**ASSIGNMENT – 8: NUMERICAL METHODS**

1. Write a menu-driven program for solving a system of linear equations using Gauss-Elimination method, Jacobi’s method and Gauss Elimination with pivoting method

Code:

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

#define MAX\_SIZE 100

void gauss\_elimination(float a[MAX\_SIZE][MAX\_SIZE+1],int n){

//forward elimination

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

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

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

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

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

}

a[j][MAX\_SIZE]-=f\*a[i][MAX\_SIZE];

}

}

//Checking for inconsistent solutions

int singular=0;

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

if (a[k][k]==0)

singular=1;

else

singular=0;

}

if (singular){

printf("Singluar Matrix, Aborting");

exit(0);

}

//back substitution

float x[MAX\_SIZE];

x[n-1]=a[n-1][MAX\_SIZE]/a[n-1][n-1];

for (int k=n-2;k>=0;k--){

x[k]=a[k][MAX\_SIZE];

for (int j=n-1;j>k;j--){

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

}

x[k]/=a[k][k];

}

printf("\nSolution of the system of equations:\n");

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

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

}

}

void gauss\_jordan(float a[MAX\_SIZE][MAX\_SIZE+1],int n){

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

//swapping of the ith and the i+1th row

int j=i+1;

while (a[i][i]==0 && j<n){

float temp;

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

temp=a[i][k];

a[i][k]=a[j][k];

a[j][k]=temp;

}

j++;

}

float f=a[i][i];

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

a[i][k]/=f;

}

a[i][MAX\_SIZE]/=f;

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

if (p!=i){

float g= a[p][i];

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

a[p][k]-=a[i][k]\*g;

}

a[p][MAX\_SIZE]-=a[i][MAX\_SIZE]\*g;

}

}

}

printf("\nSolution of the system of equations:\n");

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

printf("x[%d] = %f\n",k,a[k][MAX\_SIZE]);

}

}

void jacobi\_method(float a[MAX\_SIZE][MAX\_SIZE+1], float x[MAX\_SIZE], int n, float eps, int max\_iterations)

{

float x\_temp[MAX\_SIZE];

for (int i=0;i<max\_iterations;i++){

//j is for different equations

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

float sum=a[j][MAX\_SIZE];

//k is for dealing in the same equation

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

if (j!=k){

sum=sum-a[j][k]\*x[k];

}

}

x\_temp[j]=sum/a[j][j];

}

int converged,big =0;

float temp;

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

// temp=fabs((x\_temp[k]-x[k])/x\_temp[k]);

temp=fabs((x\_temp[k]-x[k]));

if (temp<eps){

converged=1;

}

else

converged=0;

}

if (converged){

printf("\nSolution of the system of equations:\n");

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

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

}

printf("Number of iterations: %d\n",i);

return;

}

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

x[k]=x\_temp[k];

}

}

printf("\nSolution not converged within %d iterations.\n",max\_iterations);

}

int main()

{

int i,j,n,max\_iterations;

float a[MAX\_SIZE][MAX\_SIZE+1]={0}, x[MAX\_SIZE]={0}, eps;

printf("Enter the number of equations: ");

scanf("%d",&n);

printf("Enter the coefficient matrix:\n");

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

{

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

{

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

}

}

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

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

{

scanf("%f",&a[i][MAX\_SIZE]);

}

getchar();

int choice;

printf("Enter your choice of method :\n1.Gauss Elimination\n2.Jocobi's method\n3.Gauss Pivotal method\n");

choice = getchar();

switch(choice){

case '1': gauss\_elimination(a,n); break;

case '2':

printf("Enter the initial values of x:\n");

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

{

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

}

printf("Enter the tolerance: ");

scanf("%f",&eps);

printf("Enter the maximum number of iterations: ");

scanf("%d",&max\_iterations);

jacobi\_method(a,x,n,eps,max\_iterations); break;

case '3': gauss\_jordan(a,n); break;

