ISYE 6644 Simulation: Project Final Report

Topic 14: Dice and Coin

Group 54 – Wenyu Sui

I. Abstract

This work tries to estimate the expected value and the probability distribution of the number of cycles that a Dice and Coin game will last for. Suppose that for each new game, the number of cycles follows an identical independent distribution, and the game could be simulated numerous times by computer using Python programming language. According to the Central Limit Theorem [1], the true expected number of cycles that the game will last for can be estimated using the sample mean number of cycles of all the simulated games, provided the sample size of the simulated games is large enough.

II. Problem Overview

Here is an overview of the Dice and Coin game that we try to simulate. The rules of the game are as follows:

There are two players in the game, A and B. Each player will have 4 coins at the beginning of the game, and there will be another two coins in the pot. After the game starts, the players will toss a 6-sided dice in turns, starting from player A. Each player needs to do the following activities when they rolls different numbers.

- If the player rolls 1, the player will not do anything.
- If the player rolls 2, the player will take all the coins in the pot.
- If the player rolls 3, the player will take half of the coins (rounded down) in the pot.
- If the player rolls 4 - 6, the player will put a coin back to the pot.

The game will end only if one player rolls 4, 5 or 6 and the player does not have enough coins to put back into the pot. The player who fails to put back a coin into the pot will lose the game. A "cycle" is defined as both player A and B completing their activities. The exception is that if one player loses the game before the other player rolls the dice, the game will end, but it will still be counted as a cycle (the last cycle of the game).

The purpose of this project is to determine the expected number (and the probability distribution) of cycles that the game will last for by using simulation.

III. Research Methodology - Theory

According to the rules of the game, each new game is played under the same rules and settings, and each new game is independent from all the other games. Therefore, we can assume that the number of cycles of a new game, X , follows an independent and identical distribution with mean value μ , and variance σ^2 . The target of this project is to estimate the value of μ.

Suppose that n games are played. The number of cycles of each game can be denoted as X_1 , X_2 , X_3 … X_n. According to the Central Limit Theorem [1], if the sample size n is large enough, the sample mean

$\overline{X_n} = \sum_{i=1}^{n} X_i/n$ will follow a normal distribution with mean value μ, and variance $\frac{\sigma^2}{n}$. Therefore, $\overline{X_n}$ can be used as an unbiased estimator of μ. The accuracy of estimation will increase as n increases, because the variance of $\overline{X_n}$ will get smaller when n gets larger.

In order to estimate μ using simulation, we need to determine the sample size n, which is the number of games that we need to simulate. The sample size needs to be large enough to avoid significant variances of the estimated value $\overline{X_n}$. After simulating n games, we will calculate the sample mean $\overline{X_n}$, and use it as an estimate of μ. The probability distribution of random variable X can also be estimated by observing the empirical distribution of X_1, X_2, X_3 … X_n. The next section ("Python Program and Research Findings") of this report will discuss how to implement this research method in detail using Python programming language.

IV. Python Program and Research Findings

This section will discuss in detail how to carry out the research method discussed in the last section by using Python programming language. The Python script is edited in Jupyter Notebook and submitted along with this project report, named as *"Project - topic 14_coin toss game.ipynb"*. Please open the script in Jupyter Notebook and execute the script in the order from the top to the bottom. The script may generate slightly different results from different executions, since there is some randomness involved in certain steps of the program.

A copy of the program execution results in HTML format can be found in the file "*Project - topic 14_coin toss game.html*". The data used in the following parts of this report is consistent with what displayed in the HTML file.

The script contains the following sections:

1. Import Packages

In this section, the script will import all the Python packages used in the rest of the program. Please make sure that the following packages are installed before running the script: os [2], sys [3], random [4], pandas [5], numpy [6], math [7], statsmodels [8], matplotlib [9], statistics, scipy [10], IPython [11].

2. Define a Function that Simulates the Dice and Coin Game

In this section, a python function (*run_game*) is defined to simulate a new game. The function will perform the following activities every time when it is called:

1) **Initialized the game.** Three new variables (*A_coin, B_coin, pot_coin*) will be set up, representing the number of coins owned by player A, the number of coins owned by player B, and the number of coins in the pot. The values of these variables will be initialized to make sure that each player will start the game with 4 coins, and that there are 2 coins in the pot.

2) **Simulate the game.** In each cycle of the game, the python function will simulate the process of rolling a 6-sided dice by using the random number generator, $random.uniform()$, from $random$ library [4]. When each player rolls the dice, the random number generator will generate a floating-point number from a uniform distribution, $Unif(0,6)$. This number will be rounded up to its nearest integer. Therefore, we can derive a random integer ranging between 1 to 6, with each

integer having the same probability to be generated. After the random integer is generated, the python function will then simulate the player's activity by updating the number of coins owned by player A (*A_coin*), the number of coins owned by player B (*B_coin*), and the number of coins in the pot (*pot_coin*). The script will repeat this process for player A and player B alternatively in each cycle until one player loses the game. After the game is ended, the python function will stop running, and it will return the number of cycles of the game as an integer.

