CLASS - BCSE SECOND YEAR (A2)
NAME – SOULIB GHOSH
ROLL – 0016105010 47
SUB – NUMERICAL LAB ASSIGNMENT

Content :-

Solving linear equation

1. Bisection method (24^{th} July)
2. Regula falsi method (1^{st} August)
3. Fixed point itteration method (31^{st} July)
4. Newton Rapshon method (7^{th} August)
5. Secant method (21^{st} August)

Solving Linear equation of more than 2 variable

1. Gauss Seidal Method (4^{th} Sep.)
2. Gauss elimination (28^{th} August)
3. LU decomposition (30^{th} Oct.)

Inverse of a matrix

1. Gauss Jordan elimination (11^{th} Sep.)

Finding maximum eigen value and corresponding eigen vector

1. Power method (18^{th} sep.)

Interpolation

1. Newton forward interpolation (9^{th} Oct.)
2. Newton backward interpolation (9^{th} Oct.)
3. Newton divided difference interpolation (16^{th} Oct.)

Integration

1. Trapizoidal method (23^{rd} Oct)
2. Simpson’s 1/3 rd method (23^{rd} Oct)
3. Simpson’s 3/8 th method (23^{rd} Oct)

Solving Ordinary differential equation

1. Euler method (30^{th} Oct.)
2. Modified Euler method (30^{th} Oct.)
3. Range Kutta second order (30^{th} Oct.)

Solving linear equation

---

1. Bisection Method

code

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
float fun(float x);
void bisection(float a,float b,float c);

int main(){

float a,b,error; // variable

printf("write the two initial approximation\n");
scanf("%f %f",&a,&b);

if(fun(a) * fun(b) > 0){
        printf("you have entered wrong input\n");
        return 0;
}
printf("write the error\n");
scanf("%f",&error);
printf("iteration       A     B           error      root       function value\n\n");
bisection(a,b,error);
return 0;
}




float fun(float x){
        float y;
        y = x*x - 9;
        return y;
}



void bisection(float a,float b,float c){

        int itr=0;
        float root;
        float error = 0;
        while((b-a) > c){
                root = (a+b)/2;
                error = fabs(b-a);
printf("%d       %f   %f     %f     %f          %f\n",itr,a,b,error,(a+b)/2,fun((a+b)/2));
```

```
        if(fun((b+a)/2) * fun(a) > 0)
            a = (a+b)/2;
        if(fun((b+a)/2) * fun(b) > 0)
            b = (a+b)/2;
        if(fun((b+a)/2) == 0){
            printf("the root is %f\n",(b+a)/2);
            printf("you got the root after ineration  %d\n",itr+1);
            return 0;
        }

        itr = itr + 1;
    }

    printf("after iteration number %d the root is %f\n",(itr-1),root);

}
```

Sample Output

Function is x*x – 9 = 0

write the two initial approximation
1 10
write the error
0.0001

iteration	A	B	error	root	function value
0	1.000000	10.000000	9.000000	5.500000	21.250000
1	1.000000	5.500000	4.500000	3.250000	1.562500
2	1.000000	3.250000	2.250000	2.125000	-4.484375
3	2.125000	3.250000	1.125000	2.687500	-1.777344
4	2.687500	3.250000	0.562500	2.968750	-0.186523
5	2.968750	3.109375	0.140625	3.039062	0.235901
6	2.968750	3.039062	0.070312	3.003906	0.023453
7	2.968750	3.003906	0.035156	2.986328	-0.081844
8	2.986328	3.003906	0.017578	2.995117	-0.029273
9	2.995117	3.003906	0.008789	2.999512	-0.002930
10	2.999512	3.001709	0.002197	3.000610	0.003662
11	2.999512	3.000610	0.001099	3.000061	0.000366
12	2.999512	3.000061	0.000549	2.999786	-0.001282
13	2.999786	3.000061	0.000275	2.999924	-0.000458
14	2.999924	3.000061	0.000137	2.999992	-0.000046

after iteration number 14 the root is 2.999992

2. Regula Falsi

code

```
#include<stdio.h>
#include<math.h>
float f(float x)
{
   return x*x - 9;
}
void regula (float *x, float x0, float x1, float fx0, float fx1, int *itr)
{
   *x = x0 - ((x1 - x0) / (fx1 - fx0))*fx0;
   ++(*itr);
   printf("%d            %f       %f       %f       %f       %f\n",*itr,x0,x1,fabs(x0-x1),f(*x),*x);
}
void main ()
{
   int itr = 0, maxmitr;
   float x0,x1,x2,x3,allerr;
   printf("\nEnter the values of x0, x1, allowed error and maximum iterations:\n");
   scanf("%f %f %f %d", &x0, &x1, &allerr, &maxmitr);
printf("iter. no.           x1               x2                  abs. error    fn value       root\n");
   regula (&x2, x0, x1, f(x0), f(x1), &itr);

