Lingayas University



ANM

( APPLIED NUMERICAL MATHEMATICS)

LAB FILE

SUBMITTED TO: SUBMITTED BY:

DR. Ajit singh Chetna Bhardwaj

14CS043

Sec-A

INDEX

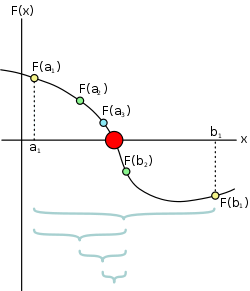
|  |  |  |  |
| --- | --- | --- | --- |
| S.NO | EXPERIMENTS | DATE | SIGNATURE |
| 1. | Program on Bisection method |  |  |
| 2. | Program on Newton rapson method |  |  |
| 3. | Program on Regula falsi method |  |  |
| 4. | Program on Gauss elimination method |  |  |
| 5. | Program on Gauss gordan method |  |  |
| 6. | Program on Gauss seidal method |  |  |
| 7. | Program on TRAPEZOIDAL method |  |  |
| 8. | Program on Simpson 1/3 method |  |  |
| 9. | Program on Euler method |  |  |
| 10. | Program on R.K method |  |  |

EXPERIMENT -1

“BISECTION METHOD”

It is based on the repeated application of intermediate value.

Let the function f(x) is continuous on interval [a,b],let f(a)<0 or f(x), f(b)<0. Again we find next approx, root by bisecting the interval and the process is continuous till desired accuracy is formed.



#include<iostream>

#include<conio.h>

#include<math.h>

#define f(x) (x\*x\*x-4\*x-9)

using namespace std;

void main()

{

int i;

float a, b, x, y, y1, y2, err;

cout << "\n Enter the Value of A & B :" << endl;

cin >> a >> b;

y1 = f(a);

y2 = f(b);

if (y1\*y2 > 0)

{

cout << "\n Wrong Interval ";

}

else

{

cout << "\n Enter the Error ";

cin >> err;

i = 1;

do

{

x = (a + b) / 2;

y = f(x);

if (y1\*y2 < 0)

{

b = x;

}

else

{

a = x;

}

cout << "\n the Iteration is : " << i << endl << i++;

cout << "\n required root is = " << x << endl;

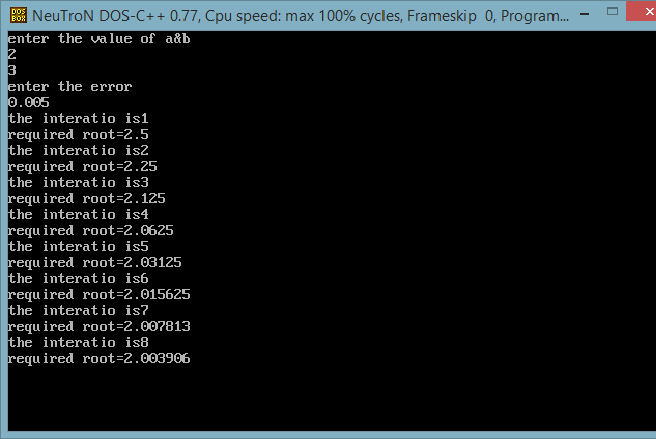
} while (fabs(b - a) > err);

system("pause");

}

}

OUTPUT



EXPERIMENT NO-2

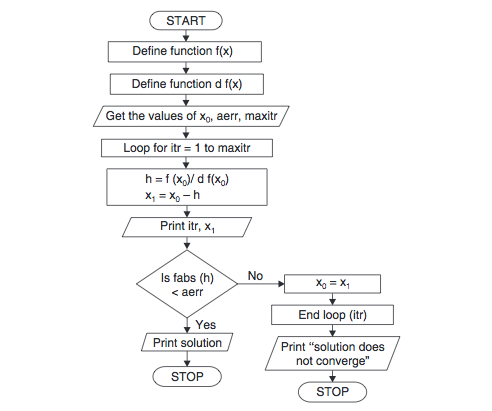
“TO FIND THE ROOT OF EQUATION COSX=Xex USING NEWTON RAPHSON METHOD”

Newton Raphson method, also called the Newton’s method, is the fastest and simplest approach of all methods to find the real root of a nonlinear function. It is an open bracket approach, requiring only one initial guess. This method is quite often used to improve the results obtained from other iterative approaches.

