CS201 - Introduction to Computational Physics
Assignment 8

Name - Omkar Damle
ID - 201401114

Billiard Table Simulation (Observation of ergodicity and butterfly effect)

Introduction

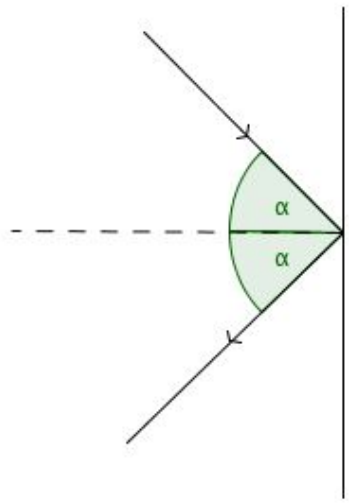
In this report we have analysed the motion of a billiard ball on a rectangular billiard table and a table with one of the sides a half circle. We have compared our observations for the two systems and shown that the latter system is **chaotic**. The law of reflection has been used while simulating the system. Also we have neglected the effect of friction of the table and assumed the collisions to be perfectly elastic.

The two important aspects of dynamical systems that were observed were :

1. Ergodicity
2. Butterfly effect

Why to analyse motion of a billiard ball ?

Simple processes like billiards can lead to chaos. So we make an attempt to analyse motion of a billiard ball.



The law of reflection: the angle of incidence equals the angle of reflection.

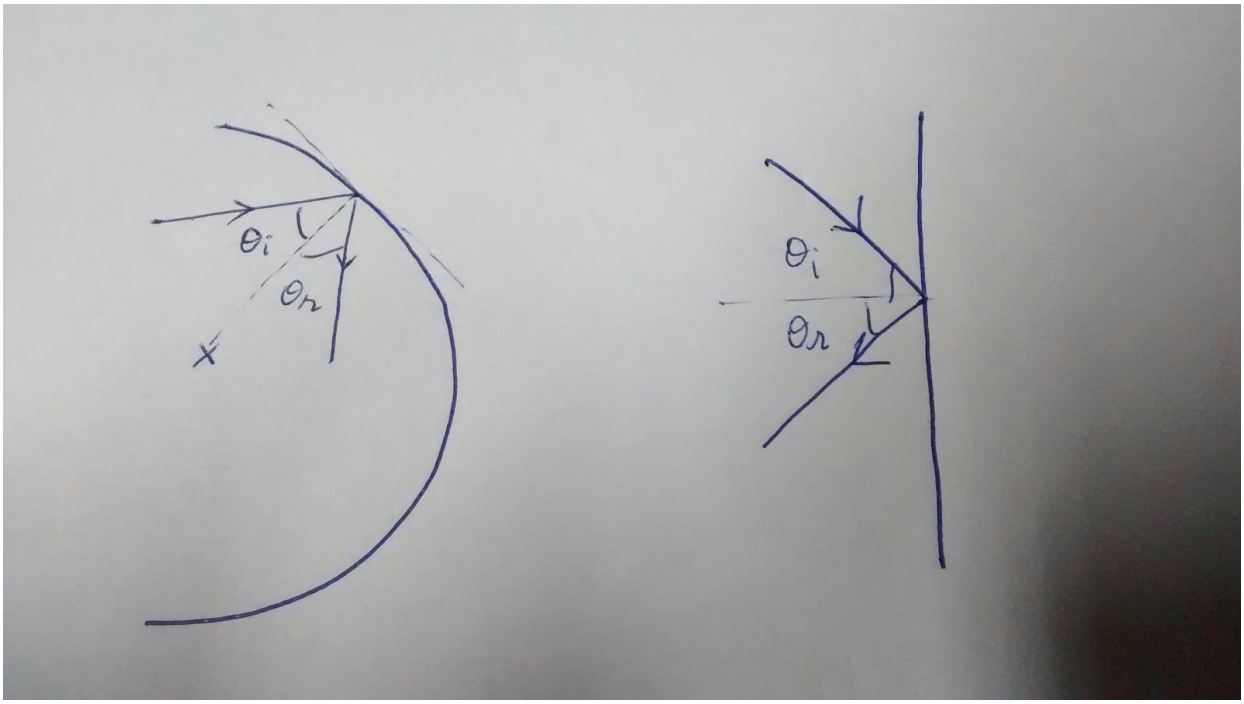
Effectively the law of reflection is fairly straightforward for reflection from a straight line. If the line is parallel to the X axis then only the y velocities will exchange and the x velocities will remain unchanged. Similarly if the line is parallel to the Y axis, only the x velocities will be exchange and the y velocities will remain the same.

