

**Collage name**

Collage address

Tribhuvan university

**Lab report of Numerical Method**

|  |  |
| --- | --- |
| **Submitted by:**  ……….  Semester: …..semester  Roll no: ….. | **Submitted to:**  Ajit poudyal  Lecturer |

Contents

[Lab 1. solution of non linear equation 3](#_Toc193402056)

[1. Bisection method 3](#_Toc193402057)

[OR shorter codes 4](#_Toc193402058)

[2. Netwon Raphson Method 6](#_Toc193402059)

[3. Secant method 8](#_Toc193402060)

[Lab2: interpolation and approximation 11](#_Toc193402061)

[1. Lagrange interpolation 11](#_Toc193402062)

[2. Newton interpolation using forward method 13](#_Toc193402063)

[3. Newton interpolation using backward method 15](#_Toc193402064)

[4. Newton interpolation using dividend method 17](#_Toc193402065)

[Lab3: numerical differentiation and integration 20](#_Toc193402066)

[1. Trapezoidal Rule 20](#_Toc193402067)

[2. Simson’s 1/3 rule 22](#_Toc193402068)

[3. Simson;s 3/8 Rule 24](#_Toc193402069)

[Lab 4:solution of linear algebraic equation 26](#_Toc193402070)

[1. Gauss elimination method 26](#_Toc193402071)

[2. Gauss Jordan method 29](#_Toc193402072)

[3. Matrix inversion using Gauss Jordan method 31](#_Toc193402073)

[4. Matrix Factorization using Doolittle LU Decomposition 34](#_Toc193402074)

[5. Matrix factorization using Cholesky’s method 37](#_Toc193402075)

[6. Jacob iterative method 39](#_Toc193402076)

[7. Gauss sedial iterative method 41](#_Toc193402077)

[8. Power method 43](#_Toc193402078)

[Lab5. Solution of ordinary differential equation 46](#_Toc193402079)

[1. Talyor series 46](#_Toc193402080)

[2. Picard’s method 48](#_Toc193402081)

[3. Euler’s method 51](#_Toc193402082)

[4. Heun’s method 53](#_Toc193402083)

[5. Range-kutta method 55](#_Toc193402084)

[6. Boundary value problem 57](#_Toc193402085)

[7. Shooting method 62](#_Toc193402086)

# Lab 1. solution of non linear equation

## Bisection method

#include<stdio.h>

#include<conio.h>

#include<math.h>

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

void main()

{

float x0, x1, x2, f0, f1, f2, e;

int step = 1;

up:

printf("\nEnter two initial guesses:\n");

scanf("%f%f", &x0, &x1);

printf("Enter tolerable error:\n");

scanf("%f", &e);

f0 = f(x0);

f1 = f(x1);

if( f0 \* f1 > 0.0)

{

printf("Incorrect Initial Guesses.\n");

goto up;

}

printf("\nStep\t\tx0\t\tx1\t\tx2\t\tf(x2)\n");

do

{

x2 = (x0 + x1)/2;

f2 = f(x2);

printf("%d\t\t%f\t%f\t%f\t%f\n",step, x0, x1, x2, f2);

if( f0 \* f2 < 0)

{

x1 = x2;

f1 = f2;

}

else

{

x0 = x2;

f0 = f2;

}

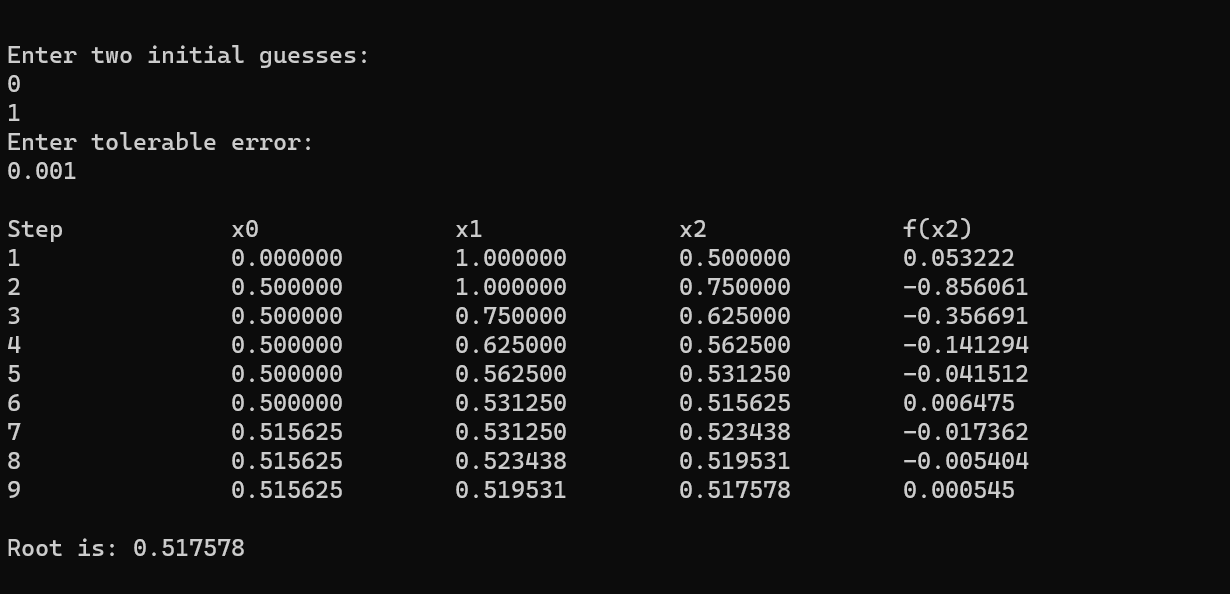
step = step + 1;

}while(fabs(f2)>e);

printf("\nRoot is: %f", x2);

getch();

output:



## OR shorter codes

#include<stdio.h>

#include<math.h>

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

void main() {

float a,b,c,fa,fb,fc,e;

do {

printf("Enter a, b: ");

scanf("%f%f",&a,&b);

fa=f(a); fb=f(b);

} while(fa\*fb>0);

printf("Enter error: ");

scanf("%f",&e);

do {

c=(a+b)/2;

fc=f(c);

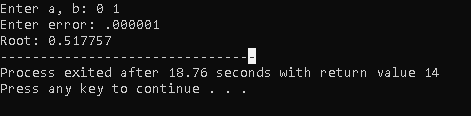
if(fa\*fc<0) b=c,fb=fc;

else a=c,fa=fc;

} while(fabs(fc)>e);

printf("Root: %f",c);

}



## Netwon Raphson Method

#include<conio.h>

#include<math.h>

#include<stdlib.h>

#define f(x) 3\*x - cos(x) -1

#define g(x) 3 + sin(x)

void main()

{

float x0, x1, f0, f1, g0, e;

int step = 1, N;

printf("\nEnter initial guess:\n");

scanf("%f", &x0);

printf("Enter tolerable error:\n");

scanf("%f", &e);

printf("Enter maximum iteration:\n");

scanf("%d", &N);

printf("\nStep\t\tx0\t\tf(x0)\t\tx1\t\tf(x1)\n");

do

{

g0 = g(x0);

f0 = f(x0);

if(g0 == 0.0)

{

printf("Mathematical Error.");

exit(0);

}

x1 = x0 - f0/g0;

printf("%d\t\t%f\t%f\t%f\t%f\n",step,x0,f0,x1,f1);

x0 = x1;

step = step+1;

if(step > N)

{

printf("Not Convergent.");

exit(0);

}

f1 = f(x1);

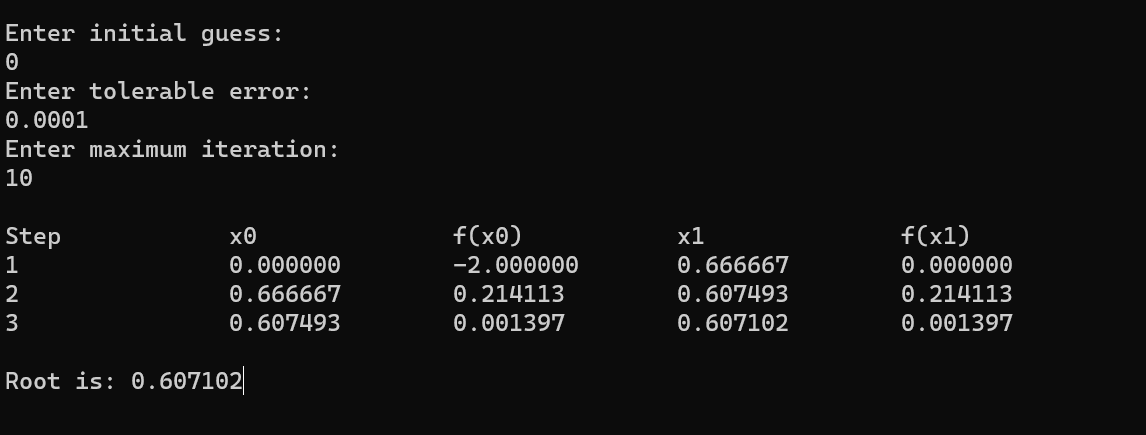
}while(fabs(f1)>e);

printf("\nRoot is: %f", x1);

getch();

