NUMERICAL METHODS LAB

The Neotia University
B.Tech , III Smester

NUMERICAL METHODS LAB
III SEM.

LIST OF EXPERIMENTS

SR. NO.	NAME OF EXPERIMENT
1	TO FIND THE ROOTS OF NON-LINEAR EQUATION USING BISECTION METHOD.
2	TO FIND THE ROOTS OF NON-LINEAR EQUATION USING NEWTON'S METHOD.
3	CURVE FITTING BY LEAST – SQUARE APPROXIMATIONS.
4	TO SOLVE THE SYSTEM OF LINEAR EQUATIONS USING GAUSS - ELIMINATION METHOD.
5	TO SOLVE THE SYSTEM OF LINEAR EQUATIONS USING GAUSS - SEIDAL ITERATION METHOD
6	TO SOLVE THE SYSTEM OF LINEAR EQUATIONS USING GAUSS - JORDEN METHOD.
7	TO INTEGRATE NUMERICALLY USING TRAPEZOIDAL RULE.
8	TO INTEGRATE NUMERICALLY USING SIMPSON'S RULES.
9	TO FIND THE LARGEST EIGEN VALUE OF A MATRIX BY POWER - METHOD.
10	TO FIND NUMERICAL SOLUTION OF ORDINARY DIFFERENTIAL EQUATIONS BY EULER'S METHOD.
11	TO FIND NUMERICAL SOLUTION OF ORDINARY DIFFERENTIAL EQUATIONS BY RUNGE- KUTTA METHOD.

Program 1

OBJECTIVES: To find the roots of non linear equations using

Bisection method.

SOURCE CODE:

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


/*  define prototype for USER-SUPPLIED function f(x)   */

    double ffunction(double x);


/*  EXAMPLE for "ffunction"    */

    double ffunction(double x)

    {
        return (x * sin(x) - 1);
    }

/*                                                      */

/*  Main program for algorithm 2.2   */

    void main()

{
    double Delta = 1E-6;       /* Tolerance for width of interval */ int
    Satisfied = 0;             /* Condition for loop termination */
    double A, B;               /* Endpoints of the interval [A,B] */
    double YA, YB;     /* Function values at the interval-borders */ int
    Max;   /* Calculation of the maximum number of iterations */ int K;
                               /* Loop Counter */
    double C, YC;   /* Midpoint of interval and function value there */


    printf("                                                    \n");
    printf("Please enter endpoints A and B of the interval [A,B]\n");
    printf("EXAMPLE : A = 0 and B = 2. Type:    0 2 \n");
    scanf("%lf %lf", &A, &B);
    printf("The interval ranges from %lf to %lf\n", A,B);

    YA = ffunction(A);     /* compute function values */
    YB = ffunction(B);
    Max = (int) ( 1 + floor( ( log(B-A) - log(Delta) ) / log(2) ) );
    printf("Max = %d\n",Max);

    /* Check to see if the bisection method applies */
```

```
    if( ( (YA >= 0) && (YB >=0) ) || ( (YA < 0) && (YB < 0) )     ) {
      printf("The values ffunction(A) and ffunction(B)\n");
      printf("do not differ in sign.\n");
      exit(0);      /* exit program */
    }

    for(K = 1; K <= Max ; K++) {

        if(Satisfied == 1)   break;

        C = (A + B) / 2;      /* Midpoint of interval */
        YC = ffunction(C);    /* Function value at midpoint */


  if( YC == 0) {          /* first 'if'          */ A
            = C;        /* Exact root is found */ B
            = C;
  }
  else if( ( (YB >= 0) && (YC >=0) ) || ( (YB < 0) && (YC < 0) )     ) {
       B = C;       /* Squeeze from the right */
      YB = YC;
  }
  else {
       A = C;       /* Squeeze from the left */
      YA = YC;
  }

  if( (B-A) < Delta ) Satisfied = 1; /* check for early convergence */

    }  /* end of 'for'-loop */

    printf("                                                  \n");
    printf("The maximum number of iterations is : %d\n",Max);
    printf("The number of performed iterations is : %d\n",K - 1);
    printf("                                                  \n");
    printf("The computed root of f(x) = 0 is : %lf \n",C);
    printf("                                  \n");
    printf("The accuracy is +-
    %lf\n", B-A); printf("                    \n");
    printf("The value of the function f(C)
    is %lf\n",YC);

