NUMERICAL METHODS LAB

MATH-204-F

IV SEMESTER
ELECTRICAL AND ELECTRONICS ENGINEERING

DEPARTMENT OF ELECTRICAL & ELECTRONICS
DRONACHARAY COLLEGE OF ENGINEERING
KHENTAWAS, GURGAON-123506

NUMERICAL METHODS LABIV SEM.

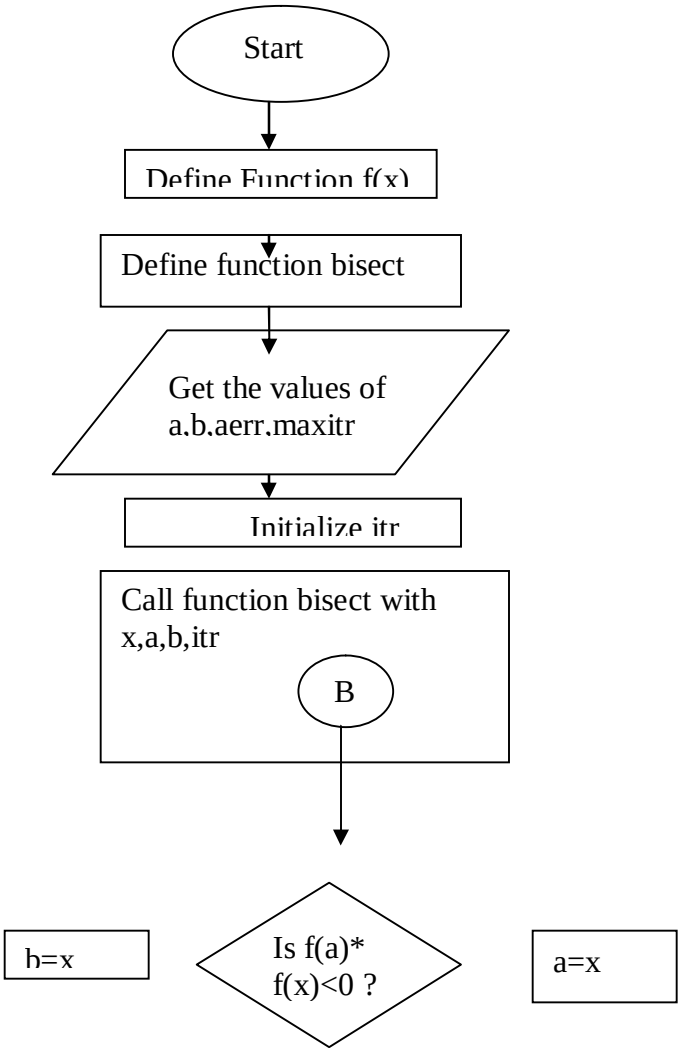
LIST OF EXPERIMENTS

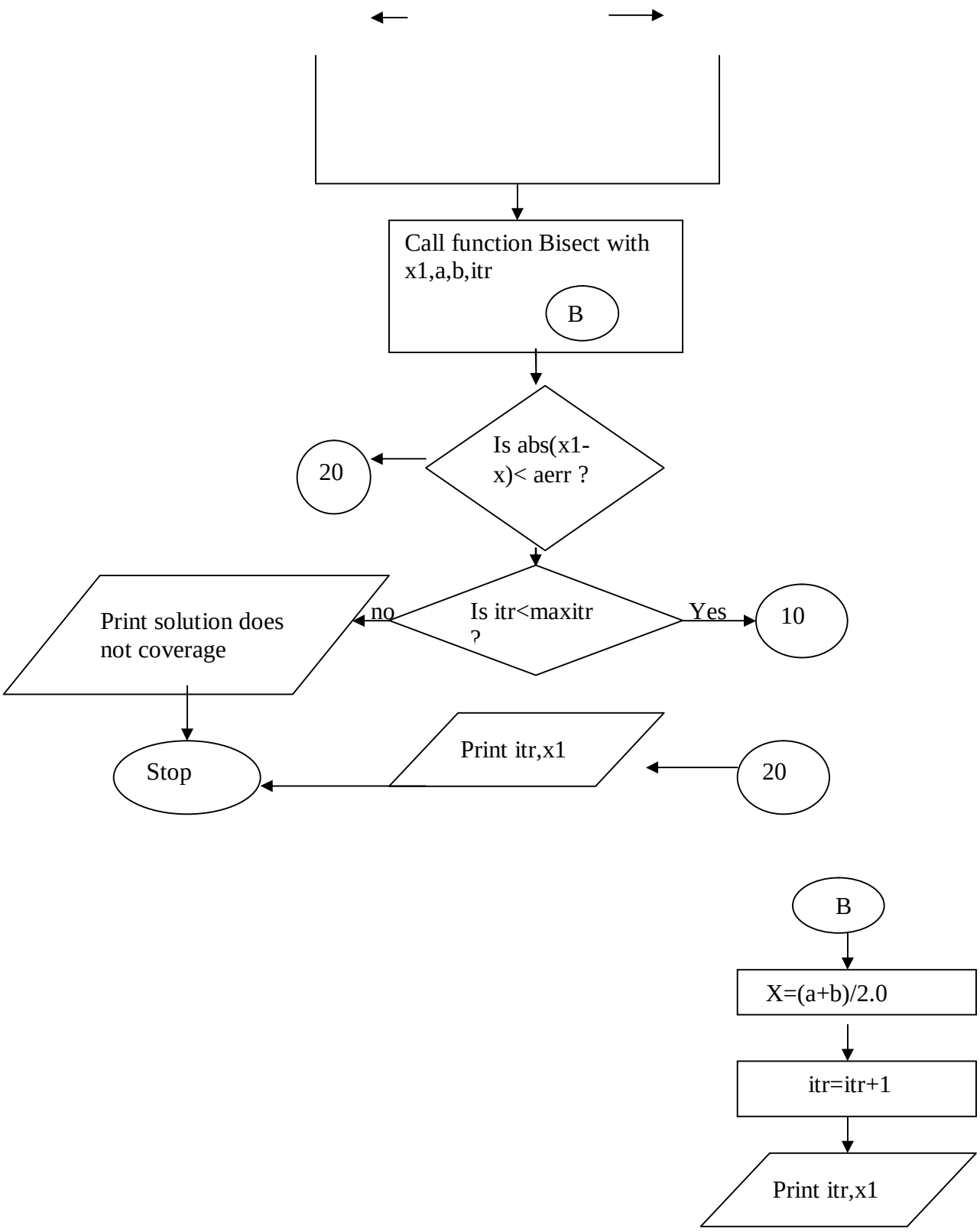
1. TO FIND THE ROOTS OF NON-LINEAR EQUATION USING BISECTION METHOD.
2. TO FIND THE ROOTS OF NON-LINEAR EQUATION USING NEWTON'S METHOD.
3. CURVE FITTING BY LEAST – SQUARE APPROXIMATIONS.
4. TO SOLVE THE SYSTEM OF LINEAR EQUATIONS USING GAUSS - ELIMINATION METHOD.
5. TO SOLVE THE SYSTEM OF LINEAR EQUATIONS 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.
12. TO FIND NUMERICAL SOLUTION OF ORDINARY DIFFERENTIAL EQUATIONS BY MILNE'S METHOD.
13. TO FIND THE NUMERICAL SOLUTION OF LAPLACE EQUATION.
14. TO FIND THE NUMERICAL SOLUTION OF WAVE EQUATION.
15. TO FIND THE NUMERICAL SOLUTION OF HEAT EQUATION.

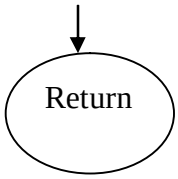
Program 1

OBJECTIVES: To find the roots of non linear equations using Bisection method.

FLOWCHART:




Call function Bisect with x1,a,b,itr
B
Is abs(x1-x)< aerr ?
20
Is itr<maxitr ?
no
Yes
10
Print solution does not coverage
Stop
Print itr,x1
20
B
X=(a+b)/2.0
itr=itr+1
Print itr,x1



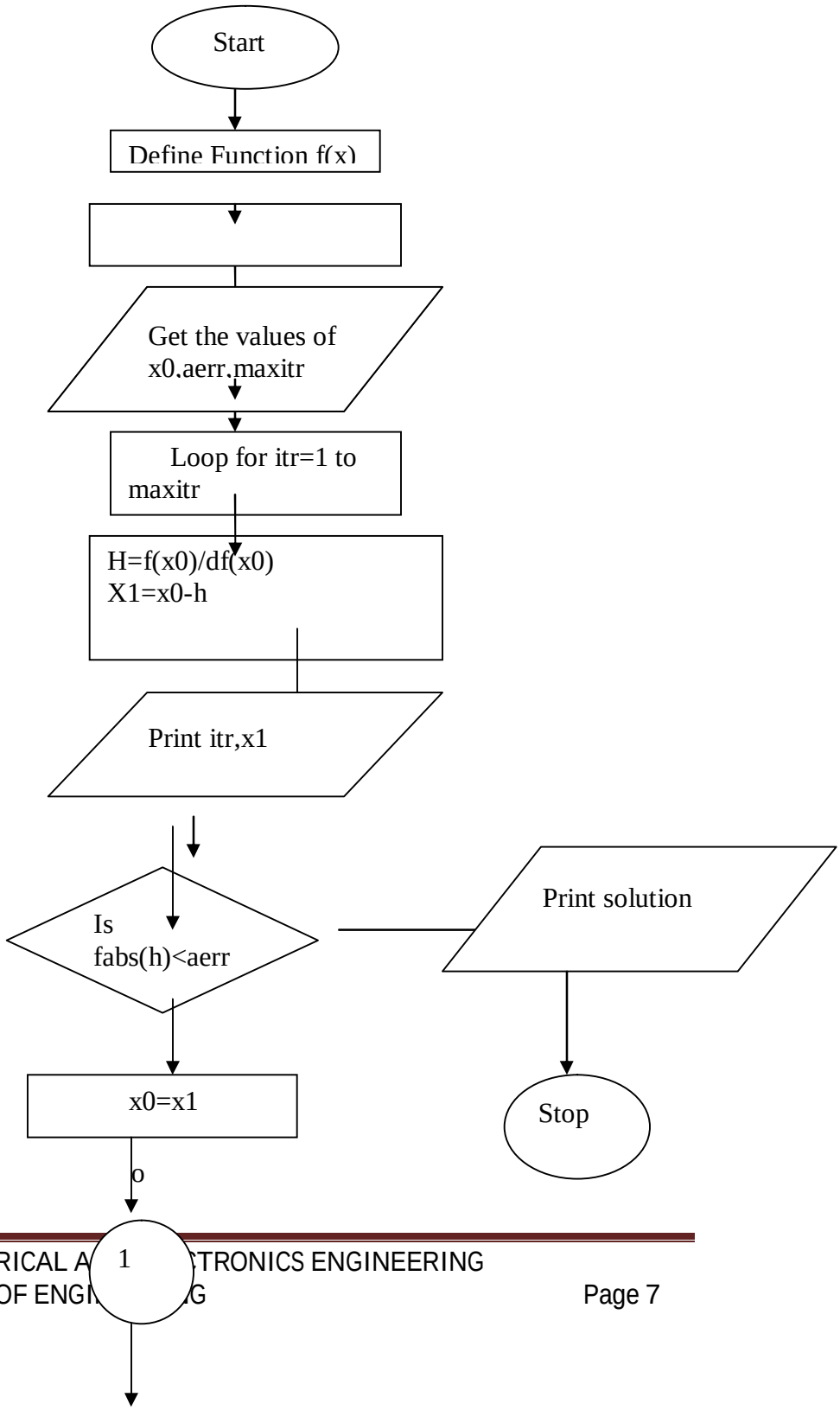
SOURCE CODE:

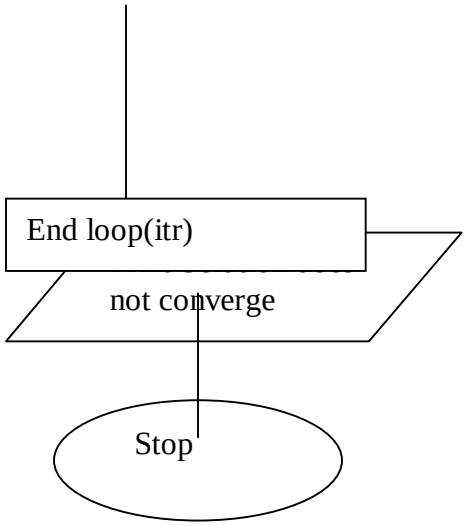
```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>


/*  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 */
```

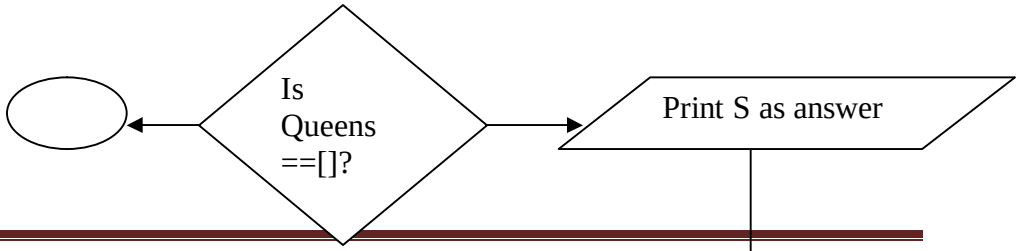
```
    if( ( (YA >= 0) && (YB >=0) ) || ( (YA < 0) && (YB < 0) )   ) {
      printf("The values ffunction(A) and ffunction(B)\n");
      printf("do not differ in sign.\n");
      exit(0);     /* exit program */
    }

    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 */
```

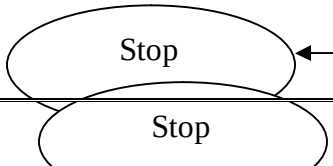
Program 2

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




End loop(itr)
not converge
Stop


Is
Queens
==[]?
Print S as answer


Stop
Stop

Sorce 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()
```

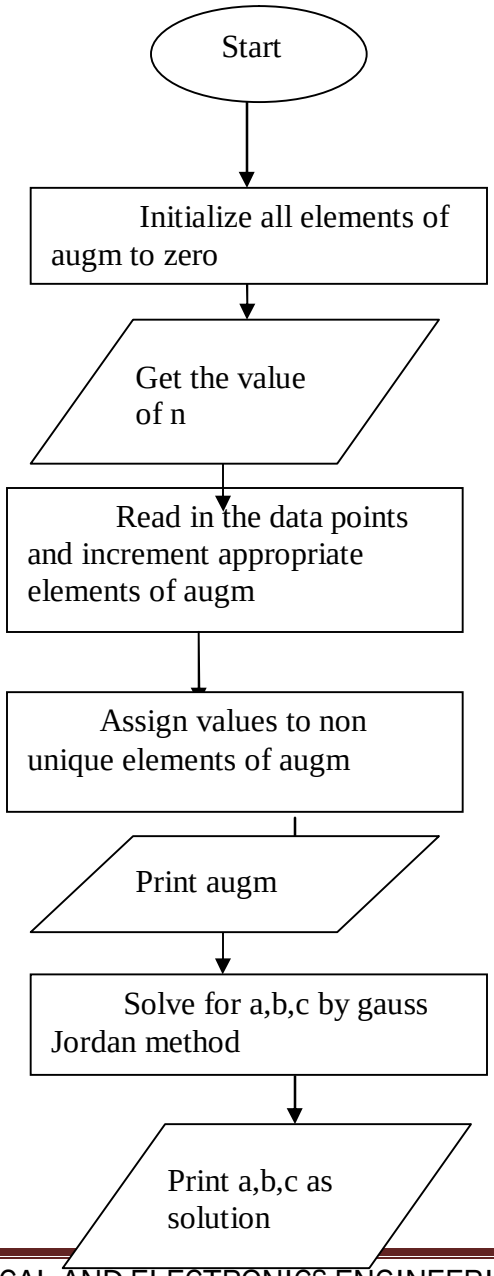
```
{
    double Delta = 1E-6;       /* Tolerance  */
    double Epsilon = 1E-6;     /* Tolerance  */
    double Small = 1E-6;       /* Tolerance  */

    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 */

        }
```

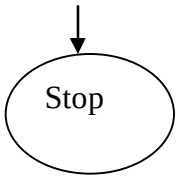
```
        P0 = P1;
        Y0 = Y1;
    }

    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

FLOWCHART:






```
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 points.\n", N);
    printf("-------------------------------------------------------\n");
    printf("Enter points in pairs like :  2.4, 4.55:\n");

    for (K = 1; K <= N; K++)
    {
      printf("Enter %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 */
```

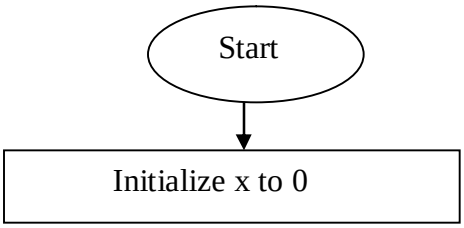
```
    DET = DET * A[Row[N-1]][N-1];


    /* 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]);

}
```

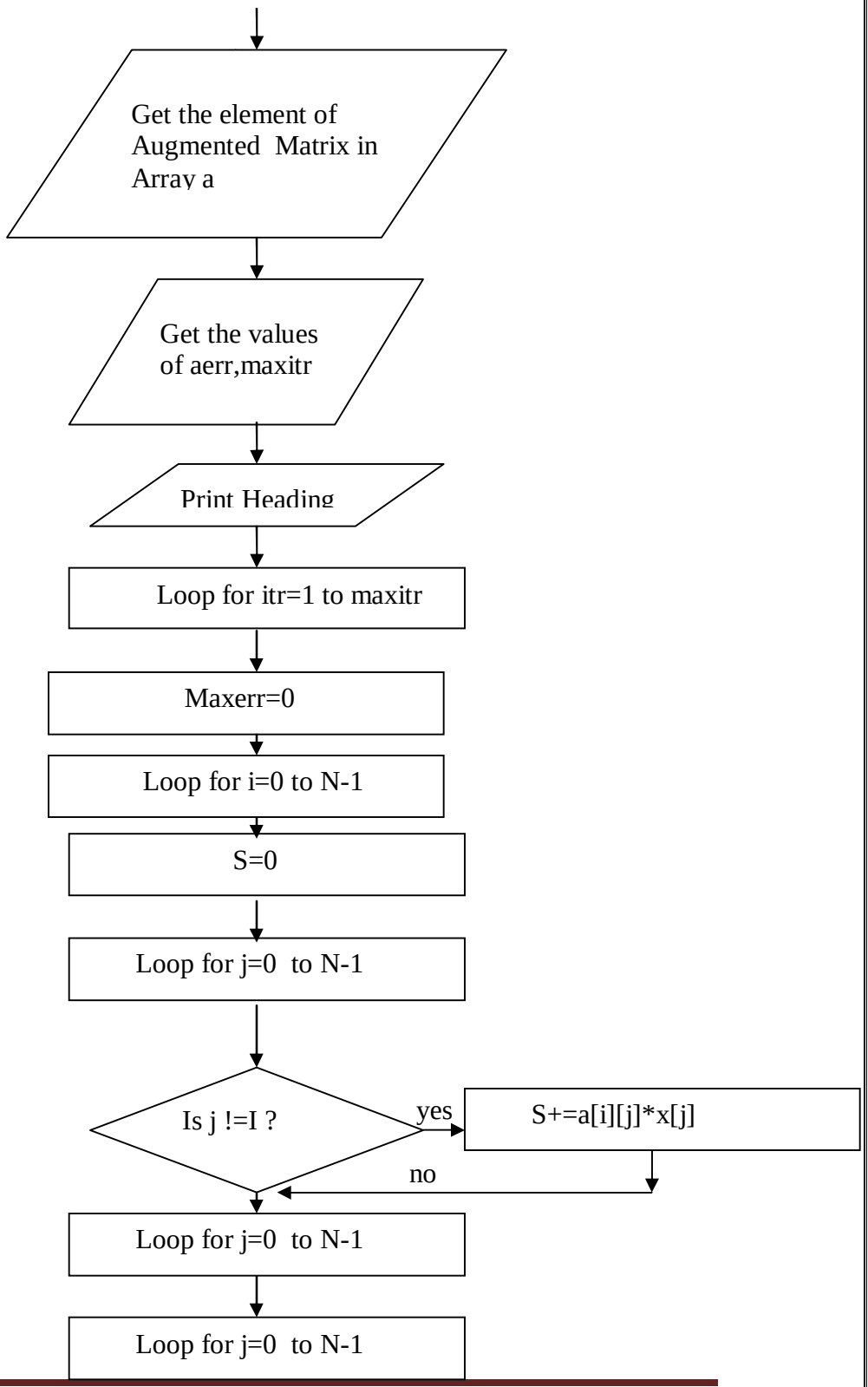
```
/*
-------------------------------------------------------------------------
----

");
}   /* End of main programm */
```

