WXML Final Report: Gravitational Billiards

Jayadev Athreya
Diaaeldin Taha
N'Vida Yotcho
Kush Gupta
Stephanie Anderson

Fall 2016

Abstract

The billiards game has a quite simple setting. The cues - sticks- are used to launch the balls, which moves on a rectangular pool. But, what would playing billiards on an upward circular pool look like? This report exposed the theoretical and computational aspect of simulating such non-traditional vertical billiards.

Contents

I Introduction

The gravitational billiards is an upward version of the well-known billiards game, in which the earth gravitational field - gravity g - influences the balls motion. Assuming that there is no friction, and that **the system is totally elastic**, we simulated the motion of one billiards ball - reduced to its center of mass- in such gravitational billiards, which pool is also reduced to its boundary function F . The simulations were made with the following four boundary functions: ***parabola, circle, paraboloid and sphere***.

II Theoretical Background

Once the ball is launched in the gravitational billiards, it is in free-fall following a projectile motion. But, **if the ball collides with the boundary, its trajectory has to change**. Subsequently, each trajectory between two points of collision could be considered as an isolated projectile motion with its own initial values. As an example, a trajectory between two points of collisions P_0 and P_1 is the ball leaving P_0 to collide with the boundary at P_1, where the ball would engage in another projectile motion with P_1 as starting point. Since the experimental settings allow no loss of energy, the collision ball-boundary would result in the ball bouncing back with the same speed of before the collision. In another word, the ball velocity-before-collision is simply reflected after collision. We can take advantage of that aspect of our system and build our simulation based on finding the different points, where the ball collides with the boundary.

III Computational Implementation

As mentioned earlier, F is the boundary function, then let's H represents the projectile motion of the ball
We then used the following lemma to code all simulations.

Step 1 The ball starts at P_0 with velocity V_0

Step 2 Find the next point P_1 where $\mathsf{F} = \mathsf{H}$

Step 3 Draw the projectile motion between P_0 and P_1

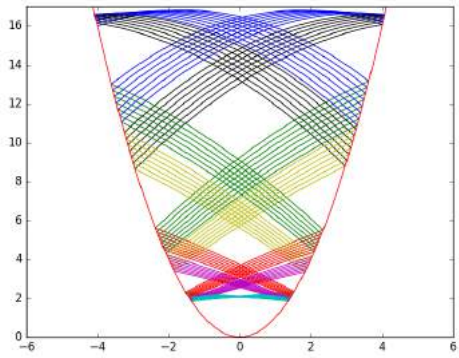
Step 4 Reflect the Velocity V_1 of the ball at P_1

Step 5 Repeat steps 1 to 5 with $\mathsf{P}_0 = \mathsf{P}_1$

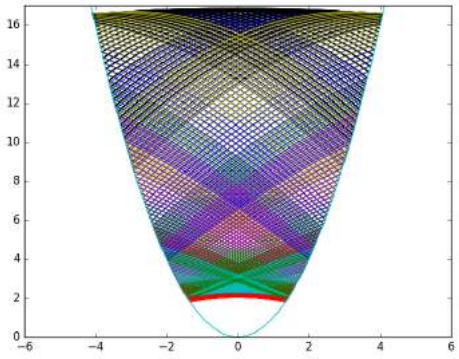
IV Results

F is paraboloid Click here to see the simulation

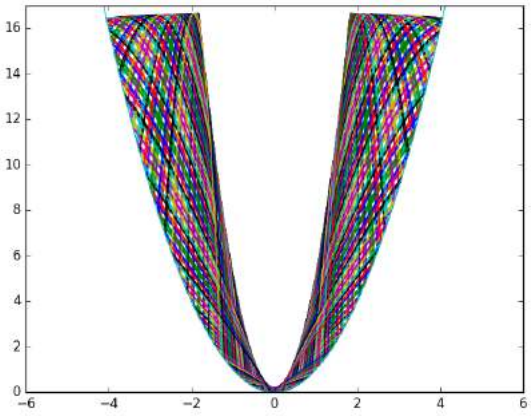
F is sphere Click here to see the simulation



