NUMERICAL METHODS

LAB ASSIGNMENTS

Priyank Lohariwal

BCSE-II (A2)
001710501055

Index

1.Bairstow's Method

```
/*Program to apply bairstow's mathod*/
#include<stdio.h>
#include<math.h>
#define max 10
int main(void)
{
	FILE *fp;
	fp=fopen("1.txt","w");
	fprintf(fp,"  LIN BAIRSTOW METHOD.\n\n");
	float a[max], b[max], bc[max], bd[max], p, q, r, s, error, errorp, errorq, rq, sq, rp, sp;
	int i, degree, iter=0;
/*accepting the polynomial equation in the following form*/
	printf("\n Enter degree of polynomial?  ");
	scanf("%d",&degree);
	fprintf(fp,"The given equation is:\n");
	for(i=degree; i>=0; i--)
	{
		printf("\n Enter coefficient of x^%d:  ",i);
		if(i<degree)
		{
			scanf("%f",&a[degree-1-i]);
	 a[degree-1-i]/=a[degree];
	 fprintf(fp,"%f * x^%d ",a[degree-1-i],i);
		}
		else
		{
	 scanf("%f",&a[i]);
	 fprintf(fp,"%f * x^%d ",a[i],i);
   }
	}
	a[degree]=1;
	printf("\n X^2 + PX + Q \n Enter guesses for P and Q:  ");
	scanf("%f %f",&p,&q);
	error=1;
	while(error>=0.0005)
	{
	/* Beginning of Lin's method */
		for(i=0;i<=degree-3;i++)
			if(i==0)
				b[i]=a[i]-p;
	 else if(i==1)
	    b[i]=a[i]-p*b[i-1]-q;
	 else
	    b[i]=a[i]-p*b[i-1]-q*b[i-2];
	  /* Modification of Lin's to Bairstow's method */
	 r = a[degree-2] - p*b[degree-3] - q*b[degree-4];
	 s = a[degree-1] - q*b[degree-3];
	 for(i=0; i<=degree-3; i++)
		if(i==0)
		bc[i]=-1;
		else if(i==1)
	       bc[i]=-b[0]+p;
		else
	       bc[i]=b[i-1]-p*bc[i-1]-q*bc[i-2];
	 rp=-b[degree-3]-p*bc[degree-3]-q*bc[degree-4];
```

```
 sp=-q*bc[degree-3];
 for(i=0;i<=degree-3;i++)

     if(i==0)
            bd[i]=0;
            else if(i==1)
            bd[i]=-1;
            else
            bd[i]=-b[i-2]-p*bd[i-1]-q*bd[i-2];

 rq=-b[degree-4]-p*bd[degree-3]-q*bd[degree-4];
 sq=-b[degree-3]-q*bd[degree-3];

 errorp = (s*rq - r*sq)/(rp*sq - sp*rq);
 p+=errorp;
 errorq = (r*sp - s*rp)/(rp*sq - sp*rq);
 q+=errorq;

error=errorp>errorq?errorp:errorq;
 iter++;                                              /* iteration no. */
}
printf("\n\nA root of the given equation is X^2 + %f*X + %f",p,q);
fprintf(fp,"\n\nA root of the given equation is X^2 + %f*X + %f",p,q);
fclose(fp);}
```

Output

```
  LIN BAIRSTOW METHOD.

The given equation is:
1.000000 * x^4 4.000000 * x^3 6.000000 * x^2 4.000000 * x^1 1.000000 * x^0

A root of the given equation is X^2 + 1.999903*X + 0.999643
```

2.Bisection Method

```
#include<stdio.h>
#include<math.h>
float func(float x)                          //function to find the value of f(x)
{
    float res;
    res= exp(-x)-x;
    return res;
}
int main(void)                               //main begins
{
   FILE *fp;
   fp=fopen("2.txt","w");                    //file opens
   int iter=1;
   float a=-1.0,b=1.0,m,f_a,f_b,f_m,abs_error,multi;
   fprintf(fp,"SL.no\t\ta\t\t\tb\t\t\tm\t\t\tf(m)\terror\n");
   do                                        //loop for finding the root of the
function
   {  abs_error=fabs(b-a);
      m=(a+b)/2;
      f_a=func(a);                           //calling the function
      f_b=func(b);
      f_m=func(m);
      multi=f_m*f_a;
      fprintf(fp,"%d\t\t%f\t%f\t%f\t%f\t%f\n",iter,a,b,m,f_m,multi,abs_error);
      if(fabs(f_m)<=0.000005)                //checking for required precision
          break;
      else if(multi<0) b=m;
      else a=m;
      iter++;
   }while(fabs(b-a)>=0.000005);              //checking for minimal error
   fprintf(fp,"\n\nroot of the function = %f",m);      //printing result in file
   fclose(fp);
   printf("res = %f",m);                     //printing result in terminal
}
```

Output

SL.no	a	b	m	f(m)	error
1	-1.000000	1.000000	0.000000	1.000000	3.718282
2	0.000000	1.000000	0.500000	0.106531	0.106531
3	0.500000	1.000000	0.750000	-0.277633	-0.029576
4	0.500000	0.750000	0.625000	-0.089739	-0.009560
5	0.500000	0.625000	0.562500	0.007283	0.000776
6	0.562500	0.625000	0.593750	-0.041498	-0.000302
7	0.562500	0.593750	0.578125	-0.017176	-0.000125
8	0.562500	0.578125	0.570313	-0.004964	-0.000036
9	0.562500	0.570313	0.566406	0.001155	0.000008
10	0.566406	0.570313	0.568359	-0.001905	-0.000002
11	0.566406	0.568359	0.567383	-0.000375	-0.000000
12	0.566406	0.567383	0.566895	0.000390	0.000000
13	0.566895	0.567383	0.567139	0.000007	0.000000
14	0.567139	0.567383	0.567261	-0.000184	-0.000000
15	0.567139	0.567261	0.567200	-0.000088	-0.000000
16	0.567139	0.567200	0.567169	-0.000041	-0.000000
17	0.567139	0.567169	0.567154	-0.000017	-0.000000
18	0.567139	0.567154	0.567146	-0.000005	-0.000000

```
root of the function = 0.567146
```

3.Chebysev's Method

```
/*Program for describing chebysev method
f(x)= x^3-x-3*/
#include<stdio.h>
#include<math.h>
float func(float x)                        //function for finding the value of f(x)
{
        return (x*x*x-x-3);
}
float dif_func(float x)                    //function for finding the value of f'(x)
{
        return (3*x*x -1);
}
float dDifFunc(float x)                    //function for finding the value of f"(x)
{
        return (6*x);
}
float ri(float r)                    //function for finding the value of r_next
{
        return (r-(func(r)/dif_func(r))-
(func(r)*func(r)*dDifFunc(r)/(2*dif_func(r)*dif_func(r)*dif_func(r))));
}
float gf(float x)        //function for checking the convergence of the function
{
        float res;
        res=fabs(func(x)*dDifFunc(x)/((dif_func(x)*dif_func(x))));
        return res;
}
int main()                                 //main function begins here
{
        int iter=1;
        float r_i,r_i1,abs_error,prev_error=0,cond,conv;

        printf("Enter the initial point:\n");            //input taken from user
        scanf("%f",&r_i);

        FILE *fp;
        fp=fopen("3.txt","w");                     //opening of file in write mode
        fprintf(fp,"Sl.no.\t\tr  \t\tcondition\t  r_next\t\terror\torder of convergence\n");
        do                                         //loop for the method
        {
        cond=gf(r_i);
        r_i1=ri(r_i);                              //updating the value of r_next
        abs_error=fabs(r_i1-r_i);
        if(prev_error!=0) conv=log(abs_error)/log(prev_error);    //finding the      order of
convergence
        prev_error=abs_error;
        if(iter==1)
        fprintf(fp,"%2d\t\t%9f\t%9f\t%9f\t%9f\t   NA\n\n",iter,r_i,cond,r_i1,abs_error);
        else

        fprintf(fp,"%2d\t\t%9f\t%9f\t%9f\t%9f\t%9f\n\n",iter,r_i,cond,r_i1,abs_error,conv)
;
        iter++;
        r_i=r_i1;
        }while(abs_error>=0.00001);                    //checking precision
```

```
    printf("root of the function=%f\n",r_i1);            //printing result


    fprintf(fp,"root of the function=%f\n\n",r_i1);
    fclose(fp);
    return 0;
}
```

