Numerical Simulations in Physics
II M.Sc Physics -PC Laboratory Manual

Semester IV
Course Code :17PPH4C15P2

Postgraduate and Research Department of Physics
JAMAL MOHAMED COLLEGE(Autonomous)
College with Potential for Excellence
Reaccredited (3rd cycle) with 'A' Grade by NAAC
DBT-Star Scheme and DST-FIST Funded
Affiliated to Bharathidasan University
TIRUCHIRAPPALLI -620 020

March 16, 2020

Contents

Chapter 1

Hermite Polynomials: LHO Wavefunctions

1.1 Aim

1. To obtain the numerical values for the Hermite polynomial functions by solving their analytical expressions.

2. To plot the wave functions for even and odd quantum states of a one dimensional Linear Harmonic oscillator (LHO) using the numerical values of the Hermite polynomial functions so obtained and

3. To plot the probability density of the one dimensional LHO for the $n = 10$ quantum state.

1.2 Theoretical Details

The Schroedinger Equation for the one dimensional LHO is given as

$$\left(-\frac{\hbar^2}{2m}\frac{d^2}{dx^2} + \frac{1}{2}m\omega^2 x^2\right)\psi(x) = E\psi(x) \tag{1.1}$$

As this is a second order inhomogenous differential equation, it is difficult to solve analytically. Hence it is converted into a dimensionless form, namely the Weber-Hermite differential equation

$$\frac{d^2\psi(\rho)}{d\rho^2} + (1 - \rho^2 + 2n)\psi(\rho) = 0, \tag{1.2}$$

by changing the independent variable from $x \longrightarrow \rho$, such that

$$\begin{aligned} \rho &= \alpha x \\ \alpha &= \sqrt{\frac{m\omega}{\hbar}}. \end{aligned} \tag{1.3}$$

and n is an arbitrary constant, that we assume as the quantum number for the system. Further assuming

$$\psi(\rho) = e^{\rho^2/2}\nu(\rho) \tag{1.4}$$

and multiplying it with a constant, the **Weber-Hermite differential equation** gets converted to the **Hermite Differential Equation**

$$H_n(\rho) - 2\rho H_n(\rho) + 2nH_n(\rho) = 0. \tag{1.5}$$

The solutions of this equation are the familiar **Hermite polynomial functions** $H_n(\rho)$. In terms of these Hermite Polynomial Functions,the wave function $\psi_n(\rho)$ for the LHO can be given as

$$\psi_n(\rho) = \left[\frac{\alpha}{\sqrt{\pi}2^n n!}\right]^{\frac{1}{2}} e^{-\rho^2/2} H_n(\rho), \tag{1.6}$$

where n denotes the quantum state of the LHO.

1.3 Analytical Expression for the Hermite Polynomial Function $H_n(\rho)$

The $H_n(\rho)$ is the Hermite Polynomial function of degree n. Its analytical expression is given as

$$H_n(\rho) = \sum_{r=0}^{m} \frac{(-1)^r n!}{r!(n-2r)!}(2\rho)^{(n-2r)} \tag{1.7}$$

where $m = \dfrac{n}{2}$.

C Function to Implement the Analytical Expression for $H_n(\rho)$

The analytical expression for the Hermite Polynomial function can be implemented in C using the FUNCTION given below.

```
double Hermite(double rho,int n)
{
    int i,r,m;
    long int p,t,u;
    double q,s,v,sum,Hn,Num,Denom;

    m = n/2;
    p = Fact(n);

    sum = 0;
    for(r = 0; r <= m; r++){
        s = (double)(n-2*r);
        t = Fact(r);
        u = Fact(n-2*r);

        Num = (double)(p)*pow((2.0*rho),s);

        if(r % 2 != 0)
            Num = -Num;          /* Implementing (-1)^r */
```

```
20
21          Denom = (double)(t)*(double)(u);
22          sum += Num/Denom;
23      }
24      Hn = sum;
25      return Hn;
26  }
```

C Function for Finding the Factorial of a number {n!}

The factorial of a number, say n, that is $n!$ can be implemented using the following C FUNCTION.

```
1  long int Fact(int n)
2  {
3      int i;
4      long int p;
5
6      if(n != 0){
7          p = 1;              /* initializing p to unity */
8          for(i = 1; i <= n; i++)
9                  p = p*i;
10     }
11     else
12         p = 1;              /* setting 0! = 1     */
13     return p;
14 }
```

1.4 C Program to Obtain the Wave functions of the LHO

The full C program to obtain numerically the wave function $\psi_n(\rho)$ as well as the probability density $|\psi_n(\rho)|^2$ of the LHO is listed below. The numerical values are stored in a data file `lho_59xx.dat`

Note: The characters `xx` stand for the last two digits of the roll number of the student.

```
1  // lho_59xx.c
2  // To plot the wave function of a Linear Harmonic Oscillator
3  // Roll No :
4  // Date    :
5
6  #include<stdio.h>
7  #include<math.h>
8
9  long int Fact(int);
10 int n;
11
```

```
12 int main()
13 {
14         int i,N;
15         long int p;
16         double m,omega,z,v,pi,rho,h,y;
17         double Hermite (double ,int);
18
19         FILE *fp;
20         fp=fopen("lho_59xx.dat","w");
21         if(fp==NULL){printf("File Open Error!\n");return 0;}
22         pi=4.0*atan(1.0);
23         h =0.01;
24         N =700;
25
26         printf("n=  ");
27         scanf("%d",&n);
28
29         for(i=-N;i<=N;i++){
30                 rho=(double)(i)*h;
31                 y=Hermite(rho,n);
32                 p=Fact(n);
33                 v=sqrt((1.0/pi))*(1.0/pow(2.0,n)*(1.0/(double)(p)));
34                 z=sqrt(v)*y*exp(-(rho*rho)/2.0);
35                 fprintf(fp,"%12.6lf%12.6lf%12.6lf\n",rho,z,pow(z
    ,2.0));
36         }
37         fclose(fp);
38 }
39 double Hermite (double rho,int n)
40 {
41         int i,r,m;
42         long int p,t,u;
43         double q,s,sum,Hn,Num,Denom,v;
44
45         m=n/2;
46         p=Fact(n);
47
48         sum=0;
49         for(r=0;r<=m;r++){
50                 s=(double)(n-2*r);
51                 t=Fact(r);
52                 u=Fact(n-2*r);
53                 Num=(double)(p)*pow((2.0*rho),s);
54                 if(r % 2 != 0)
55                         Num = -Num;
56                 Denom = (double)(t)*(double)(u);
57                 sum += Num/Denom;
```

```
58          }
59          Hn = sum;
60          return Hn;
61  }
62  long int Fact(int n)
63  {
64          int i;
65          long int p;
66
67          if(n!=0){
68                  p=1;
69                  for(i=1;i<=n;i++)
70                          p=p*i;
71          }
72          else
73                  p=1;
74          return p;
75  }
```

The command line arguments to compile and run the C code is given in Fig. (1.1).

```
File Edit View Search Terminal Help
ahmed@ahmed-PC:~/ISHAQ-PC/Classes-Nov2019/TEST/tested$ cc lho_59xx.c -lm
ahmed@ahmed-PC:~/ISHAQ-PC/Classes-Nov2019/TEST/tested$ ./a.out
n=  10
ahmed@ahmed-PC:~/ISHAQ-PC/Classes-Nov2019/TEST/tested$
```

Figure 1.1: Command line arguments for compiling and running the C code `lho_59xx.c`

1.5 Plotting LHO Wave Functions and Probabilities

Using the numerical values in the data file `lho_51xx.dat`, we can plot the wave functions as well as the probability densities using the graphics package `gnuplot`.

Plotting Using Gnuplot

To invoke the gnuplot package, just type
`gnuplot`
in the command line. The gnuplot window opens. Then type the following to obtain the graphical plot of the wave functions. A snapshot of the command line arguments is shown in Fig. (1.2). The plots of the first three even state wave functions, namely for $n = 0$, $n = 2$ and $n = 4$ of the one dimensional LHO are plotted as shown in Fig. 1.3(a) and their corresponding

```
ahmed@ahmed-PC:~$ gnuplot

        G N U P L O T
        Version 5.2 patchlevel 2    last modified 2017-11-01

        Copyright (C) 1986-1993, 1998, 2004, 2007-2017
        Thomas Williams, Colin Kelley and many others

        gnuplot home:     http://www.gnuplot.info
        faq, bugs, etc:   type "help FAQ"
        immediate help:   type "help"  (plot window: hit 'h')

Terminal type is now 'qt'
gnuplot> set title "Even States Wavefunctions"
gnuplot> set xrange [ -6:6]
gnuplot> set yrange [-0.9:0.9]
gnuplot> set label "(a)" at -5,-0.75
gnuplot> set xlabel "{/Times-Italic=30 {/Symbol=25 \\162}}
gnuplot> set ylabel "{/Times-Italic=30 {/Symbol=25 \\171(\\162)}
gnuplot> pl 'lho_59xx_00.dat' u 1:2 w l lw 3.0 lc rgb "red" t "n = 0"
```

Figure 1.2: Command line arguments for obtaining figures using gnuplot

probability densities are shown in 1.3(b). Similarly the plots of the first three odd state wave functions, namely for $n = 1$, $n = 3$ and $n = 5$ of the one dimensional LHO are plotted as shown in Fig. 1.4(a) and their corresponding probability densities are shown in 1.4(b). The wave function and the probability density for the $n = 10$ eigen state are shown in Fig. 1.5

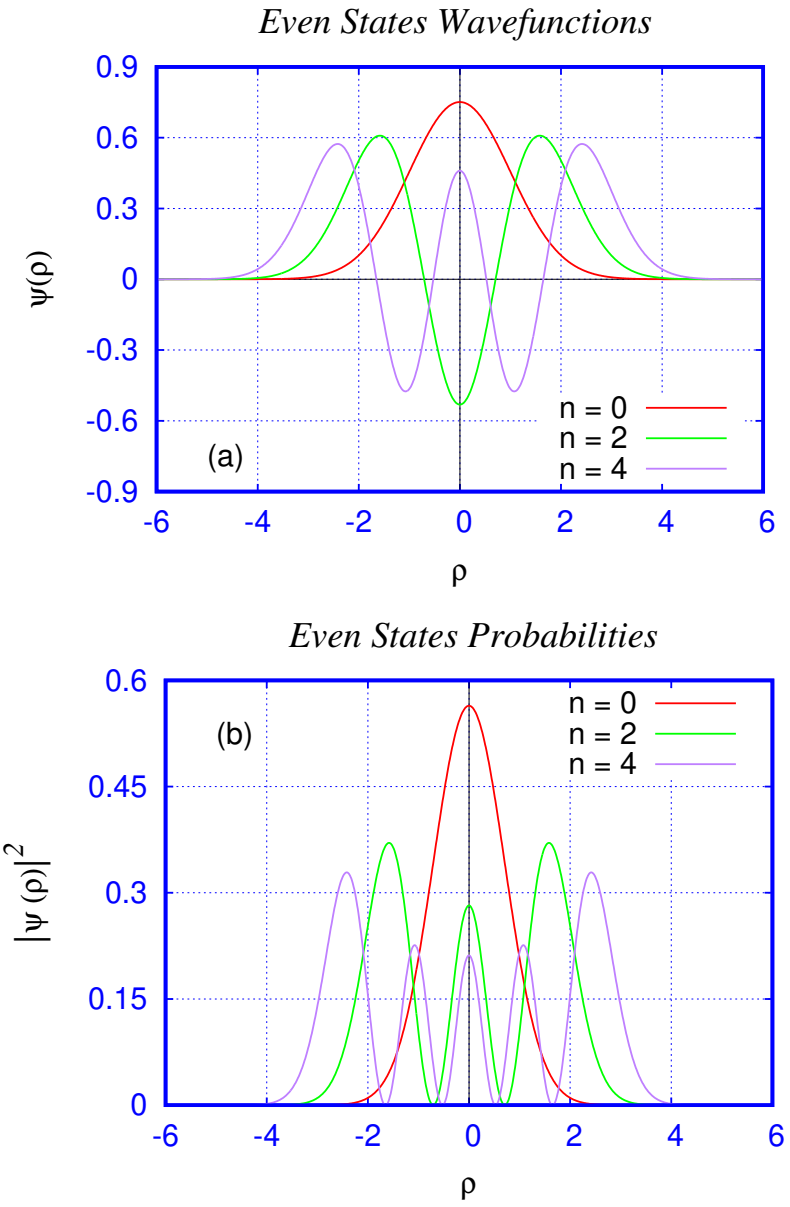

Figure 1.3: (a) The eigen functions for the first three even eigen states namely $n = 0$, $n = 2$ and $n = 4$ and (b) their corresponding probability densities

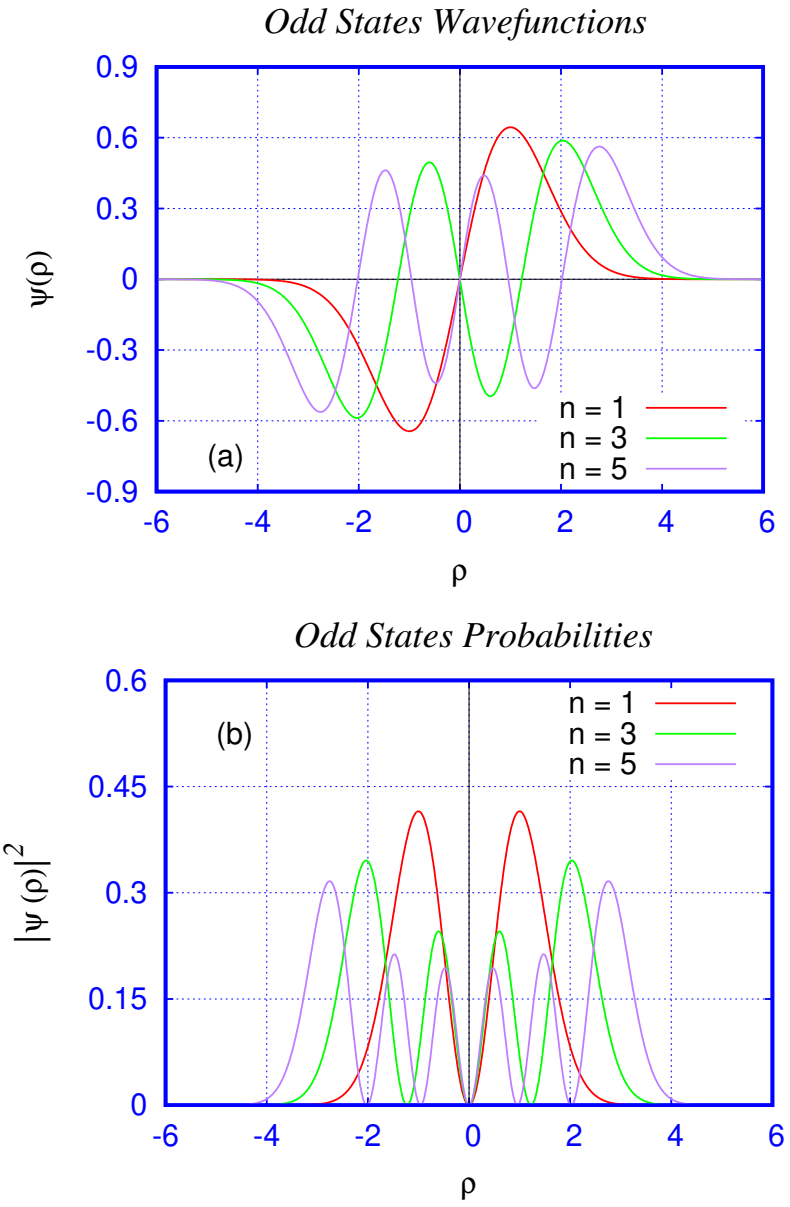

Figure 1.4: (a)The eigen functions for the first three odd eigen states namely $n = 1$, $n = 3$ and $n = 5$ and (b) their corresponding probability densities.

