

**Applied Mathematics Lab**

**Name:** Piyush singh

**Enrollment No.:** 43115003118

**Serial No.:** 54

**Batch:** IT-2(M)

**Experiment 1**

// Algebra of Matrices

#include <iostream>

using namespace std;

int a[3][3], b[3][3], result[3][3], i, j, k;

void display\_result()

{

for(i=0; i<3; i++)

{

for(j=0; j<3; j++)

cout << result[i][j] << " ";

cout << endl;

}

cout << "\n\n";

}

void get\_matrix\_a()

{

cout << "Enter the values of first matrix \n";

for(i=0; i<3; i++)

for(j=0; j<3; j++)

cin >> a[i][j];

cout << endl;

}

void get\_matrix\_b()

{

cout << "Enter the values of second matrix \n";

for(i=0; i<3; i++)

for(j=0; j<3; j++)

cin >> b[i][j];

cout << endl;

}

void addition()

{

cout << "Addition of matrices is ==> \n\n";

for(i=0; i<3; i++)

for(j=0; j<3; j++)

result[i][j] = a[i][j] + b[i][j];

display\_result();

}

void multiplication()

{

cout << "Multiplication matrices is ==> \n\n";

for(i=0; i<3; i++)

for(j=0; j<3; j++)

{

result[i][j] = 0;

for(k=0; k<3; k++)

result[i][j] += a[i][k]\*b[k][j];

}

display\_result();

}

void transpose()

{

cout << "Transpose of first matrix is ==> \n\n";

for(i=0; i<3; i++)

for(j=0; j<3; j++)

result[i][j] = a[j][i];

display\_result();

}

int main() {

get\_matrix\_a();

get\_matrix\_b();

addition();

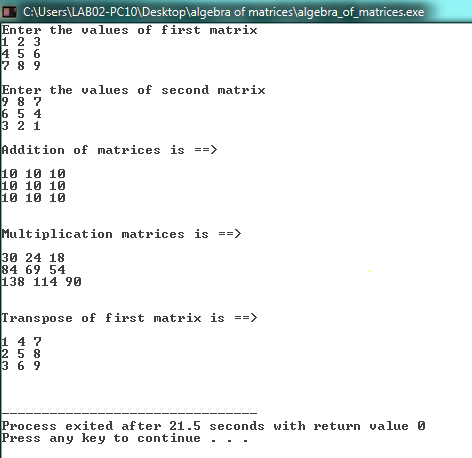
multiplication();

transpose();

return 0;

}

**Output**

****

**Experiment 2**

// Bisection Method

#include <iostream>

#define ACCURACY 0.01

using namespace std;

double calulate(double x)

{

return x\*x\*x + 3\*x\*x + 3\*x + 1;

}

int main()

{

double a =-20, b = 30;

if (calulate(a) \* calulate(b) >= 0)

{

cout << "You have not assumed right a and b\n";

exit(1);

}

double c = a;

while ((b-a) >= ACCURACY)

{

c = (a+b)/2;

if (calulate(c) == 0.0)

break;

else if (calulate(c) \* calulate(a) < 0)

b = c;

else

a = c;

}

cout << "The value of root is : " << c;

return 0;

}

**Output**

**Experiment 3**

// Newton Raphson Method.

#include<bits/stdc++.h>

#define EPSILON 0.001

using namespace std;

// The function is x^3 + 3\*x^2 + 3\*x + 1

double func(double x)

{

return x\*x\*x + 3\*x\*x + 3\*x + 1;

}

// Derivative of the above function which is 3\*x^2 + 6\*x + 3

double derivFunc(double x)

{

return 3\*x\*x + 6\*x + 3;

}

// Function to find the root

void newtonRaphson(double x)

{

double h = func(x) / derivFunc(x);

while (abs(h) >= EPSILON)

{

h = func(x)/derivFunc(x);

x = x - h; // x(i+1) = x(i) - f(x) / f'(x)

