Sina Mahbobi

February 12, 2020

# 1 EE511 Project 3

### 1.0.1 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% of the time?

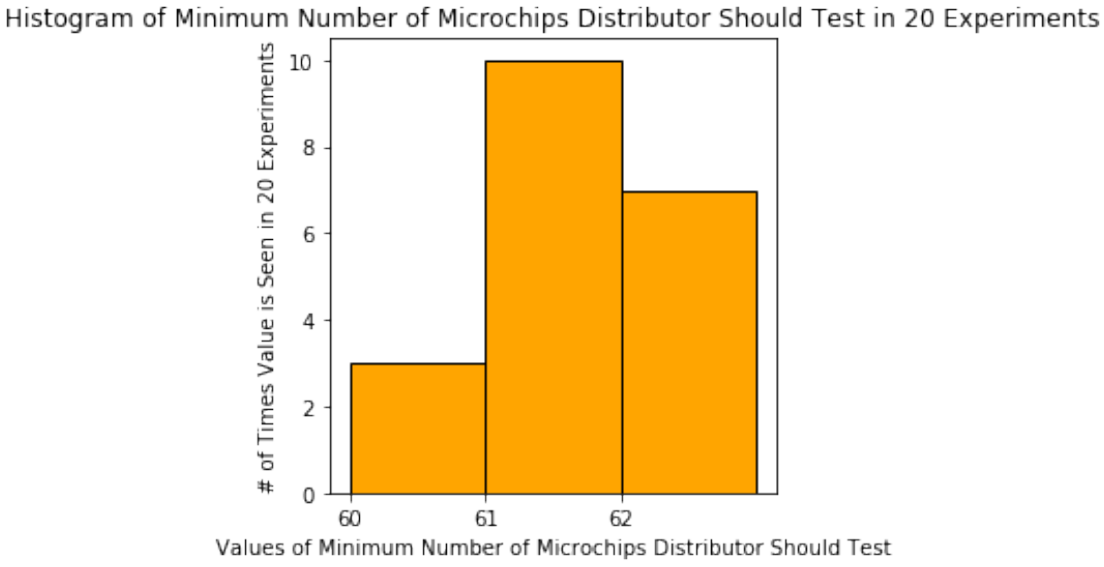
In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [7]:
```
good_chips = np.ones(119) # ones indicate 119 good chips out of 125
bad_chips = np.zeros(6)   # zeros indicate 6 defective chips out of 125
chips = np.concatenate((good_chips, bad_chips)) # concatenate for total chips

reject_count = 0;


for i in range(0,10000): #100000 samples to estimate probability
    chip_sample = np.random.choice(chips, 5)
    #print(chip_sample)
    if (0 in chip_sample):
        reject_count += 1


estimated_reject = reject_count/10000
```

- The estimated probability that the the distributor will reject the lot if it tests five microchips is: 0.2068

- Now, we need to find the fewest number of microchips that the distributor should test to reject the lot 95% of the time.

```
In [8]: estimated_arr = []
        for k in range(0,20):
            for i in range(1,126):    # testing sampling values from 1 to 125
                reject_count = 0
                estimated_rejection = 0

                for j in range(0,10000) : #10000 samples for each sampling length
                    chip_sample = np.random.choice(chips, i)
                    #print(chip_sample)
                    if (0 in chip_sample):
                        reject_count += 1
                estimated_rejection = reject_count/10000
                if (estimated_rejection >= 0.95):
                    sample_value = i
                    estimated_arr.append(sample_value)
                    break

        plt.hist(estimated_arr, bins = 3, edgecolor = 'black', facecolor = 'orange')
        plt.xlabel("Values of Minimum Number of Microchips Distributor Should Test")
        plt.xticks(np.arange(60,63))
        plt.ylabel("# of Times Value is Seen in 20 Experiments")
        plt.title("Histogram of Microchips Distributor Should Test in 20 Experiments")
        plt.tight_layout()
        plt.show()
        plt.savefig("Project1_Q1.png")
```



```
<Figure size 432x288 with 0 Axes>
```

**Q1 Analysis**

- The estimated probability that the the distributor will reject the lot if it tests five microchips is: 0.2166. This experiment was done with 10000 samples, and the number of rejections divided by the total number of samples (10000) was used to experimentally estimate this probability.
- The above graph is a histogram of the minimum number of microchips the distributor should test for 20 different experiments.
  - For each experiment, at each value of n, where n equals the number of chips selected from the 125, n chips were selected 10000 times to estimate the probability. This was done at every value of n, and this experiment was done 20 times.
  - In 20 experiments the values were either 60,61, or 62, depending on the experiment. These values are the fewest number of microchips that the distributor should reject 95% of the time for each experiment.