Output

```
Sl.no.            r          condition      r_next       error     order of convergence
 1            3.000000      0.559172      1.966488     1.033512          NA

 2            1.966488      0.276960      1.683182     0.283306       -38.262650

 3            1.683182      0.015344      1.671701     0.011481        3.541826

 4            1.671701      0.000002      1.671700     0.000001        3.053426

root of the function=1.671700
```

4. Fixed Point Iteration

```
/*Program to apply fixed point iteration*/
#include <stdio.h>
#include <math.h>
#define g(x) exp(-x)
#define dg(x) -exp(-x)
#define f(x) exp(-x)-x  //original function f(x)=e^(-x)-x
double x1,e1;
int iter=0;
int main()                                                    //main begins
{
        FILE *fp;                                                       //opening file
pointer
        fp=fopen("4.txt","w");
        double x,order,e;
        printf("Enter the starting point:");
        scanf("%lf",&x);                                                //taking
initial input from the user
        fprintf(fp,"SL\t\tx\t\tg(x)\t\tdg(x)\t\tf(x)\t\terror\t\torder\n");
        do                                                              //loop to
apply the method
        {
        iter++;
        if (fabs(dg(x))>1)                                      //checking for
convergence
        {
                printf("the process may not converge to the root\n");
                return 0;
        }
                x1=g(x);
                e=fabs(x1-x);
                if(iter==1)

        fprintf(fp,"%d\t%9f\t%9f\t%9f\t%9f\t%9f\tNA\n",iter,x,g(x),dg(x),f(x),e,order);
                else                                            //finding order of
convergence
                {
                        order=log(e)/log(e1);

        fprintf(fp,"%d\t%9f\t%9f\t%9f\t%9f\t%9f\t%9f\n",iter,x,g(x),dg(x),f(x),e,order);
                }
                e1=e;
                x=x1;
        }while(e>0.00005);                                      //checking the
precision
        printf("The root of the equation is %f\n",x1);                  //printing
final result
        fprintf(fp,"\n\nThe root of the equation is %f\n",x1);
                fclose(fp);
                return 0;}
```

Output

SL	x	g(x)	dg(x)	f(x)	error	order
1	0.000000	1.000000	-1.000000	1.000000	1.000000	NA
2	1.000000	0.367879	-0.367879	-0.632121	0.632121	-1.#INF00
3	0.367879	0.692201	-0.692201	0.324321	0.324321	2.454942
4	0.692201	0.500474	-0.500474	-0.191727	0.191727	1.466831
5	0.500474	0.606244	-0.606244	0.105770	0.105770	1.360121
6	0.606244	0.545396	-0.545396	-0.060848	0.060848	1.246114
7	0.545396	0.579612	-0.579612	0.034217	0.034217	1.205640
8	0.579612	0.560115	-0.560115	-0.019497	0.019497	1.166651
9	0.560115	0.571143	-0.571143	0.011028	0.011028	1.144723
10	0.571143	0.564879	-0.564879	-0.006264	0.006264	1.125489
11	0.564879	0.568429	-0.568429	0.003549	0.003549	1.111968
12	0.568429	0.566415	-0.566415	-0.002014	0.002014	1.100453
13	0.566415	0.567557	-0.567557	0.001142	0.001142	1.091407
14	0.567557	0.566909	-0.566909	-0.000648	0.000648	1.083687
15	0.566909	0.567276	-0.567276	0.000367	0.000367	1.077258
16	0.567276	0.567068	-0.567068	-0.000208	0.000208	1.071700
17	0.567068	0.567186	-0.567186	0.000118	0.000118	1.066912
18	0.567186	0.567119	-0.567119	-0.000067	0.000067	1.062711
19	0.567119	0.567157	-0.567157	0.000038	0.000038	1.059013

The root of the equation is 0.567157

5. Gauss Elimination Method

```
/*Program to find the solution of linear equations using gauss elimination method*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
void Gauss_Elimination();   //function prototype
int main(void)
{
        Gauss_Elimination();  //calling function inside main function
}
void Gauss_Elimination()
{       int n,i,j,k,p;    //variable decleration
        FILE *fp;
        fp=fopen("5.txt","w");
        fprintf(fp,"   GAUSSIAN ELIMINATION FOR SOLVING LINEAR EQUATIONS.\n\n");
        printf("Enter number of variables:");
        scanf("%d",&n);                     //size of matrix
        float a[n+1][n+2],m[n][n+2],sum,big,temp,x[n+1];     //variable decleration
        big=0;
        for(i=1;i<=n;i++){
                x[i] = 0;
        }
        fprintf(fp,"Augmented matrix formed is:\n");
        printf("Enter the augmented matrix.\n");
        for(i=1;i<=n;i++){
                for(j=1;j<=n+1;j++){
                        scanf("%f",&a[i][j]);     //value giving for matrix
                        fprintf(fp,"%f\t",a[i][j]);
                }
                fprintf(fp,"\n");
        }
        for(k=1;k<n;k++)
        {
                big = 0;
                for(i=k;i<n;i++)              //partial pivoting
                {
                        if(fabs(a[i][k])>big)
                        {
                                big = fabs(a[i][k]);
                                p = i;
                        }
                }
                for(j=k;j<=n+1;j++)        //swapping values
                {
                        temp = a[k][j];
                        a[k][j] = a[p][j];
                        a[p][j] = temp;
                }
                for(i = k+1;i<=n;i++)
                {   //gauss elimination
                        m[i][k] = a[i][k] / a[k][k];
                        for(j=k;j<=n+1;j++)
                        {
                                a[i][j] = a[i][j] - m[i][k]*a[k][j];
                        }
```

```
        }
    }
    for(i =n;i>=1;i--)
    {
        sum = 0;
        for(j = n;j>=i;j--)                 //back substitution
        {
            sum = sum + a[i][j]*x[j];
        }
        x[i] = (a[i][n+1] -sum)/a[i][i];     //solving and finding values
    }
    for(i = 1;i<=n;i++)
    {
        printf("x%d = %f",i,x[i]);   //print the result
        fprintf(fp,"x%d = %f",i,x[i]);
        printf("\n");
        fprintf(fp,"\n");
    }
    for(i=1;i<=n;i++){
        for(j=1;j<=n+1;j++){
            printf("%f ",a[i][j]);    //print the upper tringualr matrix after gauss
elimination
            fprintf(fp,"%f ",a[i][j]);
        }
        printf("\n");
        fprintf(fp,"\n");
    }
    fclose(fp);
}
```

Output

```
   GAUSSIAN ELIMINATION FOR SOLVING LINEAR EQUATIONS.

Augmented matrix formed is:
3.000000        1.000000        2.000000        3.000000
2.000000        -3.000000       -1.000000       -3.000000
1.000000        2.000000        1.000000        4.000000

x1 = 1.000000
x2 = 2.000000
x3 = -1.000000

Upper Triangular Matrix Formed:
3.000000 1.000000 2.000000 3.000000
0.000000 -3.666667 -2.333333 -5.000000
0.000000 0.000000 -0.727273 0.727273
```

6. Gauss Central Forward Difference Method

```
/*
  Program to implement GAUSS' FORWARD INTERPOLATION FORMULA.
       --------------------------------
*/
#include<stdio.h>
#include<math.h>

void main()
{
        FILE *fp;
        fp=fopen("6.txt","w");
        fprintf(fp,"GAUSS CENTRAL FORWARD INTERPOLATION.\n\n");
        int n,i,j;
        float y1,y2,y3,y4,x,nr,dr,y=0,h,p;
        float diff[20][20];

        // Input section.
        printf("\n\n Enter the no. of terms: ");
        scanf("%d",&n);
        float ax[n],ay[n];

        // Input Sequel for array X and Y
        printf("\n\n Enter the value in the form of x f(x)\n ");
        // Input loop for X and Y
        for(i=0;i<n;i++)
          scanf("%f%f",&ax[i],&ay[i]);

        // Inputting the required value query
        printf("\n\n Enter the required value of x.\n ");
        scanf("%f",&x);

        // Calculation and processing section.
        h=ax[1]-ax[0];
        for(i=0;i<n-1;i++)
          diff[i][1]=ay[i+1]-ay[i];

        for(j=2;j<=4;j++)
          for(i=0;i<n-j;i++)
            diff[i][j]=diff[i+1][j-1]-diff[i][j-1];

        fprintf(fp,"x\t\t\tDifference Table\n");

        for(i=0;i<n;i++)      //difference table printed
        {
                fprintf(fp,"%f|\t",ax[i]);
                fprintf(fp,"%f\t",ay[i]);
                for(j=1;j<n-i;j++)
                        fprintf(fp,"%f\t",diff[i][j]);
                fprintf(fp,"\n");
        }
        i=0;
        //Implementation of the method performed here after
        do
```

```
    {
        i++;
    }while(ax[i]<x);
    i--;
    p=(x-ax[i])/h;
    y1=p*diff[i][1];
    y2=p*(p-1)*diff[i-1][2]/2;
    y3=(p+1)*p*(p-1)*diff[i-2][3]/6;
    y4=(p+1)*p*(p-1)*(p-2)*diff[i-3][4]/24;

    // Taking sum
    y=ay[i]+y1+y2+y3+y4;

    // Outut Section
    printf("\n When x = %6.4f , y = %6.8f\n",x,y);
    fprintf(fp,"\n\nResult:\tf(%f)=%f",x,y);   //final result printed in the file
    fclose(fp);
}
```

