Kevin Long

Professor Chassiakos

EE 381

27 February 2019

Project 3 – Binomial and Poisson Distributions

1. Experimental Bernoulli Trials

a. Introduction

We have three identical multi-sided unfair dice. The probability vector *p* for the dice is [0.2, 0.1, 0.15, 0.3, 0.2, 0.05]. One roll is considered a "success" if we get 1 for the first die, 2 for the second die, and 3 for the third die. We roll the three dice *n* = 1000 times. The number of successes in *n* rolls will be contained in "X." This is considered one experiment. In conclusion, plot the PMF of "X."

b. Methodology

We first create an array "X" that will record each successful experiment. For each experiment, we establish a counter "count" initialized to 0. The rolled value of each die is simulated via "np.random.choice()." For each time all three dice are rolled, if die one is equal to 1, die two is equal 2, and die 3 is equal to 3, then we increment the counter by 1 to record the success. After each experiment, add the value of the counter into "X" to record the number of successes in that particular round. The graph of "X" is created with its' calculated histogram.

c. Result(s) and Conclusion(s)

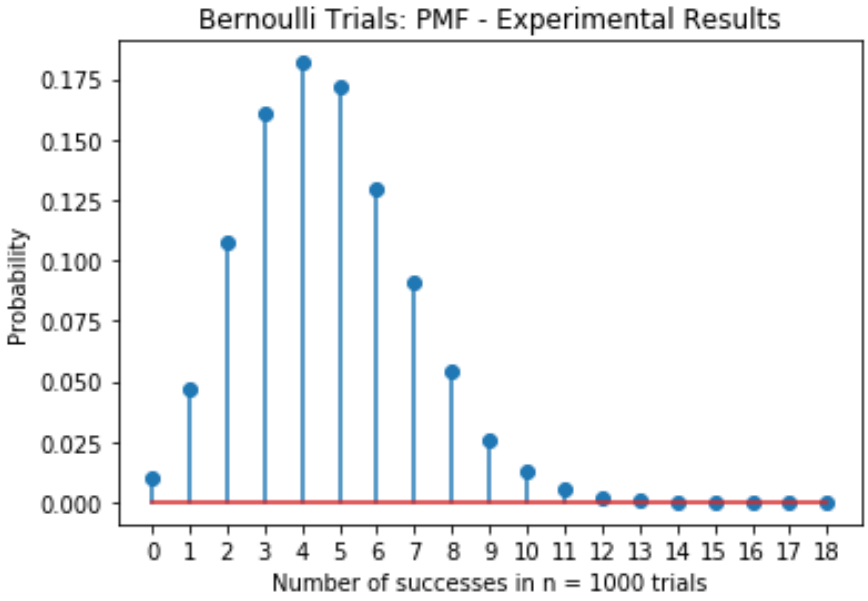

Figure 1. Results of Experimental Bernoulli Trials (PMF)

The results yield in approximations of successes due to it being a simulation as shown in **Figure 1**.

d. **Source Code**

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

#Simulation
def bernoulliTrials(n, N, c, p):
    X = np.zeros((N,1))

    for experiment in range(0, N):
        count = 0
        die1 = np.random.choice(c, n, p)
        die2 = np.random.choice(c, n, p)
        die3 = np.random.choice(c, n, p)
        for i in range(0, n):
            if die1[i] == 1 and die2[i] == 2 and die3[i] == 3:
                count += 1
        X[experiment] = count

