Manuel Beltran

Sept. 18, 2019

EE 381

**Project 1**

**Problem 1**

Introduction

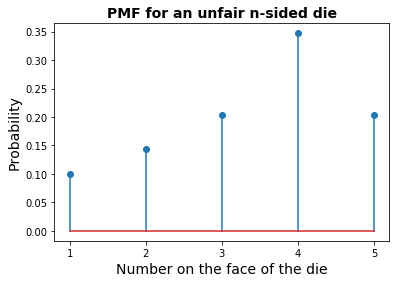
An n-sided die has an assigned unfair probability to land on one of it’s side. The die is rolled and the face is recorded. This process is repeated 10,000 times. From the recorded data, the probability that one of the faces is rolled is plotted on a PMF.

Methodology

After rolling 10,000 times, 1 through n faces have been rolled a certain number of times. Divide each face’s total rolls by the total number of rolls (10,000 in this case) and it will give the probability of rolling that face.

Results and Conclusion

The results below show that the probability of rolling a certain face closely matches the probability assigned to that face.



An Appendix

import numpy as np

import matplotlib.pyplot as plt

def nSidedDie(p):

n = len(p)

array = np.array(p)

#

cs = np.cumsum(array)

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

return d

p=[0.10, 0.15, 0.20, 0.35, 0.20]

N = 10000

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

n = len(p)

for i in range (0,N):

r = nSidedDie(p)

s[i] = r

# 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 n-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)

**Problem 2**

Introduction

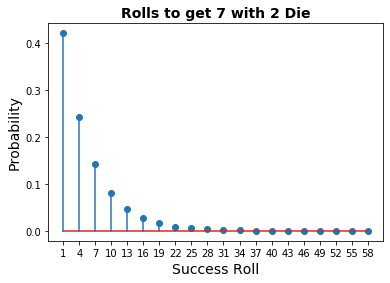
The goal is to get a roll of ‘7’ from rolling 2 dice. The number of rolls until a ‘7’ is achieved is recorded. This process of getting ‘7’ is to be repeated 100,000 times. From the recorded data, the probability of rolls until a 7 is achieved is plotted on a PMF.

Methodology

Die are rolled until a successful 7 is rolled and this “total number of rolls until 7” is recorded. This is repeated 100,000. The amount of times that a certain “total number of rolls until 7” occurs is divided by 100,000 trials to get the probability of it happening.

Results and Conclusion

The results below show that the probability of rolling without a 7 consecutively gets increasingly more difficult or decreasingly less probable.



An Appendix

import numpy as np

import matplotlib.pyplot as plt

N = 100000

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

for k in range (0,N):

total = 0

rolls = 0

while total != 7:

rolls += 1

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

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

total = d1 + d2

s[k] = rolls

# Plotting

b = range (1, int(s.max() + 2),3)

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('Rolls to get 7 with 2 Dice', fontsize = 14, fontweight = 'bold')

plt.xlabel('Success Roll', fontsize = 14)

plt.ylabel('Probability', fontsize = 14)

plt.xticks(b1)

**Problem 3**

Introduction

100 fair coins are tossed the result of exactly 50 heads is considered a success. This is done 100,000 times and the probability of a success occurring is determined.

Methodology

A trial is done 100,000 times and for each trial the number of heads occurring is recorded for each trial. Having 50 heads is considered a success; therefore, the number of times “only 50 heads” occurs across all trials is divided by 100,000 trials to get the probability of exactly 50 heads tossed out of 100 coins tossed.

Results and Conclusion

Generally there is a 0.5 chance of landing either heads or tails. The chances of 50 heads out of 100 tosses would be close to 0.5 but not exactly because there are not enough tosses to get a more statistical 0.5. If there were more tosses, the chances of head tosses would become more closer to 0.5 chance.

|  |  |
| --- | --- |
| Probability of 50 heads in tossing 100 coins |  |
| Ans. | p = 0.0791 |

An Appendix

import numpy as np

N = 100000

success = 0

for i in range (0,N):

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

heads = sum(results)

if heads == 50:

success += 1

prob = success / N #results at around 0.0791

**Problem 4**

Introduction

The number of passwords a user can make from 4 lowercase letters is 26\*\*4. A hacker has a list of passwords with size ‘m’. A trial is done to see if a random password is in the hacker’s random generated list. If it is, then it is a success. This process is done 1000 times to see the probability of success and recorded. m is increased by ‘k’ times and the process is done 1000 times again to see the probability of success and recorded. ‘m’ is then increased until the probability of success is near 0.5 and the size is recorded.

Methodology

The hacker’s list of passwords is labeled 1 to size of m. A random number between 1 through 26\*\*4 is generated for a random password. If the random number is less than or equal to m, then the password is in the list and the trial is a success. The number of successes is divided by the trials to get the probability of success. This is done again with a new ‘m’ \* ‘k’. ‘m’ is then continuously increased until the previous process returns a probability slightly within reach of 0.5 probability.

Results and Conclusion

|  |  |
| --- | --- |
| Hacker creates words Prob. that at least one of the words matches the password | p = .135 |
| Hacker creates k\* words Prob. that at least one of the words matches the password | p = .629 |
| p = 0.5 Approximate number of words in the list | m = 280,000 |

An Appendix

import numpy as np

def probOfHack(m,k):

success = 0

N = 1000

n = 26\*\*4

for i in range(0,N):

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

hackerList = np.random.randint(0,n,m\*k)

if (password in hackerList):

success += 1

prob = success / N

return prob

m = 70000

prob1 = probOfHack(m,1)#recorded answer .135

prob2 = probOfHack(m,7)#recorded answer .629

expectedProb = 0.5

result = 0

while result < (expectedProb - .02):

m += 1000

result = probOfHack(m,1) #recorded answer m = 280000