Output

```
GAUSS CENTRAL FORWARD INTERPOLATION.

x                        Difference Table
1.000000|                1.000000        1.000000        0.000000
2.000000|                2.000000        1.000000
3.000000|                3.000000


Result: f(2.530000)=2.530000
```

7. Gauss Jordan Method

```
/*Program to solve linear equations using gauss jordan method*/
#include<stdio.h>
#include<math.h>
void pivot(double a[100][100], int, int);
int main()
{
	FILE *fp;
	fp=fopen("7.txt","w");
	int i,j,k,n,maxpos,l,ch,loop1,loop2;
	double m,s,temp,max;
	printf("Enter Value of n:\n");	//number of variables is scanned
	scanf("%d", &n);
	printf("Enter the augmented matrix:\n"); //augmented matrix is taken from the user
	fprintf(fp,"The augmented matrix is:\n");
	double a[n][n+1], x[n];
	for(i=0; i<n; i++)
	{
		for(j=0; j<=n; j++){
			scanf("%lf", &a[i][j]);
			fprintf(fp,"%lf", &a[i][j]);
		}
		fprintf(fp,"\n");
	}
	for(k=0; k<n; k++)	//loop for the implementation of the method
	{
		for(i=0; i<n; i++){
			if(i==k)
				continue;
			m=a[i][k]/a[k][k];
			for(j=k; j<n+1; j++)
				a[i][j]= a[i][j]-(m*a[k][j]);
		}
	}
	printf("\n");
	fprintf(fp,"\n");
	for(i=0; i<n; i++)
	{
		printf("\n");
		fprintf(fp,"\n");
		for(j=0; j<=n; j++)
		{
			printf("%lf   ", a[i][j]);
			fprintf(fp,"%lf   ", a[i][j]);
		}
	}
	for(i=0; i<n; i++)
	{
		x[i]=a[i][n]/a[i][i];
	}
	printf("\n\nSolution to the Equation:\n");  //printing the results
	fprintf(fp,"\n\nSolution to the Equation:\n");
	for(i=0; i<n; i++){
		printf("%lf\n", x[i]);
		fprintf(fp,"%lf\n",x[i]);
	}
	fclose(fp);
	return 0;
}
```

Output

```
The augmented matrix is:
0.0000000.0000000.0000000.000000
0.0000000.0000000.0000000.000000
0.0000000.0000000.0000000.000000


3.000000    0.000000    0.000000    3.000000
0.000000    -3.666667    0.000000    -7.333333
0.000000    0.000000    -0.727273    0.727273

Solution to the Equation:
1.000000
2.000000
-1.000000
```

8. Gauss Jordan Method(for finding inverse of a matrix)

```
/*Program to find inverse of a given matrix*/
#include<stdio.h>
int main()  //main begins here
{
            FILE *fp;
            fp=fopen("8.txt","w");
            fprintf(fp,"   GAUSS JORDAN METHOD FOR FINDING INVERSE OF A MATRIX.\n\n");
    int iter1, iter2, k, order;
    printf("Enter order of matrix: ");
    scanf("%d", &order);                                          //taking the input for the oder of the matrix
    float matrix[order][order*2], ratio,a;
    printf("Enter the matrix: \n");
    fprintf(fp,"The matrix is:\n\n");
    for(iter1 = 0; iter1 < order; iter1++)              //taking the input for the matrix
    {
        for(iter2 = 0; iter2 < order; iter2++)
        {
            scanf("%f", &matrix[iter1][iter2]);
            fprintf(fp,"%f\t",matrix[iter1][iter2]);
        }
        fprintf(fp,"\n");
    }
    for(iter1 = 0; iter1 < order; iter1++)                            //making an identity matrix
    {
        for(iter2 = order; iter2 < 2*order; iter2++)
        {
            if(iter1==(iter2%order))
                matrix[iter1][iter2] = 1.0;
            else
                matrix[iter1][iter2] = 0.0;
        }
    }
   for(iter2=0; iter2<order; iter2++)
 {
  int temp=iter2;

 /* finding maximum jth column element in last (dimension-j) rows */

  for(iter1=iter2+1; iter1<order; iter1++)
if(matrix[iter1][iter2]>matrix[temp][iter2])
                        temp=iter1;
 /* swapping row which has maximum jth column element */

  if(temp!=iter2)
            for(k=0; k<2*order; k++)
            {
            float temporary=matrix[iter2][k] ;
            matrix[iter2][k]=matrix[temp][k] ;
            matrix[temp][k]=temporary ;
            }

/* performing row operations to form required identity matrix out of the input matrix */

  for(iter1=0; iter1<order; iter1++)
            if(iter1!=iter2)
            {
            ratio=matrix[iter1][iter2];
            for(k=0; k<2*order; k++)
              matrix[iter1][k]-=(matrix[iter2][k]/matrix[iter2][iter2])*ratio ;
            }
            else
            {
            ratio=matrix[iter1][iter2];
            for(k=0; k<2*order; k++)
              matrix[iter1][k]/=ratio ;
            }
```

```
}

    fprintf(fp,"\nThe inverse of the given matrix is:\n");



printf("The inverse matrix is: \n");     //printing the final result
    for(iter1 = 0; iter1 < order; iter1++){
        for(iter2 = order; iter2 < 2*order; iter2++)
        {
            printf("%f\t", matrix[iter1][iter2]);
            fprintf(fp,"%f\t",matrix[iter1][iter2]);
        }
        printf("\n");
        fprintf(fp,"\n");
    }
    fclose(fp);
    return 0;
}
```

Output

```
   GAUSS JORDAN METHOD FOR FINDING INVERSE OF A MATRIX.

The matrix is:

2.000000        1.000000        1.000000
3.000000        2.000000        3.000000
1.000000        4.000000        9.000000

The inverse of the given matrix is:
-3.000003       2.500002        -0.500000
12.000009       -8.500007       1.500001
-5.000003       3.500003        -0.500000
```

9. Gauss Seidel Method

```
/*Program for implementing Gauss seidel method*/
#include<stdio.h>
#include<math.h>
int main(void)  //main begins here
{
	FILE *fp;
	fp=fopen("9.txt","w");
	fprintf(fp,"   GAUSS SEIDEL METHOD.\n\n");
	int order,i,j,k;
	printf("Enter the number of variables\n");
	scanf("%d",&order);                          //scans the number of variables
	float A[order][order+1],x[order],x_new[order],error;
	for(i=0;i<order;i++)                    //seeding the initial result
	{
		x[i]=0;x_new[i]=0;
	}
	printf("Enter the augmented matrix\n");
	fprintf(fp,"The augmented matrix is:-\n");
	for(i=0;i<order;i++){                   //augmented matrix is scanned from the user row-wise
		for(j=0;j<order+1;j++){
			scanf("%f",&A[i][j]);
			fprintf(fp,"%f\t",A[i][j]);
		}fprintf(fp,"\n");
		}
	do                                                          //loop for implementing the method
	{
		for(i=0;i<order;i++)
		{
			x_new[i]=A[i][order]; //updating the value of x begins here
			for(j=0;j<i;j++)
			{
				x_new[i]-=A[i][j]*x_new[j];
			}
			for(j=i+1;j<order;j++)
			{
				x_new[i]-=A[i][j]*x[j];
			}
			x_new[i]/=A[i][i];
		}
		error=fabs(x_new[0]-x[0]); //error calculation
		for(i=0;i<order;i++)
		{
			if(error>fabs(x_new[i]-x[i])) error=fabs(x_new[i]-x[i]);
		}
		for(i=0;i<order;i++) x[i]=x_new[i];
	}while(error>=0.00001);       //minimum error calculation
	printf("The solution set is:\n");  //printing the results
	fprintf(fp,"The solution set is:\n");
	for(i=0;i<order;i++){
		printf("%0.5f\t",x[i]);
		fprintf(fp,"%0.5f\t",x[i]);
	}
	printf("\n");
	fclose(fp);
}
```

