NUMERICAL PROGRAMMING IN PHYSICS – PRACTICAL

II MSc Physics-III Semester

Course Code: 20PPH3C12P2

PG AND RESEARCH DEPARTMENT OF PHYSICS

JAMAL MOHAMED COLLEGE

College with Potential for Excellence
Reaccredited (3rd Cycle) with 'A' Grade by NAAC
Autonomous and Affiliated to Bharathidasan University
Tiruchirappalli-620 020.

Contents

1. FALSE POSITION METHOD: ROOTS OF A QUADRATIC EQUATION

Aim: To find the real roots of the John Wallis' nonlinear equation using False Position Method.

Theoretical Background: Mathematical models for a wide variety of problems in physics can be formulated into equations of the form

$$f(x) = 0$$

where x and $f(x)$ may be real, complex or vector quantities. The solutions of this equation are called as ***roots*** of the equation.

The equation $y = f(x)$ is called as a linear if $f(x)$ is a linear function, for example

$$y = 3x + 5$$

and is called as nonlinear equation if $f(x)$ is a non-linear function, for example

$$y = x^2 + 1$$

Note:

The Wallis's nonlinear equation is

$$x^3 - 2x - 5 = 0$$

This equation was first solved by John Wallis whose works influenced Issac Newton very much. It has two other roots which are complex. This equation was so famous that it was used as a bench mark for testing new techniques of solving nonlinear equations.

False Position Method:

The False Position Method is an iterative method that starts with two initial guesses of the solution, that is bracketing the solution between the two initial guesses, and then gradually reducing the bracket until the true solution is obtained.

Algorithm:

1. Assume the initial guesses x_0 and x_1
2. Let $x_0 = x_1 - f(x_1) \times \dfrac{x_2 - x_1}{f(x_2) - f(x_1)}$
3. If $f(x_0) \times f(x_1) < 0$

   Set $x_2 = x_0$
4. Otherwise

   Set $x_1 = x_0$
5. Repeat till convergence is reached

The C program `false_position_xx.c` that implements this method is as follows

```
// false_position_xx.c
// To find the real root of Wallis's nonlinear equation
// x^3-2x-5= 0
// Roll No     :
// Date        :

#include<stdio.h>
#include<math.h>
#define EPS 1E-6

float FUN (float); // Function Declaration
void main()
{
     int i;
     float x0,x1,x2,f0,f1,f2;

     printf("\n Enter the value of x0="); // Let x0 = 1.0
     scanf("%f",&x0);
     printf("\n Enter the value of x1="); // Let x1 = 2.0
     scanf("%f",&x1);

     printf("\n-----------------------------------------------\n");
     printf("\n\tx0\tx1\tx2\tf0\tf1  \tf2\n");
     printf("\n-----------------------------------------------\n");
     i=1;
     do {
          f0=FUN(x0); // Function Call
          f1=FUN(x1); // Function Call

          x2=x0-((f0*(x1-x0))/(f1-f0));
          f2=FUN(x2);
          printf("\n %10.5f%10.5f%10.5f%10.5f%10.5f%10.5f\n",
                                          x0,x1,x2,f0,f1,f2);

          if((f0*f2)<EPS)
               x1 = x2;
          else
               x0 = x2;
          i++;

     } while(fabs(f2)>EPS);
     printf("\n-----------------------------------------------\n");
     printf("The solution converges after %5d iterations\n",(i-1));

     printf("\n The root of the Wallis's nonlinear equation by\n");
```

```
     printf("\n  False Position Method is, x=%f\n\n",x2);
}
/*************************Function Definition ******************/
float FUN(float x)
{
     float f;

     f = pow(x,3.0)-2.0*x-5;
     return f;
}



/*****************************************************************/
                         OUTPUT
/*****************************************************************/


/*********************First Run *********************************/

     Enter the value of x0=1

     Enter the value of x1=2

-------------------------------------------------------------------
   x0            x1           x2         f0        f1       f2
-------------------------------------------------------------------

1.000000    2.000000    2.200000   -6.000000   -1.000000    1.248001

1.000000    2.200000    1.993377   -6.000000    1.248001   -1.065963

1.993377    2.200000    2.088561   -1.065963    1.248001   -0.066634

2.088561    2.200000    2.094210   -0.066634    1.248001   -0.003814

2.088561    2.094210    2.094553   -0.066634   -0.003814    0.000014

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

     The solution converges after 6 iterations

     The root of the Wallis's non linear equation by

     False Position Method x=2.094552
```

```
/************************Second Run ******************************/

     Enter the value of x0=1.5

     Enter the value of x1=1.75

     The solution converges after 7 iterations

     The root of the Wallis's non linear equation by

     False Position Method x=2.094552


/*********************** Third Run ******************************/

     Enter the value of x0=0.5

     Enter the value of x1=1.5

     The solution converges after 36 iterations

     The root of the Wallis's non linear equation by

     False Position Method x=2.094552
/*****************************************************************/
```

Result: We find that the real root of the Wallis's equation converges quickly if the initial values for the roots are closer to the true solution.

The true real root of the equation by False Position Method is x = 2.094552

2. NEWTON RAPHSON'S METHOD: ROOTS OF A POLYNOMIAL EQUATION.

Aim: To find the multiple roots of a polynomial equation using Newton-Raphson Method, deflation and synthetic division techniques. Hence to find the fixed points of a logistic map equation.

Theory: We can locate all the real roots of a polynomial equation by repeatedly applying Newton –Raphson Method and deflation and synthetic division. If $f(x)$ is a function and is x_n an estimate of its root, and if h is a small increment ,such that $x_{n+1} = x_n + h$, then by Taylor Expansion we have,

$$f(x_{n+1}) = f(x_n) + f'(x_n)h + f''(x_n)\frac{h^2}{2!} + ...$$

Neglecting higher orders, and if x_{n+1} is a root of $f(x)$, then

$f(x_{n+1}) = 0$,or $0 = f(x_n) + f'(x_n)h + ...$ Therefore, $h = \frac{f(x_n)}{f'(x_n)}$. But $h = x_{n+1} - x_n$ also.

Hence, $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ .Here if $x_{n+1} = x_r$,the root of the polynomial and $x_n = x_0$, the initial guess, then, the Newton Raphson's Algorithm becomes

$$x_r = x_0 - \frac{f(x_n)}{f'(x_n)}$$

A polynomial of degree n can be expressed as $p(x) = (x - x_r)q(x)$ where x_r is a root of the polynomial $p(x)$ and $q(x)$ is a quotient polynomial of degree $n-1$, obtained by *synthetic division*. Here the degree of the polynomial is reduced by this synthetic division. This process is called as *deflation*.

Note: The quotient polynomial can be further deflated till all the roots are obtained.