}   /* End of main program */
```

Program 2

OBJECTIVES: To find the roots of non linear equations using Newton's method.

Source code:

```
Algorithm 2.5 (Newton-Raphson Iteration). To find a root
 f(x) = 0 given one initial approximation   p_0  and using the iteration

                          f(p_(k-1))
      p_k  =  p_(k-1)  -  ----------      for k = 1, 2, ...
                          f'(p_(k-1))

 ------------------------------------------------------------------------
*/

/* User has to supply a function named :   ffunction
   and its first derivative :              dffunction
   An example is included in this program */

#include<stdio.h>

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

/*  define prototype for USER-SUPPLIED function f(x)    */
    double ffunction(double x);
    double dffunction(double x);


/*  EXAMPLE for "ffunction"    */

    double ffunction(double x)

    {
        return ( pow(x,3) - 3 * x + 2 );
    }

/*  EXAMPLE for "dffunction" , first derivative of ffunction. */

    double dffunction(double x)

    {
        return ( 3 * pow(x,2) - 3 );
    }

/*                                                              */
```

```
/*  Main program for algorithm  2.5  */

    void main()

{
    double Delta = 1E-6;        /* Tolerance  */
    double Epsilon = 1E-6;      /* Tolerance  */
    double Small = 1E-6;        /* Tolerance  */

    int Max = 99; /* Maximum number of iterations */
    int Cond = 0; /* Condition fo loop termination */
    int K;        /* Counter for loop              */

    double P0; /* INPUT : Must be close to the root */
    double P1; /* New iterate */
    double  Y0;  /*  Function  value  */
    double  Y1;  /*  Function  value  */
    double   Df;   /*   Derivative    */
    double Dp;
    double RelErr;


    printf("                                                  \n");
    printf("Please enter initial approximation of root !\n");
    scanf("%lf",&P0);
    printf("                                                  \n");
    printf("Initial value for root: %lf\n",P0);

    Y0 = dffunction(P0);

    for ( K = 1; K <= Max ; K++) {

        if(Cond) break;
        Df = dffunction(P0);       /* Compute the derivative   */

         if( Df == 0) {        /* Check division by zero */

             Cond = 1;
             Dp   = 0;
         }

         else Dp = Y0/Df;

        P1 = P0 - Dp;        /* New iterate */
        Y1 = ffunction(P1);    /* New function value */

        RelErr = 2 * fabs(Dp) / ( fabs(P1) + Small );    /* Relative
error */

        if( (RelErr < Delta)   && (fabs(Y1) < Epsilon) ) { /* Check for
*/

            if( Cond != 1) Cond = 2;                        /*
convergence */

        }
```

```
        P0 = P1;
        Y0 = Y1;
    }

    printf("                                                    \n");
    printf("The current %d -th iterate is %lf\n",K-1, P1);
    printf("Consecutive iterates differ by %lf\n",Dp);
    printf("The value of f(x) is %lf\n",Y1);
    printf("                                                    \n");

if(Cond == 0) printf("The maximum number of iterations was exceeded
!\n");
if(Cond == 1) printf("Division by zero was encountered !\n");
if(Cond == 2) printf("The root was found with the desired tolerance
!\n");

    printf("                                                    \n");

}   /* End of main program */
```


Program 3

OBJECTIVES: Curve fitting by least square approximations

```
Algorithm 5.2 (Least-Squares Polynomial).

