Department of Electrical Engineering

California State University, Long Beach

Probability and Statistics with Applications to Computing

Duc Tran

Random Number Experiments

Submitted by

Devin Suy

2020 September 24

Introduction

The purpose of this laboratory was to utilize the Python scripting language along with a few additional numerical libraries to perform four different probability simulations. The simulations conducted were: Rolling two dice until obtaining values that sum to exactly seven, keeping track of the number of rolls taken, discarding trials exceeding sixty rolls; Rolling a six-sided unfair die 10,000 times, corresponding to a set of given probabilities for each value; Performing 100,000 trials of 100 coin flips, logging the amount of trials in which exactly 35 heads were achieved; Lastly, calculating the probability of drawing a “four-of-a-kind” in a six-card poker draw.

Methods

In the first experiment the Python random library was utilized to simulate a dice roll by randomly selecting a value between one and six inclusive for each “die”, the values were summed and checked if equal to seven, moving on to the next trial if the sum of the two dice equal seven or the number of attempts exceeded sixty. The numerical python and matplotlib libraries were utilized to visualize the data through generation of a histogram.

In the following experiment involving an unfair six-sided die, the given probabilities were weighted to represent the number of times we should expect to find that value in a Python list of 100 numbers. For example, the value of 3 on the die corresponds to $P_3 = 0.25$, therefore our list of numbers will be populated with the value 3 a total of 25 times. This process is repeated for each of the corresponding die values with respect to their given probability frequencies. Following this process, the list of 100 numbers is shuffled a total of 10,000 times using the randomization library to ensure a fair distribution of the values throughout the list. To simulate the 10,000 die rolls, we simply use randomization to select a value from the list. Keeping track of the amount of times we “roll” each value in our 10,000 trials, the data is visualized as a stem plot of the value rolled and the number of occurrences of rolling that value.

The next experiment involved performing 100,000 trials of a 100 coin toss, counting the number of trials in which exactly 35 of the 100 tosses yielded a heads. This was simulated by using the Boolean variables *True and False* through random selection, in which a *True* value represents a

"heads" and a *False* a "tails". Simply keep a *num_heads* count for each trial of the experiment and an *exact_count* representing the number of trials yielding a value of *num_heads* = 35.

In the last experiment we calculate the probability of drawing a four-of-a-kind hand in a 6-card poker draw by dividing the number of possible four-of-a-kind hands by the total number of possible hands we can draw. This largely revolves around the number of combinations:

$$C(n,r) = \frac{n!}{r! \times (n-r)!}$$

C is the number of combinations

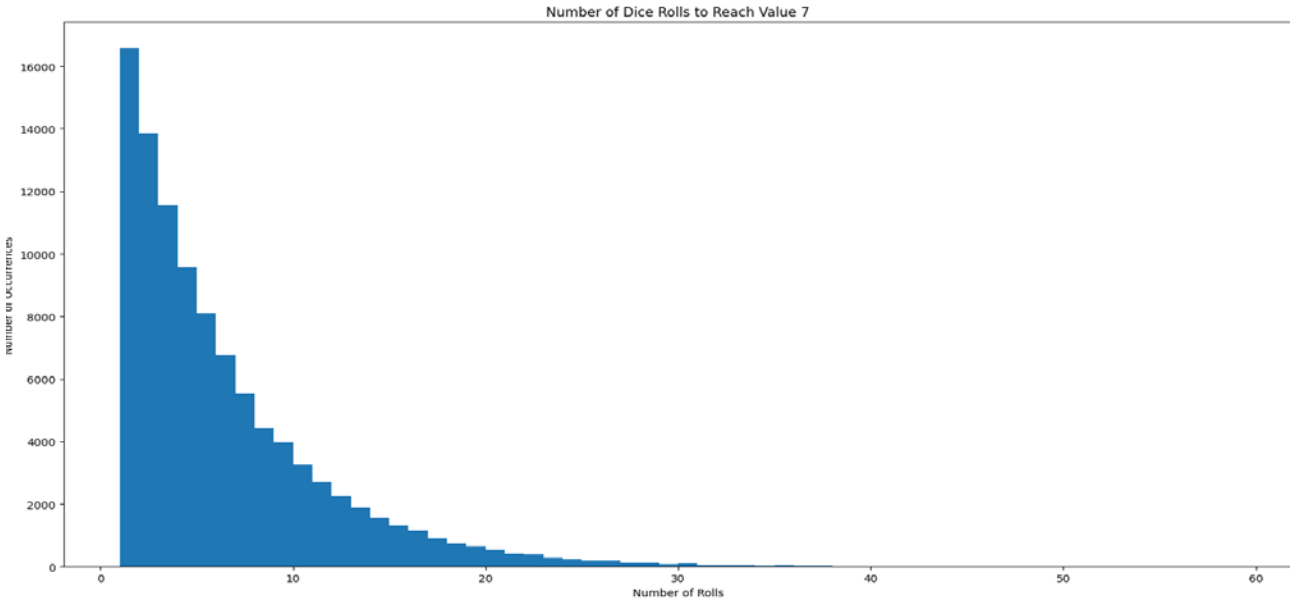
n is the total number of objects in the set

r is the number of choosing objects from the set

The above equation is implemented as a combinations(n, r) function utilizing the factorial function from the Python math library. Using the number of combinations, we can calculate a count of our possible hands and the corresponding probability of interest.