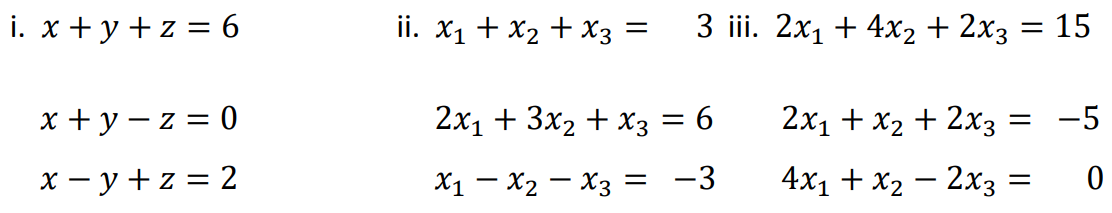
default : printf("Invalid Choice");break;

}

return 0;

}

1. Using the above program solve the following system of equations :



Output:

i)Using Gauss Jordan

Enter the number of equations: 3

Enter the coefficient matrix:

1 1 1

1 1 -1

1 -1 1

Enter the constant terms:

6 0 2

Enter your choice of method :

1.Gauss Elimination

2.Jocobi's method

3.Gauss Pivotal method

3

Solution of the system of equations:

x[0] = 1.000000

x[1] = 3.000000

x[2] = 2.000000

ii)Using Jacobi’s Method

Enter the number of equations: 3

Enter the coefficient matrix:

1 1 1

2 3 1

1 -1 -1

Enter the constant terms:

3 6 -3

Enter your choice of method :

1.Gauss Elimination

2.Jocobi's method

3.Gauss Pivotal method

2

Enter the initial values of x:

0 0 0

Enter the tolerance: 0.0001

Enter the maximum number of iterations: 100

Solution of the system of equations:

x[0] = -0.000152

x[1] = 1.499975

x[2] = 1.499924

Number of iterations: 28

iii)Using Gauss Elimination

Enter the number of equations: 3

Enter the coefficient matrix:

2 4 2

2 1 2

4 1 -2

Enter the constant terms:

15 -5 0

Enter your choice of method :

1.Gauss Elimination

2.Jocobi's method

3.Gauss Pivotal method

1

Solution of the system of equations:

x[0] = -3.055555

x[1] = 6.666667

x[2] = -2.777777

1. Write a menu-driven program for implementing Interpolation using Lagrange's formula, Newton’s forward difference formula, and Newton’s backward difference formula.

Code:

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

#define MAX\_SIZE 10

void lagrange(float x[MAX\_SIZE], float y[MAX\_SIZE], int n, float x0);

void forward\_diff(float x[MAX\_SIZE], float y[MAX\_SIZE], int n, float x0);

void backward\_diff(float x[MAX\_SIZE], float y[MAX\_SIZE], int n, float x0);

int factorial(int n);

int main()

{

int i,n;

float x[MAX\_SIZE], y[MAX\_SIZE], x0;

printf("Enter the number of data points: ");

scanf("%d",&n);

printf("Enter the data points:\n");

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

{

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

}

printf("Enter the value of x at which the function is to be interpolated: ");

scanf("%f",&x0);

getchar();

int choice;

printf("Enter your choice of method :\n1.Lagrange\n2.Netwons Forward Difference\n3.Newton's backward difference\n");

choice = getchar();

switch(choice){

case '1': lagrange(x,y,n,x0); break;

case '2': forward\_diff(x,y,n,x0); break;

case '3': backward\_diff(x,y,n,x0); break;

}

return 0;

}

void lagrange(float x[MAX\_SIZE], float y[MAX\_SIZE], int n, float x0)

{

int i,j;

float l, sum = 0;

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

{

l = 1;

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

{

if(j!=i)

{

l \*= (x0 - x[j])/(x[i] - x[j]);

}

}

sum += l\*y[i];

}

printf("The value of the function at x = %f is %f\n",x0,sum);

}

void forward\_diff(float x[MAX\_SIZE], float y[MAX\_SIZE], int n, float x0)

{

int i,j;

float fd[MAX\_SIZE][MAX\_SIZE], h, p, sum = 0;

// calculate forward difference table

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

{

fd[i][0] = y[i];

}

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

{

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

{

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

}

}

// calculate h and p

h = x[1] - x[0];

p = (x0 - x[0])/h;