 To construct the least-squares polynomial of degree   M  of the form

                                  2        3             M-1             M
 P_M(x)   =  c_1 + c_2*x + c_3*x   + c_4*x   +...+ c_M*x     + c_(M+1)*x

 that fits the N data points (x_1, y_1), ... , (x_N, y_N).

 ------------------------------------------------------------------------
 -----
*/

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

/*                                                                    */

/*  Main program for algorithm 5.2   */

/*  remember : in C the fields begin with element 0   */

#define DMAX   15     /* Maximum degree of polynomial */
#define NMAX   20     /* Maximum number of points     */


void main(void)
{
    extern void FactPiv();

    int R, K, J;                        /*  Loop  counters  */
    double X[NMAX-1], Y[NMAX-1];  /* Points (x,y)  */  double
    A[DMAX][DMAX];                      /* A                */
    double B[DMAX];                     /* B                */
    double C[DMAX];
    double P[2*DMAX];
    int N;                              /* Number of points :      INPUT */
    int M;                              /* Degree of polynomial : INPUT */
    double x, y;
    int p;


    printf("Try the examples on page 277 or 281 of the book !\n");
    printf("                                                  \n");
```

```
    do  /* force proper input */
    {
      printf("Please enter degree of polynomial [Not more than
%d]\n",DMAX);
      scanf("%d", &M);
    } while( M > DMAX);

    printf("                                                \n");
    do  /* force proper input */
    {
      printf("Please enter number of points [Not more than
%d]\n",NMAX);
      scanf("%d", &N);
    } while( N > NMAX);

    printf("You say there are %d                            \n");
    points.\n", N); printf("
    printf("Enter points in pairs like :

    2.4, 4.55:\n");

    for (K = 1; K <= N; K++)
    {

                                          printf("Ente
                                          r %d
                                          st/nd/rd
                                          pair of
                                          points:\n",
                                          K);
                                          scanf("%lf,
                                          %lf",
                                          &X[K-1],
                                          &Y[K-1]);
                                          printf("You entered the pair (x,y) = %lf,
                                          %lf\n", X[K-1], Y[K-
1]);
    }


    /* Zero the array */

    for (R = 1; R <= M+1; R++) B[R-1] = 0;
    /* Compute the column vector */
    for (K = 1; K <= N; K++)
    {
      y = Y[K-1];
      x = X[K-1];
      p = 1;

      for( R = 1; R <= M+1; R++ )
      {
        B[R-1] += y * p;
        p = p*x;
      }
    }

    /* Zero the array */

    for (J = 1; J <= 2*M; J++) P[J] = 0;
```

```
P[0] = N;

/* Compute the sum of powers of x_(K-1) */

for (K = 1; K <= N; K++)
{
```

```
        x = X[K-1];
        p = X[K-1];

        for (J = 1; J <= 2*M; J++)
        {
            P[J] += p;
            p = p * x;
        }
    }

    /* Determine the matrix entries */

    for (R = 1; R <= M+1; R++)
    {
        for( K = 1; K <= M+1; K++) A[R-1][K-1] = P[R+K-2];
    }



    /* Solve the linear system of M + 1 equations : A*C = B
       for the coefficient vector C = (c_1,c_2,..,c_M,c_(M+1)) */


    FactPiv(M+1, A, B);

}   /* end main */

/*                                                              */

void FactPiv(N, A, B)
int N;
double A[DMAX][DMAX];
double *B;
{

    int K, P, C, J;                 /* Loop counters                */
    int    Row[NMAX];               /* Field with row-number        */
    double X[DMAX], Y[DMAX
    ];
    double SUM, DET = 1.0;

    int  T;


    /* Initialize the pointer vector */

    for (J = 1; J<= N; J++) Row[J-1] = J - 1;


    /* Start LU factorization */
    for (P = 1; P <= N - 1; P++)
    {
      /* Find pivot element */
```

```
      for (K = P + 1; K <= N; K++)
      {
        if ( fabs(A[Row[K-1]][P-1]) > fabs(A[Row[P-1]][P-1]) )
        {
        /* Switch the index for the p-1 th pivot row if necessary */
           T        = Row[P-1];
           Row[P-1] = Row[K-1];
           Row[K-1] = T;
           DET      = - DET;
        }

