PROJECT 1 – STOCHASTIC EXPERIMENTS

EE 381 – Probability and States Computing

Abstract

This project is an introduction to stochastic experiments and random variables

Brian Nguyen

015188925

1. Function for an n-sided Die

Introduction

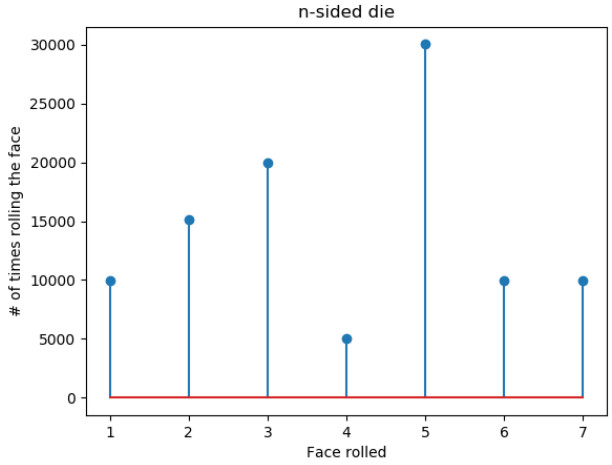
The experiment simulates a single roll of an n-sided die. In this case, we have 7-sided die for this experiment given by *p = [0.10, 0.15, 0.20, 0.05, 0.30, 0.10, 0.10],* which is the probabilities for each side being rolled. All elements inside *p* must add up to "1."

Methodology

In my original function, I took the vector *p* and stored of the length of it into variable *n*; that way, this program will work for all lengths of *p*. I stored the cumulative sum of *p* into *cs* and used that variable to store it into *cp*; in addition, I generated a random number to be used in my for loop. If the random number is greater than the number of the current position in the array and less than or equal to the number in the next position, then I incremented *d* by *k + 1.*

I created a tester function so that the original function is tested *N = 100,000* times.

Results and Conclusions



From the results, the number five is rolled the most. This is because the fifth element from the given vector *p = [0.10, 0.15, 0.20, 0.05, 0.30, 0.10, 0.10]* is 0.30, which is the highest from the rest of the probabilities.

Appendix

```
import matplotlib
import matplotlib.pyplot as plt

def nSidedDie(p):
    # p = [0.10,  0.15,  0.20,  0.05,  0.30, 0.10, 0.10]
    n = len(p)
    cs = np.cumsum(p)
    cp = np.append(0, cs)
    r = np.random.rand()
    for k in range(0, n):
        if r > cp[k] and r <= cp[k + 1]:
            d = k + 1
    return d

def nSidedDieTester(N):
    theList = []
    for x in range(0, N):
        r = nSidedDie([0.10, 0.15, 0.20, 0.05, 0.30, 0.10, 0.10])
        theList.append(r)
    b = range(1, max(theList) + 2)
    sb = np.size(b)
    h1, bin_edges = np.histogram(theList, bins=b)
    b1 = bin_edges[0:sb - 1]
    plt.close('all')

    plt.figure(1)
    plt.stem(b1, h1)
    plt.title('n-sided die')
    plt.xlabel('Face rolled')
    plt.ylabel('# of times rolling the face')
    plt.show()

nSidedDieTester(100000)
```

2. Number of Rolls Needed to Get a “7” With Two Die

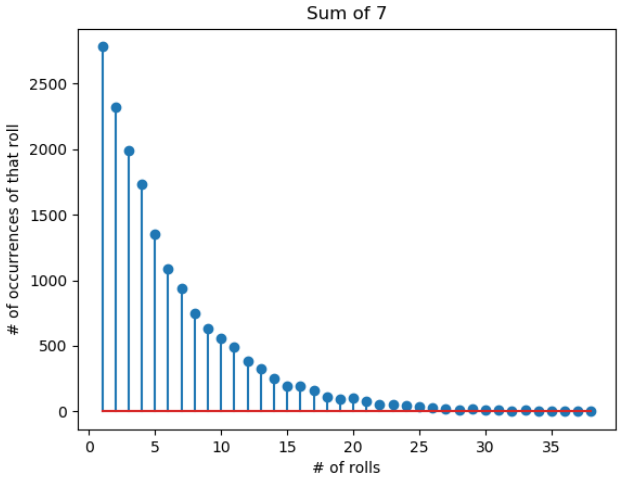
Introduction

The following experiment consists of rolling two die and keeping track of the number of rolls it will take until the summation of the faces is “7.” When the sum is a 7, the experiment is considered a “success” and will stop. The experiment is repeated N = 100,00 times, and the program keeps track of the number of rolls it would take to get a sum of 7 each time.

Methodology

I initially created an empty list which would later store the number of rolls it would take to get a sum of 7. Using a for loop, from 0 to N, I randomly generated two numbers between 1 and 6 and summed those two numbers together. This event counts as a “roll” and I would add the number of rolls by one every time it happens. If the sum of the two dice is “7,” the number of rolls it took to get that will be stored into an array; the number of rolls will reset to “0” and the experiment will start again.

Results and Conclusions



From the results that I have gathered, it appears that the stem plot shows a graph that resembles an exponential decay. The two dice are very likely to have rolled a sum of 7 in the first few tries as opposed to many tries. The worst-case scenario for rolling a sum of 7 would take infinite tries, but that is extremely unlikely from this experiment as the number stops at around 35.

