EE511 Assignment 3

Zechen Ha

February 10, 2020

Question 1

Experiment

This is a hypergeometric process when we choose 5 out of 125 without replacement. We can calculate the probability with the hypergeometric formula and find the probability of test fails.

Code

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

simu_times = 1000
lot_num = 125
defective_unit_num = 6
test_num = 5
reject_times = 0

for simulation in range(simu_times):
    result = 0
    lot = np.append(np.zeros((lot_num-defective_unit_num)),np.ones((
   defective_unit_num)))
    sample = random.sample(range(125),5)
    for  i in sample:
        result = result or lot[i]
    if result:
        reject_times += 1

print(reject_times)
```

Listing 1: Question 1 Code

Result

I simulate the process 1000 times and get rejected 225 times, in conclusion:

The probability of rejection if 5 microchips are tested is 22.5%

In probability theory, if we choose k microchips from the 125 ones, the probability of test failure is equal to:

$$p = 1 - \frac{C_{119}^{k}}{C_{125}^{k}}$$

After several trials, I find that $k = 49$ is the smallest number of microchips that the distributor should test to reject this lot 95% of the time.

Question 2

Experiment

The question can be divided into two parts. In the first part, I will simulate the process using binomial random variable while in the second I will generate a Poisson random variable. In the first part, split an hour into 4000 time intervals and I can get the probability that one car arrive in the interval, which is a Bernoulli trial. Compute and sum the intervals to simulate the hour process. In the second part, I will generate the Poisson random variable with inverse transform method. Compute the theoretical pmf of each distribution and plot on the figure.

Code

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import comb,perm
import math

simu_times = 1000
lmbda = 120
n = 4000
p = lmbda/n
sample_B = []
sample_P = []
pmf_poisson = []
pmf_x = np.arange(85,156,1,dtype=float)

for trial in range(simu_times):
    u = np.random.rand(1,n)
    bernoulli = u<p
    sample_B.append(np.sum(bernoulli))

pmf_binomial = [comb(n,k)*pow(p,k)*pow(1-p,n-k) for k in pmf_x]
pmf_poisson = [pow(lmbda,k)*7.66*10**(-53)/perm(k,k) for k in pmf_x]
# print(pmf_poisson)

fig,ax = plt.subplots(1,2)
ax[0].hist(sample_B,rwidth=0.9,edgecolor='black',alpha=0.8)
# ax[0].step(pmf_x,pmf_binomial,'r')
ax[0].set_title('Histogram of Poisson likely Binomial',fontsize=15)
ax[0].set_ylabel('Count',fontsize=12)

ax0 = ax[0].twinx()
ax0.step(pmf_x,pmf_binomial,'r')
ax0.set_ylabel('Probability',fontsize=12)

sample_P = np.random.poisson(lmbda,simu_times)

ax[1].hist(sample_P,rwidth=0.9,edgecolor='black',alpha=0.8)
```

```
# ax[1].step(pmf_x,pmf_poisson,'r')
ax[1].set_title('Histogram of Poisson variable',fontsize=15)
ax[1].set_ylabel('Count',fontsize=12)

ax1 = ax[1].twinx()
ax1.step(pmf_x,pmf_poisson,'r')
ax1.set_ylabel('Probablity',fontsize=12)

plt.show()
```

Listing 2: Question 2 Code

Result

To compare the histogram of the Poisson random variable and the summation of Bernoulli trail, I plot the subplots in one figure.

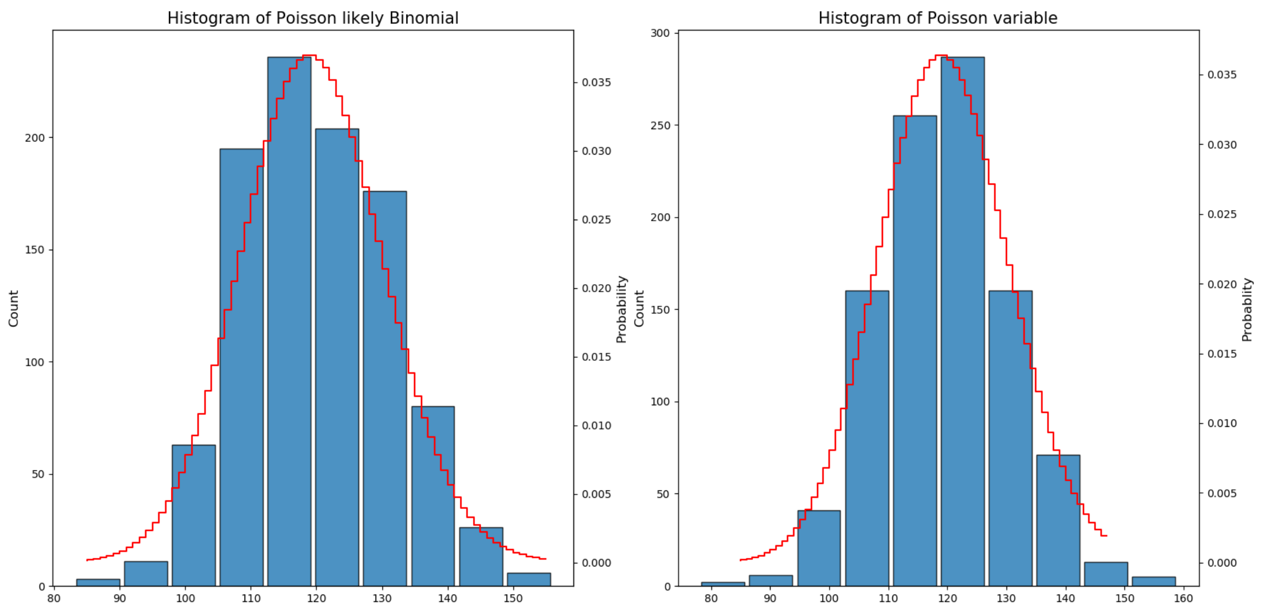

Figure 1: Comparison of two histogram

The left is Binomial and the right is Poisson. We can see that they share a similar shape both in pmf and sample histogram, and we have proved in the EE503 lecture that when n is very large and p is very small, and $\lambda = n * p$, the binomial is similar to Poisson distribution.

Question 3

Experiment

In this experiment, I will generate a sequence of uniform random variable until the summation is larger or equal than 4. The size of the sequence is the random variable N. Plot the histogram of N when simulating 100, 1000, 10000 times to complete the requirement.

Code

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

simu_num = 10000
boundary = 4
N = []

for i in range(simu_num):
    sample = []
    while(np.sum(sample)<boundary):
        sample.append(np.random.rand(1))

    N.append(len(sample))
# print(N)

num = np.unique(N)
plt.hist(N,rwidth=0.9,alpha=0.8,edgecolor='black',bins=range(min(num),max(
    num)+1))
plt.xlabel('Value of N',fontsize=12)
plt.ylabel('Count',fontsize=12)
plt.title('Histogram of N(Size=10000)')
plt.show()

Expectation = sum(N)/len(N)
print(Expectation)
```

Listing 3: Question 3 Code

Result

The histogram figures when the simulation times is 100, 1000, 10000:

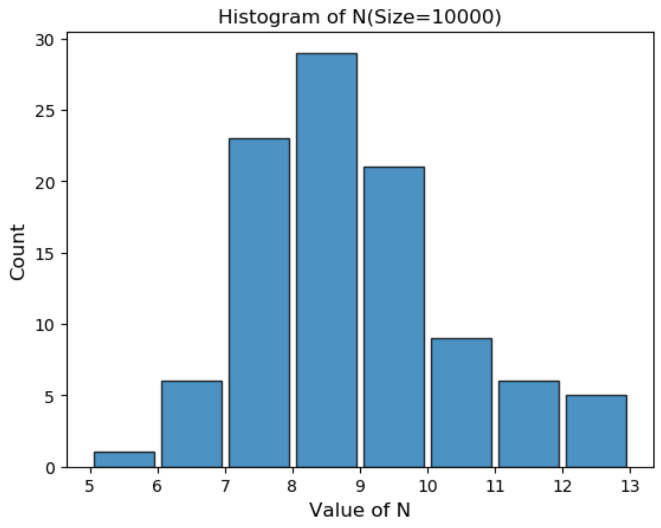

Figure 2: Histogram of N(size=100)

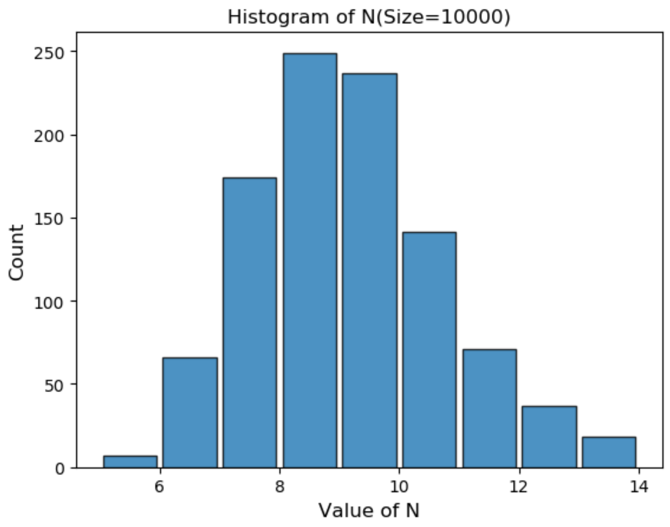

Figure 3: Histogram of N(size=1000)

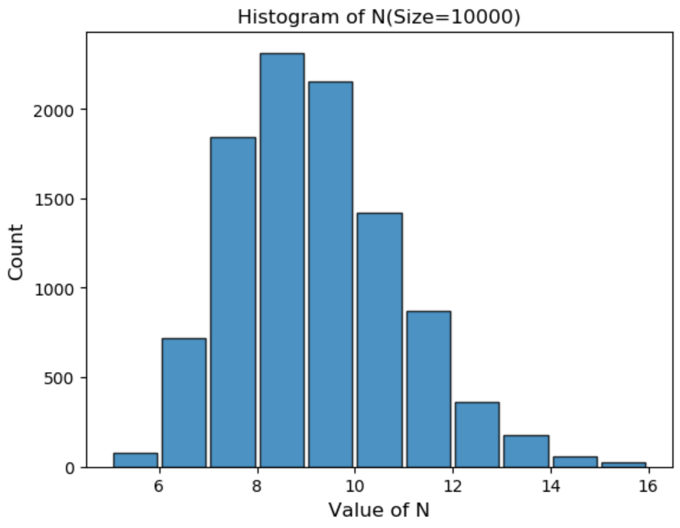

Figure 4: Histogram of N(size=10000)

We can compute the sample mean of N with the formula:

$$\bar{X}_n = \frac{1}{n}\sum_{k=1}^{n} k$$

and get $E[N] = 8.66$.

In probability theory, the random variable N is independent of X_k, and X_k are random, which is called doubly random sum. The expectation of doubly random sum is :

$$E[\sum_{k=1}^{N} X_k] = E_N[N] * \mu_x$$

Question 4

Experiment

The question 4 can be divided into two parts. In the first part, I will create a random sequence X_k with pmf $p_j = \frac{p}{j}$ for $j = 1, 2, ..., 60$. The constant p can be an arbitrary real number, and I set p as 10. Although the sum of weight $\frac{p}{j}$ is not equal to 1, I can still use the inverse transform method in a general way. Generate an uniform number u range from zero to total weight, and find the first number that summation of the weight is larger than u.

The second part is to generate a random variable $N_j = min\{k : X_k = j\}$ and simulate N_{60}. I will generate a sequence and find the index of the first 60 in the sequence. Plot histogram of the both parts to show the result. Finally calculate the estimator of expectation and variance of the N_{60} according to the formula:

$$\bar{X}_n = \frac{1}{n}\sum_{k=1}^{n} k$$

$$S_x^2 = \frac{1}{n-1}\sum_{k=1}^{n}(X_k - \bar{X}_n)^2$$

Code

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

ran_sequence_num = 1000
simu_num = 1000
j = 60
ran_sequence = []
N = []
p=10 #OR any constant number
weight = [p/i for i in range(1,61)]
total_weight = sum(weight)

#Generate random sequence Xk
for i in range(ran_sequence_num):
    u = random.uniform(0,total_weight)
    ran = 1
    while (sum(weight[0:ran])<=u):
        ran+=1
    ran_sequence.append(ran)

plt.figure(1)
plt.hist(ran_sequence,rwidth=0.9,alpha=0.8)
plt.title('Histogram of random sequence Xk',fontsize=15)
plt.xlabel('Random value',fontsize=12)
plt.ylabel('Count',fontsize=12)
# plt.show()

#Generate random variable Nj
for simu in range(simu_num):
    i = 0
    while(True):
        u = random.uniform(0,total_weight)
        ran = 1
        while (sum(weight[0:ran])<=u):
            ran+=1
        if (ran == j):
                N.append(i)
                break
            i+=1

plt.figure(2)
plt.hist(N,rwidth=0.9,alpha=0.8,bins=10)
plt.title('Histogram of random variable Nj',fontsize=15)
plt.xlabel('Random value',fontsize=12)
plt.ylabel('Count',fontsize=12)
plt.show()

#Calculate the sample expectation and variance
N = np.array(N)
expected_N = np.sum(N)/N.size
variance_N = np.sum(np.square(N-expected_N))/(N.size-1)
print(expected_N,variance_N)

#The random variable Nj is geometric
p60 = p/60/total_weight
threm_E = 1/p60
threm_V = (1-p60)/math.pow(p60,2)
print(threm_E,threm_V)
```

Listing 4: Question4 Code

Result

The histogram of the the random sequence X_k is:

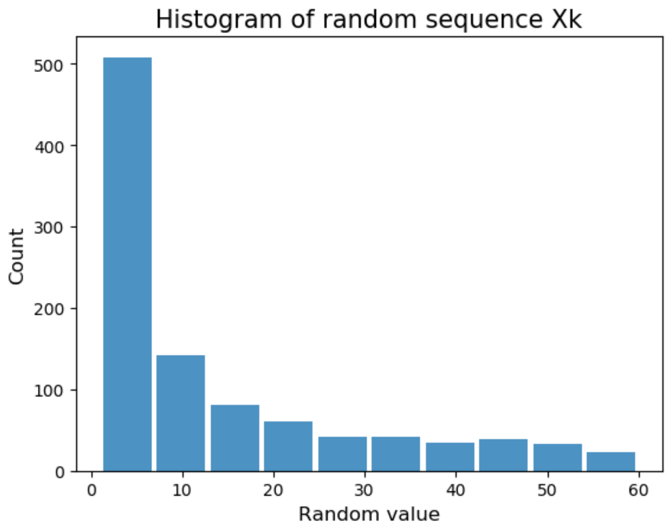

Figure 5: Histogram of random sequence X_k

The histogram of the random variable N_j is:

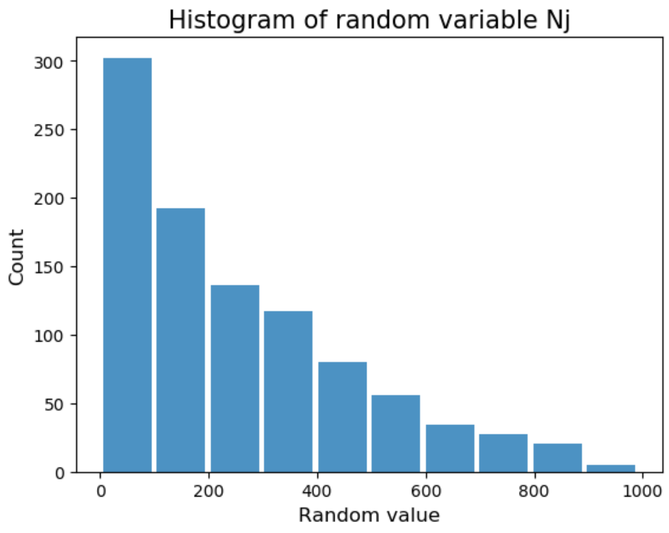

Figure 6: Histogram of random variable N_j

I find that N_{60} is a geometric random variable, and for each Bernoulli trial the success probability is p_{60}. In probability theory, we can get the theoretical expectation and variance value of N_{60}:

$$Totalweight = \sum_{j=1}^{60} \frac{p}{j}$$

$$p_{60} = \frac{p}{60 * Totalweight}$$

$$E[N_{60}] = \frac{1}{p_{60}}$$

$$V[N_{60}] = \frac{1 - p_{60}}{p_{60}^2}$$

Thus, I can get the theoretical and sample value of the expectation and variance.

	$E[N_{60}]$	$V[N_{60}]$
Sample	273.53	79528.72
Theory	280.79	78563.49

Table 1: Sample and Theoretical value of $E[N_{60}]$ $V[N_{60}]$

It is obvious that the estimated sample value and the theoretical value are close to each other. In consideration that the sample expectation and variance is unbiased, the simulation has shown the point and the value is right.

Question 5

Experiment

I will use accept-reject method to generate random variable p. Firstly, generate the given uniform distributed random variable q, then generate a uniform random variable u range in [0,1]. Compute the random variable $ratio = \frac{p_j}{c*q_j}$, and compare it with u. If ratio is smaller than u, accept the q as p, otherwise reject it and loop until find the required value. Simulate the all process and count the number of acceptance and rejection, and calculate the efficiency.

Code

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

pmf_q = [0.05]*20
pmf_p = 5*[0.06]+[0.15,0.13,0.14,0.15,0.13]+[0]*10
simu_times = 1000
c=3
ran_sequence = []

for simu in range(simu_times):
    u = np.random.rand()
    q = np.random.randint(1,21)
    ratio = pmf_p[q-1]/pmf_q[q-1]/c
    if (u<ratio):
        ran_sequence.append(q)

fig = plt.figure()
ax = fig.add_subplot(111)
ax.hist(ran_sequence,rwidth=0.9,alpha=0.8,bins=range(1,12))
ax.set_title('Histogram of random variable',fontsize=15)
ax.set_xlabel('Random variable value',fontsize=12)
ax.set_ylabel('Count',fontsize=12)

ax1 = ax.twinx()
ax1.step(range(1,12),[0.06]+pmf_p[0:10],'r')
ax1.set_ylabel('pmf',fontsize=12)
ax1.set_ylim([0,0.16])

plt.show()

ran_sequence = np.array(ran_sequence)
sample_mean = np.sum(ran_sequence)/ran_sequence.size
sample_variance = np.sum(np.square(ran_sequence-sample_mean))/(
    ran_sequence.size-1)
print(sample_mean,sample_variance)

expectation = 0
```

```
moment = 0
for i in range(10):
    expectation += pmf_p[i]*(i+1)
    moment += pmf_p[i]*(i+1)**2
varaince = moment-expectation**2
print(expectation,varaince)

sample_efficiency = len(ran_sequence)/simu_times
print(sample_efficiency)
```

Listing 5: Question 5 Code

Result

The histogram and the probability mass function(pmf) of the random variable is:

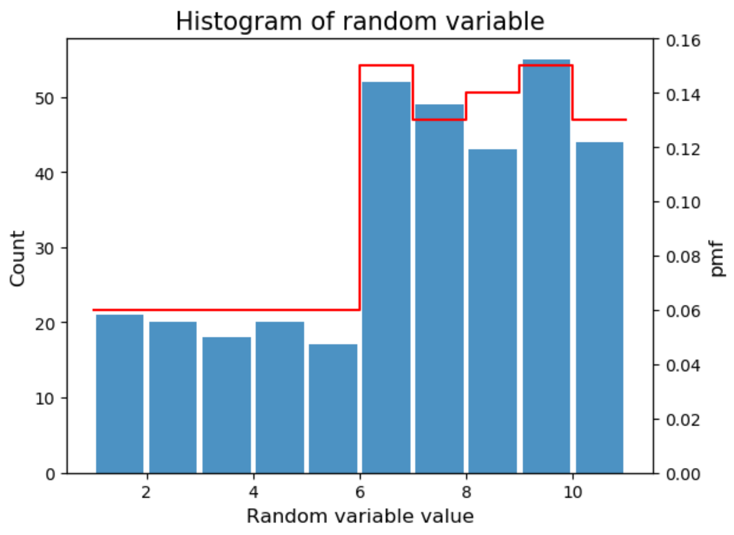

Figure 7: Histogram and pmf of random variable p

It is obvious that the pmf has the similar shape with the histogram, so the accept-reject method of generating random variable p meets the requirement. Compute the sample mean and sample variance using the formula:

$$\bar{X}_n = \frac{1}{n}\sum_{k=1}^{n} k$$

$$S_x^2 = \frac{1}{n-1}\sum_{k=1}^{n}(X_k - \bar{X}_n)^2$$

and compute the theoretical value of the expectation and variance of variable p using the formula:

$$E[X] = \sum_{k=1}^{n} p_k * x_k$$

$$E[X^2] = \sum_{k=1}^{n} p_k * x_k^2$$

$$V[X] = E[X^2] - E^2[X]$$

Thus, we can get the table of sample and theoretical value:

	$E[p_j]$	$V[p_j]$
Sample	6.25	7.26
Theory	6.48	7.19

Table 2: Sample and Theoretical value of $E[p_j]$ and $V[p_j]$

We can conclude that the theoretical value and the estimated sample value are very close, so the estimator is good.

We can also estimate the efficiency of the sampler using the formula:

$$\text{Effiency} = \frac{\#accepted}{\#accepted + \#rejected}$$

and compute the theoretical value according to the distribution. The total number N needed to get one success is a geometric random variable.

$$p_s = \frac{\sum_{k=1}^{8} p_k}{\sum_{k=1}^{20} c * q_k}$$

$$E[X] = \frac{1}{p_s}$$

$$\text{Effiency} = \frac{1}{E[X]}$$

The probability of success is $\frac{1}{3}$, so the $E[X] = 3$ and the theoretical efficiency is 33.3%. In conclusion

The theoretical effiency is 33.3%
The sample effiency is 32.1%

The estimated sample value is close to theoretical value, so the estimator is good, and the simulation is successful.

As said above, the number N needed to get one acceptance is a geometric random variable. The sum of $c*q_j$ is equal to c, and the sum of p_j is equal to 1. So the probability of acceptance is $p_a = \frac{1}{c}$ and the expectation of N is $E[N] = c$. Thus, the efficiency is $\frac{1}{c}$