Name: Deepika Kanade
Assignment 2- EE 511

## Q 1)

**Task:** Writing a routine to plot network samples and calculate the degree of vertex and plot histograms of the same. The main purpose of this exercise to make us acquainted with the structure of graphs for different values of samples and probabilities.

## Code in Python :-

```
import numpy as np
from scipy.special import comb
import networkx as nx
import matplotlib.pyplot as plt
import itertools
import pandas as pd


#function to calculate probabilities from the uniform random variables
def calculateprobabilities(N,p):
    a=np.zeros(N)
    g=[]
    for j in range (0,N):
        a[j]=np.array(np.random.uniform(0,1))
        if a[j]<p:
            g.append(1)
        else:
            g.append(0)
    return(g)


def problem2(n,p):
    N=int(comb(n,2))
    g1=calculateprobabilities(N,p)
    edges=list(itertools.combinations(range(n), 2))  #To generate edges for each possible node

    edge_frames=pd.DataFrame(edges,columns=['Node1','Node2'])
    edge_frames['Edges']=edge_frames[['Node1','Node2']].apply(tuple, axis=1)
    edge_frames['Selection1']=g1
    edge_final1=list(edge_frames.loc[edge_frames['Selection1']==1,'Edges'])  #To find the
location where value of edges is 1

    degree1=edge_frames.loc[edge_frames['Selection1']==1,'Node1']  #gives location of nodes
where edges have value 1
```

```
    degree_count1=pd.Series.value_counts(degree1)  #counts degree of vertex for each node

    G1 = nx.Graph()
    G1.add_edges_from(edge_final1)
    G1.add_nodes_from(range(n))
    plt.figure()
    plt.suptitle('Graph for n=100 and p=0.06')
    nx.draw(G1)
    plt.figure()
    plt.suptitle('Histogram of degree of count for n=100 and p=0.06')
    plt.xlabel('Number of degrees')
    plt.ylabel('Count of each degree')
    plt.hist(degree_count1)

n=50
N=int(comb(n,2))
g1=calculateprobabilities(N,0.02)   #calculation with different probabilities
g2=calculateprobabilities(N,0.09)
g3=calculateprobabilities(N,0.12)
edges=list(itertools.combinations(range(n), 2))

edge_frames=pd.DataFrame(edges,columns=['Node1','Node2'])
edge_frames['Edges']=edge_frames[['Node1','Node2']].apply(tuple, axis=1)
edge_frames['Selection1']=g1
edge_frames['Selection2']=g2
edge_frames['Selection3']=g3
edge_final1=list(edge_frames.loc[edge_frames['Selection1']==1,'Edges']) #To find location
where value of edge us 1 for 1st set of probabilities
edge_final2=list(edge_frames.loc[edge_frames['Selection2']==1,'Edges']) #To find location
where value of edge us 1 for 2nd set of probabilities
edge_final3=list(edge_frames.loc[edge_frames['Selection3']==1,'Edges']) #To find location
where value of edge us 1 for 3rd set of probabilities

degree1=edge_frames.loc[edge_frames['Selection1']==1,'Node1']
degree2=edge_frames.loc[edge_frames['Selection2']==1,'Node1']
degree3=edge_frames.loc[edge_frames['Selection3']==1,'Node1']

degree_count1=pd.Series.value_counts(degree1)
degree_count2=pd.Series.value_counts(degree2)
degree_count3=pd.Series.value_counts(degree3)

#plot for graphs for all probabilities
G1 = nx.Graph()
G1.add_edges_from(edge_final1)
G1.add_nodes_from(range(n))
plt.figure()
```

```
plt.suptitle('Graph for n=50 and p=0.02')
nx.draw(G1)

G2 = nx.Graph()
G2.add_edges_from(edge_final2)
G2.add_nodes_from(range(n))
plt.figure()
plt.suptitle('Graph for n=50 and p=0.09')
nx.draw(G2)

G3 = nx.Graph()
G3.add_edges_from(edge_final3)
G3.add_nodes_from(range(n))
plt.figure()
plt.suptitle('Graph for n=50 and p=0.12')
nx.draw(G3)

#Histograms for degree of vertex
plt.figure()
plt.hist(degree_count1)
plt.suptitle('Histogram of degree of count for n=50 and p=0.02')
plt.xlabel('Number of degrees')
plt.ylabel('Count of each degree')

plt.figure()
plt.suptitle('Histogram of degree of count for n=50 and p=0.09')
plt.xlabel('Number of degrees')
plt.ylabel('Count of each degree')
plt.hist(degree_count2)

plt.figure()
plt.suptitle('Histogram of degree of count for n=50 and p=0.12')
plt.xlabel('Number of degrees')
plt.ylabel('Count of each degree')
plt.hist(degree_count3)

#To calculate and plot hitogram for N=100 and p=0.06
problem2(100,0.06)
```

