ÉCOLE D INGÉNIEUR DENIS DIDEROT

1^{st} *year common core*

Circular billiard/semi-circular Simulator – report.

BENLAFQIH Ahmed

BERKANI Oumaima

EIDD Engineering school

Numerical Methods.

Tutored by: Mr. BELMOND Renaud

Written on: 30-12-2023

Summary:

1- Introduction:

Billiards, a game deeply rooted in history and spanning several centuries, has not only withstood the test of time but has also dynamically adapted to the ever-evolving landscape of technology. The shift from traditional billiard tables to virtual simulations marks a significant paradigm shift in the narrative of this timeless game. This composition delves into the captivating journey of billiards pool simulation, intricately tracing its origins, charting its growth trajectory, and exploring the profound impact of technology on its evolutionary path.

The genesis of billiards can be pinpointed to the 15th century in Europe, where it emerged as a favored pastime among the nobility. The game's early incarnation unfolded on grassy lawns before evolving onto wooden tables adorned with green cloth. The strategic nuances and skillful maneuvers required in billiards entranced players, eventually leading to the establishment of dedicated billiard halls during the 18th century.

In a contemporary world dominated by digital media and various forms of digital entertainment, the logical progression for billiards was its digitization. This involved the intricate process of converting and adapting the physical and mathematical equations inherent in the game into executable code—a task easier said than done. Billiard simulations have become a subject of intense interest for mathematicians and physicists alike. Throughout the project, we encountered numerous theses and papers that delved into complex issues related to simulations, exploring hypotheses based on various billiard forms, ranging from classical geometric shapes like rectangles to more intricate ones like ellipses, triangles, and even stadium-shaped billiards (reminiscent of pool). These studies often focused on the chaotic and seemingly unpredictable nature of pool balls inside the billiard,

emphasizing how minimal changes in the initial states of the system could significantly influence its trajectory, leading to increased chaos and unpredictability over time.

Our project narrowed its focus to a fundamental geometric shape—the circle. We subsequently explored variations, including a half-circle and a truncated circle. The challenges we encountered were multifaceted, revolving around decisions on how to best represent the movement of the ball within our virtual pool. Debates surfaced on whether to work with vectors, how to define time within the simulation, and numerous other intricacies. Another significant hurdle was defining Alpha—the reflection angle of the ball after collision—and determining its point of origin. In the pages that follow, we will expound upon our project, providing insights into our code, methodologies, and a comprehensive exploration of the multifaceted challenges we navigated throughout the process.

2- *Mathematical equations:*

For this simulation, we will need a few very basic equations:

- Equation of a circle.
- Equation of a straight line, decomposed on two cartesian axes, Y and X.

The equation of a circle is written as following:

$$x^2 + y^2 = r^2$$

For a ball to stay withing the bounds of our billiard, it should follow the previous equation, with x and y being its coordinates in the cartesian system and "r" being the radius of our pool table.

It is given that the equation of a line is written as following:

$$y = ax + b$$

We can decompose this equation further more to obtain two related equation, one following y and the other following x:

$$x = x_0 + \cos(\alpha) \, . \, t$$

$$y = y_0 + \sin(\alpha) \, . \, t$$

With y_0 and x_0 being the coordinates of the ball's initial position *(or the position it last collided),* α being the angle the ball is coming at and t being the time.

This means that y and x must follow and respect all the previous equation.

For it to be accurate, the reflection angle must follow the laws of Descartes:

$$\alpha' = 2\theta - \alpha + \pi$$

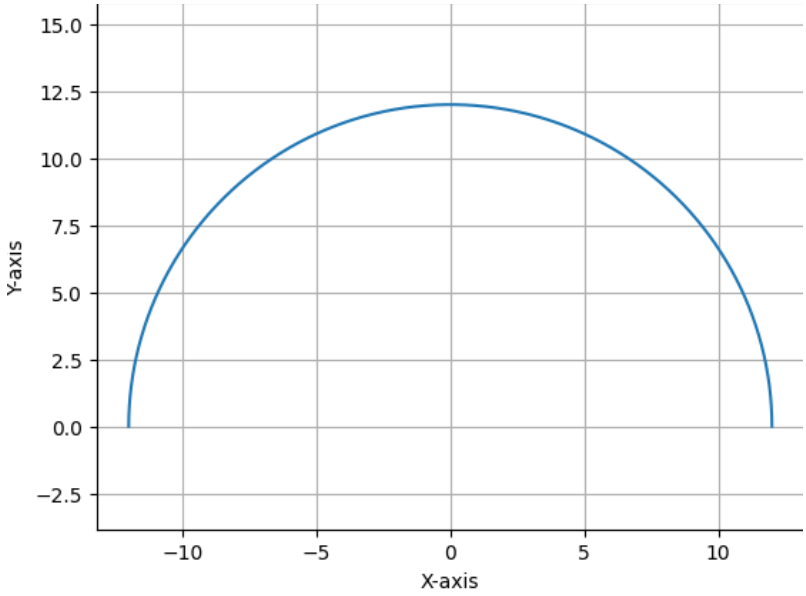
With θ being the normal angle to the ball's coordinates.