The convergence is fastest of all the root-finding methods we have discussed in Code with C. The algorithm and flowchart for Newton Raphson method given below is suitable for not only find the roots of a nonlinear equation, but the roots of algebraic and [transcendental equations](http://en.wikipedia.org/wiki/Transcendental_equation)as well.

The overall approach of Newton’s method is more useful in case of large values the first derivative of f(X) i.e f'(X). The iterative formula for Newton Raphson method is:

[highlight color=”yellow”]Xn+1 = Xn – f(Xn)/f'(Xn)[/highlight]



//To calcute cosx=xe^x by Using Newton rapson method

#include<iostream>

#include<conio.h>

#include<stdio.h>

#include<math.h>

using namespace std;

#define f(x) (cos(x)-x\*exp(x))

#define df(x) (-sin(x)-exp(x)-x\*exp(x))

int main()

{

system("clear");

int i;

float x0, x1, h, err;

cout << "Enter x0 and error :" << endl;

cin >> x0 >> err;

i = 1;

do

{

h = f(x0) / df(x0);

x1 = x0 - h;;

x0 = x1;

cout << "The iteration is" << i << endl;

i++;

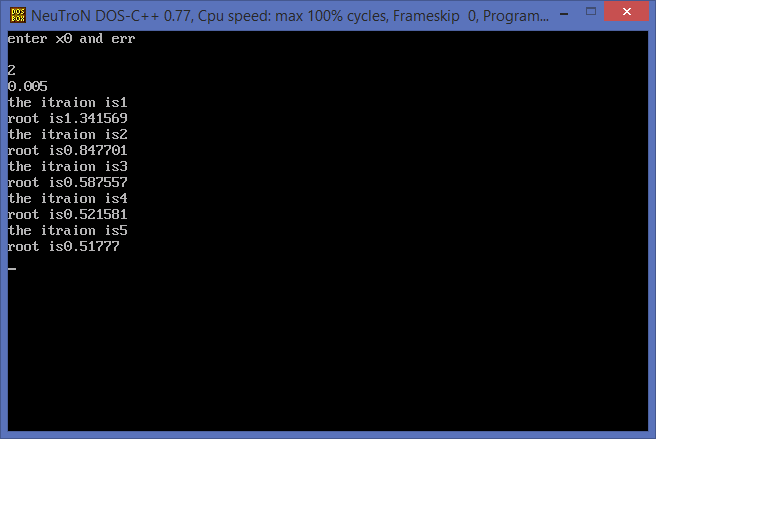
cout << "Root is " << x1 << endl;

} while (fabs(h)>err);

system("pause");

}

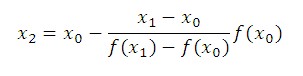
OUTPUT



EXPERIMENT NO-3

“ REGULA FALSI METHOD”

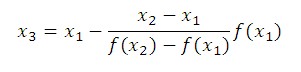
False Position method is the oldest method for finding the real roots of an equation f(x)=0. Also this method is closely resembled with Bisection method. In False Position method we choose two points x0 and x1, such that f(x0) and f(x1) are of opposite sign. So the abscissa of point where the chords cuts the x-axis (y=0) is given by,

[](http://myclassbook.files.wordpress.com/2013/04/11.jpg)

