Implementation of a Two-Dimensional Random Walk Simulation, Parameter Estimation and Error Calculation

Gbenga Agunbiade[1]

[1]Department of Physics and Astronomy, University of Kansas, Lawrence, KS 66045

gsagunbiade@ku.edu

Abstract. *The project creates a theoretical Rayleigh distribution plot for comparison with the data. It generates an array f1 of evenly spaced values between 0 and the maximum final displacement and calculates the corresponding Rayleigh distribution values f2 using the estimated parameter. The theoretical distribution is plotted as a dashed line using matplotlib.pyplot.plot() with a label that includes the estimated parameter value and confidence interval. The data distribution is plotted as a histogram using matplotlib.pyplot.hist() with customization options for color, fill, hatch, and linewidth, and a label. The plot is displayed using matplotlib.pyplot.show().*

1. Introduction

Python is usually used to develop software and websites and analyze data, task automation, and data visualization. Since it's relatively easy to learn, Python has been adopted by many non-programmers such as accountants and scientists, for a variety of everyday tasks, like organizing finances. There are several ways programmers demonstrate how to use Python to simulate and visualize the behavior of a two-dimensional (2D) random walk. A 2D random walk is a stochastic process where an object or a particle starts at a given point in a 2D space and moves randomly in a sequence of steps. It can be extended and modified to study different types of random walks or to explore different aspects of random processes. This project is designed using Python code to generate two-dimensional (2D) random walks by employing the Monte Carlo method. The code is an implementation of a 2D random walk.

2. Implementation, Results and Discussion

The "Random"class has the following methods: "init(self, seed = 4445)"is the initialization method for the "Random"class, which takes an optional seed value for initializing the random number generator. The method initializes the seed, as well as several internal variables used in the random number generation process. "int64(self)"returns a random 64-bit integer using a combination of bitwise operations and arithmetic operations. "rand(self)"returns a random floating-point number between 0 and 1 (exclusive) using the "int64"method and scaling the result. "BoxMuller(self, Lstep)"implements the Box-Muller method for generating two Gaussian numbers with a given step size "Lstep". It uses the "rand"method for generating random numbers. "Categorical(self, p1=0.45, p2=0.45, p3=0.45, p4=0.45, p5=0.45, p6=0.45, p7=0.45, p8=0.45)"generates a random integer (0 to 7) according to a categorical distribution with given probabilities for each category. It uses the "rand"method for generating random numbers. "Exponential(self, beta=1.)"generates a random double (greater than 0) according to an exponential distribution with a given beta parameter. It uses the "rand"method for generating random numbers and the "math.log"function from the math module for computing the natural logarithm. "Bernoulli(self, p=0.6)"generates a random integer (0 or 1) according to a Bernoulli distribution with a given probability parameter "p". It uses the "rand"method for generating a random number. "TruncExp(self, beta, bottom, top)"generates a random double according to a truncated exponential distribution with a given beta parameter and lower and upper bounds "bottom"and "top". It uses the "Exponential"method for generating random numbers and repeats the process until the generated number falls within the specified range.

