CALIFORNIA STATE UNIVERSITY, LONG BEACH

EE 381 – Probability and Statistics with Applications to Computing

Laboratory Projects

Project on

Random Numbers and Stochastic Experiments

---

0. Introduction and Background Material

0.1. Simulating Coin Toss Experiments

As mentioned in class, there are many ways to model stochastic experiments. The following two programs simulate the toss of a fair coin N times, and calculate the experimental probability of getting heads (`p_heads`) or tails (`p_heads`). Both programs provide the same results, but they differ in the way the models are coded.

- The first model is programmed in Python using "for loops".
- The second model makes use of the arrays, and it is computationally very efficient.

MODEL 1

```
import numpy as np
def coin():
        coin=np.random.randint(0,2)
        return coin

def CoinToss(N):
    heads, tails = 0, 0
    for k in range(0,N):
        toss=coin()
        if toss==1:
            heads=heads+1
        else:
            tails=tails+1
    #
    p_heads=heads/N
    p_tails=tails/N
    print('probability of heads = ', p_heads)
    print('probability of tails = ', p_tails)
```

MODEL 2 – MORE EFFICIENT CODE

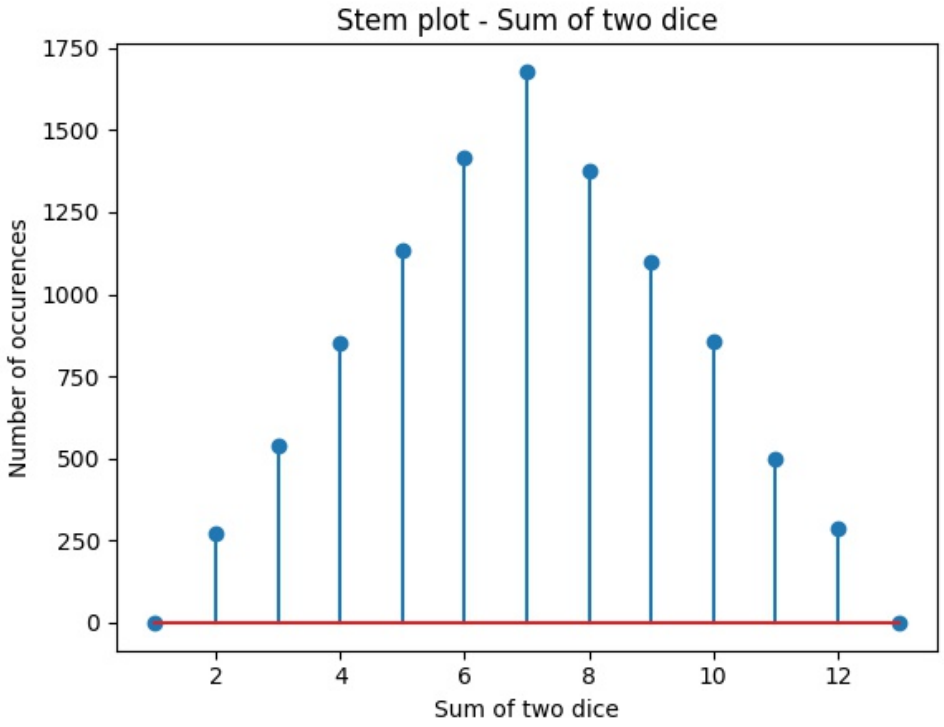
```
import numpy as np
def MultCoinToss(N):
    coin=np.random.randint(0,2,N)
    heads=sum(coin)
    tails=N-heads
    #
    p_heads=heads/N
    p_tails=tails/N
    print('probability of heads = ', p_heads)
    print('probability of tails = ', p_tails)
```