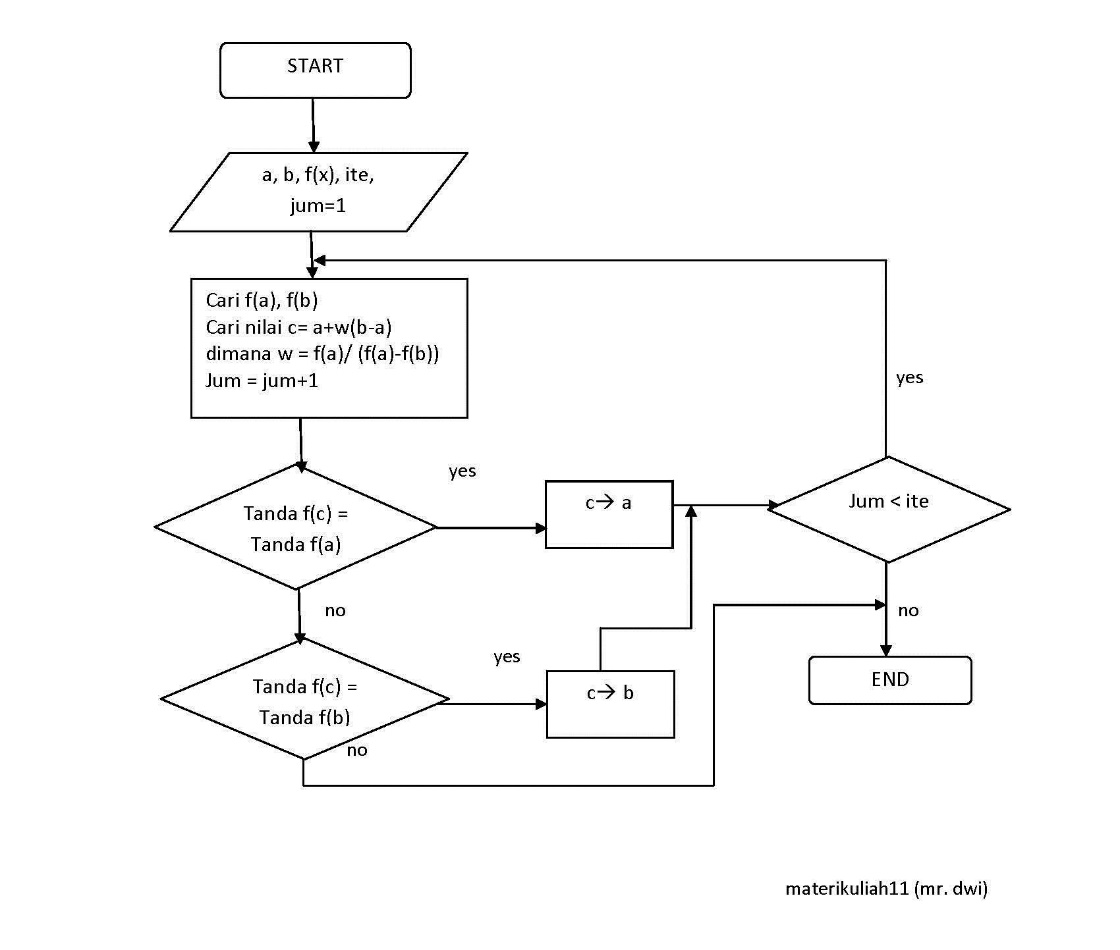
This is the first approximation to the root.

If now f(x0) and f(x2) are of opposite sign then the root lies between x0 and x2, so by replacing x1 by x2.

By using the following equation we obtain the second approximation:

[](http://myclassbook.files.wordpress.com/2013/04/2.jpg)

Similarly, we can obtain next approximations using same equations as above



SOURCE CODE

#include<iostream>

#include<conio.h>

#include<stdio.h>

#include<math.h>

using namespace std;

#define f(x) (x\*x\*x-2\*x-5)

int main()

{

int i;

float x0,x1,x,y,y1,y2,err;

cout<<"Enter x0 and x1"<<endl;

cin>>x0>>x1;

y1=f(x0);

y2=f(x1);

if(y1\*y2>0)

cout<<"Wrong interval:";

else

{

cout<<"Enter the error";

cin>>err;

i=1;

do

{

x=x0-((x1-x0)/(f(x1)-f(x0))\*f(x0));

y=f(x);

if(y1\*y<0)

{x0=x;}

else

{x1=x;}

cout<<"The iteration is"<<i<<endl;

i++;

cout<<"Root is "<<x<<endl;