In [ ]:

### 1.0.2 Question 2

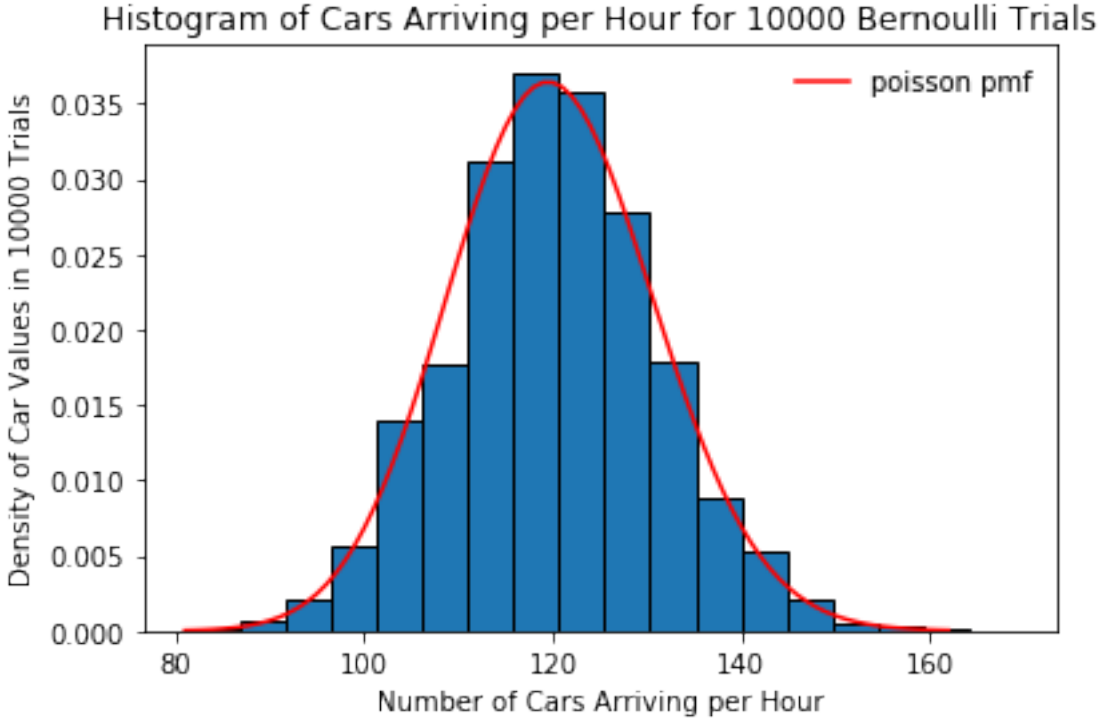
Suppose that 120 cars arrive at a freeway onramp per hour on average. Simulate one hour of arrivals to the freeway onramp: (1) subdivide the hour into small time intervals (< 1 second) and then (2) perform a Bernoulli trial to indicate a car arrival within each small time [U+2010] interval. Generate 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.

In [9]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import poisson
import math
```

In [ ]:

In [29]:
```
sub_intervals = 60 * 60 * 20 # intervals of .05 seconds
lmbda = 120
p = lmbda/sub_intervals;
bernoulli_samples = []
for i in range(0,10000): # 10000 Bernoulli Trials
    u = np.random.rand(sub_intervals,1);
    bernoulliTrials = u<p
    x = np.sum(bernoulliTrials); # This is a sample of Poisson(120), 1 Bernoulli Trial
    bernoulli_samples.append(x)

num_bins = 18
```

```
fig, ax = plt.subplots()
# the histogram of the data
n, bins, patches = ax.hist(bernoulli_samples, num_bins, edgecolor = 'black', density =


# adding the poisson pmf line
mean, var, skew, kurt = poisson.stats(lmbda, moments='mvsk')
x = np.arange(poisson.ppf(0.0001, lmbda), poisson.ppf(0.9999, lmbda))
ax.plot(x, poisson.pmf(x, lmbda), ms=8, label='poisson pmf', color = 'red')
ax.legend(loc='best', frameon=False)


plt.xlabel("Number of Cars Arriving per Hour")
plt.ylabel("Density of Car Values in 10000 Trials")
plt.title("Histogram of Cars Arriving per Hour for 10000 Bernoulli Trials")
fig.tight_layout()
plt.show()
print("The mean of the poisson pmf: ", str(mean))
print("The variance of the poisson pmf: ", str(var))
bernoulli_mean = sum(bernoulli_samples)/len(bernoulli_samples)
bernoulli_variance = sum((i - bernoulli_mean) ** 2 for i in bernoulli_samples) / len(be