The euler equations in the case of rectangular table are :

```
x(step + 1) = x(step) + vx(step) * dt;

y(step + 1) = y(step) + vy(step) * dt;

vx(step + 1) = vx(step);
```

vy(step + 1) = vy(step);

For a billiard table with one side half circular, the law of reflection is still valid. However the complications increase in this case.



The code used for simulating a collision on circular surface is as follows :

((hypot(x(step + 1) - (-0.5) , y(step + 1) - 0) >= radius) && x(step + 1) < -0.5) %left circular

m1 = (y(step + 1) - y(step)) / (x(step + 1) - x(step)) ; % slope of velocity before reflection

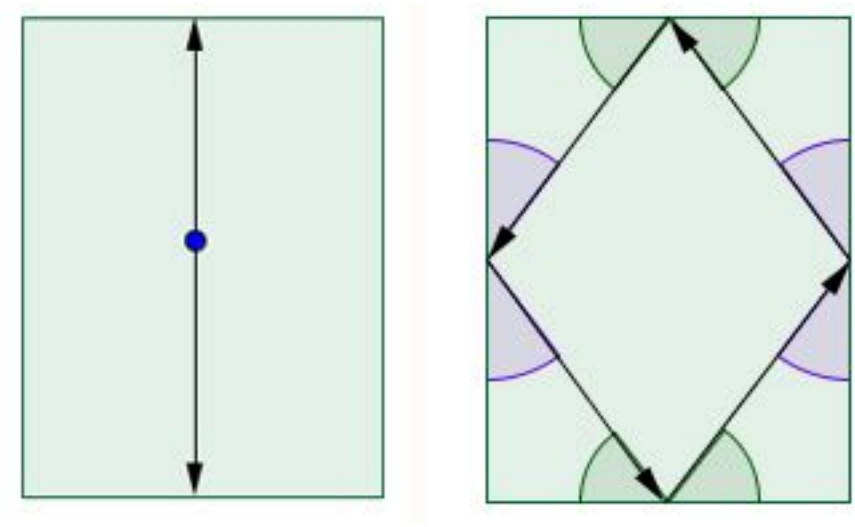
m2 = (y(step + 1) - 0) / (x(step + 1) - (-0.5)) ; % slope of radius

theta_pos = atan2(m1-m2 , 1 + m1*m2) % angle between initial velociy and radius

```
theta_neg = -theta_pos;

m_reflected = (tan(theta_neg) + m2 ) / (1-tan(theta_neg)*m2) % slope of velocity after reflection

theta_reflected = atan(m_reflected);

v = hypot(vx(step) , vy(step));


%assuming velocity after reflection is same as velocity before

%reflection


vx(step + 1) = v*cos(theta_reflected);

vy(step + 1) = v*sin(theta_reflected);


x(step + 1) =x(step);

y(step + 1) = y(step);

% pause(1000);

collision_flag = 1;
```

Periodic Orbits :

