Randomized Algorithm

Fall 2025

Assignment #1

Papageorgiou Nikoleta

nikoleta.papageorgiou@estudiantat.upc.edu

GitHub link:

https://github.com/nickolpap/-Galton-board

Universitat Politècnica de Catalunya

Table of Contents

1. Introduction

In this report, I implemented the Galton board simulation using Python. Each ball performs n random steps, moving left or right with equal probability ½. The final number of right moves is recorded for each ball. We then compare the empirical distribution of outcomes with the theoretical binomial distribution Bin(n, ½), and approximate it using a normal distribution N(n/2, n/4).

The comparison is visualized through plots, and the accuracy is measured using the Mean Squared Error (MSE).

All the steps are explained in the notebook *galtonbox_simulation.ipynb* and the plots and results are included in this report.

2. Code Explanation

2.1 Simulation

For the first step I simulate N balls falling through a Galton board with n steps. Each ball starts at cell (0,0) and moves randomly left or right with probability 1/2. After n steps, the ball ends at cell (i,j), where i = n -j and the final number of right moves j $\in$ {0, …, n} is recorded.

The results are stored in an array “counts”, where each entry corresponds to the number of balls with j right moves.

To illustrate how the simulation works, here is a small example using 20 balls and 6 steps.

For example 3 balls had 1 right move.

```
[9]: counts, right_moves = simulate_galton_board(6,20)
     print("Right moves per ball :" ,right_moves)
     print("Counts per position: ", counts)

     Right moves per ball : [1 3 2 3 3 2 5 3 3 1 1 3 4 3 3 5 4 3 2 2]
     Counts per position:  [0 3 4 9 2 2 0]

[ ]:
```

So, this function provides a practical way to observe the binomial distribution and enables further analysis or visualization.

2.2 Binomial Distribution

Here calculate the theoretical probabilities for each final position using the binomial distribution Bin(n, 1/2). This gives the expected probability that a ball ends up with j right moves after n steps.

The formula: p{j,n} = C(n, j) * (1/2)^n

The binomial probabilities for all positions are calculated using *scipy.stats.binom.pmf*

2.3 Expected Value and Standard deviation

This part is for the computation of the expected value of balls in each cell and the standard deviation using binomial probabilities. These values help us understand the theoretical spread and variability of the distribution.

A simple example to understand the functions. We dropped 20 balls through 6 steps. We counted how many balls ended up with 0 to 6 right moves. Then we calculated what we expect to happen using binomial probabilities.

The plot in the Figure 1 shows the real results (red) and the expected ones (blue). Small differences are normal because of randomness.

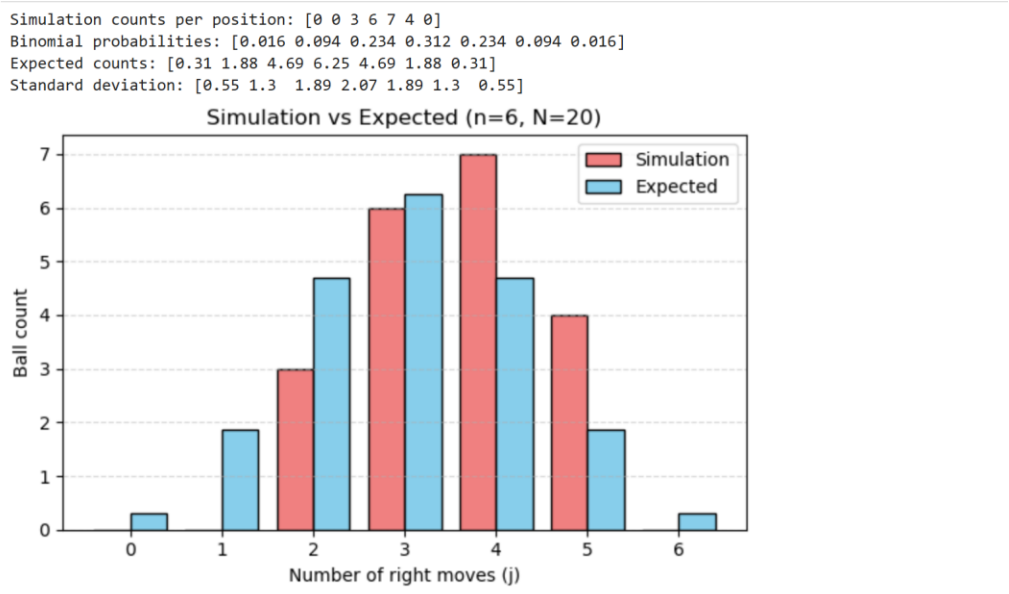

Figure 1

2.4 Normal approximation

In this part, I define and compute the normal distribution that approximates the binomial one.