print("The mean of the Bernoulli trials: ", str(bernoulli_mean))
print("The variance of the Bernoulli trials: ", str(bernoulli_variance))
plt.savefig("Project3_Q2_1.png")
```

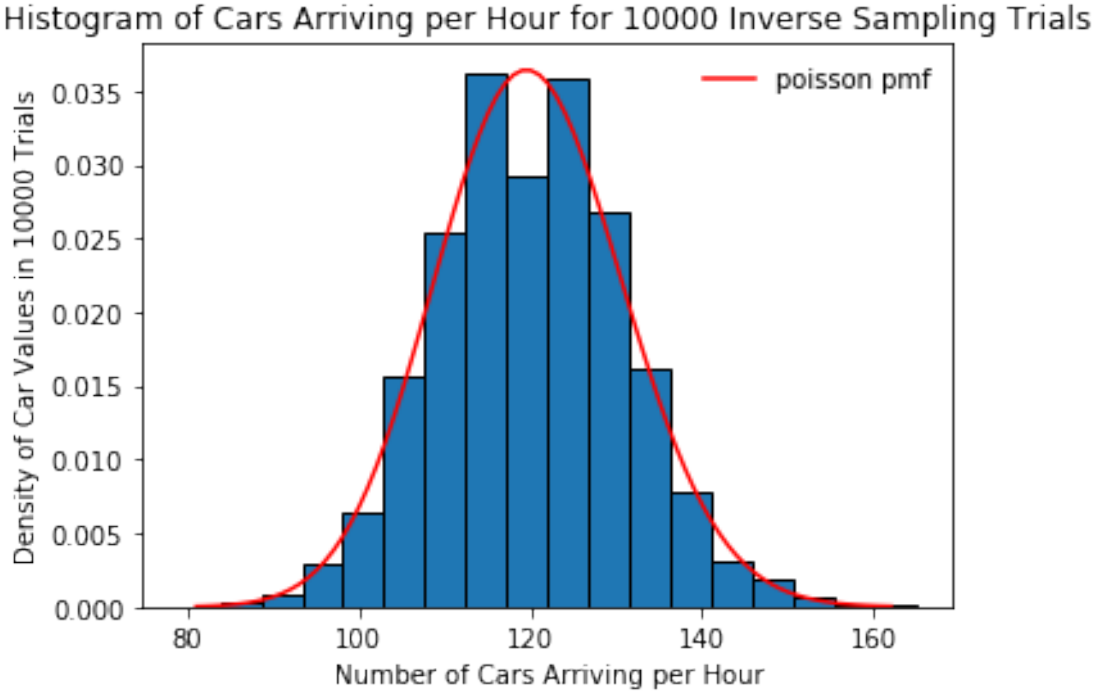


```
The mean of the poisson pmf:  120.0
The variance of the poisson pmf:  120.0
The mean of the Bernoulli trials:  120.0069
The variance of the Bernoulli trials:  121.22265239000059


<Figure size 432x288 with 0 Axes>
```

- Now, in this section below the counting experiment is repeated by sampling directly from an equivalent Poisson distribution by using the inverse transform method

```
In [28]: lmbda = 120
         inverse_samples = []

         for i in range(0,10000): # 10000 inverse transform simulations
             j = 0
             p = math.exp(-1 *lmbda)
             inverse_sum = p
             uniform = np.random.uniform(0,1)

             while uniform > inverse_sum:
                 p = p * lmbda / (j+1)
                 inverse_sum += p
                 j +=1
             inverse_samples.append(j)
         #print(inverse_samples)

         num_bins = 18
         fig, ax = plt.subplots()
         # the histogram of the data
         n, bins, patches = ax.hist(inverse_samples, num_bins, edgecolor = 'black', density = Tr


         # adding the poisson pmf line
         mean, var, skew, kurt = poisson.stats(lmbda, moments='mvsk')
         x = np.arange(poisson.ppf(0.0001, lmbda), poisson.ppf(0.9999, lmbda))
         ax.plot(x, poisson.pmf(x, lmbda), ms=8, label='poisson pmf', color = 'red')
         ax.legend(loc='best', frameon=False)


         plt.xlabel("Number of Cars Arriving per Hour")
         plt.ylabel("Density of Car Values in 10000 Trials")
         plt.title("Histogram of Cars Arriving per Hour for 10000 Inverse Sampling Trials")
         fig.tight_layout()
         plt.show()
         print("The mean of the poisson pmf: ", str(mean))
```

```
print("The variance of the poisson pmf: ", str(var))
inverse_mean = sum(inverse_samples)/len(inverse_samples)
inverse_variance = sum((i - inverse_mean) ** 2 for i in inverse_samples) / len(inverse_

print("The mean of the Inverse Sampling trials: ", str(inverse_mean))
print("The variance of the Inverse Sampling trials: ", str(inverse_variance))
plt.savefig("Project3_Q2_2.png")
```




```
The mean of the poisson pmf:  120.0
The variance of the poisson pmf:  120.0
The mean of the Inverse Sampling trials:  119.9942
The variance of the Inverse Sampling trials:  118.46716636000026