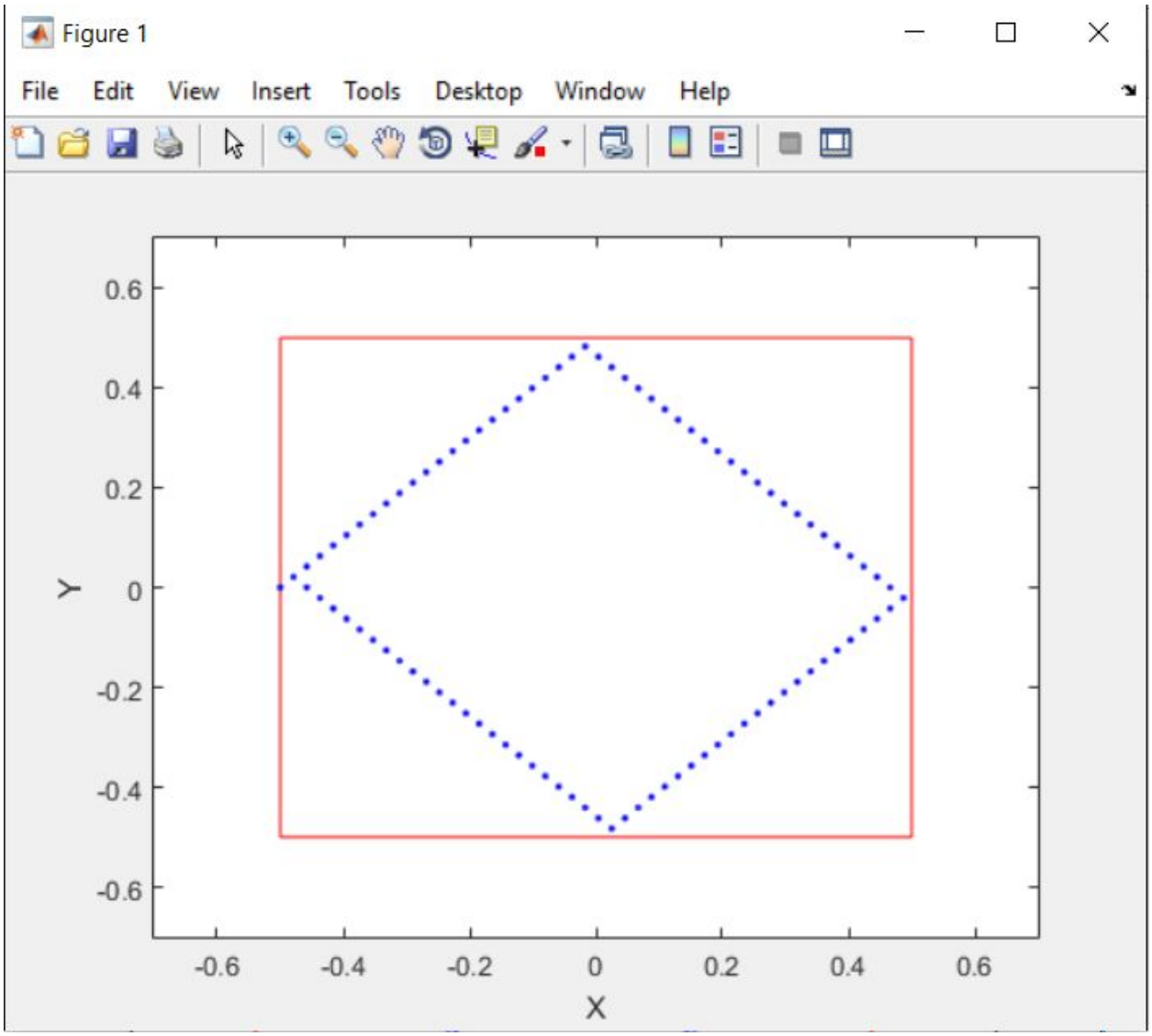
For certain initial conditions of position and velocity, we get periodic orbits.



The plot below shows periodic orbits example -

Initial Position(in metres) : x = -0.5 y = 0

Initial Velocity(in metres/second) : vx = 2 vy = 2



If you shoot a ball so it meets the wall at right angles, it will bounce between opposite points on the table forever. Similarly, you can make the ball travel between four points. These are examples of periodic trajectories.

But it turns out that this regular behaviour is very rare. It was proved that for the vast majority of initial directions the trajectory will be much wilder: **not only will it not retrace its steps, but it will eventually explore the whole of the table, getting arbitrarily close to every point on it. A typical trajectory will visit each part of the table in equal measure: if you take two regions of the table whose areas are equal,**

then the trajectory will spend an equal amount of time in both. This behaviour is a consequence of billiards being *ergodic*.

Ergodicity of billiards means that it's really hard to predict where a ball will be after a given amount of time as no pattern may emerge.

Observation of ergodicity in matlab program -

Initial Position(in metres) : x = -0.5 y = 0

Initial Velocity(in metres/second) : vx = 1 vy = 3

The region taken for investigation are two circles with radius 0.15 with centers at-