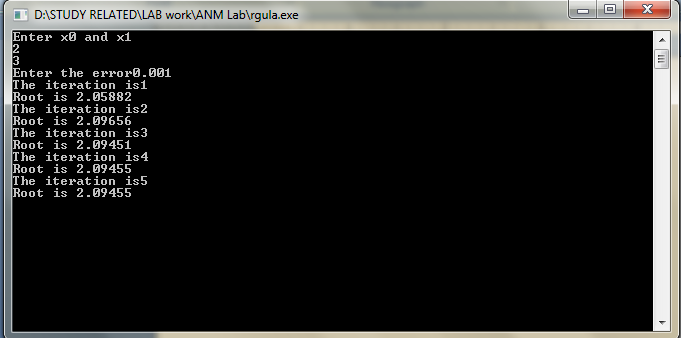
} while(fabs(x1-x0)>err);

}

getch();

}

OUTPUT



EXPERIMENT NO-4

“ GAUSS ELIMINATION METHOD”

In Gauss-Elimination method, these equations are solved by eliminating the unknowns successively. The C program for Gauss elimination method reduces the system to an upper triangular matrix from which the unknowns are derived by the use of backward substitution method.

Pivoting, partial or complete, can be done in Gauss Elimination method. So, this method is somewhat superior to the Gauss Jordan method. This approach, combined with the back substitution, is quite general. It is popularly used and can be well adopted to write a program for Gauss Elimination Method in C.

For this, let us first consider the following three equations:

a1x + b1y + c1z = d1

a2x + b2y + c2z = d2

a3x + b3y + c3z = d3

Assuming a1 ≠ 0, x is eliminated from the second equation by subtracting (a2/ a1) times the first equation from the second equation. In the same way, the C code presented here eliminates x from third equation by subtracting (a3/a1) times the first equation from the third equation.

Then we get the new equations as:

a1x + b1y + c1z = d1

b’2y + c’2z = d’2

c’’3z = d’’3

The elimination procedure is continued until only one unknown remains in the last equation. After its value is determined, the procedure is stopped. Now, Gauss Elimination in C uses back substitution to get the values of x, y and z as:

z= d’’3 / c’’3

y=(d’2 – c’2z) / b’2

x=( d1- c1z- b1y)/ a1

//gausslimitation method//

#include<iostream.h>

#include<conio.h>

#include<math.h>

#include<iomanip.h>

#define n 3

void main()

{

clrscr();

float a[n][n+1],t,s,x[n];

int i,j,k;

cout<<"Enter the value of matrix";

for(i=0;i<n;i++)

for(j=0;j<n+1;j++)

cin>>a[i][j];

for(j=0;j<n;j++)

for(i=j+1;i<n;i++)

{

t=a[i][j]/a[j][j];

for(k=0;k<n+1;k++)

{

a[i][k]-=a[j][k]\*t;

}

}

cout<<"The upper triangular matrix"<<endl;

for(i=0;i<n;i++)

{

for(j=0;j<n+1;j++)

{

cout<<a[i][j]<<"\t";

}

cout<<endl;

}

for(i=n-1;i>=0;i--)

{

s=0;

for(j=i+1;j<n;j++)

s+=a[i][j]\*x[j];

x[i]=(a[i][n]-s)/a[i][i];

}

cout<<"The solution"<<endl;

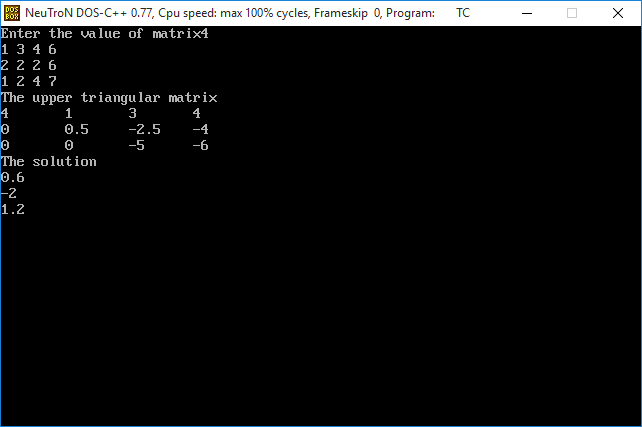
for(i=0;i<n;i++)

cout<<x[i]<<endl;

getch();

}

OUTPUT



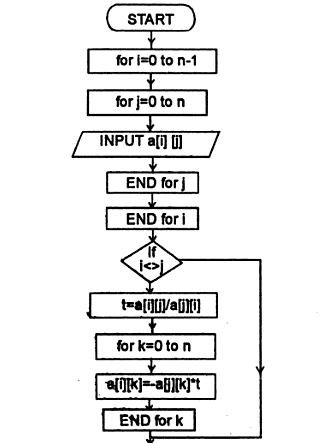
EXPERIMENT NO-5

“GAUSS JORDAN METHOD”

The program of **Gauss-Jordan Method in C** presented here diagonalizes the given matrix by simple row operations. The additional calculations can be a bit tedious, but this method, overall, can be effectively used for small systems of linear simultaneous equations.