Output

```
   GAUSS SEIDEL METHOD.

The augmented matrix is:-
10.000000       -2.000000       -1.000000       -1.000000       3.000000
-2.000000       10.000000       -1.000000       -1.000000       15.000000
-1.000000       -1.000000       10.000000       -2.000000       27.000000
-1.000000       -1.000000       -2.000000       10.000000       -9.000000
The solution set is:
1.00000 2.00000 3.00000 -0.00000
```

10. Jacobi's Method

```
/*Program for implementing Jacobi's method*/
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
int main(void)  //main begins here
{
        FILE *fp;
        fp=fopen("10.txt","w");
        fprintf(fp,"   JACOBI METHOD FOR SOLVING EQUATIONS.\n\n");
        int order,i,j,k;
        printf("Enter the number of variables\n");
        scanf("%d",&order);                         //scans the number of variables
        float A[order][order+1],x[order],x_new[order],error,max;
        for(i=0;i<order;i++)                        //seeding the initial result
        {
                x[i]=0;x_new[i]=0;
        }
        printf("Enter the augmented matrix\n");
        fprintf(fp,"The augmented matrix is:-\n");
        for(i=0;i<order;i++){                       //augmented matrix is scanned from the user row-wise
                for(j=0;j<order+1;j++){
                        scanf("%f",&A[i][j]);
                        fprintf(fp,"%f\t",A[i][j]);
                }fprintf(fp,"\n");
        }
        do                                          //loop for implementing the method
        {
                for(i=0;i<order;i++)
                {
                        x_new[i]=A[i][order]; //updating the value of x begins here
                        for(j=0;j<order;j++)
                        {
                                if(j!=i) x_new[i]-=A[i][j]*x[j];
                        }
                        x_new[i]/=A[i][i];
                }
                max=fabs(x_new[0]-x[0]); //error calculation
                for(i=0;i<order;i++)
                {
                        error=fabs(x_new[i]-x[i]);
                        if(error>max) max=error;
                }
                for(i=0;i<order;i++) x[i]=x_new[i];
        }while(max>=0.000005);        //minimum error calculation
        printf("The solution set is:\n");  //printing the results
        fprintf(fp,"The solution set is:\n");
        for(i=0;i<order;i++){
                printf("%0.5f\t",x[i]);
                fprintf(fp,"%0.5f\t",x[i]);
        }
        printf("\n");
        fclose(fp);
}
```

Output

```
    JACOBI METHOD FOR SOLVING EQUATIONS.

The augmented matrix is:-
10.000000       -2.000000       -1.000000       -1.000000       3.000000
-2.000000       10.000000       -1.000000       -1.000000       15.000000
-1.000000       -1.000000       10.000000       -2.000000       27.000000
-1.000000       -1.000000       -2.000000       10.000000       -9.000000
The solution set is:
1.00000 2.00000 3.00000 -0.00000
```

11. Finding Eigen values of a matrix using Jacobi's Method

```
/*Program to find eigen value using jacobi's method*/
#include<stdio.h>
#include<math.h>
int main(void)
{
        FILE *fp;
        fp=fopen("11.txt","w");
        fprintf(fp,"  Finding Eigen value using Jacobi's method\n\n");
        int i,j,n,r,s;
        printf("Enter the order of the matrix\n");
        scanf("%d",&n);
        float x[n][n],max=0,sum=0;
        printf("Enter the symmetrix of order %d\n",n);
        fprintf(fp,"The given matrix is:\n");
        for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                {
                        scanf("%f",&x[i][j]);
                        fprintf(fp,"%f\t",x[i][j]);
                        if(j>i)
                        {
                                sum+=x[i][j];
                                if(max<fabs(x[i][j]))
                                {
                                        max=x[i][j];r=i;s=j;
                                }
                        }
                        fprintf(fp,"\n");
                }
        while(sum!=0)
        {
                float ang=atan((2*x[r][s])/(x[r][r]-x[s][s]))/2;
                for(i=0;i<n;i++)
                        for(j=0;j<n;j++)
                        {
                                if(j>i && j!=r &&j!=s)
                                {
                                        x[j][r]=x[j][r]*cos(ang)+x[j][s]*sin(ang);
                                        x[r][j]=x[j][r];
                                        x[j][s]=x[j][s]*cos(ang)-x[j][r]*sin(ang);
                                        x[s][j]=x[j][s];
                                }
                        }
                x[r][r]=x[r][r]*cos(ang)*cos(ang)+2*x[r][s]*cos(ang)*sin(ang)+x[s][s]*sin(ang)*sin(ang);
                x[s][s]=x[s][s]*cos(ang)*cos(ang)-2*x[r][s]*cos(ang)*sin(ang)+x[r][r]*sin(ang)*sin(ang);
                x[r][s]=0;x[s][r]=0;
                max=0,sum=0;
                for(i=0;i<n;i++)
                        for(j=0;j<n;j++)
                        {
                                if(j>i)
                                {
                                        sum+=x[i][j];
                                        if(max<fabs(x[i][j]))
                                        {
                                                max=x[i][j];r=i;s=j;
                                        }
                                }
                        }
                printf("%f\n",sum);
        }
        printf("The eigen values are:\n");
        fprintf(fp,"The eigen values are:\n");
        for(i=0;i<n;i++){
                printf("%f\n",x[i][i]);
                fprintf(fp,"%f\n",x[i][i]);;}}
```

Output

```
   Finding Eigen value using Jacobi's method

The given matrix is:
5.000000
0.000000
1.000000
0.000000
-2.000000
0.000000
1.000000
0.000000
5.000000
0.000000
The eigen values are:
6.000000
-2.000000
4.500000
```

12. Curve Fitting using Least Squares Method

```
/*Program for applying least square method of curve fitting*/
#include<stdio.h>
#include<math.h>
int main()  //main begins here
{
        FILE *fp;   //file pointer declared
        fp=fopen("12.txt","w");  //file opened
        fprintf(fp,"     Linear Curve fitting using least square method\n\n");
  int n,i, x[20],y[20],sumx=0,sumy=0,sumxy=0,sumx2=0;   //variables declared
  float a,b;
  printf("\n Enter the value of number of terms n:");  //scanning begins from here
  scanf("%d",&n);
  printf("\n Enter the values of x:\n");
  for(i=0;i<=n-1;i++)
  {
    scanf(" %d",&x[i]);

  }
  printf("\n Enter the values of y:");
  fprintf(fp,"Given inputs are:\n");
  for(i=0;i<=n-1;i++)
  {
    scanf("%d",&y[i]);
    fprintf(fp," f(%d)=%d\n",x[i],y[i]);
  }
  for(i=0;i<=n-1;i++)    //calculation of the function
  {
    sumx=sumx +x[i];
    sumx2=sumx2 +x[i]*x[i];
    sumy=sumy +y[i];
    sumxy=sumxy +x[i]*y[i];

  }
  a=((sumx2*sumy -sumx*sumxy)*1.0/(n*sumx2-sumx*sumx)*1.0);   //final sum calculated
  b=((n*sumxy-sumx*sumy)*1.0/(n*sumx2-sumx*sumx)*1.0);
  printf("\n\nThe line is Y=%3.3fx +%3.3f \n",b,a);   //final result being printed here
  fprintf(fp,"\n\nThe line is Y=%3.3fx +%3.3f \n",b,a);
  fclose(fp);
  return 0;
}
```

Output

```
     Linear Curve fitting using least square method

Given inputs are:
 f(2)=8
 f(4)=14
 f(6)=20
 f(8)=26


The line is Y=3.000x +2.000
```