For Bin(n, 1/2), the normal approximation uses:

- Mean $\mu = n / 2$

- Variance $\sigma^2 = n / 4$

This helps me see how close the binomial shape is to the normal curve, especially when n is not too small.

Because of that the normal distribution is continuous, the PDF values are not automatically sum to 1 over discrete points. So, to use them as probabilities for j = 0 to n, I normalize the array by dividing by its total sum.*(p_norm /= p_norm.sum())*

This makes sure the values behave like a proper discrete probability distribution.

2.5 Plotting Binomial vs Normal

The fifth step compares the binomial distribution with its normal approximation using a single plot. It helps us to observe how close they are.

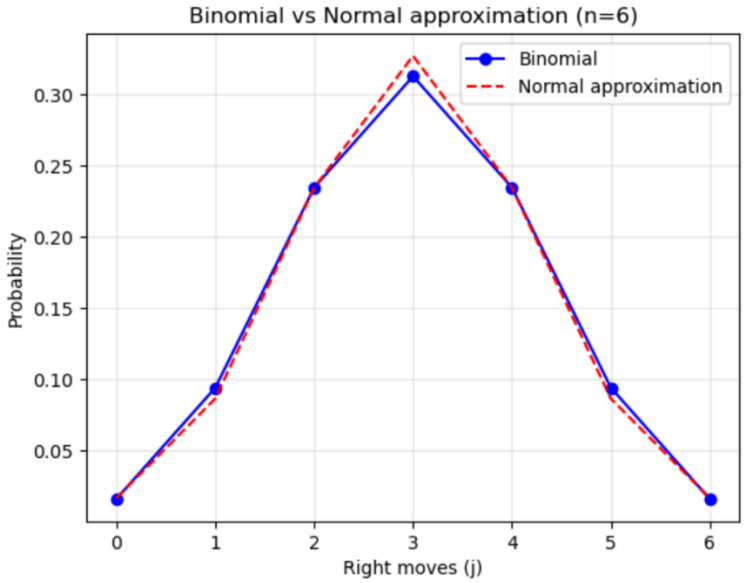

Figure 2

From the *Figure 2* we can understand that the binomial and normal curves are very close near the center (around j = 3), showing that the normal approximation works well for typical values. However, at the edges (j = 0 and j = 6), the normal curve slightly underestimates the binomial probabilities, which is expected for small n.

2.6 Mean quadratic error -MSE

The MSE quantifies how close the simulation is to the binomial distribution. Lower MSE means the results match the binomial model better.

The plot above *(Figure 2)* visually shows how close the binomial and normal distributions are.

In Example 3 of my notebook, the Mean Squared Error (MSE) provides a numerical measure of this closeness. In this case, the MSE is small, confirming that the simulation matches the binomial model well.

```
#Example3 : compute MSE

n=6
N=20
counts, _ = simulate_galton_board(n, N)
_, p_binom = binomial_distribution(n)

mse = compute_mse(counts, p_binom, N)
print("Mean Squared Error between simulation and binomial: ", mse)
```

```
Mean Squared Error between simulation and binomial:  0.0018694196428571432
```

2.7 Error between binomial and normal distribution

In Step 7, I compare the binomial distribution with its normal approximation. As the MSE gets smaller for larger values of n, it shows that the normal distribution becomes a better match for the binomial. This agrees with the **Central Limit Theorem**, which says that when n is big enough, the binomial starts to look like a normal distribution.

This is clearly shown in Example 4

```
# Example4: compute MSE for different n
for n in [6, 10, 20, 50, 100]:
    error = mse_binomial_vs_normal(n)
    print("n =", n, "MSE:", format(error, ".6f"))
```

```
n = 6 MSE: 0.000046
n = 10 MSE: 0.000008
n = 20 MSE: 0.000001
n = 50 MSE: 0.000000
n = 100 MSE: 0.000000
```

A small but important observation here is the difference between Step 6 and Step 7. Both use MSE but from different perspectives. Step 6 involves randomness from the simulation, comparing the experimental results to the binomial model. In contrast, Step 7 is a purely theoretical comparison between two probability distributions: binomial and normal.