**Output:-**

Figure 1-Graph for Number of samples=50 and probability=0.02

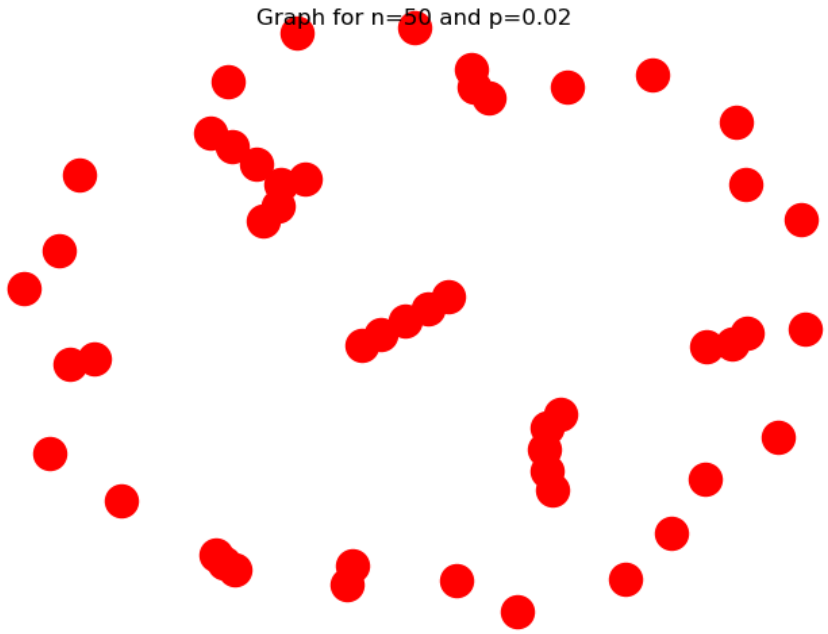



Figure 2-Graph for Number of samples=50 and probability=0.09

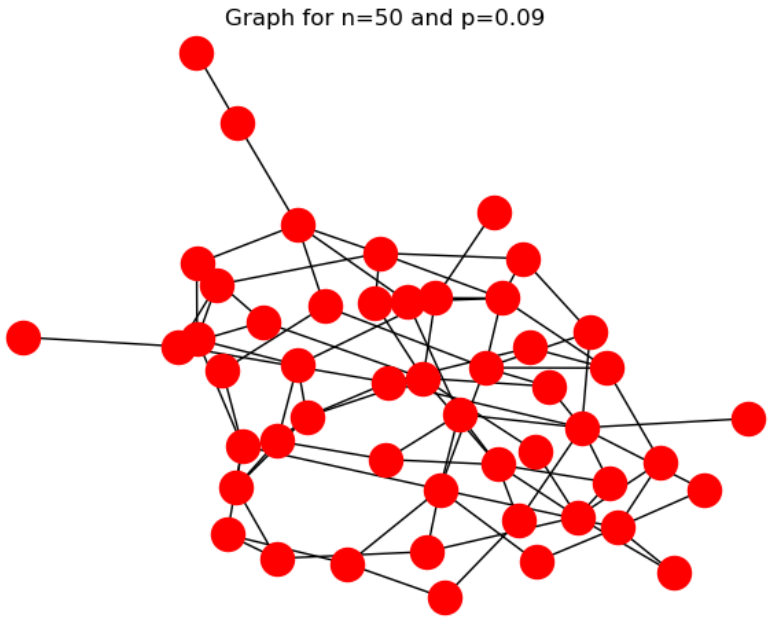


Figure 3-Graph for Number of samples=50 and probability=0.12