<Figure size 432x288 with 0 Axes>
```

**Q2 Analysis**

- The first histogram is that of the number of cars arriving per hour using 10000 Bernoulli trials.
  - Intervals of 0.05 seconds were used, and to be able to fit both the poisson pmf and the Bernoulli Trials on the same histograms, the density had to be used in order for there to be proper scaling.

- The second histogram is that of the number of cars arriving per hour, but using 10000 Inverse Sampling trials.
  - To be able to fit both the poisson pmf and the Inverse Scaling on the same histograms, the density had to be used in order for there to be proper scaling.
- The means and variances of each type of experiment used are listed above, and both methods generate means and variances that are incredibly close to the poisson mean and variance, which is equal to lambda (in this case lambda = 120).

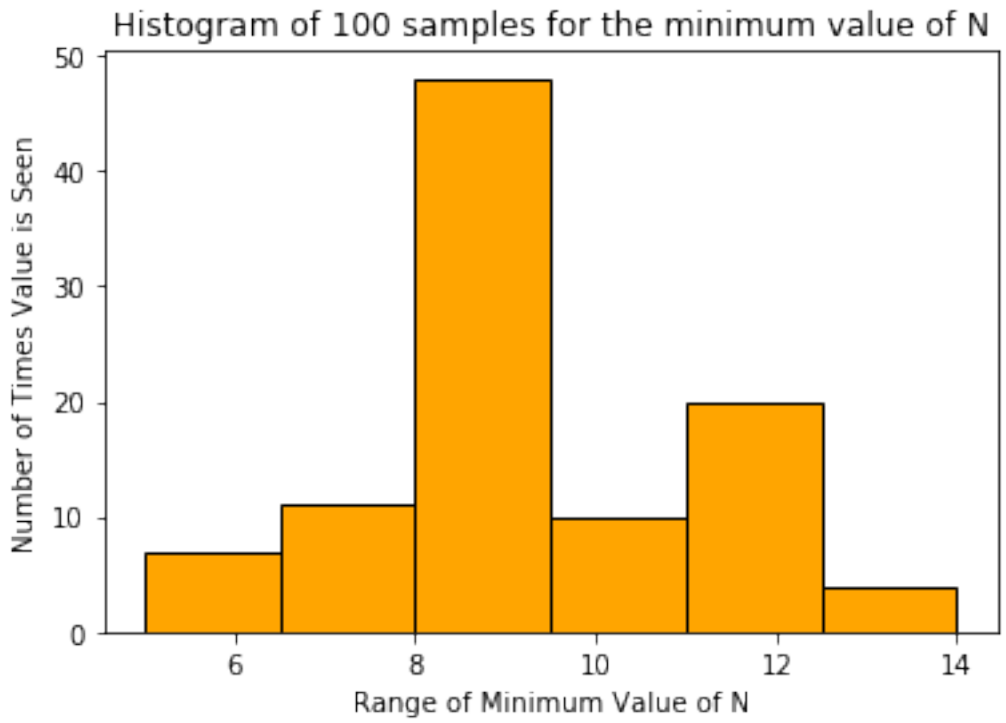
In [ ]:

### 1.0.3 Question 3

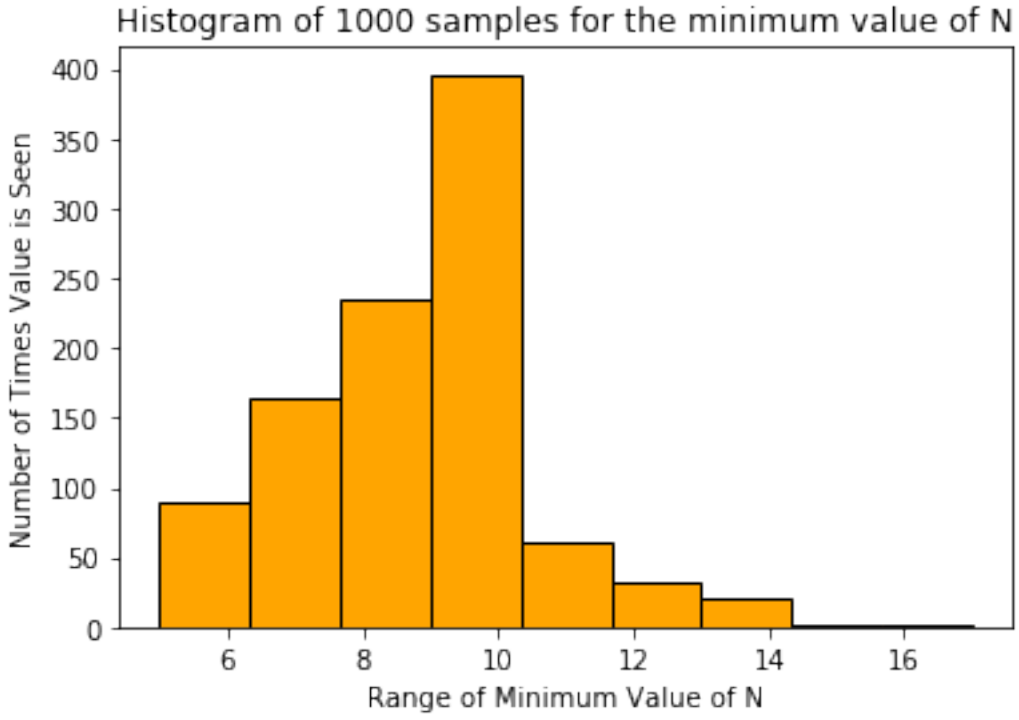
Define the random variable N = min{ n: $\sum_{i=1}^{n}$ Xi > 4} as the smallest number of uniform random samples whose sum is greater than four. Generate a histogram using 100, 1000, and 10000 samples for N. Comment on E[N].

In [61]:
```
import numpy as np
import matplotlib.pyplot as plt
```

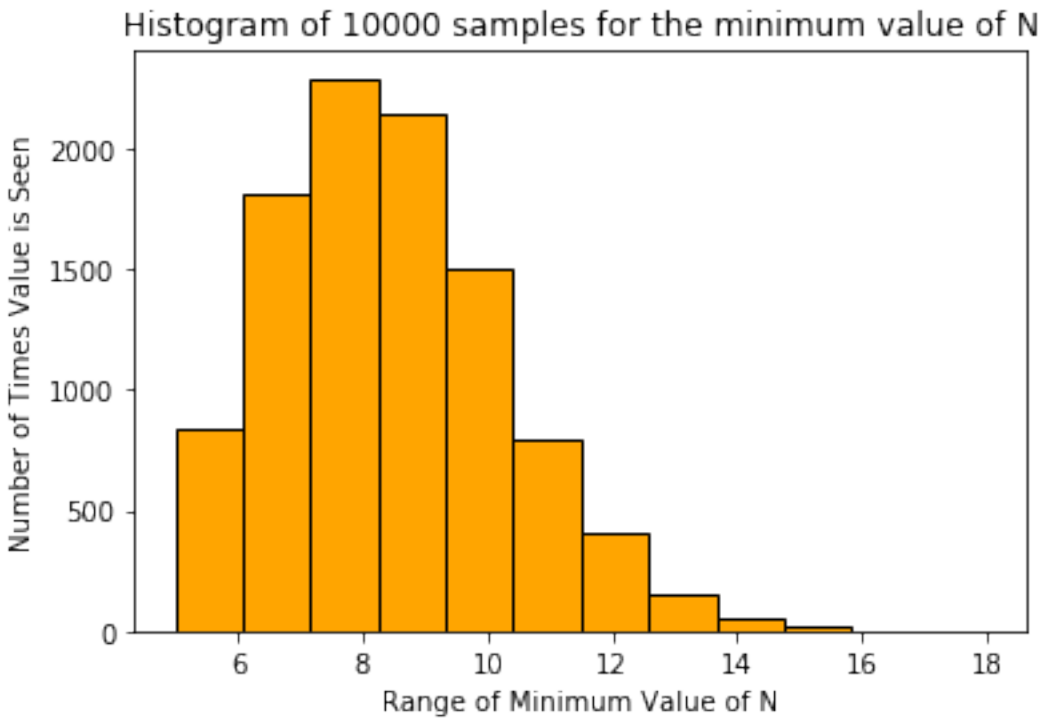
In [65]:
```
sample_lengths = [100 ,1000, 10000]
N_samples = [] #list to store the 100 1000 10000 samples

