Dynamics of triangular billiards

Serena Miari

25 August 2025

1 Main design and implementation

The main design of the program is centered around the representation of different border types. The particle's behaviour and collision geometry varies depending on whether the border is opened (R1 < R2), straight (R1 = R2) or closed (R1 > R2); therefore, dynamic polymorphism is implemented. The abstract class Border is created and specific border types implemented as derived classes.

2 Libraries

The following libraries are used:

- SFML: for the visual simulation of the particle's trajectory;
- doctest: for testing;

ROOT was used for the generation of the histograms displayed in the results section; however, it is unnecessary for the compilation of the program.

3 How to run the program

The project is composed of the following source files:

- main.cpp: handles user interaction.
- triangularbilliards.cpp and triangularbilliards.hpp: handle the computation of the trajectory of the particle and its final coordinates; the particle's collision against the borders, if any, and its initial and final positions are stored.
- triangularbilliards.test.cpp: contains the tests for triangularbilliards.cpp
- statistics.cpp and statistics.hpp: handle the computation of mean, standard deviation, skewness, and kurtosis.

- statistics.test.cpp: contains the tests for the statistics functions.
- simulation.cpp and simulation.hpp: generate the visualization of the particle's trajectory in a triangular billiard.

To run the program using CMake, after compiling:

cmake -S . -B build -G"Ninja Multi-Config"
cmake –build build –config Debug
cmake –build build –config Release

The program may be executed by typing:

build/Debug/progetto

Once the program is running, the following functionalities are available:

- add borders [b R1 R2 L]
- calculate final conditions [f Y0 Theta0]
- run simulation of the trajectory [v (Y0) (Theta0)]
- generate data [g N $Y0_mean$ $Y0_err$ $Theta0_mean$ $Theta0_err$]
- erase all values [e]
- print data [o]
- quit [q]

Before executing any of the options, it is necessary to set the borders parameters. The values should be positive or equal to zero; should an invalid input be inserted, an exception will be thrown. Valid parameters are, for example: (20, 50, 100), (30, 30, 200), (50, 10, 100).

Command 'f' and 'v' refer to single particles. Valid parameters to run command f are, for example: (5., 0.785), (0., 0.,), (-2., .145). A Y value exceeding the parameter R1 in magnitude and a theta value greater than PI/2 or smaller than -PI/2 will be rejected.

If command 'f' is called prior to command 'v', the values inserted will be used for the visualization, regardless of whether (v) or (v Y0 Theta0) is called. However, it is possible to call command 'v' without determining the final conditions first. In this case, (v Y0 Theta 0) should be called.

The 'g' command determines the parameters of the distributions of Y_f and $Theta_f$. It may be called multiple times; however, the final values calculated will be stored until 'e' (erase all values) is called, or the program is terminated.

Valid parameters are, for example (1000000, 5., 0.1, 0.785, 0.01). Should a negative standard deviation be inserted, its absolute value will be considered.

Additionally, the number of accepted and rejected values is displayed. This distinction is necessary since generated random inputs may fall outside the valid ranges.

The command o prints the list of Yf and Thetaf values to results.txt. These values were used for histogram generation in the results section.

4 Results

The distributions of Y_f and $Theta_f$ are hereby reported, subdivided by border type. The initial parameters of the particle were the following:

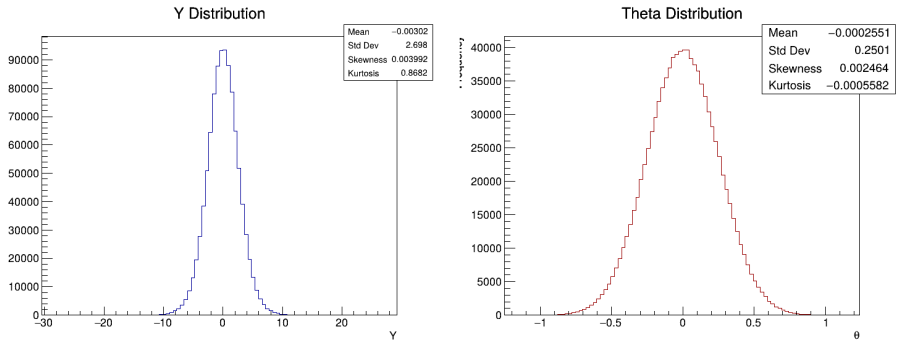
- $Y_0 = 0.$;
- $Theta_0 = 0.25$;
- $Y_0, err = 0.$;
- $Theta_0, err = 0.25$;

N = 1E6 initial conditions were generated for all cases.

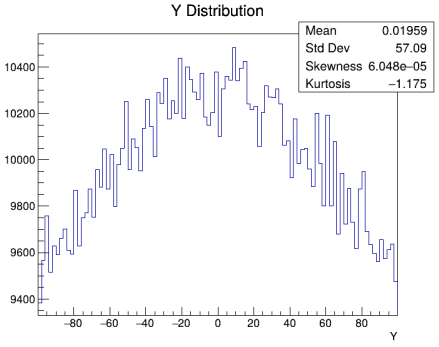
4.1 Opened borders

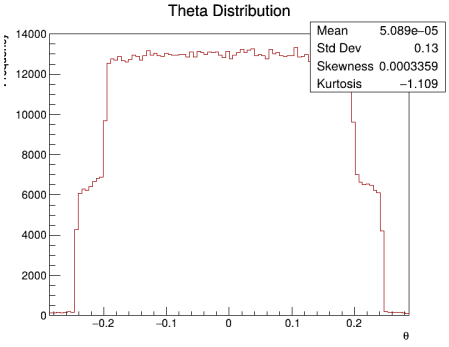
Setting R1 = 10 and R2 = 100:

For L = 100:

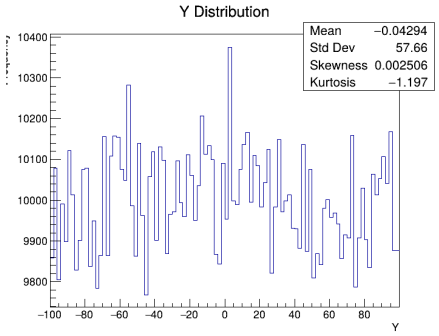


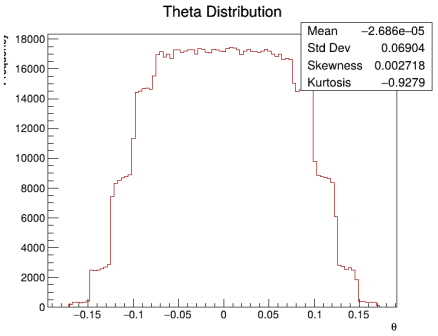
For L = 400:





For L = 800:



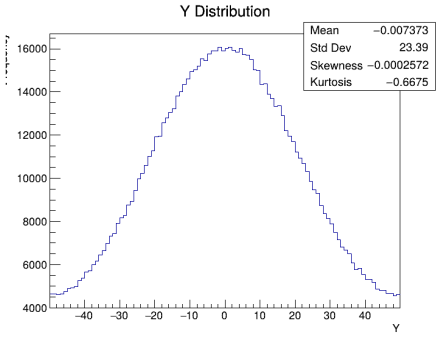


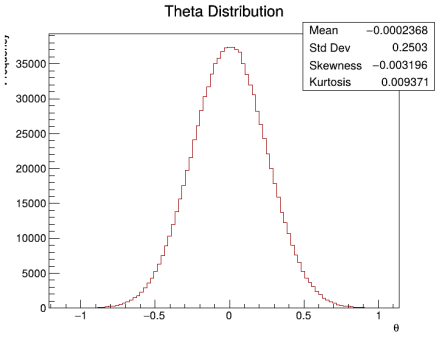
In the case of opened borders, as L increases, the distribution of Y_f converges towards a uniform distribution, whereas the $Theta_f$ forms more and more 'steps'.

4.2 Straight borders

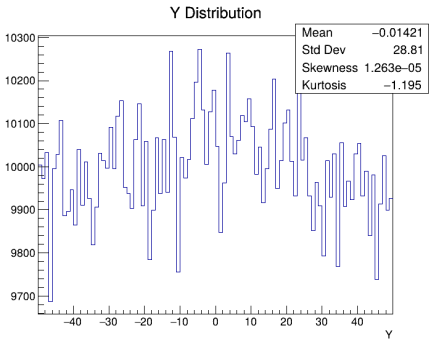
Setting R1 = 50 and R2 = 50.

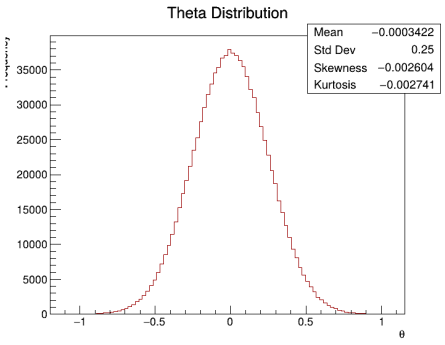
For L = 100:



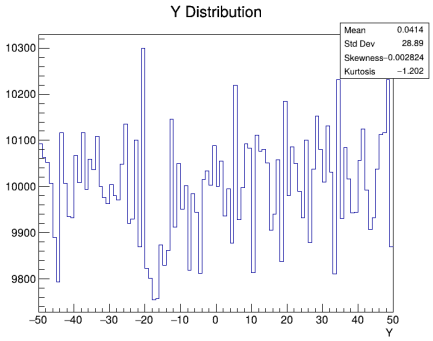


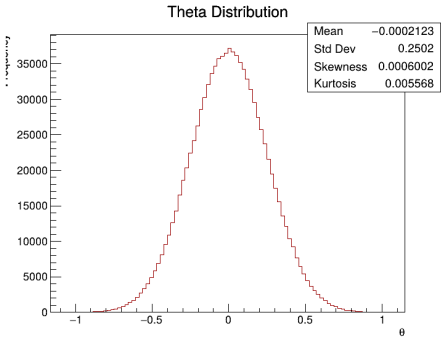
For L = 250:





For L = 500:





In the case of straight borders, as L increases, the distribution of Y_f once again converges towards a uniform distribution, whereas $Theta_f$ maintains a normal distribution.

5 Testing strategies

All major functions (e.g., computeNextCollision(), computeFinalPosition(), etc.) were tested against all possible border and trajectory combinations. Edge cases, such as a theta = PI/2 or initial values causing the particle to move backwards, were tested where possible to ensure proper handling. Due to its stochastic nature, the function runMultipleSimulations() was tested through consistency checks, e.g., by calling the size() method on the finalY and finalTheta vectors and comparing it with the "accepted" variable.

6 AI

Partial use of AI was made for the following:

- Writing the code to create distribution histograms in ROOT.
- Writing SFML code to display the trajectory of the particle.

- Calculating skewness and kurtosis using the Kahan summation algorithm. Standard formulae were initially implemented; however, precision errors emerged when calculating the values for large samples ($N > 30$).
- General proofreading to catch common syntax errors.