}

cout << "The value of the root is : " << x;

}

int main()

{

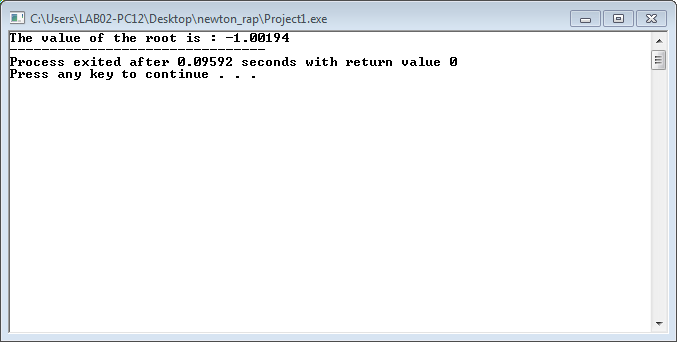
double x0 = -50; // Initial values assumed

newtonRaphson(x0);

return 0;

}

**Output**

****

**Experiment 4 (a)**

// Aim = WAP to solve the given system of linear equations using Gauss Jacobi's iterative method

#include <bits/stdc++.h>

using namespace std;

/\*

Given system of linear equation

a1x + b1y + c1z = d1

a2x + b2y + c2z = d2

a3x + b3y + c3z = d3

Simplified Equations:

x = (d1 - b1y - c1z)/a1

y = (d2 - a2x - c2z)/b2

z = (d3 - a3x - b3y)/c3

\*/

float f1(float y0, float z0, int a1, int b1, int c1, int d1) //Equation 1

{

return (d1 - b1\*y0 - c1\*z0)/a1;

}

float f2(float x0, float z0, int a2, int b2, int c2, int d2) //Equation 2

{

return (d2 - a2\*x0 - c2\*z0)/b2;

}

float f3(float x0, float y0, int a3, int b3, int c3, int d3) //Equation 3

{

return (d3 - a3\*x0 - b3\*y0)/c3;

}

int main()

{

int a1, b1, c1, d1;

int a2, b2, c2, d2;

int a3, b3, c3, d3;

float x0=0, y0=0, z0=0, x1, y1, z1, e1, e2, e3;

const float error = 0.001;

int step=1;

cout << "\tGauss Jacobi's Iteration method \n\n";

cout << "Enter the co efficients of three equations ax + by + cz = d\n";

cin >> a1 >> b1 >> c1 >> d1;

cin >> a2 >> b2 >> c2 >> d2;

cin >> a3 >> b3 >> c3 >> d3;

cout<< setprecision(3)<< fixed <<"\nIteration\tx\t\ty\t\tz\n"<< endl;

do

{

x1 = f1(y0,z0, a1, b1, c1, d1);

y1 = f2(x0,z0, a2, b2, c2, d2);

z1 = f3(x0,y0, a3, b3, c3, d3);

cout <<setw(5) << step

<<setw(14) << x1

<< setw(15) << y1

<< setw(17) << z1 << endl;

/\* Error in x, y and z \*/

e1 = fabs(x0-x1);

e2 = fabs(y0-y1);

e3 = fabs(z0-z1);

step++;

/\* Set value for next iteration \*/

x0 = x1;

y0 = y1;

z0 = z1;