      } /* End of simulated row interchange */
        if (A[Row[P-1]][P-1] == 0)
        {
           printf("The matrix is SINGULAR !\n");
           printf("Cannot use algorithm --> exit\n");
           exit(1);
        }

      /* Multiply the diagonal elements */

      DET = DET * A[Row[P-1]][P-1];

      /* Form multiplier */

      for (K = P + 1; K <= N; K++)
      {
        A[Row[K-1]][P-1]= A[Row[K-1]][P-1] / A[Row[P-1]][P-1];

        /* Eliminate X_(p-1) */

        for (C = P + 1; C <= N + 1; C++)
        {
           A[Row[K-1]][C-1] -= A[Row[K-1]][P-1] * A[Row[P-1]][C-1];
        }
      }

  } /* End of  L*U  factorization routine */


    DET = DET * A[Row[N-1]][N-1];



    /* Start the forward substitution */
    for(K = 1; K <= N; K++) Y[K-1] = B[K-1];
    Y[0] = B[Row[0]];
    for ( K = 2; K <= N; K++)
    {
     SUM =0;
     for ( C = 1; C <= K -1; C++) SUM += A[Row[K-1]][C-1] * Y[C-1];
     Y[K-1] = B[Row[K-1]] - SUM;
    }



    if( A[Row[N-1]][N-1] == 0)
    {
```

```
      printf("The matrix is SINGULAR !\n");
      printf("Cannot use algorithm --> exit\n");
      exit(1);
    }

    /* Start the back substitution */

    X[N-1] = Y[N-1] / A[Row[N-1]][N-1];

    for (K = N - 1; K >= 1; K--)

    {
      SUM = 0;
      for (C = K + 1; C <= N; C++)
      {
           SUM += A[Row[K-1]][C-1] * X[C-1];
      }

      X[K-1] = ( Y[K-1] - SUM) / A[Row[K-1]][K-1];

    }  /* End of back substitution */

    /* Output */

    printf("                                                    :\n");
    printf("The components of the vector with the solutions are:\n");
    for( K = 1; K <= N; K++) printf("X[%d] = %lf\n", K, X[K-1]);

}




/*
  ------------------------------------------------------------------------------
  -----


");

}   /* End of main programm */
```

Program 4

OBJECTIVES: To solve the system of linear equations using

gauss elimination method

Source Code:

```
(Gauss-Elimination-Iteration).

 To solve the linear system   AX = B  by starting with   P_0 = 0
 and generating a sequence   { P_K}that converges to the
 solution   P  (i.e., AP = B).A sufficient condition for the
 method to be applicable is that   A  is diagonally dominant.

 ------------------------------------------------------------------
 -----
*/

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

/*                                                                  */

/*  Main program for algorithm 3.5  */

/*  remember : in C the fields begin with element 0   */

#define N 4

void main(void)
{
       Float a[N][N+1],x[N],t,s;
       Int i,j,k;
       Printf("Enter the element of the   argumented matrix row
wise\n");

       For(i=0;i<N;i++)
       For(j=0;j<N+1;j++)
           Scanf("%f",&a[i][j]);
       For(j=0;j<N-1;j++)
       For(i=j+1;i<N;i++)
       {
           t=a[i][j]/a[j][j];
           for(k=0;k<N+1;k++)
           a[i][k]=a[j][k]*t;
       }
```

Printf("The upper triangular matrix is:\n");

For(i=0;i<N;i++)

```
        For(j=0;j<N+1;j++)
                printf("%8.4f%",a[i][j]);

                Printf("\n");

        For(i=N-1;i>=0;i--)
        {

        s=0;

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

                s+=a[i][j]*x[j];

                x[i]=(a[i][N]-s)/a[i][i]

                ;

        }

        Printf("The solution is :\n");