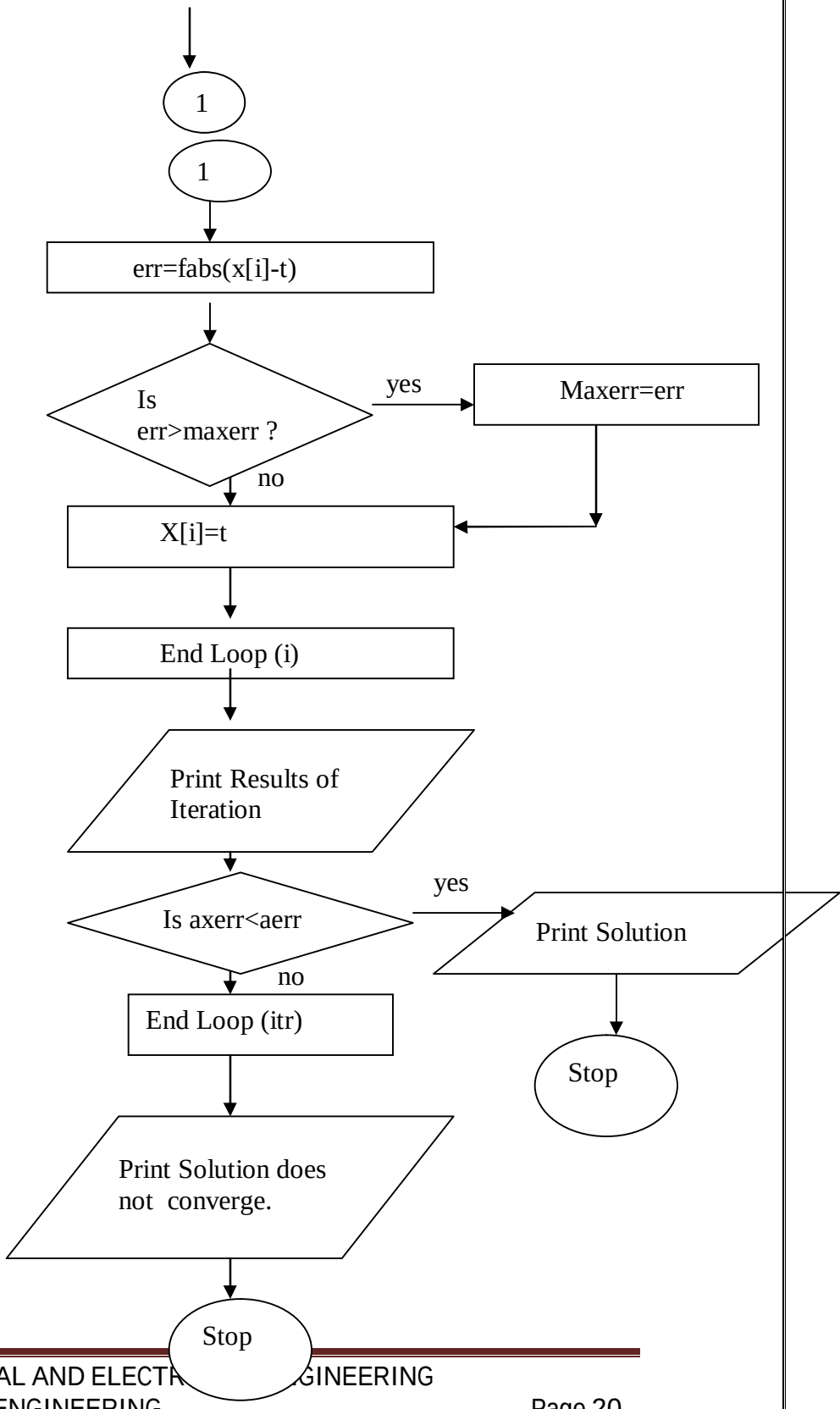
Program 4

OBJECTIVES: To solve the system of linear equations using gauss sedial method

FLOWCHART:




Get the element of Augmented Matrix in Array a
Get the values of aerr,maxitr
Print Heading
Loop for itr=1 to maxitr
Maxerr=0
Loop for i=0 to N-1
S=0
Loop for j=0 to N-1
Is j !=I ?
yes
S+=a[i][j]*x[j]
no
Loop for j=0 to N-1
Loop for j=0 to N-1


1
1
err=fabs(x[i]-t)
Is
err>maxerr ?
yes
Maxerr=err
no
X[i]=t
End Loop (i)
Print Results of
Iteration
Is axerr<aerr
yes
Print Solution
no
End Loop (itr)
Stop
Print Solution does
not converge.
Stop

Source Code:

```
Algorithm 3.5 (Gauss-Seidel-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 Limit  20

void main(void)
{
    double Tol = 10E-6;              /* Tolerance                   */
    double Sep  = 1.0;               /* Initialize                  */
    int K    = 1;                    /* Counter for iterations      */
    int Max  = 99;                   /* Maximum number of iterat.   */
    int Cond = 1;                    /* Condition of matrix         */
    int R, C, J;                     /* Loop counters               */
    double A[Limit][Limit];          /* A in  AX = B ,  INPUT       */
    double B[Limit];                 /* B in  AX = B ,  INPUT       */
    int N;                           /* Dimension of A, INPUT       */
                                     /* = Number of equations       */
    double Row;                      /* Variable in dominance check */
    double P[Limit];
    double Pold[Limit];
    double Sum;

    printf("Try the examples on page 188 of the book !\n");
    printf("-----------------------------------------\n");
    do  /* force proper input */
    {
      printf("Please enter number of equations [Not more than
%d]\n",Limit);
      scanf("%d", &N);
    } while( N > Limit);

    printf("You say there are %d equations.\n", N);
    printf("-------------------------------------------------\n");
```

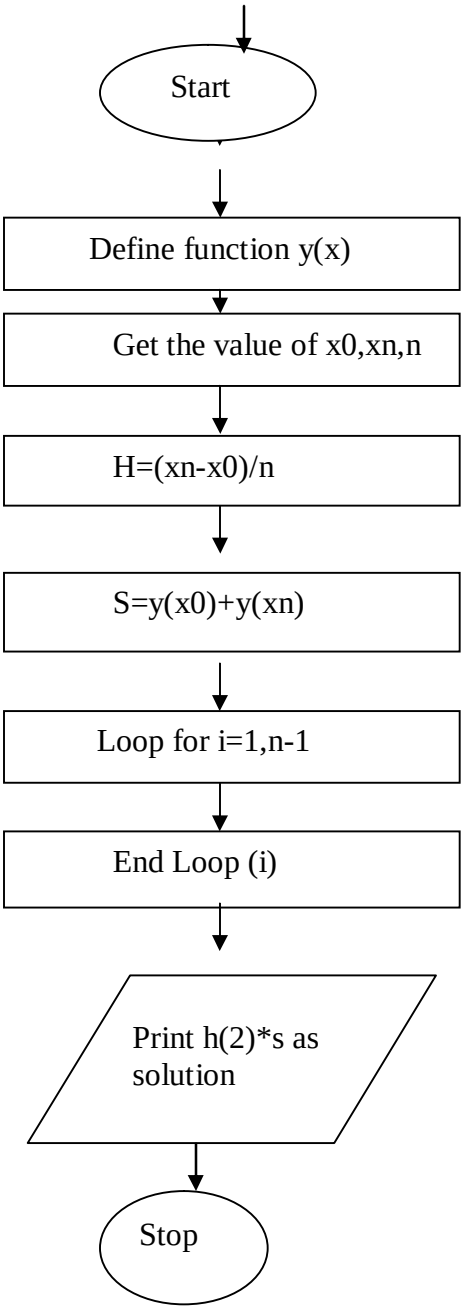
```
    printf("From  AX = B enter components of vector B one by one:\n");

    for (R = 1; R <= N; R++)
    {
        printf("Enter %d st/nd/rd component of vector B\n", R);
        scanf("%lf", &B[R-1]);
    }
    printf("-------------------------------------------------------\n");
    printf("From  AX = B enter elements of A row by row:\n");
    printf("-------------------------------------------------------\n");

    for (R = 1; R <= N; R++)
    {
        for (J = 1; J <= N; J++)
        {
             printf(" For row %d enter element %d please :\n", R, J);
             scanf("%lf", &A[R-1][J-1]);
        }
    }

    /* Check for diagonal dominance. */
    for (R = 1; R <= N; R++)
    {
       Row = 0.0;
       for (C = 1; C <= N; C++) Row += fabs(A[R-1][C-1]);
       if( Row >= 2.0 * fabs(A[R-1][R-1]) ) Cond = 0;
    }

    if( Cond == 0 )
    {
        printf("The matrix is not diagonally dominant.\n");
        printf("Cannot apply this algorithm ---> exit\n");
        exit(1);
    }

    /* Initialize : allow user to do this in order to create a
                    sequence of P_k values  */
    for ( J = 0; J < N; J++)
    {
     P[J]    = 0;
     Pold[J] = 0;
    }

    /* Perform Gauss-Seidel iteration */
    while( (K < Max) && (Sep > Tol) )
    {
     for (R = 1; R <= N; R++)
     {
        Sum = B[R-1];
        for (C = 1; C <= N; C++) if(C != R) Sum -= A[R-1][C-1] * P[C-
1];
        P[R-1] = Sum / A[R-1][R-1];
     }
```

```
     /* Convergence criterion */
     Sep = 0;
     for (J = 1; J <= N; J++) Sep += fabs(P[J-1] - Pold[J-1]);
     /* Update values and increment the counter */
     for (J = 1; J <= N; J++) Pold[J-1] = P[J-1];
     K++;
    }

     /* output */
     if(Sep < Tol) printf("The solution to the linear system is :\n");
     else          printf("Gauss-Seidel iteration did not
converge:\n");

     for (J = 1; J <= N; J++) printf("P[%d] = %lf\n", J, P[J-1]);

}
```

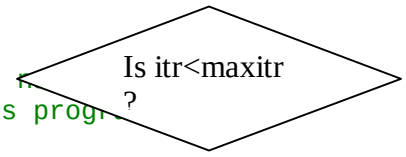
Program 5

OBJECTIVES: To integrate numerically using trapezoidal rule.