Algorithm:

1. Obtain the degree and coefficients of a polynomial (n & a)
2. Decide an error estimate for the first root and error criterion
3. Start FOR LOOP while (n>1)
4. Find the root using Newton –Raphson Algorithm

$$x_r = x_0 - \frac{f(x_n)}{f'(x_n)}$$

5. Root (n) = x_r
6. Deflate the polynomial using synthetic division algorithm and make the factor polynomial as the new polynomial of order $n-1$.
7. Set $x_0 = x_r$ (initial value for next root)
8. End FOR LOOP
9. $Root(1) = -\frac{a_0}{a_1}$

10.Stop

Problem : Find the roots of the logistic map equation if the coefficients are given as $a_3 = 2.9, a_3 = -1.9$ and $a_1 = 0$.

Procedure: The logistic map equation depicting population dynamics of a region is given as $x_{n+1} = kx_n(1-x_n) = f(x_n)$, where k is a control parameter. If x* is a solution such that $x_{n+1} = x^*$ and $x_n = x^*$, then the map reduces to a polynomial

$$a_3 x^{*2} + a_2 x^* + a_1 = 0$$

where $a_3 = k, a_3 = 1-k$ and $a_1 = 0$.

Write a C – program to implement the algorithm given above and obtain the fixed points of this equation.

```
// newton.c
// Program to find the multiple roots of a polynomial by
// Newton Raphson Method.
// Roll No     :
// Date   :

#include<stdio.h>
#include<math.h>
#define EPS 1E-6
#define MAXIT 50

void main()
{
     int n,status,i,j;
     float a[11],root[10],x0,xr;

     void deflate (int,float[],float);
     void newton(int,float[],float,int*,float*);

     printf("Enter the degree of polynomial,n=");
     scanf("%d",&n);

     printf("Enter the coefficients,a(1)-to-a(n+1)\n");
     for(i=1;i<=n+1;i++){
          printf("a[%d] = ",i);
          scanf("%f",&a[i]);
     }
     printf("Enter the initial guess of x, x0= ");
     scanf("%f",&x0);
```

```
     LOOP:
     for(i=n;i>=2;i--){
          newton(n,a,x0,&status,&xr);

          if(status==2){
               for(j=n;j>=i+1;j--)
                    printf(" root %d = %f\n",j,root[j]);
               printf("Next root does not converge in \n");
               printf("%d iterations \n",MAXIT);
               goto LOOP;
          }
          root[i]= xr;

          deflate(n,a,xr);
          x0=xr;
     }

     root[1]=-a[1]/a[2];

     printf("the roots of the polynomial are :\n");
     for(i=1;i<=n;i++)
          printf("root %d = %f\n",i,root[i]);
}
void newton(int n,float a[11],float x0,int *status,float *xr)
{
     int i,count;
     float fx,fdx;

     count=1;
     begin:
     fx = a[n+1];
     for(i=n;i>=1;i--)
          fx= fx*x0 + a[i];
          fdx = a[n+1]*n;
          for(i=n;i>=2;i--)
          fdx=fdx *x0 +a[i]*(i-1);
          *xr = x0-fx/fdx;
          if(fabs((*xr-x0)/(*xr))<=EPS){
               *status = 1;
               return;
          }
          if(count < MAXIT){
               x0=*xr;
               count += 1;
               goto begin;
          }
          else{
```

```
                *status = 2;
                return;
           }
}
void deflate(int n,float a[11],float xr)
{
     float b[11];
     int i;

     b[n+1]= 0;
     for(i=n;i>=1;i--)
          b[i]=a[i+1]+xr*b[i+1];

     for(i=1;i<=n+1;i++)
          a[i]=b[i];
}

/*************************************************/
                OUTPUT
/*************************************************/

     Enter the degree of polynomial,n=2

     Enter the coefficients,a(1)-to-a(n+1)

     a[1] = 0
     a[2] = -1.9
     a[3] = 2.9

     Enter the initial guess of x, x0= 0.5

     The roots of the polynomial are :

     root 1 = -0.000000
     root 2 = 0.655172

/*************************************************/
```

Result : The fixed points of the logistic map equation are , $x_1^* = \ldots\ldots\ldots$, $x_{12}^* = \ldots\ldots\ldots$

3. GAUSS ELIMINATION METHOD: APPLICATION TO ELECTRICAL NETWORK.

Aim: To solve a system of linear algebraic equations for an electrical network using simple Gauss Elimination Algorithm.

Theoretical Details: The Gaussian Elimination Method proposes a systematic strategy for reducing the system of equations to upper triangular form using a forward elimination process and then obtaining the values of unknowns (x's) using a back substitution process.

Let a general set of equations in n unknowns be

$$a_{11}x_1 + a_{12}x_2 + ...a_{1n}x_n = b_1$$
$$a_{21}x_1 + a_{22}x_2 + ...a_2x_n = b_2$$
$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$
$$a_{n1}x_1 + a_{n2}x_2 + ...anx_n = b_n$$

The coefficients for the k^{th} – derived system has the form

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} a_{ij}^{(k-1)}$$

where
$$i = k+1 \to n$$
$$j = k+1 \to n$$

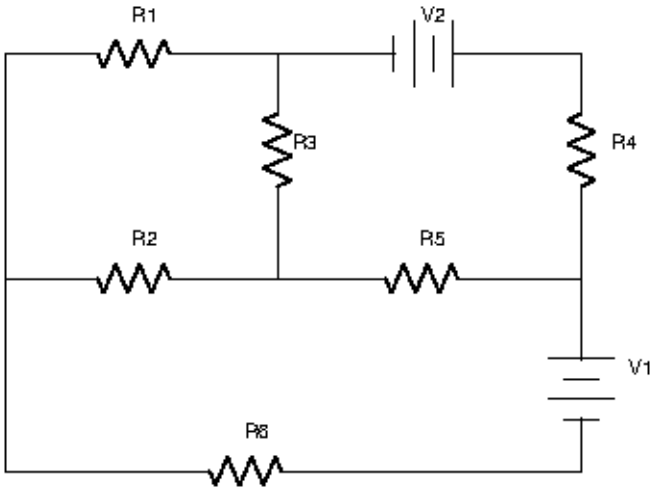
$a_{ij}^{(0)} = a_{ij}$ for i = 1 to n, j = 1 to n. the k^{th} unknown x^k is given by

$$x_k = \frac{1}{a_{kk}^{(k-1)}}\left[b_k^{(k-1)} - \sum_{j=k+1}^{n} a_{kj}^{(k-1)} x_j \right]$$

where k varies from n-1 to 1

$$x_n = \frac{b_n^{(n-1)}}{a_{nn}^{(n-1)}}$$

Problem: For the electrical network shown in figure write down the current / voltage equations in matrix form. Solve these equations using a C-Progam and hence find out put currents i_1, i_2, i_3.