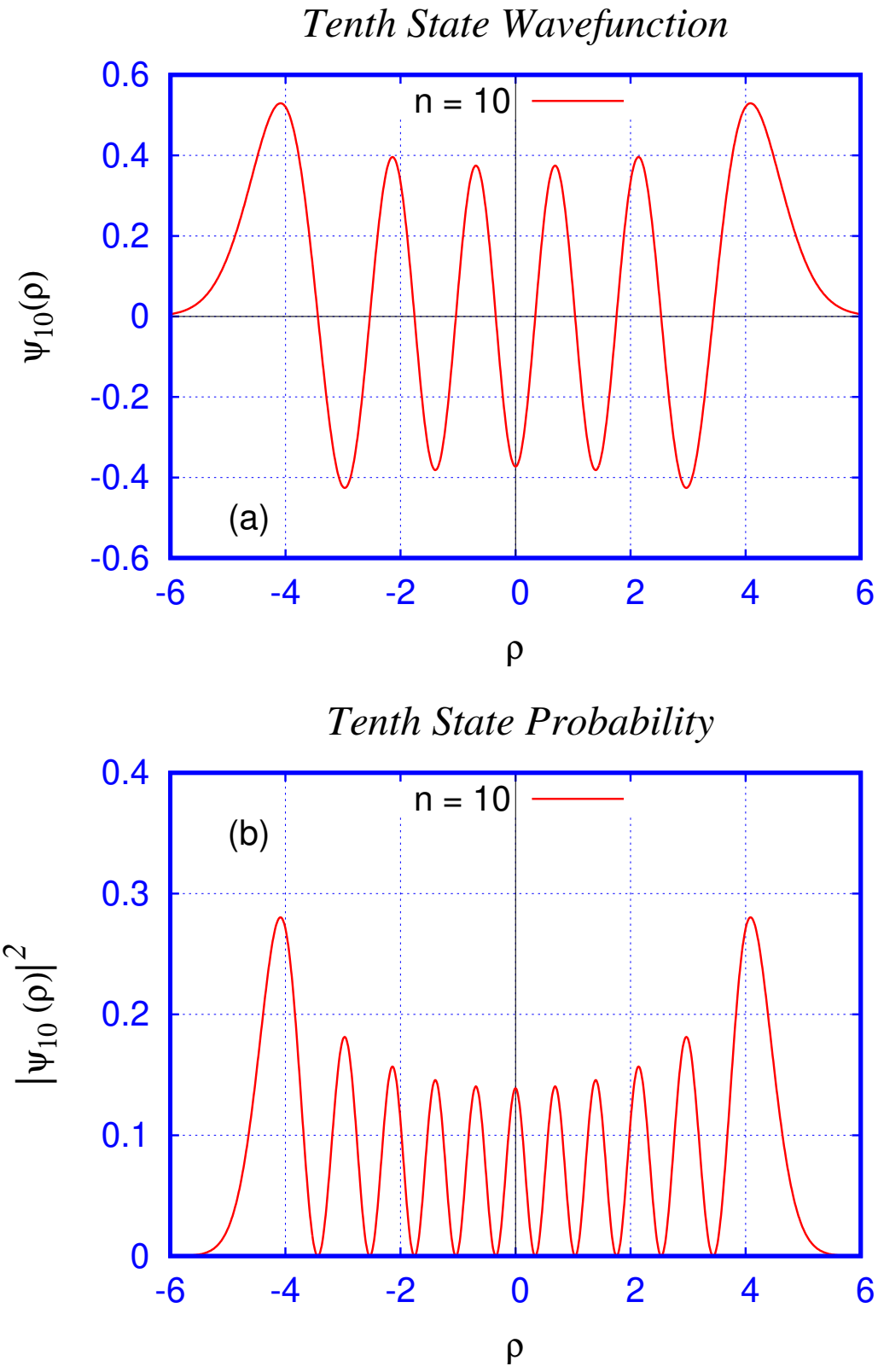

Figure 1.5: The eigenfunction and the probability density for tenth eigen state of the LHO, $n = 10$

1.6 Result

1. A C program for implementing the analytic expression for the wave function of a one dimensional LHO is run.

2. The even and odd state wave functions and probability densities are plotted using `gnuplot` graphics package.

3. Further the wave function and the probability density for the $n = 10^{th}$ are also plotted.

Chapter 2

Simulation of Beats, Polarization and Lissajous Figures

2.1 Aim

To study numerically the superposition of two simple harmonic motions (SHMs)

1. moving in the same direction and observe the phenomenon of *beats*.

2. moving in the perpendicular directions and observe the phenomena of *circular* and *elliptic polarizations*

3. moving in the perpendicular directions and plot the *Lissajous Figures*

2.2 Theoretical Details

- When two SHMs differing slightly in frequency $\omega_1 \simeq \omega_2$, same phase $\delta = 0$ and moving in the same direction interfere, then *beats* phenomenon is observed.

- When two SHMs having the same frequency $\omega_1 = \omega_2$ but a finite phase difference $\delta \neq 0$ interfere in mutually perpendicular directions, then *circular* and *elliptic polarizations* are observed.

- When two SHMs differing in both the frequency $\omega_1 \neq \omega_2$ and phase $\delta \neq 0$ interfere in mutually perpendicular directions, then *Lissajous Figures* are observed.

2.3 Beats Phenomenon

Let us consider two Simple Harmonic Waves moving in the same direction. Let these waves be described mathematically as

$$x = a_1 cos(\omega_1 t), \tag{2.1}$$
$$y = a_2 cos(\omega_2 t + \delta). \tag{2.2}$$

As time passes, these two waves will interfere constructively and destructively. If these two waves differ in frequency by a very small amount, then their interference will give rise to beats. If $a_1 = a_2 = a$, $\delta = 0$, then adding the above equations gives the resultant of the two SHMs as

$$\begin{aligned} z &= x + y \\ &= a(\cos\omega_1 t + \cos\omega_2 t) \\ &= \left\{2a\cos\left(\frac{\omega_1 - \omega_2}{2}\right)t\right\}\cos\left(\frac{\omega_1 + \omega_2}{2}\right)t \end{aligned}$$

or

$$z = \mathcal{A}\cos(\omega t), \tag{2.3}$$

where $\mathcal{A}$ is the amplitude of the resultant SHM, given by

$$\mathcal{A} = \left\{2a\cos\left(\frac{\omega_1 - \omega_2}{2}\right)t\right\} \tag{2.4}$$

and ω is its angular frequency which is the average of the frequencies of the two SHMs and is given as

$$\omega = \left(\frac{\omega_1 + \omega_2}{2}\right) \tag{2.5}$$

C Program to Observe the Phenomenon of Superposition of TWO SHMs

The full C program `shm.c` to simulate numerically the phenomenon of superposition of two SHMs is given below. To observe beats as well as to draw the envelope for the beats we assume the values for the parameters as $a_1 = a_2 = 0.75$, $\omega_1 = 0.5$, $\omega_2 = 0.55$ and $\delta = 0$. The numerical values resulting from the program are stored in a data file `beats_59xx.dat`.The observed beats phenomenon as well as the envelopes of the resulting signal, given by Eqn. (2.3) are shown in Fig.(2.3).

```
1  // shm_59xx.c
2  // program to observe numerically the superposition of two SHMs and
       // hence to simulate beats, circular and elliptic polarizations
       and // Lissajous figures
3  // Roll no: 59xx
4  // Date    : 18/12/2019
5
6  #include<stdio.h>
7  #include<math.h>
8
9  void main()
10 {
11         int i,n;
12         double u[5001],v[5001], x[5001],y[5001],z[5001];
13         double a1,a2,h,t,pi;
14         double omega1,omega2,omega_diff, delta,phi;
15
16         FILE *fp1, *fp2, *fp3;
17
```

```
18          fp1=fopen("beats_59xx.dat","w");
19          if(fp1==NULL){printf("File-1 Open Error!\n");return;}
20
21          fp2=fopen("polarization_59xx.dat","w");
22          if(fp2==NULL){printf("File-2 Open Error!\n");return;}
23
24          fp3=fopen("lissajous_59xx.dat","w");
25          if(fp3==NULL){printf("File-3 Open Error!\n");return;}
26
27          n=5000;
28          h=0.1;
29
30          pi = 4.0*atan(1.0);
31
32          //a1=0.75
33          //a2=0.75
34
35          printf("Enter the amplitude of the first SHM, a1= " );
36          scanf("%lf",&a1);
37
38          printf("Enter the amplitude of the second SHM, a2= " );
39          scanf("%lf",&a2);
40
41 /*************************************************************/
42
43          printf("\n To Simulate the phenomenon of beats \n");
44
45          //omega1 = 0.5
46          //omega2 = 0.55
47          // delta = 0.0
48
49          printf("Enter the frequency of the first SHM, omega1=");
50          scanf("%lf",&omega1);
51
52          printf("Enter the frequency of the second SHM,omega2=");
53          scanf("%lf",&omega2);
54
55          printf("Enter the phase difference, delta =");
56          scanf("%lf",&delta);
57
58
59          omega_diff = (omega1-omega2)/2.0;
60
61          for(i=1;i<=n;i++){
62                  t=(double)(i)*h;
63                  x[i] = a1*cos(omega1*t);
64                  y[i] = a2*cos(omega2*t);
```

```
65                  z[i] = x[i]+y[i];
66                  u[i] = 2.0*a1*cos(omega_diff*t);
67                  v[i] = 2.0*a2*cos(omega_diff*t+pi);
68                  fprintf(fp1,"%12.6lf%12.6lf%12.6lf%12.6lf\n",t,z[i],
    u[i],v[i]);
69          }
70          fclose(fp1);
71
72 /*************************************************************/
73
74          printf("\n To Simulate the phenomenon of polarization \n");
75
76          // omega1=0.5
77          // omega2=0.5
78          // delta = 90,120,150,180,210,240,270,300,330,360
79
80          printf("Enter the frequency of the first SHM, omega1=");
81          scanf("%lf",&omega1);
82
83          printf("Enter the frequency of the second SHM,omega2=");
84          scanf("%lf",&omega2);
85
86          printf("Enter the phase difference, delta =");
87          scanf("%lf",&delta);
88
89          phi =   (delta*pi)/180.0;
90
91          for(i=1;i<=n;i++){
92                  t=(double)(i)*h;
93                  x[i] = a1*cos(omega1*t);
94                  y[i] = a2*cos(omega2*t+phi);
95                  fprintf(fp2,"%12.6lf%12.6lf%12.6lf\n",t,x[i],y[i]);
96          }
97          fclose(fp2);
98
99 /*************************************************************/
100
101         printf("\n To observe Lissajous figures \n");
102
103         //omega1=0.75
104         //omega2=1.0
105         //delta = 90
106
107         printf("Enter the frequency of the first SHM, omega1=");
108         scanf("%lf",&omega1);
109
110
```

```
111         printf("Enter the frequency of the second SHM,omega2=");
112         scanf("%lf",&omega2);
113
114         printf("Enter the phase difference, delta =");
115         scanf("%lf",&delta);
116
117         phi =   (delta*pi)/180.0;
118
119         for(i=1;i<=n;i++){
120                 t=(double)(i)*h;
121                 x[i] = a1*cos(omega1*t);
122                 y[i] = a2*cos(omega2*t+phi);
123                 fprintf(fp3,"%12.6lf%12.6lf%12.6lf\n",t,x[i],y[i]);
124         }
125         fclose(fp3);
126 }
```

The command line arguments for plotting the beats and other phenomena are listed in Fig. (2.1).

```
File  Edit  View  Search  Terminal  Help
ahmed@ahmed-PC:~/ISHAQ-PC/Classes-Nov2019/TEST$ cc shm_59xx.c -lm
ahmed@ahmed-PC:~/ISHAQ-PC/Classes-Nov2019/TEST$ ./a.out
Enter the amplitude of the first SHM, a1= 0.75
Enter the amplitude of the second SHM, a2= 0.75

 To Simulate the phenomenon of beats
Enter the frequency of the first SHM, omega1=0.5
Enter the frequency of the second SHM,omega2=0.55
Enter the phase difference, delta =0

 To Simulate the phenomenon of polarization
Enter the frequency of the first SHM, omega1=0.5
Enter the frequency of the second SHM,omega2=0.5
Enter the phase difference, delta =120

 To observe Lissajous figures
Enter the frequency of the first SHM, omega1=90
Enter the frequency of the second SHM,omega2=45
Enter the phase difference, delta =45
ahmed@ahmed-PC:~/ISHAQ-PC/Classes-Nov2019/TEST$
```

Figure 2.1: The command line arguments to plot the beats and other phenomena. The values for a_1, a_2, ω_1, ω_2 and δ are illustrative and can be changed as desired

The command line arguments in the `gnuplot` window to plot the figures is shown in Fig. (2.2).

```
File  Edit  View  Search  Terminal  Help
gnuplot> set border lw 4 lc rgb "blue"
gnuplot> set grid lw 2 lc rgb "blue"
gnuplot> set title "Phenomenon of Beats"
gnuplot> set xlabel "Time t"
gnuplot> set ylabel "Z(t)"
gnuplot> unset key
gnuplot> pl 'beats_59xx.dat' u 1:2 w l lw 3 lc rgb "red"
gnuplot>
```

Figure 2.2: The command line arguments in the `gnuplot` window to plot the beats phenomenon is shown.