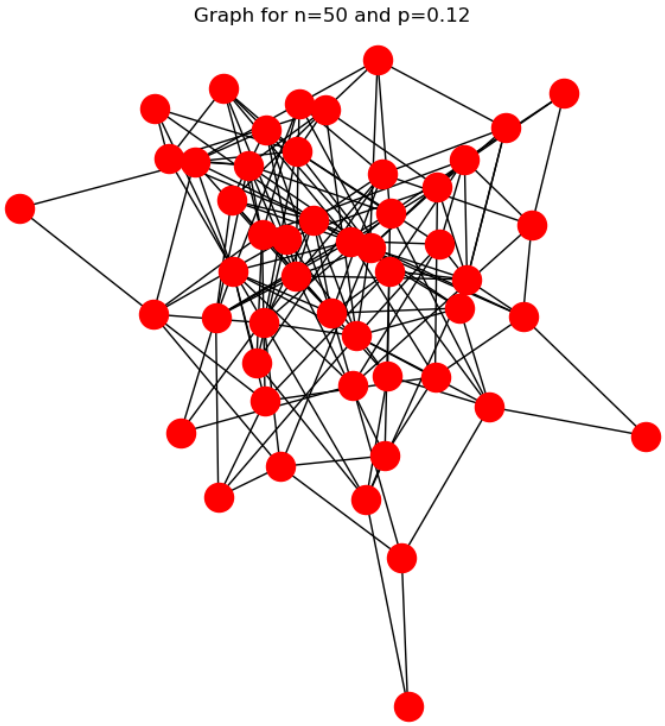



Figure 4- Histogram of degree of vertex for number of samples=50 and probability=0.02

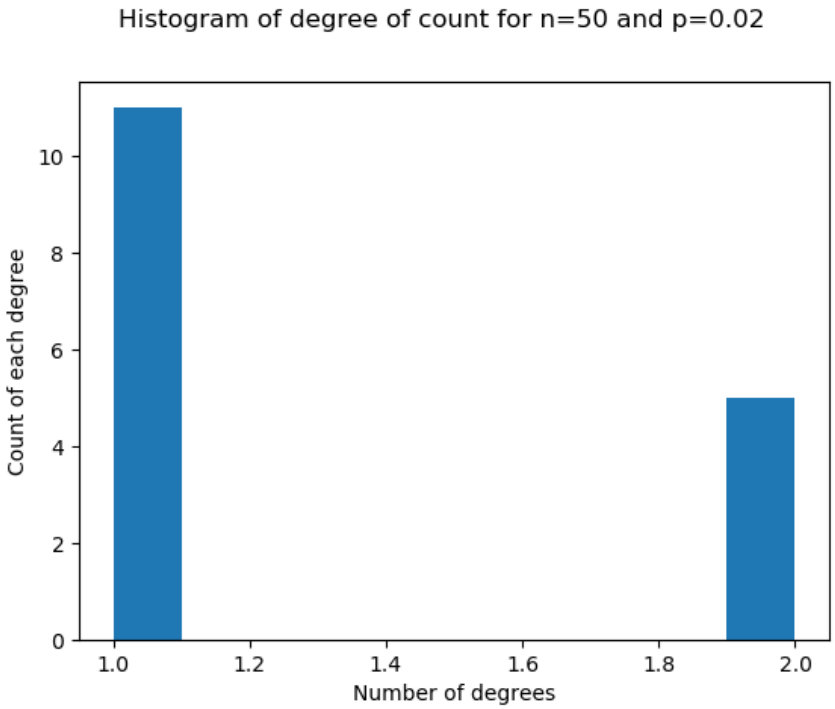


Figure 5-Histogram of degree of vertex for number of samples=50 and probability=0.09

