Indian Institute of Technology Bombay

PH 435: Electronics Lab IV - Microprocessors

September - November 2023

Visualising CLT through a Digital Galton Board

Kandarp Solanki - 210260026

Agnipratim Nag - 210260005

Guide - Prof. Pramod Kumar

Abstract

The Central Limit Theorem, a fundamental concept in statistics, postulates that the distribution of the sample mean from a population approaches a Gaussian or normal distribution as the sample size increases, regardless of the underlying population distribution. In this project, we explore a tangible and visual way to illustrate this theorem through the creation of a Digital Galton Board. Utilizing Arduino microprocessors, we have developed a digital simulation of the classic Galton Board, which replicates the stochastic process of balls falling through a series of pegs.

By conducting numerous simulations and varying the number of balls released, we can observe how the resultant distribution evolves. As we increase the number of samples (balls), we witness a transformation from irregular, jagged distributions to smoother, bell-shaped curves resembling the Gaussian distribution. This visual representation offers an intuitive understanding of the Central Limit Theorem, demonstrating how, with a sufficient sample size, even data from non-normally distributed populations converges towards a normal distribution.

Through this project, we provide an interactive and instructive tool for educators and students to grasp the principles behind the Central Limit Theorem, reinforcing the importance of sample size in statistical analysis. Additionally, it serves as a captivating visual demonstration of probabilistic concepts, bridging the gap between theory and real-world application. This report documents the design, implementation, and results of the Digital Galton Board, shedding light on the power of simulation and hands-on learning in the field of statistics and data science.

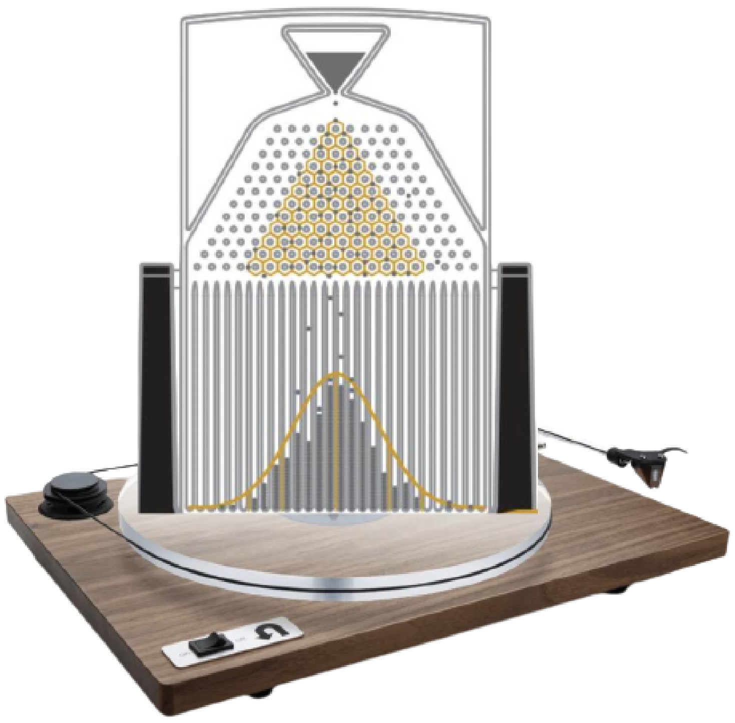


Fig. 1: A Classical Galton Board

Contents

1 Computer Implementation

1.1 Writing the Python code

The program for the theoretical implementation is quite trivial. Balls are spawned in the centre of the board and randomly shifted left or right with equal probability for every row that they move downward. Finally, at the bottom of the Galton Board, their final position is recorded and is processed by the data analysis program which illustrates it's normally distributed nature, thereby verifying CLT.