   do
   {
      if (f(x0)*f(x2) < 0)
         x1=x2;
      else
         x0=x2;
      regula (&x3, x0, x1, f(x0), f(x1), &itr);
      if (fabs(x3-x2) < allerr)
      {
         printf("After %d iterations, root = %6.4f\n", itr, x3);
         return 0;
      }
      x2=x3;
   }
   while (itr<maxmitr);
   printf("Solution does not converge or iterations not sufficient:\n");
   return 1;
}
```

Sample Output

Function is x*x – 9 = 0

Enter the values of x0, x1, allowed error and maximum iterations:
1 10 0.0001 50

iter. no.	x1	x2	abs. error	fn value	root
1	1.000000	10.000000	9.000000	-6.016529	1.727273
2	1.727273	10.000000	8.272727	-3.981010	2.240310
3	2.240310	10.000000	7.759690	-2.417964	2.565548
4	2.565548	10.000000	7.434452	-1.393568	2.757976
5	2.757976	10.000000	7.242024	-0.779123	2.867207
6	2.867207	10.000000	7.132793	-0.428232	2.927758
7	2.927758	10.000000	7.072242	-0.233171	2.960883
8	2.960883	10.000000	7.039117	-0.126312	2.978873
9	2.978873	10.000000	7.021127	-0.068235	2.988606
10	2.988606	10.000000	7.011395	-0.036806	2.993859
11	2.993859	10.000000	7.006141	-0.019837	2.996692
12	2.996692	10.000000	7.003308	-0.010687	2.998218
13	2.998218	10.000000	7.001781	-0.005757	2.999040
14	2.999040	10.000000	7.000959	-0.003099	2.999483
15	2.999483	10.000000	7.000517	-0.001669	2.999722
16	2.999722	10.000000	7.000278	-0.000899	2.999850
17	2.999850	10.000000	7.000150	-0.000484	2.999919

After 17 iterations, root = 2.9999

---

3. Fixed point itteration

code

```
#include<stdio.h>
#include<math.h>

float function (float x)
{
   return (x*x - x -6);
}

float af (float x)
{
float y;
y = sqrt(x+6);
return y;
}


float derivative (float x)
{
         return (2*x - 1);
}

int main()
{
int count=1;
float x0,e;
```

```
printf("write the initial approximation\n");

 printf("write the value of e\n");
 scanf("%f",&x0);
 scanf("%f",&e);
printf("iteration number        X_i       X_i+1       absolute error\n");
float x1;
x1 = af(x0);

while((fabs(x1-x0) > e) && (count < 15)){
printf("%d                  %f      %f        %f\n",count,x0,x1,fabs(x0-x1));
x0 = x1;
x1 = af(x0);
count++;
}

printf(" The final root is %f\n",x1);
return 0;
}
```

Sample Output

Function is x*x – x – 6 = 0

write the initial approximation
write the value of e
8
0.00001

iteration number	X_i	X_i+1	absolute error
1	8.000000	3.741657	4.258343
2	3.741657	3.121163	0.620495
3	3.121163	3.020126	0.101037
4	3.020126	3.003352	0.016774
5	3.003352	3.000559	0.002794
6	3.000559	3.000093	0.000465
7	3.000093	3.000015	0.000078
8	3.000015	3.000003	0.000013

The final root is 3.000000

---

4. Newton Rapshon

code

```
#include<stdio.h>
#include<math.h>
```

```
float f(float x)
{
   return x*x - 9;
}
float df (float x)
{
   return 2*x;
}
void main()
{
   int itr, maxmitr;
         float g,i;
   float h,h1=0, x0, x1, allerr;
   printf("\nEnter x0, allowed error and maximum iterations\n");
   scanf("%f %f %d", &x0, &allerr, &maxmitr);
printf("itteration number       x1         x2        absoulute_error         function_value\n");

   for (itr=1; itr<=maxmitr; itr++)
   {
      h=f(x0)/df(x0);
      x1=x0-h;
         g = h1/h;
      i = log(g);

      if (fabs(h) < allerr)
      {
         printf("After %3d iterations, root = %8.6f\n", itr-1, x1);
         return 0;
      }

         h1 = fabs(x1-x0);

printf("%d                    %f    %f       %f             %f          \n",itr,x0,x1,h1,f(x0));
      x0=x1;

   }
   printf(" The required solution does not converge or iterations are insufficient\n");
   return 1;
}
```

Sample Output

Function is x*x – 9 = 0

Enter x0, allowed error and maximum iterations
9
0.00001
50

itteration number	x1	x2	absoulute_error	function_value
1	9.000000	5.000000	4.000000	72.000000
2	5.000000	3.400000	1.600000	16.000000
3	3.400000	3.023530	0.376471	2.560000
4	3.023530	3.000092	0.023438	0.141731
5	3.000092	3.000000	0.000092	0.000549

After   5 iterations, root = 3.000000

---

5. Secant Method

code

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>

float f(float x); // the function

int main()
{
    float a,b,c=0,e;
    int count=0,n;
    float error;
    float conver;
    // taking input for A and B
    printf("\nEnter the values of a and b:\n"); //(a,b) must contain the solution.
    scanf("%f%f",&a,&b);




   // checking condition for both input
   if(f(a) == 0) {
         printf("the solution is %f\n",a);
         return 0;
  }

   if(f(b) == 0){
         printf("the solution is %f\n",b);
         return 0;
    }

 if(f(a) * f(b) > 0){
         printf("the given input is wrong\n a,b should be on the opposite side of the root\n");
         return 0;
   }
 // checking of condition ends
```

```
// taking input for approximation and iteration number
    printf("Enter the values of allowed error and maximun number of iterations:\n");
    scanf("%f %d",&e,&n);



 // printing the heading
 printf("Iteration No.     X0                X1          Rooterror\n\n");




// the main loop begins
    do
    {

        c=(a*f(b)-b*f(a))/(f(b)-f(a)); // the main formula
        error = fabs(a-b);
printf("%d              %f      %f      %f  %f\n",count+1,a,b,c,error);

        a=b;
        b=c;

        count++;

        // checking the condition if the iteration number exceeds
        if(count==n)
        {
            printf("the iteration number is very less to get the given approximation\n");
        break;
        }
        // condition checking ends



     }
    // main loop ends
     while(fabs(f(c))>e); // the while loop for terminating  the do loop




    printf("\n\n\n The required solution is %f\n",c);// printing a final solution



    return 0; // return type

}
```

```
// user defined function begins
float f(float x)
{
   return (x*x-9);
}
// user defined function ends
```

Sample Output

Function is x*x – 9 = 0

Enter the values of a and b:
1 10
Enter the values of allowed error and maximun number of iterations:
0.000001
50