        For(i=0;i<N;i++)

        Printf("x[%3d]=%7.4f\n", i+1,x[i]);

}
```

Program 5

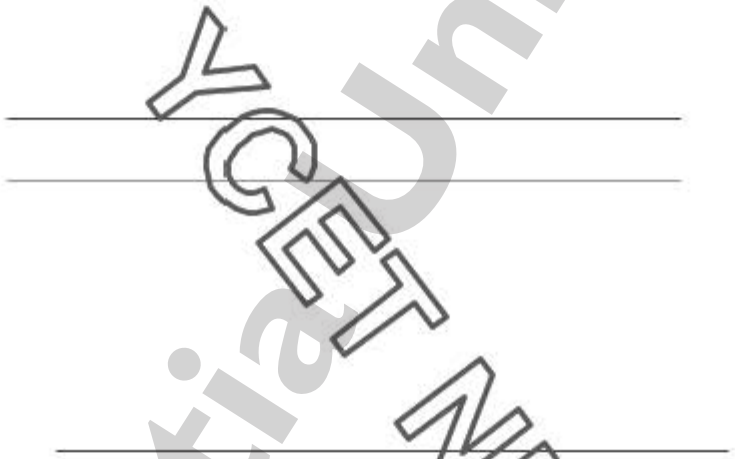
OBJECTIVES: To Solve The System Of Linear Equations Using

Gauss - Seidal Iteration Method

(Code for GAUSS SEIDEL METHOD)

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define ESP 0.0001
#define X1(x2,x3) ((17 - 20*(x2) + 2*(x3))/20)
#define X2(x1,x3) ((-18 - 3*(x1) + (x3))/20)
#define X3(x1,x2) ((25 - 2*(x1) + 3*(x2))/20)


void main()
                                                                {
                                                                \n");
                                                                \n");
```



```
  double
  x1=0,x2=0,x3=0,y1,y2,y3; int
  i=0;
  clrscr(
  );
  printf(
  "\n
  printf("\n   x1\t\t   x2\t\t
  x3\n"); printf("\n
  printf("\n%f\t%f\t%f",x1,x2,x3);
  do
  {
   y1=X1(x2,x3);
   y2=X2(y1,x3);
   y3=X3(y1,y2);

   if(fabs(y1-x1)<ESP && fabs(y2-x2)<ESP && fabs(y3-x3)<ESP )
   {
```

```
    printf("\n                                                    \n");
    printf("\n\nx1 = %.3lf",y1);
    printf("\n\nx2 = %.3lf",y2);
    printf("\n\nx3 = %.3lf",y3);
    i = 1;
}
```

```
   else
   {
     x1 = y1;
     x2 = y2;
     x3 = y3;
     printf("\n%f\t%f\t%f",x1,x2,x3);
   }
  }while(i != 1);
getch();
}

/*

     OUT PUT
        ----------
```

x1	x2	x3
0.000000	0.000000	0.000000
0.850000	-1.027500	1.010875
1.978588	-1.146244	0.880205
2.084265	-1.168629	0.866279
2.105257	-1.172475	0.863603
2.108835	-1.173145	0.863145
2.109460	-1.173262	0.863065
2.109568	-1.173282	0.863051

```
x1 = 2.110

                                                    x2 = -1.173

                                                    x3 = 0.863

                                                    */
```

Program 6

OBJECTIVES: To Solve The System Of Linear Equations Using

Gauss - Jorden Method.

(Code Gauss Jordan method)

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
void main()
{float a[6][6],b[6],x[6],t,s;
int i,j,n,k;
clrscr();
cout<<"Enter the maximum no. of matrix"<<endl;
cin>>n;
cout<<"Enter th elements of matrix"<<endl;
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
cin>>a[i][j];
}
}
cout<<"enter the right constant"<<endl;
for(i=0;i<n;i++)
{
cin>>a[i][n];
}
for(k=0;k<n;k++)
{
for(i=0;i<n;i++)
if(i!=k)
{
for(j=k+1;j<n+1;j++)
{
a[i][j]=a[i][j]-(a[i][k]/a[k][k])*a[k][j];
}}}
cout<<"the solution is"<<endl;
for(i=0;i<n;i++)
{
x[i]=(a[i][n]/a[i][i]);
cout<<"x["<<i<<"]="<<x[i]<<endl;
}
getch();
}
```

Program 7

OBJECTIVES: To integrate numerically using trapezoidal rule.

