Vinh Vu

Dr. Chassiakos

EE 381-04

February 13th, 2019

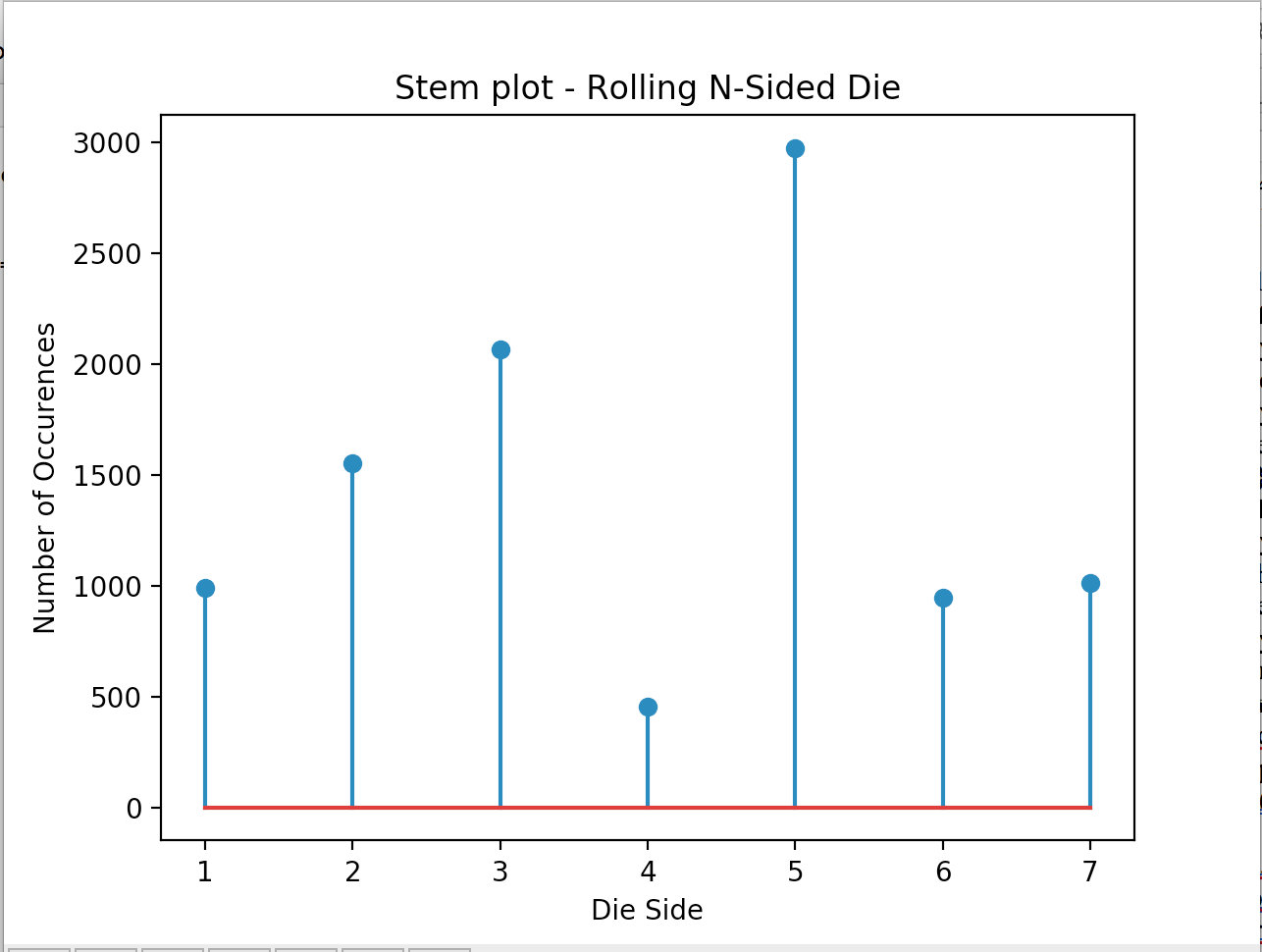
Project 1: Random Numbers and Stochastic Experiment