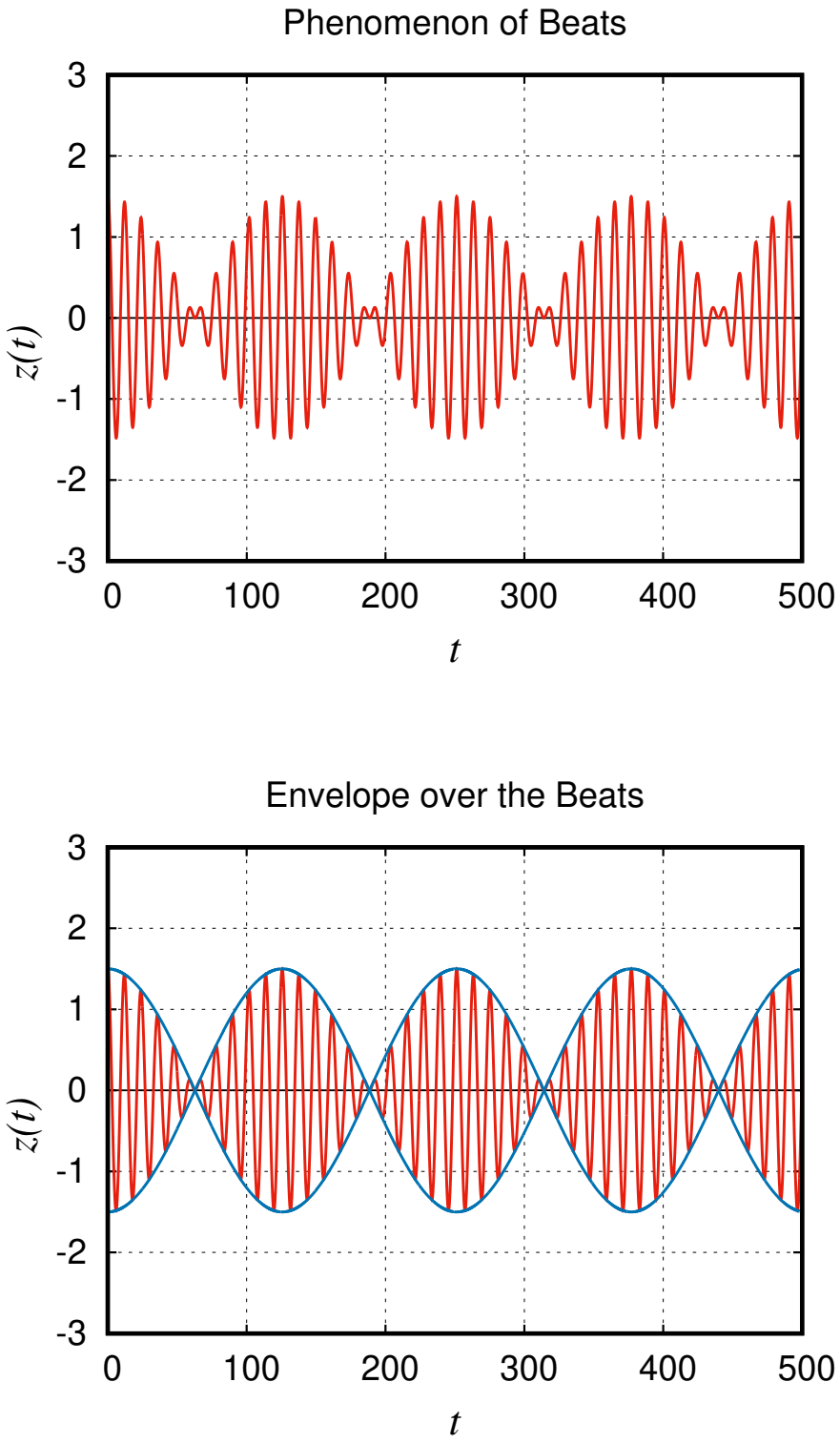

Figure 2.3: (a) The beats observed for $a_1 = a_2 = 0.75$ and $\omega_1 = 0.55$ and $\omega_2 = 0.55$ and (b) the envelope satisfying the Eqn. (2.3) for the same

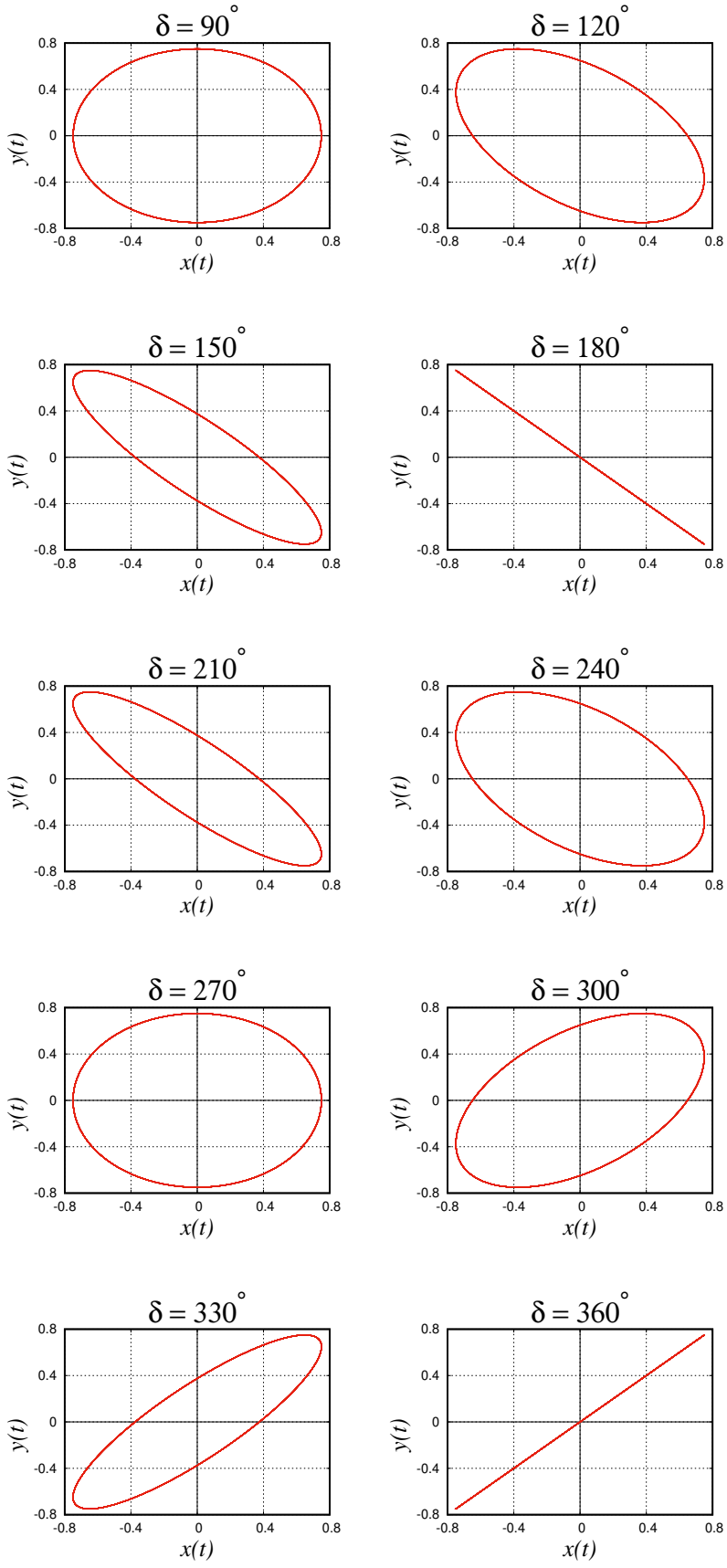

Figure 2.4: The polarization phenomena obderved for various phase differences are shown

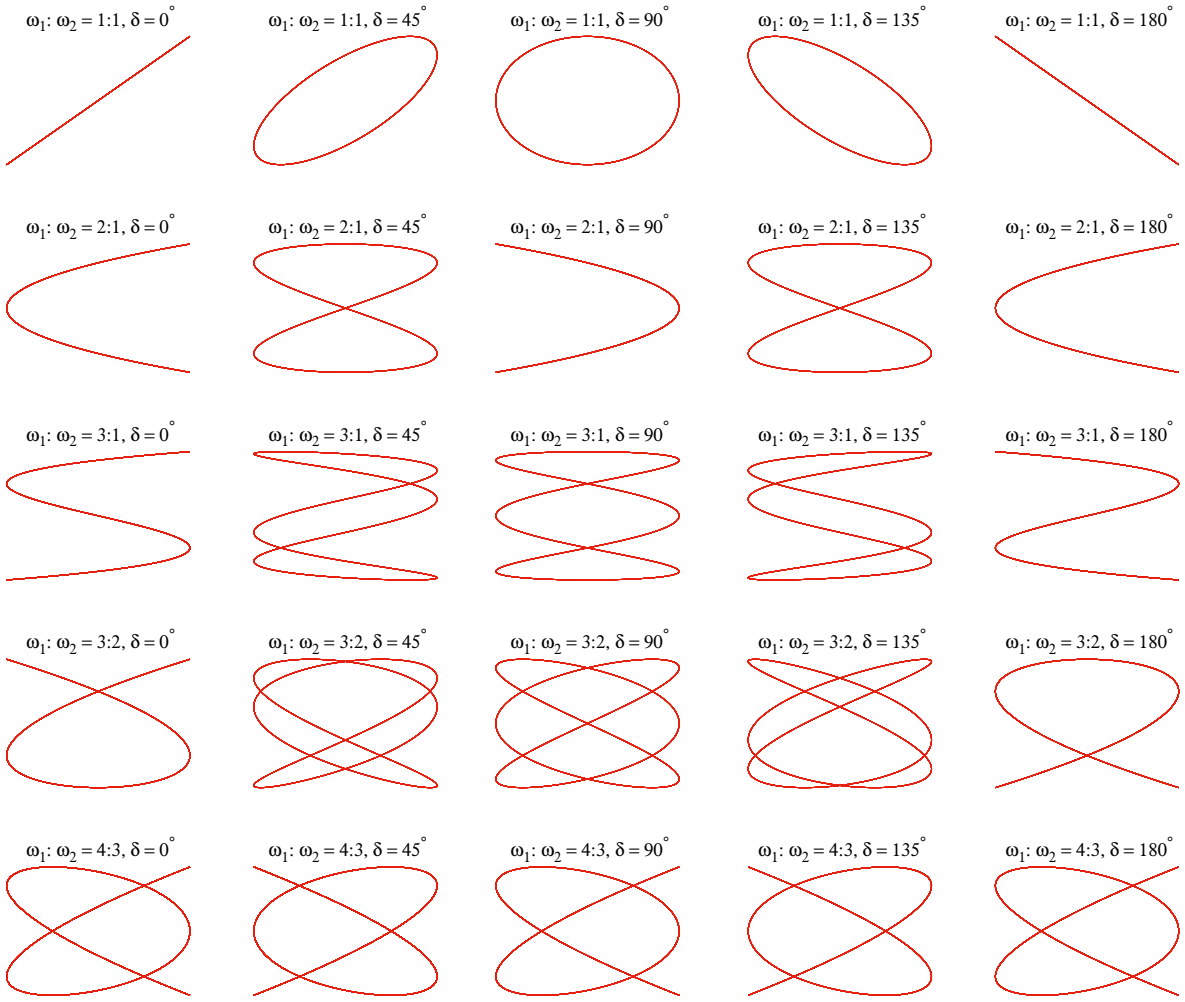

Figure 2.5: The Lissajous figures obtained for different ratios of $\omega_1 : \omega_2$ and different phase differences δ are shown.

2.4 Circular and Elliptic Polarization

If two SHMs having the same frequency, but differing in phase, interact along mutually perpendicular directions, then the resultant motion will be still simple harmonic, but the trajectory will trace a curved path whose shapes depend on their phase difference. If the two oscillations are given as

$$x = a_1 cos(\omega_1 t), \tag{2.6}$$
$$y = a_2 cos(\omega_2 t + \delta), \tag{2.7}$$

where δ the phase difference between the two SHMs, then the resultant of the two SHMs is given by

$$\frac{x^2}{a_1^2} + \frac{y^2}{a_2^2} - \frac{2xy}{a_1 a_2} \cos\delta = \sin^2\delta \tag{2.8}$$

Eqn. (2.8) is a general equation for an ellipse.

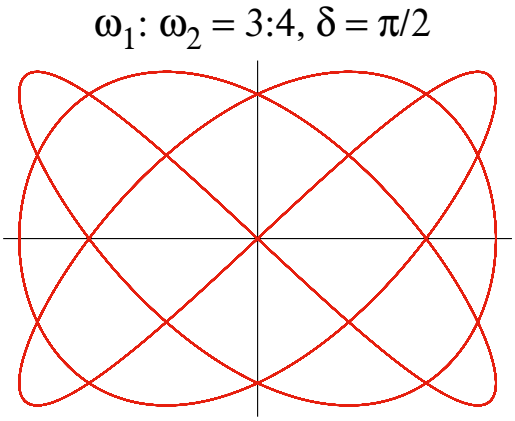

Figure 2.6: The Lissajous figure obtained for $\omega_1 : \omega_2 = 3 : 4$ and phase difference $\delta = \pi/2$ is shown.

Case 1: $(\delta = 0)$

Here Eqn. (2.7) reduce to

$$\left(\frac{x}{a_1} - \frac{y}{a_2}\right)^2 = 0.$$

or

$$y = \frac{a_2}{a_1}x. \tag{2.9}$$

Hence the trajectory is a straight line.

Case 2: $(\delta = \pm\pi)$

Here Eqn. (2.7) reduces to

$$\left(\frac{x}{a_1} + \frac{y}{a_2}\right)^2 = 0.$$

or

$$y = -\frac{a_2}{a_1}x. \tag{2.10}$$

Here also the trajectory is a straight line but with a negative slope.

Case 3(a): $(\delta = \pm\pi/2)$

Here Eqn. (2.7) reduces to

$$\frac{x^2}{a_1^2} + \frac{y^2}{a_2^2} = 1 \tag{2.11}$$

This equation (2.11)is the familiar equation for an ellipse. Hence the resultant of the two SHM will trace an ellipse.

Case 3(b): $(\delta = \pm\pi/2, a_1 = a_2 = a)$

Here Eqn. (2.7) reduces to

$$\frac{x^2}{a^2} + \frac{y^2}{a^2} = 1 \tag{2.12}$$

This is the familiar equation for a circle. Hence the resultant of the two SHM will therefore trace an circle.

The same C program `shm.c` is run to plot the paths of polarizations. Here we assume the values of the parameters as $a_1 = a_2 = 0.75$, $\omega_1 = \omega_2 = 0.5$ and the successive values of the phase differences as $\delta = 90^o, 120^o, 150^o, 180^o, 210^o, 240^o, 270^o, 300^o, 330^o$ and 360^o. The numerical results are stored in the data file `polarization_59xx.dat`. The polarization phenomena observed for various phase differences are shown in Fig. (2.4).

2.5 Lissajous Figures

If two SHMs differing in frequency by a small amount, say $(\Delta\omega)$ interact along mutually perpendicular directions, then they can be considered as oscillations of an identical frequency, but with a slowly changing phase difference. Let the two oscillations are given as

$$x = a_1 cos(\omega_1 t), \tag{2.13}$$

$$y = a_2 cos[\omega_2 t + (\Delta\omega + \delta)]. \tag{2.14}$$

Then the expression $(\Delta\omega + \delta)$ can be considered as the phase difference changing with time according to a linear law. The resultant of the two SHMs will trace many intricate curves called as *Lissajous Figures*. The Lissajous figures for various values of ω_1 and ω_2 and the phase differences δ are shown in Fig. (2.5). The Lissajous figure for $\omega_1 : \omega_2 = 3 : 4$ and phase difference $\delta = \pi/2$ is shown in Fig. (2.6).