13. LU Decomposition Method

```
/*Program to implement LU decomposition method*/
#include<stdio.h>
int main(void)  //main begins here
{
        FILE *fp;
        fp=fopen("13.txt","w");
        fprintf(fp,"   LU Decomposition method.\n\n");
        int size,i,j,p,k;
        printf("Enter the order of the matrix.\n");
        scanf("%d",&size);  //size of the matrix scanned from the terminal
        float A[size][size],L[size][size],U[size][size];
        printf("Enter the matrix.\n");  //matrix is scanned
        fprintf(fp,"A:");
        for(i=0;i<size;i++){
                for(j=0;j<size;j++){
                        scanf("%f",&A[i][j]);
                        fprintf(fp,"%f\t",A[i][j]);
                }
                fprintf(fp,"\n  ");
        }
        for(i=0;i<size;i++)  //matrices initialised
                for(j=0;j<size;j++)
                {
                        L[i][j]=0;U[i][j]=0;
                }
        for(j=0;j<size;j++)  //loop of implementation
                for(i=0;i<size;i++)
                {
                        if(i>=j)   //for lower matrix
                        {
                                L[i][j]=A[i][j];
                                for(k=0;k<=j-1;k++) L[i][j]-=L[i][k]*U[k][j];
                                if(i==j) U[i][j]=1;
                        }
                        else   //for upper matrix
                        {
                                U[i][j]=A[i][j];
                                for(k=0;k<=i-1;k++)
                                        U[i][j]-=L[i][k]*U[k][j];
                                U[i][j]/=L[i][i];
                        }
                }
        printf("\nL\n");  //printing results
        fprintf(fp,"\nL:");
        for(i=0;i<size;i++)
        {
                for(j=0;j<size;j++){
                        printf("%f\t",L[i][j]);
                        fprintf(fp,"%f\t",L[i][j]);
                }
                printf("\n");
                fprintf(fp,"\n  ");
        }
        printf("\nU\n");
        fprintf(fp,"\nU:");
        for(i=0;i<size;i++)
        {
                for(j=0;j<size;j++){
                        printf("%f\t",U[i][j]);
                        fprintf(fp,"%f\t",U[i][j]);
                }
                fprintf(fp,"\n  ");
                printf("\n");
        }
        fclose(fp);
}
```

Output

```
      LU Decomposition method.

A:2.000000      3.000000        1.000000
  1.000000      2.000000        3.000000
  3.000000      1.000000        2.000000

L:2.000000      0.000000        0.000000
  1.000000      0.500000        0.000000
  3.000000      -3.500000       18.000000

U:1.000000      1.500000        0.500000
  0.000000      1.000000        5.000000
  0.000000      0.000000        1.000000
```

14. Modified Euler's Method

```
a/*Program to apply modified euler's method for solving linear differential equation*/
#include<stdio.h>
#include<math.h>
#define fun(x,y) y-x    //function used is y'=y-x
void main()       //main begins here
{
            FILE *fp;
            fp=fopen("modified euler.txt","w");
            fprintf(fp,"   Modified euler method for solving differential equation.\n\nDE: y'=y-x\n\n");
   int i,j,c;
   float x,y,h,m,m1,m2,a,s[100],w;
            printf("  Enter the value of x0,xn,h,y0:\n");   //initial and final values of x, height of x and initial value of
y is taken as
                       input
            scanf("%f%f%f%f",&x,&a,&h,&y);
            s[0]=y;
            printf(" The respective values of x and y are\n     x  \t     y\n\n");
            fprintf(fp," The respective values of x and y are\n     x  \t     y\n\n");
            for(i=1;x<a;i++)   //method application here
   {
                     printf("  %f\t%f\n",x,y);
                     fprintf(fp,"  %f\t%f\n",x,y);
              w=100;
              m=fun(x,y);
              x= x+h;
              c=0;
              while(w>0.00001)            //convergence of y at ith iteration
              {
                 m1=fun(x,s[c]);
                 m2=(m+m1)/2;
                 s[c+1]=y+m2*h;
                 w=fabs(s[c]-s[c+1]);
                 c++;
              }
              y=s[c];
   }
   printf("  %f\t%f\n",x,y);
            fprintf(fp,"  %f\t%f\n",x,y);   //printing final results
            fclose(fp);
}
```

Output

```
   Modified euler method for solving differential equation.

DE: y'=y-x

 The respective values of x and y are
     x               y

  0.000000      2.000000
  0.050000      2.101282
  0.100000      2.205194
  0.150000      2.311871
  0.200000      2.421453
```

15. Multi Point Iteration Method

```
/*Program to apply multi point iteration method to find the root of the equation e^x-2x-1*/
#include<stdio.h>
#include<math.h>

double f(double x)                              //function to return f(x)
{
            double y;
            y=exp(x)-2*x-1;
            return y;
}

double df(double x)
            //function to return f'(x)
{
            double y;
            y=exp(x)-2;
            return y;
}
double soln(double r)
            //fuunction to return root of the equation using multipoint iteration method
{
            int i=1;
            double ri,ri_s,pre_err,err,order=0;
            FILE *fp;
                        //opening the file
            fp=fopen("15.txt","w");
            fprintf(fp,"  MULTI POINT ITERATION METHOD.\n\n");
            pre_err=0;
            err=0;
            ri_s=r-(f(r)/(2*df(r)));
            //calculating ri+1*
            ri=r-(f(r)/df(ri_s));
                  //calculating ri+1
            pre_err=fabs(r-ri);
            fprintf(fp,"i       x           f(x)          a_error         order");
            fprintf(fp,"\n%d         %lf       %lf       %lf       %lf",i,r,f(r),pre_err,order);
            while(fabs(r-ri)>=0.0001)
                  //doing the iteration till error becomes less than 10^-4
            {
                  r=ri;i++;
                  ri_s=r-(f(r)/(2*df(r)));
            //calculating ri+1*
                  ri=r-f(r)/df(ri_s);
                  //calculating ri+1
                  err=fabs(r-ri);
                        //calculating error
                  order=log(err)/log(pre_err);
            //calculating order of convergence
                  fprintf(fp,"\n%d    %lf       %lf       %lf       %lf",i,r,f(r),err,order);//printing results
                  pre_err=err;
                        //updating previous error
            }
            fprintf(fp,"\n\nThe root of the equaion is %lf\n",r);            //printing the final result
            fclose(fp);
                        //closing the file
            return r;
                        //returning the root
}
/*main begins here*/
void main()
{
            double r;
            printf("enter any starting point:"); //taking input from the user
            scanf("%lf",&r);
            printf("\nroot=%lf",soln(r));        //printing the result by calling the function
}
```

Output

```
  MULTI POINT ITERATION METHOD.

i          x            f(x)          a_error        order
1       2.000000      2.389056      0.609448      0.000000
2       1.390552      0.235963      0.131908      4.090557
3       1.258644      0.003356      0.002213      3.018085
4       1.256431      0.000000      0.000000      2.964229

The root of the equaion is 1.256431
```

16. Newton's Backward Difference Interpolation Method

```
/*Program to implement backward interpolation technique*/
#include<stdio.h>
#include<math.h>

int fact(int n)    //function to find the factorial of n
{
        int f=1,i;
        for(i=2;i<=n;i++)
            f*=i;
        return f;
}
int main(void)     //main begins
{
        FILE *fp;
        fp=fopen("16.txt","w");   //file opened to print the data in it
        printf("Enter the number of inputs.\n");
        int n,i,j;
        scanf("%d",&n);
        printf("Enter the values of the function as x,f(x)\n");  //inputs taken
        float x[n];
        float y[n][n];
        for(i=0;i<n;i++)
            scanf("%f%f",&x[i],&y[i][0]);
        for(i=1;i<n;i++)      //difference table created
            for(j=n-1;j>=i;j--)
                    y[j][i]=y[j][i-1]-y[j-1][i-1];
        fprintf(fp,"    x\t\t       Difference table.\n");
        for(i=0;i<n;i++)      //difference table printed
        {
            fprintf(fp,"%f\t",x[i]);
            for(j=0;j<=i;j++)
                    fprintf(fp,"%f\t",y[i][j]);
            fprintf(fp,"\n");
        }
        float val;     //interpolated value taken from the user
        printf("Enter the value to interpolat at:(between %f and %f)\n",x[0],x[n-1]);
        scanf("%f",&val);
        float sum=y[n-1][0];
        float u=(val-x[n-1])/(x[1]-x[0]);   //height is found
        float temp=u;
        for(i=1;i<n;i++)
        {
            sum+=((u*y[n-1][i])/fact(i));    //formula of the technique implemented here
            u*=(temp+i);
        }
        printf("Value at %f is %f\n",val,sum);   //final result printed in the terminal and the file
        fprintf(fp,"\n\n\nf(%f)=%f\n",val,sum);
}
```