}

Output:



## S**ecant method**

#include<stdio.h>

#include<conio.h>

#include<math.h>

#include<stdlib.h>

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

void main()

{

float x0, x1, x2, f0, f1, f2, e;

int step = 1, N;

printf("\nEnter initial guesses:\n");

scanf("%f%f", &x0, &x1);

printf("Enter tolerable error:\n");

scanf("%f", &e);

printf("Enter maximum iteration:\n");

scanf("%d", &N);

printf("\nStep\t\tx0\t\tx1\t\tx2\t\tf(x2)\n");

do

{

f0 = f(x0);

f1 = f(x1);

if(f0 == f1)

{

printf("Mathematical Error.");

exit(0);

}

x2 = x1 - (x1 - x0) \* f1/(f1-f0);

f2 = f(x2);

printf("%d\t\t%f\t%f\t%f\t%f\n",step,x0,x1,x2, f2);

x0 = x1;

f0 = f1;

x1 = x2;

f1 = f2;

step = step + 1;

if(step > N)

{

printf("Not Convergent.");

exit(0);

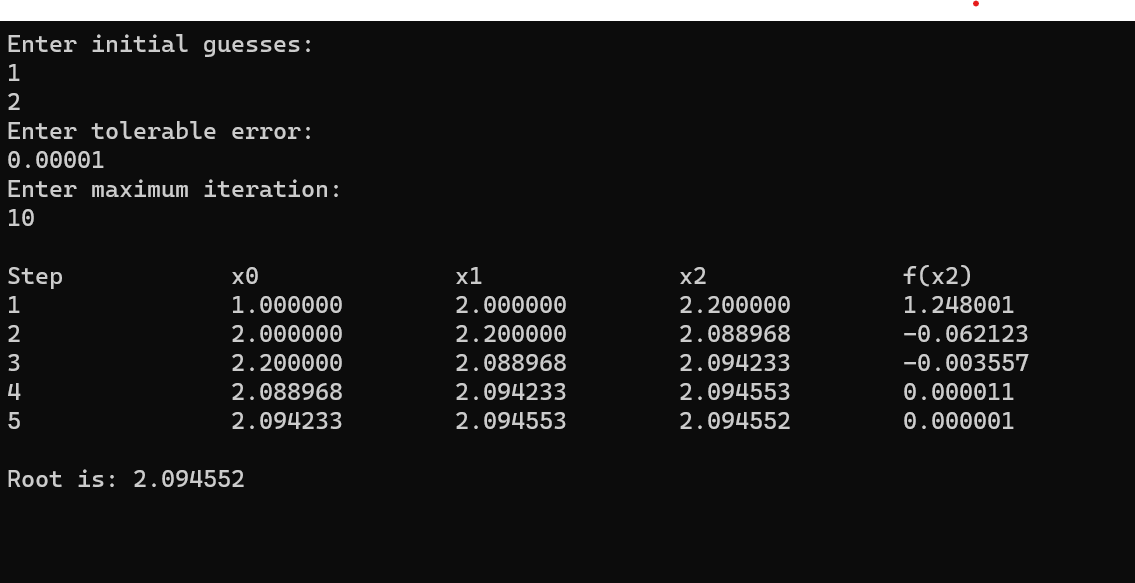
}

}while(fabs(f2)>e);

printf("\nRoot is: %f", x2);

getch();

output:



1. **False Position Method (Regula-Falsi Method)**

#include <stdio.h>

#include <math.h>

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

int main() {

float a, b, c, fa, fb, fc, e;

int step=1,N;

printf("Enter a, b, error: ");

scanf("%f%f%f", &a, &b, &e);

fa = f(a), fb = f(b);

if (fa \* fb > 0) {

printf("No root here.\n");

return 1;

}

printf("\na\t\tb\t\tc\t\tf(a)\t\tf(b)\t\tf(c)\n");

do {

c = (a\*fb - b\*fa)/(fb - fa);

fc = f(c);

printf("%f\t%.6f\t%.6f\t% 6f\t%.6f\t% .6f\n", a, b, c, f(a),fb,fc);

if (fa \* fc < 0)

b = c, fb = fc;

else

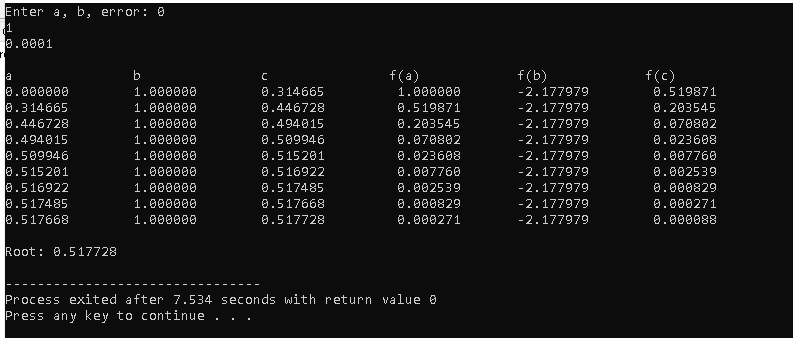
a = c, fa = fc;

} while (fabs(fc) > e);

printf("\nRoot: %.6f\n", c);

return 0;

}



# Lab2: interpolation and approximation

## Lagrange interpolation

#include<stdio.h>

#include<conio.h>

void main()

{

float x[100], y[100], xp, yp=0, p;

int i,j,n;

printf("Enter number of data: ");

scanf("%d", &n);

printf("Enter data:\n");

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

{

printf("x[%d] = ", i);

scanf("%f", &x[i]);

printf("y[%d] = ", i);

scanf("%f", &y[i]);

}

printf("Enter interpolation point: ");

scanf("%f", &xp);

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

{

p=1;

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

{

if(i!=j)

{

p = p\* (xp - x[j])/(x[i] - x[j]);

}

}

yp = yp + p \* y[i];

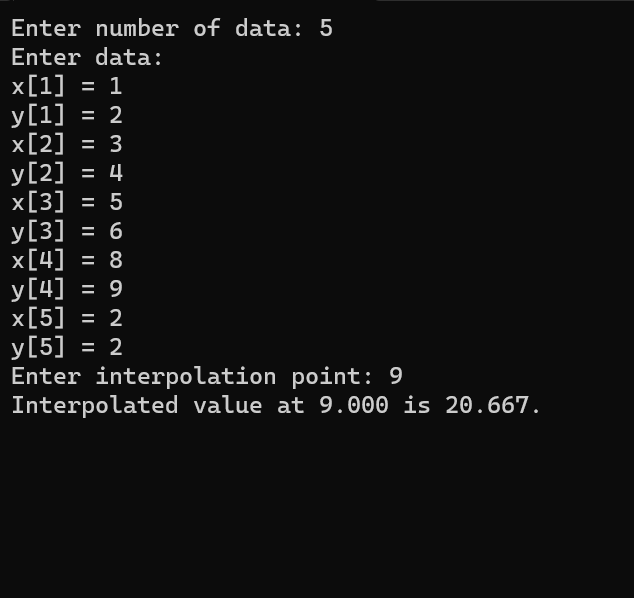
}

printf("Interpolated value at %.3f is %.3f.", xp, yp);

getch();

}

Output:



## Newton interpolation using forward method

#include<stdio.h>

#include<conio.h>

int main()

{

float x[20], y[20][20];

int i,j, n;

/\* Input Section \*/

printf("Enter number of data?\n");

scanf("%d", &n);

printf("Enter data:\n");

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

{

printf("x[%d]=", i);

scanf("%f", &x[i]);

printf("y[%d]=", i);

scanf("%f", &y[i][0]);

}

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

{

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

{

y[j][i] = y[j+1][i-1] - y[j][i-1];

}

}

printf("\nFORWARD DIFFERENCE TABLE\n\n");

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

{

printf("%0.2f", x[i]);

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

{

printf("\t%0.2f", y[i][j]);

}

printf("\n");