In the Gauss-Jordan C program, the given matrix is diagonalized using the following step-wise procedure.

1. The element in the first column and the first row is reduced 1, and then the remaining elements in the first column are made 0 (zero).
2. The element in the second column and the second row is made 1, and then the other elements in the second column are reduced to 0 (zero).
3. Similarly, steps 1 and 2 are repeated for the next 3rd, 4th and following columns and rows.
4. The overall diagonalization procedure is done in a sequential manner, performing only row operations.



#include<iostream>

#include<conio.h>

#include<math.h>

using namespace std;

int main()

{

int i,j,k,n;

float a[10][10],t,x[10];

cout<<"Enter n: "<<endl;

cin>>n;

cout<<"Enter the elements of matrix in rowwise"<<endl;

for(i=1;i<n;i++)

for(j=1;j<n+1;j++)

cin>>a[i][j];

cout<<"Entered elements are"<<endl;

for(i=1;i<n;i++)

{for(j=1;j<n+1;j++)

cout<<a[i][j]<<"\t";

cout<<endl;

}

/\* Diagonal matrix \*/

for(j=1;j<n;j++)

for(i=1;i<n;i++)

{

if(i!=j)

t=a[i][j]/a[j][j];

for(k=1;k<n+1;k++)

a[i][k]-=(a[j][k]\*t);

}

cout<<"Diagonal Matrix is"<<endl;

for(i=1;i<n;i++)

{for(j=1;j<n+1;j++)

cout<<a[i][j]<<"\t";

cout<<endl;}

for(i=1;i<n;i++)

x[i]=(a[i][n])/(a[i][i]);

/\*Printing the results\*/

cout<<"The solution is"<<endl;

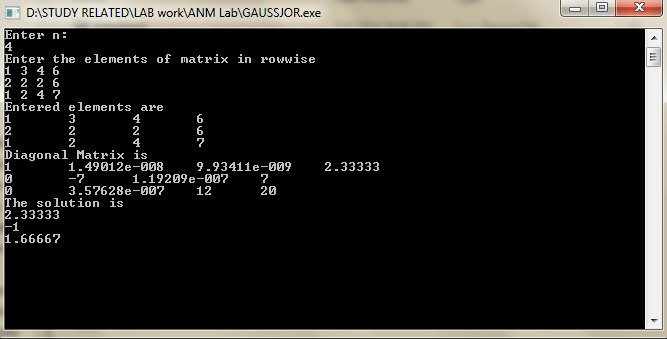
for(i=1;i<n;i++)

cout<<x[i]<<endl;

getch();

}

OUTPUT



EXPERIMENT NO-6

“GAUSS SEIDAL METHOD”

This C program for Gauss-Seidel method has been designed for the solution of linear simultaneous algebraic equations based on the principle of iteration.

In this program, a certain approximate value of solution is assumed and further calculations are done based on the result of assumed approximate solution.The program for Gauss-Seidel method in C works by following the steps listed below:

When the program is executed, first of all it asks for the value of elements of the augmented matrix row wise.

Then, the program asks for allowed error and maximum number of iteration to which the calculations are to be done. The number of iterations required depends upon the degree of accuracy.

