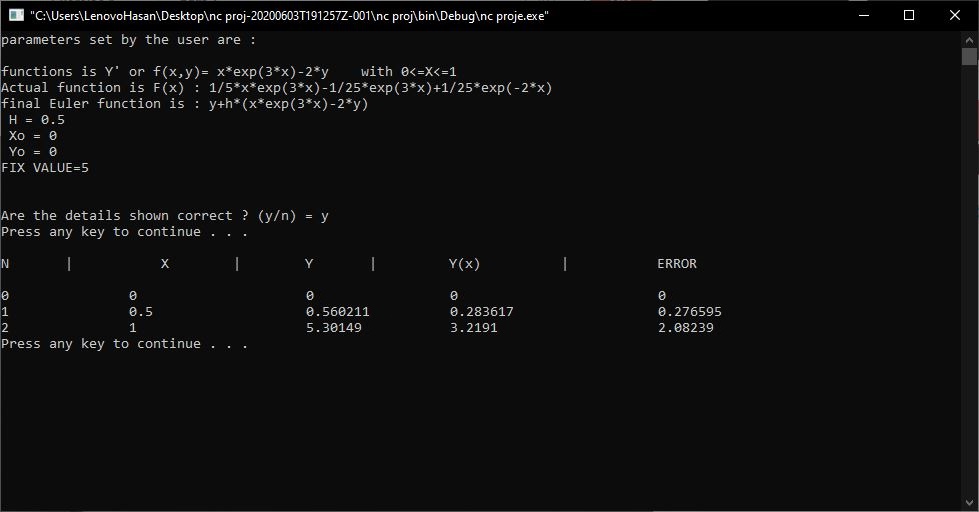


SOLUTION OF ORIDINARY DIFFERENTIAL EQUATIONS:

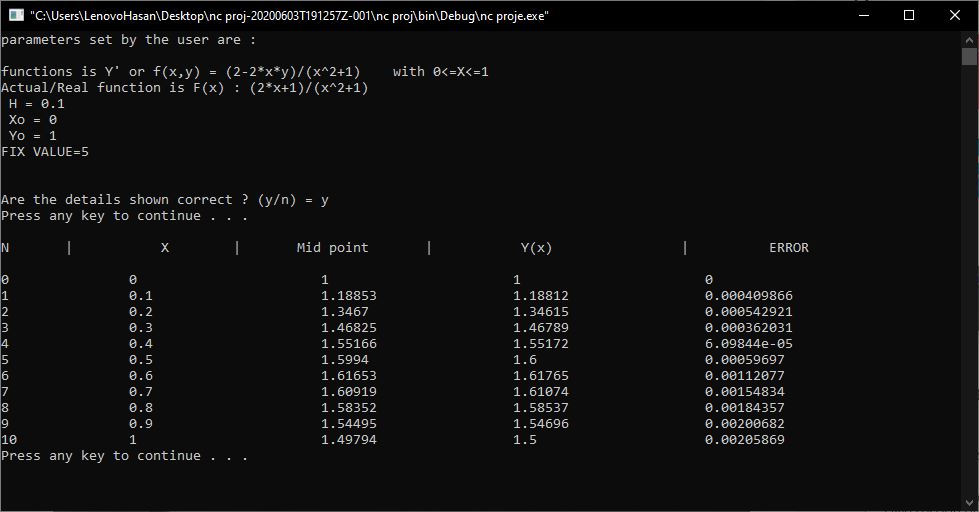
Euler Method:



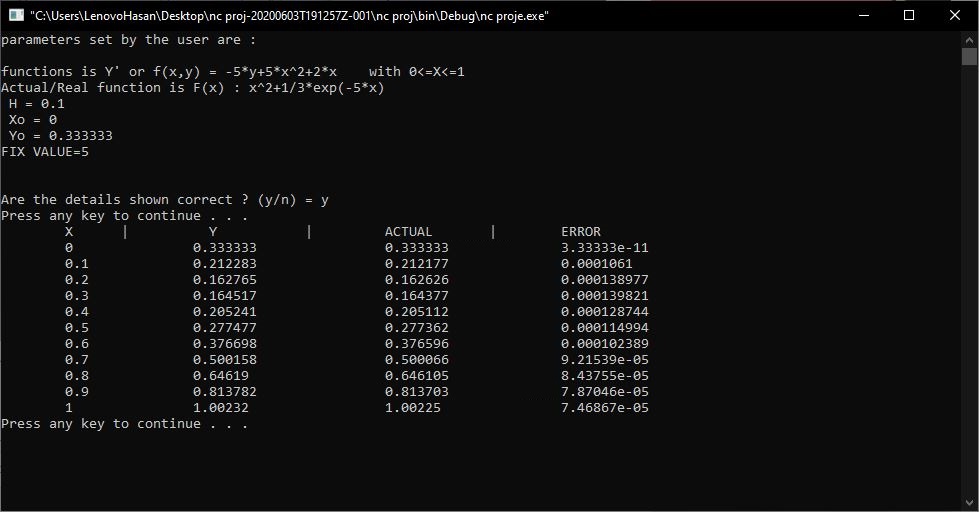
Modified Euler Method:



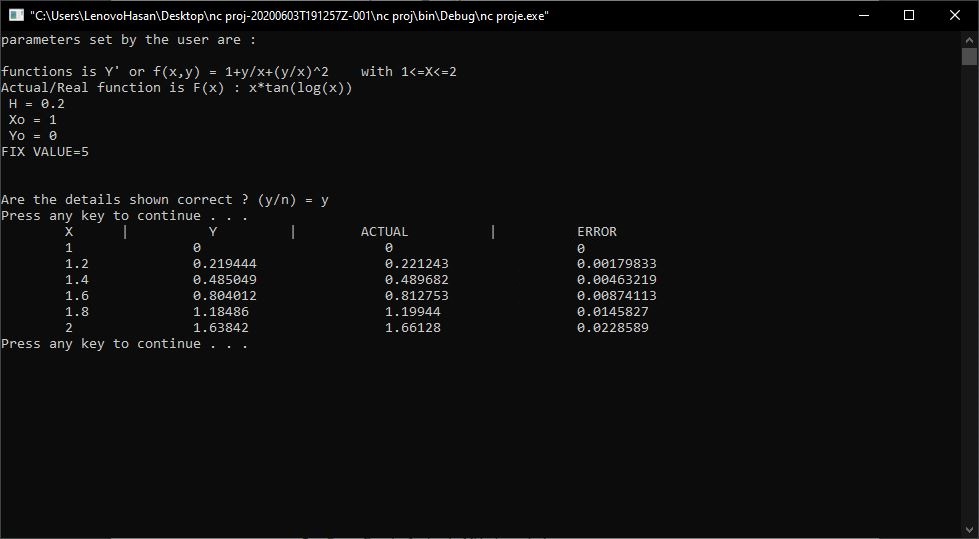
Midpoint Formula:



Runge Kutta 4th order:

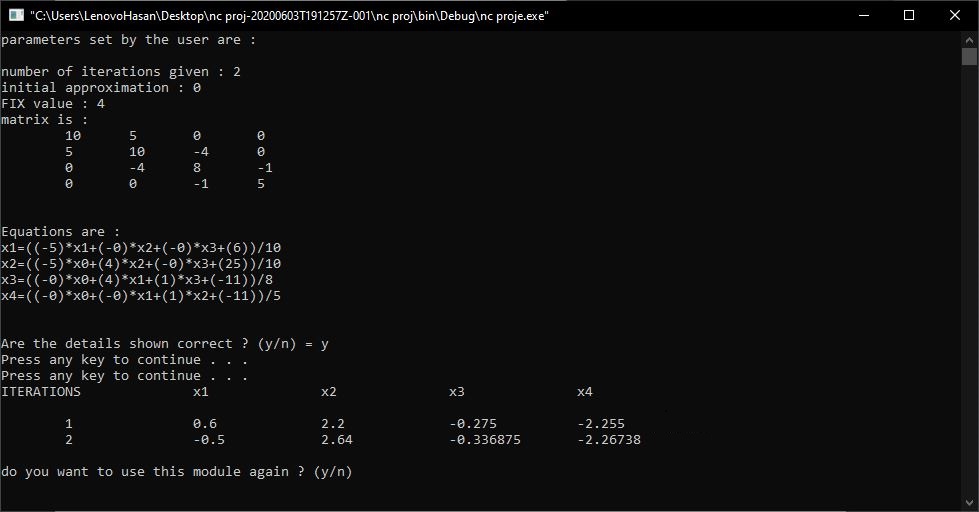


Huen Method:

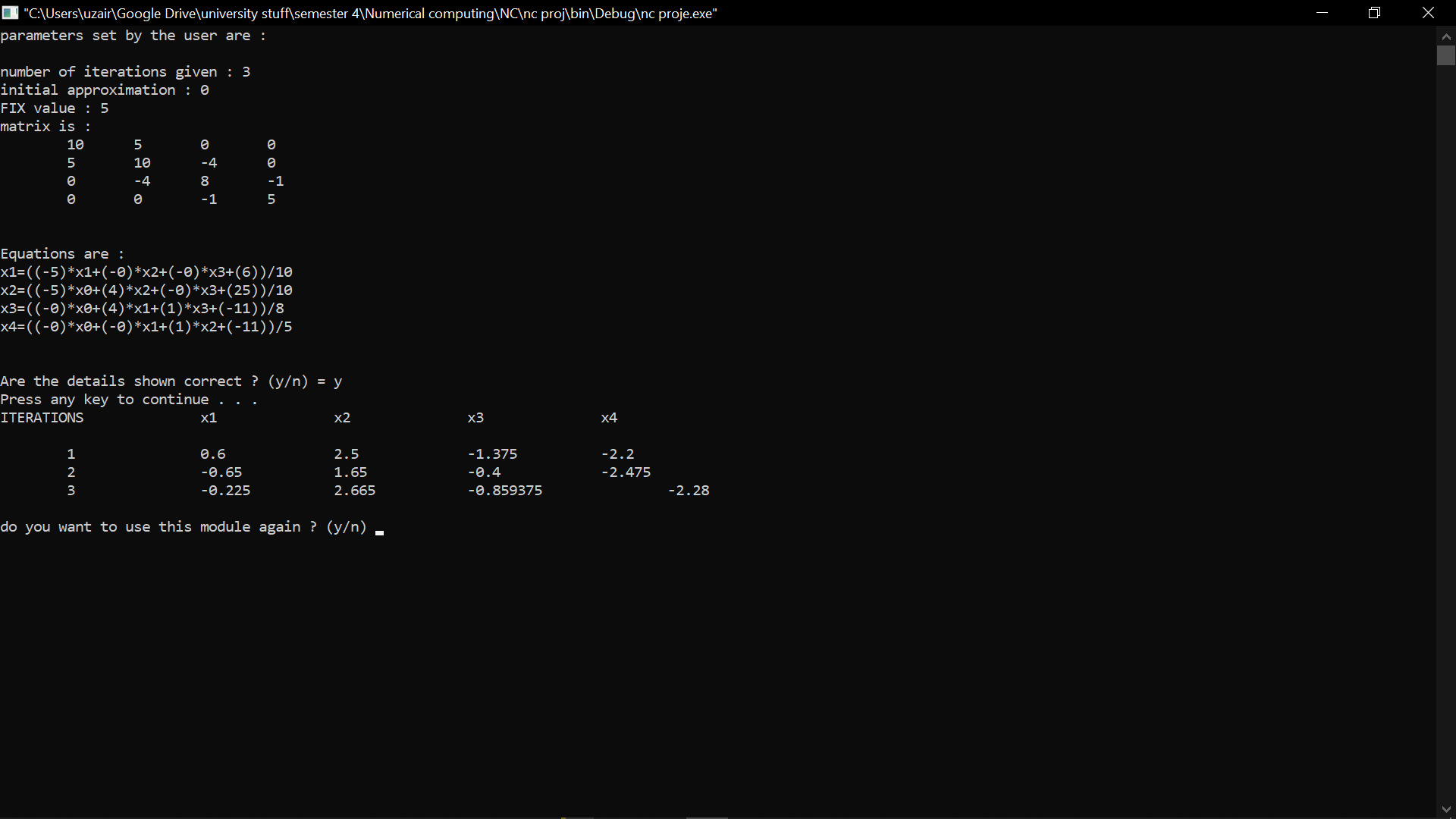


SYSTEMS OF LINEAR EQUATION:

Gauss–Seidel Method:



Jacobi Method:



**COMPARISSION**

**&**

**ANALYSIS OF MEHTOD**

**SOLUTION OF NON-LINEAR EQUATIONS:**

There are a few basic Iterative method for solution of nonlinear equations, differing in a way of approximation and in the number of converging iterations:

* Bisection Method
* Fixed Point Method
* Regular False Method
* Newton Method
* Secant Method
* Iterative method is one in which we start from an approximation to the true solution and if successful, obtain better approximation from a computational cycle repeated as often as necessary for achieving a required accuracy, so that the amount of arithmetic depends upon the accuracy required.
* It also shows the rate of convergence, with regard to convergence, we can summarize that a numerical method with a higher rate of convergence may reach the solution of the equation with less iterations in comparison to another method with a slower convergence.
* The existence of a solution of a nonlinear equation *f* (*x*) = 0with one variable is determined by the intermediate value theorem. From the methods explained in this work, it can be discovered that the newton method is the best iterative method for solving the nonlinear equations.
* **Newtons Method:**

Newton method is the best iterative method for solving the nonlinear equations with one variable as it converges more rapidly and accurately to the root of the equation when the initial guess is close to the roots of the equation.