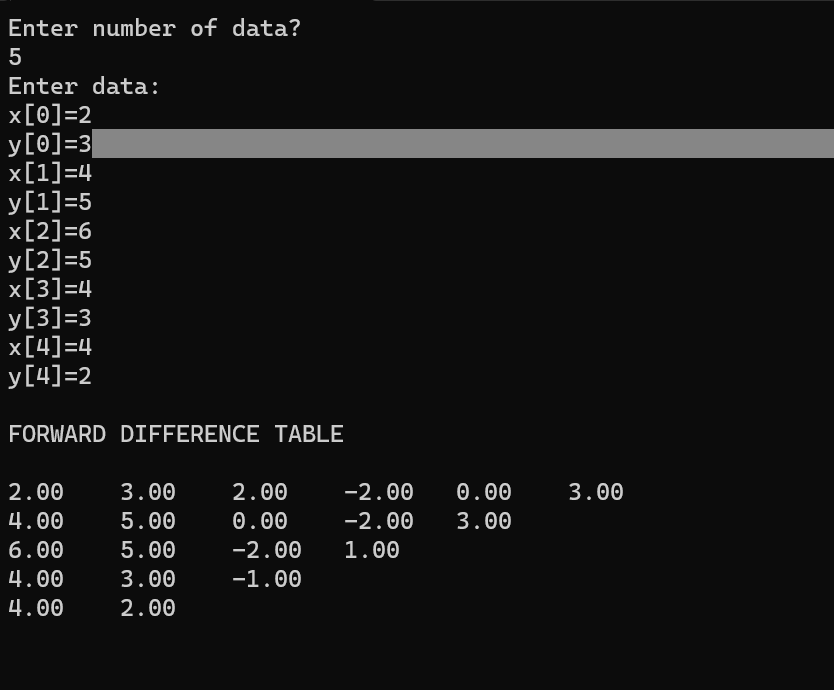
}

getch();

return 0;

}

Output:



## Newton interpolation using backward method

#include<stdio.h>

#include<conio.h>

int main()

{

float x[20], y[20][20];

int i,j, n;

printf("Enter number of data?\n");

scanf("%d", &n);

printf("Enter data:\n");

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

{

printf("x[%d]=", i);

scanf("%f", &x[i]);

printf("y[%d]=", i);

scanf("%f", &y[i][0]);

}

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

{

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

{

y[j][i] = y[j][i-1] - y[j-1][i-1];

}

}

printf("\nBACKWARD DIFFERENCE TABLE\n\n");

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

{

printf("%0.2f", x[i]);

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

{

printf("\t%0.2f", y[i][j]);

}

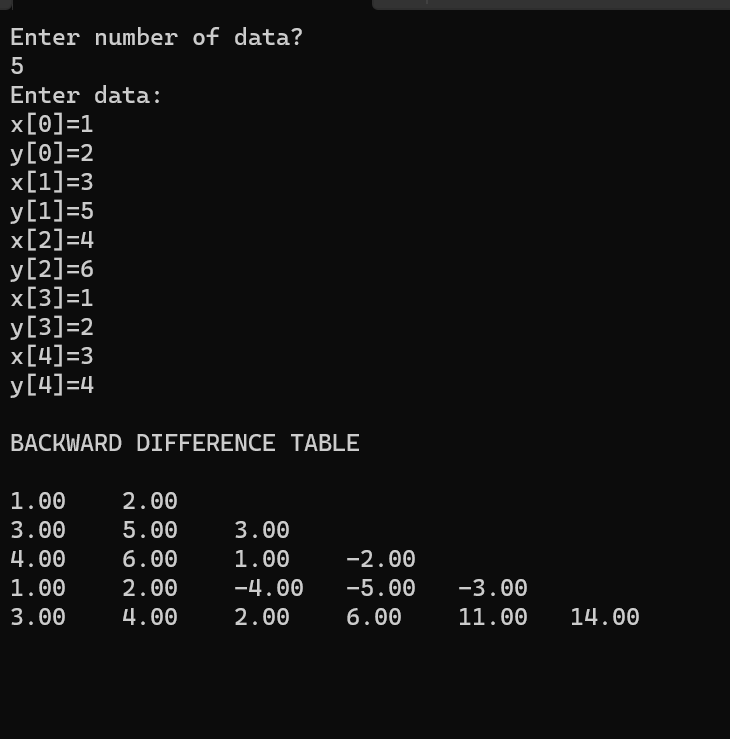
printf("\n");

}

getch();

return 0;

}



## Newton interpolation using dividend method

#include<stdio.h>

#include<conio.h>

void main()

{

int x[10], y[10], p[10];

int k,f,n,i,j=1,f1=1,f2=0;

printf("\nEnter the number of observations:\n");

scanf("%d", &n);

printf("\nEnter the different values of x:\n");

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

scanf("%d", &x[i]);

printf("\nThe corresponding values of y are:\n");

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

scanf("%d", &y[i]);

f=y[1];

printf("\nEnter the value of 'k' in f(k) you want to evaluate:\n");

scanf("%d", &k);

do

{

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

{

p[i] = ((y[i+1]-y[i])/(x[i+j]-x[i]));

y[i]=p[i];

}

f1=1;

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

{

f1\*=(k-x[i]);

}

f2+=(y[1]\*f1);

n--;

j++;

}

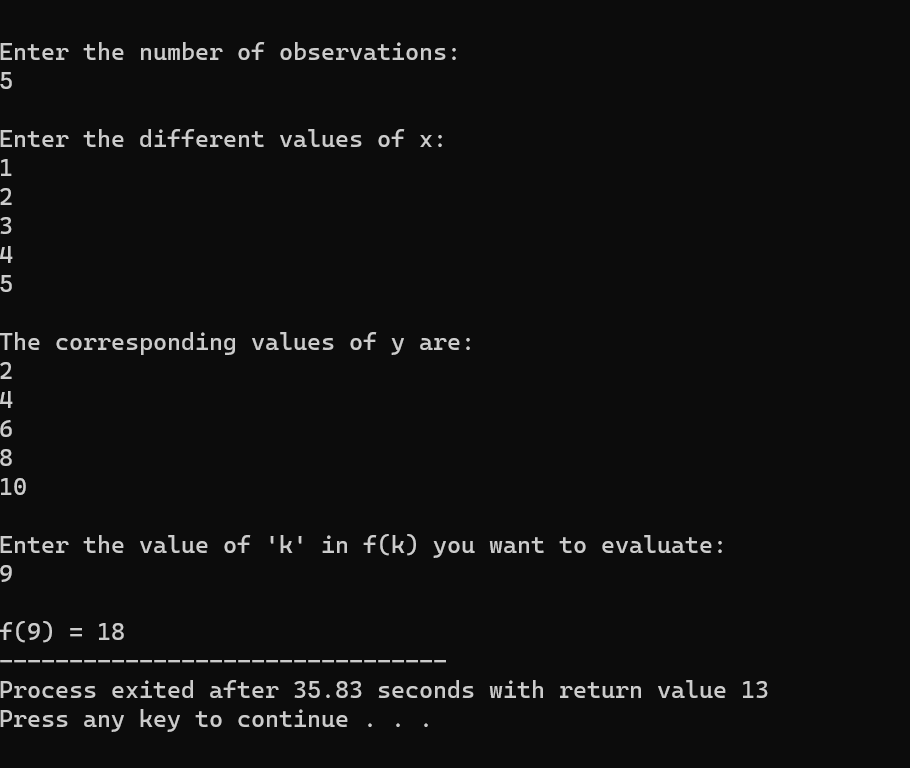
while(n!=1);

f+=f2;

printf("\nf(%d) = %d", k , f);

getch();

Output:



# Lab3: numerical differentiation and integration

## Trapezoidal Rule

#include<stdio.h>

#include<conio.h>

#include<math.h>

/\* Define function here \*/

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

int main()

{

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

int i, subInterval;

printf("Enter lower limit of integration: ");

scanf("%f", &lower);

printf("Enter upper limit of integration: ");

scanf("%f", &upper);

printf("Enter number of sub intervals: ");

scanf("%d", &subInterval);

stepSize = (upper - lower)/subInterval;

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;

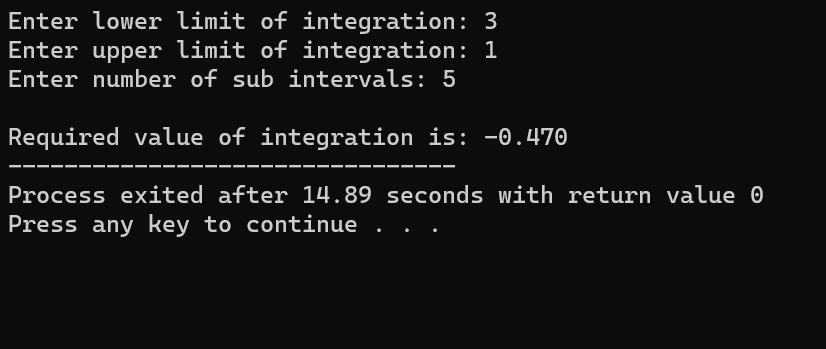
printf("\nRequired value of integration is: %.3f", integration);

getch();

return 0;