By Ohm's Law, the equations of the circuit are

$$R_6 I_1 + R_5(I_2 - I_1) + R_2(I_1 - I_3) = V_1$$
$$R_4 I_2 + R_3(I_2 - I_3) + R_5(I_2 - I_1) = V_2$$
$$R_1 I_3 + R_2(I_3 - I_1) + R_3(I_3 - I_2) = 0$$

These equations can be rearranged as

$$(R_2 - R_5 + R_6) I_1 + R_5 I_2 - R_2 I_3 = V_1$$
$$-R_5 I_1 + (R_3 + R_4 + R_5) I_2 - R_3 I_3 = V_2$$
$$-R_2 I_1 - R_3 I_2 + (R_1 + R_2 + R_3) I_3 = 0$$

In matrix form, these equations can be written as

$$\begin{bmatrix} (R_2 - R_5 + R_6) & R_5 & -R_2 \\ -R_5 & (R_3 + R_4 + R_5) & -R_3 \\ -R_2 & -R_3 & (R_1 + R_2 + R_3) \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} = \begin{bmatrix} V_1 \\ V_2 \\ V_3 \end{bmatrix}$$

In a general format, the equations are written as

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

The various coefficients a_{11} etc can be found from the values of the resistances given below.

$$R_1 = R_2 = R_3 = 2\Omega$$
$$R_4 = R_5 = R_6 = 3\Omega$$
$$V_1 = V_2 = 5Volts$$

The matrix equation therefore becomes

$$\begin{bmatrix} 2 & 3 & -2 \\ -3 & 8 & -2 \\ -2 & -2 & 6 \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \\ 0 \end{bmatrix}$$

Algorithm :

3. Arrange equations such that $a_{11} \neq 0$.
4. Eliminate x_1 from all but the first equation. This is done as follows :

   (i). Normalise the first equation by dividing it by a_{11}.

   (ii). Subtract from the second equation. a_{21} times the normalized first equation. The result is,

$$\left[a_{21} - a_{21}\frac{a_{11}}{a_{11}}\right]x_1 + \left[a_{22} - a_{21}\frac{a_{12}}{a_{11}}\right]x_2 + \ldots\ldots = b_2 - a_{21}\frac{b_{11}}{a_{11}}$$

We can see that, $a_{21} - a_{21}\frac{a_{11}}{a_{11}} = 0$

Thus, the resultant equation does not contain x_1. The new second equation is, $0 + a'_{22}x_2 + \ldots.. + a'_{2n}x_n = b'_2$

(iii). Similarly, subtract from the third equation. a_{31} times the normalized first equation. The result would be $0 + a'_{32}x_2 + \ldots\ldots + a'_{3n}x_n = b'_3$. If we repeat this procedure till the n^{th} equation is operated on, we will get the following new system of equations:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots\ldots + a_{1n}x_n &= b_1 \\ a'_{22}x_2 + \ldots\ldots + a'_{2n}x_n &= b'_2 \\ \ldots\ldots \quad & \ldots\ldots \\ \ldots\ldots \quad & \ldots\ldots \\ a'_{n2}x_2 + \ldots\ldots + a'_{nn}x_{nn} &= b'_n \end{aligned}$$

The solution of these equations is the same as that of the original equations.

5. Eliminate x_2 from the third to the last equation in the new set. Again, we assume that $a'_{22} \neq 0$.

   (i). Subtract from the third equation a'_{32} times the normalised second equation.

   (ii). Subtract from the fourth equation, a'_{42} times the normalised second equation, and so on.

   This process will continue till the last equation contains only one unknown, namely, x_n.

   The final form of the equations will look like this:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots\ldots + a_{1n}x_n &= b_1 \\ a'_{22}x_2 + \ldots\ldots + a'_{2n}x_n &= b'_2 \\ \ldots\ldots \quad & \ldots\ldots \\ \ldots\ldots \quad & \ldots\ldots \end{aligned}$$

$$a_{nn}^{(n-1)} x_n = b_n^{(n-1)}$$

This process is called triangularisation. The number of primes indicate the number of times the coefficient has been modified.

6. Obtain solution by back substitution. The solution is as follows:

$$x_n = \frac{b_n^{(n-1)}}{a_{nn}^{(n-1)}}$$

This can be substituted back in the $(n-1)^{th}$ equation to obtain the solution for x_{n-1}. This back substitution can be continued till we get the solution for x_1's

A C pogram `gauss_xx.c` to solve a system of linear algebraic equations using simple Gauss Elimination Algorithm is given below.

```
// guass_xx.c
// A C program to solve a set of linear algebraic equations using a
// simple Gauss Elimination method
// Roll No:
// Date   :

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

void main()
{
     float a[20][20],ratio,x[20];
     int i,j,k,n;

     printf("\n Enter the order of the matrix n=");
     scanf("%d",&n);

     printf("\n Enter the coefficients and RHS \n");

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

```
     }
     for(k=1;k<=n-1;k++){
          for(i=k+1;i<=n;i++){
               ratio=a[i][k]/a[k][k];
               for(j=1;j<=n+1;j++)
                    a[i][j]=a[i][j]-ratio*a[k][j];
          }
     }
     x[n]=a[n][n+1]/a[n][n];


     for(k=n-1;k>=1;k--){
          x[k]=a[k][n+1];
          for(j=k+1;j<=n;j++)
               x[k]=x[k]-a[k][j]*x[j];
          x[k]=x[k]/a[k][k];
     }
     printf("\n The current in the circuit are \n");
     for(i=1;i<=n;i++)
          printf("\n I(%d)= %f",i,x[i]);
     printf("\n");

}
 /*******************************************************/
                OUTPUT
/*******************************************************/