Iteration No.	X0	X1	Root	error
1	1.000000	10.000000	1.727273	9.000000
2	10.000000	1.727273	2.240310	8.272727
3	1.727273	2.240310	3.243695	0.513037
4	2.240310	3.243695	2.966242	1.003384
5	3.243695	2.966242	2.998675	0.277453
6	2.966242	2.998675	3.000007	0.032434
7	2.998675	3.000007	3.000000	0.001332

The required solution is 3.000000

******************** END OF SOLVING EQUATION***************************

Solving Linear equation of more than 2 variable

1. Gauss Seidal method

code

```
#include<stdio.h>
#include<math.h>

int main()
{
     int count, t, limit;
     float temp, error, a, sum = 0;
```

```
float matrix[10][10], y[10], allowed_error;
printf("\nEnter the Total Number of Equations:\t");
scanf("%d", &limit);
printf("Enter Allowed Error:\t");
scanf("%f", &allowed_error);
printf("\nEnter the Co-Efficients\n");
for(count = 1; count <= limit; count++)
{
    for(t = 1; t <= limit + 1; t++)
    {
        printf("Matrix[%d][%d] = ", count, t);
        scanf("%f", &matrix[count][t]);
    }
}

// scanning end

for(count = 1; count <= limit; count++)
{
  float max = -10000;
    for(t = 1; t <= limit; t++)
    {
      if(matrix[count][t] > max) max = matrix[count][t];
    }
    if(max != matrix[count][count])
    {
      printf("You cannot solve as this does not satisfy the condition\n");
      return 0;
    }
}




for(count = 1; count <= limit; count++)
{
    y[count] = 0;
}
do
{
    a = 0;
    for(count = 1; count <= limit; count++)
    {
        sum = 0;
        for(t = 1; t <= limit; t++)
        {
            if(t != count)
            {
                sum = sum + matrix[count][t] * y[t];
            }
        }
```

```
            temp = (matrix[count][limit + 1] - sum) / matrix[count][count];
            error = fabs(y[count] - temp);
            if(error > a)
            {
                a = error;
            }
            y[count] = temp;
            printf("\nY[%d]=\t%f", count, y[count]);
        }
        printf("\n");
    }
    while(a >= allowed_error);
    printf("\n\nSolution\n\n");
    for(count = 1; count <= limit; count++)
    {
        printf("\nY[%d]:\t%f", count, y[count]);
    }
    printf("\n");
    return 0;
}
```

Sample Output

```
Enter the Total Number of Equations:    3
Enter Allowed Error:  0.0001

Enter the Co-Efficients
Matrix[1][1] = 5
Matrix[1][2] = 3
Matrix[1][3] = 2
Matrix[1][4] = 1
Matrix[2][1] = 2
Matrix[2][2] = 8
Matrix[2][3] = 3
Matrix[2][4] = 6
Matrix[3][1] = 1
Matrix[3][2] = 4
Matrix[3][3] = 9
Matrix[3][4] = 4

Y[1]=  0.200000
Y[2]=  0.700000
Y[3]=  0.111111

Y[1]=  -0.264444
Y[2]=  0.774444
Y[3]=  0.129630

Y[1]=  -0.316518
```

Y[2]= 0.780519
Y[3]= 0.132716

Y[1]= -0.321398
Y[2]= 0.780581
Y[3]= 0.133230

Y[1]= -0.321641
Y[2]= 0.780449
Y[3]= 0.133316

Y[1]= -0.321596
Y[2]= 0.780405
Y[3]= 0.133330

Solution

Y[1]: -0.321596
Y[2]: 0.780405
Y[3]: 0.133330

---

2. Gauss elimination method

code

```
#include<stdio.h>
#include<stdlib.h>

int main()
{
	float **A,*x,max,var,m,s;
	int n,i,j,l,k=0,temp=0;

	printf("\nenter the degree of the equation: ");
	scanf("%d",&n);

	A=(float **)malloc(n*sizeof(float*));
	for(i=0;i<n;i++)
		A[i]=(float*)malloc((n+1)*sizeof(float));

	x=(float *)malloc(n*sizeof(float));

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

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

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

	n=n-1;

	while(k<=n)
	{
		l=k;
		max=A[k][k];
		while(k+1<=n)
		{
			if(max<A[k+1][k])
			{
				max=A[k+1][k];
				temp=k+1;
			}
			k++;
		}
		k=l;
		if(temp!=0)
		{
			for(i=0;i<=n+1;i++)
			{
				var=A[k][i];
				A[k][i]=A[temp][i];
				A[temp][i]=var;
			}
			printf("\n \n");
			for(i=0;i<=n;i++)
			{
				printf("\t");
				for(j=0;j<=n+1;j++)
					printf("%f ",A[i][j]);
				printf("\n");
			}
			printf("\n\n");
		}
		i=k+1;
		while(i<=n)
		{
			m=A[i][k]/A[k][k];
			j=k;
			while(j<=n+1)
			{
				A[i][j]=A[i][j]-m*A[k][j];
				j++;
			}
			i++;
		}
		k++;

		printf("\n \n");
		for(i=0;i<=n;i++)
```

```
		{
			printf("\t");
			for(j=0;j<=n+1;j++)
				printf("%f ",A[i][j]);
			printf("\n");
		}
		printf("\n\n");
	}

	printf("\n \n");
	for(i=0;i<=n;i++)
	{
		printf("\t");
		for(j=0;j<=n+1;j++)
			printf("%f ",A[i][j]);
		printf("\n");
	}
	printf("\n\n");
	x[n]=A[n][n+1]/A[n][n];
	i=n-1;
	while(i>=0)
	{
		j=i+1;
		s=A[i][n+1];
		while(j<=n)
		{
			s=s-A[i][j]*x[j];
			j++;
		}
		s/=A[i][i];
		x[i]=s;
		i--;
	}
	printf("\n\n\t So the solution is \n");
	for(i=0;i<=n;i++)
		printf("\t X%d->%f\n",i+1,x[i]);
	return 0;
}
```