```
(Trapezoidal Rule).
 Composite Trapezoidal
     b
     /              h                              M-1
     | f(x)dx   =   - * [ f(A) - f(B)]  +  h * sum  f(x_k)
     /              2                              k=1
     a

 by sampling f(x) at the  M + 1  equally spaced points

 x_k = A + h*k  for  k = 0,1,2...,M.

 Notice that  x_0 = A  and  x_M = B.

 ------------------------------------------------------------------------
*/

/* User has to supply a function named : ffunction
   An example is included in this program */

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


/*  define prototype for USER-SUPPLIED function f(x)    */

    double ffunction(double x);


/*  EXAMPLE for "ffunction"    */

    double ffunction(double x)

    {
        return ( 1 / ( 1 + pow(x,2) ) );
    }

/*                                                       */

/*  Main program for algorithm 7.1   */

    void main()

{
    int K;                      /* loop counter                    */
    int M;                      /* INPUT : number of subintervals */
```

```
    double A,B;                 /* INPUT : boundaries of integral */
    double H;                   /* Subinterval width              */
    double SUM = 0;             /*  approx.  integral  value      */
    double X;


    printf("                                                    \n");
    printf("Please enter the boundaries of the integral [A,B]\n");
    printf("EXAMPLE: A = -1 and B = 1, so type: -1 1\n");
    scanf("%lf %lf",&A, &B);
    printf("The boundaries of the integral are: %lf %lf\n",A, B);
    printf("                                    \n");
    printf("Please enter the number of
    SUBINTERVALS.\n"); scanf("%d",&M);

    printf("You say : %d
    subintervals.\n",M);


    H = (B - A)/M;               /*  Subinterval width             */

    for ( K = 1; K <= M-1; K++ ) {
         X = A + H*K;
         SUM = SUM + ffunction(X);
    }

    SUM = H * ( ffunction(A) + ffunction(B) + 2*SUM )/2;


    printf("                                                \n");

    printf(" The approximate value of the integral of f(X)\n");
    printf(" on the interval %lf %lf\n", A,B);
    printf(" using %d subintervals computed using the
trapezoidal\n",M);
    printf(" rule is : %lf\n",SUM);

    printf("                                                \n");

}   /* End of main program */
```

Program 8

OBJECTIVES: To Integrate Numerically Using Simpson's Rules.

(Code for SIMPSON'S 1/3 RULE)

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
  float x[10],y[10],sum=0,h,temp;
  int i,n,j,k=0;
  float fact(int);
  clrscr();
  printf("\nhow many record you will be enter: ");
```

```
  scanf("%d",&n);
  for(i=0; i<n; i++)
  {
   printf("\n\nenter
   the value of x%d:
   ",i);
```

```
   scanf("%f",&x[i]);
   printf("\n\nenter the value of f(x%d): ",i);
   scanf("%f",&y[i]);
  }
  h=x[1]-x[0];
  n=n-1;
  sum = sum + y[0];
  for(i=1;i<n;i++)
  {
    if(k==0)
    {
     sum = sum + 4 * y[i];
     k=1;
    }
    else
    {
     sum = sum + 2 * y[i];
     k=0;
    }
   }
   sum = sum + y[i];
   sum = sum * (h/3);
   printf("\n\n   I = %f  ",sum);
getch();
}

/*
  ------------------------------------------

         OUT PUT

  ------------------------------------------

how many record you will be enter: 5


enter the value of x0: 0


enter the value of f(x0): 1


enter the value of x1: 0.25


enter the value of f(x1): 0.8


enter the value of x2: 0.5


enter the value of f(x2): 0.6667


enter the
value of x3:
0.75
```

```
enter the value of f(x3): 0.5714

enter the value of x4: 1

enter the value of f(x4): 0.5

       I = 0.693250

*/
```