     Enter the order of the matrix n = 3
     Enter the coefficients and RHS

     a[1][1]= 2
     a[1][2]= 3
     a[1][3]= -2
     a[1][4]= 5

     a[2][1]= -3
     a[2][2]= 8
     a[2][3]= -2
     a[2][4]= 5

     a[3][1]= -2
     a[3][2]= -2
     a[3][3]= 6
     a[3][4]= 0

     The current in the circuit are
```

```
      I(1)= 1.363636
      I(2)= 1.363636
      I(3)= 0.909091
/*************************************************/
```

Result: A program in C is run to solve a system of simultaneous equation for the physical model of an electrical circuit and the currents in the various branches are obtained.

4. LINEAR LEAST SQUARES FIT METHOD – CHARGE OF AN ELECTRON

Aim : To fit a given set of data, from Millikan's Experiment, to a straight line and find out the charge of the electron and the error in its measurement using using linear least squares fit method.

Theory : To fit a set of data, say

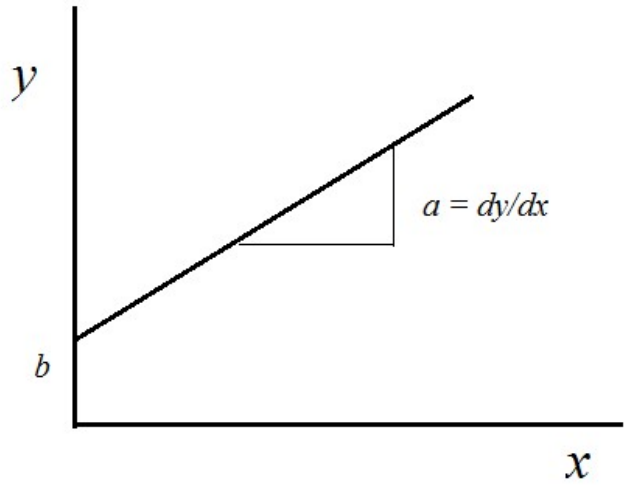
x_i	1	2	3	4	5	6	7
y_i	3	4	5	6	8	10	12

to a straight line y = a*x + b, the normal equations are

$$a = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{n\sum x_i^2 - \left(\sum x_i\right)^2}$$

$$b = \frac{\sum y_i}{n} - a\frac{\sum x_i}{n} \quad \text{(or)} \quad b = \bar{y} - a\bar{x}$$

where $\bar{y}$ & $\bar{x}$ are the mean values of y and x respectively, a = the slope of a straight line and b = y-intercept of the straight line.



Problem : Fit the data from Milikan's Oil Drop Experiment to the linear equation $e_n = n * e_0 + \Delta e$, using least squares fit and (i).Charge of an electron (ii). Error in measurement of charge.

n	4	5	6	7	8	9	10	11
e_n	6.558	8.206	9.88	11.50	13.140	14.82	16.40	18.04

n	12	13	14	15	16	17	18
e_n	19.68	21.32	22.96	24.60	26.24	27.88	29.52

Result :

The given data is fit to a straight line using a C program for linear least squares fit algorithm and the quantities evaluated are

(i). Charge of an electron e= ……. Coulombs.

(ii). Error in measurement of charge, Δe=…… Coulombs.

Algorithm :

1. Read data values.
2. Compute sum of powers and products $\sum x_i, \sum y_i, \sum x_i^2, \sum x_i y_i$.
3. Check whether the denominator of the equation for b is zero.
4. Compute b and a.

5. Print out the equation.
6. Interpolated data, if required.

```
// linreg.c
// Program to fit data to a straight line using linear least square
// fits method
// Roll No:
// Date:

#include<stdio.h>
#include<math.h>
#define EPS 1E-25

float a,b;
void LINREG(int ,float [],float []);

void main()
{
     int i,n;
     float x[20],y[20];

     printf("Enter the number of data points n = ");
     scanf("%d",&n);

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

          printf("\n");

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

          printf("y[%d] = ",i);
          scanf("%f",&y[i]);

          y[i] = y[i]*1E-19;
     }

     LINREG(n,x,y);

     printf("The charge of an electron, e = %12.6e Coulombs\n",a);
     printf("The error in the measurement of charge,");
     printf(" delta[e] =%12.6e Coulombs\n",b);

}
```

```
void LINREG(int n,float x[20],float y [20])
{
     int i;
     float sumx,sumy,sumxx,sumxy;
     float xmean,ymean,denom;

     sumx = 0.0;
     sumy = 0.0;
     sumxx = 0.0;
     sumxy = 0.0;

     for(i=1;i<=n;i++) {
          sumx = sumx + x[i];
          sumy = sumy + y[i];
          sumxx = sumxx + x[i] * x[i];
          sumxy = sumxy + x[i] * y[i];
     }

     xmean = sumx/(float)(n);
     ymean = sumy/(float)(n);
     denom = n*sumxx - sumx*sumx;

     if(fabs(denom)>EPS) {
          a = (n*sumxy - sumx*sumy)/denom;
          b = (ymean - a*xmean);
     }
}


 /*****************************************************/
                          OUTPUT
/*****************************************************/