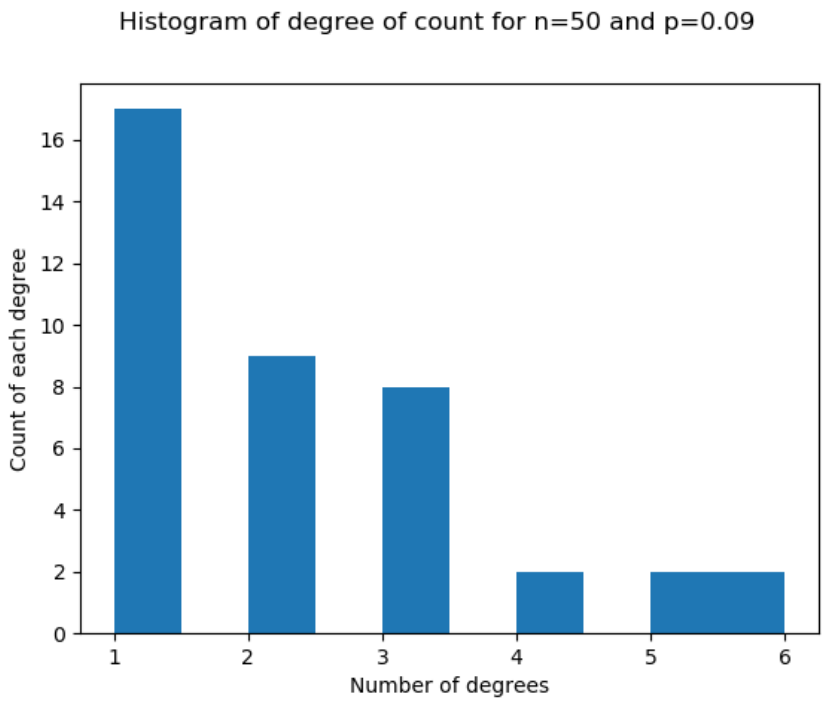
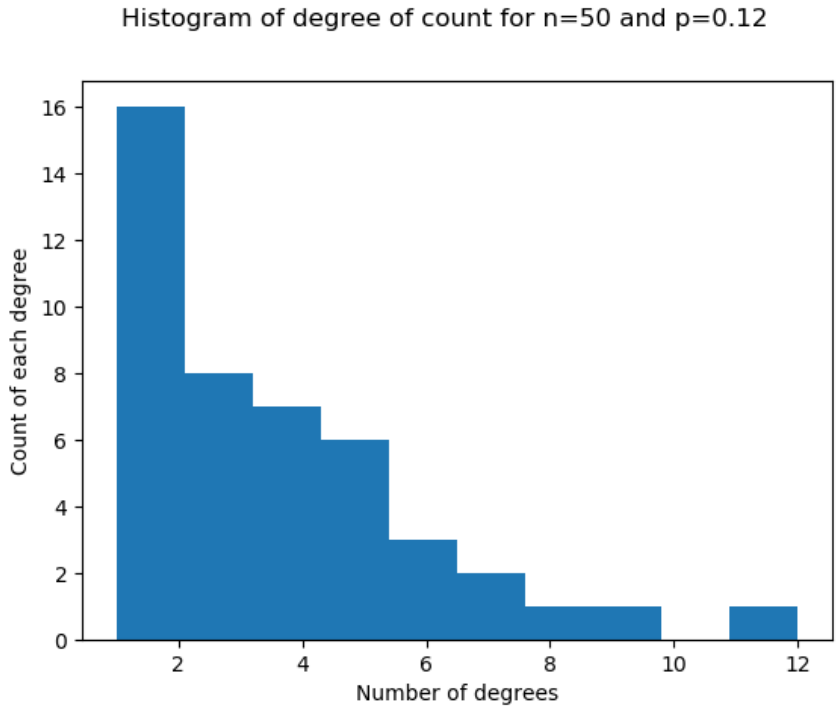



Figure 6-Histogram of degree of vertex for number of samples=50 and probability=0.12



As is visible from the graphs, as the probability increases, the graph becomes more and more dense. This happens because a lot of uniform samples are accepted and hence the number of edges is in turn increased as probability increases.

As seen from figure 1, the probability for acceptance of sample is 0.02 which is too less. Hence, a lot of samples are rejected, and the graph appears sparse as the number of connections between the nodes is less. As is visible from the figure, the nodes form clusters where the connection between the nodes is possible. The graph in this case is beautifully placed.

However, in figures 2 and 3, the graph appears denser as the number of samples getting accepted increases. Hence, a lot of nodes are connected to each other. Figure 3 has a lot of interconnected nodes.

The degree of vertex is the number of connections of each node in the graph. A Histogram of the degree of vertex was plotted which gives an idea about the count of connectivity in the nodes. For instance, in figure 4 which displays the degree for n=50 and p=0.02, a lot of nodes are connected to only a single node whereas in figure 6 which displays n=50 and p=0.12, a large number of nodes are connected to a lot of other nodes.