The code has been attached below.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize

import random

def simulate_galton_board(rows, columns, balls):
    positions = [0] * (columns)

    for _ in range(balls):
        position = columns//2
        for row in range(rows):
            direction = int(random.choice([-1, 1]))  # Left or right
            position += direction
        positions[position] += 1

    return positions

# Set the number of rows and balls
num_rows = 16
num_columns = 32
num_balls = 1000

final_positions = simulate_galton_board(num_rows, num_columns, num_balls)
```

1.2 Statistical Analysis

Once the program has generated the output, we use the *NumPy* library to visualise the obtained data and look for interesting conclusions, if any. What grabs our eye is that on plotting the histogram for the number of balls that end up in each column, we are able to observe an approximately bell-shaped

curve! Note that for our computer simulation, we have modelled our Arduino setup identically - using a grid which has 32 columns and 16 rows, and simulating the falling of 1000 balls. After running the simulator, this is the array obtained which denotes the number of balls in each column $\{c_i\}_{i=1}^{32}$.
The array is:

$$0, 0, 0, 0, 2, 0, 11, 0, 31, 0, 67, 0, 117, 0, 192, 0, 198, 0, 161, 0, 121, 0, 63, 0, 33, 0, 4, 0, 0, 0, 0, 0$$

.

Some observations:

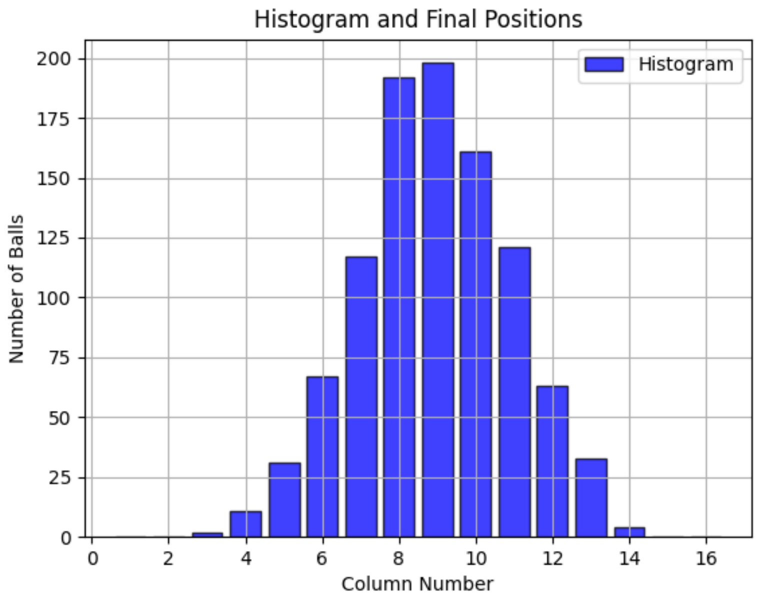
- Trailing zeros on both sides: This is because the computer model we have created is identical to our physical LED matrix which only has 16 rows, therefore the maximum leftward or rightward displacement is 16. Hence, a ball dropped in the middle (the 16th column) can move to the 0th or the 32nd column at max, and the further to an extreme it moves, the lower the probability of it happening is - therefore, it is natural to see very few balls making it to the extreme ends.
- Alternating zeros in the array: This is because we have 8 rows, which means that the ball's column number is increased or decreased by 1, 16 times. This necessarily means that the final column number must be even. Therefore all the odd numbered columns have 0 balls in the final positions array.

Therefore, we pre-process the data to remove these artifacts before we fit it to a normal curve.

The final positions array is:

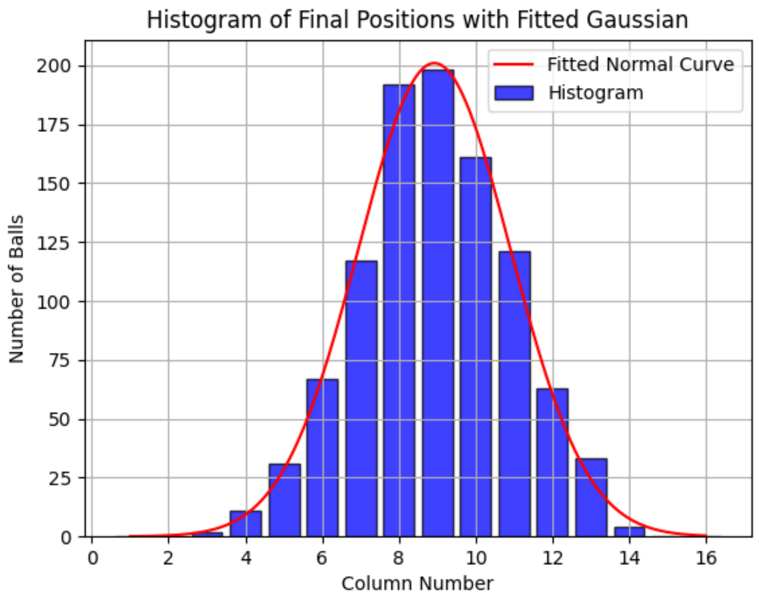