    # Plotting
    b=range(0, 20)
    sb=np.size(b)
    h1, bin_edges=np.histogram(X, bins=b)
    b1=bin_edges[0:sb-1]
    plt.close('all')
    prob=h1/N
    plt.stem(b1,prob)
    # Graph labels
    plt.title('Bernoulli Trials: PMF - Experimental Results')
    plt.xlabel('Number of successes in n = 1000 trials')
    plt.ylabel('Probability')
    plt.xticks(b1)
    plt.show()

if __name__ == '__main__':
    #The probability vector for the multi-sided dice is:  p=[0.2,  0.1,  0.15, 0.3, 0.2, 0.05]
    n = 1000 #Num of times to roll 3 dice
    N = 10000 #Num of experiments
    c = [1, 2, 3, 4, 5, 6]
    p = [0.2, 0.1, 0.15, 0.3, 0.2, 0.05]
    bernoulliTrials(n, N, c, p)
```

2. Calculations using the Binomial Distribution

a. **Introduction**

We will use the theoretical formula for the Binomial distribution to calculate the probability *p* of successes in a single roll of the three dice. The *p* array is given as [0.2, 0.1, 0.15, 0.3, 0.2, 0.05]. We will then compare the results to that of Problem 1.

b. **Methodology**

The library scipy.stats will be imported for its' binomial function. This will save us time from hardcoding the formula. The variable X will hold the number of successes in *n* Bernoulli trials. The variable y will store our probability calculated as PMF form by the function "binom.pmf()." The results are then graphed.

c. **Result(s) and Conclusion(s)**

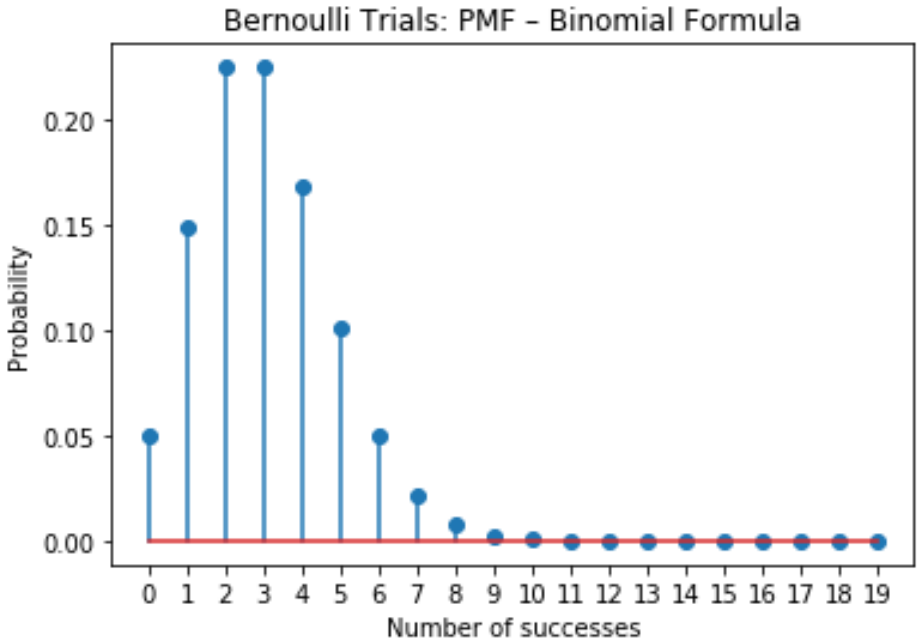

Figure 2. Results of Binomial Distribution (PMF)

The Binomial distribution formula yields more accurate results because it is not a simulation as shown by **Figure 2** in comparison to **Figure 1**.

d. **Source Code**

```
21  import numpy as np
22  from scipy.stats import binom
23  import matplotlib.pyplot as plt
24
25  #Actual formula - Binomial Formula
26  def binomial(n, pArr):
27      p = pArr[0] * pArr[1] * pArr[2]
28      X = np.arange(0,20)
```

```
29      y = binom.pmf(k, n, p) #Binomial formula
30
31      #Graphing
32      plt.stem(X, y)
33      plt.title('Bernoulli Trials: PMF – Binomial Formula')
34      plt.xlabel('Number of successes')
35      plt.ylabel('Probability')
36      plt.xticks(np.arange(min(X),max(X)+1, 1.0)) #Fix x-axis scale
37      plt.show()
38
39  if __name__ == '__main__':
40      #The probability vector for the multi-sided dice is:  p=[0.2,  0.1,  0.15, 0.3, 0.2, 0.05]
41      n = 1000 #Num of times to roll 3 dice
42      N = 10000 #Num of experiments
43      c = [1, 2, 3, 4, 5, 6]
44      p = [0.2, 0.1, 0.15, 0.3, 0.2, 0.05]
45      binomial(n, p)
```

3. Approximation of Binomial by Poisson Distribution

a. Introduction

Think of the case when the probability *p* in a Bernoulli trial is small and the number of trials *n* is large (in actuality $n >= 50$ and $np < 5$). In this event, the Poisson distribution formula is used to approximate the probability of successes in *n* trials. The parameter "lambda" is needed for the Poisson formula and is calculated as such: lambda = *np*. The probability array has been given as [0.2, 0.1, 0.15, 0.3, 0.2, 0.05].

b. Methodology

The library scipy.stats will be imported for its' poisson function. This will save us time from hardcoding the formula. The variable "lambd" is a representation of the lambda value necessary for the poisson formula to function. The variable X will hold the number of successes in *n* Poisson trials. The variable y will store our probability calculated as PMF form by the function "ss.poisson.pmf()." The results are then graphed.

c. Result(s) and Conclusion(s)

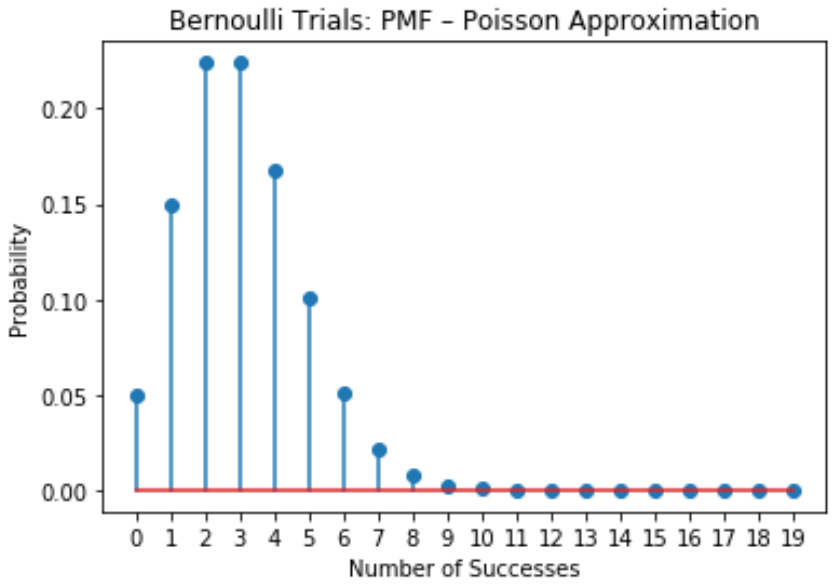

Figure 3. Results of Poisson Distribution (PMF)

Although the Poisson formula is an approximation of the Binomial under certain conditions, it (**Figure 2**) still yields more accurate results in comparison to **Figure 1**.

d. **Source Code**

```
import numpy as np
import scipy.stats as ss
import matplotlib.pyplot as plt

#Approximation to Binomial - Poisson Formula
def poisson(n, p):
    lambd = n * p[0] * p[1] * p[2] #Lambda
    X = np.arange(0,20)
    y = ss.poisson.pmf(x, lambd) #Poisson formula

    #Graphing
    plt.stem(X, y)
    plt.title('Bernoulli Trials: PMF – Poisson Approximation')
    plt.xlabel('Number of Successes')
    plt.ylabel('Probability')
    plt.xticks(np.arange(min(X),max(X)+1, 1.0)) #Fix x-axis scale
    plt.show()

if __name__ == '__main__':
    #The probability vector for the multi-sided dice is:  p=[0.2,  0.1,  0.15, 0.3, 0.2, 0.05]
    n = 1000 #Num of times to roll 3 dice
    N = 10000 #Num of experiments
    c = [1, 2, 3, 4, 5, 6]
```

```
27.  p = [0.2, 0.1, 0.15, 0.3, 0.2, 0.05]
28.  poisson(n, p)
```