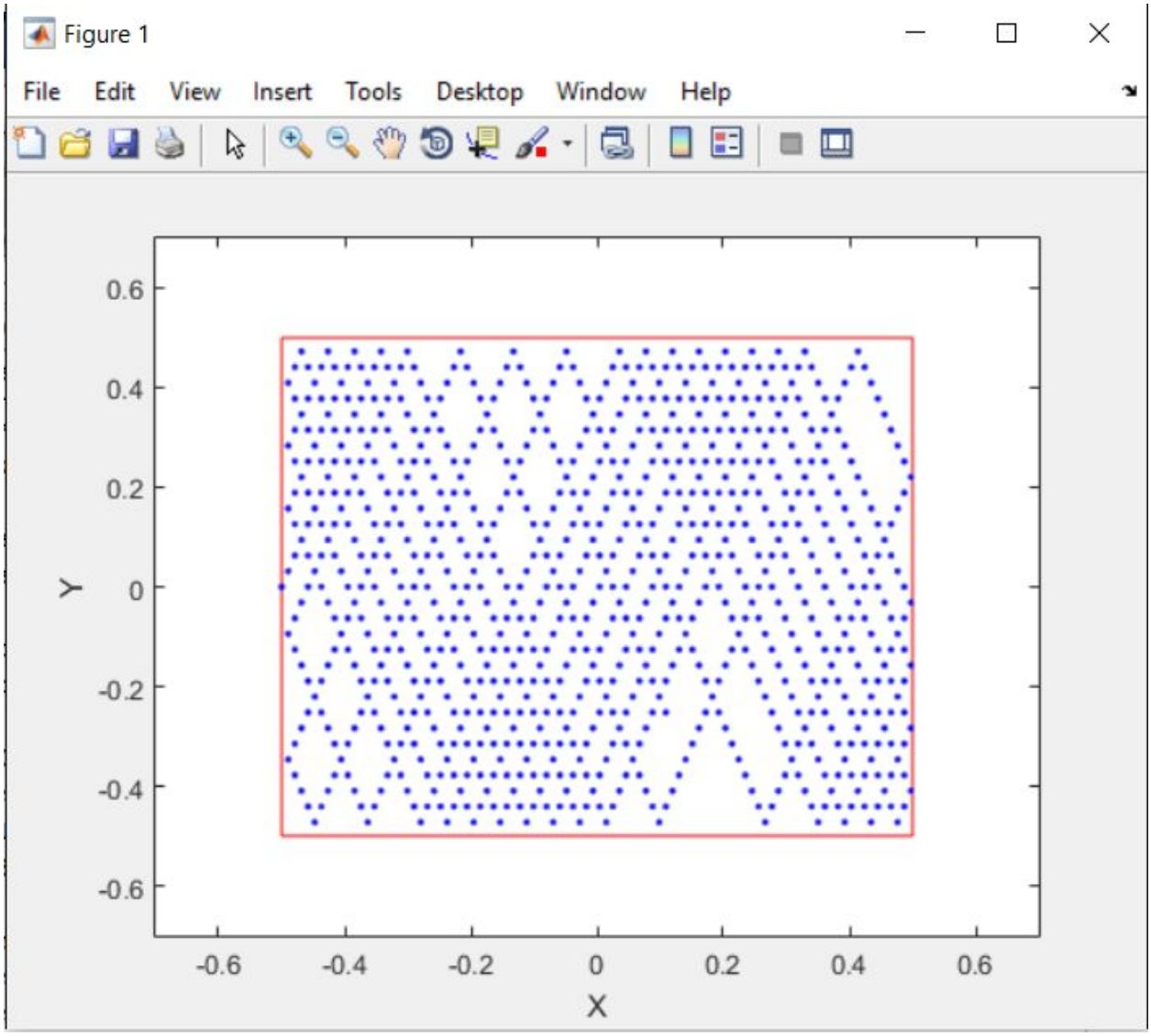
x = -0.25, y = -0.25 and x = 0.25, y = 0.25

The code used it is -

```
if(hypot( x(step) - (-0.25) , y(step) - (-0.25) ) <= 0.15)
    count1 = count1 + 1;
end

if(hypot(x(step) - (0.25) , y(step) - (0.25)) <= 0.15)
    count2 = count2 + 1;
end
```

When we did this we got the values as count1 = 87 and count2 = 93. This values are pretty close and they become equal if we run it for more time(i.e more simulations). Thus the ball spends almost equal amount of time in the two regions. Here the regions were taken to be symmetric. However we can take any arbitrary