FLOWCHART:



```
                                                    (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 f.
   An example is included in this progr

#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()
```



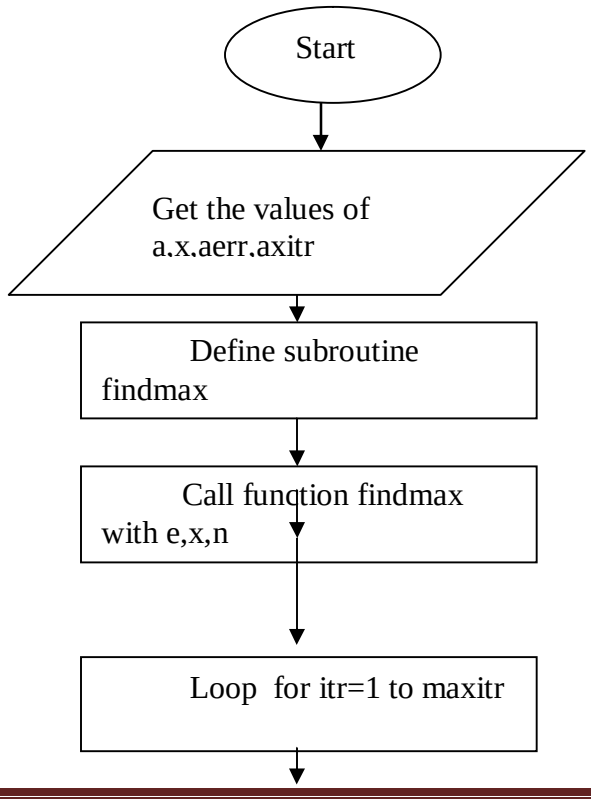
```
{
    int K;                      /* loop counter                    */
    int M;                      /* INPUT : number of subintervals */
    double A,B;                 /* INPUT : boundaries of integral */
    double H;                   /* Subinterval width               */
    double SUM = 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 */
```

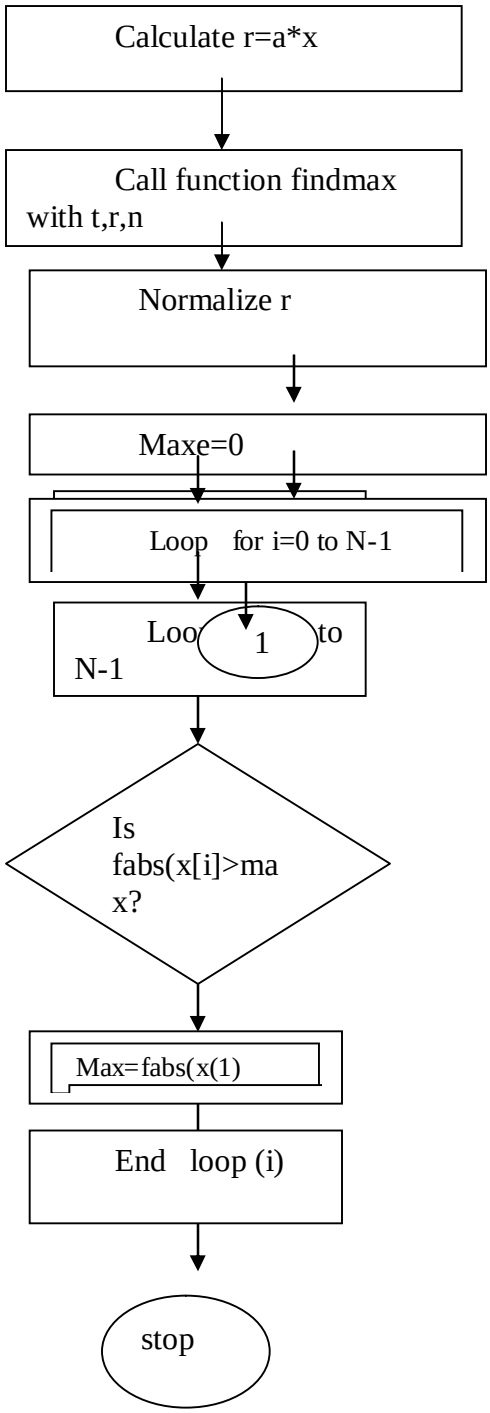
Program 6

OBJECTIVES: To find the largest eigen value of matrix by 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

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

FLOWCHART:




Calculate r=a*x
Call function findmax with t,r,n
Normalize r
Maxe=0
Loop for i=0 to N-1
Loo 1 to N-1
Is fabs(x[i]>ma x?
Max=fabs(x(1)
End loop (i)
stop

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 ( < %d !)\n", MaxOrder +
1);
    scanf("%d",&N);
    if( N > MaxOrder )
    {
    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");
```

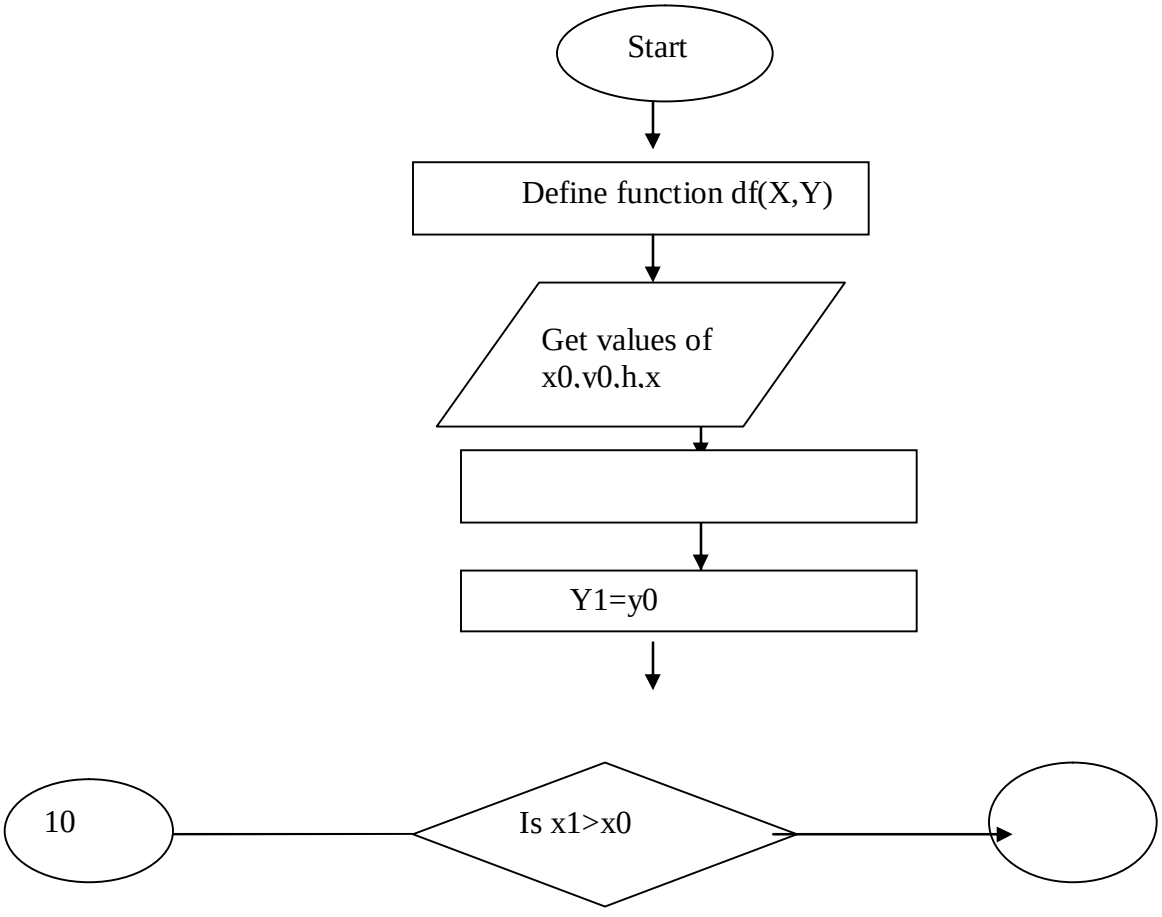
```
    for ( i = 0; i < N; i++)
    {
        for ( j = 0; j < N; j++)
        {
            printf(" Enter Element No. %d of row %d\n", j+1, i+1);
            scanf("%lf", &A[i][j]);
            printf("You entered A[%d][%d]= %lf\n", i+1, j+1, A[i][j] );
            printf("-----------------------------------------------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

    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 */
```

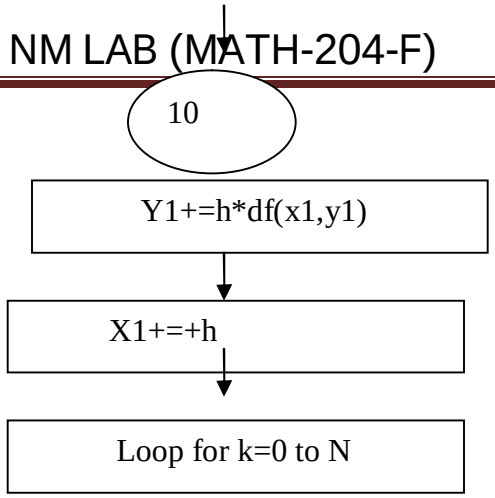
```
    for ( i = 0; i < N; i++) X[i] = Y[i];
    Lambda = C1;

