**PROJECT #3 – SAMPLE & STATISTICS**

**Divyanshu Sahay  
(dsahay@usc.edu)**

Important Note:

Tool Used: **Python 3.7.4.**  
Editor Used: Visual Studio Code.

1. **Simulate Sampling from a lot:**

**Problem Statement:** A batch of 125 microchips are delivered, out of which 6 are defective. A sample of five is taken out without replacement for conformance. The lot is rejected even if one of the microchip is bad in the sample of 5. Following are the requirements:

* Simulate lot sampling to estimate the probability that the lot is rejected.
* Fewest number of chips to be selected to reject the lot 95% of the times.

Solution:

**Understanding the problem statement:** This is an exampling of sampling without replacement which can be treated as hypergeometric distribution.

**Methodology:** The presence of a defected microchip is indicated by a ‘0’ and a ‘1’ indicates a non-defective microchip. The program uses an in built command *random.shuffle* to simulate different arrangements of a lot. To sample, chips at random without replacement the *sample* command is used. Every rejected chip is recorded. The probability that a chip is rejected is then calculated and recorded. This entire procedure is repeated 10 times to get a reliable idea of probability of rejection.

**Program Written**:

import numpy as np

import scipy.stats as stats

import matplotlib.pyplot as plt

from random import sample

n = 0

Lot = []

rejected = 0

prob\_reject\_list = []

for i in range(124):

    if (i < 6):

        Lot.append(0) # 0 DEPICTS A DEFECTIVE CHIP

    else:

        Lot.append(1) # 1 DEPICTS A DEFECTIVE CHIP

total\_trial = 10000

for i in range(10):

    np.random.shuffle(Lot) #to simulate randomly placed microschips

    prob\_reject = 0

    rejected = 0

    for i in range(total\_trial):

        sample\_check = sample(Lot,5)

        for i in range(0,len(sample\_check)):

            if (sample\_check[i] == 0):

                rejected = rejected + 1

                break

            else:

                pass

    prob\_reject = rejected / total\_trial # the probability of rejection

    prob\_reject\_list.append(prob\_reject)

print(prob\_reject\_list)

prob\_reject = rejected / total\_trial

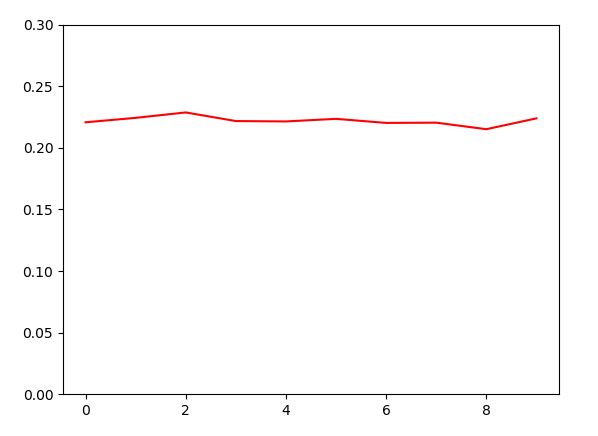
fig = plt.figure()

plt.ylim(0, 0.3)

plt.plot(prob\_reject\_list,"r-")

plt.show()

**Results and Observations:**

****

Probabilities

Attempts

Fig.1. Variation of Probability of Acceptance

For all the 10 attempts, approximately, the Probability of Rejection was obtained near to 0.22 as shown in Fig. 1. This was verified with the theoretical value as follows:

**2nd Part:** In the 2nd part it was asked to calculate the number of samples that must be selected so that the Lot is rejected 95% of the times. Thismeansthe . This problem statement was dealt with the method of hit and trail; by substituting different valued of ***n*** in the following equation.

A small program was written to find n (the function **ncr** is an adaptation from an open source code)

import operator as op

from functools import reduce

def ncr(n, r):

    r = min(r, n-r)

    numer = reduce(op.mul, range(n, n-r, -1), 1)

    denom = reduce(op.mul, range(1, r+1), 1)

    return numer / denom

n = 5

i = 0

while 1:

  i = ncr(119,n)/ncr(125,n)

  n = n + 1

  if (i <= 0.05):

      break

print('----value of n----')

print(n)

print('---- rejection ---')

print(1-i)

print('---- acceptance ---')

print(i)

----value of n----

50

---- rejection ---

0.9533923695251305

---- acceptance ---

0.0466076304748695

Fig. 2. Output Screen

It can be seen that for value of n being 50, the is greater than 0.95. So, a value of   
**n = 49** was selected. Hence, the sample size should be 49 to reject the samples 95% of the times.

1. **Poisson Process:**

**Problem Statement**: Arrival of 120 cars per hour at a freeway onramp has to be simulated. Following are the requirements.

* One hour of arrival has to be simulated for smaller intervals (< 1 second) where the arrival of car is indicated by a Bernoulli Trial. The corresponding histogram should be generated for number of arrivals per hour.
* The sampling experiment has to be repeated by sampling from an *equivalent* Poisson Distribution using Inverse Transform Method.

**Understanding the problem statement:** This problem is a classic example of a Poisson process where we are interested in counting the occurrence of an event in a given time interval. The basic need of treating a process as a Poisson’s is providing more granular to the assessment of an event. The given problem asks to examine the arrival of 120 cars in a duration of 1 hour. It is trivial to visualize that the process cannot be assessed efficiently if the unit measurement for sampling the number of cars is a minute. So, when we try to analyse this event for a larger sample, thereby making the unit of measurement much smaller than a minute, then the system can be modelled as a Poisson Process. The probability mass function is given as:

Here, is the number of success and k is the number of trials.

**Methodology (1a)**: In this part the arrival of a car is required to be treated as a Bernoulli outcome where a ‘1’ would indicate the arrival. In this problem 5000 units of time in an hour is considered to sample the event. A random number generator generates a 5000 samples of numbers between 0 and 1. These numbers can be treated as the probability of arrival of a car and can be compared with (120/5000) to decide whether a car has arrived or not.

**Program Written**:

import numpy as np

import random

import scipy.stats as stats

import matplotlib.pyplot as plt

from scipy.stats import poisson

p = 120/5000

arrival = []

arrival\_list = []

for i in range(1000):

    arrival\_sum = 0

    arrival = []

    random\_prob =  np.random.rand(5000) # generating random number in (0,1)

    for i in range(5000):

        if( p >= random\_prob[i]):

            arrival.append(1) # 1 indicated arrival of a car

    arrival\_sum = sum(arrival)

    arrival\_list.append(arrival\_sum)

print(arrival\_list)

fig = plt.figure()

plt.figure(figsize=(10, 3))

x= np.arange(50,200)

scale = poisson.pmf(x,120) # the theoretical pmf

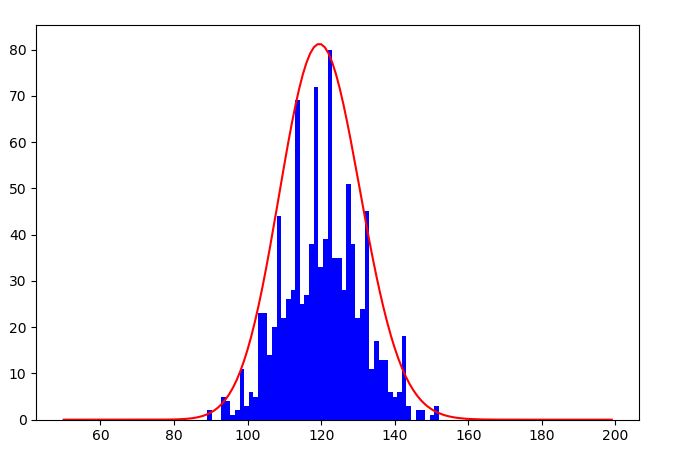
plt.plot(x, scale\*2230, "r-")

plt.hist(arrival\_list,50,facecolor='b', alpha=1)

plt.show()

**Results and Observations:**

It can be seen in the output (Fig.3) that the mean value for the different samples obtained over 1000 samples is 120. Also the theoretical Poisson distribution (in red) when overlayed over the histogram, perfectly fits it.

****Fig. 3. Histogram overlayed with Poisson Distribution.

Frequency of Occurrence in Samples

Number of Cars

**Methodology (1b)**: In this part the same process has to be repeated by equivalent Poisson distribution by using inverse transform method. The steps to be taken were discussed in the class and is reproduced here:

1. Generate X ~U (0,1)
2. When i = 0, p = , F = P
3. If U < F, X = i, Stop
4. P = / i +1, F = F+P, i = i+1
5. Go to 3.

**Program Written**:

import math

import numpy as np

import random

import scipy.stats as stats

import matplotlib.pyplot as plt

from scipy.stats import poisson

sampled = []

lam\_da = 120

for j in range(1000):

    i = 0

    p = math.exp(-lam\_da)

    F = p

    random\_prob = np.random.rand(1)

    while(random\_prob >= F):

        i = i + 1

        p = (lam\_da \* p)/(i)

        F = F + p

    sampled.append(i)

print(sampled)

fig = plt.figure()

plt.figure(figsize=(10, 3))

x= np.arange(50,200)

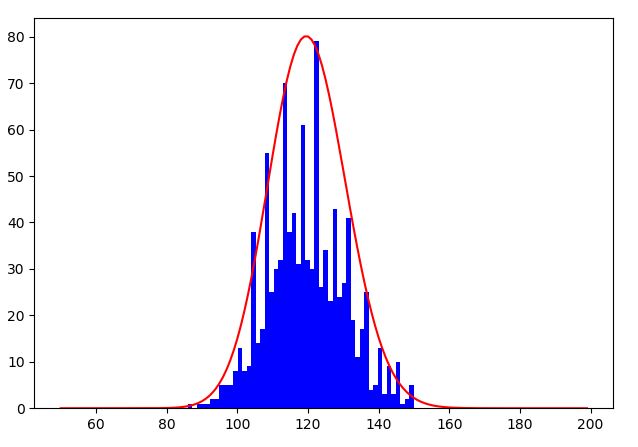
scale = poisson.pmf(x,120)

plt.plot(x, scale\*2200, "r-")

plt.hist(sampled,50,facecolor='b', alpha=1)

plt.show()

**Results and Observations:**

****Fig. 4. Histogram (inverse transform method) overlayed with Poisson Distribution.

Frequency of Occurrence in Samples

Number of Cars

When overlayed with the theoretical Poisson distribution the Histogram perfectly fits. Also, like before, congruent to our expectation, the mean value is 120.

1. **Uniform Random Sampling**

**Problem Statement:** A random variable has to be defined as the minimum number of uniform random samples (0,1) such that the sum of these random samples is greater than 4. Following are the requirements:

* Generate histogram for 100, 1000, and 10000 samples of the random variable.
* Comment has to be made on ***E[N].***

**Methodology**: The implementation for this question is straight forward. A random number is generated between 0 and 1 (as uniform random samples), every time the number is generated the it is summed with the previous number. As soon as the sum reaches 4 we stop the process. This is for 100, 1000 and 10000 samples and the expectation is calculated.

**Program Written:**

import numpy as np

import scipy.stats as stats

import matplotlib.pyplot as plt

from random import sample

import sys

sum\_N = 0

n=0

min=sys.maxsize

range1 = 100

range2 = 1000

range3 = 10000

list1=[]

for i in range(range3):

    n=0

    sum\_N=0

    while 1:

        value = np.random.rand(1)

        sum\_N = sum\_N + value

        n = n + 1

        if (sum\_N > 4):

            break

    list1.append(n)

if (n < min): # calculating the minimum value

        min = n

print(min)

print(list1)

avg = sum(list1)/len(list1)

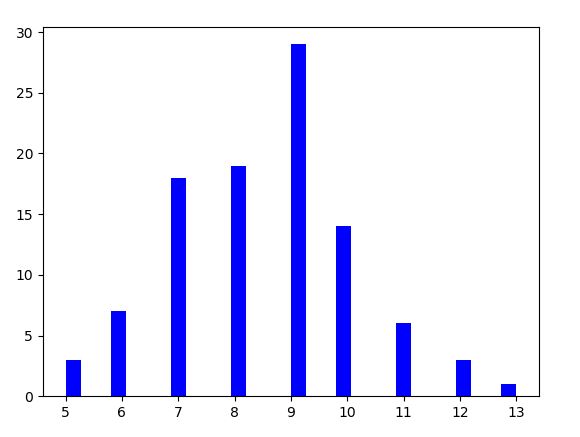
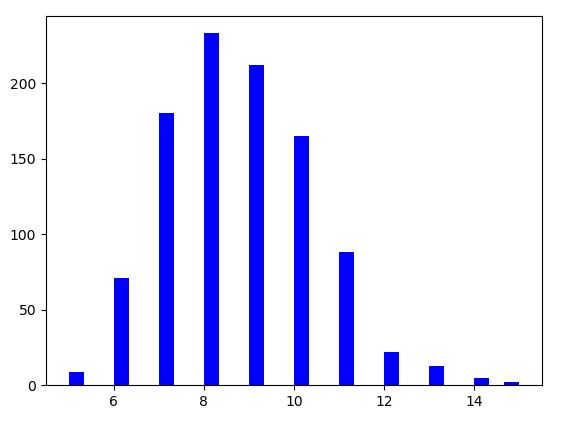
print(avg)

fig = plt.figure()

plt.hist(list1,40,facecolor='b', alpha=1)

plt.show()

**Results and Observation:**

**** 

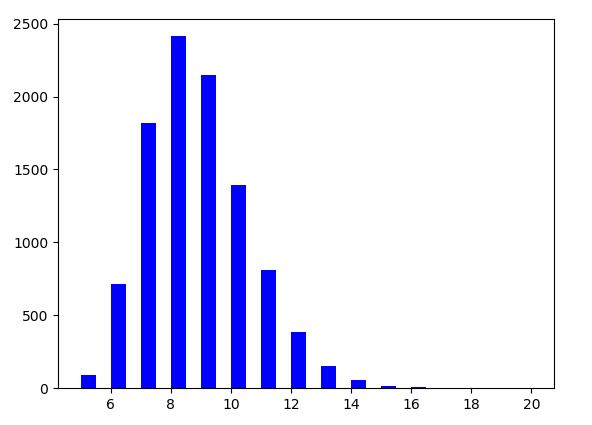
Number of Random Samples

Number of Random Samples

Frequency of Occurrence in Samples

Frequency of Occurrence in Samples

Fig. 5. The Histogram for 100 (LEFT) and 1000 (RIGHT) samples.

  
Fig. 6. The Histogram for 1000 samples.

Number of Random Samples

Frequency of Occurrence in Samples

It can be seen in the histogram that the Expectations obtained for the samples are as follows:

|  |  |
| --- | --- |
| Samples | Expectation |
| 100 | 8.69 |
| 1000 | 8.63 |
| 10000 | 8.67 |

The expectations obtained in all the cases is slighlty greater than 8. As the numbers in (0,1) is uniformly distributed, it can be assumed to that if mean of those numbers, 0.5, is the number which is obtained on every random number generation, we woud need 8 numbers to sum to 8. Hence, the obtained expectation is in congruence with what we would epxect theoretically.

1. **Sequence Generation**

**Problem Statement:** This problem expects us to generate a sequence with following properties:

* The probability of occurrence of each element is given by where j takes value from 1 to 60. So, the probability of occurrence of 60 will be least and 1 the most.
* Further, the question asks us to define a random variable , thereby perform sampling for
* Finally, its required to calculate and .

**Understanding the problem:** It can be deduced from the 1st bullet that, . This can help to find the value of p. Now to generate the sequence as the specified probability densities, we need to use inverse transform method which was discussed in class. This can be repeated several times to calculate the occurrence of 60. For , in every sequence the first occurrence of 60 (minimum in ) is recorded. It can be observed that this probability distribution is taking the shape of a geometric distribution as we need to search until we get a 60. Thus, the theoretical expectation and variance is given as follows:

**Methodology**: The methodology was explained in the understanding part. The code itself has comments which makes the methods adopted evident.

**Program Written:**

import numpy as np

import scipy.stats as stats

import matplotlib.pyplot as plt

from random import sample

import statistics

sum\_60 = 0

for i in range(61):

    if(i==0):

        pass

    else:

        sum\_60 = sum\_60 + (1/i)

p = 1 / sum\_60        # calculating the value of p

print(sum\_60)

print(p)

sum\_prob\_list = []

sum\_prob = 0

X\_k = []

N\_60 = 0

N\_60\_list = []

Expect = 0

for i in range(60):

    sum\_prob = sum\_prob + p/(i+1)

    sum\_prob\_list.append(sum\_prob)

print(sum\_prob\_list) # calculating the vector with specified probabilities

for i in range(1000):#peforming the experiment 1000 times

    X\_k = []

    N\_60 = 0

    for i in range(1000): # creating a sequence of 1000 numbers

        u = np.random.rand(1)

        for i in range(len(sum\_prob\_list)):

            if (u < sum\_prob\_list[i]): #performing the discrete inverde transform method to generate the samples.

                X\_k.append(i+1)

                break

    for i in range(len(X\_k)):

        if (X\_k[i] == 60): #checking for the occurrane of 60

            N\_60 = i

            break

    N\_60\_list.append(N\_60)

print(N\_60\_list) #the list with the numbers of 60 in each sequence generated

MEAN = statistics.mean(N\_60\_list)

VARIANCE = statistics.variance(N\_60\_list)

print('---MEAN---')

print(MEAN)

print('---VARIANCE---')

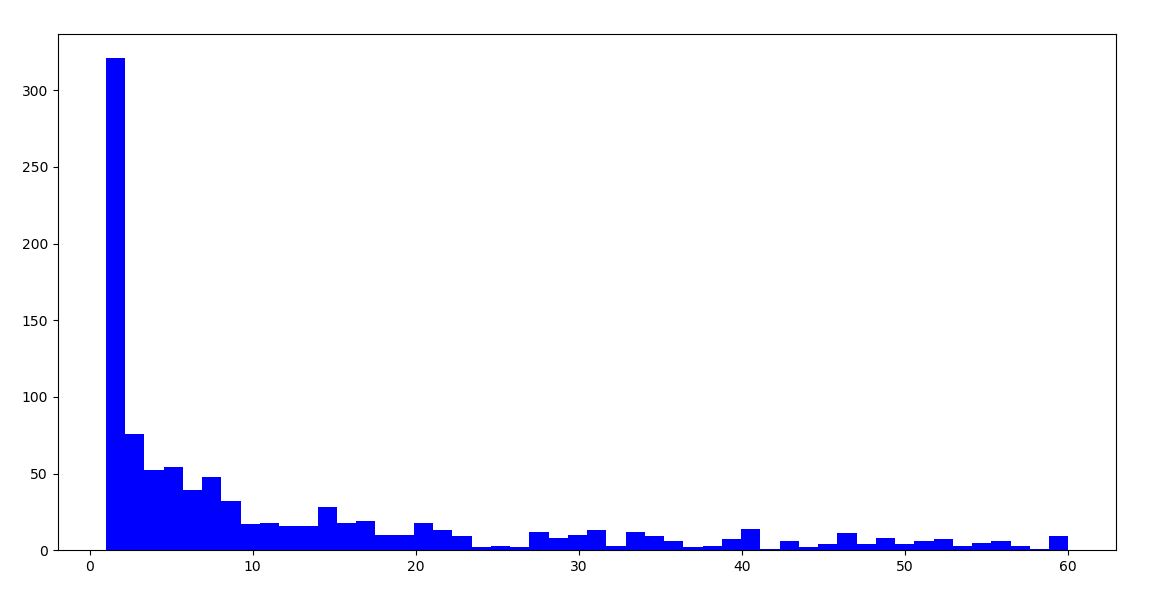
print(VARIANCE)

fig = plt.figure()

plt.hist(X\_k,50,facecolor='b', alpha=1)

plt.show()

**Results and Observation:**

  
Fig.7. The Histogram of Sequence (1000)

Numbers

Frequency of Occurrence in Samples

The histogram is in congruence with our expectation. The occurrence of 1 is the largest the occurrence of 60 is least. The expectation and variance values obtained are as follows:

**Output Screen:**

---MEAN---

273.965

---VARIANCE---

73358.59036536536

The values are almost same as the ones theoretically obtained.

1. **Accept-Reject Sampling**

**Problem Statement:** This problem requires to sample a given distribution ***pj*** by sampling from another auxiliary sequence ***qj*** by using accept-reject method. This method is useful in cases where it is difficult to sample a given probability distribution. In such scenarios another probability distribution (called auxiliary here) can be used to estimate the probability distribution. The algorithm is as follows:

If we wish to obtain samples from a distribution ***X*** ***f(X),*** which is hard to sample. Then,

1. Sample a proposal distribution ***Y*** ***g(X)*** and ***U Unif(0,1)***
2. Accept ***X = Y*** if ***U* ≤ *f(Y)/C\* g(Y)***, otherwise return to 1, where ***C \**** ***g(X) ≥ f(X)***

**Methodology**: The Methodology is just the adaptation of what has been explained in the previous section. Also, the comments provide a detailed explanation on the methodology.

**Program Written:**

import random

import numpy as np

import scipy.stats as stats

import matplotlib.pyplot as plt

from random import sample

from matplotlib import pyplot

import statistics

import collections

C = 0.15/0.05 #maximum value of the target

accepted = 0

rejected = 0

sample\_data = []

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,]  #extra zeros added added to accoujt for the size of q

p\_plot  = [0,0.06,0.06,0.06,0.06,0.06,0.15,0.13,0.14,0.15,0.13,0,0,0,0,0,0,0,0,0] #to plot the first value is taken as zero as the first value doesn't get printed.

I = np.ones(20)

q = 0.05\*I

# print(p)

# print(q)

for i in range(100000):

    Y= random.randrange(1,21)

    value\_test = C\*np.random.rand(1)

    if(value\_test <= p[Y-1]/0.05): # the algorithm

        sample\_data.append(Y)

        accepted = accepted + 1

    else:

        rejected = rejected + 1

efficiency = accepted / (accepted + rejected)

print('---efficiency---')

print(efficiency)

print('--calculated---')

print(1/C)

print(sample\_data)

value = []

freqList = (collections.Counter(sample\_data)) #the disctionary that stores the frequency of occurrence of numbers

print(freqList)

denom = (len(sample\_data))

for i in range(1, (len(freqList)+1)):

    value.append(freqList[i]/denom) #calculating the Probabilities

print('---value---')

print(value)

x = [1,2,3,4,5,6,7,8,9,10]

y = value

print('---x---')

print(x)

plt.figure(figsize=(10, 3))

plt.bar(x,y,align='center') # A bar chart

plt.plot(p\_plot,"r-")

plt.xlabel('Numbers')

plt.ylabel('Probabilities')

plt.show()

#calculating the sample means and variance

sam\_mean = statistics.mean(sample\_data)

sam\_variance = statistics.variance(sample\_data)

sam\_mean = statistics.mean(sample\_data)

print('---sample mean---')

print(sam\_mean)

sam\_variance = statistics.variance(sample\_data)

print('---sample variance---')

print(sam\_variance)

#calculating the theoretical means and variance

theo\_mean = 0

theo\_var = 0

p\_cal = [0.06,0.06,0.06,0.06,0.06,0.15,0.13,0.14,0.15,0.13]

for i in range(len(p\_cal)):

    theo\_mean = theo\_mean + (i+1)\*p\_cal[i]

for i in range(len(p\_cal)):

    theo\_var = theo\_var + (p\_cal[i]\*((i+1) - theo\_mean)\*((i+1) - theo\_mean))

print('--theoretical mean---')

print(theo\_mean)

print('--theoretical variance---')

print(theo\_var)

fig = plt.figure()

plt.hist(p\_plot,30,facecolor='b', alpha=1)

plt.hist(sample\_data,30,facecolor='r', alpha=1)

plt.show()

**Output Screen:**

---sample mean---

6.486702047914605

---sample variance---

7.231959590952028

--theoretical mean---

6.4799999999999995

--theoretical variance---

7.1895999999999995

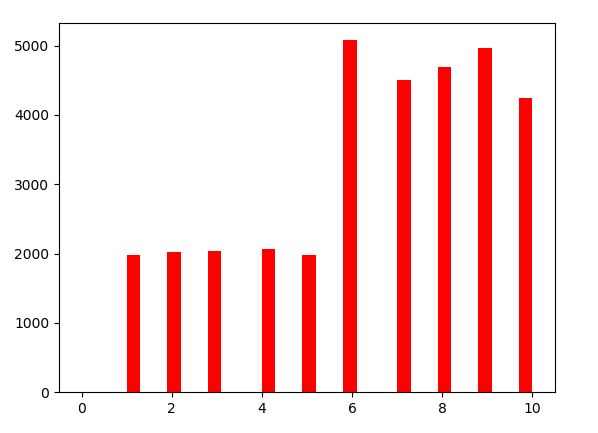
---efficiency---

0.33351

--calculated---

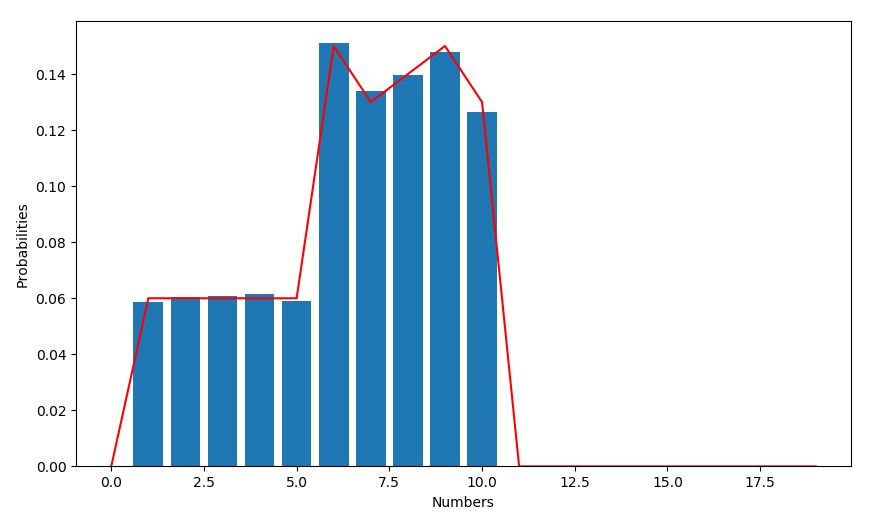
0.33333333333333337

**Results and Observation:**

****Fig. 8.The Frequency of occurance of the numbers

Numbers

Frequency of Occurrence in Samples

****Fig. 8.The histogram generated by accept-reject with overlayed target distribution

There are two histograms shown here. Fig. 7 shows the distributoin of numbers when we sample for 100000 times. The histogram generated by the accpet-reject sampling fits on the target distribution perfectly. Hence we can conclude that accept-reject sampling is reliable.

The sample mean and sample variance values are 6.48670 and 7.23195 respectively, which is very close to the theoretical values of 6.4799 and 7.189. Futhermore, the theoretically calculated efficiency and the one obtained by our experiments are almost same. We can happily conclude that Accept-Reject method does wonders.