



California State University Long Beach

EE 381 – Probability and Statistic Computing

PROJECT 01

RANDOM NUMBER EXPERIMENTS

Professor: Duc Tran

Team members: Alyssa Faiferlick - Thanh Nguyen

Class: EE 381 – Section 13

Due date: 09/24/2020

I. Question 1: Sum of “7” when rolling two dice

1. Introduction:

The purpose of this function is generating a probability histogram of the number of rolls required of two dice before a sum of “7” appears. The probabilities are calculated after 1000000 experiments and the limit of number of rolls is 60.

2. Procedure:

- First, we create an empty list which stores the number of needed rolls before getting sum of 7 for each experiment.
- Then we use a for loop from 0 to N (N = 1000000: the number of experiments). For each experiment:
 - + Step 1: Initialize the counter and set the counter equal to 0.
 - + Step 2: Generate randomly two numbers between 1 to 6 which represent the number on two dice when tossing.
 - + Step 3: Sum these two numbers together.
 - + Step 4: If the sum of two dice equal to 7 then save the counter, which is the number of needed rolls to get sum of 7, into the list and reset the counter and repeat step 1. If the sum of two dice unequal to 7 then increase the counter by 1 and repeat step 2.

3. Results and Discussion:

- Results:

Figure 1

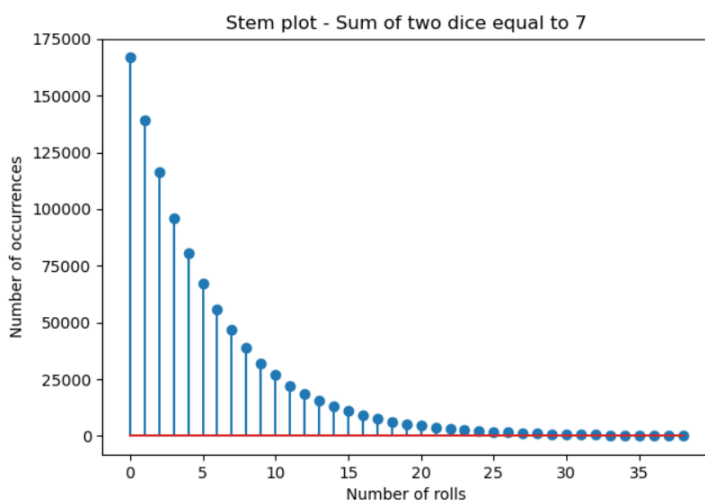
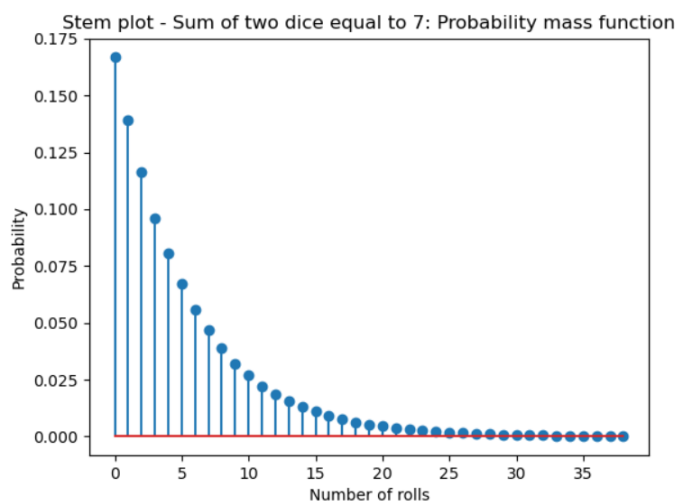


Figure 2



- The results are what we expected them to be. They're close to the probabilities that we calculate when rolling two dice to have sum of 7. Example, belong to the result we had when running our program, to have sum of 7 in the first roll, we had the probability equal to 0.167013, which is close to $\frac{1}{6} \approx 0.1667$.