2.6 Result

The superposition of two simple harmonic motions have been made and

1. Beats phenomenon has been observed
2. Circular and elliptic polarization phenomena for different phase differences δ have been observed
3. Lissajous figures for various frequencies $\omega_1 : omega_2$ and phase differences δ have been observed.

Chapter 3

Lagrange's Interpolation: Nuclear Scattering Energies

3.1 Aim

To interpolate the set of data for the nuclear scattering cross-section as a function of the energies of the projectile particles using Lagrange's interpolation scheme.

3.2 Theoretical Details

The Lagrange interpolation is a well known, classical technique for interpolation. Given a set of $N+1$ known samples $y(x_i)$, $i = 0, 1, 2, ...N$, the problem is to find the *unique* order N polynomial $f_N(x)$ which interpolates the samples. The solution can be expressed as a linear combination of elementary N^{th} order polynomials:

$$f_N(x) = \Sigma_{i=0}^{N} \lambda_i(x) y(x_i), \tag{3.1}$$

where

$$\lambda_i(x) \triangleq \frac{(x-x_0)....(x-x_{j-1})(x-x_{j+1})....(x-x_N)}{(x_i-x_0)....(x_i-x_{j-1})(x_i-x_{j+1})....(x_i-x_N)} \tag{3.2}$$

The above equation can be elegantly written as

$$\lambda_i(x) \triangleq \Pi_{j=0, j\neq i}^{N} \frac{(x-x_j)}{(x_i-x_j)} \tag{3.3}$$

Here $\lambda_i(x_j)$ can be interpreted as a polynomial having zeros at all of the samples except at the i^{th} value, for which it is 1. Hence it can be given as

$$\lambda_i(x_j) = \delta_{ij} \triangleq \begin{cases} 1, & j = i, \\ 0, & j \neq i. \end{cases}$$

3.3 C-Program to Implement Lagranges Interpolation

The C code `lag_59xx.c` to implement Lagrange's Interpolation Scheme for the data provided is given below.

Energy (MeV)	Scattering Crossection (σ) milli-Barns
0	10.6
25	16.0
50	45.0
75	83.5
100	52.8
125	19.9
150	10.8
175	8.25
200	4.7

```
// lag_59xx.c
// To interploate the given set of data using Lagrangian
// Interpolation
// Roll No:59xx
// Date    :02/01/2020

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

double inter(double[],double[],int,double);
void main()
{
        int i,j,n;
    double x,h,xin[20],yin[20],f_x;

        FILE*fp1,*fp2;

    fp1=fopen("lag_59xx_1.dat","w");
    if(fp1==NULL){printf("File open error!\n");return;}

    fp2=fopen("lag_59xx_2.dat","w");
        if(fp2==NULL){printf("File open error!\n");return;}

    h = 0.2;

        printf("n = ");
    scanf("%d",&n);

        for(i=1;i<=n;i++){
        printf("\nxin[%d]= ",i);
        scanf("%lf",&xin[i]);

                printf("yin[%d]= ",i);
        scanf("%lf",&yin[i]);
        }
```

```
36      for(i=1;i<n;i++)
37                      fprintf(fp1,"%12.6lf%12.6lf\n",xin[i],yin[i]);
38
39              for(i=1;i<=1000;i++){
40              x=h*(double)(i);
41              f_x=inter(xin,yin,n,x);
42                      fprintf(fp2,"%12.6lf%12.6lf\n",x,f_x);
43              }
44      fclose(fp1);
45      fclose(fp2);
46  }
47  double inter(double xin[20],double yin[20],int n,double x)
48  {
49              int i,j;
50      double f,lambda[20];
51
52      f=0;
53      for(i=1;i<=n;i++){
54                      lambda[i]=1.0;
55              for(j=1;j<=n;j++){
56                              if(i!=j)
57                              lambda[i]=lambda[i]*((x-xin[j])/(xin[i]-xin[
    j]));
58                      }
59              f=f+yin[i]*lambda[i];
60       }
61              return f;
62  }
```

The command line arguments to compile and run the C code `lag_59xx.c` is shown in Fig. (3.1).

```
File  Edit  View  Search  Terminal  Help
ahmed@ahmed-PC:~/ISHAQ-PC/Classes-Nov2019/TEST/tested$ clear

ahmed@ahmed-PC:~/ISHAQ-PC/Classes-Nov2019/TEST/tested$ cc lag_59xx.c -lm
ahmed@ahmed-PC:~/ISHAQ-PC/Classes-Nov2019/TEST/tested$ ./a.out
n = 9

xin[1]= 0
yin[1]= 10.6

xin[2]= 25
yin[2]= 16.0

xin[3]= 50
yin[3]= 45.0

xin[4]= 75
yin[4]= 83.5

xin[5]= 100
yin[5]= 52.8

xin[6]= 125
yin[6]= 19.9

xin[7]= 150
yin[7]= 10.8

xin[8]= 175
yin[8]= 8.25

xin[9]= 200
yin[9]= 4.7
ahmed@ahmed-PC:~/ISHAQ-PC/Classes-Nov2019/TEST/tested$
```

Figure 3.1: The command line arguments for compiling and running the C code `lag_59xx.c`

```
gnuplot> set border lw 4
gnuplot> set grid lw 3
gnuplot> set title "{/Times-Italic=15 Scattering Crossection ({/Symbol=15 \\163}) Graph}"
gnuplot> set xrange [0:200]
gnuplot> set xtics 0,40
gnuplot> set yrange [0:100]
gnuplot> set ytics 0,20
gnuplot> set xlabel "Energy E (MeV)"
gnuplot> set ylabel "{/Symbol=15 \\163}(milli-Barns)"
gnuplot> pl 'lag_59xx_2.dat' u 1:2 w l lw 3 lc rgb "blue"
gnuplot> rep 'lag_59xx_1.dat' u 1:2 w p pt 7 ps 2 lc rgb "red"
gnuplot>
```

Figure 3.2: Command line arguments for obtaining Scattering Cross-section σ using gnuplot

Graph

A snapshot of the command line arguments in the `gnuplot` window to plot the graph is shown in Fig. (3.2).The interpolated graph using Lagange's scheme is shown in Fig. (3.3).

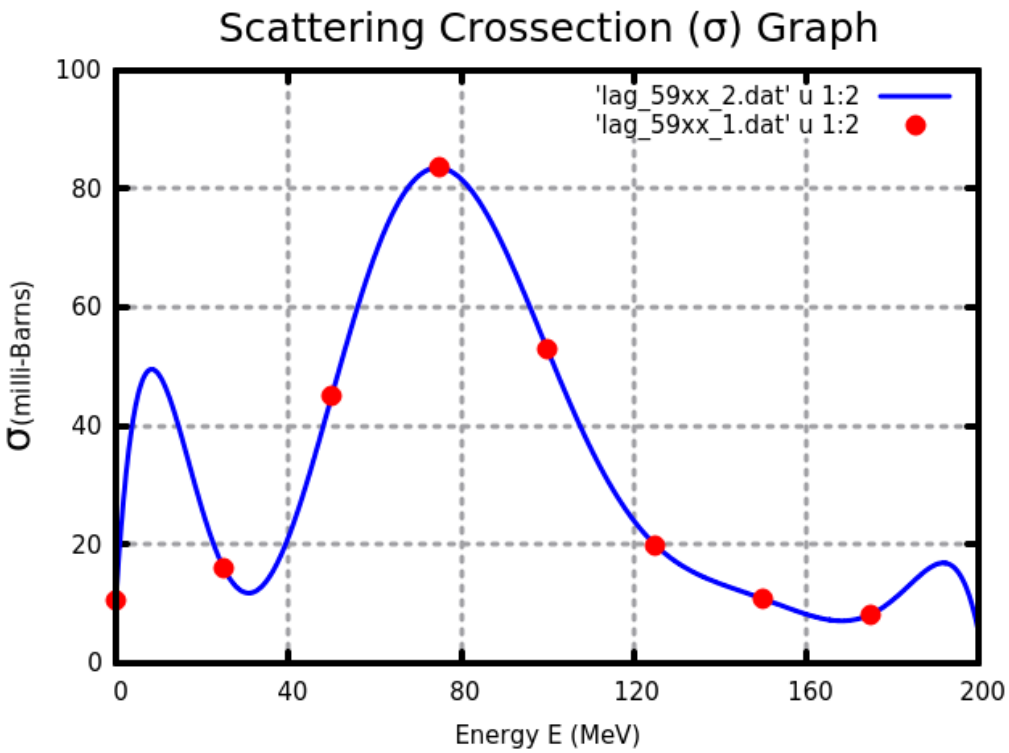

Figure 3.3: Plot of the Scattering Cross-section σ as a function of the energy of the projectile particles.

3.4 Result

The given data for the scattering of neutrons is interpolated using Lagrange's Interpolation scheme and the nuclear cross-sections for a wide range of neutron energies from 0 to 200 MeV is plotted using gnuplot.

Chapter 4

Simulation of Brownian Motion in a Fluid

4.1 Aim

1. To generate random numbers using Park – Miller Algorithm and
2. To simulate Brownian motion and hence find the root mean sqare displacement for the particles executing Brownian Motion.

4.2 Theoretical Details

Brownian motion refers to the irregular motion of small particles suspended in a liquid or a gas, due to their bombardment by molecules of the medium. This was first observed by Robert Brown in 1827. This can be simulated on a Personal Computer using random numbers.

4.3 Random Numbers Generation

Random numbers are the values taken by random variables. These sequences of values are generated by physical process like radio activity, diffusion, percolation etc will be truly random (i.e.) they will be unpredictable and hence irreproducible. A true sequence of random numbers cannot therefore be generated using a computer. However there are a large number of pseudo-random number generating algorithms, which can generate seemingly random numbers. As these generators are based on strict mathematical formulas, they can be easily reproduced.
In this experiment we use the Park-Miller algorithm for the Lehmer's modulus multiplicative congruential generator. This Park-Miller algorithm is given as follows.
Here there are four variables a, m,q and r which are initialized as given below.

$$\begin{aligned} a &= 16807 \qquad \left(7^5\right) \\ m &= 2147483647 \qquad \left(2^{31}-1\right) \\ q &= 127773 \qquad (m/a) \\ r &= 2836 \qquad (m \mod a)\,, \end{aligned}$$

where *mod* stands for modulo division. Find

```
low = SD mod  q
high = SD/q
test = a*low-r*high
if(test>0)
        SD = test
else
        SD = test + m

ran_no = SD/m
```

The subroutine function `mmc_ gen()` implements this algorithm. It uses a SEED (SD) which can be any number between 1 and 2^{45}. In our program we assume it for simplicity as $SD = 12345$.

4.4 Brownian Motion

If the *x-coordinates* and *y-coordinates* of the particles in a liquid, say pollen grains, are chosen to be random variables, then the zig-zag motion of these particles, can be simulated numerically. The root mean square displacement suffered by the particles (pollen grains) is given by the formula

$$RMSD = \sqrt{\lambda}$$

where λ is the average of the squares of the displacements suffered by the particles between successive collisions with the molecules of the liquid and is given as

$$\lambda = \Sigma_{i=0}^{n} \left[(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2\right]$$

4.5 C - Program to Simulate Brownian Motion

A program `brownian_59xx.c` to generate random numbers and simulate Brownian motion is given.

```
// brownian_xx.c
// C Program to simulate Brownian Motion and find
// the Root Mean Square Displacement using Random Numbers
// Roll No: 59xx
// Date   : 09/01/2020

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

long SD;
double mmc_gen();

int main()
{
        int i,n;
        double x[201]={},y[201]={},pi,lambda,RMSD,p,q;
```

```
17
18          FILE *fp;
19          fp= fopen("brownian_59xx.dat","w");
20          if(fp==NULL){printf("File Open Error!\n"); return 0;}
21
22          // any five digit number, say SD = 12345
23          printf("Seed SD = ");
24          scanf("%ld",&SD);
25
26          n =200;
27
28          for(i=1;i<=n;i++){
29                  p = mmc_gen();
30                  q = mmc_gen();
31
32                  x[i] = p;
33                  y[i] = q;
34
35          }
36
37          printf("-------------------------------------------------\n");
38          printf(" Table of a few Random Numbers \n");
39          printf("-------------------------------------------------\n");
40
41          for(i=1;i<=10;i++)
42                  printf("%5d\t\t%12.6lf\n",i,x[i]);
43
44          printf("-------------------------------------------------\n");
45
46          lambda = 0.0;
47          for(i=2;i<=n;i++){
48                  lambda = lambda + sqrt(pow((x[i-1]-x[i]),2.0)+pow((y
    [i-1]-y[i]),2.0));
49                  if(i>125)
50                  fprintf(fp,"%12.6lf%12.6lf\n",x[i],y[i]);
51          }
52          RMSD = lambda/(double)(n);
53
54          printf(" The Root Mean Square Displacement is RMSD = %12.6lf
    \n",RMSD);
55
56          fclose(fp);
57  }
58  double mmc_gen()
59  {
60          long m,a,q,r,high,low,test;
61          double ran_no;
```

```
62
63
64          m = 2147483647;
65          a = 16807;
66          q = 127773;
67          r = 2836;
68
69          high = SD/q;
70          low  = SD%q;
71          test = a*low-r*high;
72
73          if(test>0)
74                  SD = test;
75          else
76                  SD = test + m;
77
78          ran_no = (double)(SD)/(double)(m);
79
80          return ran_no;
81  }
```

The program on being run prints out a table of ten random numbers as shown in Fig. (4.1).

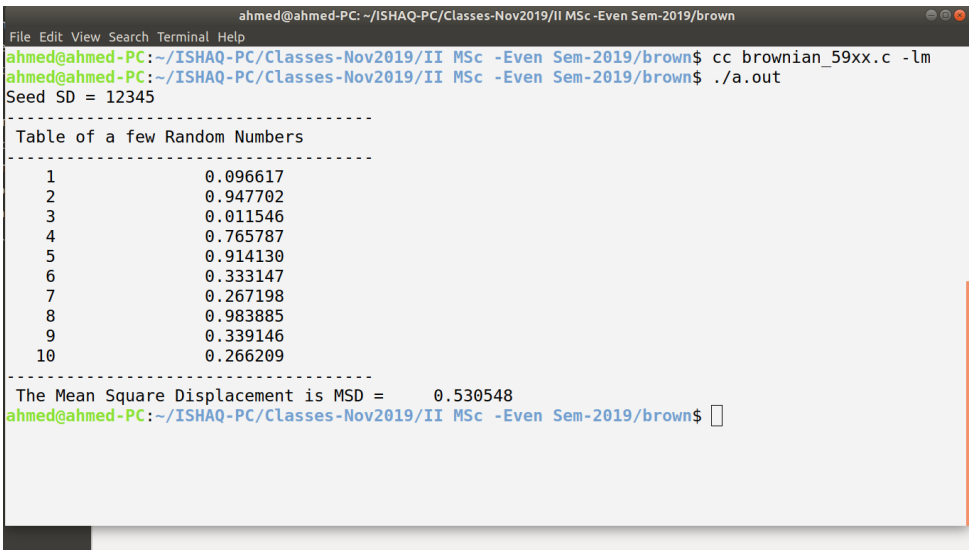

Figure 4.1: The command line arguments for compiling and executing the program `brownian_59xx.c` and the table of ten random numbers printed on the screen as the output.