Results and Discussion

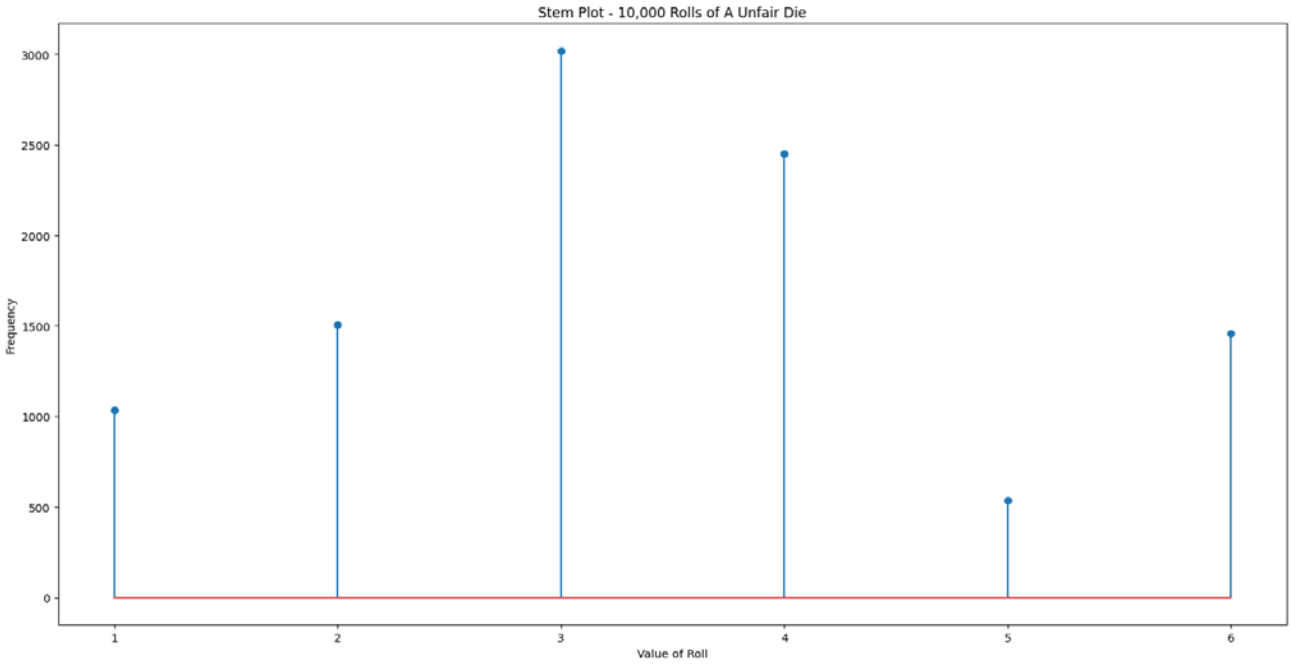
Experiment One:



```
Results
-------
Target value: 7
   Average rolls required: 6.023
   Minimum number of rolls: 1
   Maximum number of rolls: 60
```