* **Secant Method:**

It is also clear to say that for secant, at any given time before it converges to the root of the equation where 1, *k k x* + = *x* then it means that the method will fail from that point.

* **Regular False Method:**

The regular False method converges more slowly compare to the Newton and secant method respectively and the bisection method requires a large number of iterations before it converges to the root of the equation irrespective of how one starts close to the roots of the equation.

* **Bisection Method:**

Bisection method took a lot of iteration to converge so it is the worst method for solving and finding the solution to nonlinear equations.

* **Fixed Point Method:**

Fixed point method took less iteration to converge but it if we talk about iterations so secant method was better than fixed point method.

CONCLUSSION:

From the methods tested, the Newton method appeared to be the most robust and capable method of solving the nonlinear equation *f* (*x*) = 0.Results obtained from the four methods above show that the Newton Method is the most efficient method in finding the roots of non-linear equations seeing that it converges to the roots of the non-linear equation faster than the other methods. That is, it converges after a few iterations unlike the other methods which converges after many iterations.

**BEST ACCURATE & EFFICIENT METHOD: NEWTONS METHOD**

**NUMERICAL INTEGRATION**

There are a few basic methods of numerical integration, differing in a way of approximation:

* Midpoint Rule
* Trapezoidal Rule
* Simpson’s Rule
* Boole’s Rule

In all of them we get approximated value of integral, but they are determined with various errors and speed of convergence to the correct result. All of these methods consist on dividing the interval [a, b] on n same subintervals and calculating area of function for each of the subintervals with using specific formulas.

* **Simple functions**

Simple functions, here f(x) =x and f(x) = x3−x, calculated numerically have a precise solution very fast. Sometimes even we don’t need to divide the interval into many subintervals to find very approximated result and often additionally the chosen rule covers precisely the graph of the function. For f(x) =x all of these three methods find the precise solution and for f(x) =x3−xSimpson’s rule finds it very fast, because in specific division graphs in subintervals are covered with Simpson’s parabolas.

* **Imperfection of midpoint rule**

Midpoint rule is one of the least accurate methods and can lead to very wrong result, for example it can calculate that the integral equals0. It is visible inf(x) =x3−x, where for not divided interval, son= 1, the middle of the interval is x= 1, for which value of function equals0. From the formula for the area of the rectangle we can see, that whole integral equals0.However it changes when we start to divide the intervals and eliminate the x= 1as the midpoint.

* **Trapezoidal rule:**

Trapezoid rule is similar to midpoint rule, but instead of taking rectangles, we use trapezoid. In other words, we approximate by inscribing polygonal chain in the graph of the function, taking separate segment for each subinterval. As Simpsons rule is also similar to mid-point rule so it also has imperfections and the answer not as much accurate as compared to Simpson’s rule.

* **Simpson’s Rule:**

Simpson’s rule is the most accurate method and the fastest convergent. The easiest way to see this is the more complicated function, where none of the rules find precise solution. For example, for f(x) =x3 or f(x) =sin(x) we can see, that the error for n= 1 is big, but it rapidly decreases for n= 4 and n= 10.

* **Boole’s rule:**

Boole’s rule is also quite accurate method just like Simpson’s rule as Simpson’s rule is for even “n” whereas Boole’s is for odd “n” so it had a just a small difference in accuracy. Just the fact that as many numbers of the answer would be closer to the actual solution.

**BEST ACCURATE & EFFICIENT METHOD: SIMPSON’S RULE**

**SOLUTION OF ORDINARY DIFFERENTIAL EQUATIONS**

There are a few basic methods for the solution of ordinary differential equations, differing in a way of approximation:

* Euler’s Method
* Huen’s Method
* 4-RK Method
* **Euler’s Method:**

The simplest numerical method for solving the initial value problem is called Euler’s method. We first define it and give some numerical illustrations, and then we analyze it mathematically. Euler’s method is not an efficient numerical method, but many of the ideas involved in the numerical solution of differential equations are introduced most simply with it. To obtain an approximate solution some form of interpolation must be used. We will not consider that problem here, although there are standard techniques from the theory of interpolation that can be easily applied.For the initial guess, use y0=Y0 or some close approximation of Y0. SometimesY0 is obtained empirically and thus may be known only approximately. Formula (2.5) gives a rule for computing 1,y2,...,yn in succession. This is typical of most numerical methods for solving ordinary differential equations.

* **Huen’s Method:**

Euler's method is used as the foundation for Heun's method. Euler's method uses the line tangent to the function at the beginning of the interval as an estimate of the slope of the function over the interval, assuming that if the step size is small, the error will be small. However, even when extremely small step sizes are used, over a large number of steps the error starts to accumulate and the estimate diverges from the actual functional value.

* **4-RK Method:**

To improve on the speed of convergence of Euler’s method, we look for approximations to Y(tn+1)that are more accurate than the approximation. which led to Euler’s method. One of the most popular methods with a constant step size is the fourth-order Runge-Kutta (RK4) method. Reasonably the Runge-Kutta method can [16] obtain the accuracy of a Taylor Series approximation without the need of higher derivative calculations. This method can be considered as the basic form of other methods. However, in terms of error estimation, the one-step method with an adaptive step size like the Runge-Kutta-Fehlberg (RKF) method [17, 18] gives better error estimation than one-step method with a constant step size like the Runge-Kutta method. At each step the Runge-Kutta-Fehlberg method described the calculation of two Runge-Kutta methods of different order (RK4 and RK5). If the two answers are close enough then continue for the next step with the same step size. For a fixed accuracy the step size should be reduced. If the answers agree with more significant digits than required, the step size is increased. So we conclude that the one-step algorithm method associated with an adaptive step size automatically organizes the step size as a recompositing to the calculation truncation errors. This method has shown that it works in the case of nonlinear models and, hence, its application is found in wide range of deterministic and stochastic problems, linear and nonlinear problems, physics, biology, and chemical reactions problems, and so forth.

**Conclusion:**

We try to compare the solutions by some numerical techniques when we apply the methods on some mathematical biology problems. The Runge-Kutta-Fehlberg (RKF) method is a promising method to give an approximate solution of nonlinear ordinary differential equation systems

**BEST ACCURATE & EFFICIENT METHOD: 4-RK METHOD**

**SYSTEM OF LINEAR EQUATIONS ITERATIVE METHOD**

There are a few basic methods for the system of linear equations (iterative method), differing in a way of approximation and number of converging iterations:

* Gauss-seidel Method
* Jacobi Method

The term” iterative method” refers to a wide range of techniques that use successive approximations to obtain more accurate solutions to a linear system at each step. Stationary methods are older, simpler to understand and implement, but usually not as effective. Nonstationary methods are a relatively recent development; their analysis is usually harder to understand, but they can be highly effective. The nonstationary methods we present are based on the idea of sequences of orthogonal vectors. The rate at which an iterative method converges depends greatly on the spectrum of the coefficient matrix. Hence, iterative methods usually involve a second matrix that trans-forms the coefficient matrix into one with a more favorable spectrum.

* **GAUSS-SEIDEL METHOD:**

The two iterative methods of solving system of linear equation has been compared, the iterative methods are used for solving sparse and dense system of linear equation and the methods were being considered are: Jacobi method and Gauss-Seidel method. The results show that Gauss-Seidel method is more efficient than Jacobi method by considering maximum number of iterations required to converge and accuracy.

* **JACOBI METHOD:**

The Jacobi method used had a greater number of iterations as well as it took more time to compute the solution so gauss-seidel method was better and faster as compared to Jacobi method in terms of accuracy as well.

**Conclusion**:

We have compared two method implemented to solve systems of linear equations. This paper can help us to decide, which method is better. After all research, we can conclude that Gauss-Seidel method is better method than Jacobi method, because it faster and required a smaller number of iterations. So, if we want to get better results and do it faster, we should use Gauss-Seidel method.

**BEST ACCURATE & EFFICIENT METHOD: GAUSS-SEIDEL METHOD**

PROJECT SOURCE CODE

**CODE STARTS HERE: TOTAL LINES OF CODE: 2563**

**#include<iostream>**

**#include<math.h>**

**#include<iomanip>**

**#include<queue>**

**#include<string>**

**#include <sstream>**

**#include "fparser.hh"**

**using namespace std;**

**class bisection**

**{**

**private:**

**long double ebsilon;**

**double \*Lbound,\*Ubound;**

**int fixval;**

**double \*p;**

**long double error;**

**int iteration;**

**string funstring;**

**FunctionParser func;**

**double templ,tempu;**

**///////////////////////////// PRIVATE FUNCTIONS //////////////////////////**

**void backupbounds(void)**

**{**

**templ=\*Lbound;**

**tempu=\*Ubound;**

**}**

**void restorebounds(void)**

**{**

**\*Lbound=templ;**

**\*Ubound=tempu;**

**}**

**//////////////////////////////////////////////////////////////////////////**

**public:**

**bisection()**

**{**

**func.AddConstant("pi", 3.1415926535897932);**

**Lbound=new double[1];**

**Ubound=new double[1];**

**p=new double[1];**

**}**

**void setparameters(void)**

**{**

**\*p=iteration=0;**

**error=999;**

**system("cls");**

**cout<<"\nenter value for EBSILON:";**

**cin>>ebsilon;**

**cout<<"\nenter value for lower bound:";**

**cin>>Lbound[0];**

**cout<<"\nenter value for upper bound:";**

**cin>>Ubound[0];**

**cout<<"\nenter fix value:";**

**cin>>fixval;**

**while(true)**

**{**

**cout << "enter function f(x) = ";**

**cin>>funstring;**

**if(cin.fail())**

**continue;**

**int res = func.Parse(funstring, "x");**

**if(res < 0)**

**break;**

**cout << string(res+7, ' ') << "^\n"<< func.ErrorMsg() << "\n\n";**

**}**

**backupbounds();**

**printdetails();**

**}**

**void evaluate(void)**

**{**

**restorebounds();**

**if(!((func.Eval(Lbound)>0 && func.Eval(Ubound)<0) || (func.Eval(Lbound)<0 && func.Eval(Ubound)>0)))**

**{**

**cout<<"\nsigns are not opposite";**

**system("pause");**

**return;**

**}**

**else**

**{**

**queue <long double> pq;**

**cout<<"\n ITERATIONS N | A | B | P | ERROR"<<endl;**

**while(error>=ebsilon)**

**{**

**\*p=(\*Lbound+\*Ubound)/2;**

**pq.push(\*p);**

**if(func.Eval(p)>=0)**

**\*Ubound=\*p;**

**else**

**\*Lbound=\*p;**

**if(iteration==0)**

**{**

**iteration++;**

**continue;**

**}**

**error=fabsf(\*p-pq.front());**

**cout<<setprecision(fixval)<<" "<<iteration<<" "<<Lbound[0]<<" "<<Ubound[0]<<" "<<\*p<<" "<<\*p<<"-"<<pq.front()<<" = "<<error<<endl;**