     Enter the number of Data points n= 15

     x[1]= 4
     y[1]= 6.558

     x[2]= 5
     y[2]= 8.206

     x[3]= 6
     y[3]= 9.88
```

```
x[4]= 7
y[4]= 11.50

x[5]= 8
y[5]= 13.140

x[6]= 9
y[6]= 14.82

x[7]= 10
y[7]= 16.40

x[8]= 11
y[8]= 18.04

x[9]= 12
y[9]= 19.68

x[10]= 13
y[10]= 21.32

x[11]= 14
y[11]= 22.96

x[12]= 15
y[12]= 24.60

x[13]= 16
y[13]= 26.24

x[14]= 17
y[14]= 27.88

x[15]= 18
y[15]= 29.52

The charge of an electron,e=1.638278e-19 Coulombs

The error in the measurement of charge,

delta[e]=2.853786e-21 coulombs


/*************************************************/
```

5. CHARACTERISTIC EQUATION OF A MATRIX: FADEEV-LEVERRIER METHOD

Aim : To generate the characteristic equation of a matrix using Fadeev-Leverrier Method.

Theory: Some boundary value problems may be converted to matrix form and may be expressed as $[A-\lambda I][X]=0$, where I is the identity matrix, A is called as the matrix of coefficients, λ is the eigen value, $[X]$ is the eigen vector and and $[A-\lambda I]$ is called the characteristic matrix of the coefficient matrix. Expansion of the characteristic matrix results in a polynomial of degree n in λ such that

$$\lambda^n + p_1\lambda^{n-1} + p_2\lambda^{n-2} + p_3\lambda^{n-3} + ...p_{n-1}\lambda^1 + p_n = 0$$

If we know the coefficients $p_i's$, then we can construct the characteristic polynomial easily. For this we employ the Fadeev-Leverrier Algorithm. This algorithm consists of generating a sequence of matrices A_i which can be used to determine the values of the coefficients $p_i's$.

Let $A_1 = A$

Then the first coefficient is $p_1 = Tr(A_1)$,

where $Tr(A_1)$ is the trace of the matrix A_1

The other higher order coefficients are found using the recurrence relation

$$A_i = A(A_{i-1} - p_{i-1}I)$$

where $p_i = \frac{Tr(A_i)}{i}$, i=2,3,4,5,.....

Example: Let us consider the system of equations as

$$-x_1 = 0$$
$$x_1 - 2x_2 + 3x_3 = 0$$
$$2x_2 - 3x_3 = 0$$

From this the matrix A can be written as

$$A = \begin{bmatrix} -1 & 0 & 0 \\ 1 & -2 & 3 \\ 0 & 2 & -3 \end{bmatrix}$$

Let $A_1 = A$

Then $p_1 = TrA_1 = -6$

$$A_2 = A(A_1 - p_i I)$$

$$= \begin{bmatrix} -1 & 0 & 0 \\ 1 & -2 & 3 \\ 0 & 2 & -3 \end{bmatrix} \left\{ \begin{bmatrix} -1 & 0 & 0 \\ 1 & -2 & 3 \\ 0 & 2 & -3 \end{bmatrix} - \begin{bmatrix} -6 & 0 & 0 \\ 0 & -6 & 0 \\ 0 & 0 & -6 \end{bmatrix} \right\}$$

$$= \begin{bmatrix} -1 & 0 & 0 \\ 1 & -2 & 3 \\ 0 & 2 & -3 \end{bmatrix} \begin{bmatrix} 5 & 0 & 0 \\ 1 & 4 & 3 \\ 0 & 2 & 3 \end{bmatrix}$$

$$= \begin{bmatrix} -5 & 0 & 0 \\ 3 & -2 & 3 \\ 2 & 2 & -3 \end{bmatrix}$$

Then $$P_2 = \frac{TrA_2}{2} = -5$$

Similarly $$A_3 = A(A_2 - p_2 I)$$

$$= \begin{bmatrix} -1 & 0 & 0 \\ 1 & -2 & 3 \\ 0 & 2 & -3 \end{bmatrix} \left\{ \begin{bmatrix} -5 & 0 & 0 \\ -3 & -2 & 3 \\ 2 & 2 & -3 \end{bmatrix} - \begin{bmatrix} -5 & 0 & 0 \\ 0 & -5 & 0 \\ 0 & 0 & -5 \end{bmatrix} \right\}$$

$$= \begin{bmatrix} -1 & 0 & 0 \\ 1 & -2 & 3 \\ 0 & 2 & -3 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ -3 & 3 & 3 \\ 2 & 2 & 2 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 0 \\ 6 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Then $$P_3 = \frac{TrA_3}{3} = 0$$

Therefore the characteristic polynomial is $\lambda^3 + 6\lambda^2 + 5\lambda = 0$

Fadeev-Leverrier Algorithm

1. Read the order of the matrix A, n
2. Read the elements of the matrix A
3. Find p_1 as the trace of matrix A
4. Begin for Loop (i=2,i<=n;i++)
5. Calculate the matrices A_i using the recurrence relation $A_i = A(A_{i-1} - p_{i-1} I)$
6. Calculate the coefficients p_i using the relation $p_i = \frac{TrA_i}{i}$

The C program to implement this algorithm is given as follows

```
// charac_eqn_xx.c
// C Program to write down the characteristic equation of a matrix
// using Fadeev-Leverrier Method
// Roll No     :
// Date        :

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

double TRACE_MATRIX(double [10][10],int );
void CHANGE_MATRIX(double [10][10], double [10][10], int);
void SUBTRACT_MATRIX(double [10][10], double [10][10], int, double);
void MUL_MATRIX(double [10][10], double [10][10], double [10][10],int);

main()
{
     int i,j,n;
     double A[10][10],B[10][10],AC[10][10], C[10][10];
     double P[10],Q;
     printf("\n Enter the order of the matrix A, n = ");
     scanf("%d",&n);

     printf("\n Enter the elements of the matrix A \n");

     for(i=1;i<=n;i++){
          for(j=1;j<=n;j++){
               printf("A[%2d][%2d] = ",i,j);
               scanf("%lf",&A[i][j]);
          }
          printf("\n");
     }
     Q = 0.0;
     SUBTRACT_MATRIX(A,B,n,Q);

     Q = TRACE_MATRIX(A,n);
     P[1] = Q;

     for(i=2;i<=n;i++){
          SUBTRACT_MATRIX(B,C,n,Q);
          MUL_MATRIX(A,C,AC,n);
          Q = TRACE_MATRIX(AC,n)/(double)(i);
          CHANGE_MATRIX(AC,B,n);
          P[i]=Q;
```

```
	}

	printf("The Characteristic Equation is \n\n");
	for(i=1;i<=n;i++){
		if(i==1)
			printf("S^%d - P%dS^%d - ",n,i,n-1);
		else
		if((i>1)&&(i<n))
			printf("P%dS^%d - ",i,n-2);
		else
		if(i==n)
			printf("P%d = 0",n);
	}
	printf("\n\n");

	printf("The Coeffecients are \n\n");
	for(i=1;i<=n;i++)
		printf("P%d = %3.1lf\t",i,P[i]);
	printf("\n\n");
}
double TRACE_MATRIX(double M[10][10], int n)
{
	int i,sum;

	sum = 0;
	for(i=1;i<=n;i++)
		sum += M[i][i];
	return sum;
}

void SUBTRACT_MATRIX(double M[10][10], double N[10][10], int n,double S)
{
	int i,j;

	for(i=1;i<=n;i++){
		for(j=1;j<=n;j++){
			if(i==j)
				N[i][j] = M[i][j]-S;
			else
				N[i][j] = M[i][j];
		}
	}
}
void CHANGE_MATRIX(double M[10][10], double N[10][10], int n)
{
	int i,j;
```

```
     for(i=1;i<=n;i++){
          for(j=1;j<=n;j++)
               N[i][j] = M[i][j];
     }
}
void MUL_MATRIX(double M[10][10],double
N[10][10],double,U[10][10],int n)
{
     int i,j,k;

     for(i=1;i<=n;i++){
          for(j=1;j<=n;j++){
               U[i][j] = 0.0;
               for(k=1;k<=n;k++)
                    U[i][j] += M[i][k]*N[k][j];
          }
     }
}