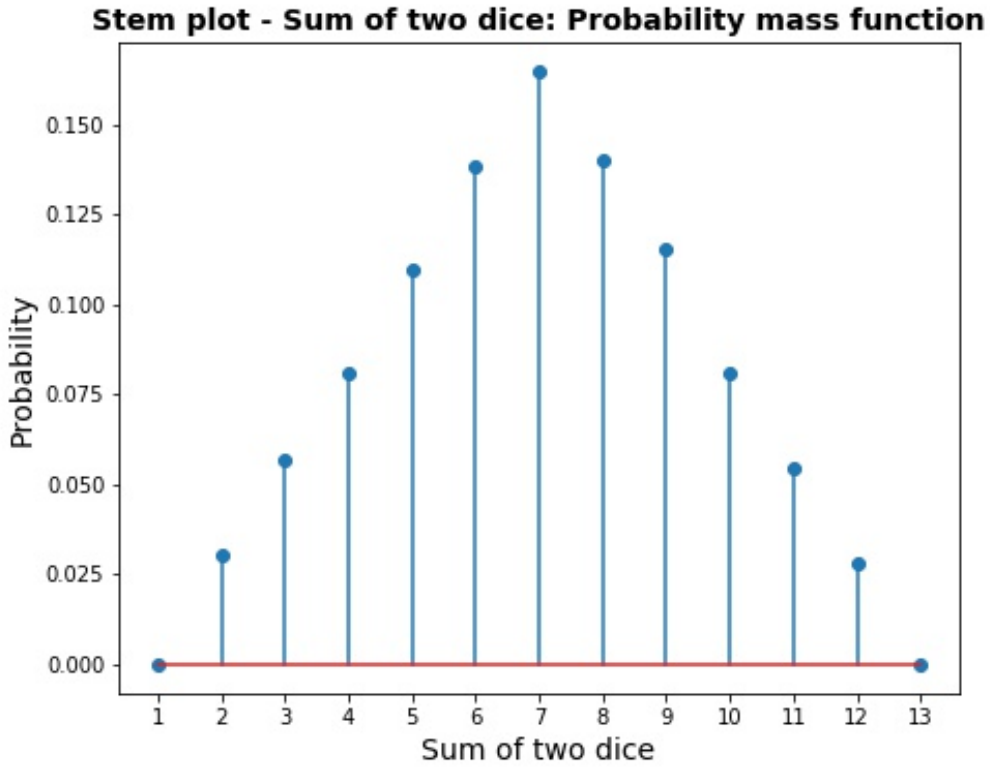
0.2. Roll of Two Fair Dice; Probability Mass Function (PMF)

This experiment models the roll of a pair of dice for N times. The sum each roll is recorded, and stored in vector "s". The probability of each possible outcome is calculated and plotted in a "Probability Mass Function" (PMF) plot
To create the plots, the simulation has been run for N=100000 times.

SUM OF THE ROLLS OF TWO FAIR DICE

```
import numpy as np
import matplotlib.pyplot as plt
#
def sum2dice(N):
    d1=np.random.randint(1,7,N)
    d2= np.random.randint(1,7,N)
    s=d1+d2
    b=range(1,15) ; sb=np.size(b)
#
    h1, bin_edges = np.histogram(s,bins=b)
    b1=bin_edges[0:sb-1]
#
    plt.close('all')
    fig1=plt.figure(1)
    plt.stem(b1,h1)
    plt.title('Stem plot - Sum of two dice')
    plt.xlabel('Sum of two dice')
    plt.ylabel('Number of occurrences')
    fig1.savefig
    #
    fig2=plt.figure(2)
    p1=h1/N
    plt.stem(b1,p1)
    plt.title('Stem plot - Sum of two dice: Probability mass function',
              fontsize=14, fontweight='bold')
    plt.xlabel('Sum of two dice', fontsize=14)
    plt.ylabel('Probability', fontsize=14)
    plt.xticks(b1)
    fig2.savefig
```


Stem plot - Sum of two dice
Number of occurences
Sum of two dice
1750
1500
1250
1000
750
500
250
0
2
4
6
8
10
12


Stem plot - Sum of two dice: Probability mass function
Probability
Sum of two dice
0.150
0.125
0.100
0.075
0.050
0.025
0.000
1
2
3
4
5
6
7
8
9
10
11
12
13

0.3. Generating an unfair three-sided die

This example models the roll of a 3-sided die, with non-uniform probabilities. The die has three sides $[1,2,3]$ with probabilities: $[p_1, p_2, p_3] = [0.3,\ 0.6,\ 0.1]$.

The experiment simulates the roll of the die for N=10,000 times, and the outcome of the N rolls is plotted as a stem plot. The stem plot verifies that the three sides of the die follow the required probabilities.

The following code will simulate **a single roll** of the three-sided die. The variable "d" represents the number after the roll.

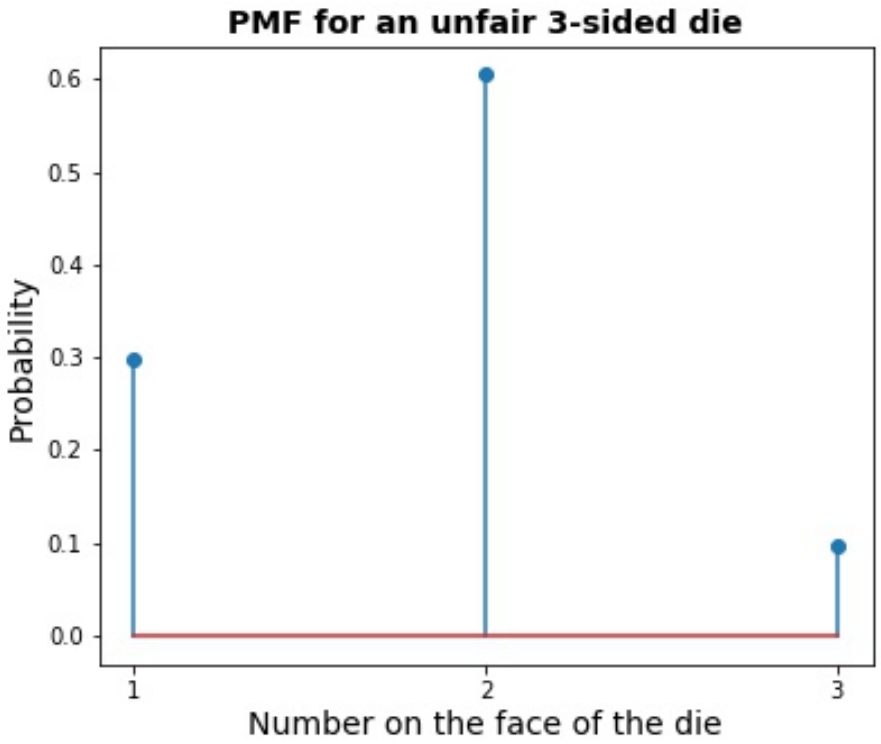
```
import numpy as np
n=3
p=np.array([0.3, 0.6, 0.1])
cs=np.cumsum(p)
cp=np.append(0,cs)
r=np.random.rand()
for k in range(0,n):
    if r>cp[k] and r<=cp[k+1]:
        d=k+1
```

The following code will simulate the rolling of the three-sided die for N=10,000 times and will plot the outcome as a stem plot.

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

    N=10000
    s=np.zeros((N,1))
    n=3
    p=np.array([0.3, 0.6, 0.1])
    #
    cs=np.cumsum(p)
    cp=np.append(0,cs)
    #
    for j in range(0,N):
        r=np.random.rand()
        for k in range(0,n):
            if r>cp[k] and r<=cp[k+1]:
                d=k+1
        s[j]=d
#
```

