Project 3

Qidi Han

February 12, 2020

Question 1

A components manufacturer delivers a batch of 125 microchips to a parts distributor. The distributor checks for lot conformance by counting the number of defective chips in a random sampling (without replacement) of the lot. If the distributor finds any defective chips in the sample they reject the entire lot. Suppose that there are six defective units in the lot of 125 microchips. Simulate the lot sampling to estimate the probability that the distributor will reject the lot if it tests five microchips. What is the fewest number of microchips that the distributor should test to reject this lot 95 percent of the time?

Theory

In this question, I will generate 10000 sample tests. The total microchips are 125, the probability of the defective units is 6/125. The first loop will generate a test if this test will reject or not. 0 mean reject, 1 mean good. There are 10000 samples in my experiment. Finally, I will calculate the average of percentage of rejection. The second step is I will run through the defective unit number from 1 to 125, find the percentage of this rejection greater 95, and repeat the same process above with 10000 times. When the percentage greater than 95, I will save the number of defective unit to the array. Finally, I will calculate the percentage of this array. I will discuss the result with the theoretical result

Algorithm 1: Code for theoretical result of hyper-geometric distribution

```
/* Theoretical result of hyper-geometric distribution */
[M, n, N] = [125, 6, 5]
rv = hypergeom(M, n, N)
x = np.arange(1, N+1)
pmf_dogs = rv.pmf(x)
out=sum(pmf_dogs)
print(out)

```

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
from collections import Counter
from scipy.stats import hypergeom
import matplotlib.pyplot as plt

total_chip = 125;
defective_chip=6;
total_sampling=10000;
arr_fail=[];
fail=0;
for j in range(0,total_sampling):
    total_chip=125;
    fail=0;
    for i in range(0,5):
        randomnumber = np.random.rand();
        if randomnumber<(defective_chip/total_chip):
            fail=1;
            break;
        total_chip=total_chip-1;
    if fail==1:
        arr_fail.append(0) # fail
    else:
        arr_fail.append(1) #good

counter = Counter(arr_fail)
print(counter)
print(counter[0]/total_sampling)
[M, n, N] = [125, 6, 5]
rv = hypergeom(M, n, N)
x = np.arange(1, N+1)
pmf_dogs = rv.pmf(x)
out=sum(pmf_dogs)
print(out)
```

```
Counter({1: 7751, 0: 2249})
0.2249
0.22126687368241194
```

Figure 1: Code and results for average of probability of rejection

[21]:
```
import matplotlib.pyplot as plt
import numpy as np
from collections import Counter
total_chip = 125;
defective_chip=6;
total_sampling=10000;
arr_fail=[];
fail=0;
minm_rate=[];
minm_chip=[];
for run in range(0,50):
    for minm in range(6,125):
        arr_fail=[];
        for j in range(0,total_sampling):
            total_chip=125;
            fail=0;
            for i in range(0,5):
                randomnumber = np.random.rand();
                if randomnumber<(minm/total_chip):
                    fail=1;
                    break;
                total_chip=total_chip-1;
            if fail==1:
                arr_fail.append(0) # fail
            else:
                arr_fail.append(1) #good

        counter = Counter(arr_fail)
        minm_rate.append(counter[0]/total_sampling);
        if(counter[0]/total_sampling >= 0.95):
            minm_chip.append(minm);
            break;

avg = sum(minm_chip)/len(minm_chip)
print(avg)
```

```
55.78
```

Figure 2: Code and results for average of the fewest number of microchips that the rejection precentage above 95

Explanation

In this question, we used hyper-geometric distribution. The hyper-geometric distribution means the the probability of successes without replacement, but in our case, the successes

will be the case of fail. From the result, the average probability that the batch will reject is around 22.49 percentage. The theoretical result is around 22.12 percentage. As the number of sampling increasing, the percentage will close to 22.12. For the second test, the average of fewest number of microchips that the reject of this lot 95 percent will be 55.78. Some of the number are 47 and 48, the majority of the result are 54. I think this is a reasonable result.

Question 2

Suppose that 120 cars arrive at a freeway onramp per hour on average. Simulate one hour of arrivals to the freeway onramp: (1)subdivide the hour into small time intervals($<$ 1 second) and then (2) perform a Bernoulli trial to indicate a car arrival within each small time-interval. Generate a histogram for the number of arrivals per hour. Repeat the counting experiment by sampling directly from an equivalent Poisson distribution by using the inverse transform method (described in class). Generate a histogram for the number of arrivals per hour using this method. Overlay the theoretical p.m.f. on both histograms. Comment on the results.

Theory

In this question, I will use Poisson arrival theory. Poisson arrival is counting the number of event that happened during certain period. In our experiment, I will random generate 10000 tests for the number of arrivals per hour by using the scipy.stats.poisson() function. Then I will plot the data into a histogram figure. Then i will repeat the same experiment by using the inverse transform method. Also, I will plot the data with histogram figure same as last step. For both histograms, I will plot the the theoretical p.m.f. Finally, I will discuss the result on the explanation part and I will show the code on the figures and show the formula in the equation part.

Algorithm 2: Code and formula for Poisson distribution.

```
   /* Code for generate sample of Poisson distribution                    */
1  lmbda = 50;
2  N = 1000;
3  p = lmbda/N;
4  u = np.random.rand(N,1);
5  bernoulliTrials = u<p
6  x = np.sum(bernoulliTrials);
7  print(x)
   /* Code for Poisson distribution by using inverse transform method     */
8  x = 0 p = m.exp(-lmbda) s1 = p u = np.random.rand(); while u > s1:  x = x + 1  p
       = p*lmbda/x  s1 = s1 + p  /* Formula for p.m.f                     */
9
```

$$Px(x) = \frac{(\lambda^x) * (e^-(\lambda))}{x!}(if x = 1, 2, 3...) \tag{1}$$

$$Px(x) = 0(otherwise) \tag{2}$$

[8]:
```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import poisson
import math
lmbda = 120;
N = 7200;
p = lmbda/N;
u = np.random.rand(N,1);
bernoulliTrials = u<p
s = np.random.poisson(lmbda, N)
print(s)
max_value = int(max(s));
min_value = int(min(s));

x=np.arange(0, max_value, 1).tolist()

y=[];
for i in range(1,max_value):
    y.append(-lmbda + x[i]*np.log(lmbda)-sum(np.log(x[1:i+1])))


bin1 = np.arange(min_value,max_value)-0.5;
plt.hist(s,bin1,color='r')
xplot=np.arange(min_value, max_value-1, 1).tolist()
yplot=np.exp(y[min_value:max_value])*N

plt.plot(xplot,yplot,color='g')
plt.title('Histogram for the number of arrivals per hours')
plt.show()
```

```
[133 115 107 ... 125 107 131]
```

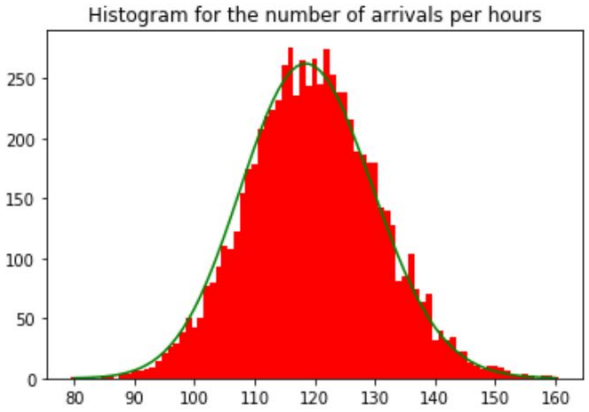


Figure 3: Code for generating the histogram by using formula

```
import numpy as np
from scipy.stats import poisson
import math
import math as m
lmbda = 120;
N = 7200;
p = lmbda/N;
u = np.random.rand(N,1);
bernoulliTrials = u<p
s=[];
for i in range(0,N):
    x = 0
    p = m.exp(-lmbda)
    s1 = p
    u = np.random.rand();
    while u > s1:
        x = x + 1
        p = p*lmbda/x
        s1 = s1 + p
    s.append(x)
# print(s)
max_value = int(max(s));
min_value = int(min(s));
x=np.arange(0, max_value, 1).tolist()
y=[];
for i in range(1,max_value):
    y.append(-lmbda + x[i]*np.log(lmbda)-sum(np.log(x[1:i+1])))
bin1 = np.arange(min_value,max_value+1)+0.5;
plt.hist(s,bin1,color='r')
xplot=np.arange(min_value, max_value-1, 1).tolist()
yplot=np.exp(y[min_value:max_value])*N
plt.plot(xplot,yplot,color='g')
plt.title('Histogram for the number of arrivals per hours')
plt.show()
```

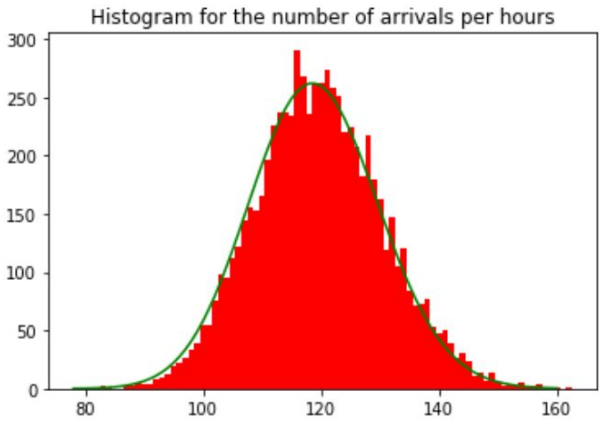


Figure 4: Code for generating the histogram by using inverse transform method

Explanation

In this question, Poisson distribution is the combination of the Bernoulli trial. There are two ways of generating the histogram, the first way is using the code above, the second is using the code with inverse transform method above. For both histograms, the experiments result is sames as theoretical results.

Question 3

Define the random variable N=min (n: Xi>4) smallest number of uniform random samples whose sum is greater than four. Generate a histogram using 100, 1000, and 10000 samples for N. Comment on E[N]

Theory

In this question, I will random generate the random number between 0 and 1 since there are standard uniform random variable. Then I will write down the index that the sum of the preview number is greater than four. I will repeat the experiment with 100, 1000 and 10000 samples. Finally, I will discuss the result and mean of those different samples.

Algorithm 3: .

```
   /* formula for define the random variable                    */
1  N=min(n: Xi>4)
```

[8]:
```
import matplotlib.pyplot as plt
import numpy as np
a = 0
b = 1
rand_arr1 = [] # create an empty array to store the random samples
sumnumber = [];
number_sampling=100;
for k in range(0,number_sampling):
    sum=0;
    i=0;
    while True:
        i=i+1;
        sum=sum+(b-a)*np.random.rand() + a;
        if sum>4 :
            sumnumber.append(i)
            break;
avg=np.average(sumnumber)
print(avg)
plt.hist(sumnumber,color='r')
```

```
8.74
```

[8]:
```
(array([ 1.,  5., 19., 22.,  0., 21., 18.,  8.,  5.,  1.]),
 array([ 5. ,  5.8,  6.6,  7.4,  8.2,  9. ,  9.8, 10.6, 11.4, 12.2, 13. ]),
 <a list of 10 Patch objects>)
```

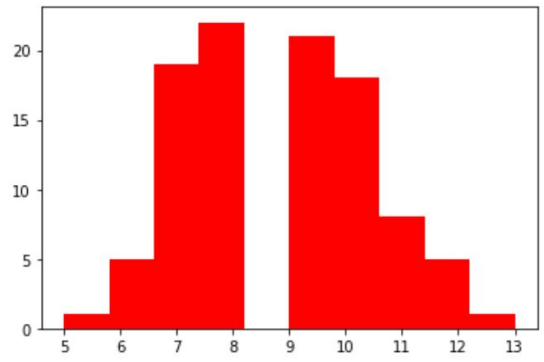


Figure 5: Code for plot and average of smallest number of uniform random samples for 100 samples for N

[7]:
```
import matplotlib.pyplot as plt
import numpy as np
a = 0
b = 1
rand_arr1 = [] # create an empty array to store the random samples
sumnumber = [];
number_sampling=1000;
for k in range(0,number_sampling):
    sum=0;
    i=0;
    while True:
        i=i+1;
        sum=sum+(b-a)*np.random.rand() + a;
        if sum>4 :
            sumnumber.append(i)
            break;
avg=np.average(sumnumber)
print(avg)
plt.hist(sumnumber,color='r')
```

```
8.628
```

[7]:
```
(array([  7.,  82., 176., 239., 211., 157.,  73.,  37.,  12.,   6.]),
 array([ 5.,  6.,  7.,  8.,  9., 10., 11., 12., 13., 14., 15.]),
 <a list of 10 Patch objects>)
```

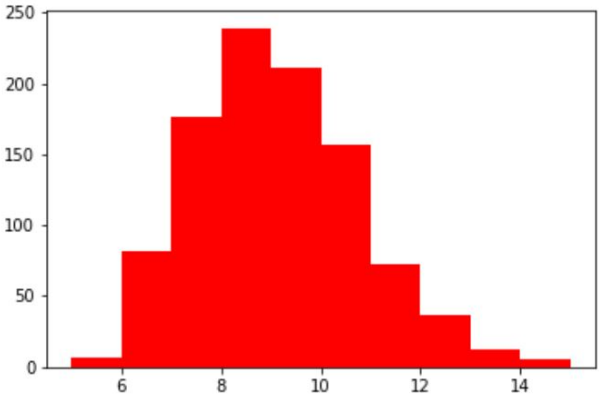


Figure 6: Code for plot and average of smallest number of uniform random samples for 1000 samples for N

[6]:
```
import matplotlib.pyplot as plt
import numpy as np
a = 0
b = 1
rand_arr1 = [] # create an empty array to store the random samples
sumnumber = [];
number_sampling=10000;
for k in range(0,number_sampling):
    sum=0;
    i=0;
    while True:
        i=i+1;
        sum=sum+(b-a)*np.random.rand() + a;
        if sum>4 :
            sumnumber.append(i)
            break;
avg=np.average(sumnumber)
print(avg)
plt.hist(sumnumber,color='r')
```

```
8.6559
```

[6]:
```
(array([8.290e+02, 1.806e+03, 2.362e+03, 3.649e+03, 7.650e+02, 3.540e+02,
        2.090e+02, 1.500e+01, 8.000e+00, 3.000e+00]),
 array([ 5. ,  6.3,  7.6,  8.9, 10.2, 11.5, 12.8, 14.1, 15.4, 16.7, 18. ]),
 <a list of 10 Patch objects>)
```

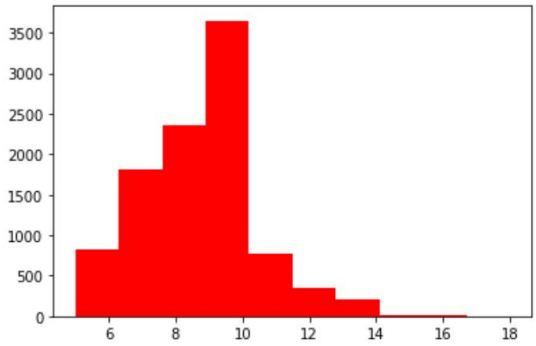


Figure 7: Code for plot and average of smallest number of uniform random samples for 10000 samples for N

Explanation

In this question, there are different different means and result for different samples size. As the samples size increasing, the average of number will close to 8.65. The plot are more close to a geometric random variable. Geometric random variable means the first success term of this number of trials. The E[x] = 1/p for geometric random variable, and V[x] = p(1-p)$^(n-1)$

Question 4

Produce a sequence Xk where Pi=P/j for j = 1,2, , 60 where p is a constant for you to determine. [This is equivalent to spinning the minute hand on a clock and observing the

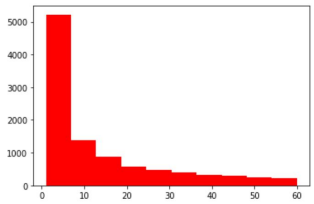
stopping position if P[stop on minute j]=P/j]. Generate a histogram. Define the random variable Nj = mink:Xk=j. sampling from N60. Estimate E[N60] and Var[N60]. Compare you estimates with the theoretical values

Theory

In this question, the sequence (xk) are i.i.d samples where pj=p/j for j=1,2,...60. The first step we will find a p to make this sequence a valid distribution. The sum of Pj = 1 will make the distribution valid. Then I will use np.random.choice function to generate random number between 1 to 60 with probability pj. Then I will calculate the mean and variance by those data point. Next, I will use geometric distribution formula to calculate the theoretical mean and variance.

Algorithm 4: Code for question 4.

```
/* Code for compute p                                                    */
for i in range(1,61):
sum1 = sum1 + (1/i)
p=1/sum1;
```




```
[3]: import numpy as np
import matplotlib.pyplot as plt

sum1=0
number_lopp=1000;
for i in range(1,61):
    sum1 = sum1 + (1/i)
p=1/sum1;
xk =[];
for j in range(1,61):
    xk.append(p/j);
once_1=[];
for j in range(1,number_lopp):
    i=0;
    while True:
        number = np.random.choice(np.arange(1,61), p=xk);
        i=i+1;
        if(number == 60):
            once_1.append(i)
            break;

print(xk[6])
print('Mean of sampling:',str(np.mean(once_1)),'Theoretical Mean of X:', 1/xk[59])
print('Variance of sampling:',str(np.var(once_1)),'Theoretical Variance of X:', (1-xk[59])/(xk[59]*xk[59]))

plt.hist(once_1,color='r')

0.030525875772495704
Mean of sampling: 284.3853853853854 Theoretical Mean of X: 280.79222477710414
Variance of sampling: 73181.03966629291 Theoretical Variance of X: 78563.48127049867
```

Figure 8: Code, experiment and theoretical result and figure for histogram of N60

Explanation

In this question, the mean of sampling is around 284.386, the theoretical mean of the sampling is around 280.792. The variance of sampling is around 73181.03966629291, and the

theoretical variance of this sampling is 78563.48127049867. The experiment and theoretical result are very close. Therefore, the experiment is success. The histogram figure also shows a good geometric distribution.

Question 5

Use the accept-reject method to sample from the following distribution Pj by sampling from the (non-optimal) uniform auxiliary distribution ...

Theory

In this question, I will do an experiment with discrete p.m.f accept-reject theorem. The question will give the density Pq, it's uniform distribution. Then, find a constant c such that c=max(pj qj) = 3. Next, generating a uniform random number u and v. Use the equation cu<=pv/qv to determine if we need to reject v. Record the number of v and the number of accepts. There are 20 qj and pj in our question. All qj equal 0.05. The question already given P1 to p10, and I will make P11 to P20 to 0. Then I will use the record data to plot the histogram, Finally, I will compute the sample mean and sample variance to compare the theoretical values. I will also plot the theoretical value into the histogram. I will discuss the efficiency in the explanation part.

Algorithm 5: Process of accept-reject method

```
/* for loop for the process                                    */
for i from 0 to 10000 do
    k = 0
    while True:
    j = int(1 + np.floor(20*np.random.rand()))
    k = k + 1
    if (3*np.random.rand()) <= p[j-1]/0.05:
    X.append(j)
    C.append(k)
    break
```

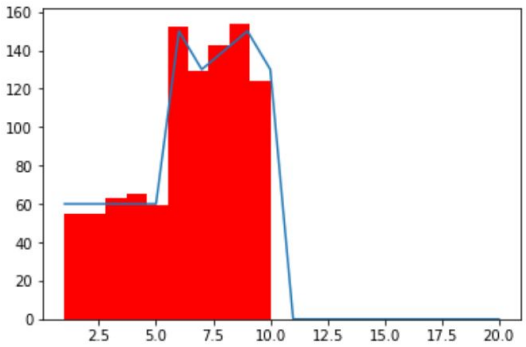
[11]:
```
import numpy as np
import matplotlib.pyplot as plt

p=[0.06, 0.06, 0.06, 0.06, 0.06, 0.15, 0.13, 0.14, 0.15, 0.13, 0, 0, 0, 0, 0, 0, 0 ,0, 0 ,0]

q=[];
for j in range(1,21):
    q.append(0.05);
N = 1000
X = []
C = []
the_var=[];
for i in range(1,N):
    the_var.append(1)
    the_var.append(2)
    the_var.append(3)
    the_var.append(4)
    the_var.append(5)
    the_var.append(6)
    the_var.append(7)
    the_var.append(8)
    the_var.append(9)
    the_var.append(10)
for i in range(1,N):
    k = 0
    while True:
        j = int(1 + np.floor(20*np.random.rand()))
        k = k + 1
        if (3*np.random.rand()) <= p[j-1]/0.05:
            X.append(j)
            C.append(k)
            break


plt.hist(X,color='r')
y= [x*N for x in p]
plt.plot([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20], y)
print(np.dot(p,np.arange(1,21)))
print('Variance of X:',str(np.var(X)),'Theoretical Variance of X:',str(np.var(the_var)), )
print('Mean of X:',str(np.mean(X)),'Theoretical Mean of X:',str(np.dot(p,np.arange(1,21))))
print('Mean of C:',str(np.mean(C)),'Theoretical Mean of C:',str(3))
print('Efficiency of C:',str(1/np.mean(C)),'Theoretical Efficiency:',str(1/3))
```

```
6.48
```

Figure 9: Code for accept-reject method

```
print('Mean of C:',str(np.mean(C)),'Theoretical Mean of C:',str(3))
print('Efficiency of C:',str(1/np.mean(C)),'Theoretical Efficiency:',str(1/3))
```

```
6.48
Variance of X: 6.968718468217969 Theoretical Variance of X: 8.25
Mean of X: 6.5005005005005 Theoretical Mean of X: 6.48
Mean of C: 3.009009009009009 Theoretical Mean of C: 3
Efficiency of C: 0.3323353293413174 Theoretical Efficiency: 0.3333333333333333
```



[ ]:

[ ]:

Figure 10: Histogram and result for this experiment

Explanation

In this question, I have 6.5 as the mean of X. The theoretical mean is 6.48.I have 6.968 as variance of X, the theoretical variance is around 8.25. The mean of C is around 3, and the theoretical mean of c is 3. The efficiency is equal 1/c. Therefore, The experiment and theoretical result is closed to each other. The graph also shows a good performance.