4. Conclusion:

In this part of the lab we used aspects from the lecture such as the concept of probability and independent events to find out. Using these we found the probability of how many rolls is required of two dice before a sum of “7” appears and applied it to our code. Some challenges we overcame were small errors in the code and understanding and using the “histogram”.

5. Appendix:

```
1. """
2.     Class: EE 381 - Section 13
3.     Project 01 - Random Number Experiments _Part 01
4. """
5. import random
6. import matplotlib.pyplot as plt
7. import numpy as np
8.
9. #Function
10. """
11.     Number of rolls that needs to get a sum of 7 appears when rolling 2 dice
12.
13.     Input: N (The number of Experiments)
14.     Output: None
15. """
16. def sumOf2Dice(N):
17.     result = []
18.     for i in range(0, N): #for each experiment
19.         count = 0
20.         total = 0
21.         while (total != 7): #roll the 2 dice until get the sum of 7
22.             count += 1
23.             dice1 = random.randint(1, 6)
24.             dice2 = random.randint(1, 6)
25.             total = dice1 + dice2
26.         result.append(count - 1) #count - 1 because don't include the total of 7
27.     #Stem plot graphing
28.     b = range(0, 40)
29.     sb = np.size(b)
30.     h1, bin_edges = np.histogram(result, bins = b)
31.     b1 = bin_edges[0: sb - 1]
32.     plt.close('all')
33.     print(h1[0])
34.
35.     #Generate plot
```

```
36.     #First figure
37.     fig1 = plt.figure(1)
38.     plt.stem(b1, h1)
39.     plt.title('Stem plot - Sum of two dice equal to 7')
40.     plt.xlabel('Number of rolls')
41.     plt.ylabel('Number of occurrences')
42.     fig1.show()
43.     fig1.savefig('Lab01-1_Sum7OfTwoDice.jpg')
44.
45.     #Second figure
46.     fig2 = plt.figure(2)
47.     p1 = h1 / N
48.     print(p1[0])
49.     plt.stem(b1, p1)
50.     plt.title('Stem plot - Sum of two dice equal to 7: Probability mass function')
51.     plt.xlabel('Number of rolls')
52.     plt.ylabel('Probability')
53.     fig2.show()
54.     fig2.savefig('Lab01-1_PMF of Sum7OfTwoDice.jpg')
55.
56. #Main
57. if __name__ == "__main__":
58.     sumOf2Dice(1000000)
```

II. Question 2:

1. Introduction:

Write a program that generates an unfair six-sided die. The die has six sides [1, 2, 3, 4, 5, 6] with probabilities: $[p_1, p_2, p_3, p_4, p_5, p_6] = [0.1, 0.15, 0.3, 0.25, 0.05, 0.15]$. Simulating the roll of the die 10,000 times, and plot the PMF of your unfair die as a stem plot. The stem plot should verify that the six sides of your unfair die follow the required probabilities.

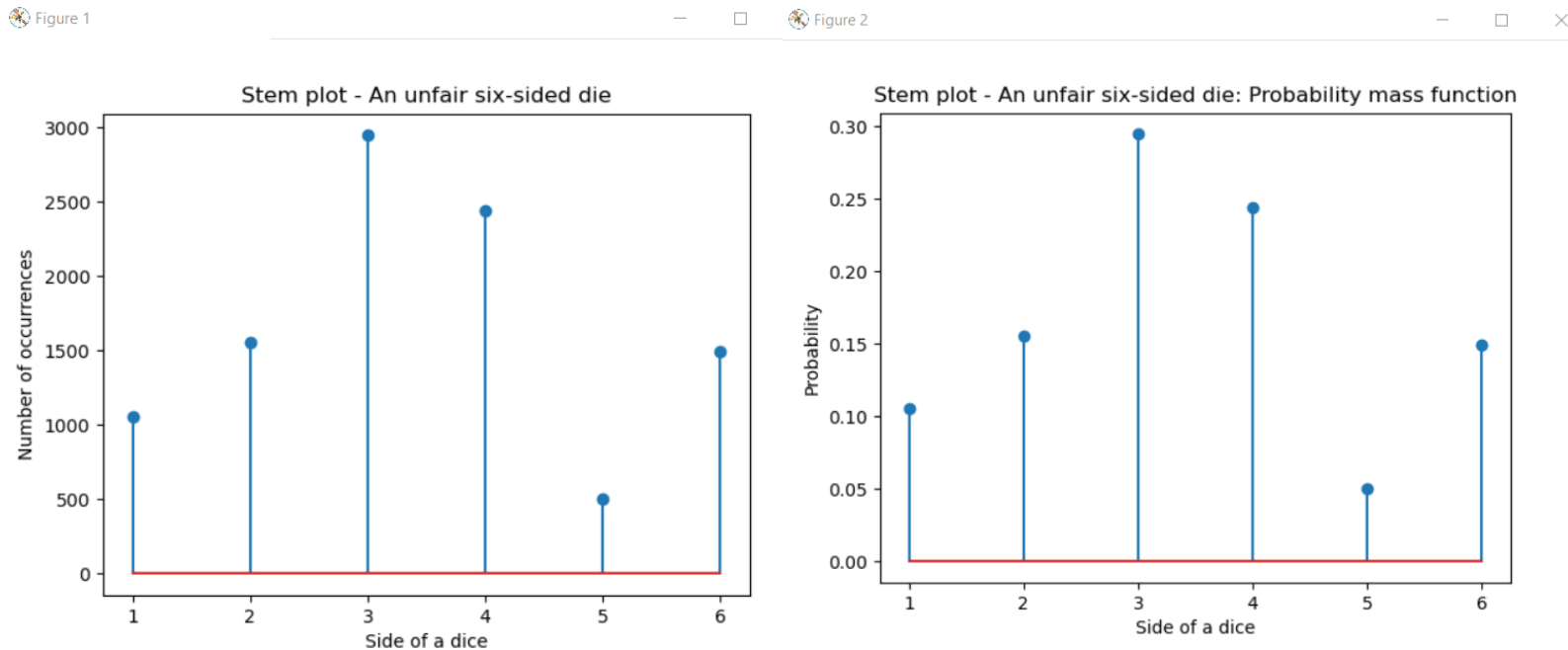
2. Procedure:

- Using the probabilities of rolling the sides of die given to us we can create a probability function. Then using that and graphing it, we can then make a cumulative probability function (CDF). This tells us that in between what specific thresholds what side of the dice we will roll.
- Python Code: Generate a random float between 0.00 and 1.00
- Using the CDF thresholds create an “if, elif, else” statement to find out where the random float number categorizes, which then tells us which side of the dice was rolled.
- Create a loop to roll the dice 10,000 times. Every time the dice is rolled, add a count to the side that shows up.
- Divide the each sides number’s total amount rolled, and then divide it by 10,000 to receive the probability.

- Then using the 'histogram' feature to create a graph with the x-axis representing the 6 sides of the dice, and the y-axis representing the range of probabilities.

3. Results and Discussion:

- Result:



- The results are what we expected them to be. If you output each side's number of total times it showed up and divided it by 10,000, you would receive numbers extremely close to the given probabilities. For example, the probability given for rolling a "1" was 0.1, our code result was 0.097. The graph above shows each side's probability that we calculated in our program.

4. Conclusion:

In this part of the lab we used aspects from our lecture such as random variables, discrete probability distributions and distribution functions for random variables. Being a beginner python coder, this part of the lab helped teach classes in python and how to debug the code when errors occurred.

5. Appendix:

```
1. """  
2.     Class: EE 381 - Section 13
```

```
3. Project 01 - Random Number Experiments _Part 02
4. """
5. import random
6. import matplotlib.pyplot as plt
7. import numpy as np
8.
9. #Function
10. """
11. Simulate the roll of six sides of an unfair die follow the required
    probabilities
12.
13. Input:
14.     p = [0.1, 0.15, 0.3, 0.25, 0.05, 0.15] (The required probabilities for each
    side)
15.     N (The number of Experiments)
16. Output: None
17. """
18. def rollAnUnfairDie(p, N):
19.     #Randomize result for each experiment
20.     result = []
21.     sides = len(p) # the number of sides of the die
22.     cs = np.cumsum(p) # the cumulative sum of the elements along a given axis
23.     cp = np.append(0, cs) # cp = [0 0.1 0.25 0.55 0.8 0.85 1]
24.
25.     for i in range(0, N): #for each experience
26.         # generate the random number between 0 and 1
27.         r = np.random.rand()
28.         # check which range r is falling in [0, 0.1), or [0.1, 0.25),...
29.         for j in range(0, sides):
30.             if (r >= cp[j] and r < cp[j+1]):
31.                 # get the number on the right (the bigger one)
32.                 temp = j + 1
33.                 # add the result into the list
34.                 result.append(temp)
35.
36.     #Stem plot graphing
37.     b = range(1, max(result) + 2)
38.     sb = np.size(b)
39.     h, bin_edges = np.histogram(result, bins = b)
40.     b1 = bin_edges[0: sb - 1]
41.     plt.close('all')
42.
43.     #Generate plot
44.     #First figure
45.     fig1 = plt.figure(1)
46.     plt.stem(b1, h)
47.     plt.title('Stem plot - An unfair six-sided die')
48.     plt.xlabel('Side of a dice')
49.     plt.ylabel('Number of occurrences')
50.     fig1.show()
51.     fig1.savefig('Lab01-2_UnfairSixSidedDice.jpg')
52.
53.     #Second figure
54.     fig2 = plt.figure(2)
55.     p = h / N
56.     plt.stem(b1, p)
57.     plt.title('Stem plot - An unfair six-sided die: Probability mass function')
58.     plt.xlabel('Side of a dice')
59.     plt.ylabel('Probability')
```

```
60.     fig2.show()
61.     fig2.savefig('Lab01-1_PMF of UnfairSixSidedDice.jpg')
62.
63.
64. #Main
65. if __name__ == "__main__":
66.     rollAnUnfairDie([0.1, 0.15, 0.3, 0.25, 0.05, 0.15], 10000)
```

III. Question 3: Tossing coins

1. Introduction:

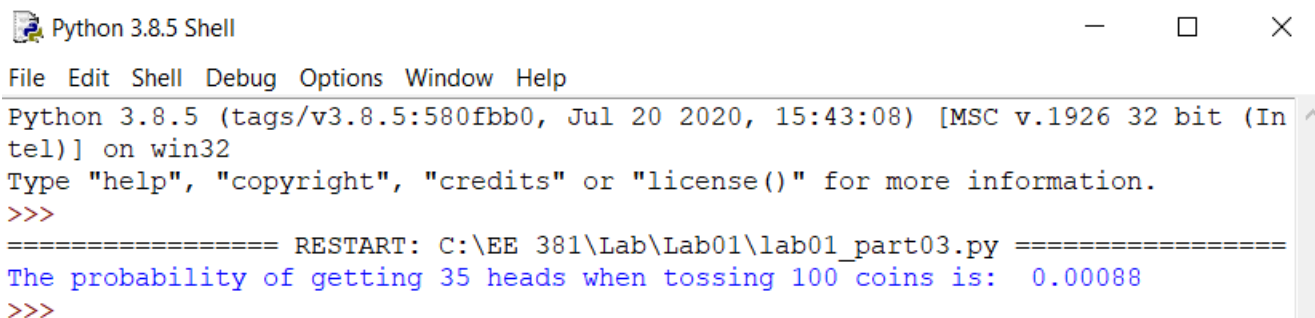
Write the program to calculate the probability of getting exactly 35 heads when tossing 100 coins. This probability is calculated after 100000 experiments.

2. Procedure:

- First, we generate the counter and set it equal to 0.
- Then we use a for loop from 0 to N (N = 100000: the number of experiments). For each experiment:
 - + Create an array that stores 100 coins with 2 values: 0 is tail and 1 is head.
 - + Calculate the sum of the array. Because we considered that 1 is heads, the sum of the array is the number of heads.
 - + Check the sum with 35. If sum equal to 35, it meets the requirement then increase the counter by 1.
- After 100000 loops, we got the counter is the number of experiments which meet the requirement. The probability we need to calculate equal to the counter divide to 100000 (N)

3. Results and Discussion:

- Result:



```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\EE 381\Lab\Lab01\lab01_part03.py =====
The probability of getting 35 heads when tossing 100 coins is:  0.00088
>>>
```

- The probability of getting 35 heads when tossing 100 coins equal to $\frac{C(100,35)}{2^{100}} \approx 8.638 \times 10^{-4}$. Our result quite close to this result, so it is what we expected.

4. Conclusion:

For this question, some aspects we used from lectures include probability and combinations. We found that no problems or errors occurred when writing this program.

5. Appendix:

```
1. """
2.     Class: EE 381 - Section 13
3.     Project 01 - Random Number Experiments _Part 03
4. """
5. import numpy as np
6.
7. #Function
8. """
9.     Find the probability of getting 35 heads when tossing 100 coins
10.
11.     Input: N (The number of experiments), numberOfCoins = 100, numberOfHeads = 35
12.     Output: The probability
13. """
14. def tossTheCoins(N, numberOfCoins, numberOfHeads):
15.     count = 0 # count the number of experiments which have 35 heads
16.     for i in range(0, N): #for each experience
17.         #create an array that contains the results after tossing 100 coins
18.         sideOfCoins = np.random.randint(0, 2, numberOfCoins) #0 is tail and 1 is
head
19.         #count the number of heads
20.         heads = sum(sideOfCoins)
21.
22.         #if the number of heads equals to 35 => increase count
23.         if (heads == numberOfHeads):
24.             count += 1
25.
26.     print('The probability of getting', numberOfHeads, 'heads when tossing',
numberOfCoins, 'coins is: ', count / N)
27.
28. #Main
29. if __name__ == "__main__":
30.     tossTheCoins(100000, 100, 35)
```

IV. Question 4: “4 of a kind”

1. Introduction:

A deck of cards contains 52 cards. They are divided into four suits: spades, diamonds, clubs and hearts. Each suit has 13 cards: ace through 10, and three picture cards: Jack, Queen, and King. A '4 of a kind' or a

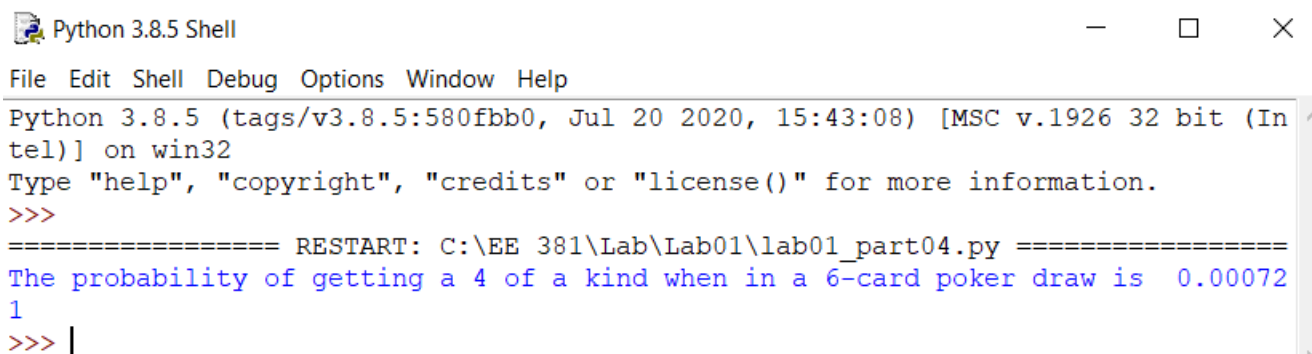
'quad' is a hand that contains four cards of one rank, such as 9♣ 9♠ 9♦ 9♥ ('four of a kind, nines'). The program is written to determine the probability of "4 of a kind" in a 6-card poker draw. This probability is calculated after 100000 experiments.

2. Procedure:

- First, we generate the list that contains 52 cards in the deck.
- Next, we create the counter and set it equal to 0. We will use it to count the experiment that meets the requirement.
- Then we use a for loop from 0 to N (N = 10000000: the number of experiments). For each experiment:
 - + Step 1: Generate randomly a list that stores 6 random cards from the deck.
 - + Step 2: Sort the list we got. If the list contains 4 cards of a kind, these cards will have the index from 0 to 3, or 1 to 4, or 2 to 5.
 - + Step 3: Check that if the card at 0 and 3, or 1 and 4, or 2 and 5 have the same rank then increase the counter by 1.
 - + Step 4: Repeat the loop.
- After 10000000 loops, we got the counter is the number of experiments which meet the requirement. The probability we need to calculate is equal to the counter divide to 10000000 (N)

3. Results and Discussion:

- Result:



```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\EE 381\Lab\Lab01\lab01_part04.py =====
The probability of getting a 4 of a kind when in a 6-card poker draw is  0.000721
>>> |
```

- The result is what we expected them to be because the probability of getting a 4 of one kind when drawing 6 cards equal to $\frac{C(13,1) \times C(4,4) \times C(48,2)}{C(52,6)} \approx 7.2 \times 10^{-4}$ and our result quite close to it.

4. Conclusion:

For this part of the lab, aspects we used from the lectures include probability, combinations and the product rule. Some challenges that occurred were struggling to write efficient code and errors when comparing the cards.

5. Appendix:

```
1. """
2.     Class: EE 381 - Section 13
3.     Project 01 - Random Number Experiments _Part 04
4. """
5. import random
6.
7. """
8.     Problem:
9.     Determine the probability of "4 of a kind" in a 6-card poker draw.
10.
11.
12.     Background:
13.     A deck of cards contains 52 cards. They are divided into four suits:
14.     spades, diamonds, clubs and hearts. Each suit has 13 cards: ace through 10,
15.     and three picture cards: Jack, Queen, and King.
16.
17.     A '4 of a kind' or a 'quad' is a hand that contains four cards of one rank,
18.     such as 9♣ 9♠ 9♦ 9♥ ('four of a kind, nines').
19.
20.
21.     Solution:
22. """
23.
24. DeckOfCards = ['2♣', '3♣', '4♣', '5♣', '6♣', '7♣', '8♣', '9♣', '10♣', 'J♣', 'Q♣',
25.                'K♣', 'A♣',
26.                '2♦', '3♦', '4♦', '5♦', '6♦', '7♦', '8♦', '9♦', '10♦', 'J♦', 'Q♦',
27.                'K♦', 'A♦',
28.                '2♥', '3♥', '4♥', '5♥', '6♥', '7♥', '8♥', '9♥', '10♥', 'J♥', 'Q♥',
29.                'K♥', 'A♥',
30.                '2♠', '3♠', '4♠', '5♠', '6♠', '7♠', '8♠', '9♠', '10♠', 'J♠', 'Q♠',
31.                'K♠', 'A♠']
32.
33. def fourOfAKind(N):
34.     count = 0
35.     for i in range(0, N):
36.         MyDraw = random.sample(DeckOfCards, 6)
37.         MyDraw.sort()
38.
39.         if (MyDraw[0][0] == MyDraw[3][0]) or (MyDraw[2][0] == MyDraw[5][0]) or
40.           (MyDraw[1][0] == MyDraw[4][0]):
41.             count += 1
42.
43.     print("The probability of getting a 4 of a kind when in a 6-card poker draw is
44.           ", count/N)
45.
46. fourOfAKind(10000000)
```