3. Experimental Results

In order to carry out the experiments, I created the python script (galton_experiment.py) that includes all the necessary functions for:

- Simulating the Galton board using binomial.
- Computing the theoretical binomial and normal distributions
- Quantifying the differences not only between simulation&binomial but also between the pdf and the probability of the binomial distribution using MSE
- Visualizing the results using graphical plots

At the end of the script , a loop executes three experiments with increasing values of n(number of steps) and N(number of balls).

1st Experiment (n=10, N=1000)

MSE (Simulation vs Binomial): 0.000102

MSE (Binomial vs Normal): 0.000008

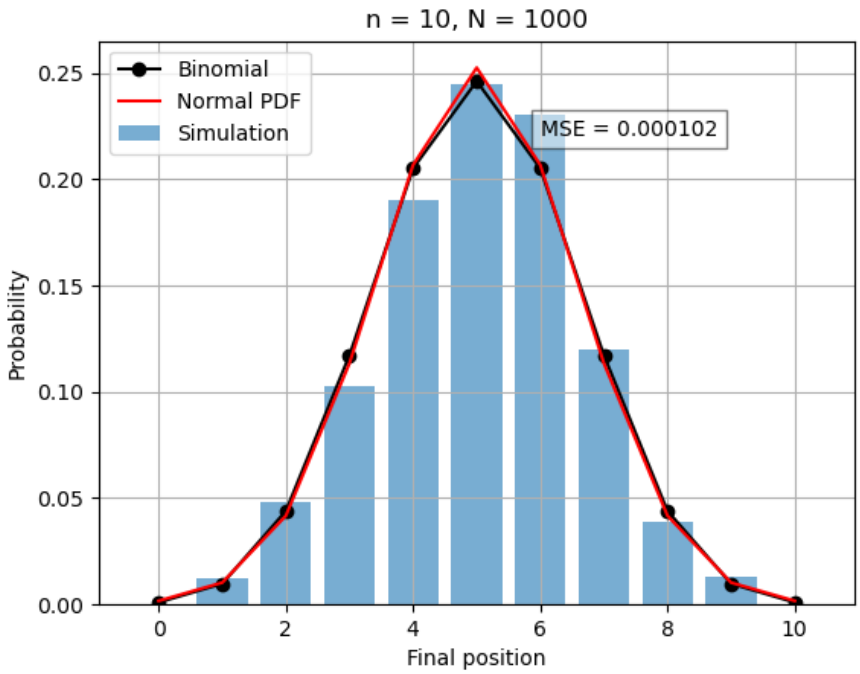

Figure 3

From the *Figure 3* we can see that the simulation is already quite close to the binomial distribution, but the normal curve doesn't match perfectly yet.

The MSE is small, but not very close to zero. This means there's still a visible difference between the distributions. The normal approximation is not very accurate at this scale.

2nd Experiment (n=20, N=5000)