Now on to the semi-circular and truncated circle, for these shapes the same previous rules apply once again, just instead of the of:

$$y = y_0 + \sin(\alpha) \, . \, t$$

We will have:

$$y = 0 \;\; \text{or} \;\; y = y' \; \textit{where} \; (|y| < r)$$



truncated circle where y=0.

Which ultimately leads to:

$$x = x_0 - y_0 . \tan(\alpha)$$

When it comes the reflection angle, the same formula applied before in the circular billiard is used once again, the only difference is that if |y|<r, the normal angle θ is always equal to: $\theta = \frac{\pi}{2}$.

3- *Choice of Numerical methods:*

- *For the circular billiard:*

Due the nature of the problem and the simplicity of the equations, the best and most efficient method we could chose was the Euler method that says:

$$y_{n+1} = y_n + \delta t . f(t_n, y_n)$$

That is by analogy:

$$y = y_0 + \sin(\alpha) . t$$

And thus, so on and so forth as much as required by the user.

- *For the semi-circular billiard:*

This one took us a significant amount of time trying to figure out which zero-finding method to pick from. The difficulty with the truncated circles is that there are two different bounds to work on, the circular and straight one. For the circular bound the calculations are straightforward, but if "y" is ever below the value we picked for our "y'", the code should imperatively set y to y' and look for the new "x" in function of the modified y.

We initially came up with an equation that made use of diameters and radiuses, we've set the distance between the previous coordinates and the freshly calculated one X' (but where out of bound) as D, the distance between the previous point Xi and the point X, the one whom y equals y'(the one we were looking for its x coordinate) as R, and finally the distance between the points X' and X as R', so that we get the formula:

$$S = R + R'$$

After some calculation and transformations, we defined a new function f(x), where x is the coordinate, we're looking for and set its value to zero:

$$f(x) = 0$$

And using the dichotomy method between the x coordinate of X' and Xi we could determine our x with a precision of 10^{-6}.

After some tinkering the code worked just fine but later down our revisions we figured out we could do it just using the linear equations used earlier, this modification not only reduced the number of lines on our code and improved its readability, it also improved its complexity, making it run faster and more efficiently.

We were also convinced at a certain point of time we could determine the reflections by just mirroring whatever trajectory happens in the lower bottom ($y<0$) in the upper one regarding the symmetry of the circle by flipping the sign of y. We ended up soon abandoning that idea.

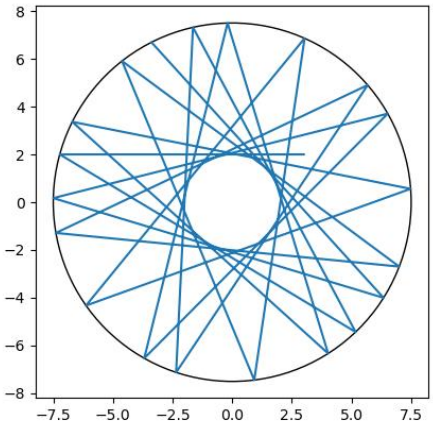
4- Code structure:

The code begins as usual declaring all the headers and essential libraries required to its functioning, then the first thing we started off with was the definition of a new structure "ball", that will contain the coordinates the ball and the angle it forms with cartesian plan at each collision throughout the whole simulation. This structure will help us keep track and organize everything within it.

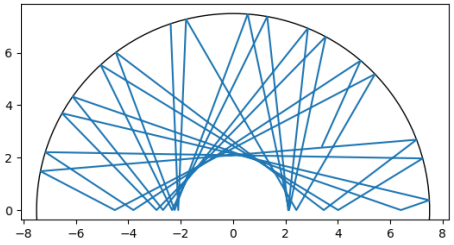
What follows next is the "simulation" function, it calculates the coordinates and reflection angles after a collision, this function basically solves the two linear equations expressed in the *mathematical equations* section, and that by determining t, the time the ball actually collides with the billiard, and it does that by implementing the x and y equation in the circle equation. To simplify the code and for general readability purposes we decided on introducing a and b variable that simplified the form of our equation.

After calculating t, the code determines the new coordinates, calculates the reflection angle using the normal angle, alpha and Descartes's laws of reflection to finally update them in our structure.

And finally, in the "main" section, we give our users the ability to input the initial coordinates of the ball as well as the initial angle alpha. For uniformity reasons with our python code, we decided to set the codes to 30 iterations for each simulation. We also fixed the Billard radius to 15dm. The code then starts a loop set to those iterations and executes the "simulation" function for each updated position. Simultaneously, a .txt file is created and opened in such a way that stores the coordinate data at each collision, that same exact .txt file is utilized by our python code, executed separately to finally visualize the ball's trajectory. The same structure and principles were used for both the circular, and half circular billiards.



Trajectory in a circular billiard:



Trajectory in semi-circular billiard:

5- Conclusion:

The code presented offers a sophisticated simulation capturing the intricate dynamics of a point particle navigating within the boundaries of a circular billiard table. Its seamless integration of mathematical concepts, including trigonometry and classical mechanics, results in an authentic representation of the physical motion involved. The systematic organization of the code, complemented by the utilization of a ball structure, not only enhances clarity but also fosters modularity, making it an insightful model for understanding and extending.

Thank you for reading.