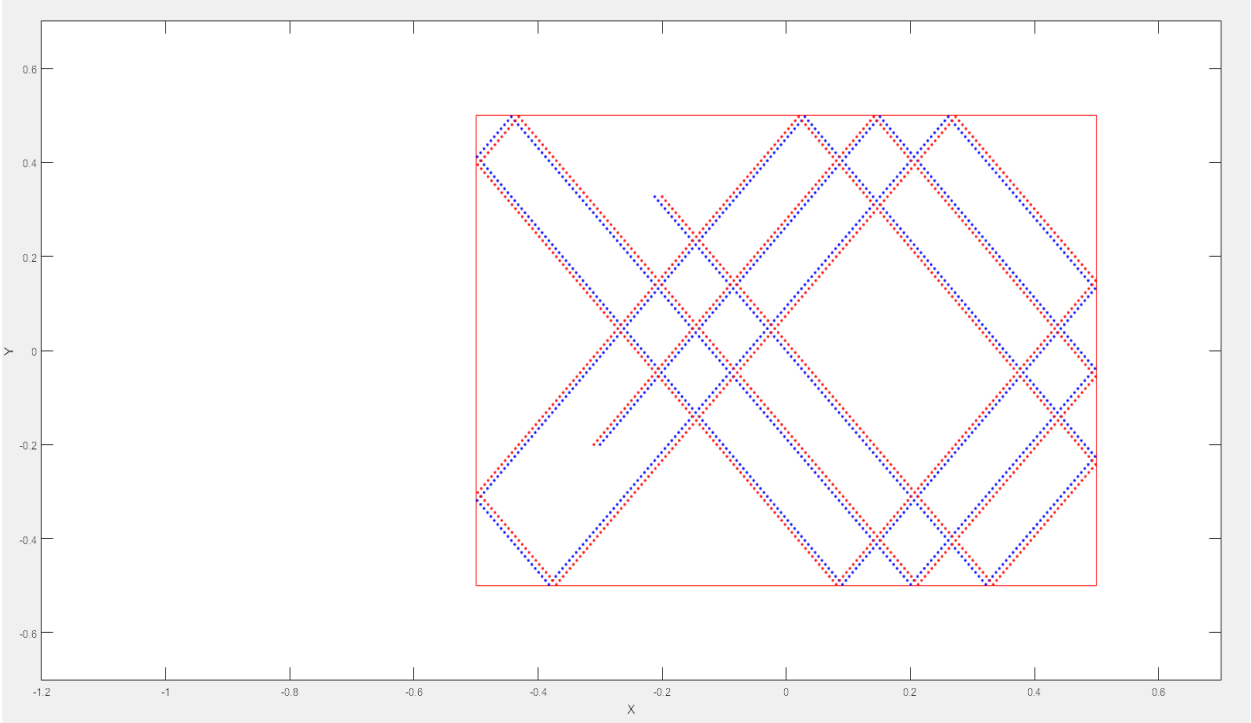
two regions of equal area and the time spent by ball in them could be almost the same. The difference reduces as we simulate for more and more time.



Observation of butterfly effect :

The *sensitive dependence on initial conditions* is popularly known as the butterfly effect, and is one of the hallmarks of chaos.

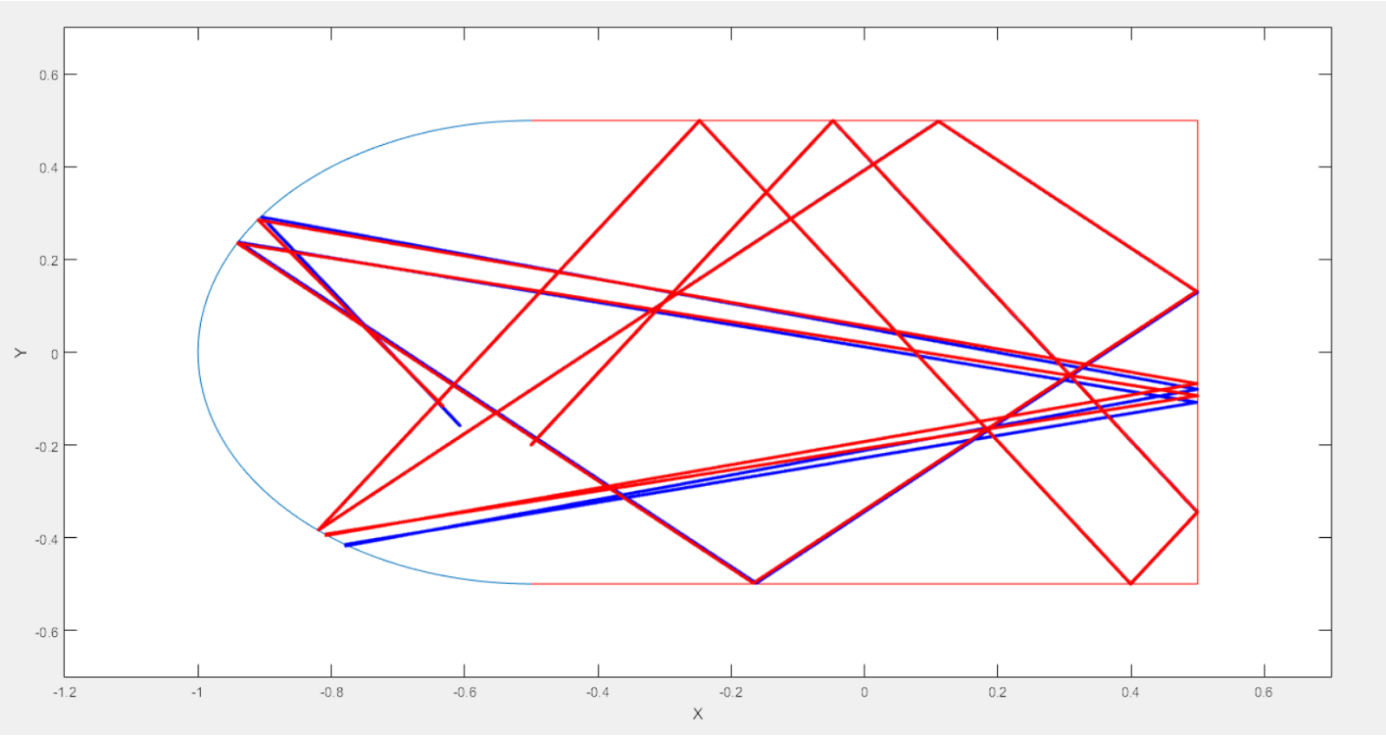
The plot for two bodies with a **difference of 1e-2** in x in initial position -



