"Coins in a Pot" Simulation

ISYE-6739 - Statistical Methods

Kadhir Umasankar

December 7th, 2021

Abstract

Statistical analysis of games can be done by running many simulations. Statistics of interest can be recorded over many simulations, and when running thousands of such simulations, the estimations of statistics will begin to converge to the parameters' actual values. In this report, analysis of the "Coins in a Pot" game will be described (the rules of this game will be explained in the report). The expected value and distribution of the number of cycles that the game will last for will be found. The process that was followed to write the Python code to simulate the game will be detailed. Plots of the distribution at various timesteps and the movement of the mean of the distribution over many simulations will be shown. Finally, the importance of writing code and simulating games as opposed to doing calculations by hand will be discussed.

1 Introduction

Analysis of games can be done by running many simulations. Code must be written to convert the problem at hand into a simulation that the computer can run, and statistics of interest can be recorded over many runs. By running thousands of such simulations, the estimates will begin to converge to the parameters' actual values, following the Law of Large Numbers, which can be seen in Equation 1. The Law of Large Numbers says that "the average of the results obtained from a large number of trials should be close to the expected value and will tend to become closer to the expected value as more trials are performed"[1].

$$\bar{X_n} = \frac{1}{n}(X_1 + ... + X_n); \bar{X_n} \to \mu \text{ as } n \to \infty \tag{1}$$

In this report, analysis of the "Coins in a Pot" game will be described. In this game, two players start with 4 coins each, and the pot starts with 2 coins. The players take turns rolling a die, and they perform one of the following actions based on the number they roll:

Number on Die	**Action**
1	Player does nothing
2	Player takes all coins in the pot
3	Player takes half of the coins in the pot (rounded down)
4, 5, 6	Player puts a coin in the pot

The game ends when a player must put a coin in the pot but has no coins left. The number of cycles that the game lasts must be recorded, and its expected value and distribution are to be found.

Following these rules, some Python code was written to simulate the game. Section 2 of this report will discuss this code. The number of cycles that were played until a player ran out of coins was recorded in a list, and this process was repeated many times. The mean of the number of cycles will be found, and the distribution of the number of cycles will be plotted. Section 3 will contain plots of the simulation, as well as a discussion of the results.

2 Simulation Code

First, the logic of the game was coded in Python. A `Player` class was created to make it convenient to hold and handle transactions of coins between the players and the pot, and the two players and the pot were created as `Player` objects. A `for` loop was created, which would run iterations as specified by `num_trials`. A nested `while` loop was created, which would run until a player had to put their coins into the pot but had none left. The number of cycles until the end of the game was recorded in a list, and the process was repeated. At the end of `num_trials` iterations, the code plotted a histogram showing the distribution of the number of cycles, and calculated the mean of the distribution of cycles.

[1] `https://en.wikipedia.org/wiki/Law_of_large_numbers`

Some quality-of-life improvements were also made, such as using the `tqdm` package to show a progress bar as the code was performing iterations, and using `bashplotlib` to plot an ASCII version of the histogram in the command-line as the code was executing, an example of which can be seen in Figure 1. This code can be seen in the project's GitHub repository[2], as well as in Appendix A.