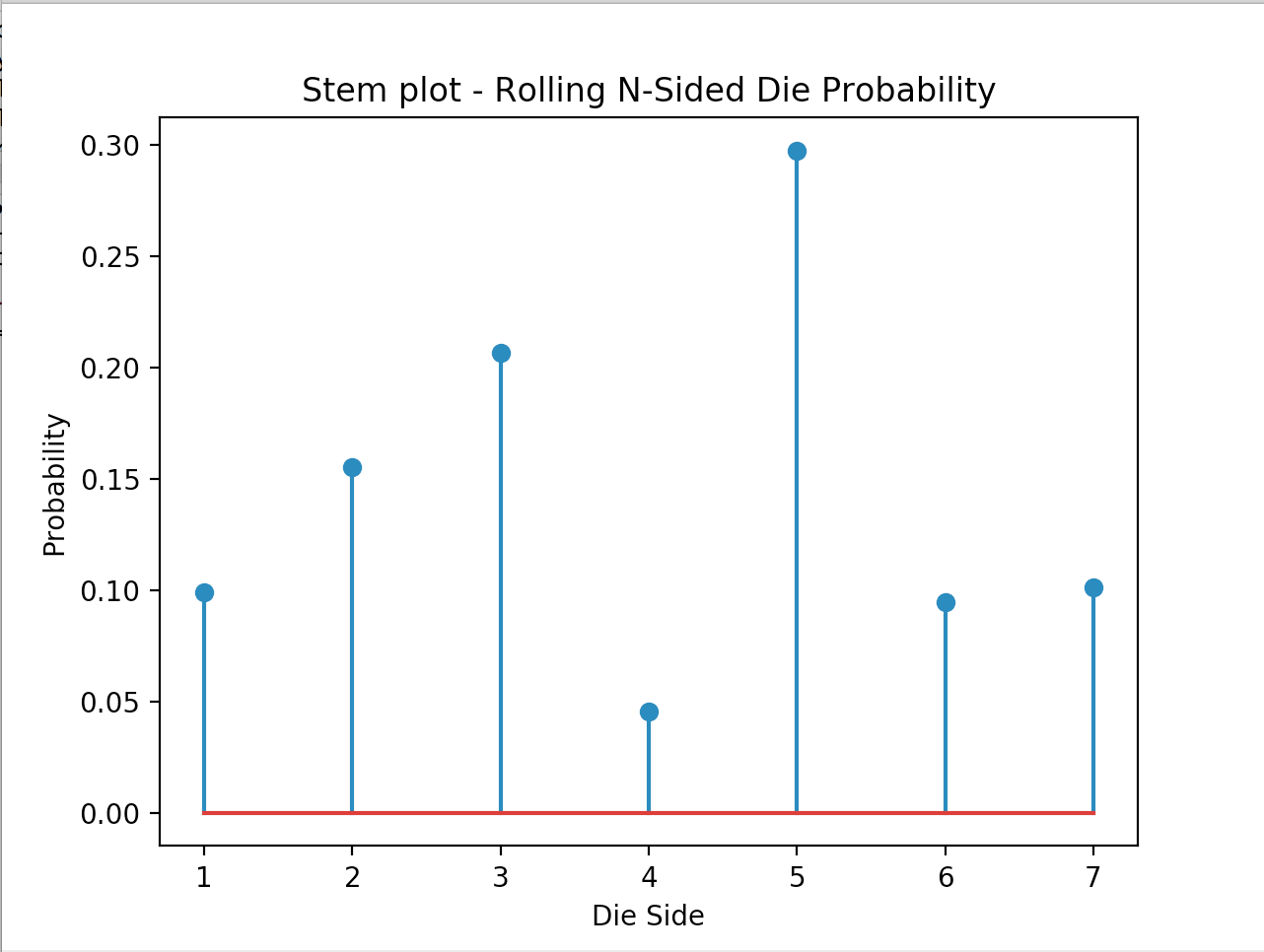
**Problem 1: Function for a n-sided die**

**Introduction:** The program simulates a single roll of an n-sided die. It inputs an array of probability values of rolling the n-th side from users and such array is passed into a function to roll the die. The inputted possibilities, p1+p2+…+pn, must be equal to 1.0 otherwise the probability values are incorrect. After passing the array into the function nSideDie(p), it outputs the number on the face of the die after rolling it once from the list of possible integers {1,2,…,n}.