}

Output:



## Simson’s 1/3 rule

#include<stdio.h>

#include<conio.h>

#include<math.h>

#define f(x) 1/(1+x\*x)

int main()

{

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

int i, subInterval;

printf("Enter lower limit of integration: ");

scanf("%f", &lower);

printf("Enter upper limit of integration: ");

scanf("%f", &upper);

printf("Enter number of sub intervals: ");

scanf("%d", &subInterval);

stepSize = (upper - lower)/subInterval;

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

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;

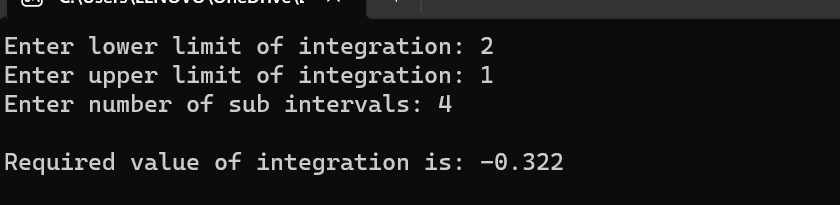
printf("\nRequired value of integration is: %.3f", integration);

getch();

return 0;

}

Output:



## Simson;s 3/8 Rule

#include<stdio.h>

#include<conio.h>

#include<math.h>

/\* Define function here \*/

#define f(x) 1/(1+x\*x)

int main()

{

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

int i, subInterval;

printf("Enter lower limit of integration: ");

scanf("%f", &lower);

printf("Enter upper limit of integration: ");

scanf("%f", &upper);

printf("Enter number of sub intervals: ");

scanf("%d", &subInterval);

stepSize = (upper - lower)/subInterval;

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

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\*3/8;

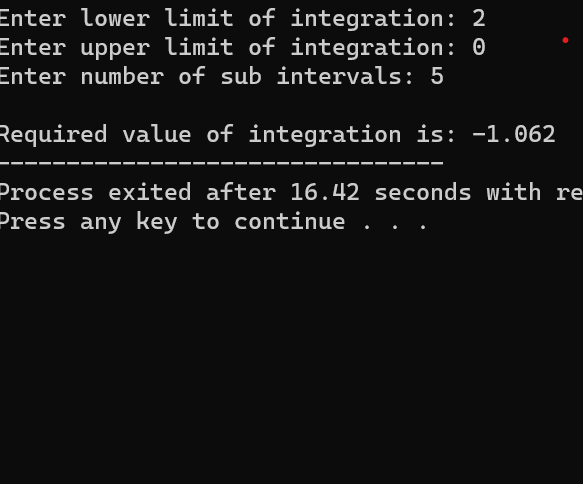
printf("\nRequired value of integration is: %.3f", integration);

getch();

return 0;

}

Output:



# Lab 4:solution of linear algebraic equation

## Gauss elimination method

#include<stdio.h>

#include<conio.h>

#include<math.h>

#include<stdlib.h>

#define SIZE 10

int main()

{

float a[SIZE][SIZE], x[SIZE], ratio;

int i,j,k,n;

printf("Enter number of unknowns: ");

scanf("%d", &n);

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

{

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

{

printf("a[%d][%d] = ",i,j);

scanf("%f", &a[i][j]);

}

}

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

{

if(a[i][i] == 0.0)

{

printf("Mathematical Error!");

exit(0);

}

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

{

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

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

{

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

}

}

}

x[n] = a[n][n+1]/a[n][n];

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

{

x[i] = a[i][n+1];

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

{

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

}

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

}

printf("\nSolution:\n");

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

{

printf("x[%d] = %0.3f\n",i, x[i]);

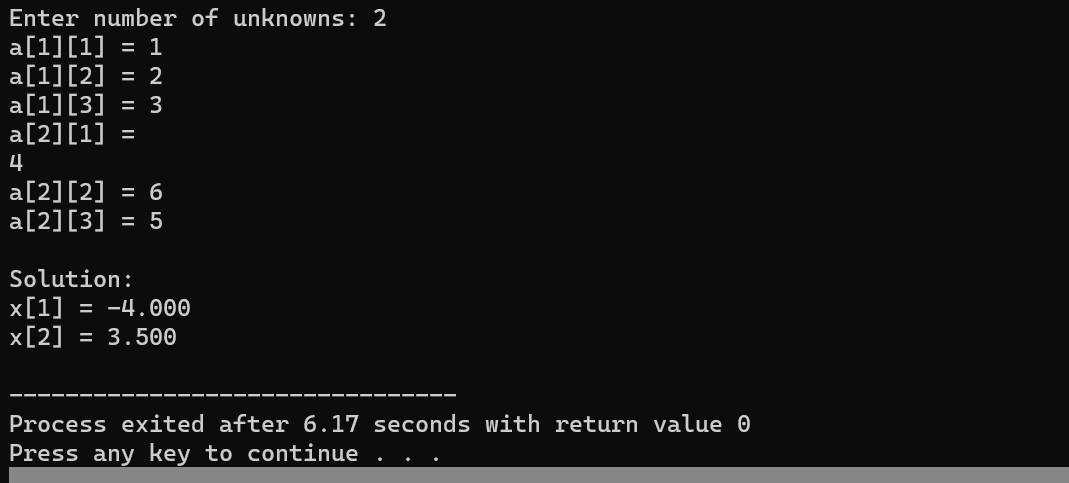
}

getch();

return(0);

}

Output:



## Gauss Jordan method

#include<stdio.h>

#include<conio.h>

#include<math.h>

#define SIZE 10

int main()

{

float a[SIZE][SIZE], x[SIZE], ratio;

int i,j,k,n;

printf("Enter number of unknowns: ");

scanf("%d", &n);

printf("Enter coefficients of Augmented Matrix:\n");

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

{

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

{

printf("a[%d][%d] = ",i,j);

scanf("%f", &a[i][j]);

}

}

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

{

if(a[i][i] == 0.0)

{

printf("Mathematical Error!");

exit(0);

}

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

{

if(i!=j)

{ratio = a[j][i]/a[i][i];

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

{

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

} }} }

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

{

x[i] = a[i][n+1]/a[i][i];

}

printf("\nSolution:\n");

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

{

printf("x[%d] = %0.3f\n",i, x[i]);

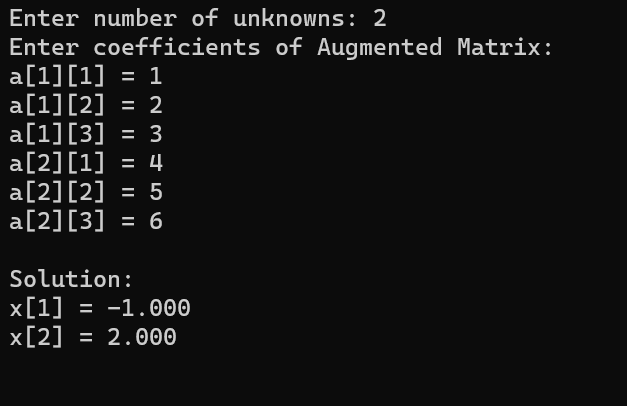
}

getch();

return(0);

}

Output:



## Matrix inversion using Gauss Jordan method

#include<stdio.h>

#include<conio.h>

#include<math.h>

#include<stdlib.h>

#define SIZE 10

int main()

{

float a[SIZE][SIZE], x[SIZE], ratio;

int i,j,k,n;

printf("Enter number of unknowns: ");

scanf("%d", &n);

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

{

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

{

printf("a[%d][%d] = ",i,j);

scanf("%f", &a[i][j]);

}

}

/\* Applying Gauss Elimination \*/

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

{

if(a[i][i] == 0.0)

{

printf("Mathematical Error!");

exit(0);

}

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

{

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

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

{

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

}

}

}

x[n] = a[n][n+1]/a[n][n];

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

{

x[i] = a[i][n+1];

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

{

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

}

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

}

printf("\nSolution:\n");

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

{

printf("x[%d] = %0.3f\n",i, x[i]);