**pq.pop();**

**iteration++;**

**}**

**}**

**system("pause");**

**}**

**void printdetails(void)**

**{**

**system("cls");**

**cout<<"parameters set by the user are :"<<endl;**

**cout<<"\nfunctions is f(x) = "<<funstring<<" with BOUNDS=["<<Lbound[0]<<","<<Ubound[0]<<"]";**

**cout<<"\nEBSILON = "<<ebsilon;**

**cout<<"\nFIX VALUE="<<fixval<<endl;**

**cout<<"\n\nAre the details shown correct ? (y/n) = ";**

**char cho;**

**cin>>cho;**

**if(cho=='n')**

**setparameters();**

**system("pause");**

**}**

**};**

**class falseposition**

**{**

**private:**

**long double ebsilon;**

**double \*Lbound,\*Ubound;**

**int fixval;**

**double \*p;**

**long double error;**

**int iteration;**

**string funstring;**

**FunctionParser func;**

**double templ,tempu;**

**///////////////////////////// PRIVATE FUNCTIONS //////////////////////////**

**void backupbounds(void)**

**{**

**templ=\*Lbound;**

**tempu=\*Ubound;**

**}**

**void restorebounds(void)**

**{**

**\*Lbound=templ;**

**\*Ubound=tempu;**

**}**

**//////////////////////////////////////////////////////////////////////////**

**public:**

**falseposition()**

**{**

**func.AddConstant("pi", 3.1415926535897932);**

**Lbound=new double[1];**

**Ubound=new double[1];**

**p=new double[1];**

**}**

**void setparameters(void)**

**{**

**\*p=iteration=0;**

**error=999;**

**system("cls");**

**cout<<"\nenter value for EBSILON:";**

**cin>>ebsilon;**

**cout<<"\nenter value for lower bound:";**

**cin>>Lbound[0];**

**cout<<"\nenter value for upper bound:";**

**cin>>Ubound[0];**

**cout<<"\nenter fix value:";**

**cin>>fixval;**

**while(true)**

**{**

**cout << "enter function f(x) = ";**

**cin>>funstring;**

**if(cin.fail())**

**continue;**

**int res = func.Parse(funstring, "x");**

**if(res < 0)**

**break;**

**cout << string(res+7, ' ') << "^\n"<< func.ErrorMsg() << "\n\n";**

**}**

**backupbounds();**

**}**

**void evaluate(void)**

**{**

**restorebounds();**

**cout<<"\nFALSE POSITION///////"<<endl;**

**if(!((func.Eval(Lbound)>0 && func.Eval(Ubound)<0) || (func.Eval(Lbound)<0 && func.Eval(Ubound)>0)))**

**{**

**cout<<"\nsigns are not opposite";**

**return;**

**}**

**else**

**{**

**queue <long double> pq;**

**cout<<"\nITERATIONS N| f(a) | f(b) | P | f(p) | ERROR"<<endl;**

**while(error>=ebsilon)**

**{**

**\*p=(\*Lbound\*func.Eval(Ubound)-\*Ubound\*func.Eval(Lbound))/(func.Eval(Ubound)-func.Eval(Lbound));**

**pq.push(\*p);**

**if(func.Eval(p)>=0)**

**\*Ubound=\*p;**

**else**

**\*Lbound=\*p;**

**if(iteration==0)**

**{**

**iteration++;**

**continue;**

**}**

**error=fabsf(\*p-pq.front());**

**cout<<setprecision(fixval+1)<<" "<<iteration<<" "<<func.Eval(Lbound)<<" "<<func.Eval(Ubound)<<" "<<\*p<<" "<<func.Eval(p)<<" "<<\*p<<"-"<<pq.front()<<" = "<<error<<endl;**

**pq.pop();**

**iteration++;**

**}**

**}**

**system("pause");**

**printdetails();**

**}**

**void printdetails(void)**

**{**

**system("cls");**

**cout<<"parameters set by the user are :"<<endl;**

**cout<<"\nfunctions is f(x) = "<<funstring<<" with BOUNDS=["<<Lbound[0]<<","<<Ubound[0]<<"]";**

**cout<<"\nEBSILON = "<<ebsilon;**

**cout<<"\nFIX VALUE="<<fixval<<endl;**

**cout<<"\n\nAre the details shown correct ? (y/n) = ";**

**char cho;**

**cin>>cho;**

**if(cho=='n')**

**setparameters();**

**system("pause");**

**}**

**};**

**class newton\_method**

**{**

**private:**

**long double ebsilon;**

**double \*Lbound,\*Ubound;**

**int fixval;**

**double \*p,\*p0;**

**long double error;**

**int iteration;**

**string funstring,FuncDerivativeString;**

**FunctionParser func,derivative;**

**double templ,tempu;**

**///////////////////////////// PRIVATE FUNCTIONS //////////////////////////**

**void backupbounds(void)**

**{**

**templ=\*Lbound;**

**tempu=\*Ubound;**

**}**

**void restorebounds(void)**

**{**

**\*Lbound=templ;**

**\*Ubound=tempu;**

**}**

**//////////////////////////////////////////////////////////////////////////**

**public:**

**newton\_method()**

**{**

**func.AddConstant("pi", 3.1415926535897932);**

**Lbound=new double[1];**

**Ubound=new double[1];**

**p=new double[1];**

**p0=new double[1];**

**}**

**void setparameters(void)**

**{**

**ebsilon=-1;**

**\*p=iteration=0;**

**error=999;**

**system("cls");**

**cout<<"\nenter value for lower bound:";**

**cin>>Lbound[0];**

**cout<<"\nenter value for upper bound:";**

**cin>>Ubound[0];**

**cout<<"\nenter fix value:";**

**cin>>fixval;**

**while(true)**

**{**

**cout << "enter function f(x) = ";**

**cin>>funstring;**

**if(cin.fail())**

**continue;**

**int res = func.Parse(funstring, "x");**

**if(res < 0)**

**break;**

**cout << string(res+7, ' ') << "^\n"<< func.ErrorMsg() << "\n\n";**

**}**

**/////////////////////TAKING DERIVATIVE INPUT/////////////////////////**

**while(true)**

**{**

**cout << "enter function's derivative f'(x) = ";**

**cin>>FuncDerivativeString;**

**if(cin.fail())**

**continue;**

**int res = derivative.Parse(FuncDerivativeString, "x");**

**if(res < 0)**

**break;**

**cout << string(res+7, ' ') << "^\n"<< derivative.ErrorMsg() << "\n\n";**

**}**

**backupbounds();**

**\*p0=((\*Lbound)+(\*Ubound))/2;**

**printdetails();**

**}**

**void evaluate(int method)**

**{**

**restorebounds();**

**switch(method)**

**{**

**case 1:**

**{**

**cout<<"\nEnter the value of EPSILON = ";**

**cin>>ebsilon;**

**system("cls");**

**cout<<"\n\t\t\t\t\t\tNEWTON METHOD";**

**cout<<"\n ITERATIONS N | f(pn-1) | f'(pn-1) | P | ERROR"<<endl;**

**while(error>=ebsilon)**

**{**

**/////////////////////////////////////////**

**\*p=\*p0-(func.Eval(p0)/derivative.Eval(p0));//EQUATION**

**/////////////////////////////////////////**

**error=fabsf(\*p-\*p0);**

**cout<<fixed;**

**cout<<setprecision(fixval+1)<<" "<<iteration+1<<" "<<func.Eval(p0)<<" "<<derivative.Eval(p0)<<" "<<\*p<<" "<<\*p<<"-"<<\*p0<<" = "<<error<<endl;**

**iteration++;**

**\*p0=\*p;**

**}**

**}**

**break;**

**case 2:**

**{**

**cout<<"\nplease enter the number of iterations : ";**

**int noit;**

**cin>>noit;**

**cout<<"\n\t\t\t\t\t\tNEWTON METHOD";**

**cout<<"\n ITERATIONS N | f(pn-1) | f'(pn-1) | P | ERROR"<<endl;**

**while(iteration<=noit)**

**{**

**/////////////////////////////////////////**

**\*p=\*p0-(func.Eval(p0)/derivative.Eval(p0));//EQUATION**

**/////////////////////////////////////////**

**error=fabsf(\*p-\*p0);**

**cout<<fixed;**

**cout<<setprecision(fixval)<<" "<<iteration+1<<" "<<func.Eval(p0)<<" "<<derivative.Eval(p0)<<" "<<\*p<<" "<<\*p<<"-"<<\*p0<<" = "<<error<<endl;**

**iteration++;**

**\*p0=\*p;**

**}**

**}**

**break;**

**default:**

**{**

**cout<<"\nwrong choice selected!!";**

**return;**

**}**

**}**

**system("pause");**

**}**

**void printdetails(void)**

**{**

**system("cls");**

**cout<<"parameters set by the user are :"<<endl;**

**cout<<"\nfunctions is f(x) = "<<funstring<<" with BOUNDS=["<<Lbound[0]<<","<<Ubound[0]<<"]";**

**cout<<"\nderivative of function f'(x) = "<<FuncDerivativeString;**

**if(ebsilon!=-1)**

**cout<<"\nEBSILON = "<<ebsilon;**

**cout<<"\nFIX VALUE="<<fixval;**

**cout<<"\nAssigning Po = ("<<\*Lbound<<"+"<<\*Ubound<<")/2 = "<<\*p0<<endl;**

**cout<<"\n\nAre the details shown correct ? (y/n) = ";**

**char cho;**

**cin>>cho;**

**if(cho=='n')**

**setparameters();**

**system("pause");**

**}**

**};**

**class secant\_method**

**{**

**private:**

**long double ebsilon;**

**double \*Lbound,\*Ubound;**

**int fixval;**

**double \*p,\*p0,\*p1;**

**long double error;**

**int iteration;**

**string funstring;**

**FunctionParser func;**

**double templ,tempu;**

**///////////////////////////// PRIVATE FUNCTIONS //////////////////////////**

**void backupbounds(void)**

**{**

**templ=\*Lbound;**

**tempu=\*Ubound;**

**}**

**void restorebounds(void)**

**{**

**\*Lbound=templ;**

**\*Ubound=tempu;**

**}**

**//////////////////////////////////////////////////////////////////////////**

**public:**

**secant\_method()**

**{**

**func.AddConstant("pi", 3.1415926535897932);**

**Lbound=new double;**

**Ubound=new double;**

**p=new double;**

**p0=new double;**

**p1=new double;**

**}**

**void setparameters(void)**

**{**

**\*p=0;**

**iteration=2;**

**error=999;**

**ebsilon=-1;**

**system("cls");**

**cout<<"\nenter value for lower bound:";**

**cin>>Lbound[0];**

**cout<<"\nenter value for upper bound:";**

**cin>>Ubound[0];**

**cout<<"\nenter fix value:";**

**cin>>fixval;**

**while(true)**

**{**

**cout << "enter function f(x) = ";**

**cin>>funstring;**

**if(cin.fail())**

**continue;**

**int res = func.Parse(funstring, "x");**

**if(res < 0)**

**break;**

**cout << string(res+7, ' ') << "^\n"<< func.ErrorMsg() << "\n\n";**

**}**

**backupbounds();**

**printdetails();**

**}**

**void evaluate(int method)**

**{**

**restorebounds();**

**switch(method)**

**{**

**case 1:**

**{**

**cout<<"\nEnter the value of EPSILON = ";**

**cin>>ebsilon;**

**\*p0=\*Lbound;**

**\*p1=\*Ubound;**

**cout<<"\n\t\t\t\t\t\tSECANT METHOD";**

**cout<<"\n ITERATIONS N | f(pn-1) | f(pn-2) | P | ERROR"<<endl;**

**while(error>=ebsilon)**

**{**

**/////////////////////////////////////////**

**\*p=\*p1-(func.Eval(p1)\*(\*p1-\*p0))/(func.Eval(p1)-func.Eval(p0));//EQUATION**

**/////////////////////////////////////////**

**error=fabsf(\*p-\*p1);**

**cout<<fixed;**

**cout<<setprecision(fixval+1)<<" "<<iteration<<" "<<func.Eval(p1)<<" "<<func.Eval(p0)<<" "<<\*p<<" "<<\*p<<"-"<<\*p1<<" = "<<error<<endl;**

**iteration++;**

**\*p0=\*p1;**

**\*p1=\*p;**

**}**

**}**

**break;**

**case 2:**

**{**

**cout<<"\nplease enter the number of iterations (>=2): ";**

**int noit;**

**cin>>noit;**

**\*p0=\*Lbound;**

**\*p1=\*Ubound;**

**cout<<"\n\t\t\t\t\t\tSECANT METHOD";**

**cout<<"\n ITERATIONS N | f(pn-1) | f(pn-2) | P | ERROR"<<endl;**

**while(iteration<=noit)**

**{**

**/////////////////////////////////////////**

**\*p=\*p1-(func.Eval(p1)\*(\*p1-\*p0))/(func.Eval(p1)-func.Eval(p0));//EQUATION**

**/////////////////////////////////////////**

**error=fabsf(\*p-\*p1);**

**cout<<fixed;**

**cout<<setprecision(fixval)<<" "<<iteration<<" "<<func.Eval(p1)<<" "<<func.Eval(p0)<<" "<<\*p<<" "<<\*p<<"-"<<\*p1<<" = "<<error<<endl;**