```
# Plotting
    b=range(1,n+2)
    sb=np.size(b)
    h1, bin_edges=np.histogram(s, bins=b)
    b1=bin_edges[0:sb-1]
    plt.close('all')
    prob=h1/N
#
# Plots and labels
    plt.stem(b1,prob)
    plt.title('PMF for an unfair 3-sided die',fontsize=14,
              fontweight='bold')
    plt.xlabel('Number on the face of the die',fontsize=14)
    plt.ylabel('Probability',fontsize=14,)
    plt.xticks(b1)
```



1. Function for a n-sided die

Write a function that simulates a single roll of a n-sided die. The inputs and outputs of the function are:

Inputs:

- The probabilities for each side, given as a vector $p = [p_1, p_2, \cdots p_n]$

Outputs:

- The number on the face of the die after a single roll, i.e. one number from the set of integers $\{1, 2, \cdots n\}$

Note: The sum $p_1 + p_2 + \cdots p_n$ must be equal to 1.0, otherwise the probability values are incorrect.

Save the function as: `nSidedDie(p)`

Test the function. Use the probability vector $p = [p_1, p_2, \cdots p_n]$ which *has been given to you in a separate document*. To create a random number with a single roll of the die you must use the following command: `r=nSidedDie(p)`

To validate your function, roll the die for N=10,000 times and plot the outcome as a stem plot.

SUBMIT a report that follows the guidelines as described in the syllabus.

- The section on RESULTS must include The PMF in the form of a **stem plot**
- The code must be provided in the appendix

2. Number of rolls needed to get a "7" with two dice

Consider the following experiment:

- You roll a pair of fair dice and calculate the sum of the faces. You are interested in the number of rolls it takes until you get a sum of "7". The first time you get a "7" the experiment is considered a "success". You record the number of rolls and you stop the experiment.
- You repeat the experiment N=100,000 times. Each time you keep track of the number of rolls it takes to have "success".

SUBMIT a report that follows the guidelines as described in the syllabus.

- The section on RESULTS must include The PMF in the form of a **stem plot**
- The code must be provided in the appendix

3. Getting 50 heads when tossing 100 coins

Consider the following experiment:

- You toss 100 fair coins and record the number of "heads". This is considered a single experiment. If you get exactly 50 heads, the experiment is considered a "success".
- You repeat the experiment N=100,000 times. After the N experiments are completed count the total successes, and calculate the probability of success, i.e. the probability of getting exactly 50 heads.

SUBMIT a report that follows the guidelines as described in the syllabus.

- The section on RESULTS must include the calculated answer. *Use the table below for your answer*. Note: You will need to replicate the table in your Word file, in order to provide the answer in your report. Points will be taken off if you do not use the table.
- The code must be provided in the appendix

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

4. The Password Hacking Problem

Your computer system uses a 4-letter password for login. For our purposes the password is restricted to lower case letters of the alphabet only. It is easy to calculate that the total number of passwords which can be produced is $n = 26^4$.

- A hacker creates a list of m random 4-letter words, as candidates for matching the password. Note that it is possible that some of the m words may be duplicates. The number m *has been given to you in a separate document*.
- You are given your own 4-letter password and you are going to check if the hacker's list contains at least one word that matches your password. This process of checking is considered one experiment. If a word in the list matches your password, the experiment is considered a success. Repeat the experiment for $N = 1000$ times and find the probability that **at least one of the words** in the hacker's list will match your password.
- The hacker creates a longer list of k*m random 4-letter words. The numbers k and m *have been given to you in a separate document*. Repeat the previous experiment for $N = 1000$ times and find the probability that **at least one of the words** in the hacker's list will match your password.
- Repeat the previous experiment for $N = 1000$ times to find the **approximate number** (m) of words that must be contained in the hacker's list so that the probability of at least one word matching the password is $p = 0.5$. You should do this by trial and error: assume a value for (m) and calculate the corresponding probability as you did in the previous part. The answer will be value of (m) that makes this probability **approximately equal** to $p = 0.5$.

SUBMIT a report that follows the guidelines as described in the syllabus.

- The section on RESULTS must include the calculated answer. *Use the table below for your answer*. Note: You will need to replicate the table in your Word file, in order to provide the answer in your report. Points will be taken off if you do not use the table.
- The code must be provided in the appendix

Hacker creates m words Prob. that at least one of the words matches the password	***p* =**
Hacker creates k*m words Prob. that at least one of the words matches the password	***p* =**
$p = 0.5$ Approximate number of words in the list	***m* =**

5. References

[1] "*Introduction to Probability*," by H. Pishro-Nik. Available online at: https://www.probabilitycourse.com