$$0, 0, 2, 11, 31, 67, 117, 192, 198, 161, 121, 63, 33, 4, 0, 0$$

.



1.3 Verifying CLT

This is the most interesting outcome of this exercise. Our randomised exercises is a valid reflection of the well known **Central Limit Theorem** of statistics, which states that in many situations, for identically distributed independent samples, the standardized sample mean tends towards the standard normal distribution even if the original variables themselves are not normally distributed. To verify that this holds for our simulation, we fit a normal curve to the plotted histogram and note down the empirically estimated μ and σ. We will later check these values with theoretically predicted ones. Now, we move on to replicating the same, on a smaller scale using Arduino.



The fitted Gaussian has $\mu = 8.93$ and $\sigma = 1.99$. This data can be interpreted as - On average, a ball ends up in the "8.93'th" column which is basically the 9th column. This physically makes sense because this is the exact centre of the board, and on average the most probably ball movement is an equal displacement both rightwards and leftwards. Recall, that we pre-processed the data which squeezed our board size causing it go from 32 rows to 16 rows, and this is why the average middle position is the 9th column. Also note that the middle column has 198 out of 1000 balls fall into it. We will verify this later theoretically.

2 Arduino Implementation

2.1 Theoretical Foundation

For our Arduino implementation, we begin by initialising a ball pixel in the middle of our LED matrix. We consider the balls to fall along the dimension of size 8 of the matrix, and we let the ball move for two iterations - so that it appears to be falling through a matrix of height 16. After two iterations of randomised left and right movement, we record the column number it finally ends up in. After this we spawn another ball, and repeat this exercise for 1000 balls. Finally, we collect the data and perform statistical analysis on it.

2.2 Setup and Configuration

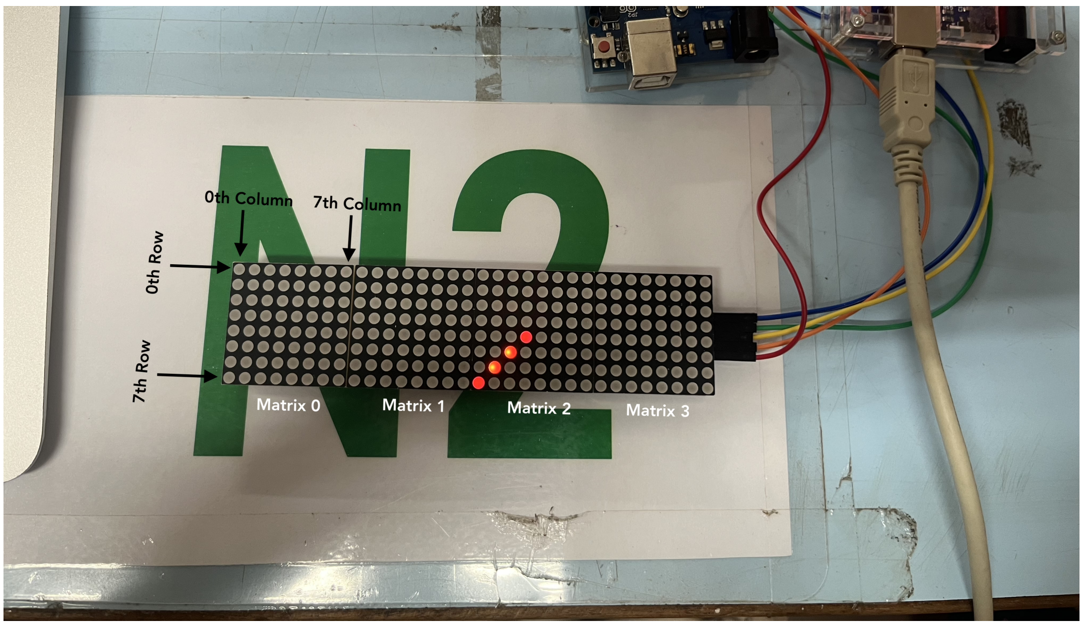


Fig 9. How are coordinates defined?

The glowing ON and OFF of an LED is controlled by three parameters namely the matrix, the row number and the column number of the LED.

- Pin 12 is connected to the DataIn
- Pin 11 is connected to the CLK
- Pin 10 is connected to LOAD
- Import the LedControl.h library

The basic flow of the code is as follows:

```
LedControl lc=LedControl(12,11,10,4); // 4 indicates total LED matrices used

lc.setLed(old_matrix,old_row,7-old_col,false); // For turning OFF
delay(1);
lc.setLed(matrix,ballRow,7-ballCol,true); // For turning ON

Some coordinate updation...

delay(1);
```

The code ensures that there is a significant delay between each glow ON and OFF so that the falling behaviour is observed nicely.

Note that we used `ballRow` and `7-ballCol` just to make sure that our coordinate system is well-defined with respect to the definition of `ballRow` and `ballCol`.

We also used a variable named `iteration` due to the physical limitation of the LED matrix. To allow the use of total available width/bins, we took two iterations of the total height since using only one iteration a ball can move a maximum of 8 coordinates but we have a space of 16 coordinates from the original spawning point.

An array named `coordinates` was created to store the index of the end position (column number) of the simulated ball.

The coordinate updation is as follows:

```
// Spawn a new ball
ballRow = 7;
ballCol = 7;
matrix = 1;/ / Always generate a ball from Matrix 1, Column 7, Row 7
iteration = 1;
ballNum += 1; // The index of the simulating ball

// Store old coordinates
old_row = ballRow;
old_col = ballCol;
old_matrix = matrix;

// Coordinate updation
ballCol += randNumber;
ballRow -= 1;
```

```
// Some condition checks to validate the ball is within bounds, else maybe jump a
matrix if it goes out of bound for a matrix

if (ballCol < 0){
matrix -= 1;
ballCol = 7;
}
else if (ballCol > 7){
matrix += 1;
ballCol = 0;
}

// Condition for starting iteration 2

if (ballRow == -1 && iteration == 1){
ballRow = 7;
iteration++;
}

// Record the X coordinate of the ball after 2 iterations
// To check if it is within bounds