**iteration++;**

**\*p0=\*p1;**

**\*p1=\*p;**

**}**

**}**

**break;**

**default:**

**{**

**cout<<"\nwrong choice selected!!";**

**return;**

**}**

**}**

**system("pause");**

**}**

**void printdetails(void)**

**{**

**system("cls");**

**cout<<"parameters set by the user are :"<<endl;**

**cout<<"\nfunctions is f(x) = "<<funstring<<" with BOUNDS=["<<Lbound[0]<<","<<Ubound[0]<<"]";**

**if(ebsilon!=-1)**

**cout<<"\nEBSILON = "<<ebsilon;**

**cout<<"\nFIX VALUE="<<fixval;**

**cout<<"\nAssigning Po = "<<\*Lbound;**

**cout<<"\nAssigning P1 = "<<\*Ubound<<endl;**

**cout<<"\n\nAre the details shown correct ? (y/n) = ";**

**char cho;**

**cin>>cho;**

**if(cho=='n')**

**setparameters();**

**system("pause");**

**}**

**};**

**class fix\_point**

**{**

**private:**

**long double ebsilon;**

**double \*Lbound,\*Ubound;**

**int fixval;**

**double \*p;**

**long double error;**

**int iteration;**

**string gx;**

**FunctionParser func;**

**public:**

**fix\_point()**

**{**

**func.AddConstant("pi", 3.1415926535897932);**

**Lbound=new double[1];**

**Ubound=new double[1];**

**p=new double[1];**

**}**

**void setparameters(void)**

**{**

**ebsilon=-1;**

**\*p=iteration=1;**

**error=999;**

**system("cls");**

**cout<<"\nenter value for lower bound:";**

**cin>>Lbound[0];**

**cout<<"\nenter value for upper bound:";**

**cin>>Ubound[0];**

**cout<<"\nEnter the value of EPSILON = ";**

**cin>>ebsilon;**

**cout<<"\nenter fix value:";**

**cin>>fixval;**

**while(true)**

**{**

**cout << "enter function g(x) (Equation in terms of x) = ";**

**cin>>gx;**

**if(cin.fail())**

**continue;**

**int res = func.Parse(gx, "x");**

**if(res < 0)**

**break;**

**cout << string(res+7, ' ') << "^\n"<< func.ErrorMsg() << "\n\n";**

**}**

**\*p=\*Lbound;**

**printdetails();**

**}**

**void evaluate(void)**

**{**

**long double temp=\*p;**

**\*p=func.Eval(p);**

**error=fabs(temp-\*p);**

**system("cls");**

**cout<<"\n\t\t\t\t\t\tFIX POINT METHOD";**

**cout<<"\n ITERATIONS N | P | ERROR"<<endl;**

**while(error>=ebsilon)**

**{**

**cout<<setprecision(fixval)<<"\t"<<iteration<<"\t\t\t"<<\*p<<"\t\t"<<error<<endl;**

**temp=\*p;**

**\*p=func.Eval(p);**

**error=fabs(temp-\*p);**

**iteration++;**

**}**

**}**

**void printdetails(void)**

**{**

**system("cls");**

**cout<<"parameters set by the user are :"<<endl;**

**cout<<"\nfunctions is g(x) = "<<gx<<" with BOUNDS=["<<Lbound[0]<<","<<Ubound[0]<<"]";**

**cout<<"\nEBSILON = "<<ebsilon;**

**cout<<"\nFIX VALUE="<<fixval;**

**cout<<"\n Po = "<<\*p<<endl;**

**cout<<"\n\nAre the details shown correct ? (y/n) = ";**

**char cho;**

**cin>>cho;**

**if(cho=='n')**

**setparameters();**

**system("pause");**

**}**

**};**

**class euler**

**{**

**private:**

**int fixval;**

**long double error,upperlimit,lowerlimit;**

**double \*vars; //vars[0] (holds X) vars[1] (holds y) vars[2] (holds h)**

**int iteration;**

**string finalfunc,Gfunc,Afunc;**

**FunctionParser mainfunc,realfunc;**

**public:**

**euler()**

**{**

**upperlimit=lowerlimit=0;**

**vars=new double[3];**

**mainfunc.AddConstant("pi", 3.1415926535897932);**

**realfunc.AddConstant("pi", 3.1415926535897932);**

**}**

**void setparameters(void)**

**{**

**iteration=0;**

**error=999;**

**finalfunc="y+h\*(";**

**system("cls");**

**cout<<"\nenter value for Xo:";**

**cin>>vars[0];**

**cout<<"\nenter value for Yo:";**

**cin>>vars[1];**

**cout<<"\nenter value for H:";**

**cin>>vars[2];**

**cout<<"\nenter value for upper limit or the approximation value :";**

**cin>>upperlimit;**

**cout<<"\nenter fix value:";**

**cin>>fixval;**

**while(true)**

**{**

**cout << "enter function Y' or f(x,y)= ";**

**cin>>Gfunc;**

**if(cin.fail())**

**continue;**

**finalfunc=finalfunc+Gfunc+')';**

**int res = mainfunc.Parse(finalfunc, "x,y,h");**

**if(res < 0)**

**break;**

**cout << string(res+7, ' ') << "^\n"<< mainfunc.ErrorMsg() << "\n\n";**

**}**

**cout<<"\ndo you want to set Actual function F(x) ? (y/n) : ";**

**char choice;**

**cin>>choice;**

**if(choice=='y')**

**while(true)**

**{**

**cout << "enter function f(x) = ";**

**cin>>Afunc;**

**if(cin.fail())**

**continue;**

**int res =realfunc.Parse(Afunc, "x");**

**if(res < 0)**

**break;**

**cout << string(res+7, ' ') << "^\n"<< realfunc.ErrorMsg() << "\n\n";**

**}**

**else**

**Afunc="null";**

**lowerlimit=vars[0];**

**printdetails();**

**}**

**void evaluate(void)**

**{**

**queue <long double>temp;**

**if(Afunc=="null")**

**{**

**cout<<"\nN | X | Y \n"<<endl;**

**while(vars[0]<=upperlimit)**

**{**

**cout<<setprecision(fixval+1)<<iteration<<"\t\t\t"<<vars[0]<<"\t\t\t"<<vars[1]<<endl;**

**vars[1]=mainfunc.Eval(vars);**

**vars[0]+=vars[2];**

**iteration++;**

**}**

**}**

**else**

**{**

**cout<<"\nN | X | Y' | Y(x) | ERROR\n"<<endl;**

**double temp=realfunc.Eval(vars);**

**while(vars[0]<=upperlimit)**

**{**

**cout<<setprecision(fixval+1)<<iteration<<"\t\t\t"<<vars[0]<<"\t\t\t"<<vars[1]<<"\t\t\t"<<temp<<"\t\t\t"<<fabs(vars[1]-temp)<<endl;**

**vars[1]=mainfunc.Eval(vars);**

**vars[0]+=vars[2];**

**temp=realfunc.Eval(vars);**

**iteration++;**

**}**

**}**

**system("pause");**

**}**

**void printdetails(void)**

**{**

**system("cls");**

**cout<<"parameters set by the user are :"<<endl;**

**cout<<"\nfunctions is Y' or f(x,y)= "<<Gfunc<<" with "<<lowerlimit<<"<=X<="<<upperlimit;**

**if(Afunc!="null")**

**cout<<"\nActual function is f(x) : "<<Afunc;**

**cout<<"\nfinal Euler function is : "<<finalfunc;**

**cout<<"\n H = "<<vars[2];**

**cout<<"\n Xo = "<<vars[0];**

**cout<<"\n Yo = "<<vars[1];**

**cout<<"\nFIX VALUE="<<fixval<<endl;**

**cout<<"\n\nAre the details shown correct ? (y/n) = ";**

**char cho;**

**cin>>cho;**

**if(cho=='n')**

**setparameters();**

**system("pause");**

**}**

**};**

**class Meuler**

**{**

**private:**

**int fixval;**

**long double error,upperlimit,lowerlimit;**

**double \*vars; //vars[0] (holds X) vars[1] (holds y) vars[2] (holds h)**

**int iteration;**

**string finalfunc,Gfunc,Afunc,modeulerfunc;**

**FunctionParser mainfunc,realfunc,modeuler,givenfunction;**

**public:**

**Meuler()**

**{**

**upperlimit=lowerlimit=0;**

**vars=new double[3];**

**mainfunc.AddConstant("pi", 3.1415926535897932);**

**realfunc.AddConstant("pi", 3.1415926535897932);**

**}**

**void setparameters(void)**

**{**

**iteration=0;**

**error=999;**

**modeulerfunc="y+(h/2)\*(A+B)";**

**modeuler.Parse(modeulerfunc,"y,h,A,B");**

**finalfunc="y+h\*(";**

**system("cls");**

**cout<<"\nenter value for Xo:";**

**cin>>vars[0];**

**cout<<"\nenter value for Yo:";**

**cin>>vars[1];**

**cout<<"\nenter value for H:";**

**cin>>vars[2];**

**cout<<"\nenter value for upper limit or the approximation value :";**

**cin>>upperlimit;**

**cout<<"\nenter fix value:";**

**cin>>fixval;**

**while(true)**

**{**

**cout << "enter function Y' or f(X,Y) = ";**

**cin>>Gfunc;**

**if(cin.fail())**

**continue;**

**finalfunc=finalfunc+Gfunc+')';**

**int res = mainfunc.Parse(finalfunc, "x,y,h");**

**if(res < 0)**

**break;**

**cout << string(res+7, ' ') << "^\n"<< mainfunc.ErrorMsg() << "\n\n";**

**}**

**givenfunction.Parse(Gfunc,"x,y");**

**cout<<"\ndo you want to set Actual function F(x) ? (y/n) : ";**

**char choice;**

**cin>>choice;**

**if(choice=='y')**

**while(true)**

**{**

**cout << "enter function f(x) = ";**

**cin>>Afunc;**

**if(cin.fail())**

**continue;**

**int res =realfunc.Parse(Afunc, "x");**

**if(res < 0)**

**break;**

**cout << string(res+7, ' ') << "^\n"<< realfunc.ErrorMsg() << "\n\n";**

**}**

**else**

**Afunc="null";**

**lowerlimit=vars[0];**

**printdetails();**

**}**

**void evaluate(void)**

**{**

**if(Afunc=="null")**

**{**

**double arr[4]; //[0] y ,[1] h,[2] A,[3] B y+(h/2)\*(A+B)**

**double B[2]; //0 x,1 y,2 h array ka structure**

**B[0]=vars[0]+vars[2];**

**B[1]=mainfunc.Eval(vars);**

**arr[1]=vars[2];**

**cout<<"\nN | X | Y \n"<<endl;**

**while(vars[0]<=upperlimit)**

**{**

**cout<<setprecision(fixval+1)<<iteration<<"\t\t"<<vars[0]<<"\t\t\t"<<vars[1]<<endl;**

**B[0]=vars[0]+vars[2];**

**B[1]=mainfunc.Eval(vars);**

**arr[3]=givenfunction.Eval(B);**

**arr[2]=givenfunction.Eval(vars);**

**arr[0]=vars[1];**

**vars[1]=modeuler.Eval(arr);**

**vars[0]+=vars[2];**

**iteration++;**

**}**

**}**

**else**

**{**

**double arr[4]; //[0] y ,[1] h,[2] A,[3] B y+(h/2)\*(A+B)**

**double B[2],temp; //0 x,1 y,2 h array ka structure**

**B[0]=vars[0]+vars[2];**

**B[1]=mainfunc.Eval(vars);**

**arr[1]=vars[2];**

**temp=vars[1];**

**cout<<"\nN | X | Y | Y(x) | ERROR\n"<<endl;**

**while(vars[0]<=upperlimit)**

**{**

**cout<<setprecision(fixval+1)<<iteration<<"\t\t"<<vars[0]<<"\t\t\t"<<vars[1]<<"\t\t\t"<<temp<<"\t\t\t"<<fabs(temp-vars[1])<<endl;**