Figure 7-Graph for Number of samples=100 and probability=0.06

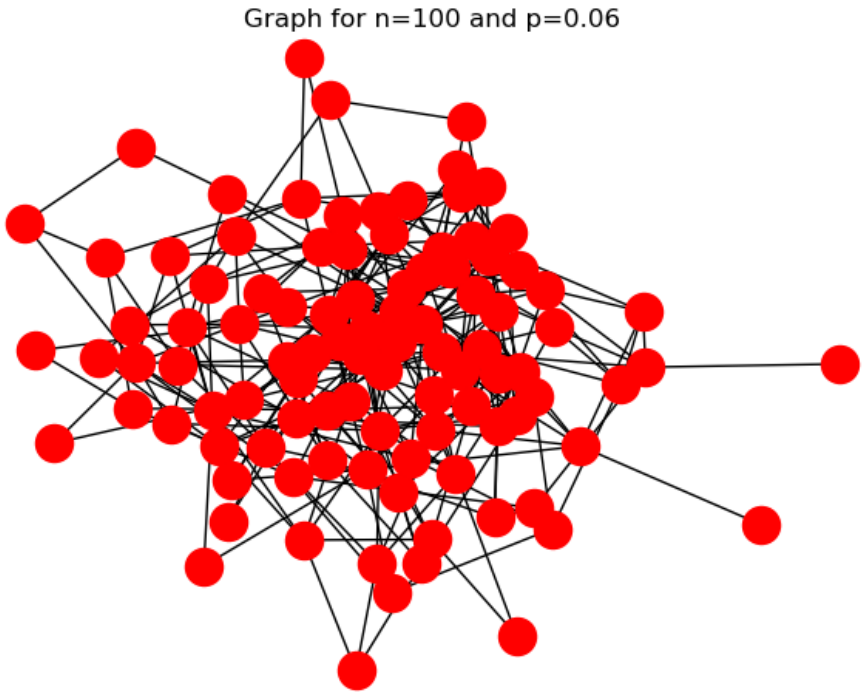
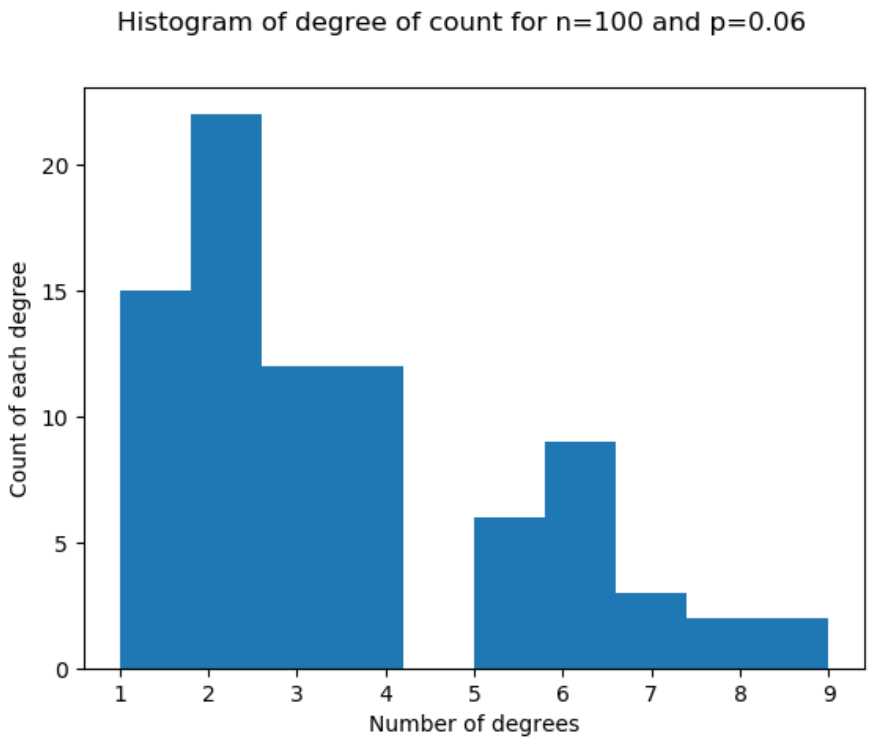


Figure 8-Histogram of degree of vertex for number of samples=100 and probability=0.06




As seen in the figure 7, the number of nodes in the graph has increased which in turn makes the graph denser. The acceptance probability is also moderate enough to generate considerable edges.