Sample Output

enter the degree of the equation: 3

enter the coefficient matrix: 2
3
1
9
1
2
3

6
3
1
2
8

2.000000 3.000000 1.000000 9.000000
0.000000 0.500000 2.500000 1.500000
0.000000 -3.500000 0.500000 -5.500000

2.000000 3.000000 1.000000 9.000000
0.000000 0.500000 2.500000 1.500000
0.000000 0.000000 18.000000 5.000000

2.000000 3.000000 1.000000 9.000000
0.000000 0.500000 2.500000 1.500000
0.000000 0.000000 18.000000 5.000000

2.000000 3.000000 1.000000 9.000000
0.000000 0.500000 2.500000 1.500000
0.000000 0.000000 18.000000 5.000000

So the solution is
X1->1.944445
X2->1.611111
X3->0.277778

3. LU decomposition

code

```
//Solution for Linear Simulteneous equations using LU Decomposition
#include<stdio.h>
```

```
#include<stdlib.h>
#include<math.h>

int main()
{
	float **A,**L,**U,*B,*Y,*X,sum=0;
	int n,i,j,k,p;
	printf("Please enter the size of the matrix:\n");
	scanf("%d",&n);
	A=(float **)malloc(n*sizeof(float *));
	for(i=0;i<n;i++)
		A[i]=(float *)malloc(n*sizeof(float));

	printf(" entry the coefficient matrix:\n");
	for(i=0;i<n;i++)
		for(j=0;j<n;j++)
			scanf("%f",&A[i][j]);

	L=(float **)calloc(n,sizeof(float *));
	for(i=0;i<n;i++)
		L[i]=(float *)calloc(n,sizeof(float));

	U=(float **)calloc(n,sizeof(float *));
	for(i=0;i<n;i++)
		U[i]=(float *)calloc(n,sizeof(float));
	B=(float *)calloc(n,sizeof(float));
	Y=(float *)calloc(n,sizeof(float));
	X=(float *)calloc(n,sizeof(float));

	for(j=0;j<n;j++)
	{
		for(i=0;i<n;i++)
		{
		   if(i>=j)
		   {
		      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
		   {

		      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 matrix:");
```

```
        for(i=0;i<n;i++)
                {
                        printf("\n\t");
                        for(j=0;j<n;j++)
                                printf("%f ",L[i][j]);
                }

                printf("\n\n");

        printf("\nU matrix:");
        for(i=0;i<n;i++)
                {
                        printf("\n\t");
                        for(j=0;j<n;j++)
                                printf("%f ",U[i][j]);
                }

        printf("\nPlease entry the constant terms\n");
        for(i=0;i<n;i++)
                scanf("%f",&B[i]);

        for(i=0;i<n;i++)
        {
        Y[i]=B[i];
        for(j=0;j<i;j++)//Forword substitution
        {
                        Y[i]-=L[i][j]*Y[j];
        }
        Y[i]/=L[i][i];
        }
        printf("\n[Y]:");

        for(i=0;i<n;i++)
                printf("\t%f ",Y[i]);

        for(i=n-1;i>=0;i--)//Back Substitution
        {
        X[i]=Y[i];
        for(j=i+1;j<n;j++)
                X[i]-=U[i][j]*X[j];
      }
        printf("\n So the result will be: \n");

        for(i=0;i<n;i++)
                printf("\t%0.5f ",X[i]);

        printf("\n\n");

        return 0;
}
```

Sample Output

Please enter the size of the matrix:
3
 entry the coefficient matrix:
2
3
1
1
2
3
3
1
2

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

U matrix:
    1.000000 1.500000 0.500000
    0.000000 1.000000 5.000000
    0.000000 0.000000 1.000000
Please entry the constant terms
9
6
8

[Y]:   4.500000   3.000000   0.277778
 So the result will be:
    1.94444   1.61111   0.27778

******************* END OF SOLVING LINEAR EQUATION **********************

Inverse of a matrix

1. Gauss Jordan elimination

Code

```
#include<stdio.h>
#include<math.h>

#define maximum 10
#define minvalue 0.0005

int main()
{
 float augmentedmatrix[maximum][2*maximum] ;
 int n,m;
 float temporary, r ;
 int i, j, k, dimension, temp;

 printf("\n Enter the dimension of the matrix to be provided as input : \n");
 scanf("%d",&dimension);


 // scanning starts

 printf("\n Enter a non-singular %dx%d matrix : \n",dimension,dimension);
 for(i=0; i<dimension; i++)
  for(j=0; j<dimension; j++)
           scanf("%f",&augmentedmatrix[i][j]) ;

// scanning ends


 // making a identity matrix

 for(i=0;i<dimension; i++)
  for(j=dimension; j<2*dimension; j++)
      if(i==j%dimension)
         augmentedmatrix[i][j]=1;
      else
         augmentedmatrix[i][j]=0;


    // making identity matrix ends


  // printing matrix starts

  for(n=0; n<dimension; n++)
 {
  for(m=0; m<2*dimension; m++)
           printf("    %4.2f",augmentedmatrix[n][m]) ;
  printf("\n");
 }

 printf("\n\n\n");
```

```
 // printing matrix ends


 // the process begins
 for(j=0; j<dimension; j++)
 {
  temp=j;

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

  for(i=j+1; i<dimension; i++)
if(augmentedmatrix[i][j]>augmentedmatrix[temp][j])
                    temp=i;

  if(fabs(augmentedmatrix[temp][j])<minvalue)
            {
              printf("\n Elements are too small to deal with !!!");
              return 0;
            }

 /* swapping row which has maximum jth column element */

  if(temp!=j)
           for(k=0; k<2*dimension; k++)
           {
           temporary=augmentedmatrix[j][k] ;
           augmentedmatrix[j][k]=augmentedmatrix[temp][k] ;
           augmentedmatrix[temp][k]=temporary ;
           }

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

  for(i=0; i<dimension; i++)
           if(i!=j)
           {
           r=augmentedmatrix[i][j];
           for(k=0; k<2*dimension; k++)
             augmentedmatrix[i][k]-=(augmentedmatrix[j][k]/augmentedmatrix[j][j])*r ;
           }
           else
           {
           r=augmentedmatrix[i][j];
           for(k=0; k<2*dimension; k++)
             augmentedmatrix[i][k]/=r ;
           }



           for(n=0; n<dimension; n++)
 {
  for(m=0; m<2*dimension; m++)
           printf("     %4.2f",augmentedmatrix[n][m]) ;
```

```
 printf("\n");
}

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

}

// the process end here


// Display the inverse


printf("\n\n\n The inverse of the entered non-singular matrix is : \n\n");

for(i=0; i<dimension; i++)
{
 for(j=dimension; j<2*dimension; j++){
          printf("     %.5f",augmentedmatrix[i][j]);
 }
          printf("\n");

}

return 0;
}
```

Sample Output

```
Enter the dimension of the matrix to be provided as input :
3
Enter a non-singular 3x3 matrix :
2 1 1 3 2 3 1 4 9

   2.00    1.00    1.00    1.00    0.00    0.00
   3.00    2.00    3.00    0.00    1.00    0.00
   1.00    4.00    9.00    0.00    0.00    1.00



    1.00     0.67     1.00     0.00     0.33     0.00
    0.00     -0.33     -1.00     1.00     -0.67     0.00
    0.00     3.33     8.00     0.00     -0.33     1.00



    1.00     0.00     -0.60     0.00     0.40     -0.20
```

```
0.00    1.00    2.40    0.00    -0.10    0.30
0.00    0.00    -0.20    1.00    -0.70    0.10



1.00    0.00    0.00    -3.00    2.50    -0.50
0.00    1.00    0.00    12.00    -8.50    1.50
-0.00    -0.00    1.00    -5.00    3.50    -0.50


The inverse of the entered non-singular matrix is :

-3.00000    2.50000    -0.50000
12.00001    -8.50001    1.50000
-5.00000    3.50000    -0.50000
```

**************************END OF MATRIX INVERSION*************************

Finding maximum eigen value and corresponding eigen vector

Power Method:-

code

```
#include<stdio.h>
#include<math.h>
void main()
{

   int i,j,n;

   float A[40][40],x[40],z[40],e[40],max1,max2;
   // A is the input matrix
   // X is the coloumn matrix


   printf("\nEnter the order of matrix:");
   scanf("%d",&n);
```

```
printf("\nEnter matrix elements row-wise\n");

// taking the input

for(i=1; i<=n; i++)
{
   for(j=1; j<=n; j++)
   {
      printf("A[%d][%d]=", i,j);
      scanf("%f",&A[i][j]);
   }
}




// asking for the coloumn vector for starting calculation

printf("\nEnter the column vector\n");
for(i=1; i<=n; i++)
{
   printf("X[%d]=",i);
   scanf("%f",&x[i]);
}



// the main process begins


do
{



// first of all the matrix is multiplied with the coloumn matrix
   for(i=1; i<=n; i++)
   {
      z[i]=0;
      for(j=1; j<=n; j++)
      {
         z[i]=z[i]+A[i][j]*x[j];
      }
   }

 // the process of multiplication ends here


 // finding the largest value of the coloumn matrix we got after multiplication


   max1=fabs(z[1]);
   for(i=2; i<=n; i++)
```

```
  {
     if((fabs(z[i]))>max1)
        max1=fabs(z[i]);
  }


// we got the largest element in the coloumn matrix after multiplication


// we divide all the elements of the coloumn matrix by the largest element
  for(i=1; i<=n; i++)
  {
     z[i]=z[i]/max1;
  }

 // division process ends

 // now we take the difference of every element with the given coloumn matrix


  for(i=1; i<=n; i++)
  {
     e[i]=0;
     e[i]=fabs((fabs(z[i]))-(fabs(x[i])));
  }

  // taking of the difference ends


  // we find the largest difference

  max2=e[1];
  for(i=2; i<=n; i++)
  {
     if(e[i]>max2)
        max2=e[i];
  }

  // we got the largest difference


  // we replace the new coloumn matrix with the previous coloumn matrix


  for(i=1; i<=n; i++)
  {
     x[i]=z[i];
  }

  // process of replacing ends

  printf("the eigen value is : %f\n\n",max1);
```

```
        printf("the error is: %f\n\n\n\n",max2);

    }

    // we set an approximation

    while(max2>0.001);

    // main process ends here



    // printing the eigrn value
    printf("\n The final eigen value is %f",max1);


    // printing the eigen vector
    printf("\n\nThe final eigen vector is as follows :\n");

    for(i=1; i<=n; i++)
    {
        printf("%f\n",z[i]);
    }


    printf("\n");
    }
```

Sample Output

```
Enter the order of matrix:3

Enter matrix elements row-wise
A[1][1]=2
A[1][2]=3
A[1][3]=4
A[2][1]=5
A[2][2]=1
A[2][3]=2
A[3][1]=6
A[3][2]=8
A[3][3]=4

Enter the column vector
```

X[1]=6
X[2]=4
X[3]=2

the eigen value is : 76.000000
the error is: 5.578948

the eigen value is : 10.526316
the error is: 0.181447

the eigen value is : 11.115000
the error is: 0.052828

the eigen value is : 11.440845
the error is: 0.016397

the eigen value is : 11.275665
the error is: 0.003139

the eigen value is : 11.318303
the error is: 0.000856

The final eigen value is 11.318303
The final eigen vector is as follows :
0.582985
0.476760
1.000000