**B[0]=vars[0]+vars[2];**

**B[1]=mainfunc.Eval(vars);**

**arr[3]=givenfunction.Eval(B);**

**arr[2]=givenfunction.Eval(vars);**

**arr[0]=vars[1];**

**vars[1]=modeuler.Eval(arr);**

**vars[0]+=vars[2];**

**temp=realfunc.Eval(vars);**

**iteration++;**

**}**

**}**

**system("pause");**

**}**

**void printdetails(void)**

**{**

**system("cls");**

**cout<<"parameters set by the user are :"<<endl;**

**cout<<"\nfunctions is Y' or f(x,y)= "<<Gfunc<<" with "<<lowerlimit<<"<=X<="<<upperlimit;**

**if(Afunc!="null")**

**cout<<"\nActual function is F(x) : "<<Afunc;**

**cout<<"\nfinal Euler function is : "<<finalfunc;**

**cout<<"\n H = "<<vars[2];**

**cout<<"\n Xo = "<<vars[0];**

**cout<<"\n Yo = "<<vars[1];**

**cout<<"\nFIX VALUE="<<fixval<<endl;**

**cout<<"\n\nAre the details shown correct ? (y/n) = ";**

**char cho;**

**cin>>cho;**

**if(cho=='n')**

**setparameters();**

**system("pause");**

**}**

**};**

**class Midpoint**

**{**

**private:**

**int fixval;**

**long double error,upperlimit,lowerlimit;**

**double \*vars; //vars[0] (holds X) vars[1] (holds y) vars[2] (holds h)**

**int iteration;**

**string midpointstring,givenfunctionstring,realfunctionstring;**

**FunctionParser midpoint,givenfunction,realfunction;**

**public:**

**Midpoint()**

**{**

**upperlimit=lowerlimit=0;**

**vars=new double[3];**

**givenfunction.AddConstant("pi", 3.1415926535897932);**

**realfunction.AddConstant("pi", 3.1415926535897932);**

**midpoint.AddConstant("pi", 3.1415926535897932);**

**}**

**void setparameters(void)**

**{**

**iteration=0;**

**error=999;**

**midpointstring="y+h\*(A)";**

**midpoint.Parse(midpointstring,"y,h,A");**

**system("cls");**

**cout<<"\nenter value for Xo:";**

**cin>>vars[0];**

**cout<<"\nenter value for Yo:";**

**cin>>vars[1];**

**cout<<"\nenter value for H:";**

**cin>>vars[2];**

**cout<<"\nenter value for upper limit or the approximation value :";**

**cin>>upperlimit;**

**cout<<"\nenter fix value:";**

**cin>>fixval;**

**while(true)**

**{**

**cout << "enter function Y' or F(X,Y) = ";**

**cin>>givenfunctionstring;**

**if(cin.fail())**

**continue;**

**int res = givenfunction.Parse(givenfunctionstring, "x,y");**

**if(res < 0)**

**break;**

**cout << string(res+7, ' ') << "^\n"<< givenfunction.ErrorMsg() << "\n\n";**

**}**

**cout<<"\ndo you want to set Actual function F(x) ? (y/n) : ";**

**char choice;**

**cin>>choice;**

**if(choice=='y')**

**while(true)**

**{**

**cout << "enter function F(x) = ";**

**cin>>realfunctionstring;**

**if(cin.fail())**

**continue;**

**int res =realfunction.Parse(realfunctionstring, "x");**

**if(res < 0)**

**break;**

**cout << string(res+7, ' ') << "^\n"<< realfunction.ErrorMsg() << "\n\n";**

**}**

**else**

**realfunctionstring="null";**

**lowerlimit=vars[0];**

**printdetails();**

**}**

**void evaluate(void)**

**{**

**if(realfunctionstring=="null")**

**{**

**double B[2],A[2],mainvars[3];**

**mainvars[1]=vars[2];**

**cout<<"\nN | X | Mid point \n"<<endl;**

**while(vars[0]<=upperlimit)**

**{**

**cout<<setprecision(fixval+1)<<iteration<<"\t\t"<<vars[0]<<"\t\t\t"<<vars[1]<<endl;**

**B[0]=vars[0];B[1]=vars[1];**

**A[0]=vars[0]+(vars[2]/2);A[1]=vars[1]+(vars[2]/2)\*givenfunction.Eval(B);**

**mainvars[2]=givenfunction.Eval(A);**

**mainvars[0]=vars[1];**

**vars[1]=midpoint.Eval(mainvars);**

**vars[0]+=vars[2];**

**iteration++;**

**}**

**}**

**else**

**{**

**double B[2],A[2],mainvars[3],temp;**

**mainvars[1]=vars[2];**

**temp=vars[1];**

**cout<<"\nN | X | Mid point | Y(x) | ERROR\n"<<endl;**

**while(vars[0]<=upperlimit)**

**{**

**cout<<setprecision(fixval+1)<<iteration<<"\t\t"<<vars[0]<<"\t\t\t"<<vars[1]<<"\t\t\t"<<temp<<"\t\t\t"<<fabs(temp-vars[1])<<endl;**

**B[0]=vars[0];B[1]=vars[1];**

**A[0]=vars[0]+(vars[2]/2);A[1]=vars[1]+(vars[2]/2)\*givenfunction.Eval(B);**

**mainvars[2]=givenfunction.Eval(A);**

**mainvars[0]=vars[1];**

**vars[1]=midpoint.Eval(mainvars);**

**vars[0]+=vars[2];**

**temp=realfunction.Eval(vars);**

**iteration++;**

**}**

**}**

**system("pause");**

**}**

**void printdetails(void)**

**{**

**system("cls");**

**cout<<"parameters set by the user are :"<<endl;**

**cout<<"\nfunctions is Y' or f(x,y) = "<<givenfunctionstring<<" with "<<lowerlimit<<"<=X<="<<upperlimit;**

**if(realfunctionstring!="null")**

**cout<<"\nActual/Real function is F(x) : "<<realfunctionstring;**

**cout<<"\n H = "<<vars[2];**

**cout<<"\n Xo = "<<vars[0];**

**cout<<"\n Yo = "<<vars[1];**

**cout<<"\nFIX VALUE="<<fixval<<endl;**

**cout<<"\n\nAre the details shown correct ? (y/n) = ";**

**char cho;**

**cin>>cho;**

**if(cho=='n')**

**setparameters();**

**system("pause");**

**}**

**};**

**class Huen**

**{**

**private:**

**int fixval;**

**long double error,upperlimit,lowerlimit,k1,k2;**

**double \*vars,\*vars4huen;**

**int iteration;**

**string fxy,yx,huenstring;**

**FunctionParser Fxy,Yx,Huenstring;**

**public:**

**Huen()**

**{**

**upperlimit=lowerlimit=0;**

**vars=new double[3];**

**vars4huen=new double[4];//0 y 1 k1 2 k2 3 h**

**Fxy.AddConstant("pi", 3.1415926535897932);**

**Yx.AddConstant("pi", 3.1415926535897932);**

**Huenstring.AddConstant("pi", 3.1415926535897932);**

**}**

**void setparameters(void)**

**{**

**iteration=0;**

**error=999;**

**huenstring="y+((1/2\*A)+(1/2\*B))\*h";//A == K1 and B == K2**

**Huenstring.Parse(huenstring,"y,A,B,h");**

**system("cls");**

**cout<<"\nenter value for Xo:";**

**cin>>vars[0];**

**cout<<"\nenter value for Yo:";**

**cin>>vars[1];**

**cout<<"\nenter value for H:";**

**cin>>vars[2];**

**cout<<"\nenter value for upper limit or the approximation value :";**

**cin>>upperlimit;**

**cout<<"\nenter fix value:";**

**cin>>fixval;**

**while(true)**

**{**

**cout << "enter function Y' or F(X,Y) = ";**

**cin>>fxy;**

**if(cin.fail())**

**continue;**

**int res = Fxy.Parse(fxy, "x,y");**

**if(res < 0)**

**break;**

**cout << string(res+7, ' ') << "^\n"<< Fxy.ErrorMsg() << "\n\n";**

**}**

**cout<<"\ndo you want to set Actual function F(x) or Y(x) ? (y/n) : ";**

**char choice;**

**cin>>choice;**

**if(choice=='y')**

**while(true)**

**{**

**cout << "enter function F(x) of Y(x) = ";**

**cin>>yx;**

**if(cin.fail())**

**continue;**

**int res = Yx.Parse(yx, "x");**

**if(res < 0)**

**break;**

**cout << string(res+7, ' ') << "^\n"<< Yx.ErrorMsg() << "\n\n";**

**}**

**else**

**yx="null";**

**lowerlimit=vars[0];**

**printdetails();**

**}**

**void evaluate(void)**

**{**

**if(yx=="null")**

**{**

**double temp[2];**

**cout<<setprecision(fixval)<<"\tX | Y "<<endl;**

**while(vars[0]<=upperlimit)**

**{**

**cout<<setprecision(fixval+1)<<"\t"<<vars[0]<<"\t\t"<<vars[1]<<endl;**

**k1=Fxy.Eval(vars);**

**temp[0]=vars[0]+vars[2];temp[1]=vars[1]+k1\*vars[2];**

**k2=Fxy.Eval(temp);**

**vars4huen[0]=vars[1];vars4huen[1]=k1;vars4huen[2]=k2;vars4huen[3]=vars[2];**

**vars[0]+=vars[2];**

**vars[1]=Huenstring.Eval(vars4huen);**

**}**

**}**

**else**

**{**

**double temp[2],ER=vars[1];**

**cout<<"\tX | Y | ACTUAL | ERROR"<<endl;**

**while(vars[0]<=upperlimit)**

**{**

**cout<<setprecision(fixval+1)<<"\t"<<vars[0]<<"\t\t"<<vars[1]<<"\t\t\t"<<ER<<"\t\t\t"<<fabs(ER-vars[1])<<endl;**

**k1=Fxy.Eval(vars);**

**temp[0]=vars[0]+vars[2];temp[1]=vars[1]+k1\*vars[2];**

**k2=Fxy.Eval(temp);**

**vars4huen[0]=vars[1];vars4huen[1]=k1;vars4huen[2]=k2;vars4huen[3]=vars[2];**

**vars[0]+=vars[2];**

**ER=Yx.Eval(vars);**

**vars[1]=Huenstring.Eval(vars4huen);**

**}**

**}**

**system("pause");**

**}**

**void printdetails(void)**

**{**

**system("cls");**

**cout<<"parameters set by the user are :"<<endl;**

**cout<<"\nfunctions is Y' or f(x,y) = "<<fxy<<" with "<<lowerlimit<<"<=X<="<<upperlimit;**

**if(yx!="null")**

**cout<<"\nActual/Real function is F(x) : "<<yx;**

**cout<<"\n H = "<<vars[2];**

**cout<<"\n Xo = "<<vars[0];**

**cout<<"\n Yo = "<<vars[1];**

**cout<<"\nFIX VALUE="<<fixval<<endl;**

**cout<<"\n\nAre the details shown correct ? (y/n) = ";**

**char cho;**

**cin>>cho;**

**if(cho=='n')**

**setparameters();**

**system("pause");**

**}**

**};**

**class RK4**

**{**

**private:**

**int fixval;**

**long double error,upperlimit,lowerlimit,k1,k2,k3,k4;**

**double \*vars,\*vars4RK;**

**int iteration;**

**string fxy,yx,rkstring;**

**FunctionParser Fxy,Yx,RKstring;**

**public:**

**RK4()**

**{**

**upperlimit=lowerlimit=0;**

**vars=new double[3];**

**vars4RK=new double[5];//0 y 1 k1 2 k2 3 k3 4 k4**

**Fxy.AddConstant("pi", 3.1415926535897932);**

**Yx.AddConstant("pi", 3.1415926535897932);**

**RKstring.AddConstant("pi", 3.1415926535897932);**

**}**

**void setparameters(void)**

**{**

**iteration=0;**

**error=999;**

**rkstring="y+(1/6)\*(A+2\*B+2\*C+D)";//A == K1 , B == K2 , C == K3 , D == K4**

**RKstring.Parse(rkstring,"y,A,B,C,D");**

**system("cls");**

**cout<<"\nenter value for Xo:";**

**cin>>vars[0];**

**cout<<"\nenter value for Yo:";**

**cin>>vars[1];**

**cout<<"\nenter value for H:";**

**cin>>vars[2];**

**cout<<"\nenter value for upper limit or the approximation value :";**

**cin>>upperlimit;**

**cout<<"\nenter fix value:";**

**cin>>fixval;**

**while(true)**

**{**

**cout << "enter function Y' or F(X,Y) = ";**

**cin>>fxy;**

**if(cin.fail())**

**continue;**

**int res = Fxy.Parse(fxy, "x,y");**

**if(res < 0)**

**break;**

**cout << string(res+7, ' ') << "^\n"<< Fxy.ErrorMsg() << "\n\n";**

**}**

**cout<<"\ndo you want to set Actual function F(x) or Y(x) ? (y/n) : ";**

**char choice;**

**cin>>choice;**

**if(choice=='y')**

**while(true)**

**{**

**cout << "enter function F(x) of Y(x) = ";**

**cin>>yx;**

**if(cin.fail())**

**continue;**

**int res = Yx.Parse(yx, "x");**

**if(res < 0)**

**break;**

**cout << string(res+7, ' ') << "^\n"<< Yx.ErrorMsg() << "\n\n";**

**}**

**else**

**yx="null";**

**lowerlimit=vars[0];**

**printdetails();**

**}**

**void evaluate(void)**

**{**

**if(yx=="null")**

**{**

**double \*temp;**

**temp=new double[2];**

**cout<<setprecision(fixval)<<"\tX | Y "<<endl;**

**while(vars[0]<=upperlimit)**

**{**

**cout<<setprecision(fixval+1)<<"\t"<<vars[0]<<"\t\t"<<vars[1]<<endl;**

**k1=vars[2]\*Fxy.Eval(vars);**

**temp[0]=vars[0]+(vars[2]/2);temp[1]=vars[1]+(k1/2);**

**k2=vars[2]\*Fxy.Eval(temp);**

**temp[0]=vars[0]+(vars[2]/2);temp[1]=vars[1]+(k2/2);**

**k3=vars[2]\*Fxy.Eval(temp);**

**temp[0]=vars[0]+vars[2];temp[1]=vars[1]+k3;**

**k4=vars[2]\*Fxy.Eval(temp);**

**vars4RK[0]=vars[1];vars4RK[1]=k1;vars4RK[2]=k2;vars4RK[3]=k3;vars4RK[4]=k4;**

**vars[1]=RKstring.Eval(vars4RK);**

**vars[0]+=vars[2];**

**}**

**}**

**else**

**{**

**double \*temp;**

**temp=new double[2];**

**cout<<setprecision(fixval)<<"\tX | Y | ACTUAL | ERROR"<<endl;**

**while(vars[0]<=upperlimit)**

**{**

**cout<<setprecision(fixval+1)<<"\t"<<vars[0]<<"\t\t"<<vars[1]<<"\t\t\t"<<Yx.Eval(vars)<<"\t\t "<<fabs(vars[1]-Yx.Eval(vars))<<endl;**