MSE (Simulation vs Binomial): 0.000013

MSE (Binomial vs Normal): 0.000001

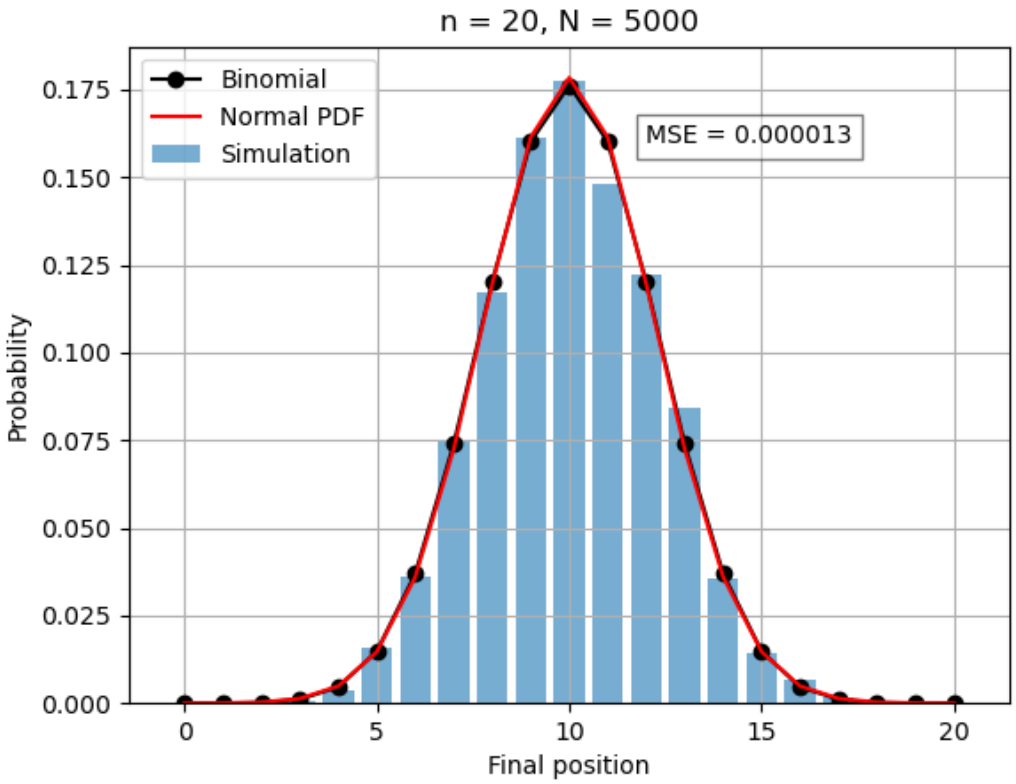

Figure 4

Here we observe that the binomial distribution starts to look more like a normal curve. The simulation matches the binomial very well. The overall shape is smoother and more symmetric. About the error, the MSE is much lower, and this shows that both the simulation and the normal approximation are getting more accurate. The distributions are very close.

3rd Experiment (n=50, N=10000)

MSE (Simulation vs Binomial): 0.000001

MSE (Binomial vs Normal): 0.000000

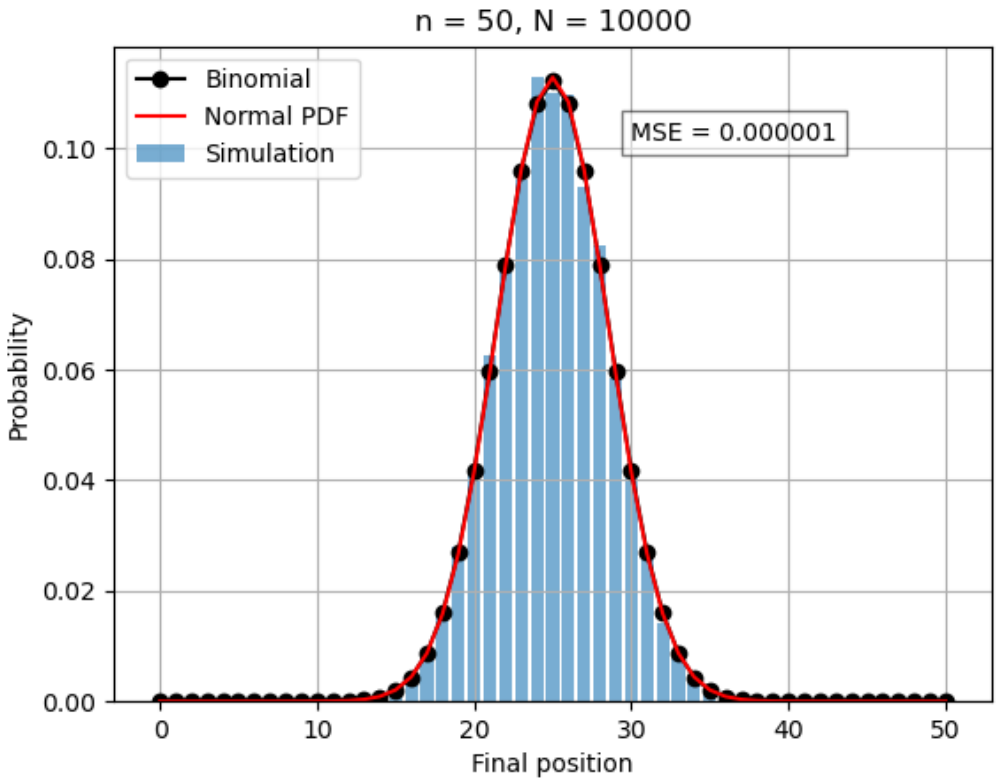

Figure 5

At this point, the Binomial and normal distributions are almost the same with also a zero MSE. Additionally, this confirms the theory that when n is big enough, the binomial starts to look like a normal distribution.

Summary

Through all three experiments, we explored how the simulation, binomial, and normal distributions evolve as the number of steps n and balls N increase. As N grows, the simulation aligns more closely with the binomial model, showing minimal error.

At the same time, increasing n leads the binomial distribution to gradually resemble the normal curve, a clear confirmation of the Central Limit Theorem.

Finally, The Mean Squared Error (MSE) helps us measure how different the distributions are. As we run more experiments, the MSE gets smaller, showing that the simulation and the normal curve become more accurate.