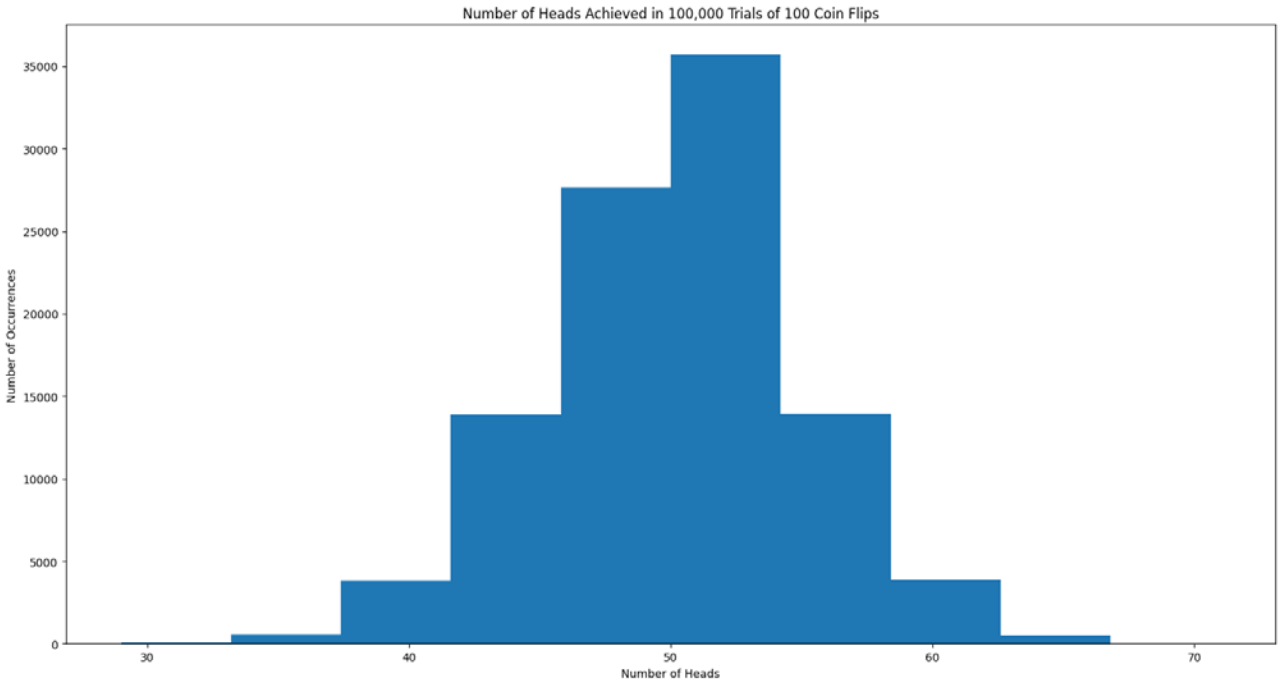
The results of the experiment align with what was expected, with *minimum* number of rolls being one, in which the value of seven was achieved on the first roll. As well as *maximum* number of rolls of sixty, in which seven was not achieved until the very last allowed roll. It was surprising to see the *average* number of rolls on the lower side at around six rolls.

Experiment Two:



Since this second experiment was simulated over 10,000 trials, we see that our data falls exactly in line with the given set of probabilities where $p_5 < p_1 < p_2 \leq p_6 < p_4 < p_3$ for the given values of $[p_1, p_2, p_3, p_4, p_5, p_6] = [0.1, 0.15. 0.3, 0.25, 0.05, 0.15]$ and is reflected in the respective frequency of the values depicted in the stem plot.

Experiment Three:




```
Results
-------
Target Number of Heads: 35
   Average Number of Heads: 50.003
   Number of trials with exactly 35 heads: 92
   Probability of getting exactly 35 heads: 0.0009
```

This results of this third experiment were taken from 100,000 trials of 100 coin flips, with the larger number of trials giving a more accurate depiction when compared to the other experiments. As expected, the *average* number of heads in 100 coin flips is about 50 or half of the time. Interestingly, the histogram shows a higher frequency for obtaining heads just below the average when compared to the drop off in the number of occurrences for heads just above the average. Moreover, the number of trials yielding exactly 35 heads and 65 tails was fairly low with a probability of about 0.09%, or 92 of the 100,000 total trials, a bit lower than expected.

Experiment 4:

```
Results
-------
   Number of possible four-of-a-kind hands: 14664
   Total number of possible hands: 20358520
   Probability of drawing a four-of-a-kind hand: 0.00072
```

From the results of this last experiment, we see that the probability of drawing a four-of-a-kind is even lower than the previous experiment at about 0.072% or 14,664 of the possible 20,358,520 six-card hands that can be drawn. The results make sense since even though there are 13 different cards that may possibly lead to a four-of-a-kind, we are limited to only drawing six cards, four of which must be of the same type of card, explaining the low probability value.

Conclusion

Performing this laboratory was a valuable way in which to apply the core probabilistic concepts we have been studying into practice with actual data. When dealing with some of the implementation logic for the different simulations, it was helpful to follow the same reasoning used when solving probability problems similar to those completed on previous assignments, while taking advantage of the computing features and capabilities writing a Python script has over manual calculations. An example of this is in experiment four in which an nCr function can be implemented in a single line and simply called when needed.

It was also useful to be able to easily modify values and directly see the change in results for the data visualization. At first it was challenging to work with matplotlib and numerical Python simply because I had never utilized them before. However, the provided examples and consulting with available documentation online proved to be useful in becoming familiar with these tools that simplify data analysis and calculations. For the last experiment I was also unfamiliar with six-card poker but doing some basic research into the game first helped me to understand the concepts and derive a method for calculating the probability of interest.