After observing the histogram in figure 8, it can be inferred the it has a Poisson distribution. This coincides with our assumption that when the number of samples is increased, the distribution tends to converge to Poisson. Hence, as number if samples is increased, the binomial distribution assumption is no more valid, the distribution becomes Poisson.

## Q 2)

**Task:** Writing a routine to generate exponential random variables from uniform distribution using inverse cdf method and to evaluate their quality using goodness of fit tests. This exercise is also aimed at helping us understand the significance of an exponential random variable in terms of the waiting time.

## Code in Python: -

```
##Generation of exponential random variable from uniform trials using inverse cdf method
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats

N=1000
u=np.zeros(N)
x=np.zeros(N)
theta=0.2   #Value of average waiting time

#Generation of exponential random variable from uniform trials using inverse cdf method
uniform_trials=np.array(np.random.uniform(size=1000))
expo_observed_values=-theta * np.log(1-uniform_trials)

#Generation of exponential randon variable using in-built function
expo_expected_values=np.array(np.random.exponential(theta,size=1000))

#Taking the x axis values of expected and observed random variables
expected=plt.hist(expo_expected_values)[0]
observed=plt.hist(expo_observed_values)[0]

#Running the chi-square goodness of fit test
[chi_square,p]=scipy.stats.chisquare(observed, expected)
print('Value of chi-square')
print (chi_square)
print('Value of p')
print (p)
#Running the ks goodness of fit test
[KS,p]=scipy.stats.kstest(expo_observed_values, 'expon')
print('Value of KS')
print (KS)
print('Value of p')
print (p)

#plt.figure()
#plt.hist(expo_expected_values)

#To calculate the number of exponential time intervals in 1 unit time
sum=0
count=0
```

```
count_array=[]
for i in range(0,1000):
    sum=sum+expo_observed_values[i]
    count=count+1
    if sum>1:
        count_array.append(count)
        count=0
        sum = 0

#Plotting the histogram of the number of exponential time intervals in 1 unit time
plt.figure()
plt.suptitle('Number of exponentially distributed time intervals in 1 unit time')
plt.xlabel('Number of exponential time intervals in 1 unit time')
plt.ylabel('Count of exponential time intervals in 1 unit time')
plt.hist(count_array)
```

## Output :-

Value of chi-square and KS test for bins=30

```
Value of chi-square
31.5613083448
Value of p
0.00462239700141
Value of KS
0.541218039962
Value of p
0.0
```