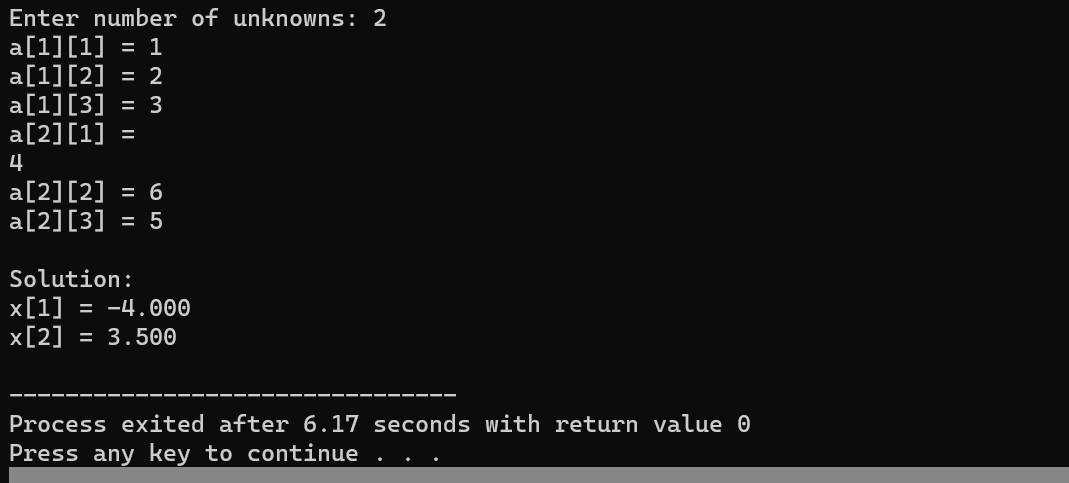
}

getch();

return(0);

}

Output:



## Matrix Factorization using Doolittle LU Decomposition

#include<stdio.h>

#include<conio.h>

void main()

{

float A[20][20]= {0},L[20][20]= {0}, U[20][20];

float B[20]= {0}, X[20]= {0},Y[20]= {0};

int i,j,k,n;

printf("Enter the order of square matrix: ");

scanf("%d",&n);

printf("\nEnter matrix element:\n");

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

{

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

{

printf("Enter A[%d][%d] element: ", i,j);

scanf("%f",&A[i][j]);

}

}

printf("\nEnter the constant terms: \n");

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

{

printf("B[%d]",i);

scanf("%f",&B[i]);

}

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

{

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

{

if(i<=j)

{

U[i][j]=A[i][j];

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

U[i][j]-=L[i][k]\*U[k][j];

if(i==j)

L[i][j]=1;

else

L[i][j]=0;

}

else

{

L[i][j]=A[i][j];

for(k=0; k<=j-1; k++)

L[i][j]-=L[i][k]\*U[k][j];

L[i][j]/=U[j][j];

U[i][j]=0;

}

}

}

printf("[L]: \n");

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

{

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

printf("%9.3f",L[i][j]);

printf("\n");

}

printf("\n\n[U]: \n");

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

{

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

printf("%9.3f",U[i][j]);

printf("\n");

}

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

{

Y[i]=B[i];

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

{

Y[i]-=L[i][j]\*Y[j];

}

}

printf("\n\n[Y]: \n");

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

{

printf("%9.3f",Y[i]);

}

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

{

X[i]= Y[i];

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

{

X[i]-=U[i][j]\*X[j];

}

X[i]/=U[i][i];

}

printf("\n\n[X]: \n");

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

{

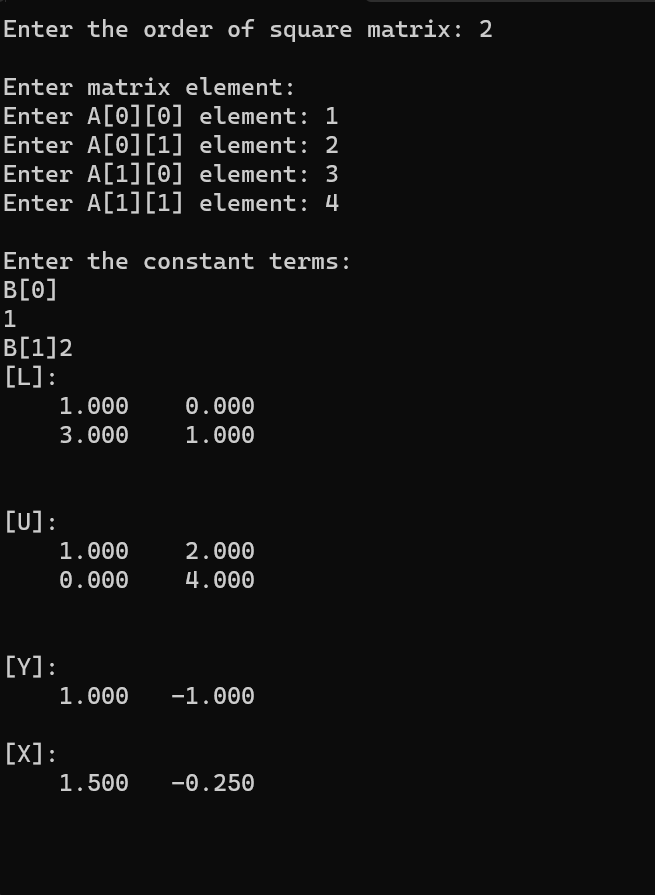
printf("%9.3f",X[i]);

}

getch();

}

Output:

,

## Matrix factorization using Cholesky’s method

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

double \*cholesky(double \*A, int n) {

double \*L = (double\*)calloc(n \* n, sizeof(double));

if (L == NULL)

exit(EXIT\_FAILURE);

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

for (int j = 0; j < (i+1); j++) {

double s = 0;

for (int k = 0; k < j; k++)

s += L[i \* n + k] \* L[j \* n + k];

L[i \* n + j] = (i == j) ?

sqrt(A[i \* n + i] - s) :

(1.0 / L[j \* n + j] \* (A[i \* n + j] - s));

}

return L;

}

void show\_matrix(double \*A, int n) {

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

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

printf("%2.5f ", A[i \* n + j]);

printf("\n");

}

}

int main() {

int n = 3;

double m1[] = {25, 15, -5,

15, 18, 0,

-5, 0, 11};

double \*c1 = cholesky(m1, n);

show\_matrix(c1, n);

printf("\n");

free(c1);

n = 4;

double m2[] = {18, 22, 54, 42,

22, 70, 86, 62,

54, 86, 174, 134,

42, 62, 134, 106};

double \*c2 = cholesky(m2, n);

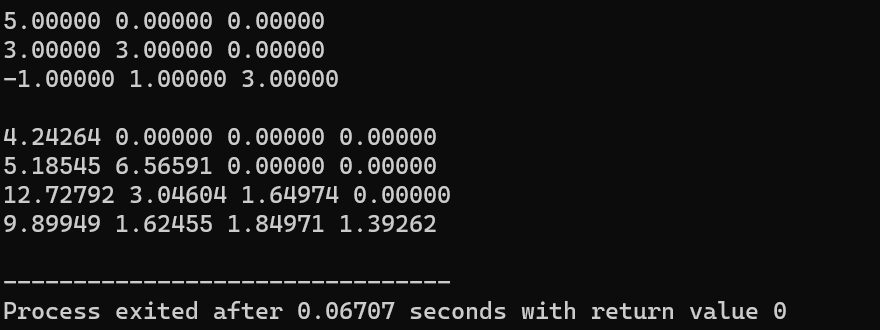
show\_matrix(c2, n);

free(c2);

return 0;

}

Output:



## Jacob iterative method

#include<stdio.h>

#include<conio.h>

#include<math.h>

#define f1(x,y,z) (17-y+2\*z)/20

#define f2(x,y,z) (-18-3\*x+z)/20

#define f3(x,y,z) (25-2\*x+3\*y)/20

/\* Main function \*/

int main()

{

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

int count=1;

printf("Enter tolerable error:\n");

scanf("%f", &e);

printf("\nCount\tx\ty\tz\n");

do

{

/\* Calculation \*/

x1 = f1(x0,y0,z0);

y1 = f2(x0,y0,z0);

z1 = f3(x0,y0,z0);

printf("%d\t%0.4f\t%0.4f\t%0.4f\n",count, x1,y1,z1);

e1 = fabs(x0-x1);

e2 = fabs(y0-y1);

e3 = fabs(z0-z1);

count++;

x0 = x1;

y0 = y1;

z0 = z1;

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

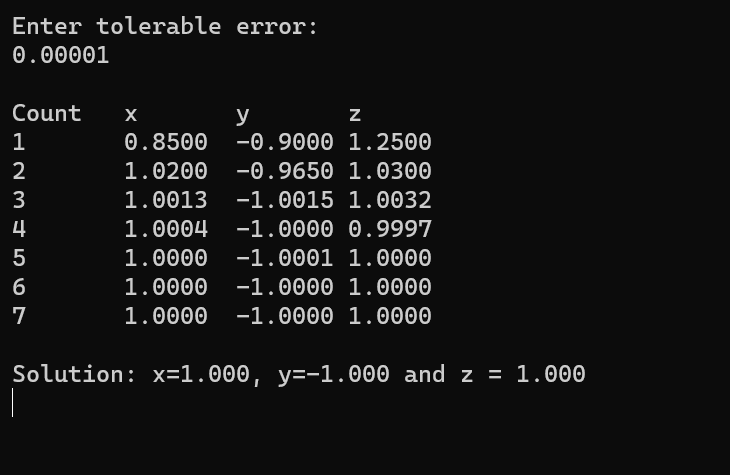
printf("\nSolution: x=%0.3f, y=%0.3f and z = %0.3f\n",x1,y1,z1);

getch();

return 0;