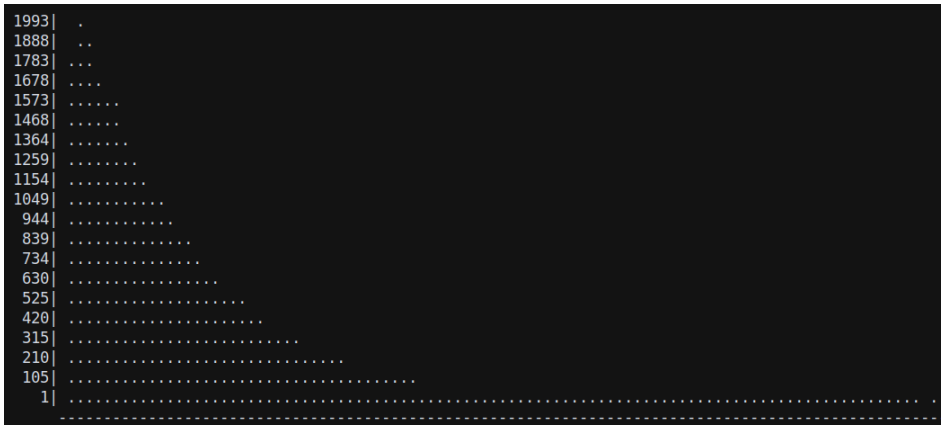
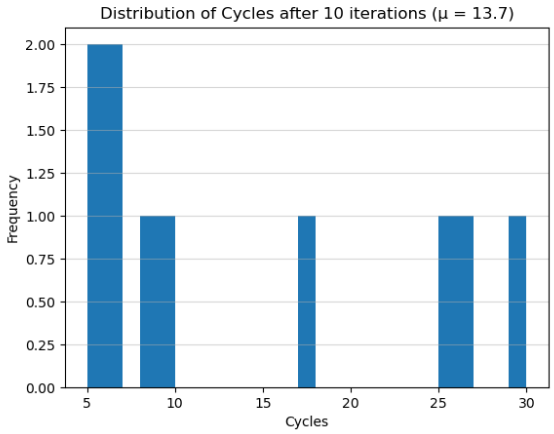


Figure 1: Screenshot of a histogram on the command-line, plotted using `bashplotlib`, to allow users to see changes in the distribution as the code is executing

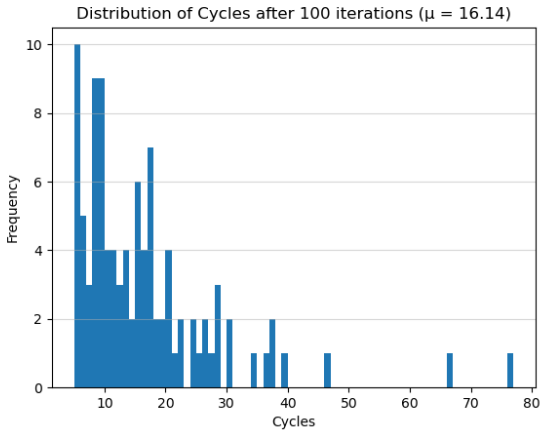
3 Results and Discussion

Plots of the distribution of cycles after simulating 10, 100, 1,000, and 10,000 games can be seen in Figures 2a through 3b. A plot of the distribution of cycles after simulating 1,000,000 games can be seen in Figure 4.

[2] `https://github.com/kadhirumasankar/isye-6739-project`

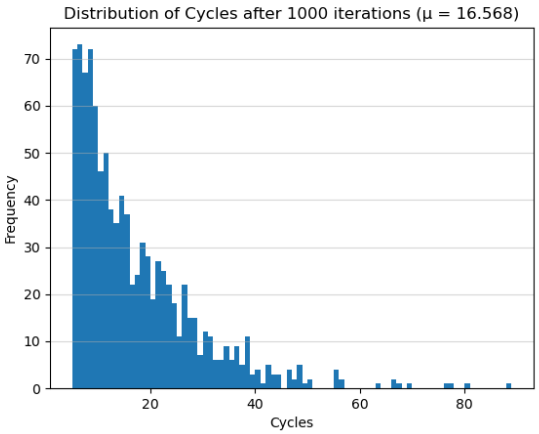


(a) 10 games

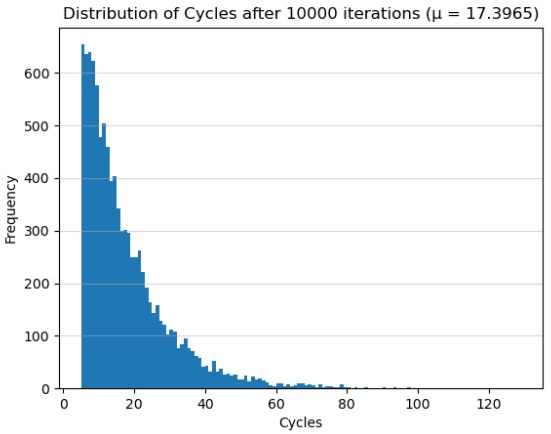


(b) 100 games

Figure 2: Plots of the distribution of cycles after simulating various numbers of games