Output

```
    x                    Difference table.
2.000000        6.000000
4.000000        12.000000       6.000000
6.000000        18.000000       6.000000        0.000000
8.000000        24.000000       6.000000        0.000000
0.000000


f(5.000000)=15.000000
```

17. Newton's Forward Difference Interpolation Method

```
/*Program to implement forward interpolation technique*/
#include<stdio.h>
#include<math.h>

int fact(int n)    //function to find the factorial of n
{
        int f=1,i;
        for(i=2;i<=n;i++)
            f*=i;
        return f;
}
int main(void)    //main begins
{
        FILE *fp;
        fp=fopen("17.txt","w");   //file opened to print the data in it
        int n=4,i,j;
        printf("Enter the values of the function as x,f(x)\n");  //inputs taken
        float x[4];
        float y[4][4];
        for(i=0;i<4;i++)
            scanf("%f%f",&x[i],&y[i][0]);
        for(i=1;i<n;i++)      //difference table created
            for(j=0;j<n-i;j++)
                y[j][i]=y[j+1][i-1]-y[j][i-1];
        fprintf(fp,"    x\t\t   f(x)\t\t  df(x)\t\td1f(x)  \td2f(x)\n");
        for(i=0;i<n;i++)      //difference table printed
        {
            fprintf(fp,"%f\t",x[i]);
            for(j=0;j<n-i;j++)
                fprintf(fp,"%f\t",y[i][j]);
            fprintf(fp,"\n");
        }
        float val;     //interpolated value taken from the user
        printf("Enter the value to interpolat at:(between %f and %f)\n",x[0],x[3]);
        scanf("%f",&val);
        float sum=y[0][0];
        float u=(val-x[0])/(x[1]-x[0]);   //height is found
        for(i=1;i<n;i++)
        {
            sum+=(u*y[0][i])/fact(i);    //formula of the technique implemented here
            u*=(u-1);
        }
        printf("Value at %f is %f\n",val,sum);   //final result printed in the terminal and the file
        fprintf(fp,"\n\n\nf(%f)=%f\n",val,sum);
}
```

Output

```
    x             f(x)          df(x)         d1f(x)        d2f(x)
2.000000      5.000000      5.000000      0.000000      0.000000
4.000000      10.000000     5.000000      0.000000
6.000000      15.000000     5.000000
8.000000      20.000000


f(5.000000)=12.500000
```

18. Newton' Forward Difference Formula for differentiation

```
/*Program for applying newton forward difference formula for differentiation*/
#include<stdio.h>
#include<math.h>
int fact(int n)     //function to find the factorial of n
{
        int f=1,i;
        for(i=2;i<=n;i++)
                f*=i;
        return f;
}
int main()
{
        FILE *fp;       //file pointer declared
        fp=fopen("18.txt","w");
        int n,i,j;
        printf("Enter the number of terms.\n");
        scanf("%d",&n);
        float x[n],y[n][n];     //variables declared
        for(i=0;i<n;i++)        //inputs taken from the user
                scanf("%f%f",&x[i],&y[i][0]);
        for(i=1;i<n;i++)        //difference table created
                for(j=0;j<n-i;j++)
                        y[j][i]=y[j+1][i-1]-y[j][i-1];
        fprintf(fp,"    x\t\t   f(x)\t\t  df(x)\t\td1f(x)  \td2f(x)\n");
        for(i=0;i<n;i++)        //difference table printed
        {
                fprintf(fp,"%f\t",x[i]);
                for(j=0;j<n-i;j++)
                        fprintf(fp,"%f\t",y[i][j]);
                fprintf(fp,"\n");
        }
        float sum=0;
        for(i=0;i<n-1;i++)      //formula for the method
                sum+=pow(-1,i)*y[0][i+1]/fact(i+1);
        sum/=(x[1]-x[0]);       //printing the final result
        printf("The f'(%f) = %f\n",x[0],sum);
        fprintf(fp,"\n\n\tThus, f'(%f) = %f\n",x[0],sum);
        fclose(fp);     //file pointer closed
}
```

Output

```
    x              f(x)          df(x)         d1f(x)        d2f(x)
2.000000       4.000000       12.000000      8.000000      0.000000
4.000000       16.000000      20.000000      8.000000
6.000000       36.000000      28.000000
8.000000       64.000000

        Thus, f'(2.000000) = 4.000000
```

19. Newton Raphson Method

```
/*Program for describing newton raphson method
f(x)= x^3-x-3*/
#include<stdio.h>
#include<math.h>
float func(float x)                         //function for finding the value of f(x)
{
            return (x*x*x-2*x-5);
}
float dif_func(float x)                     //function for finding the value of f'(x)
{
            return (3*x*x -2);
}
float dDifFunc(float x)                     //function for finding the value of f"(x)
{
            return (6*x);
}
float ri(float r)                           //function for finding the value of r_next
{
            return (r-(func(r)/dif_func(r)));
}
float gf(float x)                           //function for checking the convergence of the function
{
            float res;
            res=fabs(func(x)*dDifFunc(x)/((dif_func(x)*dif_func(x))));
            return res;
}
int main()                                  //main function begins here
{
            int iter=1;
            float r_i,r_i1,abs_error,prev_error=0,cond,conv;

            printf("Enter the initial point:\n");           //input taken from user
            scanf("%f",&r_i);

            FILE *fp;
            fp=fopen("19.txt","a");                   //opening of file in write mode
            fprintf(fp,"   NEWTON RAPHSON METHOD\n\n");
            fprintf(fp,"Sl.no.\tr\t\t\tcondition\tr_next\t\terror\t\torder of convergence\n");
            do                                        //loop for the method
            {
            cond=gf(r_i);
            if(cond>1)                                //checking the divergency of the function
            {
                printf("function diverges!!\nProgram failed!!!!\n");  // if diverges,program fails
                fprintf(fp,"Function diverges!!\n\n\n");fclose(fp);return 0;
                break;
            }
            r_i1=ri(r_i);                             //updating the value of r_next
            abs_error=fabs(r_i1-r_i);
            if(prev_error!=0) conv=log(abs_error)/log(prev_error);    //finding the order of convergence
            prev_error=abs_error;
            if(iter==1)
            fprintf(fp,"%d\t\t%f\t%f\t%f\t%f\tNA\n\n",iter,r_i,cond,r_i1,abs_error);
            else
                fprintf(fp,"%d\t\t%f\t%f\t%f\t%f\t%f\n\n",iter,r_i,cond,r_i1,abs_error,conv);
            iter++;
            r_i=r_i1;
            }while(abs_error>=0.00001);                 //checking precision
            printf("root of the function=%f",r_i1);        //printing result
            fprintf(fp,"root of the function=%f",r_i1);
            fclose(fp);
            return 0;
}
```

Output

```
   NEWTON RAPHSON METHOD

Sl.no.          r          condition       r_next          error           order of convergence
1               0.000000        0.000000        -2.500000       2.500000        NA

2               -2.500000       0.835375        -1.567164       0.932836        -0.075878

Function diverges!!

   NEWTON RAPHSON METHOD

Sl.no.          r          condition       r_next          error           order of convergence
1               2.000000        0.120000        2.100000        0.100000        NA

2               2.100000        0.006094        2.094568        0.005432        2.265048

3               2.094568        0.000019        2.094552        0.000016        2.112012

4               2.094552        0.000000        2.094552        0.000000        1.#INF00

root of the function=2.094552
```