4.6 Brownian Motion Simulation Graph

The command line arguments for obtaining the graph simulating the Brownian Motion is shown in Fig. (4.2).

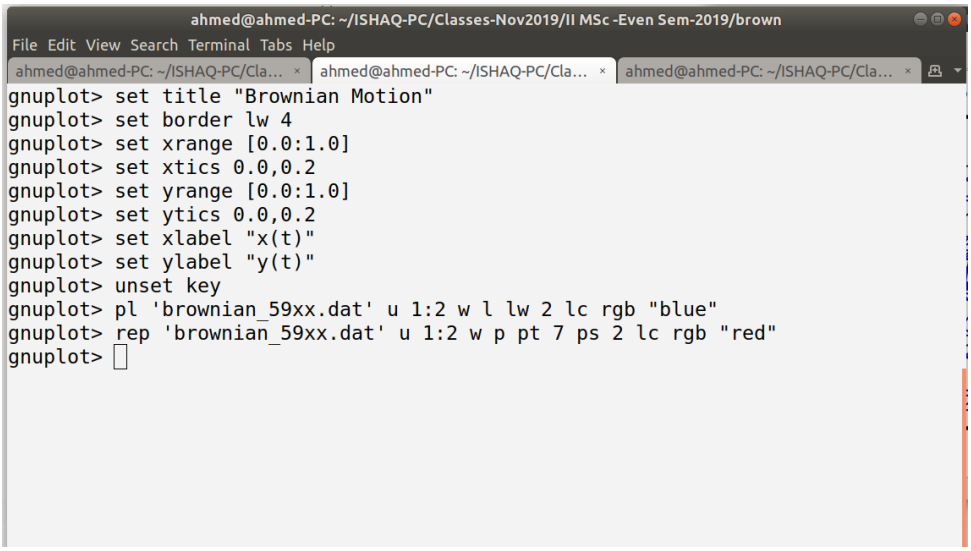

Figure 4.2: The command line arguments for plotting the Brownian Motion using `gnuplot`.

The simulation of the Brownian Motion is shown in Fig. (4.3).

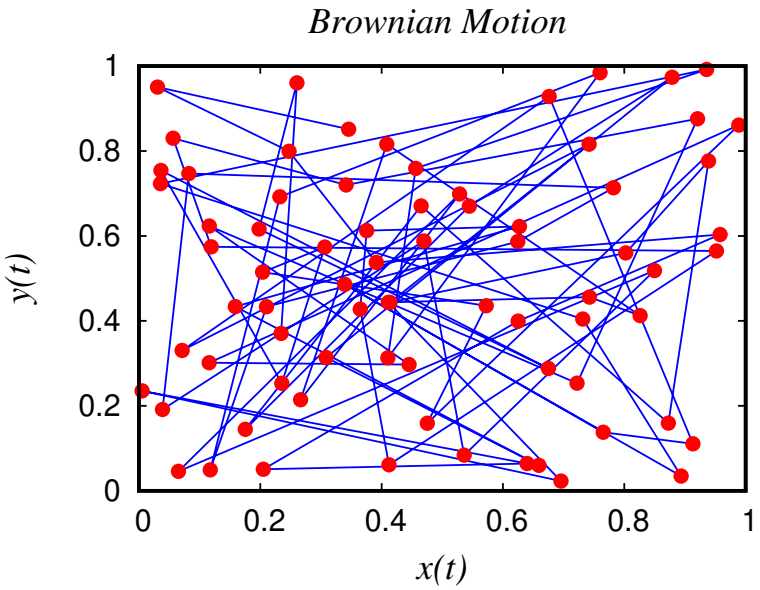

Figure 4.3: The simulation plot of the Brownian Motion.

4.7 Result

In this experiment

1. random numbers were generated and a few of them were printed on the screen.
2. the Brownian motion was simulated and the root mean square displacement was determined as $RMSD =$

Chapter 5

Simulation of Radioactive Decay

5.1 Aim

1. To generate random numbers using Park – Miller Algorithm and
2. To simulate and verify the law of radio active decay using Monte-Carlo Simulation.
3. To determine the radio decay constant λ using the *exponential fit* function of `gnuplot`

5.2 Theoretical Details

The phenomenon of radio activity is the spontaneous decay of nuclei, accompanied by emission of one more particles. If we have at any instant of time, say t, a large number say N_o of radio active nuclei and if dN nuclei decay on the average in a time interval dt, then

$$dN = -\lambda N dt, \tag{5.1}$$

where λ is the radio active decay constant. Integrating Eq. (5.1) gives

$$N = N_o e^{-\lambda t}, \tag{5.2}$$

where N is the instantaneous number of nuclei undergoing radio active decay.

5.3 Radio Activity as a Random Process

The phenomenon of radio activity is a random process involving two mutually exclusive events.

Event A: A nucleus either undergoes a radioactive decay, or

Event B: A nucleus does not undergo decay, that is, it is being stable.

Using random numbers and Monte Carlo technique we can simulate this. Let N_o (number of parent nuclei) be the total radio active nuclei in a given sample. Let the probability for a nucleus to decay in a time t be p. Let N be the instantaneous number of nuclei that undergo decay. Let D (number of daughter nuclei) be the number of the nuclei formed as a result of radioactive process. To each nucleus we assign a random number $x(t)$, lying in the interval $[0, 1]$. If $x < p$, let us assume the nucleus has decayed. Then we decrement the number of nuclei left N by one and increment the number of daughter nuclei D by one. This process is repeated for successive intervals till no parent is left.

5.4 C Code to Simulate Radioactive Decay

The C program `decay_59xx.c` to simulate the radioactive decay is given below. The initial number of radioactive nuclei is assumed as $N_o = 1000$ and the maximum time is assumed as $T_max = 25$. The `SEED` for the random number generator is assumed as $SD = 12345$. The program is run and the data generated is stored in the data file `decay_59xx.dat`. The output is plotted as shown in Fig. (5.3). Here the number of nuclei undergoing radioactive decay N is shown to vary as a function of time t.

```
// decay_59xx.cpp
// To simulate radioactive decay.
// Roll No      :
// Date         :10/02/2020

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

long SD;
void main()
{
        int i,No,Do,NU,N,D,T,T_max;
        double x,p;
        double mmc_gen();

        FILE *fp;

        fp = fopen("decay_59xx.dat","w");
        if(fp == NULL){ printf(" File Open Error!\n"); return;}

        printf(" T_max = ");
        scanf("%d",&T_max);

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

        printf("-------------------------------------------------\n");
        printf(" Table of a few Random Numbers \n");
        printf("-------------------------------------------------\n");

        for(i=1;i<=10;i++){
                p = mmc_gen();
                printf("%5d\t\t%12.6lf\n",i,p);
        }
        printf("-------------------------------------------------\n");

        N = 1000;
        D = 0;
        p = 0.2;
```

```
40          T = 0;
41
42          while((N>0)&&(T<=T_max)){
43                  NU = N;
44                  fprintf(fp,"%15d%15d%15d\n",T,N,D);
45                  for(i=1;i<=NU;i++){
46                          x = mmc_gen();
47                          if(x<=p){
48                                  N-= 1;
49                                  D+= 1;
50                          }
51                  }
52                  T++;
53          }
54          fclose(fp);
55  }
56  double mmc_gen()
57  {
58          long m,a,q,r,high,low,test;
59          double ran_no;
60
61
62          m = 2147483647;
63          a = 16807;
64          q = 127773;
65          r = 2836;
66
67          high = SD/q;
68          low  = SD%q;
69          test = a*low-r*high;
70
71          if(test>0)
72                  SD = test;
73          else
74                  SD = test + m;
75
76          ran_no = (double)(SD)/(double)(m);
77
78          return ran_no;
79  }
```

The command line arguments to compile the program `decay_59xx.c` is given in Fig. (5.1). The command line arguments on the `gnuplot` window to obtain the plot of the radioactive decay is shown in Fig. (5.2). The exponential decrease in the number of radioactive nuclei N as a function of time t is shown in Fig. (5.3).

```
File Edit View Search Terminal Tabs Help
ahmed@ahmed-PC: ~/ISHAQ-PC/Classes-Nov2019/II MSc -Even Sem-2019/decay
ahmed@ahmed-PC: ~/ISHAQ-PC/Classes-Nov2019/II MSc -Even Sem-2019/decay
ahmed@ahmed-PC:~/ISHAQ-PC/Classes-Nov2019/II MSc -Even Sem-2019/decay$ cc decay_59xx.c -lm
ahmed@ahmed-PC:~/ISHAQ-PC/Classes-Nov2019/II MSc -Even Sem-2019/decay$ ./a.out
 T_max = 25
SD = 12345
ahmed@ahmed-PC:~/ISHAQ-PC/Classes-Nov2019/II MSc -Even Sem-2019/decay$
```

Figure 5.1: The command line arguments to compile and run the C code.

```
ahmed@ahmed-PC: ~/ISHAQ-PC/Classes-Nov2019/II MSc -Even Sem-2019/decay
File Edit View Search Terminal Help
gnuplot> set border lw 4
gnuplot> set title "Radioactive Decay"
gnuplot> set xrange [0:25]
gnuplot> set xtics 0,5
gnuplot> set yrange [0:1000]
gnuplot> set ytics 0,200
gnuplot> set xlabel "Time t"
gnuplot> set ylabel "N"
gnuplot> unset key
gnuplot> pl 'decay_59xx.dat' u 1:2 w l lw 3 lc rgb "blue"
gnuplot> rep 'decay_59xx.dat' u 1:2 w p pt 6 ps 2 lc rgb "red"
gnuplot>
```

Figure 5.2: The command line arguments on the `gnuplot` window to obtain the plot of the radioactive decay.

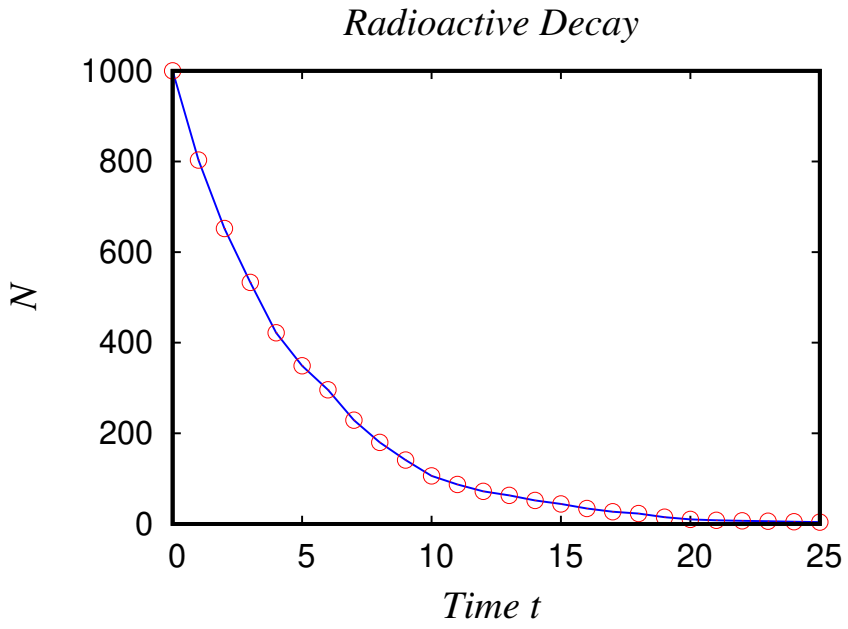

Figure 5.3: The exponential decrease in the number of radioactive nuclei N as a function of time t.

```
ahmed@ahmed-PC: ~/ISHAQ-PC/Classes-Nov2019/II MSc -Even Sem-2019/decay
File  Edit  View  Search  Terminal  Help
gnuplot> a = 1000
gnuplot> f(x) = a*exp(-x*b)
gnuplot> fit f(x) 'decay_59xx.dat' u 1:2 via a,b
iter      chisq       delta/lim  lambda   a             b
   0 1.1956890938e+06   0.00e+00  1.64e+02    1.000000e+03   1.000000e+00
   * 1.6227537477e+19   1.00e+05  1.64e+03    9.788410e+02  -6.020781e-01
   1 1.0319256084e+06  -1.59e+04  1.64e+02    1.062413e+03   8.601645e-01
   * 2.9602289047e+14   1.00e+05  1.64e+03    9.612868e+02  -3.790553e-01
   2 8.0248497188e+05  -2.86e+04  1.64e+02    1.107556e+03   6.914935e-01
   * 7.5189140989e+09   1.00e+05  1.64e+03    9.468066e+02  -1.542265e-01
   3 4.6408241178e+05  -7.29e+04  1.64e+02    1.133128e+03   4.957953e-01
   * 2.6835538311e+06   8.27e+04  1.64e+03    9.486437e+02   5.852923e-02
   4 1.0380183455e+05  -3.47e+05  1.64e+02    1.126590e+03   3.130897e-01
   5 1.0264896046e+04  -9.11e+05  1.64e+01    9.825590e+02   1.922727e-01
   6 8.1975514087e+02  -1.15e+06  1.64e+00    9.985551e+02   2.117531e-01
   7 7.8689819999e+02  -4.18e+03  1.64e-01    9.998840e+02   2.131025e-01
   8 7.8689701190e+02  -1.51e-01  1.64e-02    9.998619e+02   2.130926e-01
iter      chisq       delta/lim  lambda   a             b

After 8 iterations the fit converged.
final sum of squares of residuals : 786.897
rel. change during last iteration : -1.50983e-06

degrees of freedom    (FIT_NDF)                        : 24
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 5.72603
variance of residuals (reduced chisquare) = WSSR/ndf   : 32.7874

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a               = 999.862          +/- 4.339        (0.4339%)
b               = 0.213093         +/- 0.001451     (0.6811%)

correlation matrix of the fit parameters:
                a      b
a               1.000
b               0.629  1.000
gnuplot> print a,b
999.861850428528 0.213092638787021
gnuplot>
```

Figure 5.4: To find the decay constant using exponential fit function of `gnuplot`.

5.5 To Find The Decay Constant

The decay constant λ can be found using the *curve fitting* utility of gnuplot. This can be done as follows

- Assume *No* as a and initialize it to 1000.
- Write the expression for the fit: $f(x) = a * exp(-x * b)$, where b is assumed as the decay constant λ
- Fit the data to the expression given : fit f(x) 'decay_59xx.dat' u 1:2 via a,b
- print a,b

The value of the decay constant $\lambda = b$ as given by the exponential fit curve is shown in Fig. (5.4).

5.6 Result

1. Random Numbers are generated using Park-Miller Algorithm.
2. The law of radioactive decay is verified numerically using Monte-Carlo Simulation and the graphical plot obtained using `gnuplot`.
3. The decay constant λ is found using exponential fit function of `gnuplot` as $b = 0.21309$.