    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 7

OBJECTIVES: To find the numerical solution of ordinary differential equation by power method.

```
To approximate the solution of the initial value problem  y' = f(t,y)
with  y(a) = y_0  over [a,b] by computing

y_(k+1)  =  y_k + h * f(t_k,y_k)   for   k = 0,1,...,M-1.

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

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





Source code:

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

#define MAX 500


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

    double ffunction(double t, double y);


/*  EXAMPLE for "ffunction"   */

    double ffunction(double t, double y)

    {
        return ( (t - y) / 2.0 );
    }

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

/*  Main program for algorithm 9.1  */

    void main(void)

{
```

```
    int K;                      /* Loop counter                    */
    int M;                      /* INPUT: Number of steps          */
    double A, B, Y[MAX];        /* Endpoints and inital value      */
    double H;                   /* Compute the step size           */
    double T[MAX];


    printf("----------- Example on page 433 ------------\n");
    printf("Please enter endpoints of the interval [A,B]:\n");
    printf("For used Example type  : 0, 3\n");
    scanf("%lf, %lf", &A, &B);
    printf("You entered [%lf, %lf]\n", A, B);
    printf("----------------------------------------\n");
    printf("Please enter number of steps: (Not more than 500 !)\n");
    scanf("%d",&M);
    if(M > MAX)
    {
    printf(" Not prepared for more than %d steps. Terminating.
Sorry\n",MAX);
    exit(0);
    }

    printf("----------------------------------------\n");
    printf("Please enter initial value Y[0] :\n");
    printf("For used Example type : 1\n");
    scanf("%lf", &Y[0]);
    printf("You entered Y[0] = %lf\n", Y[0]);

    /* Compute the step size */

    H    = (B - A) / M;

    /* Initialize the variable */

    T[0] = A;

    /* Euler solution Y_(k+1)
       Compute the mesh point T_(K+1) */

    for(K = 0; K <= M-1; K++)
    {
       Y[K+1] = Y[K] + H * ffunction(T[K], Y[K]);
       T[K+1] = A + H * (K+1);
    }

    /* Output */

    for ( K = 0; K <= M; K++)
    {
       printf("K = %d, T[K] = %lf, Y[K] = %lf\n", K, T[K], Y[K]);
    }

}   /* End of main program
```

Program 8

OBJECTIVES:

To find the numerical solution of differential equation using power method.

```
To approximate the solution of the initial value problem  y' = f(t,y)
 with  y(a) = y_0  over [a,b] by using the formula

                        h
 y_(k+1)  =  y_k  +  --- [K_1 + 2*K_2 + 2*K_3 + K_4]
                        6

 User has to supply  functions named : ffunction
 An example is included in this program.

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

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

#define MAX 500


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

    double ffunction(double t, double y);

/*  EXAMPLE for "f1function"   */

    double ffunction(double t, double y)

    {
        return ( (t - y) / 2.0 );
    }

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

/*  Main program for algorithm 9.4  */

    void main(void)

{
    int J;                      /* Loop counter                    */
    int M;                      /* INPUT: Number of steps          */
    double A, B, Y[MAX];        /* Endpoints and inital value      */
    double H;                   /* Compute the step size           */
    double T[MAX];
    double K1, K2, K3, K4;      /* Function values                 */
    double t, y;
```

```
    printf("----------- Taylor's Method ------------------\n");
    printf("--------- Example 9.10 on page 454 ------------\n");
    printf("------------------------------------------------\n");
    printf("Please enter endpoints of the interval [A,B]:\n");
    printf("For used Example type  : 0, 3\n");
    scanf("%lf, %lf", &A, &B);
    printf("You entered [%lf, %lf]\n", A, B);
    printf("----------------------------------------\n");
    printf("Please enter number of steps: (Not more than 500 !)\n");
    scanf("%d",&M);
    if(M > MAX)
    {
    printf(" Not prepared for more than %d steps. Terminating.
Sorry\n",MAX);
    exit(0);
    }

    printf("----------------------------------------\n");
    printf("Please enter initial value Y[0] :\n");
    printf("For used Example type : 1\n");
    scanf("%lf", &Y[0]);
    printf("You entered Y[0] = %lf\n", Y[0]);

    /* Compute the step size */

    H    = (B - A) / M;

    /* Initialize the variable */

    T[0] = A;


    for(J = 0; J <= M-1; J++)
    {
      t  = T[J];
      y  = Y[J];
      K1 = H * ffunction(t,y);
      K2 = H * ffunction(t + H/2.0, y + 0.5 * K1);
      K3 = H * ffunction(t + H/2.0, y + 0.5 * K2);
      K4 = H * ffunction(t + H, y + K3);
      Y[J+1] = y + ( K1 + 2.0 * K2 + 2.0 * K3 + K4 ) / 6.0;
      T[J+1] = A + H * (J+1);
    }

    /* Output */

    for ( J = 0; J <= M; J++)
    {
       printf("J = %d, T[J] = %lf, Y[J] = %lf\n", J, T[J], Y[J]);
    }

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

Program 9

OBJECTIVES:

To find the numerical solution of differential equation using mline method.

```
(Milne-Simpson Method)

 To approximate the solution of the initial value problem  y' = f(t,y)
 with  y(a) = y_0  over [a,b] by using the predictor

                          4h
 p_(k+1)  =  y_(k-3)  +  --- [2*f_(k-2) - f_(k-1) + 2*f_k]
                          3

 and the corrector :

                           h
 y_(k+1)  =  y_(k-1)  +  --- [f_(K-1) + 4*f_k + f_(K+1)].
                           3

 User has to supply  functions named : ffunction
 An example is included in this program.

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

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

#define MAX 500


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

    double ffunction(double t, double y);

/*  EXAMPLE for "ffunction"   */

    double ffunction(double t, double y)

    {
        return ( (t - y) / 2.0 );
    }

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

/*  Main program for algorithm 9.7  */
```

```
void main(void)

{
    int I, K;                   /* Loop counter                    */
    int N;                      /* Number of steps > 3             */
    double A, B, Y[MAX];        /* Endpoints and initial value     */
    double H;                   /* Compute the step size           */
    double T[MAX];
    double F1, F2, F3, F4;      /* Function values                 */
    double Hmin, Hmax;          /* Minimum and maximum step size   */
    double Pold, Yold;          /* Predictor and Corrector         */
    double Pmod, Pnew;          /* Modifier                        */


    printf("------------Milne-Simpson Method----------------\n");
    printf("--------- Example 9.13 on page 468 -------------\n");
    printf("-----------------------------------------------\n");
    printf("Please enter endpoints of the interval [A,B]:\n");
    printf("For used Example type  : 0, 3.0\n");
    scanf("%lf, %lf", &A, &B);
    printf("You entered [%lf, %lf]\n", A, B);
    printf("---------------------------------------\n");
    printf("Please enter number of steps: ( > 3 and < %d !)\n", MAX+1);
    scanf("%d",&N);
    if( (N > MAX) || (N < 4) )
    {
    printf(" Number of steps must be greater than 3 and less than
%d\n",MAX+1);
    printf(" Terminating. Sorry\n");
    exit(0);
    }
    printf("---------------------------------------\n");

    printf("You need all together FOUR initial values. You can\n");
    printf("use the Runge-Kutta method to compute the 2nd, 3rd and\n");
    printf("4th from the 1st one.\n");
    printf("Please enter initial values Y[0], Y[1], Y[2], Y[3] :\n");
    printf("Example 9.13 page 468:  1, 0.94323919, 0.89749071,
0.86208736\n");
    scanf("%lf, %lf, %lf, %lf", &Y[0], &Y[1], &Y[2], &Y[3]);
    printf("You entered Y[0] = %lf\n", Y[0]);
    printf("You entered Y[1] = %lf\n", Y[1]);
    printf("You entered Y[2] = %lf\n", Y[2]);
    printf("You entered Y[3] = %lf\n", Y[3]);

    /* Compute the step size and initialize */

    H    = (B - A) / N;
    T[0] = A;

    for( K = 1; K <= 3; K++) T[K] = A + K * H;

    F1 = ffunction(T[1],Y[1]);
```

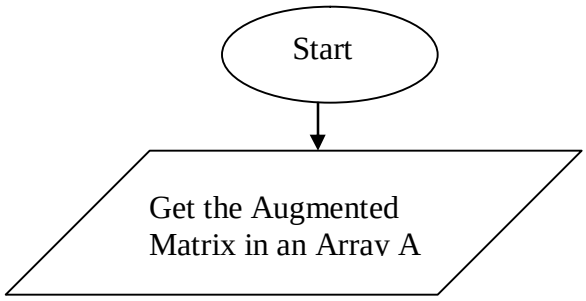
```
    F2 = ffunction(T[2],Y[2]);
    F3 = ffunction(T[3],Y[3]);

    Pold = 0;
    Yold = 0;

    for( K = 3; K <= N-1; K++)
    {
        /* Milne Predictor */
        Pnew = Y[K-3] + 4.0 * H * (2.0*F1 - F2 + 2.0*F3) / 3.0;
        Pmod = Pnew + 28.0 * (Yold - Pold) / 29.0;
        /* Next mesh point */
        T[K+1] = A + H * (K+1);
        /* Evaluate f(t,y) */
        F4 = ffunction(T[K+1],Pmod);
        /* Corrector */
        Y[K+1] = Y[K-1] + H * (F2 + 4.0 * F3 + F4) / 3.0;
        /* Update the values */
        F1 = F2;
        F2 = F3;
        F3 = ffunction(T[K+1], Y[K+1]);
    }

    /* Output */

    for ( K = 0; K <= N; K++)
    {
       printf("K = %d, T[K] = %lf, Y[K] = %lf\n", K, T[K], Y[K]);
    }