bins = [6, 9, 12]
for i in range(0,len(sample_lengths)):

    for j in range(0, sample_lengths[i]):
        s = 0
        N = 0
        while (s <= 4):
            u = np.random.uniform(0,1) # the uniform sampling  from [0,1]
            s += u  #summing uniform samples
            N += 1


        N_samples.append(N)

    plt.hist(N_samples, bins[i], edgecolor = 'black', facecolor = 'orange')
    plt.xlabel("Range of Minimum Value of N")
    #plt.xticks(np.arange(5,16,1))
    plt.ylabel("Number of Times Value is Seen")
    plt.title("Histogram of {samples} samples for the minimum value of N".format(sample
    plt.show()
    #print(N_samples)
    print("The sample mean of 100 samples is ", str(np.mean(N_samples)))
    N_samples[:] = []
    plt.savefig("value{samples}.png".format(samples=sample_lengths[i]))
```




```
The sample mean of 100 samples is  9.03
```



```
The sample mean of 100 samples is  8.633
```




```
The sample mean of 100 samples is  8.6819
```

```
<Figure size 432x288 with 0 Axes>
```

**Q3 Analysis**

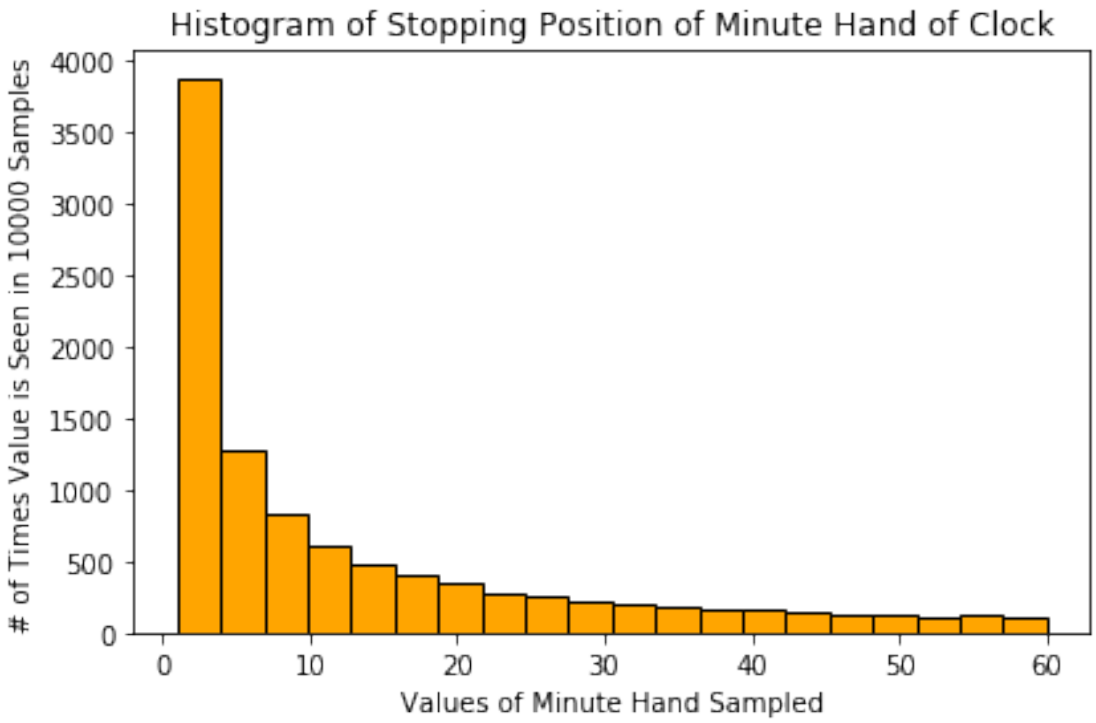
- While the three sample values all generate similar means, the distribution gets less dense near the expected value and looks more and more like a poisson distribution as the number of samples increases.
- This is evident in the histograms. In the 100 sample histogram, the distribution is varied and changes drastically each time the program is run.
- While in the 10000 sample histogram, the values get distributed more thoroughly and starts to look more and more like a poisson distribution.

In [ ]:

**Question 4** Produce a sequence {Xk} where pj = $\frac{p}{j}$ for j = 1,2, . . . , 60 where [U+FFFD] [U+FFFD] 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 = $\frac{p}{j}$ ]. Generate a histgram. Define the random variable Nj = min {k: Xk = j}. Simulate sampling from N60. Estimate E[N60] and Var[N60]. Compare you estimates with the theoretical values.

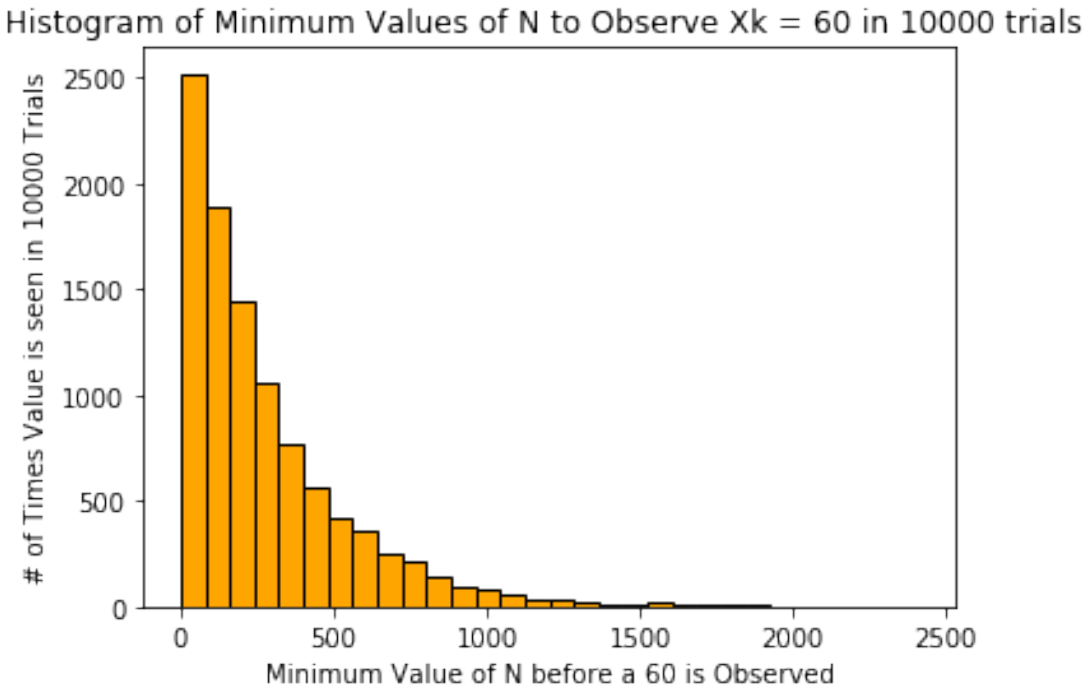
In [54]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [57]:
```
# first step is to figure out what the value of p should be.
# summation of pj from 1 to 60 should equal to 1
probs = []
sum_i = 0
for i in range(1,61):
    sum_i += (1/i)