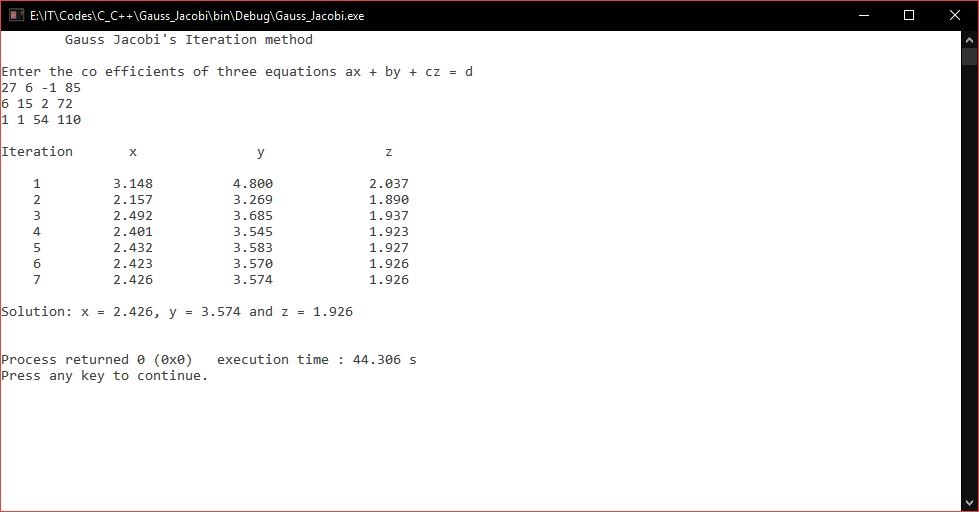
}while(e1>error && e2>error && e3>error);

cout<< endl<<"Solution: x = "<< x1<<", y = "<< y1<<" and z = "<< z1 << "\n\n";

return 0;

}

**Output**



**Experiment 4 (b)**

// Aim = WAP to solve the given system of linear equations using Gauss-Seidel iterative method

#include <bits/stdc++.h>

using namespace std;

/\*

Given system of linear equation

a1x + b1y + c1z = d1

a2x + b2y + c2z = d2

a3x + b3y + c3z = d3

Simplified Equations:

x = (d1 - b1y - c1z)/a1

y = (d2 - a2x - c2z)/b2

z = (d3 - a3x - b3y)/c3

\*/

float f1(float y0, float z0, int a1, int b1, int c1, int d1) //Equation 1

{

return (d1 - b1\*y0 - c1\*z0)/a1;

}

float f2(float x0, float z0, int a2, int b2, int c2, int d2) //Equation 2

{

return (d2 - a2\*x0 - c2\*z0)/b2;

}

float f3(float x0, float y0, int a3, int b3, int c3, int d3) //Equation 3

{

return (d3 - a3\*x0 - b3\*y0)/c3;

}

int main()

{

int a1, b1, c1, d1;

int a2, b2, c2, d2;

int a3, b3, c3, d3;

float x0=0, y0=0, z0=0, x1, y1, z1, e1, e2, e3;

const float error = 0.001;

int step=1;

cout << "\tGauss-Seidel Iteration method \n\n";

cout << "Enter the co efficients of three equations ax + by + cz = d\n";

cin >> a1 >> b1 >> c1 >> d1;

cin >> a2 >> b2 >> c2 >> d2;

cin >> a3 >> b3 >> c3 >> d3;

cout<< setprecision(3)<< fixed <<"\nIteration\tx\t\ty\t\tz\n"<< endl;

do

{

x1 = f1(y0,z0, a1, b1, c1, d1);

y1 = f2(x1,z0, a2, b2, c2, d2);

z1 = f3(x1,y1, a3, b3, c3, d3);

cout <<setw(5) << step

<<setw(14) << x1

<< setw(15) << y1

<< setw(17) << z1 << endl;

/\* Error in x, y and z \*/

e1 = fabs(x0-x1);

e2 = fabs(y0-y1);

e3 = fabs(z0-z1);

step++;

/\* Set value for next iteration \*/

x0 = x1;

y0 = y1;

z0 = z1;