Chapter 6

Simulation of Projectile Motion: Euler's Method

6.1 Aim

1. To plot the motion of a projectile particle for various angles of projection using Euler's Method.

2. To find the maximum time of flight (T), maximum range (R) and the maximum height (H) reached by the projectile particle for various angles of projection.

6.2 Theoretical Details

The motion of a body that is projected from a point on the earth so as to move along a predetermined path is called as *projectile motion*. Examples of such projectile motion are
(i) the firing of rockets
(ii) the firing of missiles to hit enemy targets

Assumptions

To study the motion of the projectile the following assumptions are made

1. The resistance offered by the air medium to the motion of the projectile is zero.

2. The acceleration due to gravity g remains a constant through out the path of the projectile.

3. In the absence of spinning motion of the projectile, the trajectory of the projectile is restricted to a two dimensional plane, say $(x - y)$ plane.

The Newton's equation of motion in its general form is

$$m\frac{d^2r}{dt^2} = f\left(t, r, \dot{r}\right) \tag{6.1}$$

However the projectile motion is a simple case wherein the forces in each directions are uncoupled, leading to separation of the equation, Eqn. (6.1) into three equations along the the three

dimensions x, y, z, namely

$$\begin{aligned} m\frac{d^2x}{dt^2} &= f\left(t, x, \dot{x}\right), \\ m\frac{d^2y}{dt^2} &= f\left(t, x, \dot{y}\right), \\ m\frac{d^2z}{dt^2} &= f\left(t, x, \dot{z}\right). \end{aligned} \tag{6.2}$$

As the projectile motion is two dimensional, say along the x and y the first two of the Eqns. (6.2) can be used for our calculation. Further as the acceleration of the particle along the horizantal direction is zero and as the accleration of the particle along the vertical direction is the acceleration due to gravity (which is acting downwards), these equations can be written as

$$\begin{aligned} m\frac{d^2x}{dt^2} &= 0, \qquad \text{or} \\ \frac{d^2x}{dt^2} &= 0. \end{aligned} \tag{6.3}$$

$$\begin{aligned} m\frac{d^2y}{dt^2} &= -mg, \qquad \text{or} \\ \frac{d^2y}{dt^2} &= -g. \end{aligned} \tag{6.4}$$

The Equations (6.3) and (6.4) are second order differential equations. In order to apply the Euler method, we convert each of these equations into two one dimensional ones as

$$\begin{aligned} \frac{dx}{dt} &= v_x, \\ \frac{dv_x}{dt} &= 0. \end{aligned} \tag{6.5}$$

$$\begin{aligned} \frac{dy}{dt} &= v_y, \\ \frac{dv_y}{dt} &= v_y - g * t. \end{aligned} \tag{6.6}$$

Parameters of Projectile Motion

1. Maximum time of flight (T) is given as

$$T = \frac{2\sin(\alpha)}{g}. \tag{6.7}$$

2. The maximum range of the projectile particle (R) is given as

$$R = \frac{u^2\sin(2\alpha)}{g}. \tag{6.8}$$

3. The maximum height reached by the projectile (H) is

$$H = \frac{u^2\sin^2(\alpha)}{2g}. \tag{6.9}$$

6.3 Euler Algorithm to solve ODE

Generally to solve an ordinary differential equation

$$\frac{dx}{dt} = f\left(t, x, \dot{x}\right) \tag{6.10}$$

using Euler's method, we use the Taylor series expansion of the function $f\left(t, x, \dot{x}\right)$ and discretise it as

$$\begin{aligned} t^{n+1} &= t^n + \Delta t, \\ x^{n+1} &= x^n + \Delta x f\left(t, x, \dot{x}\right). \end{aligned} \tag{6.11}$$

and iterate the Eqns. (6.11) for *n-times* to arrive at our desired answer.
Using this algorithm, the equations for the motion of the projectile are reduced as

$$\begin{aligned} t^{n+1} &= t^n + h, \\ x^{n+1} &= x^n + v_x * h, \\ v_x^{n+1} &= v_x^n, \\ y^{n+1} &= y^n + v_y * h, \\ v_y^{n+1} &= v_y^n - g * h. \end{aligned} \tag{6.12}$$

where $h = \Delta t$, $v_x = ucos(\alpha)$, $v_y = usin(\alpha)$ and α is the angle of projection.

6.4 C Code to Simulate Projectile Motion

The C program `euler_proj_59xx.c` to simulate the projectile motion is given below. Here the step size is assumed as $h = 0.01$, the initial velocity is assumed as $u = 20$ while the acceleration due to gravity is assumed as $g = 9.8$. The program is run for various angles of projection $\alpha = 15^o, 30^o, 45^o, 60^o, 75^o, 90^o$ and the data are stored in the file `euler_proj_59xx.dat` in each case. Then the data are plotted using `gnuplot`. Further the parameters of the projectile particle such as *maximum time of flight*, *maximum range* and *maximum height* reached by the projectile are calculated.

Note: *The angles are to be converted into radians so as to get the correct results.*

```
1  // euler_proj_59xx.c
2  // C program to simulate the motion of a projectile
3  // using Euler method
4  // Roll no: 59xx
5  // Date    : 18/02/2020
6
7  #include<stdio.h>
8  #include<math.h>
9  double g,h,x,y,vx,vy,t,alpha,u;
10
11
12
```

```
13 int main()
14 {
15         double angle,pi,t_max;
16         double R,H,T;
17         void EULER(double);
18
19         FILE *fp1;
20         fp1=fopen("euler_proj_59xx.dat","w");
21         if(fp1==NULL){printf("File-1 Open Error!\n");return 0;}
22
23         h = 0.01; // step size or delta_t
24
25         g = 9.8;
26         u = 20.0;
27         pi = 4.0*atan(1.0);
28
29         // angles alpha = 15,30,45,60,75 degrees
30
31         printf("Enter the angle of projection, alpha = ");
32         scanf("%lf",&alpha);
33
34         alpha = (pi*alpha)/180; // to convert the angle into radians
35
36         t=0.0;
37         x=0.0;
38         y=0.0;
39
40         vx=u*cos(alpha);
41         vy=u*sin(alpha);
42         t_max = (2.0*u*sin(alpha))/g;
43
44         do{
45                 EULER(t);
46                 fprintf(fp1,"%12.6lf%12.6lf%12.6lf\n",t,x,y);
47                 t = t+h;
48         }while(t<=t_max);
49
50         T = t_max;
51         R = (pow(u,2.0)*sin(2.0*alpha))/g;
52         H = (pow(u*sin(alpha),2.0))/(2.0*g);
53
54         printf("\nThe maximum time of flight is \n");
55         printf("T = %12.6lf secs\n",t_max);
56
57         printf("\nThe maximum range is \n");
58         printf("R = %12.6lf m\n",R);
59
```

```
60          printf("\nThe maximum height is \n");
61          printf("H = %12.6lf m\n",H);
62
63          fclose(fp1);
64  }
65
66  void EULER(double t)
67  {
68          x  = x + vx*h;
69          vx = vx;
70          y  = y + vy*h;
71          vy = vy-g*h;
72  }
```

The command line argument to compile and run the programme is given in Fig. (6.1).

```
ahmed@ahmed-PC: ~/ISHAQ-PC/Classes-Nov2019/II MSc -Even Sem-2019/proj
File  Edit  View  Search  Terminal  Help
ahmed@ahmed-PC:~/ISHAQ-PC/Classes-Nov2019/II MSc -Even Sem-2019/proj$ cc euler_proj_59xx.c -lm
ahmed@ahmed-PC:~/ISHAQ-PC/Classes-Nov2019/II MSc -Even Sem-2019/proj$ ./a.out
Enter the angle of projection, alpha = 45

The maximum time of flight is
T =     2.886150 secs

The maximum range of the projectile is
R =    40.816327 m

The maximum height reached by the projectile is
H =    10.204082 m
ahmed@ahmed-PC:~/ISHAQ-PC/Classes-Nov2019/II MSc -Even Sem-2019/proj$
```

Figure 6.1: The command line arguments to compile and run the C code.

The command line argument in the `gnuplot` window to plot the trajectory of the projectile is shown in Fig. (6.2).

```
File  Edit  View  Search  Terminal  Help
        Thomas Williams, Colin Kelley and many others

        gnuplot home:     http://www.gnuplot.info
        faq, bugs, etc:   type "help FAQ"
        immediate help:   type "help"  (plot window: hit 'h')

Terminal type is now 'qt'
gnuplot> !clear

gnuplot> set border lw 4 lc rgb "blue"
gnuplot> set grid lw 2 lc rgb "blue"
gnuplot> set xlabel "Range R"
gnuplot> set ylabel "Height H"
gnuplot> unset key
gnuplot> set title "Projectile Motion"
gnuplot> pl 'euler_proj_59xx.dat' u 1:2 w l lw 3 lc rgb "red"
gnuplot>
```

Figure 6.2: The command line arguments to plot the trajectory using `gnuplot`.

The figure so obtained is shown in Fig. (6.3).

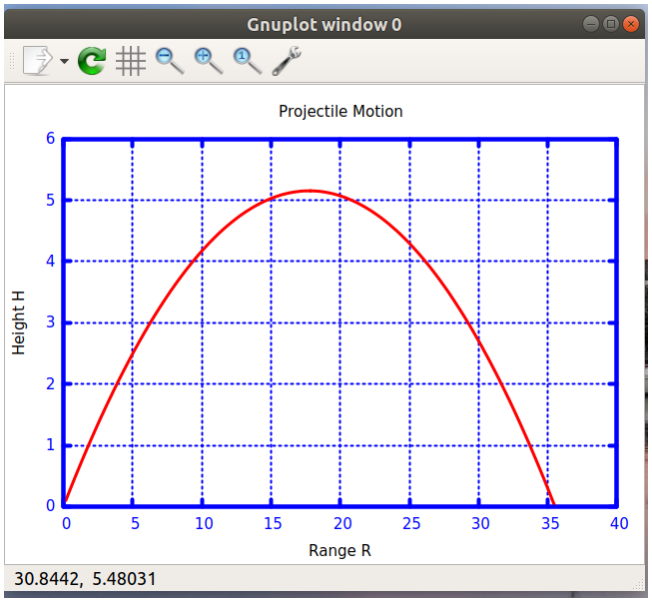

Figure 6.3: The trajectory for $\alpha = 45^o$ is shown.

In a similar manner the trajectories for various angles of projection are shown in Fig. (6.4)

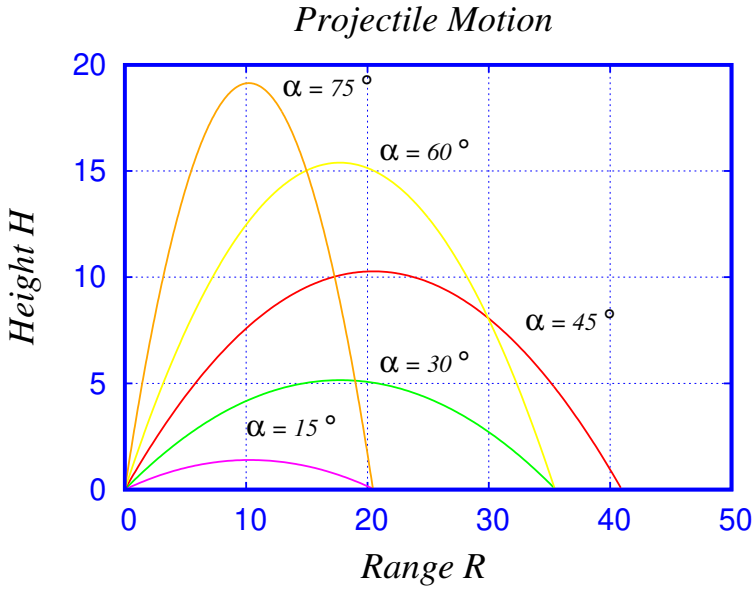

Figure 6.4: The trajectories for various angles of projection.

6.5 Result

1. The C program to observe the trajectories of projectile motion for various angles of projection using Euler's method is run and the graphs are obtained using `gnuplot`.
2. The parameters for the projectile motion for various angles of projection are also obtained and tabulated.

Angle α	Maximum Time T secs	Maximum Range R m	Maximum Height H m
15^o	1.056404	20.408163	1.367088
30^o	2.040816	35.347976	5.102041
45^o	2.886150	40.816327	10.204082
60^o	3.534798	35.347976	15.306122
75^o	3.942554	20.408163	19.041076

Chapter 7

Simpson's 1/3 Rule: Motion of a Body in a Central Potential

7.1 Aim

1. To solve an integral using Simpson's 1/3 Rule.
2. To study the motion of a body in a central potential

7.2 Motion of a Body in a Central Potential

A central force is defined as a force which is directed about a fixed point. It depends only on the radial distance and is independent of angular coordinates. Examples of central force problems are

- Motion of planets around the sun
- Motion of the electrons around the nuclei, etc.,

Integral Equation of the Orbit

The integral Equation for the orbit is

$$\theta = \int_{r_min}^{r} \frac{Jdr}{r^2\sqrt{\left[2m\left(E - V - \frac{J^2}{2mr^2}\right)\right]}} + constant \tag{7.1}$$

where m is the mass
J is the angular momentum
E is the total enerby
$V = \frac{-k}{r}$ is the potential energy
r is the radial distance
r_{min} is the minimum value of the orbital radius
of the particle in the central force field.
By solving this integral, using Simpson's 1/3 rule, the orbit of the particle in a central potential can be obtained.