p = 1/(sum_i)

for j in range(1,61):
    pj = p/j
    probs.append(pj)


X_k_samples = np.random.choice(np.arange(1,61), p = probs, size = 10000) # sample size
#print(X_k_samples == 60)
plt.hist(X_k_samples, bins = 20, edgecolor = 'black', facecolor = 'orange') #histogram
plt.xlabel("Values of Minute Hand Sampled")
plt.ylabel("# of Times Value is Seen in 10000 Samples")
plt.title("Histogram of Stopping Position of Minute Hand of Clock")
plt.tight_layout()
plt.savefig("Project3_Q4_1.png")
plt.show()
```




In [56]:
```
# Part two, generating the random variable for the minimum amount of samples needed to

N = 0
outcome = 0

N_samples = [] # list of all the n samples

for i in range(0,10000):
    N = 0
    outcome = 0

    while outcome != 60:
        outcome = np.random.choice(np.arange(1,61), p = probs, size = 1)
        # sampling one value each from Xk
        N += 1 #incrementing N until the value of 60 is seen from the sampling
    N_samples.append(N)

plt.hist(N_samples, bins = 30, edgecolor = 'black', facecolor = 'orange')
#histogram for 10000 trials of this sampling
plt.xlabel("Minimum Value of N before a 60 is Observed")
plt.ylabel("# of Times Value is seen in 10000 Trials")
plt.title("Histogram of Min. Values of N to Observe Xk = 60 in 10000 trials")
plt.tight_layout()
plt.savefig("Project3_Q4_2.png")
```

```
plt.show()

prob60 = probs[59]
theory_e = 1/prob60
theory_var = theory_e * theory_e - theory_e * theory_e * prob60
print("The theoretical expected value is ", str(theory_e))
print("The theoretical variance is ", str(theory_var))

sample_e = np.mean(N_samples)
sample_var = np.var(N_samples)
print("The sample expected value is ", str(sample_e))
print("The sample variance is ", str(sample_var))
```