}

Output:



## Gauss sedial iterative method

#include<stdio.h>

#include<conio.h>

#include<math.h>

#define f1(x,y,z) (17-y+2\*z)/20

#define f2(x,y,z) (-18-3\*x+z)/20

#define f3(x,y,z) (25-2\*x+3\*y)/20

/\* Main function \*/

int main()

{

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

int count=1;

printf("Enter tolerable error:\n");

scanf("%f", &e);

printf("\nCount\tx\ty\tz\n");

do

{

/\* Calculation \*/

x1 = f1(x0,y0,z0);

y1 = f2(x1,y0,z0);

z1 = f3(x1,y1,z0);

printf("%d\t%0.4f\t%0.4f\t%0.4f\n",count, x1,y1,z1);

/\* Error \*/

e1 = fabs(x0-x1);

e2 = fabs(y0-y1);

e3 = fabs(z0-z1);

count++;

/\* Set value for next iteration \*/

x0 = x1;

y0 = y1;

z0 = z1;

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

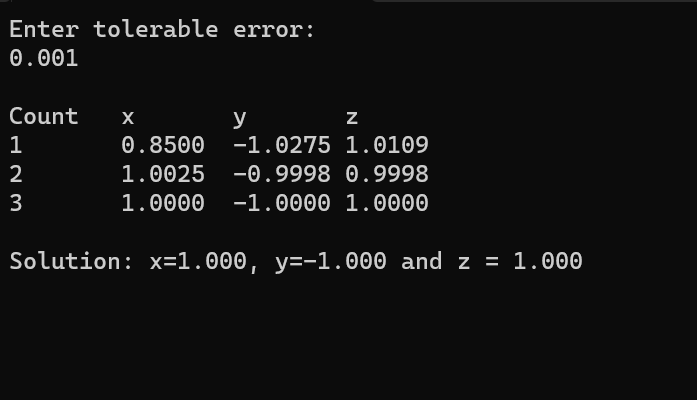
printf("\nSolution: x=%0.3f, y=%0.3f and z = %0.3f\n",x1,y1,z1);

getch();

return 0;

}

Output:



## Power method

#include<stdio.h>

#include<conio.h>

#include<math.h>

#define SIZE 10

int main()

{

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

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

int i,j,n, step=1;

/\* Inputs \*/

printf("Enter Order of Matrix: ");

scanf("%d", &n);

printf("Enter Tolerable Error: ");

scanf("%f", &error);

/\* Reading Matrix \*/

printf("Enter Coefficient of Matrix:\n");

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

{

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

{

printf("a[%d][%d]=",i,j);

scanf("%f", &a[i][j]);

}

}

/\* Reading Intial Guess Vector \*/

printf("Enter Initial Guess Vector:\n");

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

{

printf("x[%d]=",i);

scanf("%f", &x[i]);

}

/\* Initializing Lambda\_Old \*/

lambda\_old = 1;

/\* Multiplication \*/

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 \*/

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

{

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

}

/\* Finding Largest \*/

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 \*/

printf("\n\nSTEP-%d:\n", step);

printf("Eigen Value = %f\n", lambda\_new);

printf("Eigen Vector:\n");

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

{

printf("%f\t", x[i]);

}

/\* Checking Accuracy \*/

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

{

lambda\_old=lambda\_new;

step++;

goto up;

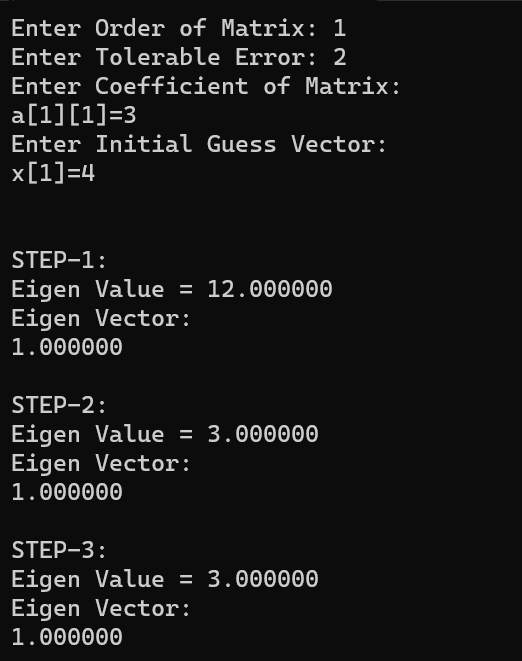
}

getch();

return(0);

}

Output:



# Lab5. Solution of ordinary differential equation

## Talyor series

#include<stdio.h>

#include<math.h>

int main()

{

int x,i;

int fact = 1,n;

float sum=0;

printf("\n\nEnter the value of x in the series : ");

scanf("%d",&x);

printf("\nEnter the number of terms in the series : ");

scanf("%d",&n);

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

{

fact = fact\*i;

sum = sum + (pow(x,i)/fact) ;

}

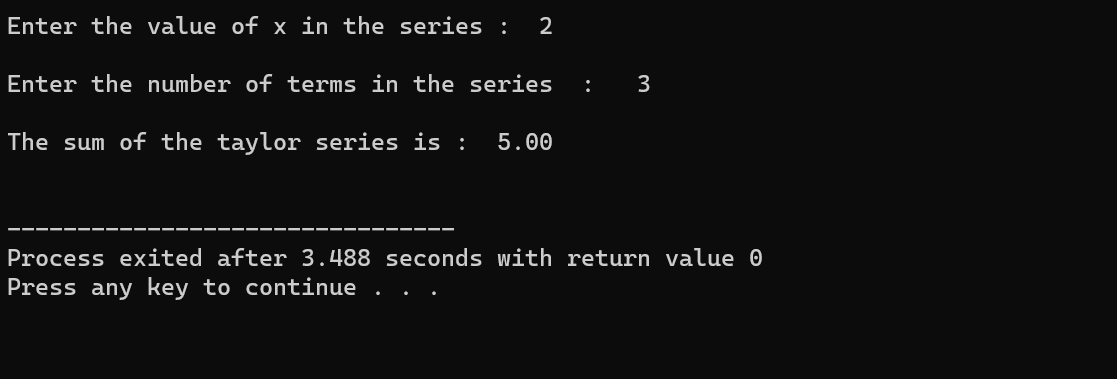
sum= sum +1; //Since series starts with 1

printf("\nThe sum of the taylor series is : %.2f\n\n",sum);

return 0;

}

Output:



## Picard’s method

#include <math.h>

#include <stdio.h>

#define Y1(x) (1 + (x) + pow(x, 2) / 2)

#define Y2(x) (1 + (x) + pow(x, 2) / 2 + pow(x, 3) / 3 + pow(x, 4) / 8)

#define Y3(x) (1 + (x) + pow(x, 2) / 2 + pow(x, 3) / 3 + pow(x, 4) / 8 + pow(x, 5) / 15 + pow(x, 6) / 48)

int main()

{

double start\_value = 0, end\_value = 3,

allowed\_error = 0.4, temp;

double y1[30], y2[30], y3[30];

int count;

for (temp = start\_value, count = 0;

temp <= end\_value;

temp = temp + allowed\_error, count++) {

y1[count] = Y1(temp);

y2[count] = Y2(temp);

y3[count] = Y3(temp);

}

printf("\nX\n");

for (temp = start\_value;

temp <= end\_value;

temp = temp + allowed\_error) {

printf("%.4lf ", temp);

}

printf("\n\nY(1)\n");

for (temp = start\_value, count = 0;

temp <= end\_value;

temp = temp + allowed\_error, count++) {

printf("%.4lf ", y1[count]);

}

printf("\n\nY(2)\n");

for (temp = start\_value, count = 0;

temp <= end\_value;

temp = temp + allowed\_error, count++) {

printf("%.4lf ", y2[count]);