3. Determine the Sample Size of Simulated Games

As discussed in the "Theory" section, the variance of $\overline{X_n}$ will decrease as the sample size n increases. Therefore, we need to determine a proper value of n which makes the variance of $\overline{X_n}$ small enough, so that the value of $\overline{X_n}$ can be used as a reliable estimate of μ. To find the variance of $\overline{X_n}$, we will run simulations multiple times, generating independent replications, and use the sample variance of $\overline{X_n}$ as an estimate of the true variance of $\overline{X_n}$.

In the python script, we will try to run the simulations with sample size 10, 100, 1000, 10^4, 10^5 and 10^6 . At each level of sample size, we will generate 40 independent replications and calculate 40 different values of $\overline{X_n}$. For example, if the sample size is 100, we will simulate 100 different games, calculate $\overline{X_n}$, and repeat the same process 40 times. After repeating this process with different sample sizes, we will derive 40 $\overline{X_n}$ s with sample size 10, 40 $\overline{X_n}$ s with sample size 100, 40 $\overline{X_n}$ s with sample size 1000 … and 40 $\overline{X_n}$ s with sample size 10^6.

For each sample size level, we can calculate the sample variance of $\overline{X_n}$ using the 40 values of $\overline{X_n}$ derived from the independent replications. The sample variance of $\overline{X_n}$ will be used as an estimate of the true variance of $\overline{X_n}$ at each sample size level. Then we can compare the sample variance of $\overline{X_n}$ at different sample size levels. Finally, we can choose a proper sample size n so that the variance of $\overline{X_n}$ will be small enough and will not affect the accuracy of our estimate of μ.

After running this section of the Python script, we can derive the sample variances of $\overline{X_n}$ at different sample size levels as below:

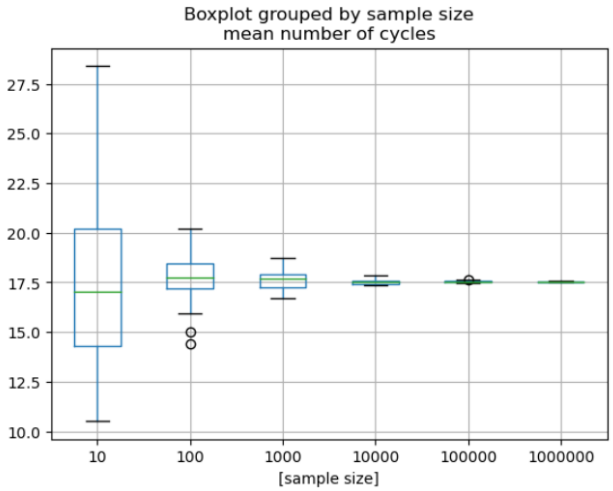

Figure 1 - Distribution of $\overline{X_n}$ at different Sample Size Level

	sample size	sample standard deviation
0	10	4.385400
1	100	1.243875
2	1000	0.449403
3	10000	0.131997
4	100000	0.042793
5	1000000	0.011380

Table 1 - Sample Variance of $\overline{X_n}$ at Different Sample Size Level

From the charts above, we can observe that the sample variance of $\overline{X_n}$ decreases quickly as n increases. Since the sample variance of $\overline{X_n}$ equals merely 0.01138 when $n = 1{,}000{,}000$, we can conclude that a reliable estimate of μ can be derived by simulating one million different games. In the following sections, we will use $n = 1{,}000{,}000$ as our sample size to run the simulation.

4. Execute Simulations

In this section, the python script will simulate one million different games ($n = 1{,}000{,}000$) by calling the function (*run_game*) defined in step 1. The number of cycles of the simulated games X_1 , X_2 , X_3 … $X_{1{,}000{,}000}$ will be stored in a list variable (*cycle_list*).

5. Statistical Analysis

After one million games are simulated, the sample mean $\overline{X_n} = \sum_{i=1}^{1{,}000{,}000} X_i / 1{,}000{,}000$ can be calculated from the simulation results X_1 , X_2 , X_3 … $X_{1{,}000{,}000}$. We would use the value of $\overline{X_n}$ to estimate μ. Here is the histogram of X_1 , X_2 , X_3 … $X_{1{,}000{,}000}$, and some descriptive statistics derived from the simulations:

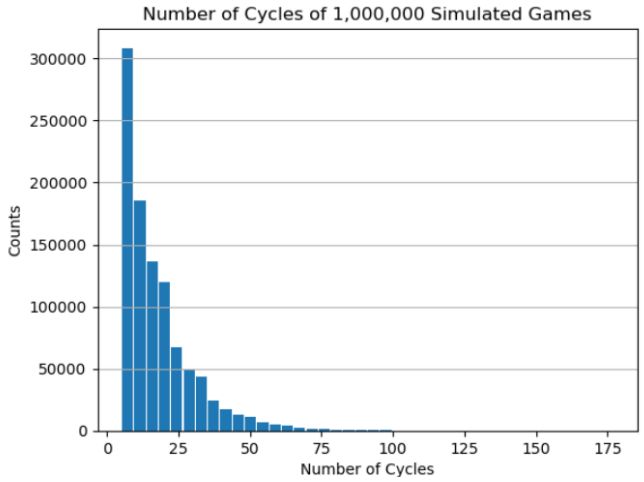

Figure 2 - Histogram of Number of Cycles

Descriptive Statistics	
count	1000000.00
mean	17.53
std	12.75
min	5.00
25%	8.00
50%	14.00
75%	22.00
max	177.00

Table 2 - Descriptive Statistics

From the results above, we can see that $\overline{X_n} = 17.53$. Therefore, we estimate that the true value of the expected number of cycles $\mu = 17.53$.