OBJECTIVES: To find the largest eigen value of matrix by

Program 9

power method.

```
 /*To compute the dominant value   Lambda_1  and its associated
 eigenvector  V_1  for the  n x n  matrix  A. It is assumed
 that the  n  eigenvalues have the dominance property

 |Lambda_1| > |Lambda_2| >= |Lambda_3| >= ... >= |Lambda_n| > 0

 ------------------------------------------------------------------------
 -----
*/
```

Source code:

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

#define MaxOrder 50

#define MAX(a,b) a > b ? a : b

/*                                                              */

/*  Main program for algorithm 11.1   */

void main(void)

{
    int i, j;                   /* Loop counter                 */
    int N;                      /* Order of Matrix              */
    double Epsilon = 1E-7;      /* Tolerance                    */
    double Max = 100;           /* Maximum number of iterations */
    double X[MaxOrder], Y[MaxOrder];
    double A[MaxOrder][MaxOrder]; /* Matrix                     */
    double C1, DC, DV, Lambda = 0;
    int Count = 0, Iterating = 1;
    double Sum, Err = 1;
    double MaxElement;


    printf("                Power Method     \n");
    printf("--------- Example 11.5 on page 550              \n");
    printf("                                                \n");

                                        printf("Please enter order of Matrix   A ( <
1);                                     %d !)\n", MaxOrder +

                                        s
                                        c
                                        a
                                        n
                                        f
                                        (
                                        "
                                        %
                                        d
                                        "
                                        ,
                                        &
```

```
N
)
;
i
f
(
N
>
M
```

```
a
x
O
r
d
e
r
)
{
```

```
printf(" Number of steps must be less than %d\n",MaxOrder+1);
printf(" Terminating. Sorry\n");
exit(0);
}
printf("Please enter elements of matrix   A  row by row:\n");

for ( i = 0; i < N; i++)
{
    for ( j = 0; j < N; j++)
    {
                                                printf(" Enter
                                                Element No. %d of
                                                row %d\n", j+1,
                                                i+1);
                                \n");           scanf("%lf",
                                                &A[i][j]);
                                    }           printf("You entered
                                                A[%d][%d]= %lf\n",
                                                i+1, j+1, A[i][j] );
                                                printf("


                                            }

printf("\n");

/* Initialize vector X */

for ( i = 0; i < N; i++)    X[i] = 1.0;

while( (Count <= Max) && (Iterating == 1) )
{
/* Perform Matrix-Vector multiplication */
for ( i = 0; i < N; i++ )
{
    Y[i] = 0;
    for ( j = 0; j < N; j++ ) Y[i] += A[i][j] * X[j];
}

/* Find largest element of vector Y */
/* Do what function MaxElement(X,N) in the book does */

MaxElement = 0;
for ( j = 0; j < N; j++ )
{
    if( fabs(Y[j]) > fabs(MaxElement) ) MaxElement = Y[j];
}

C1 = MaxElement;

DC = fabs(Lambda - C1);

for ( i = 0; i < N; i++) Y[i] *= 1.0/C1;
/* Do what function DIST(X,Y,N) in the book does */

Sum = 0;

for ( i = 0; i < N; i++) Sum += pow( ( Y[i] - X[i] ), 2.0);
```

```
DV  = sqrt(Sum);
Err = MAX(DC,DV);
```

```
    /* Update vector X and scalar Lambda */

    for ( i = 0; i < N; i++) X[i] = Y[i];
    Lambda = C1;

    Iterating = 0;

    if( Err > Epsilon) Iterating = 1;

    Count++;

