IX1501 Project 1

Sophie Malmberg & Adalet Adiljan

September 9, 2024

Probability distribution of a sum of independent random variables

sophiema@kth.se
adalat@kth.se

1 Introduction

The goal of this project was to calculate the probability of winning a game given five different types of dice consisting of platonic solids. The platonic solids consists of convex polyhedra with four, six, eight, twelve, or twenty sides [1]. The sum S represents the total value of all dice. A game is won if $S \leq 10$ or $S \geq 45$. The code in this project is based on the examples given in the assignment [2] and was written using Python.

2 Summary of the results

The Monte Carlo simulations show that the relative errors of the probability approximations decrease as the amount of trials increase, converging towards the calculated exact probability value. In our simulation, the relative error reaches below ten percent when the trial amount reaches at least ten thousand iterations. Discrete convolution was used to compare this exact probability with the simulation, to see how reliable this method is, when considering the nature of randomness. Conclusively, we saw that the result of the discrete convolution matched the result of the Monte Carlo approximations when using a large amount of trials.

3 Description of the methodology

3.1 Task 1-2

The purpose of task 1 was to determine the probability function of S, which represents the sum of all possible dice value combinations, going from the smallest to the largest value that each side of the five platonic solids can obtain after being thrown.

The discrete convolution method was [2] used to obtain the exact probability distribution. Since it is deterministic, this exact distribution was used in order to understand the probabilities of different sums that could result from rolling the dice.

3.1.1 Solving task 1-2

A probability mass function (PMF) [4] was generated for each die by creating an array with a length of the amount of sides n. Since the distribution of probabilities is uniform for these dice, each probability value in the PMF was set to 1/n.

Using the built-in `numpy.convolve()` function in Python, the PMF of each die were combined through convolution. The operation was performed iteratively for each die: starting from two dice, storing the results in a variable and proceeding with the next in the sequence.

The second task was completed by adding the probabilities for $S \leq 10$ and $S \geq 45$ in order to calculate the total probability of winning [4]:

$$P(\text{Winning}) = \sum_{s=5}^{10} P(S = s) + \sum_{s=45}^{50} P(S = s)$$

3.2 Task 3-5

While the discrete convolution is used to derive the precise probability of winning, the Monte Carlo [2] simulation can be used when approximating probabilities given a certain amount of trials. The result of tasks 1-2 was utilized when comparing the approximated probability values to the exact calculated probability.

3.2.1 Task 3

The Monte Carlo method was utilized in order to estimate the probability of winning the game using a thousand trials. The value of each die was represented by a randomized value given by the Python method `numpy.random.randint()`. These values were then added together in order to retrieve the sum S. If this sum met the requirements of winning, a `counter` variable increased. The `counter` variable represented the amount of games that had been won during each Monte Carlo simulation. This variable was then divided by the total amount of trials, resulting in the final probability of winning.

3.2.2 Task 4

The fourth task required us to analyze the approximated probabilities when increasing the amount of trials. This was solved by repeating the process with an increasing number of trials, i.e: 1000, 3000, 10 000, 30 000, 100 000, 300 000 and 1 000 000 trials. The result of these Monte Carlo simulations were plotted using the `matplotlib` library in Python (see figure 1).

3.2.3 Task 5

In task 5, it was necessary to determine the number of trials that were needed in order to keep the relative error below a certain threshold (10 percent). In order to find the relative error between the exact probability and the estimated probability, we calculated the difference between the estimated probabilities and the exact probability given in task 1-2. The resulting plot (figure 2) displays the minimum number of trials required to keep the relative error below the threshold (in percent).

4 Mathematical formulas and equations

4.1 Task 1

4.1.1 Probability mass function

Each die in the Platonic Solid collection has its own probability mass function (PMF). Since the dice are fair (uniform), meaning each face has an equal chance of being rolled, the PMF assigns an equal probability to each face given as following:

$$P(X_n = i) = \frac{1}{n} \quad \text{for} \quad i \in \{1, 2, ..., n\}.$$

where 'n' stands for the number of sides for the platonic solid, where each side is a discrete random variable.

We can extend this concept to the sum of multiple dice rolls. For example, if we roll five dice, the sum S is given by:

$$X_1 \in \{1, 2, 3, 4\}, \quad (...)\ X_5 \in \{1, 2, ..., 20\},$$

This, the total sum is:

$$S = X_1 + X_2 + (...) + X_5$$

Where X_n are the outcomes of each of the five dice. The probability distribution of S, the sum of the five dice can be determined by convolving the probability mass function of each mass function of each individual die which provides the exact probability of each possible sum S when rolling the platonic solids.