}

printf("\n\nY(3)\n");

for (temp = start\_value, count = 0;

temp <= end\_value;

temp = temp + allowed\_error, count++) {

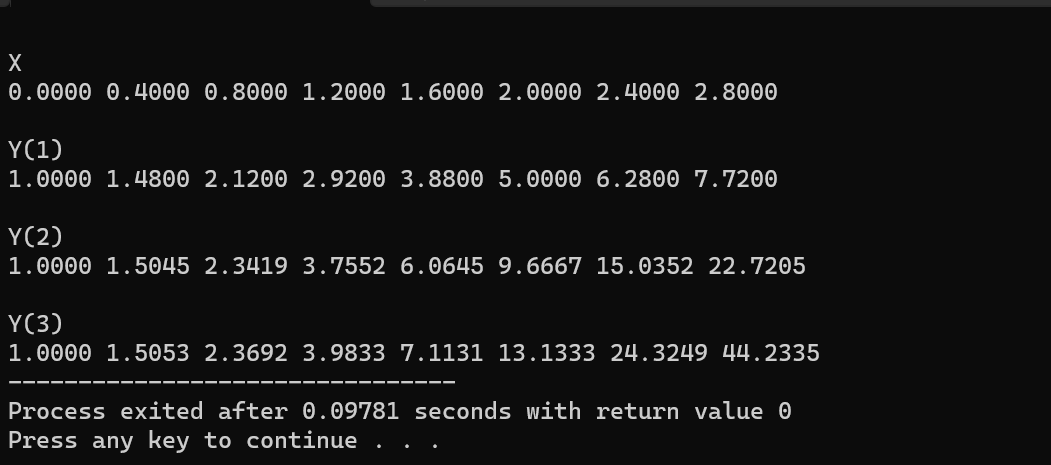
printf("%.4lf ", y3[count]);

}

return 0;

}

Output:



## Euler’s method

#include<stdio.h>

#include<conio.h>

#define f(x,y) x+y

int main()

{

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

int i, n;

printf("Enter Initial Condition\n");

printf("x0 = ");

scanf("%f", &x0);

printf("y0 = ");

scanf("%f", &y0);

printf("Enter calculation point xn = ");

scanf("%f", &xn);

printf("Enter number of steps: ");

scanf("%d", &n);

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

h = (xn-x0)/n;

/\* Euler's Method \*/

printf("\nx0\ty0\tslope\tyn\n");

printf("------------------------------\n");

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

{

slope = f(x0, y0);

yn = y0 + h \* slope;

printf("%.4f\t%.4f\t%0.4f\t%.4f\n",x0,y0,slope,yn);

y0 = yn;

x0 = x0+h;

}

/\* Displaying result \*/

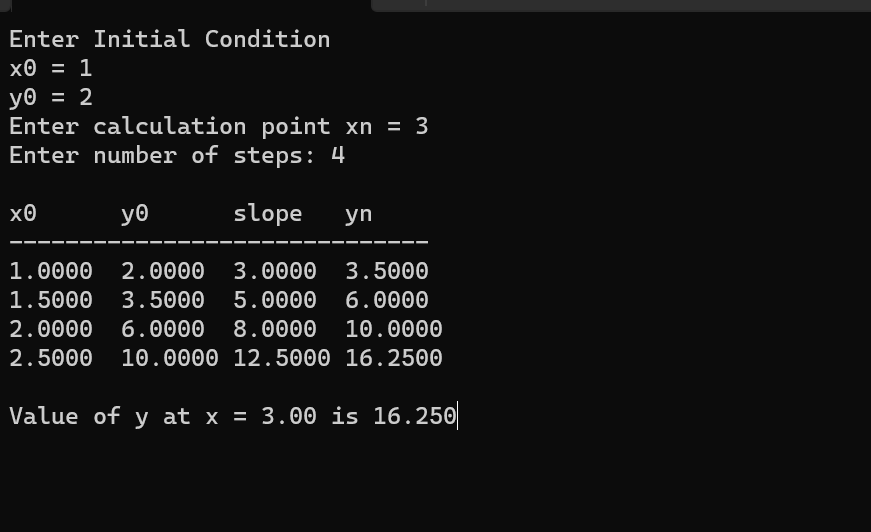
printf("\nValue of y at x = %0.2f is %0.3f",xn, yn);

getch();

return 0;

}

Output:



## Heun’s method

#include<conio.h>

#include<stdio.h>

#define f(x,y) 2\*y/x

void main()

{

float x,y,h,xn,l;

printf("Program for Solution of Ordinary Differential Equation\nHeun's Method\n");

printf("Enter value for x and y\n");

scanf("%f%f",&x,&y);

printf("Enter value for h and last of x\n");

scanf("%f%f",&h,&xn);

while(x+h<=xn)

{

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

y=y+l;

x=x+h;

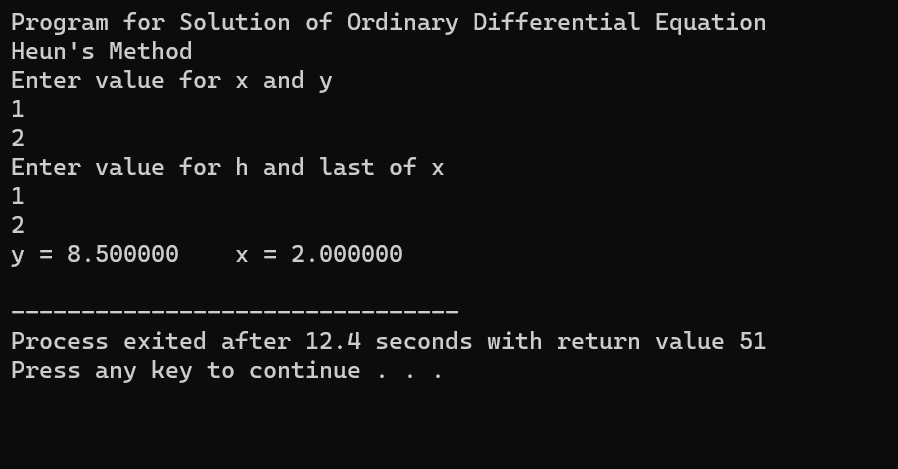
printf("y = %f\tx = %f\n",y,x);

}

getch();

}

Output:



## Range-kutta method

#include<stdio.h>

#include<conio.h>

#define f(x,y) (y\*y-x\*x)/(y\*y+x\*x)

int main()

{

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

int i, n;

printf("Enter Initial Condition\n");

printf("x0 = ");

scanf("%f", &x0);

printf("y0 = ");

scanf("%f", &y0);

printf("Enter calculation point xn = ");

scanf("%f", &xn);

printf("Enter number of steps: ");

scanf("%d", &n);

h = (xn-x0)/n;

printf("\nx0\ty0\tyn\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;

printf("%0.4f\t%0.4f\t%0.4f\n",x0,y0,yn);

x0 = x0+h;

y0 = yn;

}

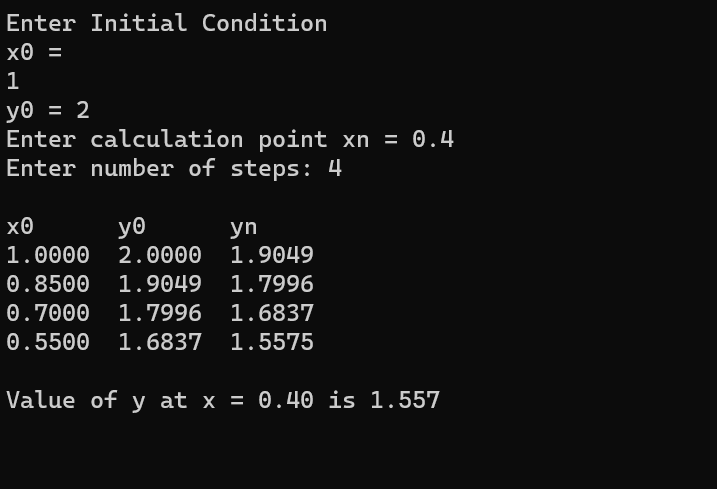
printf("\nValue of y at x = %0.2f is %0.3f",xn, yn);

getch();

return 0;

}

Output:



## Boundary value problem

#include<stdio.h>

#include<math.h>

#include<stdlib.h>

float f1(float x, float y, float z)

{

return(z);

}

float f2(float x, float y, float z)

{

return(x + y);

}

float shoot(float x0, float y0, float z0, float xn, float h, int p)

{

float x, y, z, k1, k2, k3, k4, l1, l2, l3, l4, k, l, x1, y1, z1;

x = x0;

y = y0;

z = z0;

do

{

k1 = h \* f1(x, y, z);

l1 = h \* f2(x, y, z);

k2 = h \* f1(x + h / 2.0, y + k1 / 2.0, z + l1 / 2.0);

l2 = h \* f2(x + h / 2.0, y + k1 / 2.0, z + l1 / 2.0);

k3 = h \* f1(x + h / 2.0, y + k2 / 2.0, z + l2 / 2.0);

l3 = h \* f2(x + h / 2.0, y + k2 / 2.0, z + l2 / 2.0);

k4 = h \* f1(x + h, y + k3, z + l3);

l4 = h \* f2(x + h, y + k3, z + l3);

l = 1 / 6.0 \* (l1 + 2 \* l2 + 2 \* l3 + l4);

k = 1 / 6.0 \* (k1 + 2 \* k2 + 2 \* k3 + k4);

y1 = y + k;

x1 = x + h;

z1 = z + l;

x = x1;

y = y1;

z = z1;

if (p == 1)