Appendix

Experiment 1 Implementation:

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

def rolls_to_target(N=100000, target_val=7):
    num_rolls_needed = []
    # Perform N trials
    for _ in range(N):
        roll_count = 0
        while True:
            # Perform "roll" of two dice, track the roll count
            roll_val = random.randint(1, 6) + random.randint(1, 6)
            roll_count += 1

            # Number of rolls exceeded, discard
            if roll_count > 60: break

            # Target value reached, save the amount of rolls required
            if roll_val == 7:
                num_rolls_needed.append(roll_count)
                break

    # Calculate and output roll data
    avg_rolls = round(sum(num_rolls_needed) / N, 3)
    min_roll = min(num_rolls_needed)
    max_roll = max(num_rolls_needed)
    print("Results\n-------")
    print("Target value:", target_val)
    print("   Average rolls required:", avg_rolls)
    print("   Minimum number of rolls:", min_roll)
    print("   Maximum number of rolls:", max_roll)

    # Create histogram
    plt.hist(np.asarray(num_rolls_needed), bins=range(1,max_roll))
    plt.title("Number of Dice Rolls to Reach Value 7")
    plt.xlabel("Number of Rolls")
    plt.ylabel("Number of Occurrences")
    plt.show()

rolls_to_target()
```

Experiment 2 Implementation:

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

def unfair_die(N=10000):
    # Maps the die value to the amount of times that it
    # should appear (out of 100) from the given probabilities
    occurences = {
        1 : 10,
        2 : 15,
        3 : 30,
        4 : 25,
        5 : 5,
        6 : 15
    }

    # Populate the die with 100 options proportional to
    # the probability of each die value occurring
    die = []
    for die_val, die_freq in occurences.items():
        # For each die value, it should appear die_freq
        # amount of times out of 100 possible total
        for _ in range(die_freq): die.append(die_val)

    # Randomize our "die" to equally distribute values
    for _ in range(10000): random.shuffle(die)

    # Perform N "rolls" by randomly selecting a value on
    # the die, log the result of each roll
    roll_vals = []
    for _ in range(N): roll_vals.append(die[random.randint(0, 99)])

    # Create stem plot
    hist, bin_edges = np.histogram(np.asarray(roll_vals), bins=range(1,8))
    plt.stem(bin_edges[0:6], hist)
    plt.title("Stem Plot - 10,000 Rolls of A Unfair Die")
    plt.xlabel("Value of Roll")
    plt.ylabel("Frequency")
    plt.show()

unfair_die()
```

Experiment 3 Implementation:

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

def exact_tosses(N=100000, target_freq=35):
    exact_count = 0
    heads_reached = []
    # Perform N experiments in which 100 coin flips are simulated in each
    for _ in range(N):
        num_heads = 0
        # Perform "coin flips" where a value of False represents tails
        for _ in range(100):
            # Let a boolean True value represent "heads" as result of flip
            if random.choice([True, False]): num_heads += 1
        heads_reached.append(num_heads)

        # Trial of experiment yielded exactly the correct number of heads
        # log the count, before advancing to next trial
        if num_heads == target_freq: exact_count += 1

    # Calculate and output coin flip data
    print("Results\n-------")
    print("Target Number of Heads:", target_freq)
    print("   Average Number of Heads:", round(sum(heads_reached) / N, 3))
    print("   Number of trials with exactly", target_freq, "heads:", exact_count)
    print("   Probability of getting exactly", target_freq, "heads:", round(exact
_count/N, 4))

    # Create histogram
    plt.hist(heads_reached)
    plt.title("Number of Heads Achieved in 100,000 Trials of 100 Coin Flips")
    plt.xlabel("Number of Heads")
    plt.ylabel("Number of Occurrences")
    plt.show()

exact_tosses()
```

Experiment 4 Implementation:

```
import random
import math

# Run a nCr calculation
def combinations(n, r):
    return int((math.factorial(n) / math.factorial(r)) / math.factorial(n-r))

def four_kind():
    # The number of possible ways to draw 6 cards
    num_hands = combinations(52, 6)

    # There are 13 cards we could possibly get four of a kind with:
    # 2,3,4,5,6,7,8,9,10,J,Q,K,A
    num_cards = 13

    # We are interested in drawing 4 of a kind but 2 cards will still remain,
    # 52-4 = 48, there are 48C2 ways to draw the remaining 2 cards
    remaining_draws = combinations(48, 2)

    # There are only four of each card in the 52 deck, meaning there is only one
    # possible way to get four-of-a-kind for a given card
    four_kind_hands = 1 * num_cards * remaining_draws
    probability = round(four_kind_hands / num_hands, 6)

    # Output results
    print("Results\n-------")
    print("   Number of possible four-of-a-kind hands:", four_kind_hands)
    print("   Total number of possible hands:", num_hands)
    print("   Probability of drawing a four-of-a-kind hand:", probability)

four_kind()
```