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

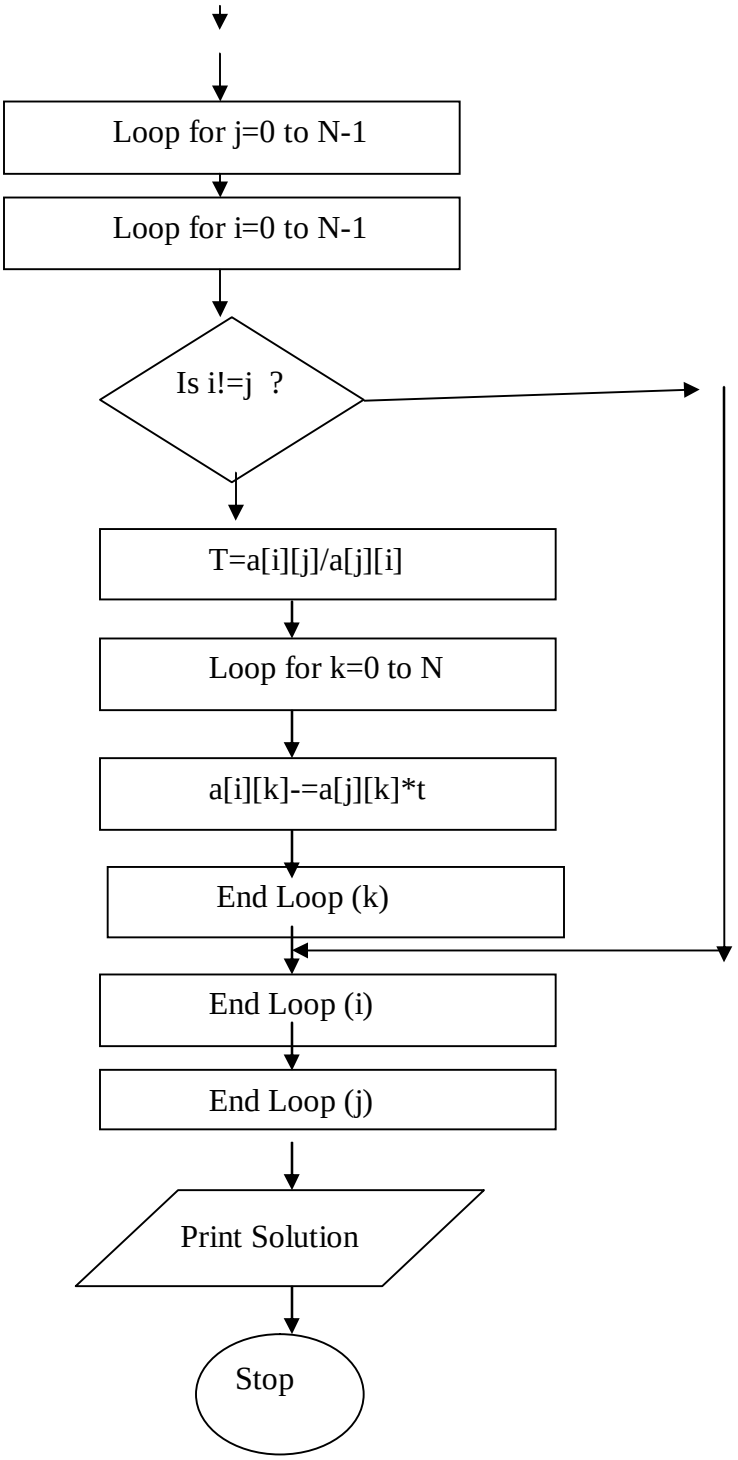
Program 10

OBJECTIVES:

To find the numerical solution of differential equation using laplace equation.

```
(Dirichlet Method for Laplace's Equation)

                                                    2
 To approximate the solution of u_(xx)(x,y) + u_(yy) (x,y)  =  0

 over R = {(x,y): 0 <= x <= a,  0 <= y <= b} with

 u(x,0) = f_1(x),  u(x,b) = f_2(x)  for  0 <= x <= a  and

 u(0,y) = f_3(y),  u(a,y) = f_4(y)  for  0 <= y <= b.

 It is assumed that  Delta(x) = Delta(y) = h  and that integers

 n  and  m  exist so that  a = nh  and  b = mh.

 User has to supply functions  F1, F2, F3, F4  with gridpoints.
 as arguments. An example is given in this program.

------------------------------------------------------------------------
----
*/Flowchart:
```




Loop for j=0 to N-1
Loop for i=0 to N-1
Is i!=j ?
T=a[i][j]/a[j][i]
Loop for k=0 to N
a[i][k]-=a[j][k]*t
End Loop (k)
End Loop (i)
End Loop (j)
Print Solution
Stop

Source code:

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

#define Max 50

/* Global Variables */

double H;                /* Step size  */

/* Prototypes */

double F1i(int i);
double F2i(int i);
double F3i(int i);
double F4i(int i);

/* Grid function for amplitude */

double F1i(int i)
{
    extern double H;
    double arg;

    arg = H * ( i - 1.0 );

    if( (arg >= 0) && (arg <= 4.0) )  return ( 180.0 );
    else
    {
        printf(" F1i() :Argument not in range ! Exiting !\n");
        printf(" arg  : %lf\n", arg);
        exit(0);
    }
}

double F2i(int i)
{
    extern double H;
    double arg;

    arg = H * ( i - 1.0 );

    if( (arg >= 0) && (arg <= 4) ) return ( 20.0 );
    else
    {
        printf(" F2i() :Argument not in range ! Exiting !\n");
        printf(" arg  : %lf\n", arg);
        exit(0);
    }
}
```

```
double F3i(int i)
{
    extern double H;
    double arg;

    arg = H * ( i - 1.0 );

    if( (arg >= 0) && (arg <= 4) ) return ( 80.0 );
    else
    {
        printf(" F3i() :Argument not in range ! Exiting !\n");
        printf(" arg  : %lf\n", arg);
        exit(0);
    }
}

double F4i(int i)
{
    extern double H;
    double arg;

    arg = H * ( i - 1.0 );

    if( (arg >= 0) && (arg <= 4) ) return ( 0.0 );
    else
    {
        printf(" F4i() :Argument not in range ! Exiting !\n");
        printf(" arg  : %lf\n", arg);
        exit(0);
    }
}


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



void main(void)

{
    int i, j;               /* Loop Counters
*/
    double A, B;            /* INPUT : Width and height of Rectangle
*/
    double Ave;             /* INPUT : Initial approximation
*/
    int N, M;               /* dimensions of the grid
*/
    double U[Max][Max];     /* Grid-Amplitudes
*/
```

```
    int Count;              /* counter for while loop
*/
    double Tol;             /* stop criteria
*/
    double w, Pi;
    double H;               /* Grid spacing
*/
    double Relax, temp;

    printf("------------------------------------------------------
\n");
    printf("------- Dirichlet Method for Laplace's Equation -------
\n");
    printf("--------- Try Example 10.6 on page 528/529------------
\n");
    printf("------------------------------------------------------
\n");

    printf("Please enter A (interval boundary of x)\n");
    printf("For present example type  4\n");
    scanf("%lf",&A);
    printf("Please enter B (interval boundary of y)\n");
    printf("For present example type  4\n");
    scanf("%lf",&B);
    printf("You entered A = %lf and B = %lf \n", A, B);
    printf("-------------------------------------------------\n");
    printf("Please enter Grid-Spacing H\n");
    printf("For present example type  0.5\n");
    scanf("%lf",&H);
    printf("You entered H = %lf\n", H);
    printf("-------------------------------------------------\n");
    printf("\n");
    printf("Please enter dimension of grid in x-direction :\n");
    printf("For present example type  9\n");
    scanf("%d",&N);
    printf("Please enter dimension of grid in y-direction :\n");
    printf("For present example type  9\n");
    scanf("%d",&M);
    printf("You entered N = %d and M = %d\n", N, M);
    printf("-------------------------------------------------\n");
    printf("\n");
    printf("Please enter the initial approximation :\n");
    printf("For present example type  70\n");
    scanf("%lf",&Ave);
    printf("You entered = %lf\n", Ave);
    printf("-------------------------------------------------\n");
    printf("\n");

    /* Compute step sizes */

    Pi = 3.1415926535;

    /* Only to make the 0-indexes save we initialize them to zero.
```

```
   This is only because in C fields begin with index 0.
   In this program we ignore the index 0. */

for ( i = 0; i < N; i++ ) U[i][0] = 0.0;
for ( i = 1; i < M; i++ ) U[0][i] = 0.0;

/* Initialize starting values at the interior points */

for ( i = 2; i <= N-1; i++ )
{
    for ( j = 2; j <= M-1; j++ )   U[i][j] = Ave;
}

/* Store boundary values in the solution matrix */

for ( j = 1; j <= M ; j++ )
{
    U[1][j] = F3i(j);
    U[N][j] = F4i(j);
}

for ( i = 1; i <= N ; i++ )
{
    U[i][1] = F1i(i);
    U[i][M] = F2i(i);
}

/* The SQR parameter */

temp =  cos( Pi/(N-1) ) + cos( Pi/(M-1) );

w = 4.0 / ( 2.0 + sqrt( 4.0 - temp * temp ) );

/* Initialize the loop control parameters */

Tol   = 1.0;
Count = 0.0;

while ( (Tol > 0.001) && (Count <= 140) )
{
   Tol = 0.0;
   for ( j = 2; j <= M - 1; j++ )
   {
       for ( i = 2; i <= N - 1; i++ )
       {
         Relax = w * ( U[i][j+1] + U[i][j-1] + U[i+1][j] +
                       U[i-1][j] - 4.0 * U[i][j] ) / 4.0;
         U[i][j] += Relax;
         if( fabs(Relax) > Tol ) Tol = fabs(Relax);
       }
       Count++;
   }
}
```

```
    /* Output the solution */

    for ( j = 1; j <= M; j++ )
    {
        for ( i = 1; i <= N; i++ ) printf("%8.4lf ", U[i][j]);
        printf("\n");
    }