4.1.2 Discrete Convolution

The distribution of a sum of two independent discrete random variables $S = X_1 + X_2$ can be determined by the following reasoning. If $S = s$ and we have $X = i$, then clearly $X_2 = s - i$. Given two dices (disc. independent variables), we will obtain: $X_tot = X_1 + X_2$ Since X_1 and X_2, where the probability is given by [2]:

$$P(X = i \cap X_2 = s - i) = P(X = i)P(X_2 = s - i) = \mathrm{pf}_X(i)\mathrm{pf}_Y(s - i).$$

4.2 Monte Carlo Simulation

In this project, the Monte Carlo method is used in order to estimate the chance of winning a game by conducting a large number of trials, each representing a single play of the game. The probability can be calculated using the following formula [2]:

$$P(\text{Winning}) = \frac{\text{Number of Wins}}{N}$$

The denominator represents the total number of trials in the simulation. For each trial 'N', the platonic solids are 'rolled' in which their sums are calculated. The numerator represents the number of trials in which the winning conditions are met: In each trial, we check whether the sum S of the dice satisfies the winning conditions $S \leq 10$ and $S \geq 45$. Mathematically, they are accumulated as we go through each given trial i . Meaning, we receive the cumulative sum of how many times the winning condition was satisfied over N trials (where each trial has its own S_i :

$$P(\text{Winning}) = \sum_{i=1}^{N} I(\text{Si})$$

where the indicator function takes the value 1 if the sum S in a trial 'i' satisfies the winning condition, and 0 otherwise.

Together with the indicator function, if we divide this by N we obtain the general formula for estimating the probability of winning (solely the met conditions) [6]:

$$P(\text{Winning}) = \frac{1}{N} \sum_{i=1}^{N} I_{\text{win}}(i)$$

4.2.1 Relative Error Formula

The relative error formula is given by [3]:

$$\text{Relative Error}_N = \frac{|P_{\text{MC}_N}(\text{Winning}) - P_{\text{exact}}(\text{Winning})|}{P_{\text{exact}}(\text{Winning})}$$

Considering the threshold of 10 % will give us:

$$\frac{|P_{\text{MC}}(\text{Winning}) - P_{\text{exact}}(\text{Winning})|}{P_{\text{exact}}(\text{Winning})} \leq 0.10$$

The nominator entails the difference between the absolute difference of how much the simulations result deviates from the true value for the winning conditions. 'N' indicates the minimum trials needed to maintain under the threshold, which can be calculated with the Monte Carlo formula.

5 Numerical results

This section focuses on the probability distribution of the sums of five Platonic solids and the probability of winning the game based on the conditions given in the assignment [2].

5.1 Task 1

The resulting probability function P(S=s) is shown in table 1, where the probabilities for each possible S are generally small while the highest probability occurring around the middle of the ranges of sum. Since each dice is considered a discrete random variable, by nature it is expected that the probability is distributed this way.

s	**P(S = s)**
5	0,000
6	0,0001
7	0,0003
8	0,0008
⋮	⋮
47	0,0008
48	0,0003
49	0,0001
50	0,000

Table 1: Probability function $P(S = s)$ for different values of the sum$s$.

5.2 Task 2

The exact probability of winning the game was calculated by summing the probabilities for the two winning conditions: $S \leq 10$ and $S \geq 45$. By using the probability distribution obtained in Task 1, the total probability of winning can be calculated as the following:

$$P(\text{Winning}) \approx 0.01067 = 1.067\%.$$

5.3 Task 3

The Monte Carlo simulation (with 1000 trials) results in the following approximation:

$$P(\text{Winning}) \approx 0.013 = 1.3\%.$$

5.4 Task 4

The Monte Carlo simulation was also applied to study the estimated probabilities when increasing the amount of trials. The result (figure 1) shows that an increasing amount of trials resulted in values that seemed to converge toward the expected value of 1.067%.

trial amount	**probability**
1 000	0.0130
3 000	0.0150
10 000	0.0100
30 000	0.0106
100 000	0.0105
300 000	0.0108
1 000 000	0.0108