/*********************************************/
                OUT PUT
/*********************************************/

Enter the order of the matrix A, n = 3

Enter the elements of the matrix A
A[ 1][ 1] = -1
A[ 1][ 2] = 0
A[ 1][ 3] = 0

A[ 2][ 1] = 1
A[ 2][ 2] = -2
A[ 2][ 3] = 3

A[ 3][ 1] = 0
A[ 3][ 2] = 2
A[ 3][ 3] = -3

The Characteristic Equation is

S^3 - P1S^2 - P2S^1 - P3 = 0

The Coeffecients are

P1 = -6.0 P2 = -5.0 P3 = 0.0
```

/**/

6. DETERIMINANT OF A MATRIX USING LU DECOMPOSITION METHOD

Aim: To find the determinant of the coefficient matrix of a system of linear algebraic equations using LU Decomposition Method.

Theoretical Details: The LU Decomposition Method is a Triangular Factorization Method of solving a set of linear algebraic equations. Let a general set of equations in n unknowns be given as

$$a_{11}x_1 + a_{12}x_2 + ...a_{1n}x_n = b_1$$
$$a_{21}x_1 + a_{22}x_2 + ...a_2x_n = b_2$$
$$.....$$
$$a_{n1}x_1 + a_{n2}x_2 + ...anx_n = b_n$$

In matrix form this set of equations can be represented as

$$Ax = b \quad \text{……..(1)}$$

where A is a $n \times n$ matrix, b is a n column vector and x is a vector of n unknowns.

Here the matrix A is factorized into two triangular matrices, L the lower triangular matrix which contains upper half off-diagonal elements as zeros and U the upper triangular matrix which contains lower half off-diagonal elements as zeros, such that

$$A = LU \quad \text{……..……..(2)}$$

where $$L = \begin{bmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} \end{bmatrix} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (3)$$

and $$U = \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{bmatrix} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (4)$$

Using equations (1) and (2) we have

$$(LU)x = b \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (5)$$

or $$L(Ux) = b \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (6)$$

Let $$Ux = z \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (7)$$

where z is the unknown vector. Then eqn (6) becomes

$$Lz = b \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (8)$$

Hence the solution for the original equation (1) can be obtained in two stages,

a. Solve the eqn. (8) $Lz = b$ by forward substitution to obtain z
b. Solve the equation (7) $Ux = z$ by back substitution

The implementation of the LU Triangular Factorization method is done by using the Dolittle Algorithm.

Dolittle Algorithm:

1. Enter the order of the matrix A, $n =$
2. Enter the elements of the matrix A and the vector b
3. Set $u_{1j} = a_{1j}$ *for j = 1 to n.*
4. Set $l_{ii} = 1$ *for i =1 to n.*
5. Ser $l_{i1} = a_{i1} / u_{11}$ *for i =2 to n.*

6. For each *j = 2 to n* do:

   (i) For *i = 2 to j*

Compute $u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj}$

Repeat *i*

(ii) For *i = j+1 to n*

Compute $l_{ij} = \frac{1}{u_{ij}} \left[a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj} \right]$

Repeat *i*

7. Set $z_1 = b_1$
8. For *i = 2 to n*

Set $sum = \sum_{j=1}^{i-1} l_{ij} z_j$

Set $z_i = b_i - sum$

Repeat *i*

9. Set $x_n = z_n / u_{nn}$
10. For *i = n-1 to 1*

Set $sum = \sum_{j=i+1}^{n} u_{ij} x_j$

Set $x_i = (z_i - sum) / u_{ii}$

Repeat *i*

11. Write the results

```
// LU_Decompose_xx.c
// Program to decompose a matrix into LU form and hence to find its
// determinant
// Roll No     :
// Date        :
```

```
#define YES 1
#define NO 0
#include<stdio.h>
#include<math.h>

double A[10][10],U[10][10],L[10][10],B[10];
void LUD(int,double[10][10],double[10][10],double[10][10],int*);

void main()
{
     int n,fact,i,j;
     double product_U,product_L,mod_L,mod_U,DET_A;

     printf("Enter the order of matrix A,n=");
     scanf("%d",&n);

     printf("enter the elements of matrix A\n");
     for(i=1;i<=n;i++){
          for(j=1;j<=n;j++){
               printf("A[%2d][%2d]=",i,j);
               scanf("%lf",&A[i][j]);
          }
          printf("\n");

     }
     LUD(n,A,U,L,&fact);
     printf("matrix U\n");

     for (i=1;i<=n;i++){
          for(j=1;j<=n;j++)
               printf("%4.2lf\t",U[i][j]);
          printf("\n");
     }
     printf("matrix L\n");

     for(i=1;i<=n;i++){
          for(j=1;j<=n;j++)
               printf("%4.2lf\t",L[i][j]);
          printf("\n");
     }

     product_U=1.0;
     product_L=1.0;

     for(i=1;i<=n;i++)
          product_U=product_U*U[i][i];
     mod_U=product_U;
```

```
     for(i=1;i<=n;i++)
          product_L=product_L*L[i][i];
     mod_L=product_L;
     DET_A=mod_U*mod_L;

     printf("the determinent of matrix A =%4.2lf\n\n",DET_A);
}
void LUD(int n,double A[10][10],double U[10][10],double
                                                  L[10][10],int *fact)
{
     int i,j,k;
     double sum;

     for(i=1;i<=n;i++){
          for(j=1;j<=n;j++){
               U[i][j]=0.0;
               L[i][j]=0.0;
          }
     }
     for(j=1;j<=n;j++)
          U[1][j]=A[1][j];

     for(i=1;i<=n;i++)
          L[i][i]=1.0;

     for(i=1;i<=n;i++)
          L[i][1]=A[i][1]/U[1][1];









     for(j=2;j<=n;j++){
          for(i=2;i<=j;i++){
               sum=A[i][j];
               for(k=1;k<=i;k++)
                    sum=sum-L[i][k]*U[k][j];
               U[i][j]=sum;
```

```
          }
          for(i=j+1;i<=n;i++){
               sum=A[i][j];
               for(k=1;k<=j-1;k++)
                    sum=sum-L[i][k]*U[k][j];
               L[i][j]=sum/U[j][j];
          }
     }
     *fact=YES;
     return;
 }

/*****************************************************/
               OUTPUT
/*****************************************************/