************************END OF FINDING MAXIMUM EIGEN VALUE****************

<u>Interpolation</u>

---

1. Newton forward Interpolation

Code

```
#include<stdio.h>
#include<math.h>


int main()
{
f();
```

```
return 0;
}

void f()
{

// taking variables
  float x[10],y[10][10],p,u,numerator=1.0, denominator=1.0,yp;
  int i,n,j,k=0,f,m;
// taking of variables ends


  // taking the input begins

  printf("\nwrite the order ");
  scanf("%d",&n); // taking the order
  for(i=0; i<n; i++)
  {
   printf("\n value of x%d: ",i);
   scanf("%f",&x[i]);  // taking the x values

   printf("\n value of f(x%d): ",i);
   scanf("%f",&y[k][i]); // taking the respective y value


}

  printf("\n\nEnter X where you want the value ");
  scanf("%f",&p); // where we have to find the value

// end of taking inputs


// checking condition for newton method
float diff;
diff = x[1] - x[0] ;
for(i=0;i<=n-2;i++){
	if((x[i+1] - x[i]) != diff){
	printf("the difference of x value is not same\n");
	return 0;
	}
}


// end of checking condition for newton method
// creating difference table
  for(i=1;i<n;i++)
  {
    for(j=0;j<n-i;j++)
    {
     y[i][j]=y[i-1][j+1]-y[i-1][j];
    }
```

```
  }

  // end of creating difference table


  //printing difference table starts
  printf("\n\n the difference table is as follows\n");
  printf("\n    x\t        y\t   ");
  printf("\n______________________________________________________________\n");
  for(i=0;i<n;i++)
  {
    printf("\n %.3f",x[i]);
    for(j=0;j<n-i;j++)
    {
     printf("   ");
     printf(" %.3f",y[j][i]);
    }
   printf("\n");
  }

// printing difference table ends


// the main interpolation process begins

f=0;
u=(p-x[f])/(x[f+1]-x[f]);
printf("\n the value of u = %.3f ",u);


yp = y[0][0];

  for (k=1;k<n;k++)
    {
        numerator *=u-k+1;
        denominator *=k;
        yp +=(numerator/denominator)*y[k][0];
    }
  //  printf("\nWhen x = %6.1f, the value of y = %6.2f\n",p,yp);

printf("\n\nWith interpolation the value is %f",yp);
// the main interpolation process ends
}
```

Sample Output

```
write the order 5

 value of x0: 1

 value of f(x0): 1
```

value of x1: 2

value of f(x1): 2

value of x2: 3

value of f(x2): 3

value of x3: 4

value of f(x3): 4

value of x4: 5

value of f(x4): 5

Enter X where you want the value : 3.5

the difference table is as follows

x	y				
1.000	1.000	1.000	0.000	0.000	0.000
2.000	2.000	1.000	0.000	0.000	
3.000	3.000	1.000	0.000		
4.000	4.000	1.000			
5.000	5.000				

the value of u = 2.500

With interpolation the value is 3.500000

---

2. Newton backward Interpolation

code

```
#include<stdio.h>
#include<math.h>



int main()
{
```

```
b();
return 0;
}


void b()
{

// taking variables
  float x[10],y[10][10],p,u,numerator=1.0, denominator=1.0,yp;
  int i,n,j,k=0,f,m;
// taking of variables ends


  // taking the input begins

  printf("\nwrite the order ");
  scanf("%d",&n); // taking the order
  for(i=0; i<n; i++)
  {
   printf("\n value of x%d: ",i);
    scanf("%f",&x[i]);
   printf("\n value of f(x%d): ",i);
   scanf("%f",&y[k][i]);

  }

  printf("\n\nEnter X where you want the value ");
  scanf("%f",&p); // where we have to find the value

// end of taking inputs


// checking condition for newton method
float diff;
diff = x[1] - x[0] ;
for(i=0;i<=n-2;i++){
	if((x[i+1] - x[i]) != diff){
	printf("the difference of x value is not same\n");
	return 0;
	}
}


// end of checking condition for newton method
// creating difference table
  for(i=1;i<n;i++)
  {
    for(j=0;j<n-i;j++)
    {
     y[i][j]=y[i-1][j+1]-y[i-1][j];
    }
```

```
  }

  // end of creating difference table


  //printing difference table starts
  printf("\n\n the difference table is as follows\n");
  printf("\n    x\t        y\t   ");
  printf("\n______________________________________________________________\n");
  for(i=0;i<n;i++)
  {
    printf("\n %.3f",x[i]);
    for(j=0;j<n-i;j++)
    {
     printf("   ");
     printf(" %.3f",y[j][i]);
    }
   printf("\n");
  }

// printing difference table ends


// the main interpolation process begins

f=n-1;
u=(p-x[f])/diff;
printf("\n the value of u = %.3f ",u);


yp = y[0][n-1];

  for (k=1;k<n;k++)
    {
        numerator *=u+k-1;
        denominator *=k;
        yp +=(numerator/denominator)*y[k][n-k-1];
    }
    printf("\nWhen x = %6.1f, corresponding y = %6.2f\n",p,yp);

printf("\n\nWith interpolation the value is %f",yp);


// the main interpolation process ends
}
```