**k1=vars[2]\*Fxy.Eval(vars);**

**temp[0]=vars[0]+(vars[2]/2);temp[1]=vars[1]+(k1/2);**

**k2=vars[2]\*Fxy.Eval(temp);**

**temp[0]=vars[0]+(vars[2]/2);temp[1]=vars[1]+(k2/2);**

**k3=vars[2]\*Fxy.Eval(temp);**

**temp[0]=vars[0]+vars[2];temp[1]=vars[1]+k3;**

**k4=vars[2]\*Fxy.Eval(temp);**

**vars4RK[0]=vars[1];vars4RK[1]=k1;vars4RK[2]=k2;vars4RK[3]=k3;vars4RK[4]=k4;**

**vars[1]=RKstring.Eval(vars4RK);**

**vars[0]+=vars[2];**

**}**

**}**

**system("pause");**

**}**

**void printdetails(void)**

**{**

**system("cls");**

**cout<<"parameters set by the user are :"<<endl;**

**cout<<"\nfunctions is Y' or f(x,y) = "<<fxy<<" with "<<lowerlimit<<"<=X<="<<upperlimit;**

**if(yx!="null")**

**cout<<"\nActual/Real function is F(x) : "<<yx;**

**cout<<"\n H = "<<vars[2];**

**cout<<"\n Xo = "<<vars[0];**

**cout<<"\n Yo = "<<vars[1];**

**cout<<"\nFIX VALUE="<<fixval<<endl;**

**cout<<"\n\nAre the details shown correct ? (y/n) = ";**

**char cho;**

**cin>>cho;**

**if(cho=='n')**

**setparameters();**

**system("pause");**

**}**

**};**

**class gauss**

**{**

**private:**

**int fixval,matsiz;**

**double \*vals,\*\*matrix,\*consmat;**

**double initialapprox;**

**int iteration;**

**string \*functions,\*varnames;**

**FunctionParser \*FUNCTIONS;**

**ostringstream ststream;**

**void makediagdom (void)**

**{**

**int sum=0;**

**for(int row=0;row<matsiz;row++)**

**{**

**for(int col=0;col<matsiz;col++)**

**{**

**for(int a=0;a<matsiz;a++)**

**{**

**if(a==col)**

**continue;**

**sum+=fabs(matrix[row][a]);**

**}**

**if(fabs(matrix[row][col])>=sum)**

**{**

**double \*temp,temp2;**

**temp2=consmat[row];**

**temp=matrix[row];**

**matrix[row]=matrix[col];**

**consmat[row]=consmat[col];**

**matrix[col]=temp;**

**consmat[col]=temp2;**

**break;**

**}**

**sum=0;**

**}**

**}**

**}**

**void generatefunctions (void)**

**{**

**ostringstream vnam,func;**

**for(int row=0;row<matsiz;row++)**

**{**

**vnam.str("");**

**func.str("");**

**vnam.clear();**

**func.clear();**

**func<<"(";**

**for(int col=0;col<matsiz;col++)**

**{**

**if(col==row)**

**continue;**

**func<<"("<<matrix[row][col]\*(-1)<<")\*x"<<col<<"+";**

**vnam<<"x"<<col<<",";**

**}**

**func<<"("<<consmat[row]<<")"<<")/"<<matrix[row][row];**

**functions[row]=func.str();**

**varnames[row]=vnam.str();**

**varnames[row].pop\_back();**

**//cout<<"\nfunction looks like : "<<functions[row]<<endl;**

**//cout<<"\nwith variable names : "<<varnames[row]<<endl;**

**FUNCTIONS[row].Parse(functions[row],varnames[row]);**

**}**

**}**

**public:**

**gauss()**

**{**

**system("cls");**

**cout<<"\t\t\t\t\t\tNOTES"<<endl<<endl;**

**cout<<"This module (Gauss method) will take input in matrix form"<<endl<<endl;**

**system("pause");**

**system("cls");**

**}**

**void setparameters(void)**

**{**

**system("cls");**

**cout<<"\t\t\t\t\tSET PARAMETERS";**

**cout<<"\nenter number of iterations given : ";**

**cin>>iteration;**

**cout<<"\nenter initial approximation : ";**

**cin>>initialapprox;**

**cout<<"\nenter FIX value : ";**

**cin>>fixval;**

**cout<<"\nplease enter the size of matrix : ";**

**cin>>matsiz;**

**if(matrix!=NULL)**

**{**

**for(int i = 0; i < matsiz; ++i)**

**delete[] matrix[i];**

**delete[] matrix;**

**delete [] vals;**

**delete [] consmat;**

**delete [] functions;**

**delete [] varnames;**

**delete [] FUNCTIONS;**

**}**

**consmat=new double[matsiz];**

**matrix=new double\*[matsiz];**

**for(int a=0;a<matsiz;a++)**

**matrix[a]=new double[matsiz];**

**for(int row=0;row<matsiz;row++)**

**{**

**for(int col=0;col<matsiz;col++)**

**{**

**cout<<"A["<<row<<"]["<<col<<"]->";**

**cin>>matrix[row][col];**

**}**

**cout<<"\n now enter constant value of equation "<<row+1<<" : ";**

**cin>>consmat[row];**

**cout<<endl;**

**}**

**functions = new string[matsiz];**

**varnames = new string[matsiz];**

**FUNCTIONS=new FunctionParser[matsiz];**

**vals=new double[matsiz];**

**makediagdom();**

**generatefunctions();**

**system("pause");**

**system("cls");**

**printdetails();**

**}**

**void evaluate(void)**

**{**

**string temp;**

**ststream.str("");**

**ststream.clear();**

**ststream<<"ITERATIONS\t\t";**

**for(int a=1;a<=matsiz;a++)**

**{**

**ststream<<"x"<<a;**

**ststream<<"\t\t";**

**}**

**temp=ststream.str();**

**cout<<temp<<endl<<endl;**

**double \*variables;**

**variables=new double[matsiz-1];**

**///////////////sari equations main initial approximation dalna**

**for(int a=0;a<matsiz-1;a++)**

**{**

**variables[a]=initialapprox;**

**vals[a]=initialapprox;**

**}**

**//////////////////////////////////////////////////////////////**

**//vals ke index pe ab x1,x2,x3,x4..... ki resulting values hain after**

**//substituting initial approximation value**

**//////////////////////////////////////////////////////////////**

**for(int loop=1;loop<=iteration;loop++)**

**{**

**for(int a=0;a<matsiz;a++)//equation number iss loop se aye gaa**

**{**

**//////////////////////////////////////////////////////**

**//har equation ke liyay input variables taiyr krna hai**

**int i;**

**i=0;**

**for(int b=0;b<matsiz;b++)**

**{**

**if(a==b)**

**continue;**

**variables[i]=vals[b];**

**i++;**

**}**

**//////////////////////////////////////////////////////**

**/////////////////////////////////////////////////////////////////**

**vals[a]=FUNCTIONS[a].Eval(variables);**

**//iss mai finally hum jo current iteration pe hain uss kay liyay sari**

**//equations ke output collect krliay**

**}**

**///////////////////////////NOW PRINTING////////////////////////**

**ststream.str("");**

**ststream.clear();**

**ststream<<"\t"<<loop<<"\t\t";**

**for(int a=0;a<matsiz;a++)**

**{**

**ststream<<vals[a];**

**ststream<<"\t\t";**

**}**

**temp=ststream.str();**

**cout<<temp<<endl;**

**//////////////////////////////////////////////////////////////**

**}**

**}**

**void printdetails(void)**

**{**

**system("cls");**

**cout<<"parameters set by the user are :"<<endl;**

**cout<<"\nnumber of iterations given : ";**

**cout<<iteration;**

**cout<<"\ninitial approximation : ";**

**cout<<initialapprox;**

**cout<<"\nFIX value : ";**

**cout<<fixval;**

**cout<<"\nmatrix is : "<<endl;**

**for(int row=0;row<matsiz;row++)**

**{**

**for(int col=0;col<matsiz;col++)**

**cout<<"\t"<<matrix[row][col]<<" ";**

**cout<<endl;**

**}**

**cout<<"\n\nEquations are : "<<endl;**

**for(int a=0;a<matsiz;a++)**

**{**

**cout<<"x"<<a+1<<"="<<functions[a]<<endl;**

**}**

**cout<<"\n\nAre the details shown correct ? (y/n) = ";**

**char cho;**

**cin>>cho;**

**if(cho=='n')**

**setparameters();**

**system("pause");**

**}**

**};**

**class jacob**

**{**

**private:**

**int fixval,matsiz;**

**double \*vals,\*\*matrix,\*consmat;**

**double initialapprox;**

**int iteration;**

**string \*functions,\*varnames;**

**FunctionParser \*FUNCTIONS;**

**ostringstream ststream;**

**void makediagdom (void)**

**{**

**int sum=0;**

**for(int row=0;row<matsiz;row++)**

**{**

**for(int col=0;col<matsiz;col++)**

**{**

**for(int a=0;a<matsiz;a++)**

**{**

**if(a==col)**

**continue;**

**sum+=fabs(matrix[row][a]);**

**}**

**if(fabs(matrix[row][col])>=sum)**

**{**

**double \*temp,temp2;**

**temp2=consmat[row];**

**temp=matrix[row];**

**matrix[row]=matrix[col];**

**consmat[row]=consmat[col];**

**matrix[col]=temp;**

**consmat[col]=temp2;**

**break;**

**}**

**sum=0;**

**}**

**}**

**}**

**void generatefunctions (void)**

**{**

**ostringstream vnam,func;**

**for(int row=0;row<matsiz;row++)**

**{**

**vnam.str("");**

**func.str("");**

**vnam.clear();**

**func.clear();**

**func<<"(";**

**for(int col=0;col<matsiz;col++)**

**{**

**if(col==row)**

**continue;**

**func<<"("<<matrix[row][col]\*(-1)<<")\*x"<<col<<"+";**

**vnam<<"x"<<col<<",";**

**}**

**func<<"("<<consmat[row]<<")"<<")/"<<matrix[row][row];**

**functions[row]=func.str();**

**varnames[row]=vnam.str();**

**varnames[row].pop\_back();**

**//cout<<"\nfunction looks like : "<<functions[row]<<endl;**

**//cout<<"\nwith variable names : "<<varnames[row]<<endl;**

**FUNCTIONS[row].Parse(functions[row],varnames[row]);**

**}**

**}**

**public:**

**jacob()**

**{**

**system("cls");**

**cout<<"\t\t\t\t\t\tNOTES"<<endl<<endl;**

**cout<<" This module (Jacob's method) will take input in matrix form"<<endl<<endl;**

**system("pause");**

**system("cls");**

**}**

**void setparameters(void)**

**{**

**system("cls");**

**cout<<"\t\t\t\t\tSET PARAMETERS";**

**cout<<"\nenter number of iterations given : ";**

**cin>>iteration;**

**cout<<"\nenter initial approximation : ";**

**cin>>initialapprox;**

**cout<<"\nenter FIX value : ";**

**cin>>fixval;**

**cout<<"\nplease enter the size of matrix : ";**

**cin>>matsiz;**

**if(matrix!