7.3 Simpson's 1/3 Rule for Integration

To integrate a function $f(x)$ from the limit a to b , according to Simpson's 1/3 rule, we consider the integral to be equal to the area enclosed by the curve $f(x)$ between the limits a to b. The area is divided into n segments or strips of interval $h = (b-a)/n$. To each segment of $f(x)$ in each subinterval, a quadratic function is fitted. The algorithm is given below

$$\int_a^b f(x)dx = \frac{h}{3}\left[A + 4B + 2C\right] \tag{7.2}$$

where

$$\begin{aligned} A &= f(a) + f(b) \\ B &= \sum_{i=1,3,\ldots}^{(n-1)} f\left(a + ih\right) \\ C &= \sum_{j=2,4,6,\ldots}^{(n-2)} f\left(a + jh\right) \end{aligned} \tag{7.3}$$

7.4 C Code for Plotting the Motion of the particle in a Central Force Field

The C program to plot the orbit of the particle in a central force field is given below,

```
// simpson_59xx.c
// program to simulate the motion of a body in central potential
   using  simpsons 1/3 rule
// Roll No:
// Date    :12/03/2020

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

double SIMS(int,double, double);
double a,b,r,r_min;

void main()
{
        int i,j,n,p;
        double r_max,delta_r,theta,x,y;

        FILE*fp;
        fp=fopen("simpson_59xx.dat","w");
        if(fp==NULL){printf("file open error!\n");return;}

        n = 20;

```

```
24          r_min = 0.7;
25          a = r_min;
26
27          r_max = 1.8;
28
29          delta_r = 0.001;
30
31          i = 1;
32          b = r_min;
33          while(b<=r_max){
34                  b += delta_r*(double)(i);
35                  theta = 4.0*SIMS(n,a,b);
36                  x= a*cos(theta);
37                  y= a*sin(theta);
38                  fprintf(fp,"%12.6lf%12.6lf\n",x,y);
39                  i++;
40          }
41          fclose(fp);
42  }
43  double SIMS(int n, double a, double b)
44
45  {
46          int i,j;
47          double sum,ics,h;
48          double func(double r);
49
50          sum=func(a)+func(b);
51
52          h=(b-a)/(double)(n);
53
54          for(i=1;i<=n-1;i+=2)
55                  sum=sum+4.0*func(a+i*h);
56
57          for(j=2;j<=n-2;j+=2)
58                  sum=sum+2.0*func(a+j*h);
59
60          ics=(h/3.0)*sum;
61
62          return ics;
63
64  }
65  double func(double r)
66  {
67                  double J,m,k,q,E,V,f;
68                  J=1.0;
69                  m=1.10;
70                  k=1.0;
```

```
71                  E=-0.4;
72                  V=-k/r;
73                  q=sqrt(2.0*(m*E-m*V)-(J*J)/(r*r));
74                  f=J/(r*r*q);
75                  return f;
76  }
```

The command line argument to compile and run the programme is given in Fig. (7.1). Here a and b are the semi-minor and semi-major axes. The step size is assumed as $h = (b-a)/n$. The other parameters are assumed as $J = 1.0$, $m = 1.0$, $k = 1.0$, $E = -0.4$ and the central potential is given as $V = -k/r$.

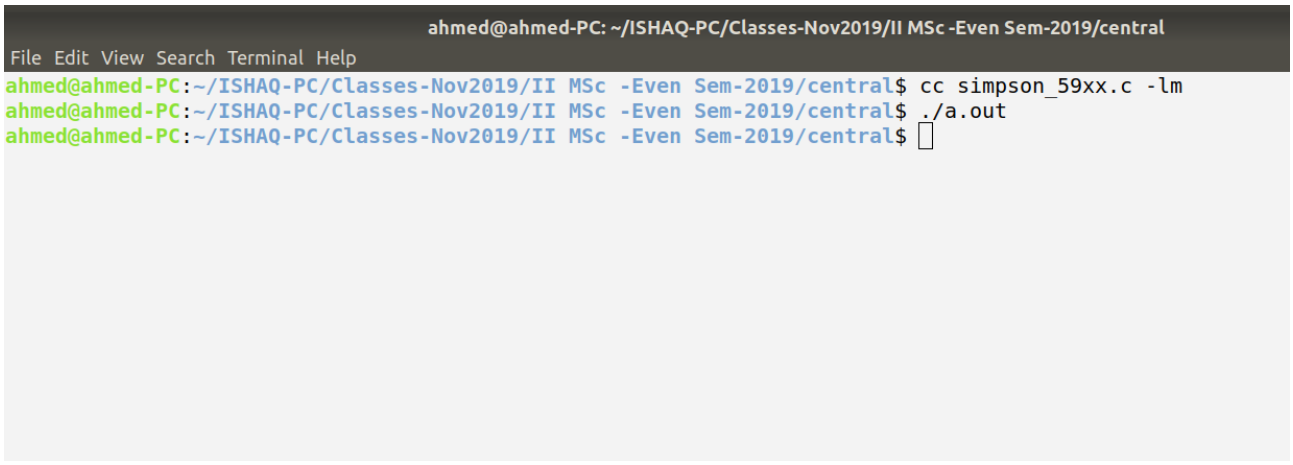

Figure 7.1: The command line arguments to compile and run the C code.

The command line arguments for plotting the orbit of the particle in the central force field using `gnuplot` is shown in Fig. (7.2).

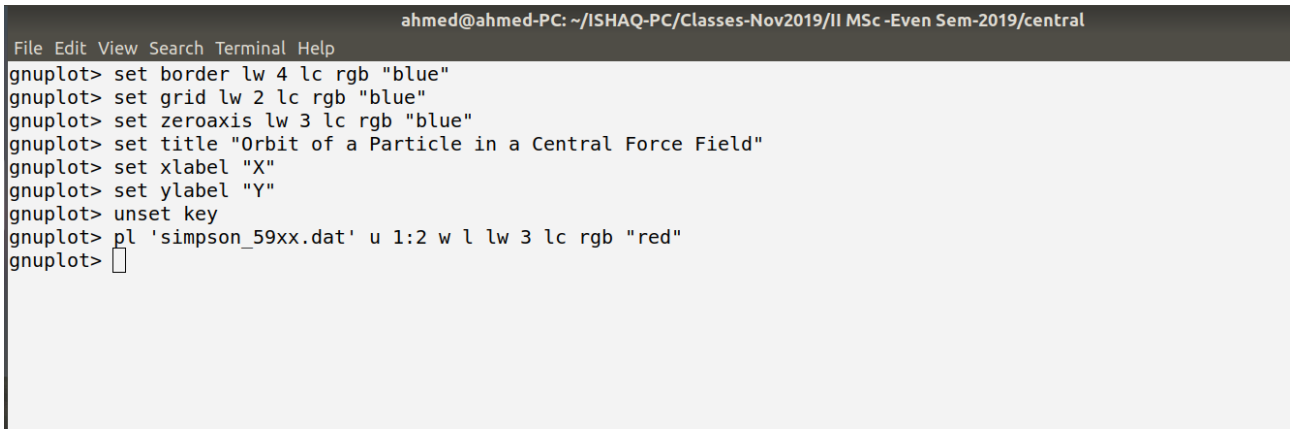

Figure 7.2: The command line arguments in the `gnuplot` window to obtain the orbit of the particle in the central potential.

The actual plot of the orbit of the particle in the central potential is shown in Fig. (7.3).

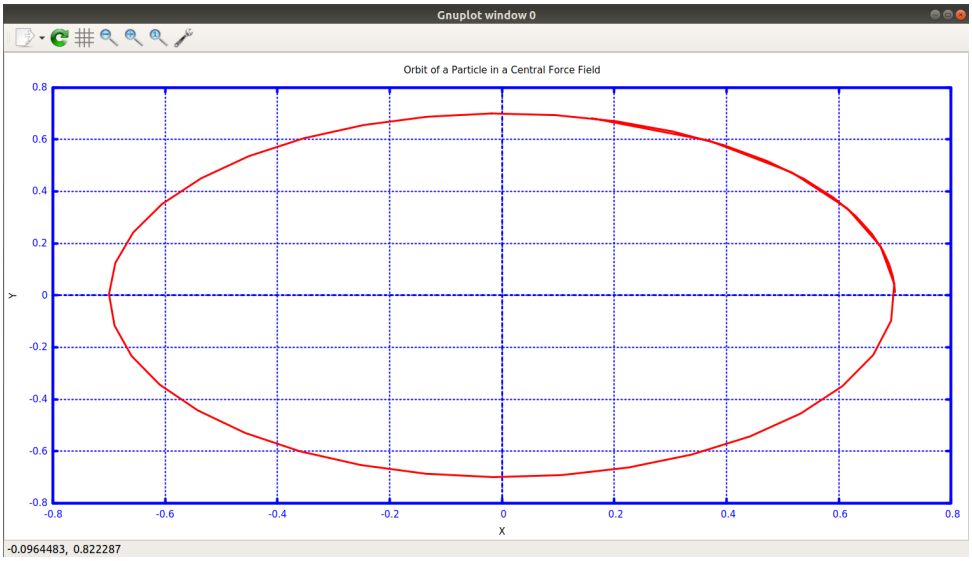

Figure 7.3: The orbit of the particle in the central potential.

7.5 Result

The motion of a partilce in a central force field is solved numerically using Simpson's $\frac{1}{3}$ method and its orbit is obtained using `gnuplot`.

Chapter 8

Electromagnetic Oscillations in a LCR Circuit: RK4 Method

8.1 Aim

1. To set up the mathematical equations for the LCR circuit and to normalize them using proper rescaling parameters.
2. To solve the normalized equations numerically using *RK4 Algorithm* and C programming language.
3. Hence to draw the time plots and phase portrait for the LCR circuit.

8.2 Series LCR Circuit

The series forced LCR circuit is a simple circuit containing an inductance, a capacitance and a resistance connected in series with a sinusoidal ac source. The circuit is shown in Fig. (8.1).

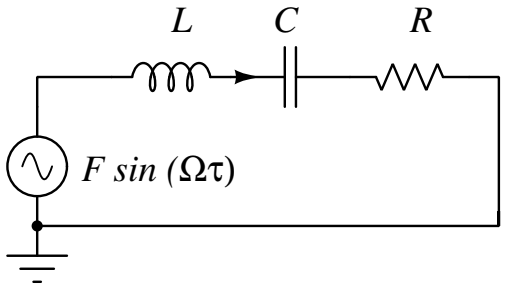

Figure 8.1: The forced series LCR circuit containing an inductance, a capacitance and a resistance connected in series with an ac sinusoidal power source.

Circuit Equations

Applying Kirchhoff's Voltage Law to the circuit, we have

$$L\frac{dI}{dt} + \frac{Q}{C} + RI = Fsin(\Omega t), \tag{8.1}$$

where L is the inductance, C is the capacitance, R is the resistance, F is the strength and Ω is the frequency of the external sinusoidal force. We know that the current is the time rate of change of charge, that is $I = \dfrac{dQ}{dt}$. Hence the Eqn. (8.1) becomes

$$L\frac{d^2Q}{dt^2} + \frac{Q}{C} + R\frac{dQ}{dt} = F sin(\Omega t). \tag{8.2}$$

This Eqn. (8.2) is a second order differential equation, which is difficult to solve. Hence we convert the equation to a set of two coupled first order differential equations, namely

$$\begin{aligned} \frac{dQ}{dt} &= I, \\ \frac{dI}{dt} &= -\left(\frac{R}{L}\right) I - \frac{Q}{LC} + F\sin(\Omega t). \end{aligned} \tag{8.3}$$

Rescaling and Normalized Forms

The Eqns. (8.3) contain terms that have inequal magnitude. For example the capacitance is of the order of micro-farads, while the inductance is of the order of milli-Henries and the resistance is of the order of Ohms. These lead to difficulties while solving them numerically. Hence to avoid this problem, the Eqns. (8.3) are converted into normalized form as

$$\begin{aligned} \frac{dx}{dt} &= y, \\ \frac{dy}{dt} &= -\alpha(y + x) + f\sin(\omega t). \end{aligned} \tag{8.4}$$

To obtain these equations we have used what are called as the rescaling factors, which convert the variables, namely charge and current into dimensionless form. The rescaling variables are $t = RC\tau$, $x = \dfrac{Q}{C}$, $y = IR$, $\alpha = \dfrac{R^2C}{L}$, $f = \alpha F$ and $\omega = \Omega RC$.

8.3 RK4 Algorithm

The Runge-Kutte IV order Method is a most commonly used method for solving ordinary differential equations. If

$$f(x, t) = 0.$$

is a first order differential equation in one independent variable, say t and one dependent variable x, then the RK4 algorithm is given as

$$\begin{aligned} m_1 &= f\left(t_i, x_i\right) \\ m_2 &= f\left(t_i + \frac{h}{2}, x_i + \frac{m_1 h}{2}\right) \\ m_3 &= f\left(t_i + \frac{h}{2}, x_i + \frac{m_2 h}{2}\right) \\ m_4 &= f\left(t_i + h, x_i + m_3 h\right) \\ x_{i+1} &= x_i + \left(\frac{m_1 + 2m_2 + 2m_3 + m_4}{6}\right) h \end{aligned}$$

Here the coefficients m_is are the slopes at various points in the interval h.

RK4 Method for Second Order Differential Equation

Any second order differential equation can be converted into two coupled first order differential equations. This algorithm can be extended to such a coupled set of two first order differential equations, say

$$\begin{aligned} f_1(x, y, t) &= 0, \\ f_2(x, y, t) &= 0. \end{aligned}$$

where the independent variable is t and the dependent variables are x and y. The extended RK4 algorithm is given as

$$\begin{aligned} k_1 &= f_1(t_i, x_i, y_i) \\ m_1 &= f_1(t_i, x_i, y_i) \\ \\ k_2 &= f\left(t_i + \frac{h}{2}, x_i + \frac{k_1 h}{2}, y_i + \frac{m_1 h}{2}\right) \\ m_2 &= f\left(t_i + \frac{h}{2}, x_i + \frac{k_1 h}{2}, y_i + \frac{m_1 h}{2}\right) \\ k_3 &= f\left(t_i + \frac{h}{2}, x_i + \frac{k_2 h}{2}, y_i + \frac{m_2 h}{2}\right) \\ m_3 &= f\left(t_i + \frac{h}{2}, x_i + \frac{k_2 h}{2}, y_i + \frac{m_2 h}{2}\right) \\ \\ k_4 &= f(t_i + h, x_i + k_3 h, y_i + m_3 h) \\ m_4 &= f(t_i + h, x_i + k_3 h, y_i + m_3 h) \\ \\ x_{i+1} &= x_i + \left(\frac{k_1 + 2k_2 + 2k_3 + k_4}{6}\right) h \\ y_{i+1} &= y_i + \left(\frac{m_1 + 2m_2 + 2m_3 + m_4}{6}\right) h \end{aligned}$$

8.4 C code to implement the RK4 Algorithm for LCR Circuit

The C code to implement the system of two coupled first order differential equations, Eqns. (8.4) describing the LCR circuit using the RK4 algorithm, is given below

```
1 // To study numerically the electromagnetic oscillations of a
2 // LCR circuit by RK4 method
3 // Roll No :
4 // Date : 14/02/2018
5
6 #include <stdio.h>
7 #include <math.h>
8
```

```
9  double alpha,omega,f;
10 double t,x,y,h;
11
12 double FUNC1(double, double, double);
13 double FUNC2(double, double, double);
14
15 void main()
16 {
17         long i,n;
18         double k1,k2,k3,k4;
19         double m1,m2,m3,m4;
20         double R,L,C,F,freq,pi;
21
22         FILE*fp;
23         fp = fopen("lcr_59xx.dat","w");
24         if(fp==NULL){printf("File open error!\n");return ;}
25
26         n = 20000;
27         h = 0.1;
28
29         pi = 4.0*atan(1.0);
30
31         R = 50;
32         L = 25E-3;
33         C = 0.2E-6;
34         freq = 1000;
35
36         alpha = (R*R*C)/L;
37         omega = (2.0*pi*freq*R*C);
38
39         printf("F = ");
40         scanf("%lf",&F);
41
42         f = alpha*F;
43
44         printf("\nalpha =%12.6lf\nomega =%12.6lf\nf =%12.6lf\n\n",
   alpha,omega,f);
45
46         x = 0.1;
47         y = 0.1;
48         t = 0.1;
49
50         for(i=1;i<=n;i++){
51                 k1 = FUNC1(t,x,y);
52                 m1 = FUNC2(t,x,y);
53
54                 k2 = FUNC1(t+0.5*h,x+0.5*k1*h, y+0.5*m1*h);
```

```
55                      m2 = FUNC2(t+0.5*h,x+0.5*k1*h, y+0.5*m1*h);
56
57                      k3 = FUNC1(t+0.5*h,x+0.5*k2*h,y+0.5*m2*h);
58                      m3 = FUNC2(t+0.5*h,x+0.5*k2*h,y+0.5*m2*h);
59
60                      k4 = FUNC1(t+h,x+k3*h,y+m3*h);
61                      m4 = FUNC2(t+h,x+k3*h,y+m3*h);
62
63                      x = x+(k1+2.0*k2+2.0*k3+k4)*h/6.0;
64                      y = y+(m1+2.0*m2+2.0*m3+m4)*h/6.0;
65
66                      t = t+h ;
67
68                      if(i>10000)
69                              fprintf(fp,"%12.6lf%12.6lf%12.6lf\n",t,x,y);
70              }
71              fclose(fp);
72      }
73      double FUNC1(double t, double x, double y)
74      {
75              double f1,g_x;
76
77              t = t*1.0;
78              f1= y;
79
80              return f1;
81      }
82      double FUNC2(double t, double x, double y)
83      {
84              double f2;
85              f2 = -alpha*(y+x)+f*sin(omega*t);
86
87              return f2;
88      }
```