This plot shows that for a rectangle table, changing the initial conditions did not affect the motion. This is a non - chaotic system.

The plot for two bodies with a **difference of 1e-4** in x in initial position -

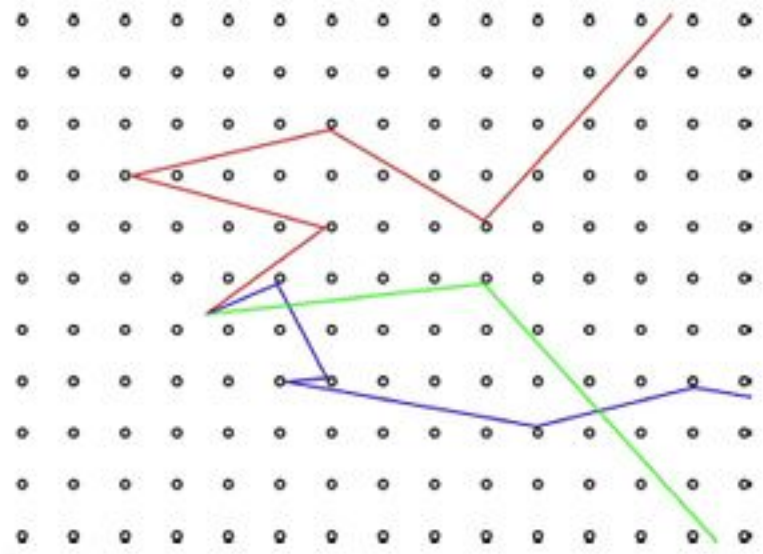
This is for table in which one side is a semi-circle.



After some time you can see the path changing though there is only a small change in initial conditions. This is due to the butterfly effect.

Applications :

1. **Molecules that make up a gas, bouncing around an ambient space, bumping into each other as they go**.
2. One natural phenomenon that quickly turns into a billiard problem when you simplify it is the **electrical conductivity of metals**. An electric current is a flow of particles called electrons. To make things simpler, suppose there's just a single electron passing through the metal, and imagine the metal as a lattice of molecules evenly arranged in the two-dimensional plane. If you think of the electron as a little ball, then it behaves just like a billiard ball bouncing around on a table that contains a lattice of obstacles.