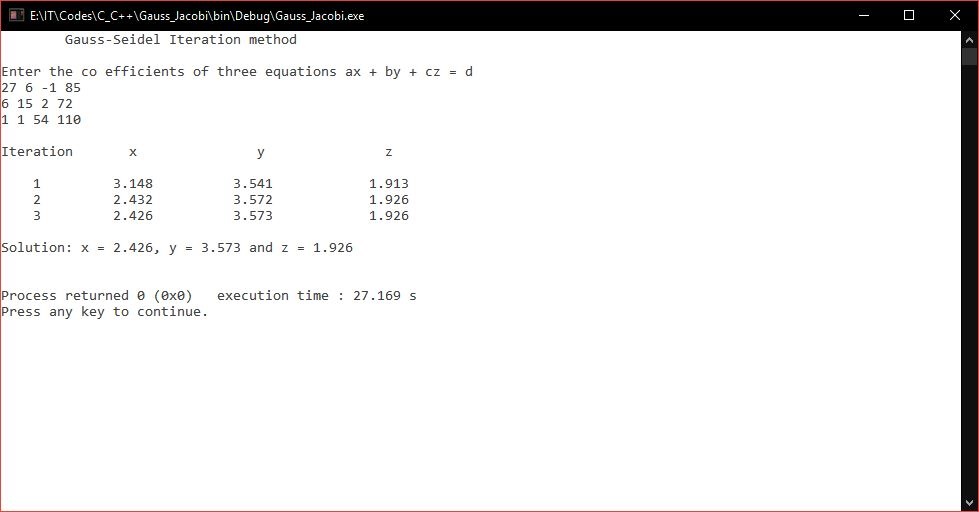
}while(e1>error && e2>error && e3>error);

cout<< endl<<"Solution: x = "<< x1<<", y = "<< y1<<" and z = "<< z1 << "\n\n";

return 0;

}

**Output**



**Experiment 5**

// Gauss Jordan Method

#include<iostream>

#include<math.h>

using namespace std;

int main()

{

cout << "\n\nGauss Jordan Method\n\n";

int i,j,k,n;

cout<<"\nEnter the no. of equations: ";

cin>>n;

/\* if no of equations are n then size of augmented matrix will be n\*n+1. So here we are declaring 2d array 'mat' of size n+n+1 \*/

float mat[n][n+1];

/\* for n equations there will be n unknowns which will be stored in array 'res' \*/

float res[n];

cout<<"\nEnter the elements of the augmented matrix: ";

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

{

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

{

cin>>mat[i][j];

}

}

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

{

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

{

if(abs(int(mat[i][i])) < abs(int(mat[j][i])))

{

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

{

/\* swapping mat[i][k] and mat[j][k] \*/

mat[i][k]=mat[i][k]+mat[j][k];

mat[j][k]=mat[i][k]-mat[j][k];

mat[i][k]=mat[i][k]-mat[j][k];

}

}

}

}

/\* performing Gaussian elimination \*/

for(i=0;i<n-1;i++)

{

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

{

float f=mat[j][i]/mat[i][i];

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

{

mat[j][k]=mat[j][k]-f\*mat[i][k];

}

}

}

/\* Backward substitution for discovering values of unknowns \*/

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

{

res[i]=mat[i][n];

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

{

if(i!=j)

{

res[i]=res[i]-mat[i][j]\*res[j];

}

}

res[i]=res[i]/mat[i][i];

}

cout<<"\nThe values of unknowns for the above equations=>\n";

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

{

cout<<res[i]<<"\n";

}

return 0;

}

**Output**

****

**Experiment 6**

// To integrate numerically using Trapezoidal Rule

#include<iostream>

#include<math.h>

/\* Define function here \*/

#define f(x) 1/(1+pow(x,2))

using namespace std;

int main()

{

cout << "\n\n To integrate numerically using Trapezoidal Rule\n\n";

float lower, upper, integration=0.0, stepSize, k;

int i, subInterval;

/\* Input \*/

cout<<" Enter lower limit of integration: ";

cin>>lower;

cout<<" Enter upper limit of integration: ";

cin>>upper;

cout<<" Enter number of sub intervals: ";

cin>>subInterval;

/\* Calculation \*/

/\* Finding step size \*/

stepSize = (upper - lower)/subInterval;

/\* Finding Integration Value \*/

integration = f(lower) + f(upper);

for(i=1; i<= subInterval-1; i++)