=NULL)**

**{**

**for(int i = 0; i < matsiz; ++i)**

**delete[] matrix[i];**

**delete[] matrix;**

**delete [] vals;**

**delete [] consmat;**

**delete [] functions;**

**delete [] varnames;**

**delete [] FUNCTIONS;**

**}**

**consmat=new double[matsiz];**

**matrix=new double\*[matsiz];**

**for(int a=0;a<matsiz;a++)**

**matrix[a]=new double[matsiz];**

**for(int row=0;row<matsiz;row++)**

**{**

**for(int col=0;col<matsiz;col++)**

**{**

**cout<<"A["<<row<<"]["<<col<<"]->";**

**cin>>matrix[row][col];**

**}**

**cout<<"\n now enter constant value of equation "<<row+1<<" : ";**

**cin>>consmat[row];**

**cout<<endl;**

**}**

**functions = new string[matsiz];**

**varnames = new string[matsiz];**

**FUNCTIONS=new FunctionParser[matsiz];**

**vals=new double[matsiz];**

**makediagdom();**

**generatefunctions();**

**system("pause");**

**system("cls");**

**printdetails();**

**}**

**void evaluate(void)**

**{**

**string temp;**

**ststream.str("");**

**ststream.clear();**

**ststream<<"ITERATIONS\t\t";**

**for(int a=1;a<=matsiz;a++)**

**{**

**ststream<<"x"<<a;**

**ststream<<"\t\t";**

**}**

**temp=ststream.str();**

**cout<<temp<<endl<<endl;**

**double \*variables;**

**variables=new double[matsiz-1];**

**///////////////sari equations main initial approximation dalna**

**for(int a=0;a<matsiz-1;a++)**

**{**

**variables[a]=initialapprox;**

**}**

**double \*temp2;**

**temp2=new double[matsiz];**

**for(int a=0;a<matsiz;a++)**

**{**

**vals[a]=FUNCTIONS[a].Eval(variables);**

**temp2[a]=vals[a];**

**}**

**ststream.str("");**

**ststream.clear();**

**ststream<<"\t"<<1<<"\t\t";**

**for(int a=0;a<matsiz;a++)**

**{**

**ststream<<vals[a];**

**ststream<<"\t\t";**

**}**

**temp=ststream.str();**

**cout<<temp<<endl;**

**//////////////////////////////////////////////////////////////**

**//vals ke index pe ab x1,x2,x3,x4..... ki resulting values hain after**

**//substituting initial approximation value**

**//////////////////////////////////////////////////////////////**

**for(int loop=2;loop<=iteration;loop++)**

**{**

**for(int a=0;a<matsiz;a++)//equation number iss loop se aye gaa**

**{**

**//////////////////////////////////////////////////////**

**//har equation ke liyay input variables taiyr krna hai**

**int i;**

**i=0;**

**for(int b=0;b<matsiz;b++)**

**{**

**if(a==b)**

**continue;**

**variables[i]=temp2[b];**

**i++;**

**}**

**//////////////////////////////////////////////////////**

**/////////////////////////////////////////////////////////////////**

**vals[a]=FUNCTIONS[a].Eval(variables);**

**//iss mai finally hum jo current iteration pe hain uss kay liyay sari**

**//equations ke output collect krliay**

**}**

**///////////////////////////NOW PRINTING////////////////////////**

**ststream.str("");**

**ststream.clear();**

**ststream<<"\t"<<loop<<"\t\t";**

**for(int a=0;a<matsiz;a++)**

**{**

**ststream<<vals[a];**

**temp2[a]=vals[a];**

**ststream<<"\t\t";**

**}**

**temp=ststream.str();**

**cout<<temp<<endl;**

**//////////////////////////////////////////////////////////////**

**}**

**}**

**void printdetails(void)**

**{**

**system("cls");**

**cout<<"parameters set by the user are :"<<endl;**

**cout<<"\nnumber of iterations given : ";**

**cout<<iteration;**

**cout<<"\ninitial approximation : ";**

**cout<<initialapprox;**

**cout<<"\nFIX value : ";**

**cout<<fixval;**

**cout<<"\nmatrix is : "<<endl;**

**for(int row=0;row<matsiz;row++)**

**{**

**for(int col=0;col<matsiz;col++)**

**cout<<"\t"<<matrix[row][col]<<" ";**

**cout<<endl;**

**}**

**cout<<"\n\nEquations are : "<<endl;**

**for(int a=0;a<matsiz;a++)**

**{**

**cout<<"x"<<a+1<<"="<<functions[a]<<endl;**

**}**

**cout<<"\n\nAre the details shown correct ? (y/n) = ";**

**char cho;**

**cin>>cho;**

**if(cho=='n')**

**setparameters();**

**system("pause");**

**}**

**};**

**class Trapezoidal**

**{**

**private:**

**int n,iteration;**

**string fx,trap,xi;**

**double \*vars,\*x,a,b,delX,finalans;**

**FunctionParser FX,TRAP,XI;**

**ostringstream ststream,vnam;**

**public:**

**Trapezoidal()**

**{**

**iteration=0;**

**a=b=0;**

**xi="a+(i\*x)";**

**XI.Parse(xi,"a,i,x");**

**}**

**void setparameters(void)**

**{**

**system("cls");**

**cout<<"\t\t\t\t\tSET PARAMETERS";**

**cout<<"\nenter value of N : ";//greater value of N better the accuracy**

**cin>>n;**

**cout<<"\nenter value of a : ";**

**cin>>a;**

**cout<<"\nenter value of b : ";**

**cin>>b;**

**while(true)**

**{**

**cout << "enter function F(x) = ";**

**cin>>fx;**

**if(cin.fail())**

**continue;**

**int res =FX.Parse(fx,"x");**

**if(res < 0)**

**break;**

**cout << string(res+7, ' ') << "^\n"<< FX.ErrorMsg() << "\n\n";**

**}**

**printdetails();**

**}**

**void evaluate(void)**

**{**

**x=new double[n+1];**

**vars=new double[n+2];**

**delX=(b-a)/n;**

**double temp[3];//0 a 1 i 2 delx**

**temp[0]=a;**

**temp[2]=delX;**

**//making array of Xs**

**for(int loop=0;loop<=n;loop++)**

**{**

**temp[1]=loop;**

**x[loop]=XI.Eval(temp);**

**}**

**string varnames;**

**ststream.str("");**

**ststream.clear();**

**vnam.str("");**

**vnam.clear();**

**ststream<<"x/2\*(x0";**

**vnam<<"x,x0";**

**for(int loop=1;loop<n;loop++)**

**{**

**ststream<<"+2\*x"<<loop;**

**vnam<<",x"<<loop;**

**}**

**ststream<<"+x"<<n<<")";**

**vnam<<",x"<<n;**

**trap=ststream.str();**

**varnames=vnam.str();**

**//cout<<"\n trapizoidal formula is "<<trap<<endl;**

**//cout<<"\n with variables : "<<varnames<<endl;**

**TRAP.Parse(trap,varnames);**

**vars[0]=delX;**

**int i=1;**

**for(int loop=0;loop<=n;loop++)**

**{**

**temp[0]=x[loop];**

**vars[i]=FX.Eval(temp);**

**i++;**

**}**

**finalans=TRAP.Eval(vars);**

**cout<<"\nCALCULATED VALUE IS : "<<finalans<<endl;**

**system("pause");**

**}**

**void printdetails(void)**

**{**

**system("cls");**

**cout<<"parameters set by the user are :"<<endl;**

**cout<<"\nvalue of N : ";**

**cout<<n;**

**cout<<"\nvalue of a : ";**

**cout<<a;**

**cout<<"\nvalue of b : ";**

**cout<<b;**

**cout<<"\n\nF(x) = ";**

**cout<<fx;**

**cout<<"\n\nAre the details shown correct ? (y/n) = ";**

**char cho;**

**cin>>cho;**

**if(cho=='n')**

**setparameters();**

**system("pause");**

**}**

**};**

**class NIMidpoint**

**{**

**private:**

**int n,iteration;**

**string fx,trap,xi;**

**double \*vars,\*x,a,b,delX,finalans;**

**FunctionParser FX;**

**ostringstream ststream,vnam;**

**public:**

**NIMidpoint()**

**{**

**iteration=0;**

**a=b=0;**

**}**

**void setparameters(void)**

**{**

**system("cls");**

**cout<<"\t\t\t\t\tSET PARAMETERS";**

**cout<<"\nenter value of N (if n is not given then enter -1): ";//greater the value of N better the accuracy**

**cin>>n;**

**cout<<"\nenter value of a : ";**

**cin>>a;**

**cout<<"\nenter value of b : ";**

**cin>>b;**

**while(true)**

**{**

**cout << "enter function F(x) = ";**

**cin>>fx;**

**if(cin.fail())**

**continue;**

**int res =FX.Parse(fx,"x");**

**if(res < 0)**

**break;**

**cout << string(res+7, ' ') << "^\n"<< FX.ErrorMsg() << "\n\n";**

**}**

**printdetails();**

**}**

**void evaluate(void)**

**{**

**x=new double[n+1];**

**vars=new double[n];**

**delX=((b)-(a))/n;**

**x[0]=a;**

**for(int loop=1;loop<=n;loop++)**

**{**

**x[loop]=x[loop-1]+delX;**

**// cout<<"\nx["<<loop<<"] = "<<x[loop];**

**}**

**for(int loop=0;loop<n;loop++)**

**{**

**vars[loop]=(x[loop]+x[loop+1])/2;**

**// cout<<"\nvars["<<loop<<"] = "<<vars[loop];**

**}**

**long double sum=0;**

**for(int loop=0;loop<n;loop++)**

**{**

**double \*temp=new double;**

**\*temp=vars[loop];**

**sum+=(FX.Eval(temp))\*(delX);**

**}**

**cout<<"\nfinal answer is : "<<sum;**

**}**

**void printdetails(void)**

**{**

**system("cls");**

**cout<<"parameters set by the user are :"<<endl;**

**cout<<"\nvalue of N : ";**

**cout<<n;**

**cout<<"\nvalue of a : ";**

**cout<<a;**

**cout<<"\nvalue of b : ";**

**cout<<b;**

**cout<<"\n\nF(x) = ";**

**cout<<fx;**

**cout<<"\n\nAre the details shown correct ? (y/n) = ";**

**char cho;**

**cin>>cho;**

**if(cho=='n')**

**setparameters();**

**system("pause");**

**}**

**};**

**class Simpson**

**{**

**private:**

**int n,iteration;**

**string fx,trap,xi;**

**double \*vars,\*x,a,b,delX,finalans;**

**FunctionParser FX,TRAP,XI;**

**ostringstream ststream,vnam;**

**public:**

**Simpson()**

**{**

**iteration=0;**

**a=b=0;**

**xi="a+(i\*x)";**

**XI.Parse(xi,"a,i,x");**

**}**

**bool setparameters(void)**

**{**

**system("cls");**

**cout<<"\t\t\t\t\tSET PARAMETERS";**

**cout<<"\nenter value of N : ";//greater value of N better the accuracy**

**cin>>n;**

**if((n%2)!=0)**

**return false;**

**cout<<"\nenter value of a : ";**

**cin>>a;**

**cout<<"\nenter value of b : ";**

**cin>>b;**

**while(true)**

**{**

**cout << "enter function F(x) = ";**

**cin>>fx;**

**if(cin.fail())**

**continue;**

**int res =FX.Parse(fx,"x");**

**if(res < 0)**

**break;**

**cout << string(res+7, ' ') << "^\n"<< FX.ErrorMsg() << "\n\n";**

**}**

**printdetails();**

**return true;**

**}**

**void evaluate(void)**

**{**

**x=new double[n+1];**

**vars=new double[n+2];**

**delX=(b-a)/n;**

**double temp[3];//0 a 1 i 2 delx**

**temp[0]=a;**

**temp[2]=delX;**

**//making array of Xs**

**for(int loop=0;loop<=n;loop++)**

**{**

**temp[1]=loop;**

**x[loop]=XI.Eval(temp);**

**}**

**string varnames;**

**ststream.str("");**

**ststream.clear();**

**vnam.str("");**

**vnam.clear();**

**ststream<<"x/3\*(x0";**

**vnam<<"x,x0";**

**for(int loop=1;loop<n;loop++)**

**{**

**if((loop%2)==0)**

**ststream<<"+2\*x"<<loop;**

**else**

**ststream<<"+4\*x"<<loop;**

**vnam<<",x"<<loop;**

**}**

**ststream<<"+x"<<n<<")";**

**vnam<<",x"<<n;**

**trap=ststream.str();**

**varnames=vnam.str();**

**// cout<<"\n simson formula is "<<trap<<endl;**

**// cout<<"\n with variables : "<<varnames<<endl;**

**TRAP.Parse(trap,varnames);**

**vars[0]=delX;**

**int i=1;**

**for(int loop=0;loop<=n;loop++)**

**{**

**temp[0]=x[loop];**

**vars[i]=FX.Eval(temp);**

**i++;**

**}**

**finalans=TRAP.Eval(vars);**

**cout<<"\nCALCULATED VALUE IS : "<<finalans<<endl;**

**system("pause");**

**}**

**void printdetails(void)**

**{**

**system("cls");**

**cout<<"parameters set by the user are :"<<endl;**

**cout<<"\nvalue of N : ";**

**cout<<n;**

**cout<<"\nvalue of a : ";**

**cout<<a;**

**cout<<"\nvalue of b : ";**

**cout<<b;**

**cout<<"\n\nF(x) = ";**

**cout<<fx;**

**cout<<"\n\nAre the details shown correct ? (y/n) = ";**

**char cho;**

**cin>>cho;**

**if(cho=='n')**

**setparameters();**

**system("pause");**

**}**

**};**

**class Boole**

**{**

**private:**

**int n,iteration;**

**string fx,trap,xi;**

**double \*vars,\*x,a,b,delX,finalans;**

**FunctionParser FX,TRAP,XI;**

**ostringstream ststream,vnam;**

**public:**

**Boole()**

**{**

**iteration=0;**

**a=b=0;**

**xi="a+(i\*x)";**

**XI.Parse(xi,"a,i,x");**

**}**

**bool setparameters(void)**

**{**

**system("cls");**