(a) 100 collisions

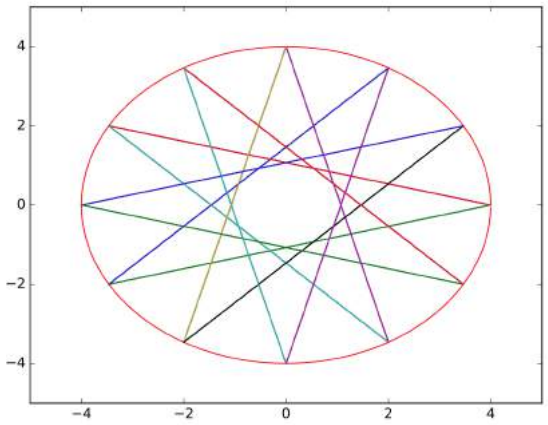


(b) At 500 collisions, the ball motion is bounded by an upper and a lower parabolas in addition to the parabolic boundary

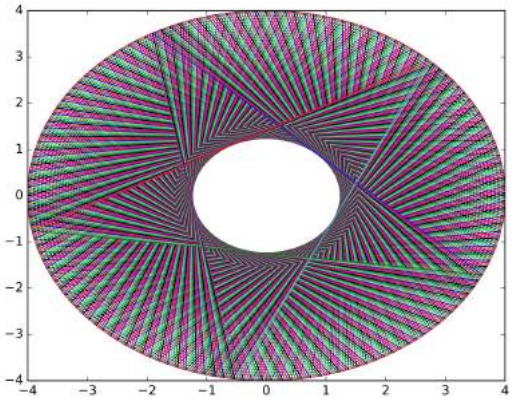


(c) Special case of 500 collisions with a higher value of g

Figure 1: **F is a parabola**



(a) 600 collisions - the ball follows a closed path



(b) 600 collisions - on a very small interval, the ball strikes the boundary at least once