Sample Output

write the order 5

value of x0: 1

value of f(x0): 1

value of x1: 2

value of f(x1): 2

value of x2: 3

value of f(x2): 3

value of x3: 4

value of f(x3): 4

value of x4: 5

value of f(x4): 5

Enter X where you want the value 4.5

the difference table is as follows

x	y				
1.000	1.000	1.000	0.000	0.000	0.000
2.000	2.000	1.000	0.000	0.000	
3.000	3.000	1.000	0.000		
4.000	4.000	1.000			
5.000	5.000				

the value of u = -0.500
When x =   4.5, corresponding y =   4.50

With interpolation the value is 4.500000

3. **Newton divide difference Interpolation**

code

```
#include<stdio.h>
void input(int n,float x[n],float y[n][n],float* f);
void createTable(int n,float x[n],float y[n][n]);
```

```
float calculate(int n,float x[n],float y[n][n],float f);
void displayTable(int n,float x[n],float y[n][n]);
int main()
{
	int i,j,n;
	printf("Enter number of records :\n");
	scanf("%d",&n);
	float x[n],y[n][n],sum,f;
	input(n,x,y,&f);
	createTable(n,x,y);
	displayTable(n,x,y);
	sum=calculate(n,x,y,f);
   printf(" The ans is %f\n",sum);
   return 0;
}
void input(int n,float x[n],float y[n][n],float* f)
{
	int i;
	printf("Enter x and f(x) respectively :\n");
	for(i=0;i<n;i++)
		scanf("%f %f",&x[i],&y[i][0]);
	printf("Enter x for finding f(x) :\n");
	scanf("%f",f);
}
void createTable(int n,float x[n],float y[n][n])
{
	int i,j;
	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])/(x[j+i]-x[j]);
}
float calculate(int n,float x[n],float y[n][n],float f)
{
	int i;
	float sum=0,term=1;
   for(i=0;i<n;i++)
   {
	sum+=(term*y[0][i]);
	term*=(f-x[i]);
   }
   return sum;
}
void displayTable(int n,float x[n],float y[n][n])
{
	int i,j;
	printf("x(i)\t");
	printf("\n");
	for(i=0;i<n;i++)
	{
		printf("%f\t",x[i]);
		for(j=0;j<n-i;j++)
			printf("%f\t",y[i][j]);
```

```
                printf("\n");
        }
        printf("\n");
}
```

Sample Output

```
Enter number of records :
4
Enter x and f(x) respectively :
5 5
9 9
11 11
20 20
Enter x for finding f(x) :
16

  x(i)
5.000000    5.000000    1.000000    0.000000    0.000000
9.000000    9.000000    1.000000    0.000000
11.000000   11.000000   1.000000
20.000000   20.000000

The ans is 16.000000
```

**************************END OF INTERPOLATION****************************

Integration

1. Trapezoidal method

code

```
#include<stdio.h>
#include<math.h>
float fn(float x);



int main()
{
int i,j,n=1;
int count = 0;
float a,b,s=0,y=0,h,error;
float y1 = 100;
```

```
printf(" lower limit=   ");
scanf("%f",&a);
printf(" upper limit=   ");
scanf("%f",&b);
printf("write the error=          ");
scanf("%f",&error);
printf("iteration no.             value of integration\n\n");



while(fabs(y1-y) > error)



{
y1 = y;
y = 0;
s = 0;
count++;

h=(b-a)/n;
for(i=1;i<=n-1;i++)
{
s=s+fn(a+i*h);
}
y=(fn(a)+fn(b)+2*s)*h/2;
n = 2*n;


printf("%d                        %f\n",count,y);
}




return 0;
}



float fn(float x)
{
return x*x;
}
```

Sample Output

Function is x*x

lower limit=   0

upper limit=   2
write the error=   0.00001
iteration no.   value of integration

1	4.000000
2	3.000000
3	2.750000
4	2.687500
5	2.671875
6	2.667969
7	2.666992
8	2.666748
9	2.666687
10	2.666667
11	2.666665

2. Simpson's 1/3 rd Method

code

```
#include<stdio.h>
#include<math.h>
float f(float x);

int main()
{



int n = 2,i;
float s1=0,s2=0,sum = 0,a,b,h;
float error;
int count = 0;
float y1 = 100;


printf("upper limit =   ");
scanf("%f",&b);
printf("lower limit =   ");
scanf("%f",&a);
printf("enter the error=           ");
scanf("%f",&error);
printf("iteration number                      value of integration\n");
```

```
while(fabs(y1 - sum) > error)
{

count++;
y1 = sum;
sum = 0;
s1 = s2 = 0;

h=(b-a)/n;
if(n%2==0)
{
for(i=1;i<=n-1;i++)
  {
     if(i%2==0)
       {
           s1=s1+f(a+i*h);
        }
        else
        {
           s2=s2+f(a+i*h);
         }
  }
  sum=h/3*(f(a)+f(b)+4*s2+2*s1);

printf("%d                                   %f\n",count,sum);
}

/*
else
{
  printf("the rule is not appliciable");
 }

*/



 n = 2*n;

}

return 0;

}


float f(float x)
{
      return x*x;
}
```

Sample Output

Function is x*x

upper limit =   2
lower limit =   0
enter the error=   0.0000001
iteration number   value of integration
1   2.666667
2   2.666667

---

3. Simpson's 3/8 th Method

code

```
#include<stdio.h>
#include<math.h>
double f(double x){
  return x*x*x;
}
int main(){
  int n = 3,i;
  float a,b,h,x,sum=0,integral = 0,error ,e,sum1 = 100;
  printf("\nenter the error ");
  scanf("%f",&e);
  printf("\nEnter the initial limit: ");
  scanf("%f",&a);
  printf("\nEnter the final limit: ");
  scanf("%f",&b);
  printf("integral                    error\n");
  error = fabs(sum1 - integral);


  while(error > e) {

  sum1 = integral;
  integral = 0;

  sum = 0;
  h=fabs(b-a)/n*1.0;
  for(i=1;i<n;i++){
    x=a+i*h;
    if(i%3==0){
      sum=sum+2*f(x);
    }
```

```
    else{
      sum=sum+3*f(x);
    }



    integral=(3*h/8)*(f(a)+f(b)+sum);
     error = fabs(sum1 - integral);

    printf("\n%f            %f",integral,error);

  }
  n = 3*n;
  }
    return 0;
}
```