**cout<<"\t\t\t\t\tSET PARAMETERS";**

**cout<<"\nenter value of N : ";//greater value of N better the accuracy**

**cin>>n;**

**if((n%2)==0)**

**return false;**

**cout<<"\nenter value of a : ";**

**cin>>a;**

**cout<<"\nenter value of b : ";**

**cin>>b;**

**while(true)**

**{**

**cout << "enter function F(x) = ";**

**cin>>fx;**

**if(cin.fail())**

**continue;**

**int res =FX.Parse(fx,"x");**

**if(res < 0)**

**break;**

**cout << string(res+7, ' ') << "^\n"<< FX.ErrorMsg() << "\n\n";**

**}**

**printdetails();**

**return true;**

**}**

**void evaluate(void)**

**{**

**x=new double[n+1];**

**vars=new double[n+2];**

**delX=(b-a)/n;**

**double temp[3];//0 a 1 i 2 delx**

**temp[0]=a;**

**temp[2]=delX;**

**//making array of Xs**

**for(int loop=0;loop<=n;loop++)**

**{**

**temp[1]=loop;**

**x[loop]=XI.Eval(temp);**

**}**

**string varnames;**

**ststream.str("");**

**ststream.clear();**

**vnam.str("");**

**vnam.clear();**

**ststream<<"(3\*x)/8\*(x0";**

**vnam<<"x,x0";**

**for(int loop=1;loop<n;loop++)**

**{**

**ststream<<"+3\*x"<<loop;**

**vnam<<",x"<<loop;**

**}**

**ststream<<"+x"<<n<<")";**

**vnam<<",x"<<n;**

**trap=ststream.str();**

**varnames=vnam.str();**

**cout<<"\n boole's formula is "<<trap<<endl;**

**cout<<"\n with variables : "<<varnames<<endl;**

**TRAP.Parse(trap,varnames);**

**vars[0]=delX;**

**int i=1;**

**for(int loop=0;loop<=n;loop++)**

**{**

**temp[0]=x[loop];**

**vars[i]=FX.Eval(temp);**

**i++;**

**}**

**finalans=TRAP.Eval(vars);**

**cout<<"\nCALCULATED VALUE IS : "<<finalans<<endl;**

**system("pause");**

**}**

**void printdetails(void)**

**{**

**system("cls");**

**cout<<"parameters set by the user are :"<<endl;**

**cout<<"\nvalue of N : ";**

**cout<<n;**

**cout<<"\nvalue of a : ";**

**cout<<a;**

**cout<<"\nvalue of b : ";**

**cout<<b;**

**cout<<"\n\nF(x) = ";**

**cout<<fx;**

**cout<<"\n\nAre the details shown correct ? (y/n) = ";**

**char cho;**

**cin>>cho;**

**if(cho=='n')**

**setparameters();**

**system("pause");**

**}**

**};**

**int main()**

**{**

**while(1)**

**{**

**system("cls");**

**int cho;**

**char choice='y';**

**cout<<"\n\t\t\t\tPROJECT CONTENTS/MODULES"<<endl<<endl;**

**cout<<"\n1 for Solution of non linear equation : \n\tBisection,Fixed Point iteration ,Regula False , Newton and Secant Method."<<endl<<endl;**

**cout<<"\n2 for Numerical Integration:\n\tTrapezoidal , Midpoint , Simpson's and Boole's formula."<<endl<<endl;**

**cout<<"\n3 for Solution of ordinary differential equation : \n\tEuler's , Huen's , 4-RK Method."<<endl<<endl;**

**cout<<"\n4 for System of linear equation: \n\t(Iterative methods)Gauss-Siedel and Jacobi's methods."<<endl<<endl;**

**cout<<"\n5 to Exit the program"<<endl<<endl;**

**cout<<"\nENTER YOUR CHOICE : ";**

**cin>>cho;**

**system("cls");**

**switch (cho)**

**{**

**case(1)://Solution of non linear equation (complete) (NOT TESTED)**

**{**

**while(choice=='y')**

**{**

**system("cls");**

**cout<<"press the number to select its corresponding module \n1 bisection \n2 false position \n3 newton \n4 secant\n5 Fix point iteration\n6 Exit to main menu : "<<endl;**

**cout<<"\nselect the method : ";**

**cin>>cho;**

**switch(cho)**

**{**

**case 1:**

**{**

**bisection \*B;**

**B=new bisection;**

**B->setparameters();**

**B->evaluate();**

**cout<<"\ndo you want to use this module again ? (y/n) ";**

**cin>>choice;**

**delete B;**

**}**

**break;**

**case 2:**

**{**

**falseposition \*F;**

**F=new falseposition;**

**F->setparameters();**

**F->evaluate();**

**cout<<"\ndo you want to use this module again ? (y/n) ";**

**cin>>choice;**

**delete F;**

**}**

**break;**

**case 3:**

**{**

**newton\_method \*N;**

**N=new newton\_method;**

**N->setparameters();**

**N->printdetails();**

**cout<<"\n1 to follow Epsilon \n2 to input no of iteration \nenter your choice ";**

**int val;**

**cin>>val;**

**system("cls");**

**N->evaluate(val);**

**cout<<"\ndo you want to use this module again ? (y/n) ";**

**cin>>choice;**

**delete N;**

**}**

**break;**

**case 4:**

**{**

**secant\_method \*S;**

**S=new secant\_method;**

**S->setparameters();**

**cout<<"\n1 to follow Epsilon \n2 to input no of iteration \nenter your choice ";**

**int val;**

**cin>>val;**

**system("cls");**

**S->evaluate(val);**

**cout<<"\ndo you want to use this module again ? (y/n) ";**

**cin>>choice;**

**delete S;**

**}**

**break;**

**case 5://fix point**

**{**

**fix\_point \*F;**

**F=new fix\_point;**

**F->setparameters();**

**system("cls");**

**F->evaluate();**

**cout<<"\ndo you want to use this module again ? (y/n) ";**

**cin>>choice;**

**delete F;**

**}**

**break;**

**case 6:**

**{**

**choice='n';**

**}**

**break;**

**default:**

**{**

**cout<<"\nWRONG OPTION SELECTED!";**

**system("pause");**

**continue;**

**}**

**}// option 1 switch ends here**

**}**

**}//module 1 case ends here**

**break;**

**case(2)://Numerical Integration (complete) (NOT TESTED)**

**{**

**while(choice=='y')**

**{**

**system("cls");**

**cout<<"press the number to select its corresponding module \n1 Trapezoidal \n2 Midpoint \n3 Simpson's \n4 Boole's formuls\n5 Exit to main menu :"<<endl;**

**cout<<"\nselect the method : ";**

**cin>>cho;**

**switch(cho)**

**{**

**case (1)://Trapezoidal**

**{**

**Trapezoidal \*T;**

**T=new Trapezoidal;**

**T->setparameters();**

**T->evaluate();**

**cout<<"\ndo you want to use this module again ? (y/n) ";**

**cin>>choice;**

**delete T;**

**}**

**break;**

**case (2)://Midpoint**

**{**

**NIMidpoint \*N;**

**N=new NIMidpoint;**

**N->setparameters();**

**N->evaluate();**

**cout<<"\ndo you want to use this module again ? (y/n) ";**

**cin>>choice;**

**delete N;**

**}**

**break;**

**case (3)://Simpson's**

**{**

**Simpson \*S;**

**S=new Simpson;**

**if(S->setparameters())**

**S->evaluate();**

**else**

**cout<<"\nvalue of N is not even so we cannot proceed with this module";**

**cout<<"\ndo you want to use this module again ? (y/n) ";**

**cin>>choice;**

**delete S;**

**}**

**break;**

**case (4)://Boole's formuls**

**{**

**Boole \*B;**

**B=new Boole;**

**if(B->setparameters())**

**B->evaluate();**

**else**

**cout<<"\nvalue of N is not odd so we cannot proceed with this module";**

**cout<<"\ndo you want to use this module again ? (y/n) ";**

**cin>>choice;**

**delete B;**

**}**

**break;**

**case (5)://Exit to main menu**

**{**

**choice='n';**

**continue;**

**}**

**break;**

**}//switch ends**

**}//loop ends here**

**}**

**break;**

**case(3)://Solution of ordinary differential equation (complete) (NOT TESTED)**

**{**

**while(choice=='y')**

**{**

**system("cls");**

**cout<<"press the number to select its corresponding module \n1 Euler's method \n2 Modified Euler's method \n3 Mid Point formula \n4 4 RK method\n5 Huen’s method\n6 Exit to main menu :"<<endl;**

**cout<<"\nselect the method : ";**

**cin>>cho;**

**switch(cho)**

**{**

**case (1)://Euler's method**

**{**

**euler \*E;**

**E=new euler;**

**E->setparameters();**

**E->evaluate();**

**cout<<"\ndo you want to use this module again ? (y/n) ";**

**cin>>choice;**

**delete E;**

**}**

**break;**

**case (2):// Modified Euler's method**

**{**

**Meuler \*ME;**

**ME=new Meuler;**

**ME->setparameters();**

**ME->evaluate();**

**cout<<"\ndo you want to use this module again ? (y/n) ";**

**cin>>choice;**

**delete ME;**

**}**

**break;**

**case (3)://midpoint**

**{**

**Midpoint \*M;**

**M=new Midpoint;**

**M->setparameters();**

**M->evaluate();**

**cout<<"\ndo you want to use this module again ? (y/n) ";**

**cin>>choice;**

**delete M;**

**}**

**break;**

**case (4): //RK method**

**{**

**RK4 \*R;**

**R=new RK4;**

**R->setparameters();**

**R->evaluate();**

**cout<<"\ndo you want to use this module again ? (y/n) ";**

**cin>>choice;**

**delete R;**

**}**

**break;**

**case (5)://Huen’s**

**{**

**Huen \*H;**

**H=new Huen;**

**H->setparameters();**

**H->evaluate();**

**cout<<"\ndo you want to use this module again ? (y/n) ";**

**cin>>choice;**

**delete H;**

**}**

**break;**

**case (6):**

**{**

**choice='n';**

**continue;**

**}**

**}//switch ends**

**}//loop ends here**

**}**

**break;**

**case(4)://System of linear equation Iterative methods (complete) (NOT TESTED)**

**{**

**while(choice=='y')**

**{**

**system("cls");**

**cout<<"press the number to select its corresponding module \n1 Jacobi's method \n2 Gauss-Siedel method \n3 Exit to main menu :"<<endl;**

**cout<<"\nselect the method : ";**

**cin>>cho;**

**switch(cho)**

**{**

**case(1)://Jacobi’s method**

**{**

**jacob \*J;**

**J=new jacob;**

**J->setparameters();**

**J->evaluate();**

**cout<<"\ndo you want to use this module again ? (y/n) ";**

**cin>>choice;**

**delete J;**

**}**

**break;**

**case(2)://Gauss-Siedel method**

**{**

**gauss \*G;**

**G=new gauss;**

**G->setparameters();**

**G->evaluate();**

**cout<<"\ndo you want to use this module again ? (y/n) ";**

**cin>>choice;**

**delete G;**

**}**

**break;**

**case(3):**

**{**

**choice='n';**

**}**

**break;**

**default:**

**{**

**cout<<"\nWRONG OPTION SELECTED!"<<endl;**

**system("pause");**

**continue;**

**}**

**}//switch ends**

**}//loop ends here**

**}//module 4 case ends here**

**break;**

**case(5)://EXIT MAIN PROGRAM**

**{**

**exit(0);**

**}**

**default:**

**{**

**cout<<"\nWRONG OPTION SELECTED!";**

**system("pause");**

**continue;**

**}**

**}// main switch ends here**

**}**

**}**

**CODE ENDS HERE!!!**

Reference/Bibliography:

* <http://www.bioline.org.br/pdf?ja08066>
* <http://ceur-ws.org/Vol-2147/p07.pdf>
* <http://www.journalarjom.com/index.php/ARJOM/article/view/24549>
* Numerical computing course book [Richard L. Burden and J. Douglas Faires]
* Numerical methods [ John H.Mathews, Kurtis D.FINK]
* lectures