According to the Central Limit Theorem [1], $\overline{X_n}$ follows a normal distribution with mean value μ , and variance $\frac{\sigma^2}{n}$. Therefore, the 95% confidence interval of μ can be calculated as follows:

$$\left(\overline{X_n} - Z \times \frac{\hat{\sigma}}{\sqrt{n}} , \overline{X_n} + Z \times \frac{\hat{\sigma}}{\sqrt{n}}\right)$$

where $\hat{\sigma}$ represents the sample standard deviation of X_1 , X_2 , X_3 … $X_{1{,}000{,}000}$, n represents the sample size which equals one million, and Z equals 1.96 (for 95% confidence interval of a normal distribution). Therefore, the 95% confidence interval of μ is (17.50884 , 17.55883).

The empirical c.d.f. function of X can be derived from the simulation results as follows [12]:

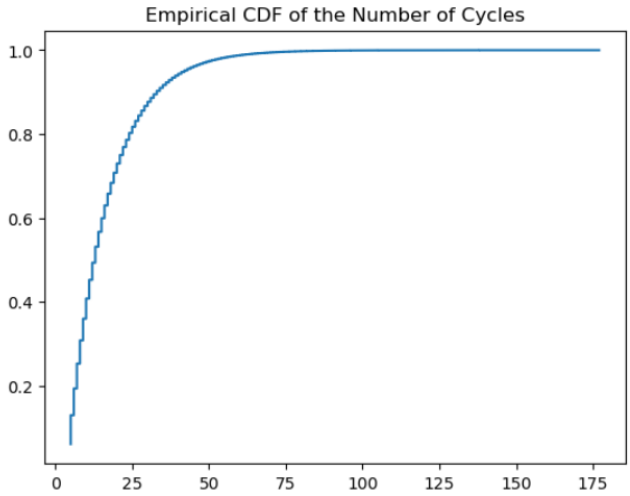

Figure 3 - Empirical c.d.f. of X

V. Conclusions

This project estimates the expected value and the probability distribution of the number of cycles that a Dice and Coin game will last for. By comparing the sample variance of the mean number of cycles $\overline{X_n}$ at different sample size levels, we determine that a reliable estimate of the true value of expected number of cycles can be derived if we simulate the game one million times.

The Python script simulates the game one million times and estimates that the expected number of cycles equals 17.53. It also shows that the 95% confidence interval of the true expected number of cycle is (17.50884 , 17.55883). The probability distribution of the number of cycles is displayed in histogram (Figure 2), and its empirical c.d.f. function is also visualized in a curve graph (Figure 3).

References

[1] Wikipedia, "Central Limit Theorem - Wikipedia," 10 July 2023. [Online]. Available: https://en.wikipedia.org/wiki/Central_limit_theorem. [Accessed 10 July 2023].

[2] "os — Miscellaneous operating system interfaces, version 3.11.4," [Online]. Available: https://docs.python.org/3/library/os.html. [Accessed 1 July 2023].

[3] "sys — System-specific parameters and functions, version 3.11.4," [Online]. Available: https://docs.python.org/3/library/sys.html. [Accessed 1 July 2023].

[4] "random — Generate pseudo-random numbers, version 3.11.4," [Online]. Available: https://docs.python.org/3/library/random.html#module-random. [Accessed 1 July 2023].

[5] Pandas, "pandas documentation," 28 June 2023. [Online]. Available: https://pandas.pydata.org/docs/.

[6] Numpy, "NumPy documentation, Version: 1.25," [Online]. Available: https://numpy.org/doc/stable/index.html. [Accessed 1 July 2023].

[7] "math — Mathematical functions, version 3.11.4," [Online]. Available: https://docs.python.org/3/library/math.html. [Accessed 1 July 2023].

[8] statsmodels, "statsmodels 0.14.0," [Online]. Available: https://www.statsmodels.org/stable/index.html. [Accessed 1 July 2023].

[9] Matplotlib, "Matplotlib 3.7.2 documentation," [Online]. Available: https://matplotlib.org/stable/index.html. [Accessed 1 July 2023].

[10] SciPy, "SciPy documentation, version 1.11.1," 28 June 2023. [Online]. Available: https://docs.scipy.org/doc/scipy/.

[11] IPython, "IPython Documentation 8.14.0," 2 June 2023. [Online]. Available: https://ipython.readthedocs.io/en/stable/#.

[12] Dave, "Python - How to plot empirical cdf (ecdf)," 7 July 2012. [Online]. Available: https://stackoverflow.com/a/11692365/22057460.