else if (ballRow == -1 && iteration == 2){
int index = ballCol + matrix*8;
if ((index >= 0) && (index <= 31)){
coordinates[index] += 1;
}
```

2.3 Statistical Analysis

Using the Arduino setup mentioned above with the simulator we have designed, we run simulations for 1000 balls being randomised to move left or right across 16 rows (2 iterations with a wrap-around effect) and then record it's final position at the bottom of the LED matrix. After this, we move to the data analysis stage. All data analysis has been performed on Python using Pandas, Matplotlib and Seaborn. The code has been attached in the appendix.

The positions array is:

$$0, 0, 9, 28, 51, 111, 169, 206, 193, 129, 68, 31, 4, 1, 0, 0$$

.

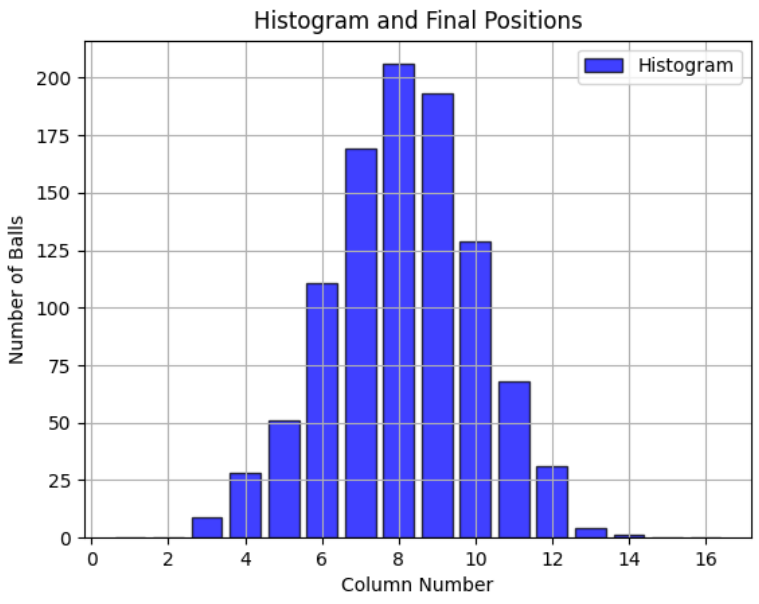


Fig 9. Data generated by Arduino circuit

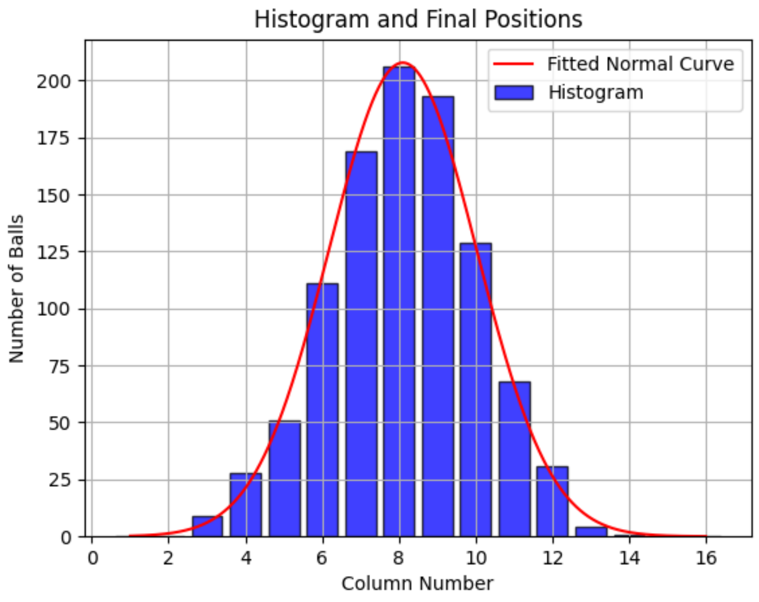


Fig 11. Gaussian Fit

Amazingly enough, the data generated by our digital circuit appears to follow a Gaussian distribution when a large number of data points are considered. The mean of our distribution is **8.10** with a standard deviation of **1.92**. The central column contains 206/1000 balls in the simulation carried out by our Arduino circuit.

Mathematical Verification:

We will calculate the probability of a ball falling into the central column i.e achieving 0 displacement while falling and compare it with experimentally obtained values.

$$P(\Delta = 0) = \binom{16}{8}(\frac{1}{2})^8(\frac{1}{2})^8$$

The reasoning behind this expression is that we can denote the displacements experienced by the ball as a 16-tuple which values equal to +1 or -1, and for a net zero displacement we require four of these values to be +1 and four to be -1. The probability of either one is equal to $\frac{1}{2}$. The expression is obtained using basic knowledge of Bernoulli Random Variables. This probability is equal to 0.196 which implies that for 1000 balls, an average of 196 balls will end up undisplaced. This is in good agreement with our computer implementation which predicted 198 undisplaced balls and in moderate agreement with our Arduino implementation which predicted 206 undisplaced balls.

3 Experimental Implementation

The experimental implementation of our project will involve a demonstration that includes the following:

- Showing the generation of a random path traversed by a simulated falling ball whose randomisation seed is decided by the analogRead1 pin which is unconnected
- Due to spatial limitations, we allow the falling simulation to happen over two iterations
- We record the final bin in which the ball reaches by storing it in an array and modify the value of the specific index corresponding to it
- We check the distribution of the array in every 1000 iterations and we expect it to get closer to normal distribution
- Owing to the fact that the total height of the digital LED board is an even number, we expect the balls to fall only in every alternative container so effectively the total bins are halved
- We will be explaining using a delay of 50 ms and then speed it up to generate data quickly!

4 Connection to Physics

The ideas used in this project have important applications in Physics and Computer Science both, in areas such as statistical mechanics and efficient search algorithms.

Statistical Mechanics:

The main motivation behind this project was to show that the **Central Limit Theorem** governs random phenomena around us, in this case - a bit sequence randomly generated by a digital circuit. However, this effect is prevalent throughout physics, and is especially useful in the field of statistical mechanics.