     Enter the order of matrix A,n=3

     Enter the elements of matrix A

     A[ 1][ 1]=3
     A[ 1][ 2]=2
     A[ 1][ 3]=1

     A[ 2][ 1]=2
     A[ 2][ 2]=3
     A[ 2][ 3]=2

     A[ 3][ 1]=1
     A[ 3][ 2]=2
     A[ 3][ 3]=3

     Matrix U

     3.00 2.00 1.00
     0.00 1.67 1.33
     0.00 0.00 1.60



     Matrix L

     1.00 0.00 0.00
     0.67 1.00 0.00
     0.33 0.80 1.00
```

```
     The determinent of matrix A =8.00

/*************************************************/
```

Result: The C program to find the determinant of a matrix using LU Decomposition Method is run and its output obtained and verified.

The determinant of the given matrix, $|A| =$ .....

7. EVALUATION OF STATISTICAL PARAMETERS: MEAN DEVIATION AND STANDARD DEVIATION

Aim: To evaluate statistical parameters such as arithmetic mean, mean deviation and standard deviation of a frequency tabulated data set.

x_i	36	21	53	43	58	91	22	67	15	87
f_i	3	4	6	3	5	8	4	3	7	5

Theoretical Background: Statistics is a discipline of science which is concerned with the study of the collection, organization, analysis, interpretation and presentation of data. This data may be a collection of discrete variables or continuous variables.

Untabulated Data:

Usually the collected data may be a very large set, with values varying in a random manner. For example, we can think of a data set of 50 elements, like marks of a group of students, which may vary in a wide range 0 -100. In order to analyze and derive meaningful conclusions, a frequency table of the collected data is constructed.

Frequency Table:

A frequency is a systematic arrangement of raw data into smaller groups of appropriate sizes called classes or intervals. The number of variates that are contained in a given interval is called as the frequency of the said interval. A frequency table for the marks of a group of students can be constructed as follows.

Frequency Tabulated Data Set

Interval	Frequency
0-10	2
10-20	4
20-30	11
30-40	4
40-50	3
50-60	7
60-70	2
70-80	5
80-90	10
90-100	2
Total Frequency	**50**

Statistical Parameters:

There are three quantities that describe concisely any given group of data. They are

1. Averages or Measures of Central Tendency or Measures of Location.
2. Measures of Dispersion
3. Measures of Skewness

Averages:

An average indicates the central value of the size of the typical member of a group of data. It represents the whole series and conveys a fairly adequate idea of the whole group. As averages tend to lie centrally within a set of data arranged according to magnitude, they are also called as measures of central tendency or measures of location.

There are three common forms of averages, namely, arithmetic mean, median and mode.

Measures of Dispersion:

While the averages (mean, median and mode) represent the typical values to which the whole series of data converge, the measures of dispersion denote the extent to which the magnitude of a given data set differ from each other. Examples of these are mean deviation, standard deviation etc.

Measures of Skewness:

This is defined as a measure of the extent to which a probability distribution of a real-valued random variable leans to one side of the mean. The skewness value can be positive or negative, or even undefined.

In this experiment we evaluate just three statistical parameters namely, arithmetic mean, mean deviation and standard deviation for a tabulated data set.

Arithmetic Mean of Untabulated Data

The arithmetic mean of a set of data n elements of untabulated data, say

$$\{x_1, x_2, x_3, \ldots\ldots\ldots\ldots x_n\}$$

is defined as

$$\bar{x} = \frac{x_1 + x_2 + x_3 + \ldots . x_n}{n}$$

or

$$\bar{x} = \frac{\sum_{i=1}^{n} x}{n}$$

Arithmetic Mean of Tabulated Data:

When a data set is given as frequency distribution of n classes or intervals and frequencies, then the arithmetic mean is expressed as

$$\bar{x} = \frac{\sum_{i=1}^{N} x_i f_i}{\sum_{i=1}^{n} f_i}$$

Mean Deviation:

Mean Deviation is the arithmetic mean of the absolute values of the differences of the variates x_i from some arbitrarily fixed constant M, where M is the mean of the data. It is expressed as

$$M_d = \frac{\sum_{i=1}^{N} f_i |x_i - M|}{\sum_{i=1}^{N} f_i}$$

where

$$M = \frac{\sum_{i=1}^{N} x_i f_i}{\sum_{i=1}^{n} f_i}$$

Standard Deviation:

The Standard Deviation gives us an idea of the spread of the variates about the mean

value. It is expressed as $$SIGMA = \sqrt{\frac{\sum_{i=1}^{N} f_i (x_i - M)^2}{\sum_{i=1}^{N} f_i}}$$

where $$M = \frac{\sum_{i=1}^{N} x_i f_i}{\sum_{i=1}^{n} f_i}$$

These three statistical parameters can be determined for a given frequency tabulated data set by implementing the formulae given above using the C-program `stat_xx.c`

```
// stat_xx.c
// C Program to calculate the mean deviation and standard deviation
// of a frequency tabulated data set
// Roll No:
// Date :

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

void main()
{
     int i,n,x[20],f[20];
     double M,Md,sum_x,sum_f,sum_xf,sum_xM;
     double x_diff,NUM,DENOM, SIGMA;

     printf(" Enter the number of data elements in the set, n = ");
     scanf("%d",&n);

     for(i=1;i<=n;i++){
          printf("x[%d] = ",i);
          scanf("%d",&x[i]);
          printf("f[%d] = ",i);
          scanf("%d",&f[i]);
          printf("\n");
     }
     /**************To find the Arithmetic Mean ***************/

     sum_x = 0.0;
```

```
sum_f = 0.0;
sum_xf = 0.0;
sum_xM = 0.0;
for(i=1;i<=n;i++){
     sum_x = sum_x +(double)(x[i]);
     sum_f = sum_f +(double)(f[i]);
     sum_xf = sum_xf+ (double)(x[i])*(double)(f[i]);
}
M = sum_xf/sum_f;
printf("mean = %lf\n",M);
printf("sum_xf = %lf\n",sum_xf);

/********** To find the mean deviation ********************/

for(i=1;i<=n;i++)
     sum_xM = sum_xM +(double)(f[i])*fabs((double)(x[i])-M);


NUM = sum_xM;
DENOM = sum_f;
Md = NUM/DENOM;

printf("The mean deviation of the given tabulated data is ");
printf(" Md = %12.6lf\n",Md);

/************ To find the standard deviation ****************/

x_diff = 0.0;
sum_xM =0.0;
for(i=1;i<=n;i++){
     x_diff = pow(fabs((double)(x[i])-M),2.0);
     sum_xM = sum_xM + (double)(f[i])*x_diff;
}
NUM = sum_xM;
DENOM = sum_f;
SIGMA = sqrt(NUM/DENOM);

printf("The std deviation of the given tabulated set is ");
printf(" SIGMA = %12.6lf\n",SIGMA);

/************* Data File ***********************************/
printf("-------------------------------------------\n");
printf("\t xi\t fi \t xi*fi\t(xi-M)^2 \tfi*(xi-M)^2\n");
printf("-------------------------------------------\n");

for(i=1;i<=n;i++){
     x_diff = pow(fabs((double)(x[i])-M),2.0);
```

```
          printf("\t %d\t %d\t %d\t %5.2lf\t%5.2lf\n", x[i],
                      f[i],f[i]*x[i],x_diff,(double)(f[i])*x_diff);
     }
     printf("-----------------------------------------\n");

     printf("mean = %lf\n",M);
     printf(" Md = %12.6lf\n",Md);
     printf(" SIGMA = %12.6lf\n",SIGMA);
}
```

```
/*****************************************************************/
                          OUTPUT
/*****************************************************************/
```

xi	fi	xi*fi	(xi-M)^2	fi*(xi-M)^2
36	3	108	249.38	748.13
21	4	84	948.13	3792.51
53	6	318	1.46	8.76
43	3	129	77.29	231.88
58	5	290	38.54	192.72
91	8	728	1537.29	12298.35
22	4	88	887.54	3550.17
67	3	201	231.29	693.88
15	7	105	1353.63	9475.39
87	5	435	1239.63	6198.13