{

printf("\n%f\t%f", x, y);

}

} while (x < xn);

return(y);

}

main()

{

float x0, y0, h, xn, yn, z0, m1, m2, m3, b, b1, b2, b3, e;

int p = 0;

printf("\n Enter x0,y0,xn,yn,h:");

scanf("%f%f%f%f%f", &x0, &y0, &xn, &yn, &h);

printf("\n Enter the trial M1:");

scanf("%f", &m1);

b = yn;

z0 = m1;

b1 = shoot(x0, y0, z0, xn, h, p = 1);

printf("\nB1 is %f", b1);

if (fabs(b1 - b) < 0.00005)

{

printf("\n The value of x and respective z are:\n");

e = shoot(x0, y0, z0, xn, h, p = 1);

return(0);

}

else

{

printf("\nEnter the value of M2:");

scanf("%f", &m2);

z0 = m2;

b2 = shoot(x0, y0, z0, xn, h, p = 1);

printf("\nB2 is %f", b2);

}

if (fabs(b2 - b) < 0.00005)

{

printf("\n The value of x and respective z are\n");

e = shoot(x0, y0, z0, xn, h, p = 1);

return(0);

}

else

{

printf("\nM2=%f\tM1=%f", m2, m1);

m3 = m2 - ((m2 - m1) / (b2 - b1)) \* (b2 - b);

if (b1 - b2 == 0)

exit(0);

printf("\nExact value of M =%f", m3);

z0 = m3;

b3 = shoot(x0, y0, z0, xn, h, p = 0);

}

if (fabs(b3 - b) < 0.000005)

{

printf("\nThere is solution :\n");

e = shoot(x0, y0, z0, xn, h, p = 1);

exit(0);

}

do

{

m1 = m2;

m2 = m3;

b1 = b2;

b2 = b3;

m3 = m2 - ((m2 - m1) / (b2 - b1)) \* (b2 - b);

z0 = m3;

b3 = shoot(x0, y0, z0, xn, h, p = 0);

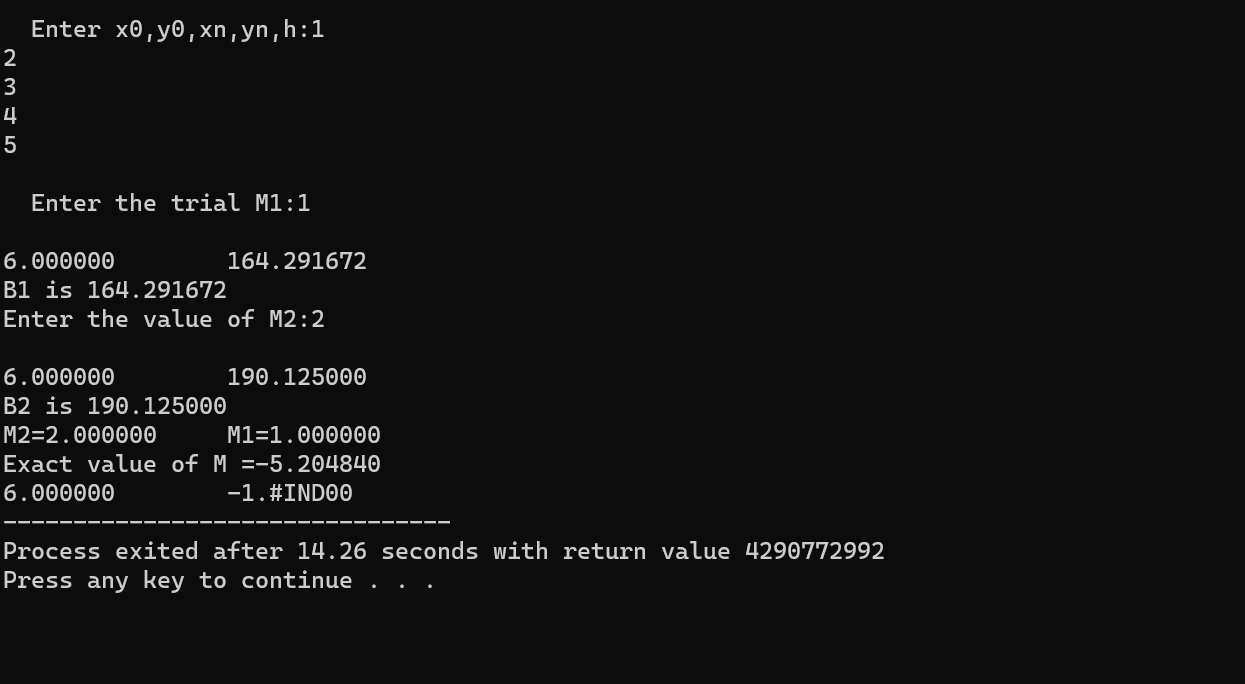
} while (fabs(b3 - b) < 0.0005);

z0 = m3;

e = shoot(x0, y0, z0, xn, h, p = 1);

}

Output:



## Shooting method

#include<stdio.h>

#include<math.h>

#include<stdlib.h>

float f1(float x,float y,float z)

{

return(z);

}

float f2(float x,float y,float z)

{

return(x+y);

}

float shoot(float x0,float y0,float z0,float xn,float h,int p)

{

float x,y,z,k1,k2,k3,k4,l1,l2,l3,l4,k,l,x1,y1,z1;

x=x0;

y=y0;

z=z0;

do

{

k1=h\*f1(x,y,z);

l1=h\*f2(x,y,z);

k2=h\*f1(x+h/2.0,y+k1/2.0,z+l1/2.0);

l2=h\*f2(x+h/2.0,y+k1/2.0,z+l1/2.0);

k3=h\*f1(x+h/2.0,y+k2/2.0,z+l2/2.0);

l3=h\*f2(x+h/2.0,y+k2/2.0,z+l2/2.0);

k4=h\*f1(x+h,y+k3,z+l3);

l4=h\*f2(x+h,y+k3,z+l3);

l=1/6.0\*(l1+2\*l2+2\*l3+l4);

k=1/6.0\*(k1+2\*k2+2\*k3+k4);

y1=y+k;

x1=x+h;

z1=z+l;

x=x1;

y=y1;

z=z1;

if(p==1)

{

printf("\n%f\t%f",x,y);

}

}while(x<xn);

return(y);

}

void main()

{

float x0,y0,h,xn,yn,z0,m1,m2,m3,b,b1,b2,b3,e;

int p=0;

printf("\n Enter x0,y0,xn,yn,h:");

scanf("%f%f%f%f%f",&x0,&y0,&xn,&yn,&h);

printf("\n Enter the trial M1:");

scanf("%f",&m1);

b=yn;

z0=m1;

b1=shoot(x0,y0,z0,xn,h,p=1);

printf("\nB1 is %f",b1);

if(fabs(b1-b)<0.00005)

{

printf("\n The value of x and respective z are:\n");

e=shoot(x0,y0,z0,xn,h,p=1);

return(0);

}

else

{

printf("\nEnter the value of M2:");

scanf("%f",&m2);

z0=m2;

b2=shoot(x0,y0,z0,xn,h,p=1);

printf("\nB2 is %f",b2);

}

if(fabs(b2-b)<0.00005)

{

printf("\n The value of x and respective z are\n");

e= shoot(x0,y0,z0,xn,h,p=1);

return(0);

}

else

{

printf("\nM2=%f\tM1=%f",m2,m1);

m3=m2+(((m2-m1)\*(b-b2))/(1.0\*(b2-b1)));

if(b1-b2==0)

exit(0);

printf("\nExact value of M =%f",m3);

z0=m3;

b3=shoot(x0,y0,z0,xn,h,p=0);

}

if(fabs(b3-b)<0.000005)

{

printf("\nThere is solution :\n");

e=shoot(x0,y0,z0,xn,h,p=1);

exit(0);

}

do

{

m1=m2;

m2=m3;

b1=b2;

b2=b3;

m3=m2+(((m2-m1)\*(b-b2))/(1.0\*(b2-b1)));

z0=m3;

b3=shoot(x0,y0,z0,xn,h,p=0);

}while(fabs(b3-b)<0.0005);

z0=m3;

e=shoot(x0,y0,z0,xn,h,p=1);

}

Output:

