Taylor Bui

Project 1 – Stochastic Experiments

February 13th 2019

1. Function for an n-sided die

Introduction:

This program simulates a singular roll of an n-sided dice given the probabilities of each side as a vector

Methodology:

Using a function called nSidedDie(p) with p being a vector with each of the probability. First I set N to be the number of trials which is 10000 in this case. Then I created an array to store the rolls of each dice. Each roll of dice is simulated through two for loops within each other. The array is then plotted on a graph showing frequencies of each roll.

Appendix :

import numpy as np

import matplotlib

import matplotlib.pyplot as plt

#

def nSidedDie(p):

N=10000

s=np.zeros((N,1))

n=len(p);

print(n)

#

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 = len(b)

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

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

plt.close('all')

prob=h1/N

plt.stem(b1,prob)

# Graph labels

plt.title('PMF for an unfair '+ str(n) + '-sided die' )

plt.xlabel('Number on the face of the die')

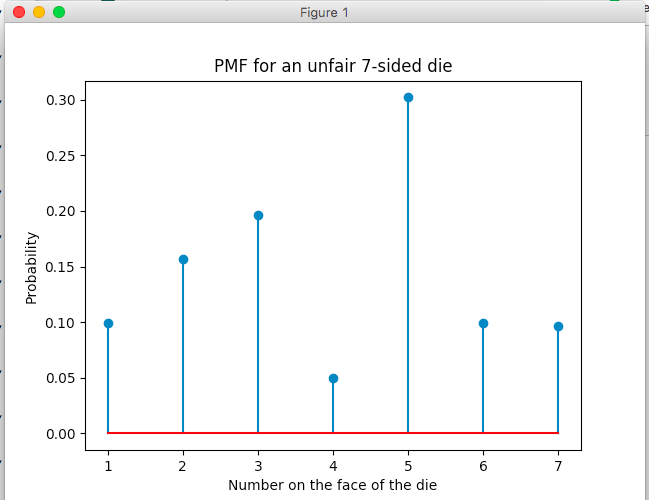
plt.ylabel('Probability')

plt.xticks(b1)

plt.show()

p=[0.10, 0.15, 0.20, 0.05, 0.3, 0.10, 0.1]

nSidedDie(p)



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

Introduction:

Two dices are rolled and the summation of the value of the two dices will be recorded. If the sum equates 7 then it is a success.

Methodology:

The method sum2dice(N) is simulated 10000 successful 7 rolls. First I created an array to store how many dice rolls it took for the experiment to roll 7. I made a variable called sevenCheck that checks if the sum equates 7 and count that stores how many tries it took the experiment to succeed. A while loop is used to check how many times the experiment failed. The probability of succeeding the experiment is recorded and shown on a graph.

Appendex:

import numpy as np

import matplotlib

import matplotlib.pyplot as plt

#

def sum2dice(N):

s = np.zeros((N,1))

for x in range(0,N):

sevenCheck = 0

count = 0

while sevenCheck != 7:

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

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

sevenCheck = d1 + d2

count += 1

s[x] = count

#

b=range(1,32) ; 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)

p1 = h1/N

plt.stem(b1,p1)

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

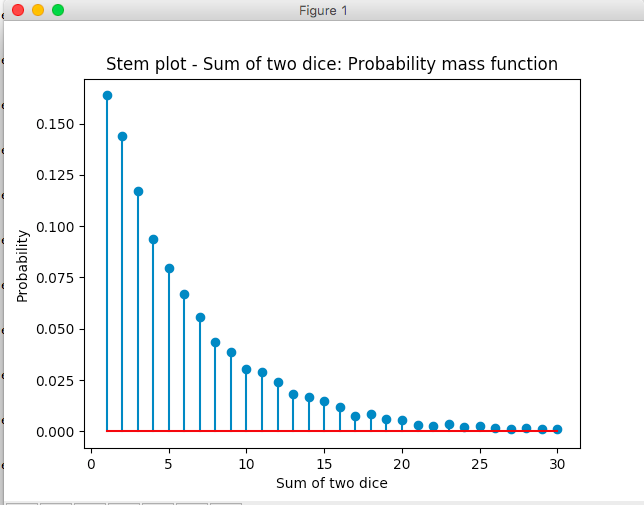
plt.xlabel('Sum of two dice')

plt.ylabel('Probability')

plt.show()

N=10000

sum2dice(N)



3. Getting 50 heads when tossing 100 coins

Introduction:

A fair coin is flipped 100 times. If there are 50 heads then it is a success and we are supposed to find the chance of flipping 50 heads exactly.

Methodology:

I made a function called coin50(n). n is the amount of times the trial is ran. I made a variable called N which stores 100 and that will be the 100 coin flips. Another variable called success which be made which will store how many times 50 heads were flipped successfully. A for loop is implemented that will randomize and find the amount of head flips and tail flips. Inside of the for loop is an if statement that checks if the probability of heads is 50% exactly if it is, then success which increase by 1. After the loop is finishes it will print the success divided by the trials n and produce the percent.

|  |  |
| --- | --- |
| Probability of 50 heads in tossing 100 fair coins |  |
| Ans. | p =0.07964 |

Appendex:

import numpy as np

def coin50(n):

success = 0

N = 100

for i in range(0,n):

coin=np.random.randint(0,2,N)

heads = sum(coin)

tails = N-heads

#

p\_heads = heads/N

p\_tails=tails/N

if p\_heads == .5:

success += 1

print (success/n)

n= 100000

coin50(n)

4. The Password Hacking Problem

Introduction: A hacker tries to crack a password with only lowercase letters. The hacker has a list that he compares and checks for the password.

Methodology: There is a check variable that starts at 0 and increases by one every time the password equals a number in the list. The password is randomly generated from 0 to 26^4 to replicate 4 letters. The array is generated from 0 to 26^4 for m times. The experiment is for looped 1000 times and the probability is printed at the end.

|  |  |
| --- | --- |
| Hacker creates m words  Prob. That at least one of the words matches the password | p = 0.155 |
| Hacker creates k\*m words  Prob. That at least one of the words matches the password | p = 0.716 |
| p = 0.5  Approximate number of words in the list | m = 45000 |

Appendix:

import numpy as np

#

def password(m):

N = 1000

check = 0

n = np.power(26,4)

for i in range (0,N):

p = np.random.randint(0,n)

H = np.random.randint(0,n,m)

if p in H:

check += 1

print(check/N)

#

m = 80000

k = 7

password(m\*k)