Matlab Code for motion of two bodies on a rectangle billiard table - Billiard.m

```
%this code simulates a billiard ball in a table


% rectangle shaped table


close all;

clear all;


radius = 0.5;


array = [-0.5 0.1 0.5];

plot(array , 0.5*ones(length(array)) , 'r');          %top end

hold on
```

```
plot(array , -0.5*ones(length(array)), 'r');        %bottom end

hold on

plot(0.5*ones(length(array)) , array, 'r');         %right end

hold on

plot(-0.5*ones(length(array)) , array, 'r');        %left end



simulation_time = 5;

npoints = 1000;

dt = simulation_time/npoints;


x = zeros(npoints , 1);

y = zeros(npoints , 1);


vx = zeros(npoints , 1);

vy = zeros(npoints , 1);


x(1) = -0.3;

y(1) = -0.2;

vx(1) = 1.1;

vy(1) = 1.7;


collision_flag = 0;
```

```
for step = 1 : npoints-1


   hold on
   axis([-0.7 0.7 -0.7 0.7]);
   plot(x(step) , y(step) , '.b')
   pause(0.001)


   collision_flag =0;




   x(step + 1) = x(step) + vx(step) * dt;
   y(step + 1) = y(step) + vy(step) * dt;
   vx(step + 1) = vx(step);
   vy(step + 1) = vy(step);


  if(y(step + 1) >= 0.5 )                    %top end
      y(step + 1) = y(step);
      vy(step + 1) = -vy(step);
      vx(step + 1) = vx(step);
      collision_flag = 1;
  elseif (y(step + 1) <= -0.5 )               %bottom end
      y(step + 1) = y(step);
      vy(step + 1) = -vy(step);
```

```
    vx(step + 1) = vx(step);
    collision_flag = 1;
elseif (x(step + 1) >= 0.5)                    %right end
    x(step + 1) = x(step);
    vy(step + 1) = vy(step);
    vx(step + 1) = -vx(step);
    collision_flag = 1;
elseif (x(step + 1) <= -0.5)                   % left end
    x(step + 1) = x(step);
    vy(step + 1) = vy(step);
    vx(step + 1) = -vx(step);
    collision_flag = 1;
  end

if(collision_flag == 1) % on collision
    x(step + 1) = x(step) + vx(step + 1) * dt;
    y(step + 1) = y(step) + vy(step + 1) * dt;
end

if(collision_flag ==0) % no collision just normal motion
    x(step + 1) = x(step) + vx(step) * dt;
    y(step + 1) = y(step) + vy(step) * dt;
    vx(step + 1) = vx(step);
    vy(step + 1) = vy(step);
```

```
  end

end



x(1) = -0.31;   %Changing the initial condition by 1e-2 in x

y(1) = -0.2;

vx(1) = 1.1;

vy(1) = 1.7;


collision_flag = 0;


for step = 1 : npoints-1


   hold on

   axis([-1.2 0.7 -0.7 0.7]);

   plot(x(step) , y(step) , '.r')

   xlabel('X')

   ylabel('Y')

   %pause(0.001)


   collision_flag =0;
```

```
 x(step + 1) = x(step) + vx(step) * dt;
 y(step + 1) = y(step) + vy(step) * dt;
 vx(step + 1) = vx(step);
 vy(step + 1) = vy(step);

if(y(step + 1) >= 0.5 )                %top end
    y(step + 1) = y(step);
    vy(step + 1) = -vy(step);
    vx(step + 1) = vx(step);
    collision_flag = 1;
elseif (y(step + 1) <= -0.5 )            %bottom end
    y(step + 1) = y(step);
    vy(step + 1) = -vy(step);
    vx(step + 1) = vx(step);
    collision_flag = 1;
elseif (x(step + 1) >= 0.5)                    %right end
    x(step + 1) = x(step);
    vy(step + 1) = vy(step);
    vx(step + 1) = -vx(step);
    collision_flag = 1;
elseif (x(step + 1) <= -0.5)                    % left end
    x(step + 1) = x(step);
    vy(step + 1) = vy(step);
```

```
    vx(step + 1) = -vx(step);
    collision_flag = 1;
  end

  if(collision_flag == 1) % on collision
    x(step + 1) = x(step) + vx(step + 1) * dt;
    y(step + 1) = y(step) + vy(step + 1) * dt;
  end

  if(collision_flag ==0) % no collision just normal motion
    x(step + 1) = x(step) + vx(step) * dt;
    y(step + 1) = y(step) + vy(step) * dt;
    vx(step + 1) = vx(step);
    vy(step + 1) = vy(step);
  end

end
```

Matlab Code for motion of two bodies on a stadium billiard table (one end semi-circle) -