(a) 1000 games



(b) 10000 games

Figure 3: Plots of the distribution of cycles after simulating various numbers of games

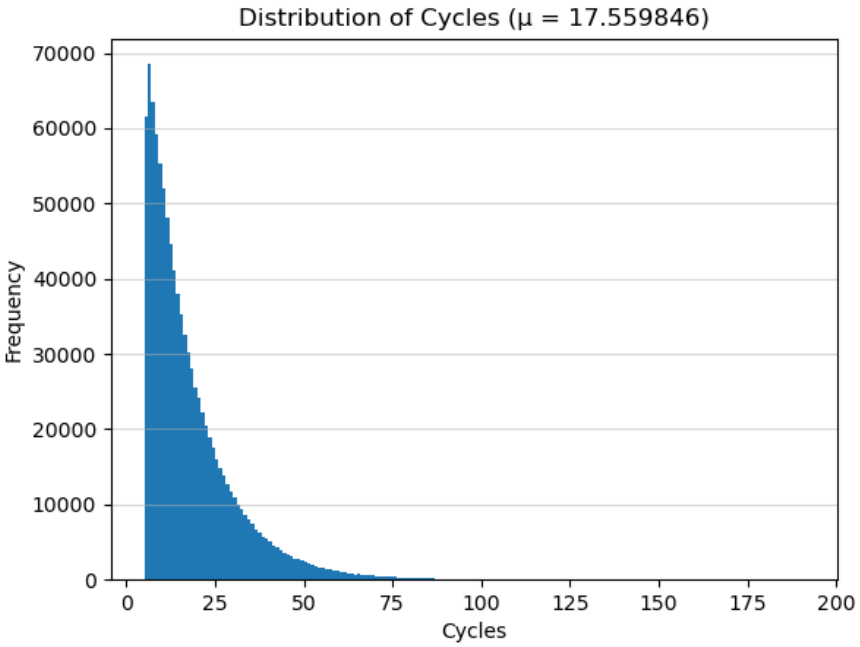


Figure 4: Plots of the distribution of cycles after simulating 1,000,000 games

It can be noted from these figures that the lowest number of cycles is 5 cycles. This is intuitive, since the players start with 4 coins, and would have to roll a 4, 5, or 6 five times in a row to lose the game in the least number of rolls. It can also be seen that the distribution of cycles becomes highly right-skewed as the number of games played increases. It is also worth noting that although the distribution looks like an exponential distribution, that is not so, as a property of the exponential distribution is that it is constantly decreasing, whereas it can be seen in Figure 4 that the frequency of '5 cycles' is lower than '6 cycles'.

To find the expected value of the distribution, the mean of the distribution of cycles was found. The movement of the mean over 1,000,000 games can be seen in Figure 5. It can be seen in this plot that the mean of the distribution eventually balances around 17.5603 cycles.