{

k = lower + i\*stepSize;

integration = integration + 2 \* (f(k));

}

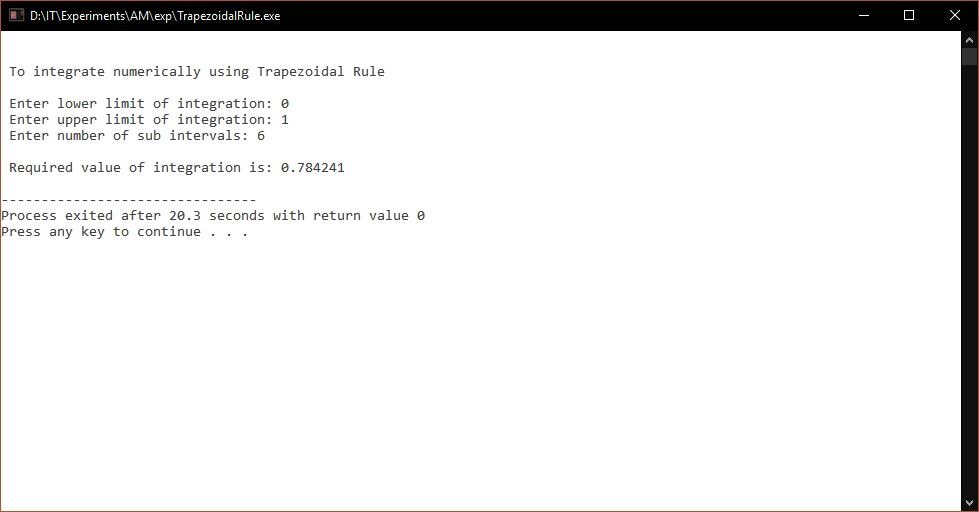
integration = integration \* stepSize/2;

cout<< endl<<" Required value of integration is: "<< integration << endl;

return 0;

}

**Output**

****

**Experiment 7**

// To integrate numerically using Simpsons Rule

#include<iostream>

#include<math.h>

/\* Define function here \*/

#define f(x) 1/(1+pow(x,2))

using namespace std;

int main()

{

cout << "\n\n To integrate numerically using Simpsons Rule\n\n";

float lower, upper, integration=0.0, stepSize, k;

int i, subInterval;

/\* Input \*/

cout<<" Enter lower limit of integration: ";

cin>>lower;

cout<<" Enter upper limit of integration: ";

cin>>upper;

cout<<" Enter number of sub intervals: ";

cin>>subInterval;

/\* Calculation \*/

/\* Finding step size \*/

stepSize = (upper - lower)/subInterval;

/\* Finding Integration Value \*/

integration = f(lower) + f(upper);

// Simpsons 1/3 Rule

for(i=1; i<= subInterval-1; i++)

{

k = lower + i\*stepSize;

if(i%2==0)

{

integration = integration + 2 \* (f(k));

}

else

{

integration = integration + 4 \* (f(k));

}

}

integration = integration \* stepSize/3;

cout<< endl <<" Required value of integration by Simpsons 1/3 Rule is: "<< integration << endl;

// Simpsons 3/8 Rule

for(i=1; i<= subInterval-1; i++)

{

k = lower + i\*stepSize;

if(i%3==0)

{

integration = integration + 2 \* (f(k));

}

else

{

integration = integration + 3 \* (f(k));