Table 2: Probability of winning, Monte Carlo simulation

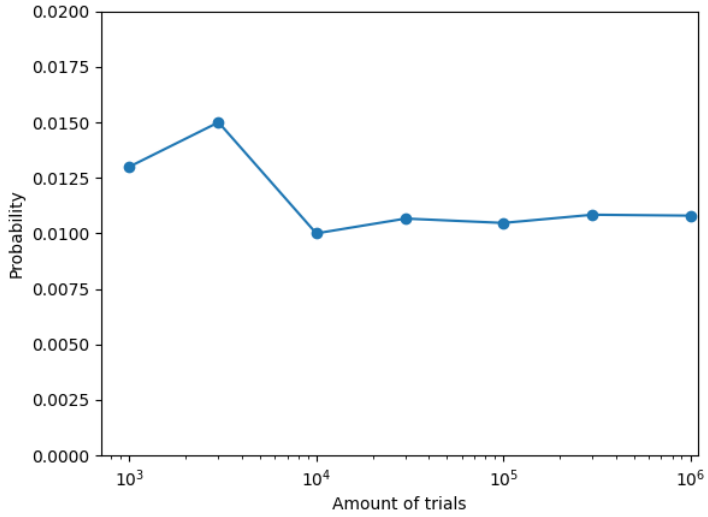

Figure 1: Probability of winning, Monte Carlo simulation

5.5 Task 5

The number of trials needed for the error rate to reach the expected threshold (below 10%) is calculated to be at least ten thousand, as shown in figure 2 and table 3.

trial amount	**error**
1 000	21.7
3 000	40.5
10 000	6.3
30 000	0.01
100 000	1.9
300 000	1.5
1 000 000	1.14

Table 3: Relative error using Monte Carlo simulation compared to the exact probability, in percent

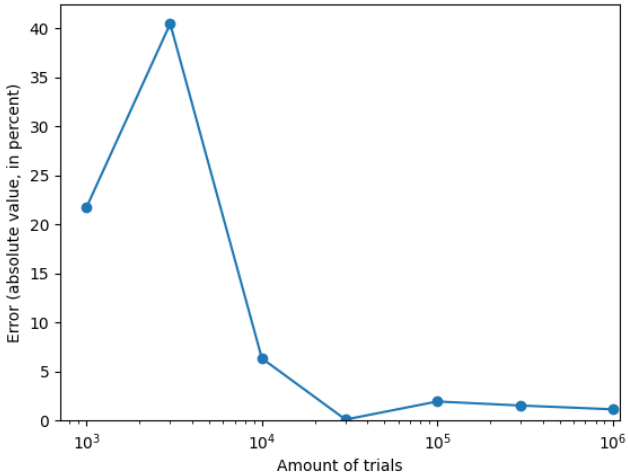

Figure 2: Relative error of the Monte Carlo simulations, in percent

6 Analysis and Discussion

The results of the Monte Carlo simulations show that an increasing amount of trials decrease the error rate compared to the exact probability rate of winning. The first trial, with one thousand iterations, had a relative error of approximately 20 percent, while the other trials approached the correct value as the amount of iterations increased.

The only exception to this pattern is the simulation with tree thousand trials, where the error rate increased compared to the previous simulation. This proves that the error rate can vary greatly when utilizing the Monte Carlo method with a smaller amount of trials.

When calculating the probability, the trials with more than ten thousand iterations approximated the correct value within a relative error rate of less than ten percent.

One explanation to the improved accuracy could be because the method relies on repeated random sampling, and by running more trials, we allow the simulation to become similar to the theoretical, exact probability. With fever trials, the simulation estimates are more susceptible to random fluctuations, because we don't have enough of data to simulate the exact prediction. Every trial has therefore a more significant impact on the overall estimation. The random variations becomes averaged out, and the simulation therefore becomes more stable.

The relative error also gives us an insight that proves this: By running enough trials, it can match the exact results within a small margin of error. The relative error decreases as the number of trials increases, therefore confirming the simulations convergence.

The discrete convolution is deterministic and doesn't include the chances of each outcome with randomness. Therefore, it shows that the values in the middle of the distribution near the average sum has a higher chance of occurring due to the central limit theorem [5]. The CLT states that, given a large sample size, the distribution of the sample means will approximate to a 'normal distribution' even if the population distribution itself is not normal. The population distribution refers to the values that make up the population, not just their sum. As we average them, extreme

values tend to cancel each other out since the lower and the higher means are less frequent. The sample mean tends to cluster around the sample means, converging to a normal distribution and making the values around the mean more probable.

Therefore by running the simulation with enough trials we can also see the same result for the Monte Carlos simulation. The reason to this result is because there are more ways of combining the values in-between $S \leq 10$ and $S \geq 45$ than for those two cases.