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

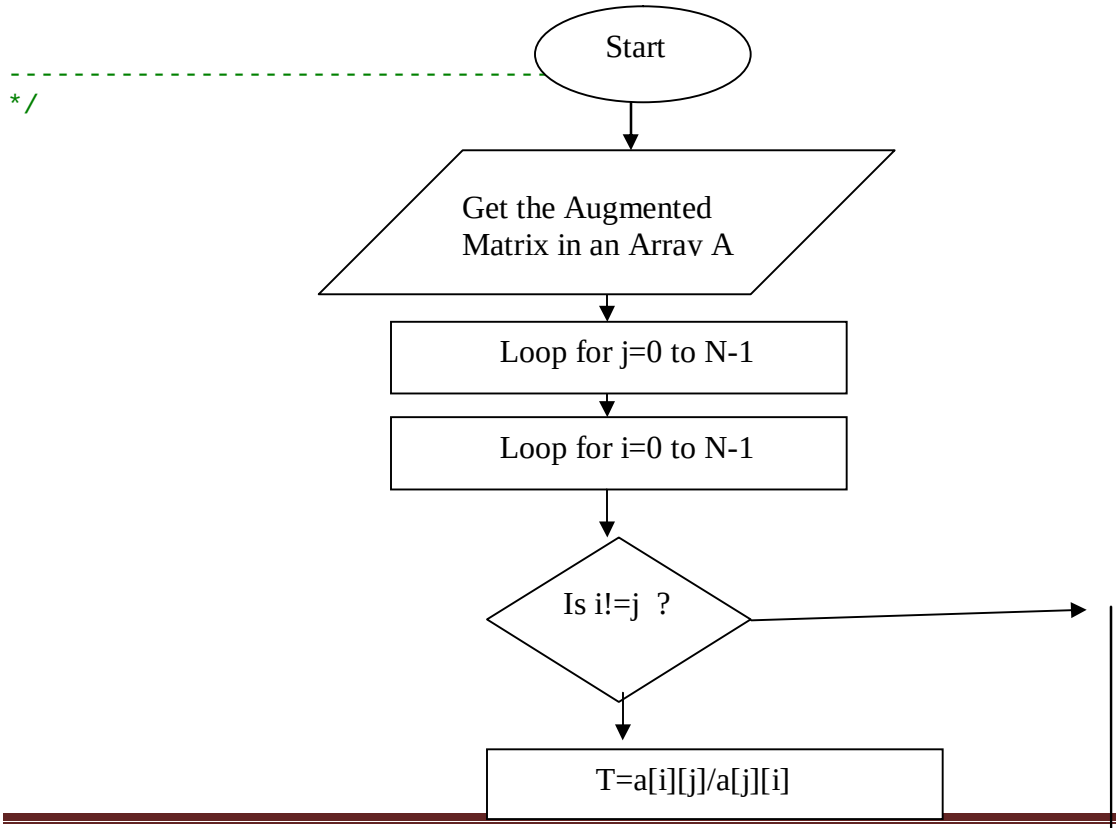
Program 11

OBJECTIVES:

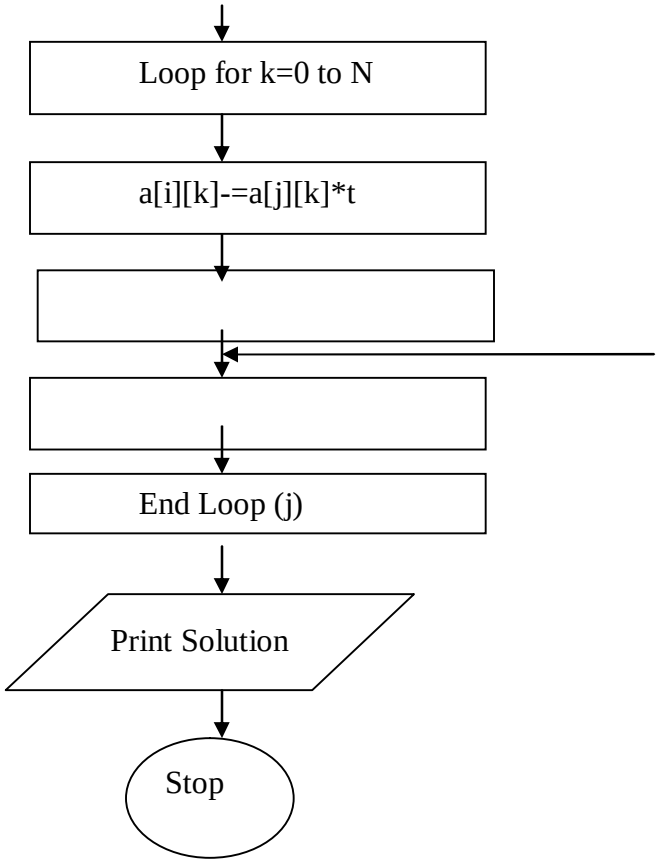
To find the numerical solution of differential equation using wave equation

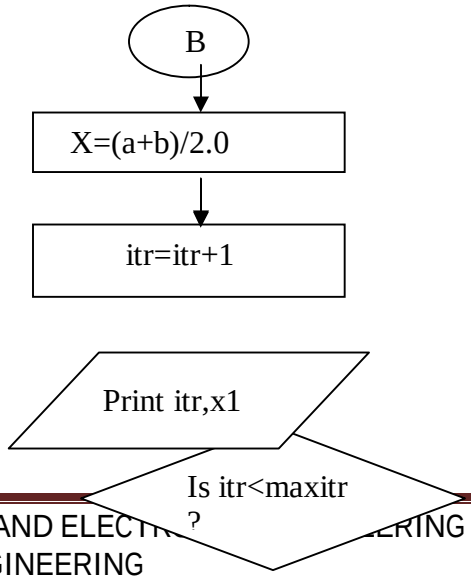
```
(Finite-Difference Solution for the Wave Equation)

                                                    2
To approximate the solution of u_(tt)(x,t)  =  c u_(xx)(x,t)

over  R = {(x,t): 0 <= x <= a, 0 <= t <= b}  with

u(0,t) = 0,       u(a,t) = 0      for  0 <= t <= b and

u(x,0) = f(x),  u_t(x,0) = g(x)   for  0 <= x <= a.

User has to supply function  Fi(gridpoint)  and
Gi(gridpoint):  boundary functions at the grid
points. An examples is implemented in this program.
```

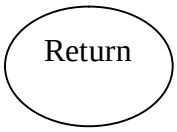
Flowchart:




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


Loop for k=0 to N
a[i][k]-=a[j][k]*t
End Loop (j)
Print Solution
Stop


B
X=(a+b)/2.0
itr=itr+1
Print itr,x1
Is itr<maxitr
?



Souce code:

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

#define Max 50

/* Global Variables */

double H;                /* Step size  */

/* Prototypes */
```

```
double Fi(int i);
double Gi(int i);

/* Grid function for amplitude */

double Fi(int i)
{
    extern double H;
    double arg;

    arg = H * (i - 1);

    if( (arg >= 0) && (arg <= 1.0) )  return ( sin ( 3.1415926 * arg)
);
    else
    {
        printf(" Fi() :Argument not in range ! Exiting !\n");
        printf(" arg  : %lf\n", arg);
        exit(0);
    }
}

/* Grid function for velocity */

double Gi(int i)
{
    extern double H;
    double arg;

    arg = H * (i - 1);

    if( (arg >= 0) && (arg <= 1) ) return ( 0.0 );
    else
    {
        printf(" Gi() :Argument not in range ! Exiting !\n");
        printf(" arg  : %lf\n", arg);
        exit(0);
    }
}

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

/*  Main program for algorithm 10-1  */

void main(void)

{
    int i, j;               /* Loop Counters
*/
    double A, B;            /* INPUT :Width and height of Rectangle
*/
    double C;               /* INPUT : Wave equation const.
*/
```

```
    int N, M;               /* Dimensions of the grid
*/
    double K, R, R2, R22, S1, S2;
    double U[Max][Max];     /* Grid-Amplitudes
*/

    printf("-- Finite-Difference Solution to the Wave Equation ----
\n");
    printf("--------- Example Problem No 4 on page 508 -----------
\n");
    printf("-----------------------------------------------------
\n");

    printf("Please enter A (interval boundary of x)\n");
    printf("For present example type : 1.0\n");
    scanf("%lf",&A);
    printf("Please enter B (interval boundary of t)\n");
    printf("For present example type : 0.5\n");
    scanf("%lf",&B);
    printf("You entered A = %lf and B = %lf \n", A, B);
    printf("-----------------------------------------------------
\n");
    printf("\n");
    printf("Please enter dimension of grid in x-direction :\n");
    printf("For present example type : 6\n");
    scanf("%d",&N);
    printf("Please enter dimension of grid in y-direction :\n");
    printf("For present example type : 6\n");
    scanf("%d",&M);
    printf("You entered N = %d and M = %d\n", N, M);
    printf("-----------------------------------------------------
\n");
    printf("\n");
    printf("Please enter the wave equation constant C :\n");
    printf("For present example type : 2\n");
    scanf("%lf",&C);
    printf("You entered C = %lf\n", C);
    printf("-----------------------------------------------\n");
    printf("\n");

    /* Compute step sizes */

    H   = A / ( N - 1 );
    K   = B / ( M - 1 );
    R   = C * K / H;
    R2  = R * R;
    R22 = R * R / 2.0;
    S1  = 1.0 - R * R;
    S2  = 2.0 - 2.0 * R * R;