}

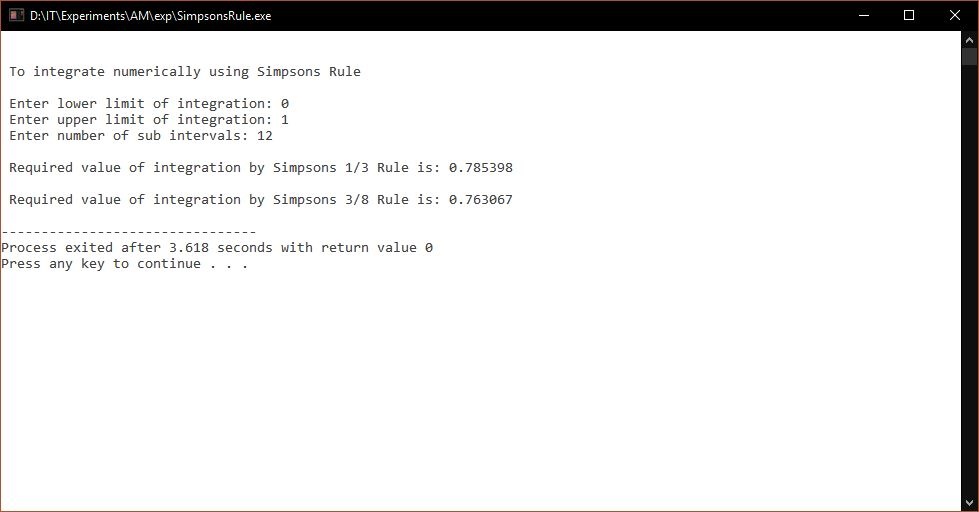
}

integration = integration \* stepSize\*3.0/8.0;

cout<< endl <<" Required value of integration by Simpsons 3/8 Rule is: "<< integration << endl;

return 0;

}

**Output**

**Experiment 8**

// To find largest Eigen Value of a matrix using Power Method

#include<iostream>

#include<iomanip>

#include<stdio.h>

#include<math.h>

#define SIZE 10

using namespace std;

int main()

{

cout << " To find largest Eigen Value of a matrix using Power Method\n\n";

float a[SIZE][SIZE], x[SIZE],x\_new[SIZE];

float temp, lambda\_new, lambda\_old, error=0.00001;

int i,j,n, step=1;

/\* Setting precision and writing floating point values in fixed-point notation. \*/

cout<< setprecision(3)<< fixed;

/\* Inputs \*/

/\* Reading order of square matrix \*/

cout<<" Enter Order of Matrix: ";

cin>>n;

/\* Reading Square Matrix of Order n \*/

cout<<" Enter Coefficient of Matrix: "<< endl;

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

{

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

{

cin>>a[i][j];

}

}

/\* Reading Intial Guess Vector \*/

cout<<" Enter Initial Guess Vector: "<< endl;

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

{

cin>>x[i];

}

/\* Initializing Lambda\_Old \*/

lambda\_old = 1;

/\* Multiplication \*/

/\* Setting label for repetition \*/

up:

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

{

temp = 0.0;

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

{

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

}

x\_new[i] = temp;

}

/\* Replacing x by x\_new \*/

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

{

x[i] = x\_new[i];

}

/\* Finding largest value from x\*/

lambda\_new = fabs(x[1]);

for(i=2;i<=n;i++)

{

if(fabs(x[i])>lambda\_new)

{

lambda\_new = fabs(x[i]);

}

}

/\* Normalization \*/

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

{

x[i] = x[i]/lambda\_new;

}

/\* Display \*/

cout<< endl<< endl<<" STEP-"<< step<< endl;

cout<<" Eigen Value = "<< lambda\_new<< endl;

cout<<" Eigen Vector: "<< endl;

cout<<" [";

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

{

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

}

cout<<"\b\b\b]"; /\* \b is for backspace \*/

/\* Checking Accuracy \*/

if(fabs(lambda\_new-lambda\_old)>error)

{

lambda\_old=lambda\_new;

step++;