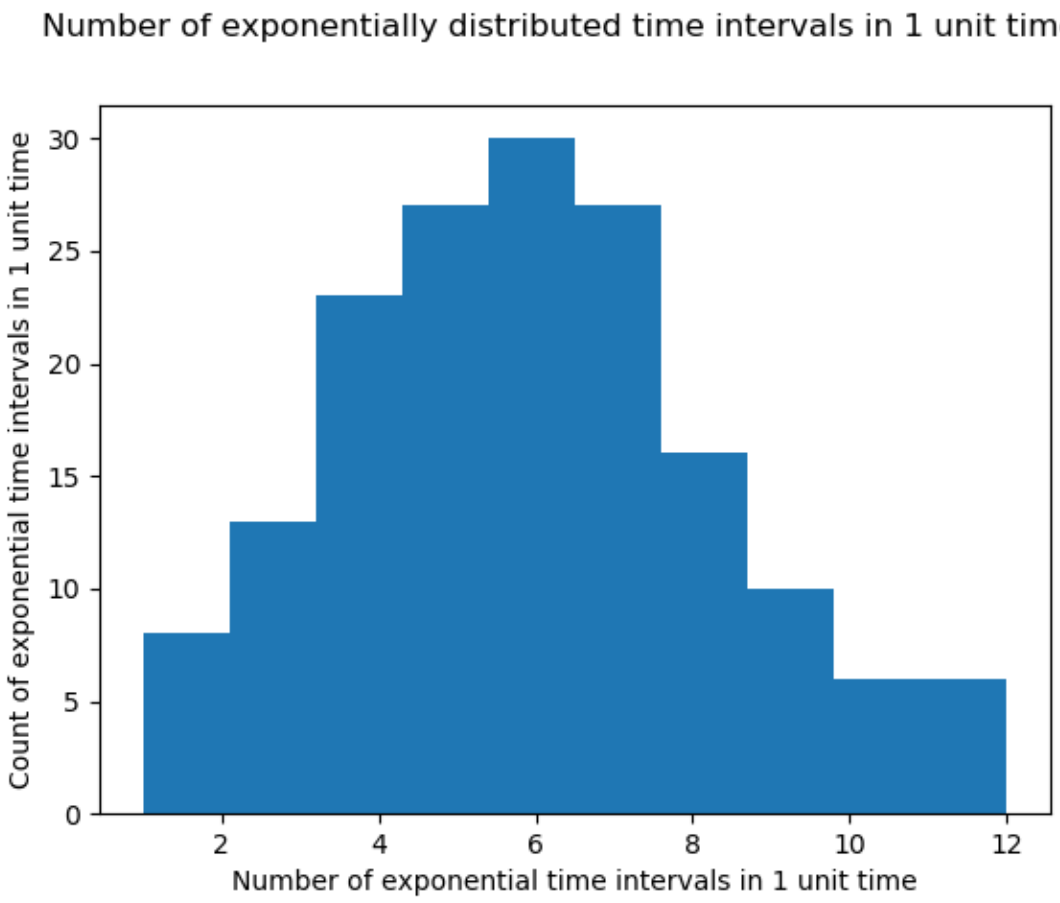
Value of chi-square and KS test for bins=40

```
Value of chi-square
83.4407765726
Value of p
4.71246864754e-10
Value of KS
0.537468781992
Value of p
0.0
```

Value of chi-square and KS test for bins=50

```
Value of chi-square
26.2442392543
Value of p
0.123526130525
Value of KS
0.542264950328
Value of p
0.0
```

Figure 9- Histogram for number of exponentially distributed time intervals in 1 unit time



The generation of 1000 exponential samples is performed using the inverse cdf method i.e. using uniform samples.
The following formula was used to calculate the chi-square goodness of fit.

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$$

Where Oi=observed value from the exponential distribution using the inverse cdf method
Ei=expected value from the exponential distribution

As seen from the outputs, as the number of bins is increased the value of chi-square becomes better. The value of p is also a good measure of the quality.
This chi-square value can be checked by comparing the chi-square value for nth degree of freedom to chi-square values for the (n-1)th degree of freedom. I checked the values for the bin sizes 30,40,50. The value from tables is as shown below.
For $14^{th}$ degree of freedom and p=0.005, chi-square=31.319
For $19^{th}$ degree of freedom and p=0.001, chi-square= 43.820
For $25^{th}$ degree of freedom and p=0.995, chi-square= 9.886

Hence, in every case, the value of chi-square obtained by running the algorithm is greater than the previous degree of freedom value from the table. Hence, the chi-square goodness of fit gives us the expected results.

Another test that I have used is the KS test. The value of KS should be between 0 and 1, which coincides with the KS values generated by my algorithm.

Observing the figure 9, it can be seen that the histogram has a normal distribution. This happens as the count of the number of time intervals between a unit time interval follows a Gaussian model. Also, as we are generating exponential random variables with waiting time=0.2, ideally it would take 5 steps to reach 1-unit time interval. Hence, the mean of the data will have an expected value of 5 which matches the mean value of the plot in Figure 9.
Hence, the count is distributed normally with mean approximately equal to 5.

## Q 3)

**Task:** Writing a routine to generate the samples in the given bimodal distribution and to calculate the rejection rate.

## Code in Python :-

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

k = 5000
a = 0
b = 6

### Generate Uniform samples
uniform_trials= np.random.uniform(low=a, high=b, size=k)

### Get the maximum value of f(x) for Accept/Reject
cmax = 1.46

### Accept/Reject Routine
accept_reject = []
uniform_trials_double_rejection = np.random.uniform(low = a, high=cmax, size=k)

reject_count = 0
tot_count = 0
number_of_rejections = []
x_list = []
function_of_x_list = []