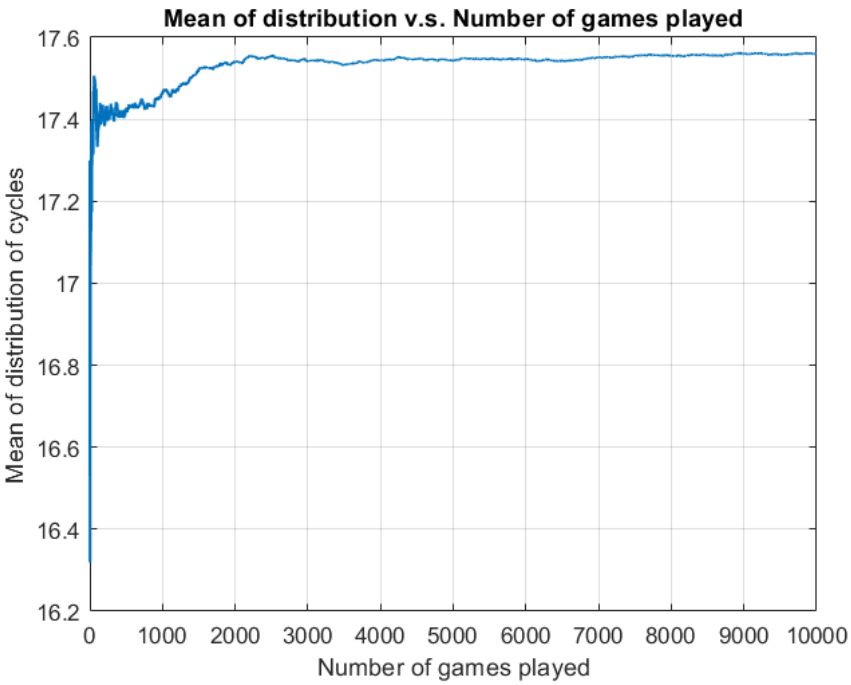


Figure 5: Mean of the distribution of cycles over 1,000,000 simulations

4 Conclusions

For this project, the expected value and distribution of the simple "Coins in a Pot" game was found using simulation. Code was written in Python following the rules of the game, and was executed many times. As the number of repetitions increased, the statistics began to converge to certain values, which is consistent with the Law of Large Numbers.

Although the "Coins in a Pot" game is seemingly simple, calculating its expected value by hand is a very arduous task. It is possible to find the expected value through first-step analysis, but, although this game is much simpler than a game such as Blackjack, the equation would grow rapidly and the number of terms in the equation would quickly become unmanageable. In more complicated games with more outcomes, finding the expected value would be unfeasible. In such cases, it is simpler to write code that accurately simulates a game, run the simulation millions of times, and record statistics of interest. Through this project, the importance of simulations for statistical analysis was revealed.

Appendix A

Python Code

```python
import csv
import math
import random
import statistics
import subprocess
import time

import matplotlib.pyplot as plt
from bashplotlib.histogram import plot_hist


class Player:
    def __init__(self, id, coins):
        self.id = id
        self.coins = coins


def print_scores():
    print("-------------")
    print("| A |Pot| B |")
    print("-------------")
    print(f"| {player_A.coins} | {pot.coins} | {player_B.coins} |")
    print("-------------")


def print_ascii_hist():
    if len(cycle_list) > 0:
        plot_hist(
            cycle_list, pch=".", bincount=math.ceil((max(cycle_list) - min(cycle_list)))
        )
        # print(f" = {mu_list[-1]}")


if __name__ == "__main__":
    cycle_list = []
    mu_list = []
```

```
num_trials = 100000
for i in range(num_trials):
    player_A = Player("A", 4)
    player_B = Player("B", 4)
    pot = Player("pot", 2)
    counter = 0
    while True:
        if counter % 2 == 0:
            current_player = player_A
        else:
            current_player = player_B
        num = random.randint(1, 6)
        if num == 1:
            # print(f"{current_player.id} rolled {num}, and does nothing")
            pass
        elif num == 2:
            current_player.coins += pot.coins
            pot.coins = 0
            # print(f"{current_player.id} rolled {num} and took all the coins from the pot. {current_player.id} has {current_player.coins} coins")
        elif num == 3:
            current_player.coins += pot.coins // 2
            pot.coins -= pot.coins // 2
            # print(f"{current_player.id} rolled {num} and took half of the coins from the pot. {current_player.id} has {current_player.coins} coins")
        else:
            if current_player.coins == 0:
                # print(f"{current_player.id} rolled {num} but has 0 coins. The game ended on cycle {counter//2 + 1}")
                cycle_list.append(counter // 2 + 1)
                break
            current_player.coins -= 1
            pot.coins += 1
            # print(f"{current_player.id} rolled {num} and put one coin in the pot. {current_player.id} has {current_player.coins} coins")
        counter = counter + 1
    if (i + 1) % 2000 == 0:
        subprocess.run(["clear", "-x"])
        print_ascii_hist()
        print(
            f"{i+1} of {num_trials} runs complete ({math.ceil(i/num_trials * 100)}%)"
        )

file = open("cycle_list.csv", "w+", newline="")
with file:
    write = csv.writer(file)
    write.writerow([str(r) for r in cycle_list])

plt.hist(cycle_list, bins=math.ceil((max(cycle_list) - min(cycle_list))))
plt.xlabel("Cycles")
plt.ylabel("Frequency")
```

```
84      plt.title(f"Distribution of Cycles ( = {statistics.mean(cycle_list)})")
85      plt.grid(True, axis="y", alpha=0.5)
86      plt.show()
```