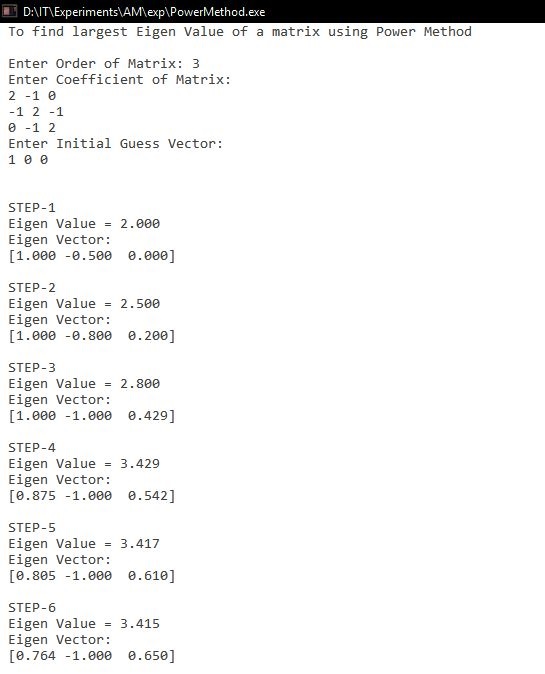
goto up;

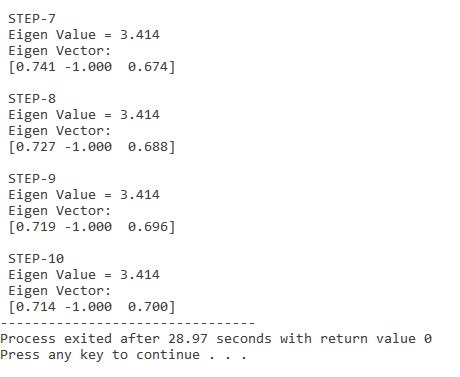
}

return(0);

}

**Output**

****

****

**Experiment 9**

// To find out solution of ordinary differential equations by Euler's Method

#include <iostream>

#include <iomanip>

/\* defining ordinary differential equation to be solved \*/

/\* In this code we are solving dy/dx = x + y \*/

#define f(x,y) x+y

using namespace std;

int main()

{

cout << "\n\n To find out solution of ordinary differential equation (dy/dx=x+y) by Euler's Method\n\n";

cout << setprecision(4);

float x0, y0, xn, h, yn, slope;

int i, n;

cout<<" Enter Initial Condition"<< endl;

cout<<" x0 = ";

cin>> x0;

cout<<" y0 = ";

cin >> y0;

cout<<" Enter calculation point xn = ";

cin>>xn;

cout<<" Enter number of steps: ";

cin>> n;

/\* Calculating step size (h) \*/

h = (xn-x0)/n;

/\* Euler's Method \*/

cout<<"\n x0\ty0\tslope\tyn\n";

cout<<"------------------------------\n";

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

{

slope = f(x0, y0);

yn = y0 + h \* slope;

cout << " " << x0<<"\t"<< y0<<"\t"<< slope<<"\t"<< yn<< endl;

y0 = yn;

x0 = x0+h;