while (np.sum(accept_reject) < 1000):
    x = np.random.uniform(low=a, high=b, size=1)
    #function_of_x = returnFunctionOfx(x)
    if ((x > 0) & (x <= 1)):
        function_of_x = 0.5 * np.random.beta(8,5)
    elif ((x > 4) & (x <= 5)):
        function_of_x =  0.5 * (x - 4)
    elif ((x > 5) & (x <= 6)):
        function_of_x = -0.5 * (x - 6)
    else:
        function_of_x = 0
    c = np.random.uniform(low = a, high=cmax, size=1)
    x_list.append(x)
    function_of_x_list.append(function_of_x)
    #print ("f(x): " , function_of_x)
    if (c <= function_of_x):
        tot_count += 1
        accept_reject.append(1)
        number_of_rejections.append(reject_count)
```

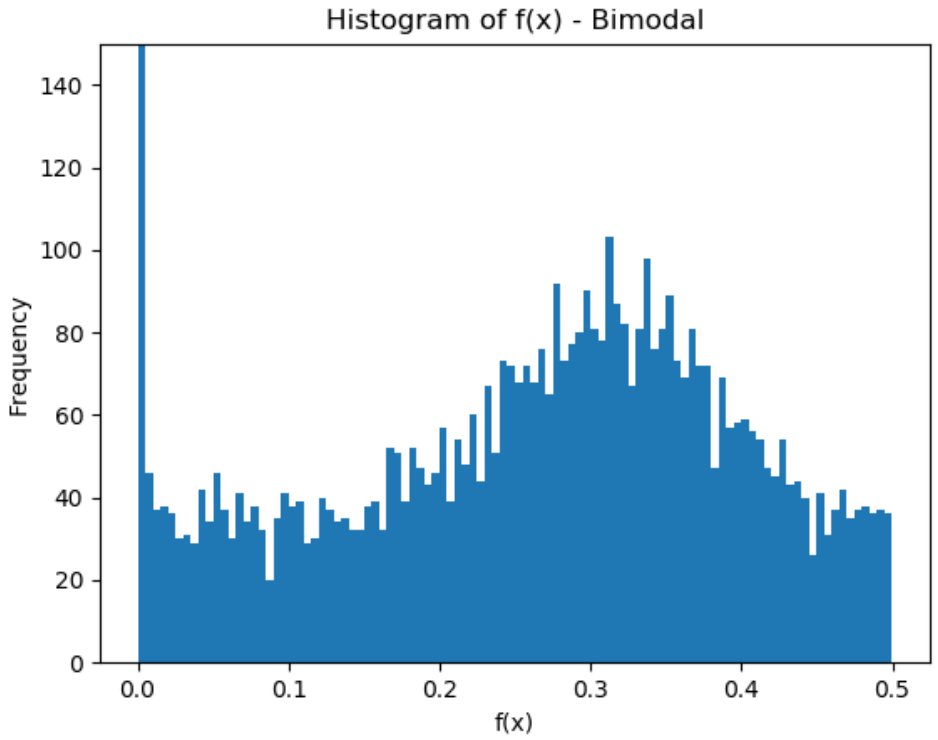
```
        reject_count = 0
        tot_count = 0
        #print (" -- Accept")
    else:
        accept_reject.append(0)
        reject_count += 1
        tot_count += 1
       # print (" -- Reject")

print ("Total Number of Iterations: ", len(accept_reject))
print ("Average number of samples rejected",np.mean(number_of_rejections))
print ("Rejection rate", float(np.sum(number_of_rejections)) / float(len(accept_reject)))


plt.figure()
plt.hist(function_of_x_list, bins=100)
plt.xlabel("f(x)")
plt.ylabel("Frequency")
plt.title("Histogram of f(x) - Bimodal")
plt.ylim([0,150])
```

## **Output :-**

Figure 10- Histogram of the bimodal distribution with respect to f(x)



```
Total Number of Iterations:  10217
Average number of samples rejected 9.217
Rejection rate 0.9021239111285113
```

The maximum value Cmax was calculated by computing the maximum value of the f(x) distribution, which came out to be Cmax=1.46.
The average rejection rate is computed by measuring the average number of rejections per sample.
As it can be seen in the output, the average number of samples rejected is 9.217 and the average rejection rate is 90.21%.

The histogram gives us an idea of the bimodal distribution as there are two distinct modes visible from the figure. One is at the value of f(x)=0 and other mode is approximately near f(x)=0.35.