    } /* End of while loop */


    /* Output vector X and scalar Lambda */

    printf("                                        \n");

    for ( j = 0; j < N; j++) printf("X[%d] = %lf\n", j, X[j]);

    printf("                                        \n");
    printf("Lambda = %lf\n", Lambda);


}   /* End of main program */
```

Program 10

OBJECTIVES: TO FIND NUMERICAL SOLUTION OF ORDINARY

DIFFERENTIAL EQUATIONS BY EULER'S METHOD.

(Code for Program of EULER'S METHOD)

```
#include<stdio.h>
#include  <math.h>
#include<conio.h>
//dy/dx = xy#define F(x,y)    (x)*(y)
void main()
{
  double y1,y2,x1,a,n,h;
  int j;
                                                          clrscr();
```

```
  printf("\nEnter the value of range: ");
  scanf("%lf  %lf",&a,&n);
  printf("\nEnter the value of y1: ");
  scanf("%lf",&y1);
  printf("\n\nEnter the h: ");
  scanf("%lf",&h);
  printf("\n\n   y1 = %.3lf ",y1);
  for(x1=a,j=2; x1<=n+h; x1=x1+h,j++)
  {
   y2= y1 + h * F(x1,y1);
   printf("\n\n   x = %.3lf => y%d = %.3lf ",x1,j,y2);
   y1=y2;
  }
getch();
}

/*
 OUT PUT
```

```
Enter the value of range: 1 1.5

Enter the value of y1: 5

Enter the h: 0.1

  y1 = 5.000

  x = 1.000 => y2 = 5.500

  x = 1.100 => y3 = 6.105

  x = 1.200 => y4 = 6.838

  x = 1.300 => y5 = 7.726
```

```
  x = 1.400 => y6 = 8.808
  x = 1.500 => y7 = 10.129

*/
```

Program 11

OBJECTIVES: To Find Numerical Solution Of Ordinary

Differential Equations By Runge- Kutta Method.

```
/* Runge Kutta for a set of first order differential equations */
#include <stdio.h>
#include <math.h>

#define N 2 /* number of first order equations */
#define dist 0.1 /* stepsize in t*/
#define MAX 30.0 /* max for t */
```

FILE *outpu
/* internal
filename */


```
void runge4(double x, double y[], double step); /* Runge-Kutta function */
double f(double x, double y[], int i); /* function for derivatives */

void main()
{
double t, y[N];
int j;

output=fopen("osc.dat", "w"); /* external filename */

y[0]=1.0; /* initial position */
y[1]=0.0; /* initial velocity */

fprintf(output, "0\t%f\n", y[0]);

for (j=1; j*dist<=MAX ;j++) /* time loop */
```

```
{
t=j*dist;
runge4(t, y, dist);
fprintf(output, "%f\t%f\n", t, y[0]);
}

fclose(output);
}

void runge4(double x, double y[], double step)
{
double h=step/2.0, /* the midpoint */
t1[N], t2[N], t3[N], /* temporary storage arrays */
```

```
k1[N], k2[N], k3[N],k4[N]; /* for Runge-Kutta */
int i;

for (i=0;i<N;i++)
{
t1[i]=y[i]+0.5*(k1[i]=step*f(x,y,i));
}

for (i=0;i<N;i++)
{
t2[i]=y[i]+0.5*(k2[i]=step*f(x+h, t1, i));
}

for (i=0;i<N;i++)
                                                                    {
                                                                    t3[i]=y[i]+
(k3[i]=step*f(x+h, t2, i));
}

for (i=0;i<N;i++)
{
k4[i]= step*f(x+step, t3, i);
}

for (i=0;i<N;i++)
{
y[i]+=(k1[i]+2*k2[i]+2*k3[i]+k4[i])/6.0;
}
}
```

```
double f(double x, double y[], int i)
{

if (i==0)
x=y[1]; /* derivative of first equation */
if (i==1)
x= -0.2*y[1]-y[0]; /* derivative of second equation */

return x;
}
```