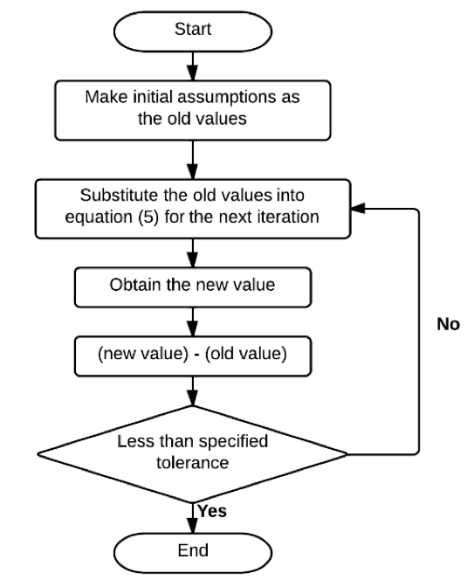
The program assumes initial or approximate solution as y=0 and z=0 and new value of x which is used to calculate new values of y and z using the following expressions:

x= 1/a1 ( d1-b1y-c1z)

y=1/b2 ( d2-a2x-c2z)

z=1/c3 ( d3-a3x-b3y)

The iteration process is continued until a desired degree of accuracy is not met.



#include<iostream>

#include<conio.h>

#include<math.h>

using namespace std;

int main()

{

float a[6][6],b[6][6],x[6],c;

int i,j,m,n,l;

cout<<"enter the max order of matrix";

cin>>n;

cout<<"enter the number of iteration";

cin>>l;

cout<<"enter the elements of matrix";

for(i=0;i<n;i++)

{

x[i]=0;

for(j=0;j<n;j++)

{

cin>>a[i][j];

}}

cout<<"enter the right hand side constant"<<endl;

for(i=0;i<n;i++)

{

cin>>b[i][j];

}

m=1;

z:

for(i=0;i<n;i++)

{

c=b[i][j];

for(j=0;j<n;j++)

{

if(i!=j)

{

c=c-a[i][j]\*x[j];

}

x[i]=c/a[i][i];

}}

m=m+1;

if(m<=1)

{

goto z;

}

else

cout<<"the solution is";

for(i=0;i<n;i++)

{

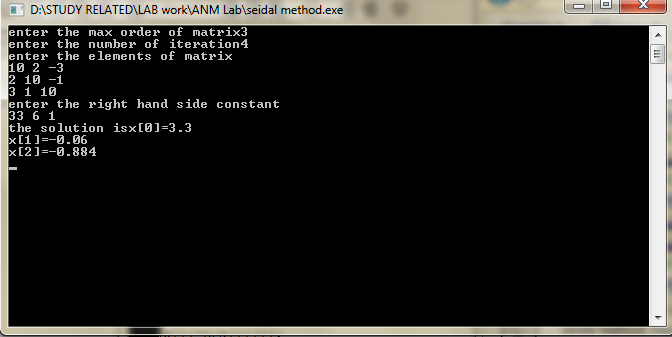
cout<<"x["<<i<<"]="<<x[i]<<endl;

}

getch();

}

OUTPUT

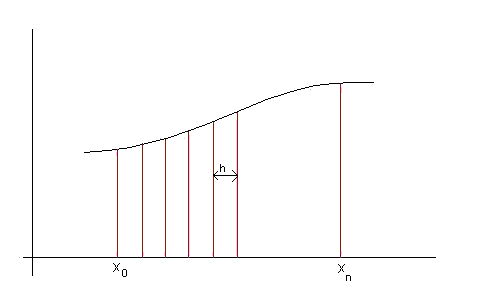


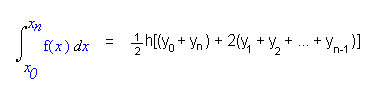
EXPERIMENT NO-7

“TRAPEZOIDAL METHOD”

Trapezoidal method is based on the principle that the area under the curve which is to be calculated is divided into number of small segments. The bounding curve in the segment is considered to be a straight line as a result the small enclosed area becomes a trapezium.

The area of each small trapezium is calculated and summed up i.e. integrated. This idea is the working mechanism in trapezoidal method algorithm and flowchart, even it sources code.

Let us consider a function f(x) representing a curve as shown in above figure. You are to find out the area under the curve from point ‘a’ to ‘b’. In order to do so, divide the distance between ab into a number vertical strips of width ‘h’ so that each strip can be considered as trapezium. 



#include<iostream>

#include<conio.h>

#include<iomanip.h>

using namespace std;

float y (float x)

{

return 1/(1+x\*x);

}

main()

{

float x0,xn,h,s;

int i,n;

cout<<"Enter the x0,xn,no of subintervals"<<endl;

cin>>x0>>xn>>n;

h=(xn-x0)/n;

s=y(x0)+y(xn);

for(i=1;i<=n;i++)

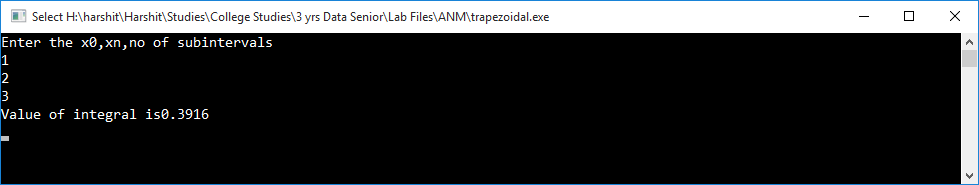
s+=2\*y(x0+i\*h);

cout<<"Value of integral is"<<setw(6)<<setprecision(4)<<(h/2)\*s<<endl;

getch();

}

Output



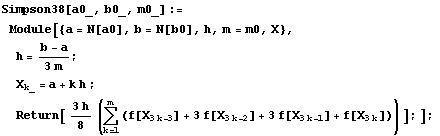
EXPERIMENT NO-8

“SIMPSON 1/3 METHOD”

Simpson1/3 rule is easy and accurate for calculation of numerical integration of any function which is defined in program.

In the source code below, a function *f(x) = 1/(1+x)* has been defined. The calculation using **Simpson** 1/3 rule in C is based on the fact that the small portion between any two points is a parabola. The program follows the following steps for calculation of the integral.

* As the program gets executed, first of all it asks for the value of lower boundary value of x i.e. x0, upper boundary value of x i.e. xnand width of the strip, h.
* Then the program finds the value of number of strip as n=( xn– x0)/h and checks whether it is even or odd. If the value of ‘n’ is odd, the program refines the value of ‘h’ so that the value of ‘n’ comes to be even.
* After that, this C program calculates value of f(x) i.e ‘y’ at different intermediate values of ‘x’ and displays values of all intermediate values of ‘y’.
* After the calculation of values of ‘c’, the program uses the following formula to calculate the value of integral in loop.  
  Integral =  \*((y0+ yn) +4(y1+ y3+ ……….+ yn-1) + 2(y2+ y4+……….+ yn-2))



#include<iostream>

#include<conio.h>

#include<iomanip>

using namespace std;

float y(float x)

{

return(1 / (1 + (x\*x)));

}

void main()

{

float x0, xn, h, s;

int i, n;

cout << "\n Enter X0 Xn , No of Subintervals \n";

cin >> x0 >> xn >> n;

h = (xn - x0) / n;

s = y(x0)+y(xn)+4 \* y(x0 + h);

for (i = 3; i <= n - 1; i += 2)

{

s += 4 \* y(x0 + (i\*h)) + 2 \* y(x0 + (i - 1)\*h);

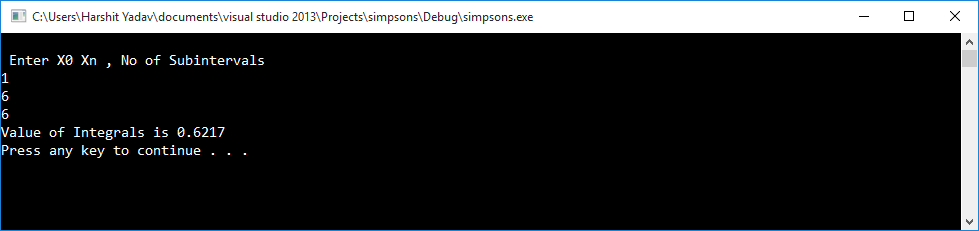
}

cout << "Value of Integrals is " << setw(6) << setprecision(4) << (h / 3)\*s << "\n";

system("pause");

}

Output



EXPERIMENT NO-9

“Euler method”

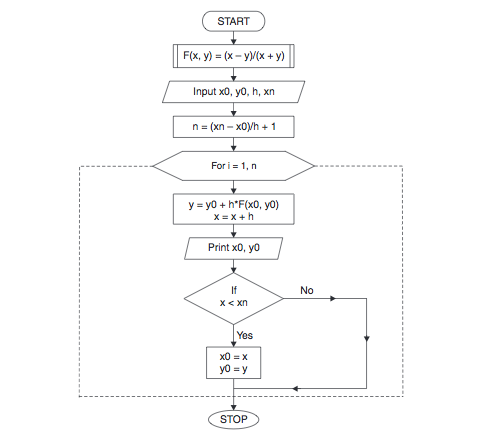
Euler’s method considers an ordinary differential equation, and the initial values of x and y are known.