Sample Output

Function is x*x*x

enter the error 0.00001

Enter the initial limit: 0

Enter the final limit: 3

integral	error
20.250000	20.250000
3.388889	16.861111
3.500000	16.750000
3.750000	16.500000
4.638889	15.611111
6.375000	13.875000
8.375000	11.875000
13.138890	7.111110
20.250002	0.000002

****************************END OF INTEGRATION******************************

Solving Ordinary differential equation

1. Euler method

code

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>



float fun(float x,float y);
float f(float a);


int main()
{



   float a,b,h,t;
   printf(" enter the initial x value and corresponding y value\n");
   scanf("%f %f",&a,&b);
   printf("enter the value of the interval\n");
   scanf("%f",&h);
   printf("enter the value of x to find the corrresponding Y value\n");
   scanf("%f",&t);


     float x,y,k;
   x=a;
   y=b;
   printf("\n  x\t  y       error\n");
   while(x<=t)
   {
      k=h*fun(x,y);
      y=y+k;
      x=x+h;
      printf("%0.3f\t%0.3f\t %0.3f\n",x,y,fabs(f(x) - y));
   }

     return 0;


}

// user defined function

float fun(float x,float y)
{
   float f;
   f=2*x;
   return f;
}
```

```
float f(float a){
        return a*a;
        }
```

sample Output

(function is Y=x^2)

```
 enter the initial x value and corresponding y value
2
4
enter the value of the interval
0.1
enter the value of x to find the corrresponding Y value
3

  x       y      error
2.100   4.400   0.010
2.200   4.820   0.020
2.300   5.260   0.030
2.400   5.720   0.040
2.500   6.200   0.050
2.600   6.700   0.060
2.700   7.220   0.070
2.800   7.760   0.080
2.900   8.320   0.090
3.000   8.900   0.100
```

---

2. Modified Euler method

code

```
#include<stdio.h>
#include<math.h>
#include<string.h>
float f(float,float);
float fun(float a);
int main()
   {

      int i,j,c;
      float x[100],y[100],h,m[100],m1,m2,a,s[100],w;
```

```
        printf("  Enter the initial value of x:");
        scanf("%f",&x[0]);
        printf("\n  Enter the initial value of the variable y corresponding to X:");
        scanf("%f",&y[0]);
        printf("\n  Enter the value of increment h:");
        scanf("%f",&h);
        printf("\n  Enter the final value of x:");
        scanf("%f",&a);

        printf("******result for modified euler method*************\n\n");
        s[0]=y[0];
        for(i=1;x[i-1]<a;i++)
           {
             w=100.0;
            x[i]= x[i-1]+h;
            m[i]=f(x[i-1],y[i-1]);
            c=0;
            while(w>0.0001)
             {
                m1=f(x[i],s[c]);
                m2=(m[i]+m1)/2;
                s[c+1]=y[i-1]+m2*h;
                w=s[c]-s[c+1];
                w=fabs(w);
                c=c+1;
             }
           y[i]=s[c];
           }
        printf("\n\n  x      y_mod_euler      error_mod_euler\n     \n");
        for(j=0;j<i;j++)
           {
             printf("  %f\t%f\t%f",x[j],y[j],fabs(fun(x[j]) - y[j]));
             printf("\n");
           }


           printf("\n");
           return 0;
    }
float f(float a,float b)
    {
        float c;
        c=2*a;
        return(c);
    }

float fun(float a){
           return a*a;
           }
```

sample Output

(function is Y=x^2)

```
  Enter the initial value of x:2

  Enter the initial value of the variable y corresponding to X:4

  Enter the value of increment h:0.1

  Enter the final value of x:3
******result for modified euler method*************



  x            y_mod_euler      error_mod_euler

  2.000000     4.000000         0.000000
  2.100000     4.410000         0.000000
  2.200000     4.840000         0.000000
  2.300000     5.289999         0.000001
  2.400000     5.759999         0.000001
  2.500000     6.249999         0.000001
  2.599999     6.759999         0.000002
  2.699999     7.289999         0.000002
  2.799999     7.839998         0.000002
  2.899999     8.409998         0.000003
  2.999999     8.999998         0.000004
```

---

3. Runge Kutta second order

code

```
#include<stdio.h>
#include<math.h>


 float f(float a);
  float fxy(float x,float y);

 void main()

 {
```

```
    float fxy(float x,float y),x0,y0,x1,y1,xn,h,k1,k2;

    int ns,i;

    printf("\nEnter x0,xn,ns,y0:");

    scanf("%f %f %d %f",&x0,&xn,&ns,&y0);

    h=(xn-x0)/ns;

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

    {

        printf("\n%f    %f     %f",x0,y0,fabs(y0 - f(x0)));

        k1=h*fxy(x0,y0);

        y1=y0+k1;

        x1=x0+h;

        k2=h*fxy(x1,y1);

        y1=y0+(k1+k2)/2;

        y0=y1;

        x0=x1;

    }

}

float fxy(float x,float y)

{

    float dy= 2*x;

    return dy;

}

float f(float a){

return a*a;
}
```

sample Output

(function is Y=x^2)

Enter x0,xn,ns,y0:2 3 10 4

2.000000	4.000000	0.000000
2.100000	4.410000	0.000000
2.200000	4.840000	0.000000
2.300000	5.289999	0.000001
2.400000	5.759999	0.000001
2.500000	6.249999	0.000001
2.599999	6.759999	0.000002
2.699999	7.289999	0.000002
2.799999	7.839998	0.000002
2.899999	8.409998	0.000003
2.999999	8.999998	0.000004

********************* END OF SOLVING DIFFERENTIAL EQUATION****************

END OF FILE