Billiard2.m

```
%this code simulates a billiard ball in a table
% stadium shaped table

close all;
clear all;
```

```
radius = 0.5;

array = [-0.5 0.1 0.5];
plot(array , 0.5*ones(length(array)) , 'r');          %top end
hold on
plot(array , -0.5*ones(length(array)), 'r');         %bottom end
hold on
plot(0.5*ones(length(array)) , array, 'r');          %right end
hold on
th = linspace(pi/2 , 3*pi/2 , 100);                  % left semi circle
x_circle = radius.*cos(th) + (-0.5);
y_circle = radius.*sin(th) + 0;
plot(x_circle , y_circle);

simulation_time = 7;
npoints = 5000;
dt = simulation_time/npoints;

x = zeros(npoints , 1);
y = zeros(npoints , 1);

vx = zeros(npoints , 1);
vy = zeros(npoints , 1);
```

```
x(1) = -0.5;

y(1) = -0.2;

vx(1) = 1.1;

vy(1) = 1.7;


collision_flag = 0;


for step = 1 : npoints-1


   hold on

   axis([-1.2 0.7 -0.7 0.7]);

   plot(x(step) , y(step) , '.b')

   xlabel('X')

   ylabel('Y')

   pause(0.001)


   collision_flag =0;


   x(step + 1) = x(step) + vx(step) * dt;

   y(step + 1) = y(step) + vy(step) * dt;

   vx(step + 1) = vx(step);

   vy(step + 1) = vy(step);
```

```
if(y(step + 1) >= 0.5 )                    %top end
    y(step + 1) = y(step);
    vy(step + 1) = -vy(step);
    vx(step + 1) = vx(step);
    collision_flag = 1;
elseif (y(step + 1) <= -0.5 )              %bottom end
    y(step + 1) = y(step);
    vy(step + 1) = -vy(step);
    vx(step + 1) = vx(step);
    collision_flag = 1;
elseif (x(step + 1) >= 0.5)                %right end
    x(step + 1) = x(step);
    vy(step + 1) = vy(step);
    vx(step + 1) = -vx(step);
    collision_flag = 1;
elseif ((hypot(x(step + 1) - (-0.5) , y(step + 1) - 0) >= radius) && x(step + 1) < -0.5)        %left circular
    m1 =  (y(step + 1) - y(step)) / (x(step + 1) - x(step)) ;   % slope of velocity before reflection
    m2 =  (y(step + 1) - 0) / (x(step + 1) - (-0.5)) ;    % slope of radius
    theta_pos = atan2(m1-m2 , 1 + m1*m2)  % angle between initial velociy and radius
    theta_neg = -theta_pos;
    m_reflected = (tan(theta_neg) + m2 ) / (1-tan(theta_neg)*m2) % slope of velocity after reflection
    theta_reflected = atan(m_reflected);
    v = hypot(vx(step) , vy(step));
```

```
    %assuming velocity after reflection is same as velocity before
    %reflection

    vx(step + 1) = v*cos(theta_reflected);
    vy(step + 1) = v*sin(theta_reflected);

    x(step + 1) =x(step);
    y(step + 1) = y(step);
   % pause(1000);
    collision_flag = 1
end

if(collision_flag == 1) % on collision
    x(step + 1) = x(step) + vx(step + 1) * dt;
    y(step + 1) = y(step) + vy(step + 1) * dt;
end

if(collision_flag ==0) % no collision just normal motion
    x(step + 1) = x(step) + vx(step) * dt;
    y(step + 1) = y(step) + vy(step) * dt;
    vx(step + 1) = vx(step);
    vy(step + 1) = vy(step);
end
```

```
end



x(1) = -0.5001;       % Changing the initial posiion by 1e-4 in x

y(1) = -0.2;

vx(1) = 1.1;

vy(1) = 1.7;


collision_flag = 0;


for step = 1 : npoints-1


   hold on

   axis([-1.2 0.7 -0.7 0.7]);

   plot(x(step) , y(step) , '.r')


   %pause(0.001)


   collision_flag =0;






   x(step + 1) = x(step) + vx(step) * dt;

   y(step + 1) = y(step) + vy(step) * dt;
```

```
 vx(step + 1) = vx(step);
 vy(step + 1) = vy(step);

if(y(step + 1) >= 0.5 )              %top end
    y(step + 1) = y(step);
    vy(step + 1) = -vy(step);
    vx(step + 1) = vx(step);
    collision_flag = 1;
elseif (y(step + 1) <= -0.5 )              %bottom end
    y(step + 1) = y(step);
    vy(step + 1) = -vy(step);
    vx(step + 1) = vx(step);
    collision_flag = 1;
elseif (x(step + 1) >= 0.5)                        %right end
    x(step + 1) = x(step);
    vy(step + 1) = vy(step);
    vx(step + 1) = -vx(step);
    collision_flag = 1;
elseif ((hypot(x(step + 1) - (-0.5) , y(step + 1) - 0) >= radius) && x(step + 1) < -0.5)        %left circular
    m1 =  (y(step + 1) - y(step)) / (x(step + 1) - x(step)) ;       % slope of velocity before reflection
    m2 =  (y(step + 1) - 0) / (x(step + 1) - (-0.5)) ;    % slope of radius
    theta_pos = atan2(m1-m2 , 1 + m1*m2)    % angle between initial velociy and radius
    theta_neg = -theta_pos;
    m_reflected = (tan(theta_neg) + m2 ) / (1-tan(theta_neg)*m2)   % slope of velocity after reflection
```

```
    theta_reflected = atan(m_reflected);
    v = hypot(vx(step) , vy(step));

    %assuming velocity after reflection is same as velocity before
    %reflection

    vx(step + 1) = v*cos(theta_reflected);
    vy(step + 1) = v*sin(theta_reflected);

    x(step + 1) =x(step);
    y(step + 1) = y(step);
   % pause(1000);
    collision_flag = 1
end

if(collision_flag == 1) % on collision
    x(step + 1) = x(step) + vx(step + 1) * dt;
    y(step + 1) = y(step) + vy(step + 1) * dt;
end

if(collision_flag ==0) % no collision just normal motion
    x(step + 1) = x(step) + vx(step) * dt;
    y(step + 1) = y(step) + vy(step) * dt;
    vx(step + 1) = vx(step);
```

```
        vy(step + 1) = vy(step);

    end


end
```