In statistical mechanics, the central limit theorem is used to explain the behavior of large systems of particles that interact with each other. The microscopic behavior of each particle in the system is typically governed by complex and often unknown physical laws. However, the macroscopic behavior of the system can be described by simple statistical laws that are based on the properties of large numbers of particles.

The central limit theorem is particularly useful in statistical mechanics because it allows us to approximate the behavior of large numbers of particles using a normal distribution. This approximation is useful because many statistical laws, such as the **Maxwell-Boltzmann distribution** and the **ideal gas law**, are based on the assumption that the particles in a system are distributed according to a normal distribution.

For example, the Maxwell-Boltzmann distribution describes the distribution of velocities of particles in a gas. This distribution is derived using the assumption that the velocities of the particles are normally distributed. The ideal gas law, which relates the pressure, volume, and temperature of a gas, is also based on the same assumption.

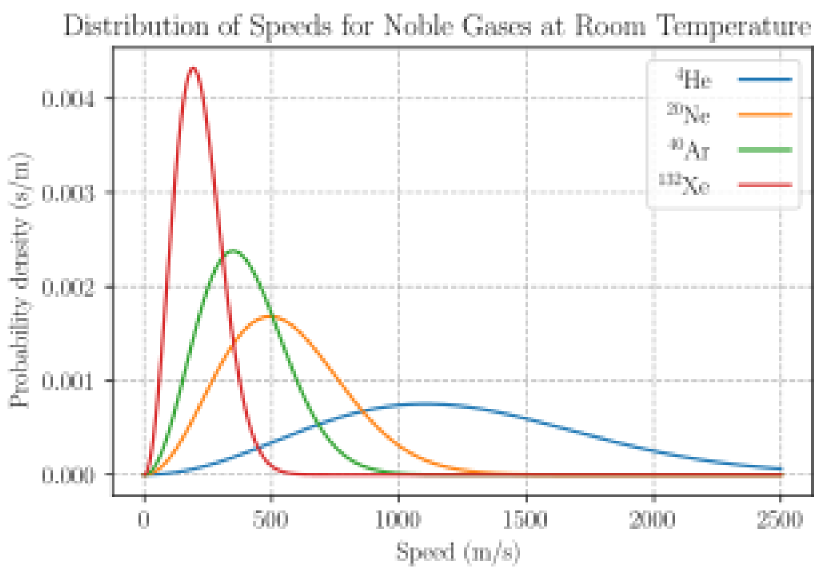


Fig 13. The Maxwell-Boltzmann Distribution

5 Conclusion

Our project successfully establishes the prevalence of the Central Limit Theorem in random distributions throughout nature, explored here through microprocessors. At first glance, we would not have expected a board of falling balls to follow such a distribution, but it turns out that it does, which is truly fascinating!

6 Acknowledgements

We would like to extend our deepest gratitude to our professors Prof. Pramod Kumar for his constant guidance throughout the duration of the course and giving us the opportunity to undertake this project. This project would also not have been possible without the support of Sir Nitin Pawar, the lab assistants and teaching assistants of PH 435.

7 Appendix

Python code for data analysis is attached below.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize

# Input your class count data as an array of 16 integers
class_counts = np.array([0, 0, 9, 28, 51, 111, 169, 206, 193, 129, 68, 31, 4, 1, 0, 0])

sum = 0

for i in class_counts:
    sum += i
print(sum)

# Create bins for your histogram (16 bins labeled 1 to 16)
bins = np.arange(1, 17)  # Bins labeled 1 to 16

# Define a function for the normal distribution
def normal_distribution(x, mu, sigma):
    return (1 / (sigma * np.sqrt(2 * np.pi)) * np.exp(-(x - mu) ** 2 / (2 * sigma ** 2)))

# Log-Likelihood function for MLE
```

```
def log_likelihood(params, x, y):
    mu, sigma = params
    expected = normal_distribution(x, mu, sigma)
    return -np.sum(y * np.log(expected + 1e-6))

# Initial guesses for mean and standard deviation
initial_params = [8, 2]  # You may adjust these as needed

# Perform MLE to estimate parameters
result = minimize(log_likelihood, initial_params, args=(bins, class_counts), method='Nelder
mu, sigma = result.x

# Plot the histogram



plt.bar(bins, class_counts, alpha=0.75, color='b', edgecolor='black', label='Histogram')
plt.xlabel('Column Number')
plt.ylabel('Number of Balls')
plt.title('Histogram and Final Positions')
plt.grid(True)

# Plot the fitted normal curve with the estimated parameters
x = np.linspace(1, 16, 1000)
y = normal_distribution(x, mu, sigma)
plt.plot(x, y * np.sum(class_counts), 'r-', label='Fitted Normal Curve')  # Scale the curve

# Show the plot with a legend
plt.legend()
plt.show()

print(f"Estimated Mean (mu): {mu:.2f}")
print(f"Estimated Standard Deviation (sigma): {sigma:.2f}")
```