**Methodology:** A general function is created to roll an n-sided die with a pre-determined probability array passed into it. Within the function, there is a variable assigned to the number of sides on the die for a random roll. An extra array is used to keep track the random roll coordinates with the pre-determined probability. The experiment rolls 10,000. The results are verified by a stem plot to make sure the rolls are according to the non-uniform probabilities.

**Results and Conclusion:**

****

****

**Appendix:**

import random

import numpy as np

import matplotlib

import matplotlib.pyplot as plt

import string

def nSidedDie(p):

if sum(p) != 1:

print('Probability values are incorrect!')

sides = len(p)

cs = np.cumsum(p)

cp = np.append(0,cs)

r = np.random.rand()

for k in range(0,sides):

if r > cp[k] and r<= cp[k+1]:

sides = k+1

return sides

p = [0.1,0.15,0.20,0.05,0.30,0.10,0.10]

N = 10000

test = []

for j in range(0,N):

r = nSidedDie(p)

test.append(r)

b=range(1,9)

sb=len(b)

h1, bin\_edges = np.histogram(test,bins=b)

b1=bin\_edges[0:sb-1]

fig1 = plt.figure(1)

plt.stem(b1,h1)

plt.title('Stem plot - Rolling N-Sided Die')

plt.xlabel('Die Side')

plt.ylabel('Number of Occurences')

plt.show()

fig2=plt.figure(2)

p1=h1/N

plt.stem(b1,p1)

plt.title('Stem plot - Rolling N-Sided Die Probability')

plt.xlabel('Die Side')

plt.ylabel('Probability')

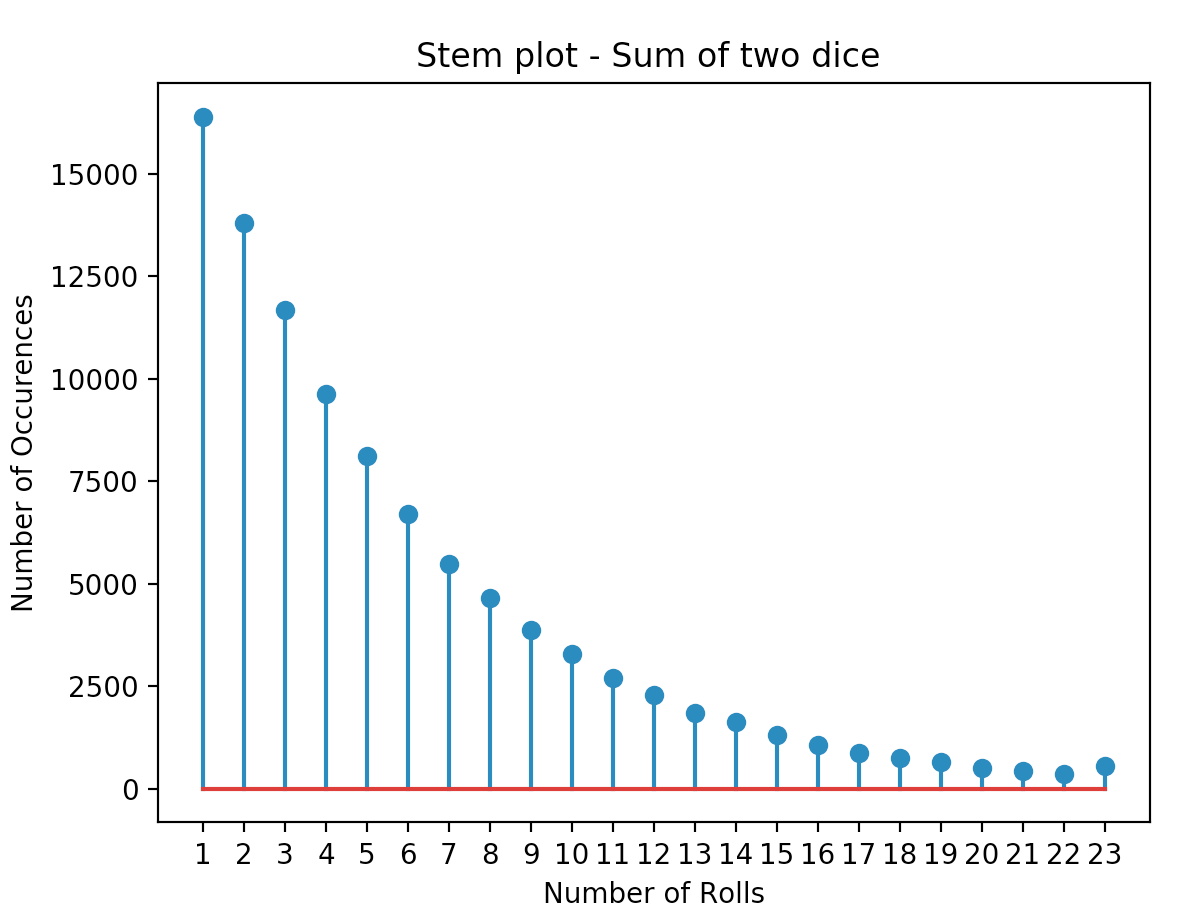
plt.show()