```
The theoretical expected value is  280.79222477710414
The theoretical variance is  78563.48127049867
The sample expected value is  276.5055
The sample variance is  75503.67916975002
```

**Q4 Analysis**

- For the first histogram, the majority of the values of the stopping position are between 1 and 10, and it drops of exponentially as the value of j increases.

- For the second histogram, the expected value and variance from the experiment are close to that of the theoretical expected value and variance. The sample expected value is 284.82, while the theoretical one is 280.8. This means that we had to sample that many times from the distribution before a value of 60 was observed on the clock. The fact that this histogram was generated from 10000 experiments points to the precision of the sample expected value in relation the theoretical expected value.
- Interestingly, the sample and theoretical variances are also close to each other, but are significantly larger than the mean. This can also be seen in the histogram, where the values of N range 1 to greater than 1500, which is visual proof of a very large variance.

In [ ]:

**Question 5** Use the accept=reject method to sample from the following distribution pj by sampling from the (non-optimal) uniform auxiliary distribution (qj = .05 for j = 1, . . . ,20): p1 = p2 = p3 = p4 = p5 = .06, p6 = p9 = 0.15, p7 = p10 = 0.13, p8 = 0.14. Estimate the efficiency of your sampler with the following ratio:

$$Efficiency = \frac{accepted}{accepted + rejected}$$

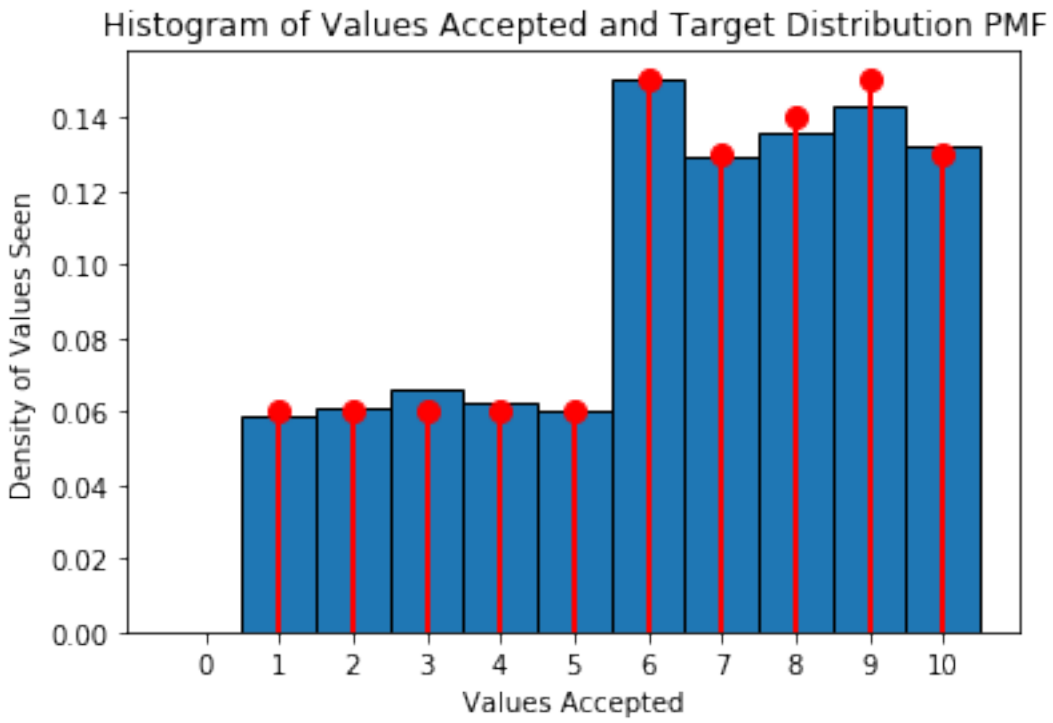
Compare the estimate of the efficiency to the theoretical efficiency given your choice for the constant c.

In [54]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
```

In [95]:
```
p = (1/100)*np.array([6, 6, 6, 6, 6, 15, 13, 14, 15 ,13])

X = []

k = 0
for i in range(0,10000): # 10000 trials

    k += 1  #total number of samples, accepted + rejected
    #print(k)
    j = int(1 + np.floor(10*np.random.rand()))   # Get Uniform j
    if (1.5*np.random.rand()) < p[j-1]/0.1: # Accept p(j) if U<p(j)/c, q(j)= 0.1
        X.append(j)



#print(k)
#print(len(C))
xk = np.arange(1,11)
pk = (.06, .06, .06, .06, .06, .15, .13, .14, .15 ,.13)
custm = stats.rv_discrete(name='custm', values=(xk, pk))
num_bins = np.arange(12)-.5
fig, ax = plt.subplots()
```

```
        # the histogram of the data
        n, bins, patches = ax.hist(X, num_bins, density=1, edgecolor = 'black')

        # add a 'best fit' line
        ax.plot(xk, custm.pmf(xk), 'ro', ms=8, mec='r')
        ax.vlines(xk, 0, custm.pmf(xk), colors='r', linestyles='-', lw=2)
        plt.xticks(range(11))
        plt.xlabel("Values Accepted")
        plt.ylabel("Density of Values Seen")
        plt.title('Histogram of Values Accepted and Target Distribution PMF')
        estimated_eff = len(X)/k
        theoretical_eff = 1/1.5
        sample_var = np.var(X)
        theo_var = 0
        for i in range(1,11):
            theo_var += i*i*p[i-1]
        theo_var = theo_var - np.mean(X)* np.mean(X)

        plt.savefig("Project3_Q5.png")
        print('Sampled expected value of X:',str(np.mean(X)))
        print("Theoretical expected value of X: ", str(np.dot(p,np.arange(1,11))))
        print('Sampled variance of X:',str(sample_var))
        print('Theoretical variance of X:',str(theo_var))
        print('Measured efficiency = ', str(estimated_eff))
        print('Theoretical efficiency = ', str(theoretical_eff ))
```

```
Sampled expected value of X: 6.438764957896292
Theoretical expected value of X:  6.4799999999999995
Sampled variance of X: 7.238715921856701
Theoretical variance of X: 7.7223058169667596
Measured efficiency =  0.6769
Theoretical efficiency =  0.6666666666666666
```




**Q5 Analysis**

- The histogram above shows the distribution of accepted values, along with the PMF in red. This method produces densities that are very similar to that of the theoretical distribution.
- This can also be seen in the measured efficiency, which is just a little higher than the theoretical efficiency.
- The sampled expected value and variance are also within hundreths of the theoretical values, another testament to the efficiency of this method at 10000 trails.

In [ ]: