Ajay Kc
013213328
Project 1 Report

1. Number of rolls needed to get a “7” with two dice

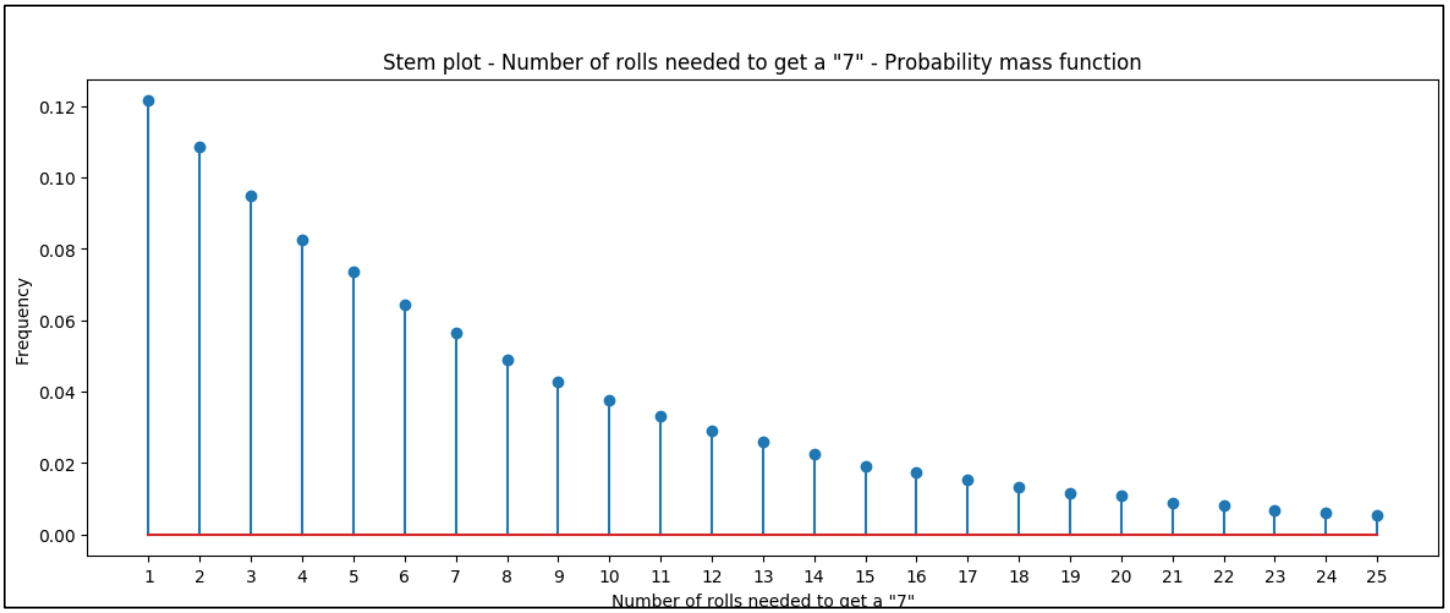
Introduction

The problem is to find the number of rolls needed to get a “7” with two dice. The problem is run 100,000 times to calculate the probability of getting various number as the number of rolls needed to get a “7”

Methodology

Each random number from 2 to 12 are sum of numbers in the dice. The program is run 100,000 times and the dices are rolled until the sum of two random number is 7. Once the sum is 7, the frequency is noted.

PMF Plot



Source code

```
"""
Ajay Kc
013213328
EE 381
Project 1 Part 1

The program counts the number of times it takes to roll a 7 with two dices and
displays the probability mass function
"""

import random as rand
import numpy as np
```

```
import matplotlib.pyplot as plt


def roll_Seven():

    count = 1

    while(True):
        die_One = rand.randint(1,7)
        die_Two = rand.randint(1,7)
        sum_Dice = die_One + die_Two

        if(sum_Dice==7):
            break
        else:
            count +=1

    return count

def make_Figure(list):
    b = range(1,27)
    h1, bin_edges = np.histogram(list, bins=b)
    b1 = bin_edges[0:25]

    figure_One = plt.figure(1)
    p1 = h1/100000
    sum = p1.sum()
    plt.xticks(b1)
    plt.stem(b1,p1)
    plt.title('Stem plot - Number of rolls needed to get a "7" - Probability mass
function')
    plt.xlabel('Number of rolls needed to get a "7"')
    plt.ylabel('Frequency')

    plt.show()


list = []
for i in range(0,100000):
    number = roll_Seven()
    if(number <=25):
        list.append(number)

make_Figure(list)
```

The stem plot of the probability mass function demonstrates that the chances of getting a 7 when two dices are rolled is high within the first few trials. As the probability of getting a seven(1/6) is much higher that any other number, it is likely to get a seven within few trials. The higher the trial number, the lower the probability of getting 7 for the first time. However, there were cases when number of rolls needed to get a 7 where more than 25 but they probability of such an event occurring was really low so we omitted those numbers for convenience of the graph.

2. Getting 50 heads when tossing 100 coins

Introduction

The problem checks the probability of getting exactly 50 heads when 100 coins are tossed.

Methodology

A random number between 0 and 1 is generated where 0 represents the head and 1 represents the tail 100 times. The number of 0 in the list is counted and if the count is equal to 50 then it would be considered a success. The program is run 100,000 times and number of success is counted and the probability is calculated.

Source code

```
"""
Ajay Kc
013213328
EE 381
Project 1 Part 2

The program checks the number of times it takes to get 50 heads when 100 coins are
tossed
"""

import numpy as np

def toss_Coin():

    coin_List = np.random.randint(0,2,100)

    count_Head = sum(coin_List)
    if(count_Head==50):
        return "Success"
    else:
        return "Fail"

count_Success = 0

for i in range(0,100000):
    if (toss_Coin()=="Success"):
        count_Success +=1

probability = count_Success/100000
print ("The probability of getting exactly 50 heads is  %s"  %(probability))
```

Probability of 50 heads in tossing 100 fair coins	
Ans	p = 0.07885

The probability of getting a head is 0.5. However, the probability of getting exactly 50 heads when 100 coins are tossed is lower that 0.5. When the process of tossing 100 coins is repeated 100,000 times, the probability of getting exactly 50 head was 0.07885.

3. Getting 4-of-a-kind

Introduction

The problem is about getting 4 of the same kind of card from a deck of card.

Methodology

The program permutes a random number ranging from 1 to 52. Only the top 5 permutation is selected and each of those numbers is modulo divided by 13 to check if the different numbers that represent card are of the same kind. If there are 4 numbers that are same after modulo division, then the cards are of a same kind and the instance is considered a success. The program is run 100,000 times and the number of success is counted and the probability of getting 4-of-a-kind is calculated.

Source code

```
"""
Ajay Kc
013213328
EE 381
Project 1 Part 3

The program checks the probability of getting 4 of a kind card
"""

import numpy as np

def fourOfAKind():

    #drawnCards = np.random.randint(1,53,5)
    drawnCards = np.random.permutation(52)[0:5]
    cardFaces = {}
    for i in drawnCards:
        if (i % 13 in cardFaces):
            cardFaces[i % 13] += 1
        else:
            cardFaces[i % 13] = 1

    for k, v in cardFaces.items():
        if v == 4:
            return 'Success'

    return 'Failure'


successCount = 0
for i in range(0, 100000):
    if fourOfAKind() == 'Success':
        successCount += 1

probability = successCount / 100000
```

```
print('The probability of getting a 4-of-a-kind when 5 cards are drawn %s' %
(probability))
```

Probability of 4-of-a-kind	
Ans	**p = 0.0022**

The program randomly permutes 52 numbers which is similar to having a deck shuffled. From the shuffled deck, 5 cards are chosen which is done in our program by slicing the first 5 numbers from the list generated from permutation. The event is repeated 100,000 times and the probability of getting 4-of-a-kind is 0.0022.

4. The Password Hacking Problem

Introduction

The problem is about checking the probability of a 4 lettered password being hacked if the hacker has lists containing different number of words.

Methodology

The program generates a random word where each letter is generated randomly. The letter could range from lowercase "a" to lowercase "b". Then, a list containing 10^5 words is generated. The password is matched with each word in that list and checked if they are the same. If the word and the password is same, it's considered a success. The program is repeated 1000 times and the number of success is counted and the probability is calculated. Secondly, the program is repeated with a list containing10^6 words. Finally, by using brute force, the program checks how many words are needed to have an approximate probability of 0.5

Source code

```
"""
Ajay Kc
013213328
EE 381
Project 1 Part 3

The program checks the probability a password getting hacked
"""
import random as rand
import numpy as np
import math


def generatePassword():
    password = ""
```

```
    for i in range(0, 4):
        password = password + chr(rand.randint(97, 122))

    return password


def checkMatch(listLength):
    n = (math.pow(26, 4)) - 1
    count = 0
    for i in range(0, 1000):
        myPassword = rand.randint(0, n)
        hackerPasswordList = np.random.randint(0, n, listLength)

        if (myPassword in hackerPasswordList):
            count += 1

    return count


successOneCount = checkMatch(100000)
successTwoCount = checkMatch(1000000)
print("The probability of matching the password with hacker's list of 10^5 words %s"
%(successOneCount / 1000))
print("The probability of matching the password with hacker's list of 10^6 words %s"
%(successTwoCount / 1000))

print(checkMatch(340000)/1000)
```

$m=10^5$ Probability that at least one of the words matches the password	**p = 0.211**
$m=10^6$ Probability that at least one of the words matches the password	**p = 0.889**
p=0.5 Approximate number of words in the list	**m=340000**

The program demonstrates that the probability of hacker hacking a password would be 0.211 if he/she has a list of 10^5 words. Moreover, the probability of hacking the password increases dramatically to 0.889 when the list is 10^6 words. Secondly, in order for the probability to hack the password to be approximately 0.5, 340,000 words are needed in the list.