Appendix

```
import numpy as np


import matplotlib
import matplotlib.pyplot as plt

def sumOfSeven(N):
    roll = 0
    list = []
    for x in range(0, N):
        d1 = np.random.randint(1, 7)
        d2 = np.random.randint(1, 7)
        roll += 1
        s = d1 + d2
        # print(d1, "+", d2, "=", sum)
        if s == 7:
            list.append(roll)
            roll = 0
    print(list)
    b = range(1, 40)
    sb = np.size(b)
    h1, bin_edges = np.histogram(list, bins=b)
    b1 = bin_edges[0:sb - 1]
    plt.close('all')

    plt.figure(1)
    plt.stem(b1, h1)
    plt.title('Sum of 7')
    plt.xlabel('# of rolls')
    plt.ylabel('# of occurrences of that roll')
    plt.show()

sumOfSeven(100000)
```

3. Getting 50 Heads When Tossing 100 Coins

Introduction

The following experiment consists of tossing 100 coins and recording the number of "heads." The experiment is considered a "success" when 50 heads are counted out of 100 tosses. The experiment is repeated for N = 100,000 times and the total number of successes are counted.

Methodology

I initialized a "success" variable to keep track of the number of times 50 heads are counted. In a for loop ranging from 0 to N, I flipped a coin 100 times. The head of the coin has the value of "1" and the tail of the coin has the value of "0." To easily count the number of heads, I added the total number of times a coin lands on "heads." If the total number of heads is 50, the "success" variable is incremented by 1. This experiment continues until it reaches N = 100,000.

Results and Conclusion

Probability of 50 heads in tossing 100 coins	
Ans.	$\boldsymbol{p}$ = 0.07955

From the result I gathered, the probability of getting 50 heads in tossing 100 coins is around 0.08, which is very unlikely considering the experiment is repeated for 100,000 times.

Appendix

```
def fiftyHeads(N):
    success = 0
    for k in range(0, N):
        coin = np.random.randint(0, 2, 100)
        heads = sum(coin)
        if heads == 50:
            success += 1
    p_success = success / N
    print('probability of success = ', p_success)

fiftyHeads(100000)
```

4. The Password Hacking Problem

Introduction

In this experiment, our computer system uses a 4-letter password for login. A hacker creates a list of *m* random 4-letter words in order to access our system; the value of *m* is 80,000. If the hacker's list contains at least one word that matches our password, then the experiment is considered a success. There are three parts to this experiment:

- The experiment is repeated for *N* = 1000 times
- The hacker creates a longer list of k**m* letter words and the experiment is repeated for *N* = 1000 times. The value of k is 7.
- The previous experiment is repeated to find the approximate number *m* of words that must be contained in the list so that the probability of finding at least one of the words is *p* = 0.5

Methodology

To create a random lowercase password, we have to consider that there are four slots in our password with 26 different choices in each slot; therefore, we have 26^4 possible outcomes. To make this experiment simple, I generated a number between 0 and 26^4, which represents the numerical value of the password.

For each for loop, I generated a list called `hackers_list` with all of the possible passwords from 0 to *m* (for problem 2, I used k * *m*). I also created a `success` counter that tracks how many times the password appears at least once in the list. When the entire experiment ends, I took the number of tries and divided by *N* to get the probability of success.

Results and Conclusions

Hacker creates *m* words Prob. That at least one of the words matches the password	***p*** = 0.206
Hacker creates k**m* words Probability that at least one of the words matches the password	***p*** = 0.815
p = 0.5 Approximate number of words in the list	***m*** = 240000

From the first part of the experiment, the chances of a password in the list matching our password is relatively low. In the second part, when the list is expanded to k**m* (7 * 80,000), the probability is greatly increased to 0.815, making the likelihood of our password appearing in the list greatly increased.

The third part of the experiment required some trial and error. Considering the second experiment, I tried using numbers below k*m and eventually ended up with m = 240000. The number is the closest approximation I could get.

Appendix

```
import random
import numpy as np

def passwordHacking(m, k, size, N):
    n = 24 ** 4
    password = np.random.randint(0, n)
    print('password = ', password)

    # --------------------
    # list of size m
    print("\n-------\nlist size of m")
    success = 0
    for x in range(0, 1000):
        hackers_list = np.random.randint(0, n, m)
        if password in hackers_list:
            success += 1
    print("# times password is found: ", success)
    print("probability of success for m: ", success / N)

    # --------------------
    # list of size k * m

    print("\n-------\nlist size of k*m")
    success = 0
    for x in range(0, 1000):
        hackers_list = np.random.randint(0, n, k * m)
        if password in hackers_list:
            success += 1
    print("# times password is found: ", success)
    print("probability of success for k*m: ", success / N)

    # ------------
    # list of randomly user-guessed sized

    print("\n-------\nlist size of", size)
    success = 0
    for x in range(0, 1000):
        hackers_list = np.random.randint(0, n, size)
        if password in hackers_list:
            success += 1
    print("# times password is found: ", success)
    print("probability of success for", size, "size:", success / N)

passwordHacking(80000, 7, 240000, 1000)
```

Bibliography

1. Dr. Chassiakos. Project on Random Numbers and Stochastic Experiments