}

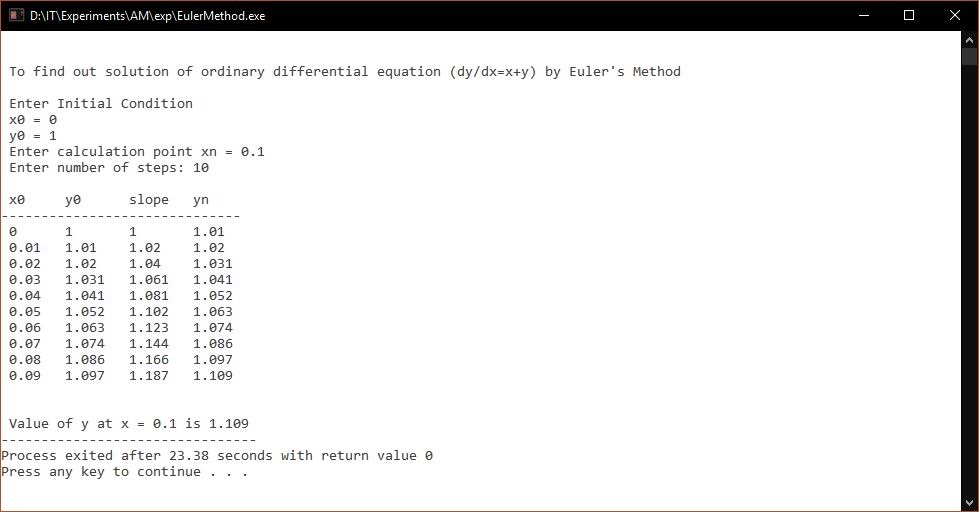
/\* Displaying result \*/

cout<<"\n\n Value of y at x = "<< xn<< " is " << yn;

return 0;

}

**Output**



**Experiment 10**

// To find out solution of ordinary differential equation dy/dx = (x-y)/2 using Runge Kutta Method

#include<iostream>

#include<iomanip>

/\* Defining ordinary differential equation to be solved \*/

using namespace std;

/\* defining ordinary differential equation to be solved \*/

#define f(x,y) (x-y)/2

using namespace std;

int main()

{

cout << "\n\n To find out solution of ordinary differential equation dy/dx = (x-y)/2 using Runge Kutta Method \n\n";

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

int i, n;

cout << setprecision(4);

cout<<" Enter Initial Condition"<< endl;

cout<<" x0 = ";

cin>> x0;

cout<<" y0 = ";

cin >> y0;

cout<<" Enter calculation point xn = ";

cin>>xn;

cout<<" Enter number of steps: ";

cin>> n;

/\* Calculating step size (h) \*/

h = (xn-x0)/n;

/\* Runge Kutta Method \*/

cout<<"\n x0\ty0\tyn\n";

cout<<"------------------\n";

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

{

k1 = h \* (f(x0, y0));

k2 = h \* (f((x0+h/2), (y0+k1/2)));

k3 = h \* (f((x0+h/2), (y0+k2/2)));

k4 = h \* (f((x0+h), (y0+k3)));

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

yn = y0 + k;

cout<< " " << x0<<"\t"<< y0<<"\t"<< yn<< endl;

x0 = x0+h;

y0 = yn;

}

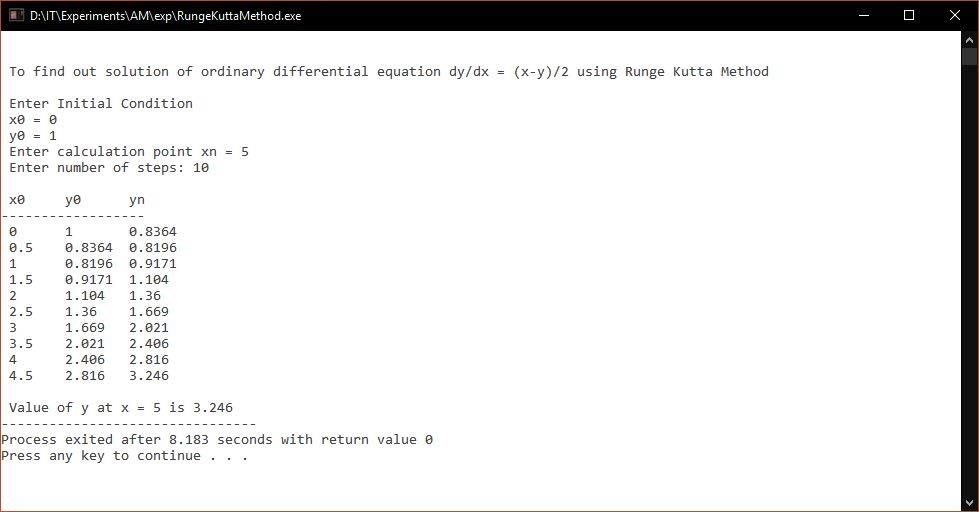
/\* Displaying result \*/

cout<<"\n Value of y at x = "<< xn<< " is " << yn;

return 0;

}

**Output**

****