    /* Boundary conditions */

    for ( j = 1; j <= M; j++ )
```

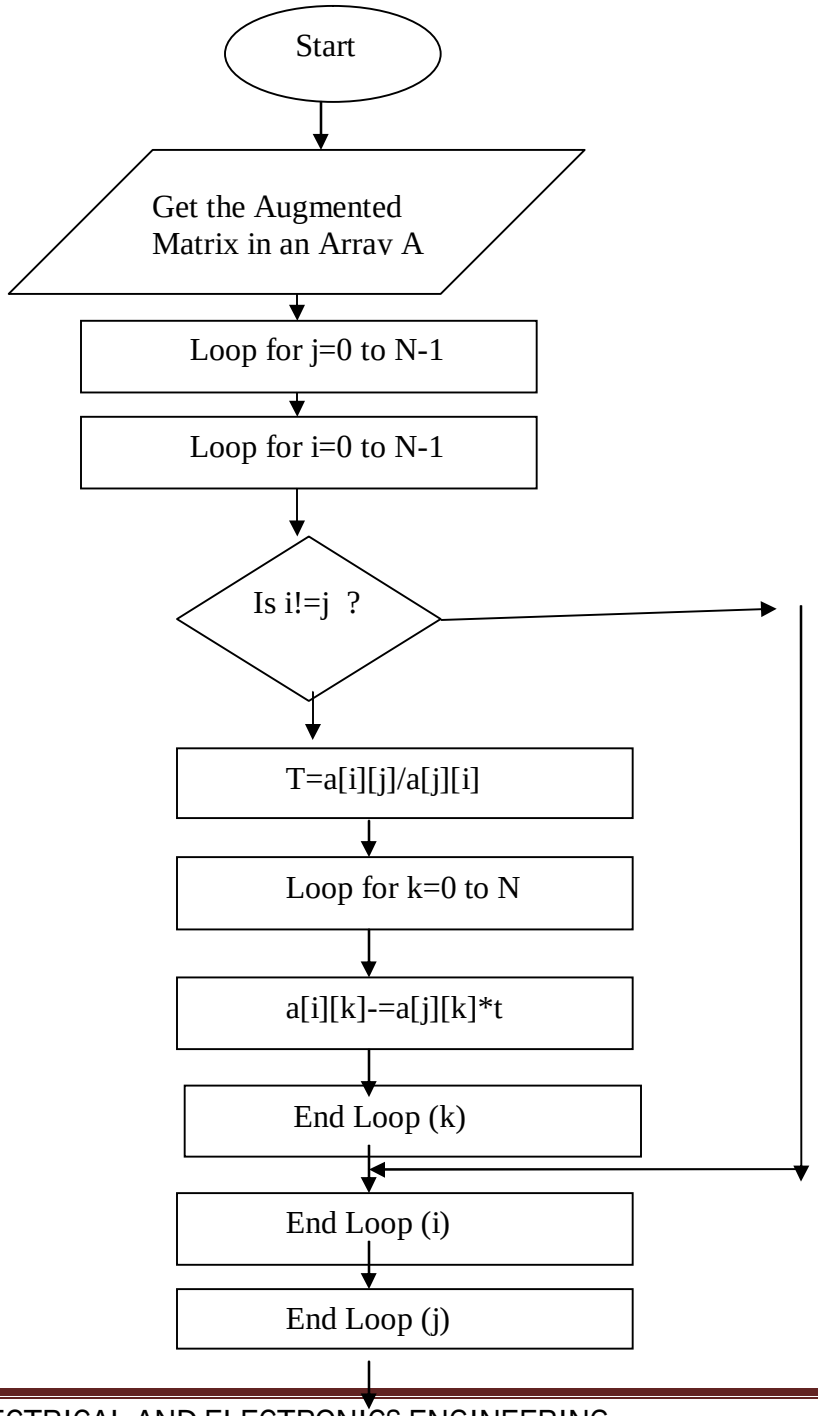
```
    {
        U[1][j] = 0;
        U[N][j] = 0;
    }

    /* First and second rows */

    for ( i = 2; i <= N - 1 ; i++ )
    {
        U[i][1] = Fi(i);
        U[i][2] = S1 * Fi(i) + K * Gi(i) + R22 * ( Fi(i+1) + Fi(i-1) );
    }

    /* Generate new waves */

    for ( j = 3; j <= M; j++ )
    {
        for ( i = 2; i <= N - 1; i++ )
        {
            U[i][j] = S2 * U[i][j-1] + R2 * ( U[i-1][j-1] + U[i+1][j-1]
)
                        - U[i][j-2];
        }
    }

    /* Output the solution */

    for ( j = 1; j <= M; j++ )
    {
        printf("%8.6lf ", K * (j - 1));

        for ( i = 1; i <= N; i++ ) printf(" %8.6lf ", U[i][j]);

        printf("\n");
    }

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

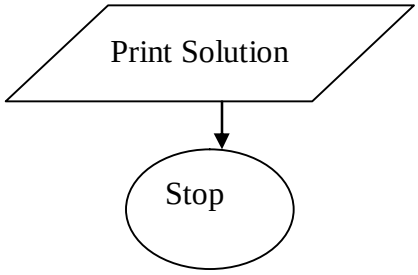
Program 12

OBJECTIVES:

To find the numerical solution of differential equation using heat equation.

Flowchart:





Source code:

```
(Forward-Difference Method for the Heat Equation).
-------------------------------------------------------------------
----
```

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

 (Forward-Difference Method for the Heat Equation)

                                                      2
 To approximate the solution of u_(t)(x,t)  =  c u_(x)(x,t)

 over R = {(x,t): 0 <= x <= a, 0 <= t <= b} with

 u(x,0) = f(x)  for  0 <= x <= a and

 u(0,t) = c_1,  u(a,t) = c_2  for  0 <= t <= b.

 User has to supply function Fi(gridpoint).
 An example is implemented in this program.

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

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

#define Max 50

/* Global Variables */

double H;               /* Step size  */

/* Prototypes */

double Fi(int i);

/* Grid function for amplitude */

double Fi(int i)
{
    extern double H;
    double arg;

    arg = H * (i - 1);

    if( (arg >= 0) && (arg <= 1.0) ) return (4.0 * arg - 4.0 * arg *
arg);
    else
    {
        printf(" Fi() :Argument not in range ! Exiting !\n");
        printf(" arg  : %lf\n", arg);
        exit(0);
```

```
    }
}

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

/*  Main program for algorithm 10-2  */

/*  Remember : in C the fields start with index 0.
    In this program this field is ignored and when you
    change the program be aware of the index 0 in the U[Max][Max]
    array ! Here - as said before - it is ignored, so don't
    access it if you do not initialize it. */

void main(void)

{
    int i, j;               /* Loop Counters
*/
    double A, B;            /* INPUT :Width and height of Rectangle
*/
    double C;               /* INPUT : Wave equation const.
*/
    int N, M;               /* Dimensions of the grid
*/
    double K, S, C1, C2, R;
    double U[Max][Max];     /* Grid-Amplitudes
*/

    printf("-- Forward-Difference Method for the Heat Equation ----
\n");
    printf("--------- Example 10.3 on page 511 --------------------
\n");
    printf("------------------------------------------------------
\n");

    printf("Please enter A (interval boundary of x)\n");
    printf("For present example type : 1.0\n");
    scanf("%lf",&A);
    printf("Please enter B (interval boundary of t)\n");
    printf("For present example type : 0.2\n");
    scanf("%lf",&B);
    printf("You entered A = %lf and B = %lf \n", A, B);
    printf("------------------------------------------------------
\n");
    printf("\n");
    printf("Please enter dimension of grid in x-direction ( < 50
):\n");
    printf("For present example type : 6\n");
    scanf("%d",&N);
    if ( N >= 50 )
    {
       printf("MUST BE SMALLER THAN 50. ... terminating.\n");
       exit(0);
```

```
    }
    printf("Please enter dimension of grid in y-direction ( < 50
):\n");
    printf("For present example type : 11\n");
    scanf("%d",&M);
    if ( M >= 50 )
    {
       printf("MUST BE SMALLER THAN 50. ... terminating.\n");
       exit(0);
    }
    printf("You entered N = %d and M = %d\n", N, M);
    printf("------------------------------------------------------
\n");
    printf("\n");
    printf("Please enter the heat-equation constant C :\n");
    printf("For present example type : 1\n");
    scanf("%lf",&C);
    printf("You entered C = %lf\n", C);
    printf("-------------------------------------------------\n");
    printf("\n");
    printf("Please enter constant C1 = u(0,t) :\n");
    printf("For present example type : 0\n");
    scanf("%lf",&C1);
    printf("You entered C1 = %lf\n", C1);
    printf("-------------------------------------------------\n");
    printf("\n");
    printf("Please enter constant C2 = u(A,t) :\n");
    printf("For present example type : 0\n");
    scanf("%lf",&C2);
    printf("You entered C2 = %lf\n", C2);
    printf("-------------------------------------------------\n");
    printf("\n");


    /* Compute step sizes */

    H   = A / ( N - 1 );
    K   = B / ( M - 1 );
    R   = C * C * K / ( H * H );
    S   = 1.0 - 2.0 * R;

    /* Boundary conditions */

    for ( j = 1; j <= M; j++ )
    {
        U[1][j] = C1;
        U[N][j] = C2;
    }

    /* First row */

    for ( i = 2; i <= N - 1 ; i++ ) U[i][1] = Fi(i);
```

```
    /* Generate new waves */

    for ( j = 2; j <= M; j++ )
    {
        for ( i = 2; i <= N - 1; i++ )
        {
            U[i][j] = S * U[i][j-1] + R * ( U[i-1][j-1] + U[i+1][j-1]
);
        }
    }

    /* Output the solution */

    printf("The grid amplitudes look like this :\n");
    printf("\n");

    for ( j = 1; j <= M; j++ )
    {
        printf("%8.6lf ", K * (j - 1));

        for ( i = 1; i <= N; i++ ) printf(" %8.6lf ", U[i][j]);

        printf("\n");
    }

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

REFRENCES:

1. Numerical methods by B.S.Grewal

2. Numericalk method :E. Balagurusamy T.M.H

NEW IDEAS /EXPERIMENTS

Apart from university syllabus following programs are performed by students to upgrade their knowledge in prolog programming::

1) program to find the roots of non linear equations by Newton ramphson's method
2) program to find the roots of non linear equations by regula falsi method.
3) program to find the multiplication of two matrices.
4) program to find the roots of non linear equations by muller method
5) program to find the root of equations by newton's forward interpolation method

FREQUENTLY ASKED QUESTIONS

Q 1) What are Numerical methods?

Q 2) What is bisection method?

Q 3) What is the equation for bisection method?

Q 4) What is newton's method?

Q 5) What is the equation for bisection method

Q 6) What is least square approximations?

Q 7) What is the equation for gauss elimination method?

Q 8) What is the equation for gauss sedial method?

Q 9) What is the equation for gauss jorden method?

Q10) What is the equation for trapezoidal rule?

Q11) What is the equation for simpson's rule?

Q12) What is the equation for euler's method?

Q13) What is the equation for miline's method?

Q14) What is laplace equation?

Q15)What is wave equation?

Q16)What is heat equation?

Q17) What are the header files required for these programs in c?

Q18)What is the use of getch() function in your program?

Q19)What are functions?

Q20)What is Newton rampson's method?