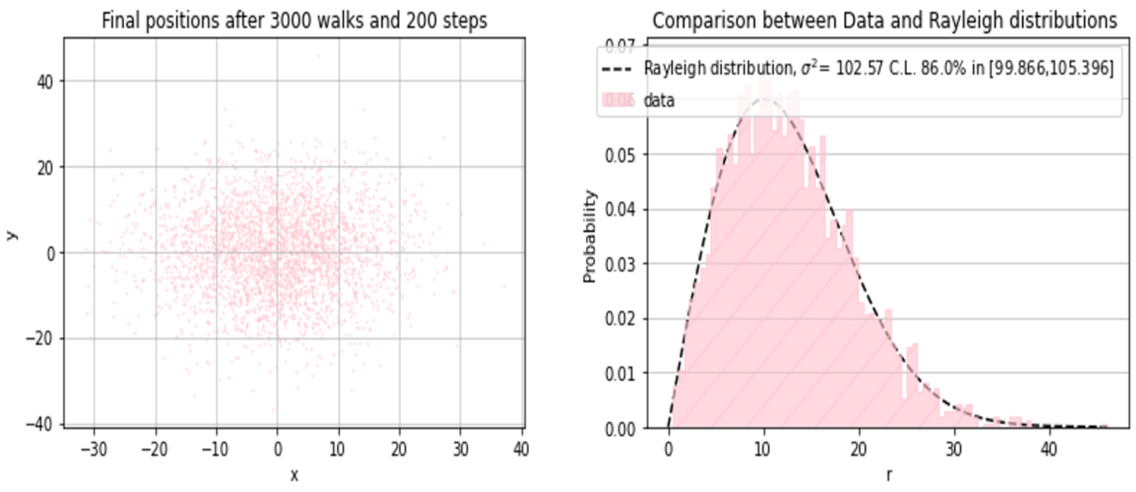

Figura 1. A scatter plot of the final positions of the random walks and a histogram of the squared Euclidean distances

A random walk is a mathematical model for a path that consists of a sequence of random steps. In this case, the steps are taken in two dimensions, so the path is represented as a sequence of points in the x-y plane. This Python code is for simulating and analyzing a 2D random walk. The random walk is performed by generating random steps in the x and y directions and updating the position accordingly. The code uses the Box-Muller method to generate random numbers from a normal distribution, and the final positions after the random walk are plotted and analyzed. Here is a step-by-step explanation of the code: Importing libraries: The following libraries are imported in the code: “Sys” provides access to command-line arguments and other system-specific functionalities. “Numpy” provides support for arrays, matrices, and mathematical functions. “Matplotlib.pyplot” provides functions for creating plots and visualizations. “Pylab” provides a procedural interface for generating plots. “Scipy.stats” provides statistical functions, including the chi-square distribution. “Math” provides mathematical functions. “Random” is a custom module that contains a Random class for generating random numbers using various methods. Parsing command-line arguments check if any command-line arguments are provided and use them to set values for various parameters, such as the seed for the random number generator, the number of walks (Nwalk), the number of steps in each walk (Nstep), the step length (Lstep), and the confidence level (CL). In generating random walks, the code uses a nested loop to perform the random walk. The outer loop iterates over the number of walks (Nwalk), and the inner loop generates random steps in the x and y directions using the Box-Muller method. The position in the x and y directions is updated based on the random steps, and the final position after the random walk is stored in arrays (xfinal, yfinal, finald, and finald2) for further analysis. The code calculates the estimated parameter par using the formula 1./(2.*Nwalk) * addt, where addt is the sum of the squares of the final positions after the random walk (finald2). This parameter is used to estimate the confidence interval later. The code calculates the errors for the estimated parameter by calculating the average of the squares of the final positions (avr2), and then using the chi-square distribution to calculate the lower and upper bounds of the confidence interval (CLmin and CLmax, respectively). The code prints the results of the parameter estimation, including the estimated parameter (par) and the confidence interval (CLmin and CLmax). The code creates a scatter plot of the final positions of the random walks using matplotlib.pyplot.scatter() function.The code creates a histogram of the squared Euclidean distances (finald2) using matplotlib.pyplot.hist() function, and overlays it with a Rayleigh distribution curve calculated based on the estimated parameter (par) and the confidence interval (CLmin and CLmax). The histogram is plotted with pink color, and the Rayleigh distribution curve is plotted with a dashed black line as shown in figure 1. The legend and titles are added to the plot using matplotlib.pyplot.legend(), matplotlib.pyplot.title(), matplotlib.pyplot.xlabel(), and matplotlib.pyplot.ylabel() functions. Finally, the plot is displayed using matplotlib.pyplot.show() function.

3. Conclusion

This Python code implements a 2D random walk simulation and performs parameter estimation and error calculation based on the results of the random walk. The code begins by importing several libraries including sys, numpy, matplotlib.pyplot, pylab, and scipy.stats which are used for various purposes. In addition, the code initializes random number generator, perform random walk simulation, estimates the parameter of interest based on the results of the random walk simulation. The code further calculates errors and creates a scatter plot of the final positions (xfinal and yfinal) in the x and y directions, respectively, after all the walks are completed.

4. Reference

1. Y. Liu, Y. Zeng, and Y. Lu. (2015). Scaling Properties of Two-dimensional Random Walks in a Confined Domain. Physical Review E, 92(3):032141

2. G. Agunbiade (2023). Simulating and Visualizing the Behavior of a Two-Dimensional Random Walk using the Monte Carlo Method. PHSX 815 Project 2