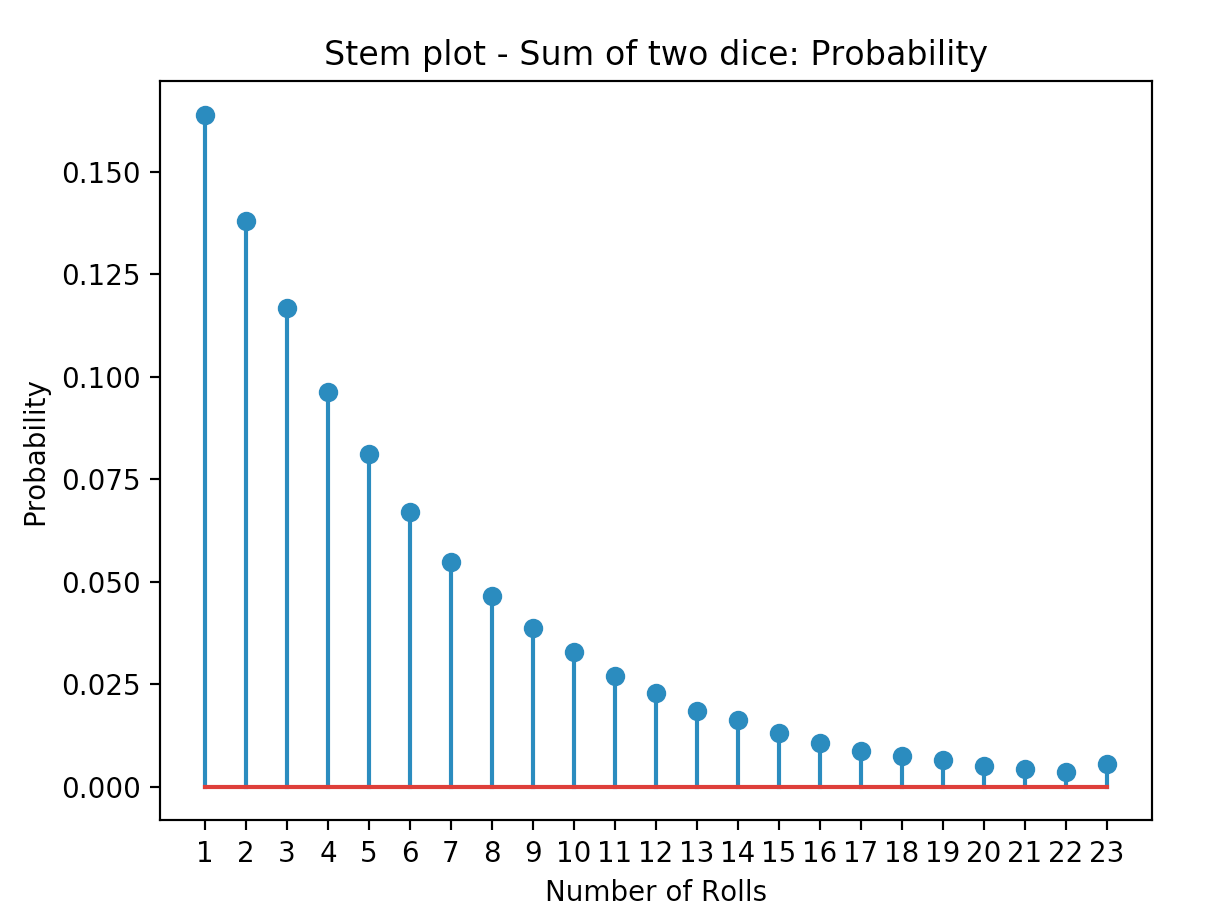
**Problem 2: Number of rolls needed to get a “7” with two dice**