```
     Arithmetic Mean                M = 51.791667

     Mean Deviation                 Md = 23.901042

     Standard Deviation          SIGMA =    27.835049

/*****************************************************************/
```

Result: The C program to evaluate the statistical parameters for the given frequency tabulated data is run and its output obtained.

Arithmetic Mean = ……..
Median Deviation Md = ……..
Standard Deviation SIGMA = ……..

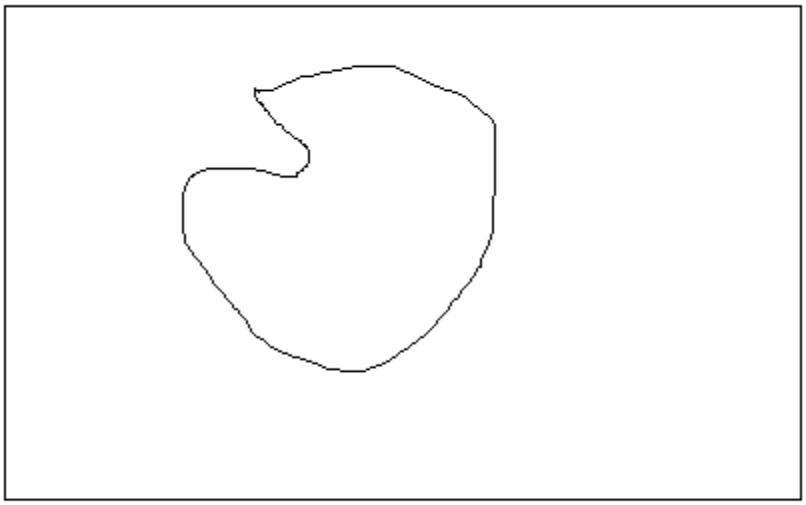
8. RANDOM NUMBER GENERATION-CALCULATION OF THE VALUE OF π

Aim : To generate random numbers by employing Park-Miller Implementation of Lehmer's Prime Modulus Multiplicative Linear Congruential Generator Algorithm and hence to estimate the value of **π** using Monte Carlo Simulation technique.

Theory :

I. Monte Carlo Method : Estimation of π Value

Monte Carlo is a technique that uses random numbers to solve physical problems. Let us imagine we want to find the area of an irregularly shaped plane (two-dimensional) figure. We can do this by enclosing the plane in a square board of known area and hang it on a wall. Let a blind foldded shooter shoot at it randomly.



Then if we assume unbiased probability of hitting we have,

$$\frac{Area\ of\ the\ figure}{Area\ of\ the\ square} = \frac{No\ of\ shots\ falling\ \text{in}\ the\ figure}{Total\ Number\ of\ shots}$$

Note: Greater the number of shots, higher is the accuracy. Using this technique, the value of π can be estimated. The area of a circle is A= πr^2. if the circle is a unit circle (i.e.) if r=1, then $A = \pi$. Thus by finding the area of the circle, we can find the value of π.

Note: If we just find the area of the first quadrant of a unit circle, it is enough. We can then multiply the answer by 4 to get the value of π.

We here generate 'x' & 'y' as two sequences of pseudo random numbers. Using the function `mmc_gen()`. And count the number of pairs lying within the first quadrant using the condition ($x^2 + y^2$) <=1.

Random Number Generation:

In digital computers, random numbers find important applications. Hence the ability to generate random numbers assumes importance. The Park-Miller implementation of the Lehmer's prime modulus multiplicative linear congruential generator algorithm is the most preferred and is implemented in many standard scientific routines. (*Ref: Communications of ACM, Oct. 1988, Vol: 31, No: 10)*. It consists of defining six parameters, a modulus ***m***, a multiplier ***a***, and four integers ***q***, ***r***, ***low*** and ***high.*** The random number - ***ran_no*** is generated using an iterative process

1. Initialize the parameters as

$$a = 16807 \quad = (7^5)$$
$$m = 2147483647 \quad = (2^{31} - 1)$$
$$q = 127773 \quad = (m/a)$$
$$r = 2836 \quad = (m\ mod\ a)$$

2. Find iteratively

$$low = SD\ mod\ q$$
$$high = SD\ /\ q$$
$$test = a * low - r * high$$

If (test >0)
SD = test
else
SD = test + m

3. Normalize

$$ran\_no = SD/m$$

A program `pi_xx.c` to calculate the value of **π** is given. It generates the random numbers using the routine **`mmc_gen()`,** that implements Lehmer's prime modulus multiplicative linear congruential random number generator using the Park-Miller Algorithm.

```
// pi_xx.c
// To estimate the value of pi
```

```
// Roll No:
// Date:

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

double mmc_gen();
long SD;

void main()
{
     int i,n,count,count_2;
     double x,y,pi,sigma,p,q;

     printf("Seed SD = ");
     scanf("%ld",&SD);

     /****** SD can be any five digit integer *******/

     /****** SD=12345 ******************************/

     printf("----------------------\n");
     printf("  n \t     pi \n");
     printf("----------------------\n");
     for(n=250;n<=10000;n=n+250){
          count=0;
          for(i=1;i<=n;i++){
               x=mmc_gen();
               y=mmc_gen();
               if((x*x+y*y)<=1.0000)
               count+=1;
          }
          pi=(4.0*count)/(float)(n);
          if(n%750==0)
               printf("%5d %12.4lf\n",n,pi);
     }
     printf("----------------------\n");
     printf(" The value of pi converges to pi = %lf \n",pi);
}

double mmc_gen()
{
     long m,a,q,r,high,low,test;
     double ran_no;

     m = 2147483647;
     a = 16807;
```

```
     q = 127773;
     r = 2836;

     high = SD/q;
     low  = SD%q;
     test = a*low-r*high;

     if(test>0)
          SD = test;
     else
          SD = test+m;

     Ran_no = (double)(SD)/(double)(m);

     return ran_no;
}



/*******************************************************/
                    OUTPUT
/*******************************************************/


            Seed SD = 12345
           ----------------------
            n       pi
           ----------------------
             750       3.0987
            1500       3.1040
            2250       3.1484
            3000       3.1013
            3750       3.1253
            4500       3.0764
            5250       3.0857
            6000       3.1720
            6750       3.1662
            7500       3.1141
            8250       3.1282
            9000       3.1276
            9750       3.1463
           ----------------------

     The value of pi converges to pi = 3.146000

/*******************************************************/
```

Result: A C program to generate random numbers and estimate the value of π is run and the result obtained. π = ……