Mathematically, here, the curve of solution is approximated by a sequence of short lines i.e. by the tangent line in each interval. ([Derivation](http://www.codewithc.com/eulers-method-matlab-program/)) Using these information, the value of  the value of ‘yn’ corresponding to the value of  ‘xn‘ is to determined by dividing the  length (xn– x) into n strips.

Therefore, strip width= (xn – x)/n and xn=x0+ nh.

Again, if m be the slope of the curve at point,  y1= y0 + m(x0 , yo)h.

Similarly, values of all the intermediate y can be found out.



#include<iostream>

#include<conio.h>

using namespace std;

float df(float x,float y)

{

return (x+y);

}

int main()

{

float x0,y0,h,x,x1,y1;

cout<<"Enter the values of x0,y0,h,x"<<endl;

cin>>x0>>y0>>h>>x;

x1=x0;

y1=y0;

while(1)

{

if(x1>x)

return 0;

y1+=h\*df(x1,y1);

x1+=h;

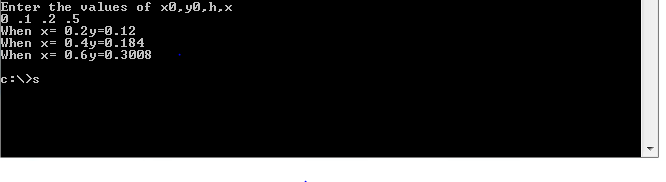
cout<<"When x= "<<x1<<"y="<<y1<<endl;

}

getch();

}

Output

****

EXPERIMENT NO-10

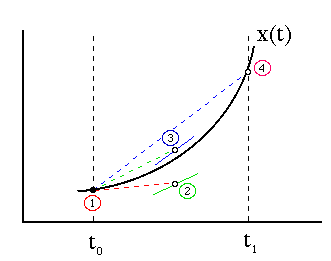
“R.K method”

**Runge Kutta method** is designed to find out the numerical solution of a first order differential equation. It is a kind of initial value problem in which initial conditions are known, i.e the values of x0 and y0are known, and the values of y at different values x is to be found out.

The source code below to solve ordinary differential equations of  first order by RK4 method first asks for the values of initial condition i.e. user needs to input x0 and y0. Then, the user has to define increment ‘h’ and the final value of x at which y is to be determined.

In this program for**Runge Kutta method in C**, a function f(x,y) is defined to calculate slope whenever it is called.

**f(x,y) = (x-y)/(x+y)**



#include<iostream>

#include<conio.h>

using namespace std;

float f(float x,float y)

{

return (x+y\*y);

}

int main()

{

float x0,y0,h,xn,x,y,k1,k2,k3,k4,k;

cout<<"Enter the values of x0,y0,h,xn"<<endl;

cin>>x0>>y0>>h>>xn;

x=x0;

y=y0;

while(1)

{

if(x==xn)

break;

k1=h\*f(x,y);

k2=h\*f(x+h/2,y+k1/2);

k3=h\*f(x+h/2,y+k2/2);

k4=h\*f(x+h,y+k3);

k=(k1+(k2+k3)\*2+k4)/6;

x+=h;

y+=k;

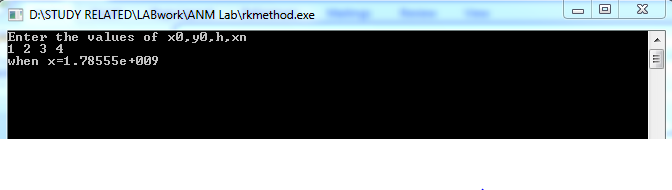
cout<<"when x="<<y<<endl;

}

getch();

}

Output

****