**Introduction:** The program simulates 2 dice rolls to get a sum of 7. Two dice rolls fairly and keep track of when the sum of the first roll equates to 7. When the sum of the two dice hits 7 for the first time, number of rolls is recorded and experiment is exited. The same experiment is repeated for 100,000 times and the number of rolls it takes to get to sum of 7 is saved.

**Methodology:** A function SumOfSeven is created. It contains a while loop to check for the sum of two random dice roll. The while loop contains: two variables declared as a random roll from 1 to 6, which simulates a normal die, a variable to keep track of how many rolls are made, and a summation variable for checking. Once the sum does equal to 7, the while loop terminates and the number of iteration to get to sum of 7 is returned. The second experiment recalls the same function SumOfSeven 100,000 times and saves the results into an array.

**Results and Conclusions:**





**Appendix**:

import random

import numpy as np

import matplotlib

import matplotlib.pyplot as plt

import string

def SumOfSeven():

sum\_7 = 0

iteration = 0

while sum\_7 != 7:

d1 = np.random.randint(1,7)

d2 = np.random.randint(1,7)

iteration+=1

sum\_7=d1+d2

return iteration

test2 = []

r = SumOfSeven()

print("Numer of rolls to achieve sum equal to 7:",r)

N\_2 = 100000

for i in range(0,N\_2):

r = SumOfSeven()

test2.append(r)

b=range(1,25)

sb=len(b)

h1, bin\_edges = np.histogram(test2,bins=b)

b1=bin\_edges[0:sb-1]

fig1 = plt.figure(1)

plt.stem(b1,h1)

plt.title('Stem plot - Sum of two dice ')

plt.xlabel('Number of Rolls')

plt.ylabel('Number of Occurences')

plt.xticks(b1)

plt.show()

fig2=plt.figure(2)

p1=h1/N\_2

plt.stem(b1,p1)

plt.title('Stem plot - Sum of two dice: Probability')

plt.xlabel('Number of Rolls')

plt.ylabel('Probability')

plt.xticks(b1)

plt.show()

**Problem 3: Getting 50 heads when tossing 100 coins**

**Introduction:** The code is to toss 100 fair coins and keep track of the number they land on heads. If exactly 50 coins out of 100 results on heads, it is a success. The program repeats the experiment 100,000 times; the number of successes is being tracked. Success rate is then calculated then outputted.