// calculate the value of the function at x0 using the forward difference table

sum = y[0];

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

{

sum += (p/factorial(i))\*fd[0][i];

p \*= (p-i);

}

printf("The value of the function at x = %f is %f\n",x0,sum);

}

void backward\_diff(float x[MAX\_SIZE], float y[MAX\_SIZE], int n, float x0)

{

int i,j;

float bd[MAX\_SIZE][MAX\_SIZE], h, p, sum = 0;

// calculate backward difference table

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

{

bd[i][0] = y[i];

}

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

{

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

{

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

}

}

// calculate h and p

h = x[1] - x[0];

p = (x0 - x[n-1])/h;

// calculate the value of the function at x0 using the backward difference table

sum = y[n-1];

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

{

sum += (p/factorial(i))\*bd[n-1][i];

p \*= (p+i);

}

printf("The value of the function at x = %f is %f\n",x0,sum);

}

// function to calculate factorial

int factorial(int n)

{

if(n==0)

{

return 1;

}

else

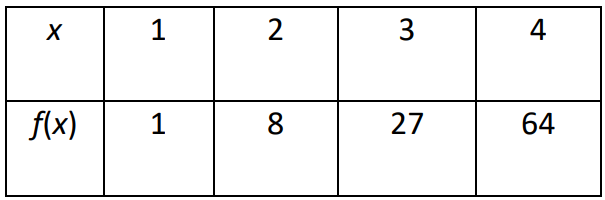
{

return n\*factorial(n-1);

}

}

1. For the following table of values:



Find f(2.5) using all three methods and comment on your answer

Output:

Enter the number of data points: 4

Enter the data points:

1 1

2 8

3 27

4 64

Enter the value of x at which the function is to be interpolated: 2.5

Enter your choice of method :

1.Lagrange

2.Netwons Forward Difference

3.Newton's backward difference

1

The value of the function at x = 2.500000 is 15.625000

2

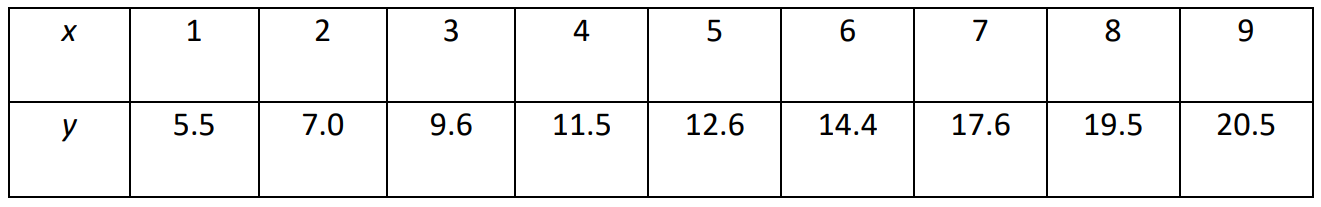
The value of the function at x = 2.500000 is 15.062500

3

The value of the function at x = 2.500000 is 17.312500

Comment : At the value 2.5, the error in interpolation is least for Lagrange’s method and maximum for Newton’s Forward difference method. (as the actual value of the function at 2.5 is 15.625).

1. An experiment gave the following table of values for the dependent variable y for a set of known values of x. Obtain an appropriate least squares fit for the data.



Code:

#include <stdio.h>

int main() {

float x[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};

float y[] = {5.5,7.0,9.6,11.5,12.6,14.4,17.6,19.5,20.5};

int n = 9;

float sum\_x = 0, sum\_y = 0, sum\_xy = 0, sum\_x2 = 0;

// Calculate the required sums

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

sum\_x += x[i];

sum\_y += y[i];

sum\_xy += x[i] \* y[i];

sum\_x2 += x[i] \* x[i];

}

// Calculate the slope and y-intercept of the line of best fit

float m = (n \* sum\_xy - sum\_x \* sum\_y) / (n \* sum\_x2 - sum\_x \* sum\_x);

float c = (sum\_y - m \* sum\_x) / n;

// Print the equation of the line of best fit

printf("y = %fx + %f\n", m, c);

return 0;

}

Output:

y = 1.940000x + 3.433332