20. Newton Raphson method for solving non-linear simultaneous equations

```
/*Program to implement Newton raphson mathod for solving non linear simultaneous equations*/
#include<stdio.h>
#include<math.h>
#define f(x,y) x*x-y*y-3   //first equation as f(x,y)=0
#define g(x,y) x*x+y*y-13  //second equation as g(x,y)=0
#define max(x,y) x>y?x:y   //max function
//main begins
int main(void)
{
                FILE *fp;
                fp=fopen("20.txt","w");
                fprintf(fp,"     Newton Raphson method for solving non-linear simultaneous equations\n\n");
                fprintf(fp,"Functions:\nf(x,y)=x^2-y^2-3\ng(x,y)=x^2+y^2-13\n\n");
                int iter=0;
                float x,y,h,k,pfx,pfy,pgx,pgy,f0,g0,jac;
                x=sqrt(6.5); //initialisation of x,y for making g(x,y)=0;
                y=x;
                fprintf(fp,"Sl\t\th\t\t\tk\t\t   x\t\t\ty\t\tjacobian term\n");
                do
                {
                    iter++;  //for printing the iterations
                    f0=f(x,y); //function values
                    g0=g(x,y);
                    pfx=2*x;  //partial derivatives of functions w.r.t variable
                    pfy=-2*y;
                    pgx=2*x;
                    pgy=2*y;
                    jac=pfx*pgy-pfy*pgx;  //value of jacobian
                    /*h and k are approximation to the value of x and y calculated by solving the equations
                                 h(pfx)+k(pfy)=f0 and
                                 h(pgx)+k(pgy)=g0*/
                    h=(pfy*g0-pgy*f0)/(pfx*pgy-pfy*pgx); //extending limits of x
                    k=(pgx*f0-pfx*g0)/(pfx*pgy-pfy*pgx); //extending limits of y
                    x=x+h;
                    y=y+k;
                    fprintf(fp,"%d\t%f\t%f\t%f\t%f\t%f\n",iter,h,k,x,y,jac);  //printing value of x and y in
each iteration
                }while((max(f(x,y),g(x,y))>=0.0005)&&jac!=0);   //loop shall run till functions tend to 0
                printf("Results:\nx=%f  y=%f  \n",x,y);   //printing final results
                fprintf(fp,"\n\nResults:\nx=%f y=%f",x,y);
                fclose(fp);
}
```

Output

```
     Newton Raphson method for solving non-linear simultaneous equations

Functions:
f(x,y)=x^2-y^2-3
g(x,y)=x^2+y^2-13

Sl        h            k            x           y           jacobian term
1       0.294174     -0.294174    2.843684    2.255336    52.000000
2       -0.015216    -0.019185    2.828468    2.236150    51.307693
3       -0.000041    -0.000082    2.828427    2.236068    50.599037


Results:
x=2.828427 y=2.236068
```

21. Power Method

```
//Program to find the eigen value of a given matrix
#include <stdio.h>
#include <math.h>
int main()
{
        FILE *fp;
        int i,j,order,iter=0;
        printf("Enter the order of matrix:");
        scanf("%d",&order);                    //scans the order of the matrix
        fp=fopen("21.txt","w");
        float A[order][order],x[order],z[order],e[order],zmax,emax;     //declaring variable
        printf("Enter the matrix\n");
        fprintf(fp,"The given matrix is\n");
        for(i=0; i<order; i++)                                  //scanning matrix
        {
                for(j=0; j<order; j++)
        {
                scanf("%f",&A[i][j]);                  //scannning elements
                fprintf(fp,"%f\t",A[i][j]);         //print matrix in file
        }
        fprintf(fp,"\n");
        }
        fprintf(fp,"\n\n");
        for(i=0; i<order; i++)                 //initialising first column vector
        {
        x[i]=1;
        }
        fprintf(fp,"Sl.No\tX1\t\tx2\t\tx3\t\tC\t\terrorMax\n");  //print column headers
        do
        {
                iter++;                                         // iteration
        for(i=0; i<order; i++)                          //matrix multiplication
        {
                z[i]=0;
                for(j=0; j<order; j++)
                {
                z[i]=z[i]+A[i][j]*x[j];
                }
        }
        zmax=fabs(z[0]);                                //calculating max value
        for(i=1; i<order; i++)
        {
                if((fabs(z[i]))>zmax)
                zmax=fabs(z[i]);
        }
        for(i=0; i<order; i++)                 //divide the column  matrix by c
        {
                z[i]=z[i]/zmax;
        }
        for(i=0; i<order; i++)                          //calculate error
        {
                e[i]=0;
                e[i]=fabs((fabs(z[i]))-(fabs(x[i])));
        }
        emax=e[0];
        for(i=1; i<order; i++)
        {
                if(e[i]>emax)
        emax=e[i];
        }
        fprintf(fp,"%d\t%9.6f\t%9.6f\t%9.6f\t%9.6f\t%9.6f\n",iter,x[0],x[1],x[2],zmax,emax);
                                        //printing iterations in files
        for(i=0; i<order; i++)                 //reinitialising matrix
        {
                x[i]=z[i];
        }
        }while(emax>0.0001);
        printf("The required eigen value is %f\n\n",zmax);
```

```
printf("The required eigen vector is :\n");
fprintf(fp,"\nThe required eigen value is %f\n\n",zmax);            //printing results
fprintf(fp,"The required eigen vector is :\n");
for(i=0; i<order; i++)
{
printf("%f\n",z[i]);
fprintf(fp,"%f\n",z[i]);
}
return 0;
fclose(fp);
}
```

Output

```
The given matrix is
5.000000        0.000000        1.000000
0.000000        -2.000000       0.000000
1.000000        0.000000        5.000000


Sl.No        X1              x2              x3              C               errorMax
1        1.000000        1.000000        1.000000        6.000000        0.666667
2        1.000000       -0.333333        1.000000        6.000000        0.222222
3        1.000000        0.111111        1.000000        6.000000        0.074074
4        1.000000       -0.037037        1.000000        6.000000        0.024691
5        1.000000        0.012346        1.000000        6.000000        0.008230
6        1.000000       -0.004115        1.000000        6.000000        0.002743
7        1.000000        0.001372        1.000000        6.000000        0.000914
8        1.000000       -0.000457        1.000000        6.000000        0.000305
9        1.000000        0.000152        1.000000        6.000000        0.000102
10       1.000000       -0.000051        1.000000        6.000000        0.000034

The required eigen value is 6.000000

The required eigen vector is :
1.000000
0.000017
1.000000
```

22. Regula Falsi Method

```
/*Program for implementing regula falsi method*/
#include<stdio.h>
#include<math.h>
float func(float x)                                   //function to find the value of f(x)
{
    float res;
    res= exp(-x)-x;
    return res;
}
int main(void)                                        //main begins
{
   FILE *fp;
   fp=fopen("22.txt","w");                            //file opens
   int iter=1;
   float a,b,c,f_a,f_b,f_c,abs_error,multi,prev_c,prev_error=0,conv;
   do                                                 //taking input from user
   {
   printf("Enter lower and upper bound of x\n");
   scanf("%f%f",&a,&b);
   if(func(a)*func(b)>=0)
   printf("ERROR!!! Enter again!\n");
   }while(func(a)*func(b)>=0);
   c=a;                                               //for initialising a prev_c within range
   fprintf(fp,"SL.no\t\ta\t\t\tb\t\t\tc\t\t\tf(c)\t\terror\torder of convergence\n");
   do                                                 //loop for finding the root of the function
   {
      prev_c=c;
      f_a=func(a);                                    //calling the function
      f_b=func(b);
      c=((a*f_b)-(b*f_a))/(f_b-f_a);
      f_c=func(c);
      multi=f_c*f_a;
      abs_error=fabs(prev_c-c);

      if(prev_error!=0){
      conv = log(abs_error)/log(prev_error);}
      prev_error=abs_error;
      fprintf(fp,"%d\t\t%f\t%f\t%f\t%f\t%f\t%f\n",iter,a,b,c,f_c,abs_error,conv);
      if(fabs(f_c)<=0.000005)                         //checking for required precision
          break;
      else if(multi<0) b=c;
      else a=c;
      iter++;
   }while(fabs(prev_c-c)>=0.000005);                  //checking for minimal error
   fprintf(fp,"\n\nroot of the function = %f",c);     //printing result in file
   fclose(fp);
   printf("res = %f",c);                              //printing result in terminal
}
```

Output

SL.no	a	b	c	f(c)	error	order of convergence
1	0.000000	2.000000	0.698162	-0.200663	0.698162	0.000000
2	0.000000	0.698162	0.581480	-0.022410	0.116682	5.979074
3	0.000000	0.581480	0.568735	-0.002494	0.012745	2.030708
4	0.000000	0.568735	0.567320	-0.000277	0.001415	1.503906
5	0.000000	0.567320	0.567163	-0.000031	0.000157	1.334723
6	0.000000	0.567163	0.567145	-0.000003	0.000017	1.251041

```
root of the function = 0.567145
```

23. Runge Kutta Method

```
/*Program to apply runge kutta method for solving linear differential equations*/
#include<stdio.h>
#include<math.h>
#define f(x,y) y-x   //function used :- y'=y-x
int main()
{
    FILE *fp;
    fp=fopen("23.txt","w");                                //file  opened in writing mode
    fprintf(fp,"  Second Order runge kutta method for solving differential equation.\n\nDE: y'=y-x\n\n");
    float x0,y0,m1,m2,m3,m4,m,y,x,h,xn;
    printf("Enter the value of x0,xn,h,y0:\n");
   //initial and final values of x, height of x and initial value of y are taken as input
    scanf("%f %f %f %f",&x0,&xn,&h,&y0);
    x=x0;
    y=y0;
    printf("\n\nX\t\tY\n");
    fprintf(fp," The respective values of x and y are\n     x  \t     y\n\n");
    m=f(x0,y0)-h+h*f(x0,y0);                               //second orderr runga kutta formula
    while(x<xn)                                            //method application here
    {
        y=y+m*h;
        x=x+h;
        printf("%f\t%f\n",x,y);
        fprintf(fp,"  %f\t%f\n",x,y);                      //printing final results at each iteration
    }
}
```

Output

```
  Second Order runge kutta method for solving differential equation.

DE: y'=y-x

 The respective values of x and y are
     x                y

  0.050000      2.102500
  0.100000      2.205000
  0.150000      2.307500
  0.200000      2.410000
```

24. Secant Method

```
/*Program to apply secant method*/
#include<stdio.h>
#include<math.h>
#define f(x) exp(-x)-x  //original function
int main()              //main begins
{
                FILE *fp;
                fp= fopen("24.txt","w");
                float x0,x1,x2,order,error,pre_error;
                int iter=1;
                fprintf(fp,"    SECANT METHOD\n\n");
                printf("Solving Equation e^(-x)-x=0 using Secant method:\n");
                printf("Please input initial approximations: ");
                scanf("%f%f",&x0,&x1);
                //taking inputs from user
                fprintf(fp,"Sl\t\tx0\t\t\tx1\t\t\tf(x0)\t\tf(x1)\t\tx2\t\t|x1-x0|\t\tOrder(c)\n");
                do
                        //method loop
                {
                    x2=(x0*(f(x1))-x1*(f(x0)))/((f(x1))-(f(x0)));         //formula to apply secant method
                    pre_error=fabs(x1-x0);
                    error=fabs(x2-x1);
                    order=log(error)/log(pre_error);

                fprintf(fp,"%d\t%9f\t%9f\t%9f\t%9f\t%9f\t%9f\t%9f\n",iter,x0,x1,(f(x0)),(f(x1)),x2,fabs(x2-
x1),fabs(order));           //printing results in the file
                    x0=x1;
                    x1=x2;
                    iter++;
                }while(pre_error>0.0005);                        //condition for loop to end
                fprintf(fp,"\n\nThe root of the equation is %f.\n",x1);
                printf("\nThe root of the equation is: %f\n",x1);            //printing final results
                return 0;
                fclose(fp);
}
```

Output

```
   SECANT METHOD

Sl         x0          x1          f(x0)        f(x1)        x2          |x1-x0|      Order(c)
1          0.000000    2.000000    1.000000     -1.864665    0.698162    1.301838     0.380550
2          2.000000    0.698162    -1.864665    -0.200663    0.541172    0.156990     7.019472
3          0.698162    0.541172    -0.200663    0.040893     0.567749    0.026577     1.959261
4          0.541172    0.567749    0.040893     -0.000949    0.567146    0.000603     2.043586
5          0.567749    0.567146    -0.000949    -0.000004    0.567143    0.000003     1.721755
6          0.567146    0.567143    -0.000004    0.000000     0.567143    0.000000     1.#INF00

The root of the equation is 0.567143.
```

25. Simpsons' 1/3 Method

```
/*Program to apply simpson's 1/3 method to solve a definite integrals*/
#include<stdio.h>
#include<math.h>
float f(float x)    //function f=1/(1+x^2)
{
   return 1/(1+x*x);
}
void main()          //main begins here
{
                              FILE *fp;
                              fp=fopen("25.txt","w");
                              fprintf(fp,"      Simpson's method for solving definite integrals\n\nfunction used:-
f(x)=1/(1+x^2)\n");
   int i,n;
   float x0,xn,h,y[20],sumo,sume,ans,x[20];    //variables declaration
   printf("\n Enter values of x0,xn,h: ");  //taking input for the limits of x
   scanf("%f%f%f",&x0,&xn,&h);
   n=(xn-x0)/h;
   if(n%2==1) n++;
   h=(xn-x0)/n;
   printf("\n   Y values: \n");
   fprintf(fp,"\n Y values: \n");
   for(i=0; i<=n; i++)  //calculation of required values of x and y
   {
      x[i]=x0+i*h;
      y[i]=f(x[i]);
      printf("\n f(%f)=%f\n",x[i],y[i]);
      fprintf(fp,"\n f(%f)=%f\n",x[i],y[i]);
   }
   sumo=0;
   sume=0;
   for(i=1; i<n; i++)
   {
      if(i%2==1) sumo+=y[i];
      else sume+=y[i];
   }
   ans=h/3*(y[0]+y[n]+4*sumo+2*sume);   //calculation of the final value
   printf("\n Final integration is %f",ans);
   fprintf(fp,"\n Final integration is %f",ans);
   fclose(fp);
}
```

Output

```
        Simpson's method for solving definite integrals

function used:- f(x)=1/(1+x^2)

 Y values:

 f(0.000000)=1.000000

 f(0.100000)=0.990099

 f(0.200000)=0.961538

 f(0.300000)=0.917431

 f(0.400000)=0.862069

 f(0.500000)=0.800000

 f(0.600000)=0.735294

 f(0.700000)=0.671141

 f(0.800000)=0.609756

 f(0.900000)=0.552486

 f(1.000000)=0.500000

 Final integration is 0.785398
```

26. Trapezoidal Method

```
/*Program to integrate f(x)=1/(1+x^2) using trapezoidal rule*/
#include<stdio.h>
#include<math.h>
#define f(x) 1/(1+pow(x,2))
int main(void)
{
                FILE *fp;
                fp=fopen("26.txt","w");
                fprintf(fp," Integrating f(x)=1/(1+x^2) using trapezoidal rule\n\n");
  int i,n;
  float x0,xn,h,sumo=0,sume=0,ans;
  printf("\n Enter values of x0,xn,h(length of sub domain):\n");
  scanf("%f%f%f",&x0,&xn,&h);
  n=(xn-x0)/h;
  if(n%2==1)
  {
    n=n+1;
  }
  h=(xn-x0)/n;
  float x[n],y[n];
  printf("\nrefined value of n and h are:%d  %f\n",n,h);
  printf("\n f(x) values \n"); //printing function values
  fprintf(fp,"\n f(x) values \n");
  for(i=0; i<=n; i++)
  {
    x[i]=x0+i*h;
    y[i]=f(x[i]);
    printf("\nf(%f)=%f\n",x[i],y[i]);
    fprintf(fp,"\nf(%f)=%f\n",x[i],y[i]);
  }
  for(i=1; i<n; i++)  //calculating integration sum
  {
    if(i%2==1)
    {
      sumo=sumo+y[i];
    }
    else
    {
      sume=sume+y[i];
    }
  }
  ans=h/3*(y[0]+y[n]+4*sumo+2*sume);   //calculating final integration sum
  printf("\nfinal integration is %f",ans); //printing final result
                fprintf(fp,"\n Final integration value is %f",ans);
                fclose(fp);
}
```

Output

```
 Integrating f(x)=1/(1+x^2) using trapezoidal rule


 f(x) values

f(0.000000)=1.000000

f(0.100000)=0.990099

f(0.200000)=0.961538

f(0.300000)=0.917431

f(0.400000)=0.862069

f(0.500000)=0.800000

f(0.600000)=0.735294

f(0.700000)=0.671141

f(0.800000)=0.609756

f(0.900000)=0.552486

f(1.000000)=0.500000

 Final integration value is 0.785398
```