7 Code

```
import random
import matplotlib.pyplot as mplot
import numpy as np

#TASK 1
#Creates a probability mass function for each die
d4 = np.ones(4) / 4
d6 = np.ones(6) / 6
d8 = np.ones(8) / 8
d12 = np.ones(12) / 12
d20 = np.ones(20) / 20

# Calculates the discrete convolution
result = np.convolve(d4, d6)
result = np.convolve(result, d8)
result = np.convolve(result, d12)
result = np.convolve(result, d20)

#Prints the probability of each sum
for i in range(0, 46):
    print("S:", i+5 , ", P:", round(result[i],4))
```

```
#TASK 2
# Adds the probability of the sum S <= 10
probabilityOfWinning = 0
for i in range(0,6):
    probabilityOfWinning = probabilityOfWinning + result[i]

# Adds the probability of the sum S >= 45
for i in range(40,46):
    probabilityOfWinning = probabilityOfWinning + result[i]

# Prints the result
print("The exact probability of winning the game is: ", probabilityOfWinning)
```

```
#TASK 3
def montecarlo(trials):
    #The amount of games that should be played
    trialAmount = trials
    #The current amount of games that resulted in a win
    counter = 0
    for i in range(trialAmount):
        #Generates a random result for the dice
        p4 = random.randint(1,4)
        p6 = random.randint(1,6)
        p8 = random.randint(1,8)
        p12 = random.randint(1,12)
        p20 = random.randint(1,20)
        #Calculates the sum of the dice
        sum = p4+p6+p8+p12+p20
```

```
        #Checks if the game has been won. In that case,
        #increases the number of successful games
        if (sum <= 10 or sum >= 45):
            counter = counter + 1

    #Calculates the final probability of winning
    probability = counter/trialAmount
    print("The probability of winning is: ", probability)
    return probability

p1000 = montecarlo(1000)
print("The probability of winning using 1000 trials: ", p1000)
```

```
#TASK 4
#Calculates the probabilities using Monte Carlo simulations,
#using different amounts of trials.
xaxis = [1000, 3000, 10000, 30000, 100000, 300000, 1000000]
mcvalues = []
mcvalues.append(p1000)
mcvalues.append(montecarlo(3000))
mcvalues.append(montecarlo(10000))
mcvalues.append(montecarlo(30000))
mcvalues.append(montecarlo(100000))
mcvalues.append(montecarlo(300000))
mcvalues.append(montecarlo(1000000))

#Displays the result as a figure
mplot.xscale("log")
mplot.plot(xaxis, mcvalues, "-o")
mplot.ylim(ymin=0)
mplot.ylim(ymax=0.02)
mplot.xlim(xmax=1100000)
mplot.xlabel("Amount of trials")
mplot.ylabel("Probability")
mplot.show()
```

```
#TASK 5
#Calculates the difference between the expected value and the
#result of each trial as a relative percentage.
expectedvalue = probabilityOfWinning
difference = []
for i in mcvalues:
    error = abs((i-expectedvalue)/expectedvalue)*100
    difference.append(error)
    print("Relative error: ", error)
#Plots the absolute values of the differences.
mplot.xscale("log")
mplot.plot(xaxis, difference, "-o")
mplot.ylim(ymin=0)
mplot.xlim(xmax=1100000)
```

```
mplot.xlabel("Amount of trials")
mplot.ylabel("Error (absolute value, in percent)")
mplot.show()
```

---

8 References

[1] Platonic solids, Smithsonian Institution. Available at: https://www.si.edu/spotlight/geometric-models/platonic-solids (Accessed Sep. 05, 2024).

[2] IX1501 HT24 Project, Canvas.kth.se, 2024. https://canvas.kth.se/courses/49301/pages/projektuppgift-1 (Accessed Sep. 08, 2024).

[3] "Relative Error Formula." Online Math Learning. Available: https://www.onlinemathlearning.com/relative-error-formula.html. (Accessed: Sep. 7, 2024).

[4] "Probability mass function". Introduction to Probability, Statistics, and Random Processes, Hossein Pishro-Nik. Available: https://probabilitycourse.com/chapter3/3_1_3_pmf.php. (Accessed: Sep. 9, 2024).

[5] " Central limit theorem". Available: https://sphweb.bumc.bu.edu/otlt/MPH-Modules/BS/BS704_Probability/BS704_Probability12.html (Accessed: Sep. 8, 2024).

[6] "Monte Carlo Methods, " Chalmers University of Technology. Available: https://www.math.chalmers.se/Stat/Grundutb/CTH/tms150/1112/MC.pdf (Accessed: Sep. 8, 2024).