The command line argument to compile and run the programme is given in Fig. (8.2). Here the step size is assumed as $h = 0.1$, the total number of iterations is assumed as $n = 20000$, the series resistance is taken as $R = 50\Omega$s, the inductance is taken as $L = 25$ milli-Henry, the capacitance is assumed as $C = 0.2$ micro-Farads and the frequency as $freq = 1000$ Hertz. The amplitude of the external sinusoidal force is assumed as $F = 5$ Volts.

```
ahmed@ahmed-PC: ~/ISHAQ-PC/Classes-Nov2019/II MSc -Even Sem-2019/lcr
File  Edit  View  Search  Terminal  Help
ahmed@ahmed-PC:~/ISHAQ-PC/Classes-Nov2019/II MSc -Even Sem-2019/lcr$ cc lcr_59xx.c -lm
ahmed@ahmed-PC:~/ISHAQ-PC/Classes-Nov2019/II MSc -Even Sem-2019/lcr$ ./a.out
F = 5

alpha =    0.020000
omega =    0.062832
f =    0.100000

ahmed@ahmed-PC:~/ISHAQ-PC/Classes-Nov2019/II MSc -Even Sem-2019/lcr$
```

Figure 8.2: The command line arguments to compile and run the C code.

8.5 Time Plots and the Phase Portrait

The evolution of a system with time can be inferred by drawing its *time plot* while the full dynamics can be pictured by its *phase portrait*. The time plot is the plot of a system variable as a function of time. It is drawn by taking time on the $x-$axis and the variable, say $x(t)$ on the $y-$axis. The command line arguments for plotting the time plot of the normalized current $x(t)$ is shown in Fig. (8.3).

```
ahmed@ahmed-PC: ~/ISHAQ-PC/Classes-Nov2019/II MSc -Even Sem-2019/lcr
File  Edit  View  Search  Terminal  Help
gnuplot> set title "Time Series Plot of x(t)"
gnuplot> set border lw 4 lc rgb "blue"
gnuplot> set grid lw 2 lc rgb "blue"
gnuplot> unset key
gnuplot> set xrange [1000:2000]
gnuplot> set xtics 1000,200
gnuplot> set yrange [-8:8]
gnuplot> set ytics -8,4
gnuplot> set xlabel "Time t"
gnuplot> set ylabel "x(t)"
gnuplot> pl 'lcr_59xx.dat' u 1:2 w l lw 2 lc rgb "red"
gnuplot>
```

Figure 8.3: The command line arguments in the `gnuplot` window to obtain the plot of the $x-$variable.

The actual time plot of the $x-$variable is shown in Fig. (8.4).

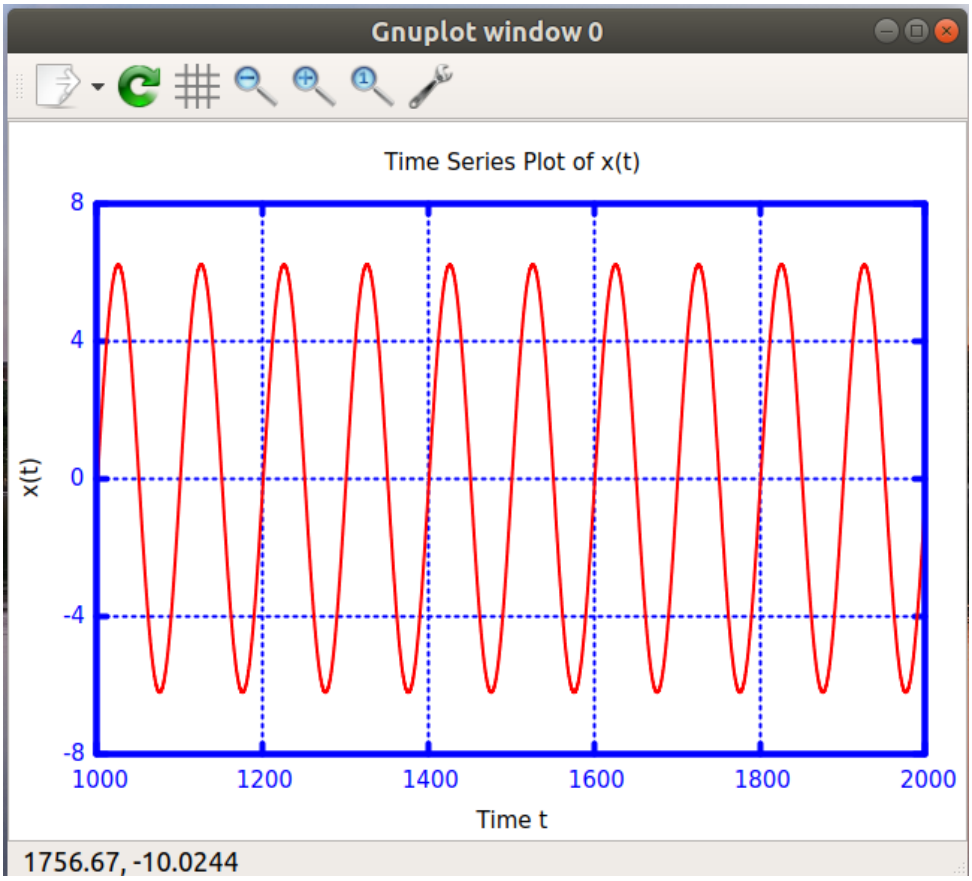

Figure 8.4: The plot of the $x-$variable as a function of time.

Similarly the command line arguments for plotting the time series of the normalized voltage $v(t)$ is shown in Fig. (8.5).

```
File  Edit  View  Search  Terminal  Tabs  Help
ahmed@ahmed-PC: ~/ISHAQ-PC/Classes-Nov2019/II MSc -Even Sem-2019/lcr
gnuplot> set border lw 4 lc rgb "blue"
gnuplot> set title "Time Plot for y(t)"
gnuplot> set grid lw 2 lc rgb "blue"
gnuplot> unset key
gnuplot> set xlabel "Time t"
gnuplot> set ylabel "y(t)"
gnuplot> set xrange [1500:2000]
gnuplot> set xtics 1500,100
gnuplot> set yrange [-4:4]
gnuplot> set ytics -4,2
gnuplot> pl 'lcr_59xx.dat' u 1:3 w l lw 3 lc rgb "red"
gnuplot>
```

Figure 8.5: The command line arguments in the `gnuplot` window to obtain the plot of the $y-$variable.

The actual time plot of the $y-$variable is shown in Fig. (8.6).

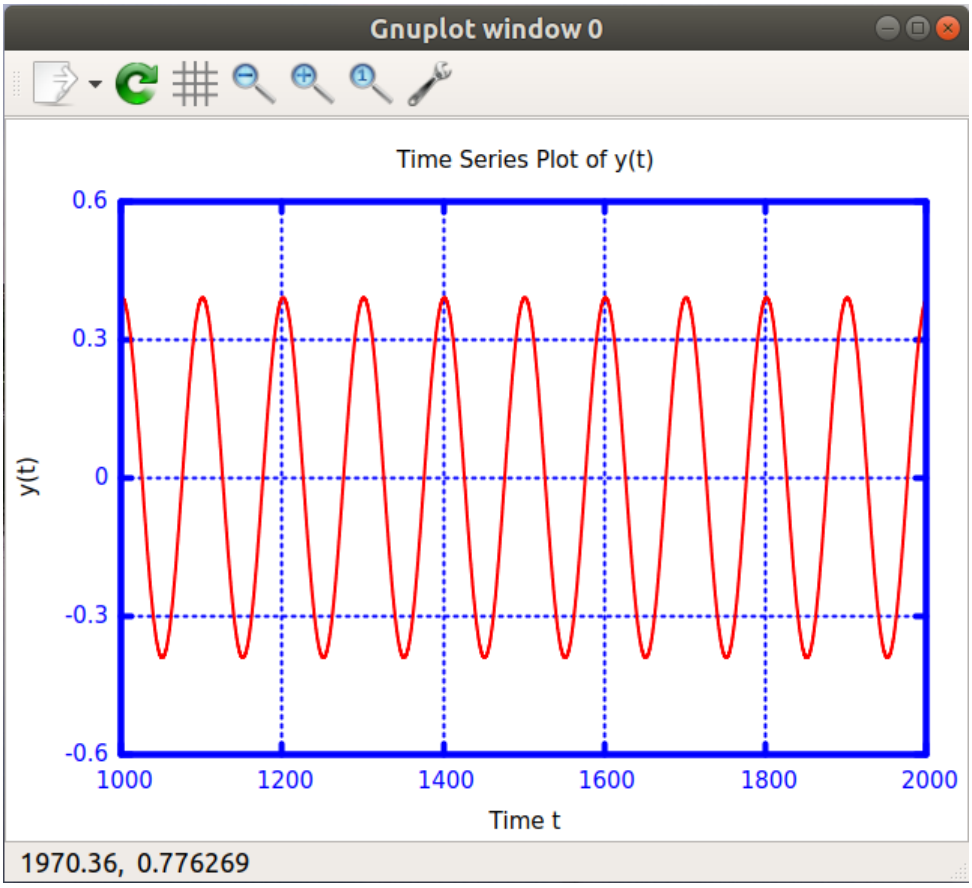

Figure 8.6: The plot of the $y-$variable as a function of time.

```
ahmed@ahmed-PC: ~/ISHAQ-PC/Classes-Nov2019/II MSc -Even Sem-2019/lcr
File  Edit  View  Search  Terminal  Help
gnuplot> set title "Phase Portrait in the (x-y) plane"
gnuplot> set border lw 4 lc rgb "blue"
gnuplot> set grid lw 2 lc rgb "blue"
gnuplot> set xlabel "x(t)"
gnuplot> set ylabel "y(t)"
gnuplot> unset key
gnuplot> set xrange [-8:8]
gnuplot> set xtics -8,4
gnuplot> set yrange [-0.6:0.6]
gnuplot> set ytics -0.6,0.3
gnuplot> set zeroaxis lw 3
gnuplot> pl 'lcr_59xx.dat' u 2:3 w l lw 2 lc rgb "red"
gnuplot>
```

Figure 8.7: The command line arguments in the `gnuplot` window to obtain the *phase plot* in the $(x-y)$ plane.

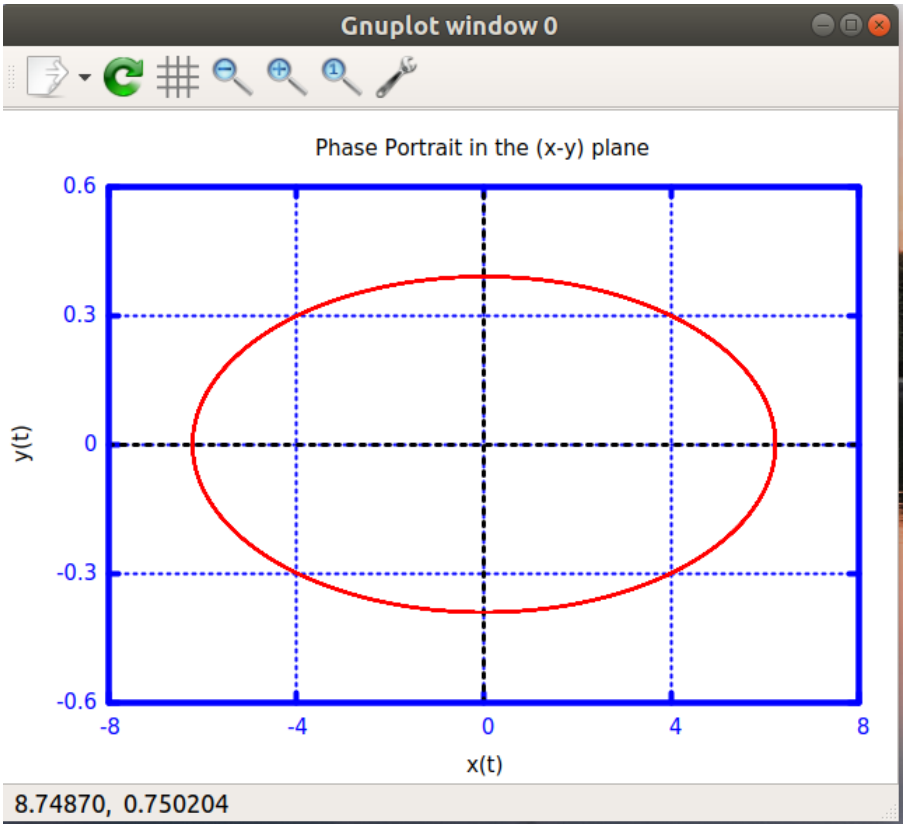

Figure 8.8: The plot of the phase portrait in the $(x - y)$ plane.

The *phase plane* is a two dimensional plane spanned by the position variable $x(t)$ and the velocity variable $(y(t) = \dot{x(t)})$. It describes clearly the dynamics of a physical system. In the phase space, each point represents a dynamical state of the system. As time passes, the state of the system changes. This causes the *phase point* to move in the phase space. The path followed by the phase point is called as the *phase trajectory*. When the energy of the system is a constant, then the phase trajectory will be a closed curve. The command line argument to obtain the phase trajectory of the LCR circuit constructed using the normalized charge and current variables is shown in Fig. (8.7), while the actual phase portrait is shown in Fig. (8.8).

8.6 Result

1. The equations for the forced series LCR circuit are obtained using Kirchhoff's Laws and are converted to normalized form using proper rescaling parameters.

2. The normalized equations are solved numerically using a C program for the RK4 algorithm and

3. The time plots for the normalized charge and current variables are obtained and the phase portrait in the $(x - y)$ plane is obtained.