**Methodology:** A function coinFlip() is created for the experiment. There are two variables declared to keep track of the number of heads and success. The instantiated for-loop iterates 100 times. To flip a fair coin, the program uses random generator between integer 1 and 2; 1 represents heads and 2 represents tails. Whenever the coin lands on 1, the variable head increments. When head is equal to 50, success increments by 1 and is returned. In order to read whether the experiment is a success, success variable has to be equal to 1 or else it’s a fail. The experiment is repeated 100,000 times and the number of success is saved.

**Results and Conclusions:**

|  |  |
| --- | --- |
| Probability of 50 heads in tossing 100 fair coins |  |
| **Ans.** | P = 0.07909 |

**Appendix:**

import random

import numpy as np

import matplotlib

import matplotlib.pyplot as plt

import string

N = 100000

s\_3 = []

def coinFlip():

head = 0

success = 0

for k in range(100):

flip = np.random.randint(1,3)

if flip == 1:

head+=1

if head == 50:

success += 1

return success

flip1 = coinFlip()

if flip1 == 1:

print('Success!')

else:

print('Fail!')

for i in range(0,N\_2):

times = coinFlip()

if times == 1:

s\_3.append(times)

rate = len(s\_3)/N\_2

print('The success probability is',rate)

**Problem 4: The Password Hacking Problem**

**Introduction:** A computer system uses a 4-letter password for login. This password is restricted only to lowercase letters. A hacker compiles a list of m 4-letter words and tries to match his list to the user’s password. For the first experiment, the hacker’s randomly generated password list contains 80,000 elements to match the user’s password. Once the password matches, it is considered a success. The experiment is then repeated for 1000 times and the probability at least one of the words matching is calculated. The second experiment implements the same logic but this time, the hacker’s list is k, or 7, times bigger. For the last section, it is a trial and error experiment. We are trying to get the probability approximately equal to 0.5 by plugging in different number of m.

**Methodology:** A function passMatch() is created with a parameter k. In the function, there is a nested for-loop. One for-loop is to generate 80,000 different 4-letter passwords and one for-loop to randomly generate a 4-letter word with strictly lowercase letters. All of the randomly generated passwords are then saved into an array and then returned. As for the main function, the function’s parameter is 1 when called and then saved. The user’s password is randomly generated as well and being compared to the hacker’s list. If a word matches, a success variable is incremented and is saved into an array. To calculate the probability, the length of the success array is divided by 1000 and then outputted. For the second experiment, the function’s parameter is changed to 7 and then the same logic is repeated. As for getting probability around 0.5, different numbers for the variable m are plugged in and the results are observed.

**Results and conclusions:**

|  |  |
| --- | --- |
| Hacker creates m words  Prob. That at least one of the words matches the password | P = 0.157 |
| Hacker creates k\*m words  Prob. That at least one of the words matches the password | P = 0.707 |
| P = 0.5  Approximate number of words in the list | M = 310000 |

**Appendix:**

import random

import numpy as np

import matplotlib

import matplotlib.pyplot as plt

import string

#Number 4

def passMatch(k):

password=[]

for i in range(k\*310000):

word = ''

for j in range(4):

let=random.choice(string.ascii\_lowercase)

word+=let

password.append(word)

return password

matching = passMatch(1)

success\_rate1 = []

success\_rate7=[]

for k in range(1000):

success = 0

my\_pass = ''

for j in range(4):

my\_letter = random.choice(string.ascii\_lowercase)

my\_pass += my\_letter

for i in range(len(matching)):

if my\_pass == matching[i]:

success+=1

if success != 0:

success\_rate1.append(success)

rate = len(success\_rate1)/1000

print("The success rate:",rate)

matching2 = passMatch(7)

for k in range(1000):

success = 0

my\_pass = ''

for j in range(4):

my\_letter = random.choice(string.ascii\_lowercase)

my\_pass += my\_letter

for i in range(len(matching2)):

if my\_pass == matching2[i]:

success+=1

if success != 0:

success\_rate7.append(success)

rate = len(success\_rate7)/1000

print("The success rate:",rate)