Figure 2: F **is a circle** - no gravity

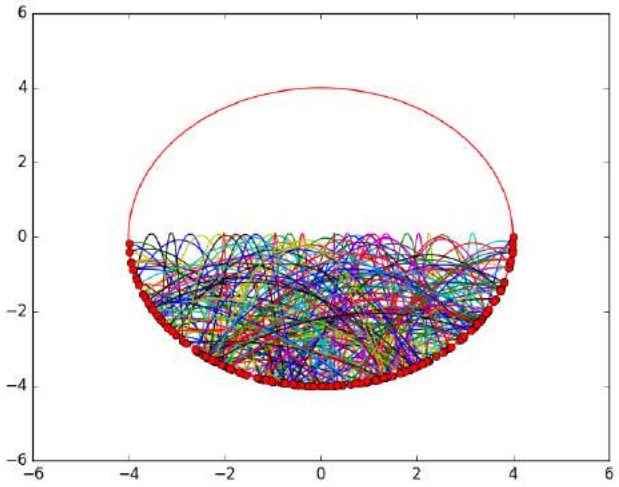


Figure 3: F **is a circle** - with gravity

V Example of the sage code for F is a paraboloid

```
# inspired from Stephanie Anderson's paraboloid.m
# created by N'Vida A. Yotcho
# Fall 2016

import math
import numpy as np
import matplotlib.pyplot as plt


######################## SPHERE- GRAV BILLIARDS  ########################

def X_func(Vx0, t, x0):
    return RR(Vx0*t + x0)

def Y_func(Vy0, t, y0):
    return RR(Vy0*t + y0)

def Z_func(g, t, Vz0, z0):
    return RR(-0.5*g*(t**2) +( Vz0*t)+z0)


def Vz_func(t,Vz0, g):
    return RR(-g* t + Vz0)


############ NEXT P1 ###############

def find_t(r, g,x0, y0, z0, Vx0, Vy0, Vz0):
    real_t = (Vz0-2*x0*Vx0-2*y0*Vy0)/(0.5*g +Vy0**2 +Vx0**2)

    #print real_t
    P = np.array([real_t, X_func(Vx0,real_t,x0),Y_func(Vy0,real_t,y0), Z_func(g,real_t,Vz0,z
    return P


############ NORMAL VECTOR  #########
def normal(P):
return np.array([ 2*P[0] , 2*P[1], -1])

def norm_Of(V):
return np.sqrt( V[0]**2 + V[1]**2 + V[2]**2)


########## REFLECTION OF V #############
```

```
#Vprime = V - 2Vn with Vn : proj of V onto n
def reflect_V( normal, V) :
    Vprime = V - 2*( np.dot(normal, V)/ (norm_Of(normal)**2))*normal
    return Vprime

################## PLOTTING ##############




def motion(g,x1,x0,y0,z0,Vx0,Vy0,Vz0, colors):

    #print "... PLOTTING TRAJECTORY ..."
    u = var("u")

    z(t) = -0.5 * g * t**2 + Vz0*t + z0
    x(t) = Vx0*t + x0
    y(t) = Vy0*t + y0

    return parametric_plot3d( (x(u), y(u), z(u)), (u,0,x1),color=colors, opacity=0.40)


############### BILLIARDS ##############

def paraboloid_billiards( radius, x0, z0, y0, Vx0, Vy0,Vz0, g, thresh):

    x,y, z= var("x, y, z")
    #graph = plot3d(x**2+y**2, (x,-radius, radius),(y,-radius, radius), color="yellow", opac
    graph = point3d((x0,y0,z0), color="blue", opacity=0.1)

    col = 5
    colors = ["red","yellow", "green","magenta", "cyan"]

    for dec in range(thresh):
        if col ==0:
            col=3
        else :
            col = col-1

#Calculations
        PV= find_t(radius, g,x0, y0, z0, Vx0, Vy0, Vz0)
        V= np.array([ Vx0, Vy0, PV[4]])
        Vreflected = reflect_V( normal(np.array([PV[1], PV[2], PV[3]])), V)
        graph= graph + motion(g,PV[0],x0,y0,z0,Vx0,Vy0,Vz0, colors[col])

#updates
```

```
        x0 = PV[1]
        y0 = PV[2]
        z0 = PV[3]
        Vy0 = Vreflected[1]
        Vx0 = Vreflected[0]
        Vz0 = Vreflected[2]

        graph = graph + point3d((x0,y0,z0), color=colors[col], opacity=0.1)

    show(graph)

######### MAIN ##########


def check_initial_condition(a, b, c):
    p=[]
    if a > 0 and b + c < 0:
        return 0
    elif abs(a) > 2*(b + c):
        return 0
    return 1


def main(x0, y0, Vx0, Vy0, Vz0, thresh):

    radius= 4 #not important
    g = 9.8

    #case 1
    z0 = sqrt( x0**2 + y0**2)

    print " P = ( {0}, {1}, {2})  ; V = < {3}, {4}, {5} > ".format(x0,y0,z0,Vx0,Vy0,Vz0)


    if check_initial_condition(Vz0, x0*Vx0, y0*Vy0) == 0:
        paraboloid_billiards( radius, x0, z0,y0, Vx0, Vy0, Vz0, g, thresh)
        print " "
    else :
        print " IVP not in Domain "

    #case 2
    # z0 = x0**2 + y0**2
#
#     print " P = ( {0}, {1}, {2})  ; V = < {3}, {4}, {5} > ".format(x0,y0,z0,Vx0,Vy0,Vz0)
#
#
```

```
#     if check_initial_condition(Vz0, x0*Vx0, y0*Vy0) == 0:
#         paraboloid_billiards( radius, x0, z0,y0, Vx0, Vy0, Vz0, g, thresh)
#         print " "
#     else :
#         print " IVP not in Domain "

############################## OBSERVATION ##############################

collisions = 500


#Flowers - like
#main(1, 1, -10, 10, 20, collisions)
#main(1, 1, 10, -10, 20, collisions)
#
#Picks - like
#main( 2, 3, -5, -5, 10 , collisions)

#closed path - like
#main( -3, -4, 0.2, -0.3, 2, collisions)
#main( 0, 2, -0.2, -0.3, 2, collisions)


# only on the z plane
main( 5,5,-10,-5,25, collisions)
```

VI Future directions

- Develop a code that simulates trajectories for any boundary.
- Create a game.
- Simulate trajectories for other type of boundaries.