EE 511 – PROJECT 2 – SPRING 2018 - RANDOM NUMBER GENERATION

ABINAYA MANIMARAN

PROBLEM – 1 NETWORKING

Given n=50 people in a social network. Given that un ordered pairs are connected at random and independently with probability p.

PLOT FOR DIFFERENT VALUES OF PROBABILITY:

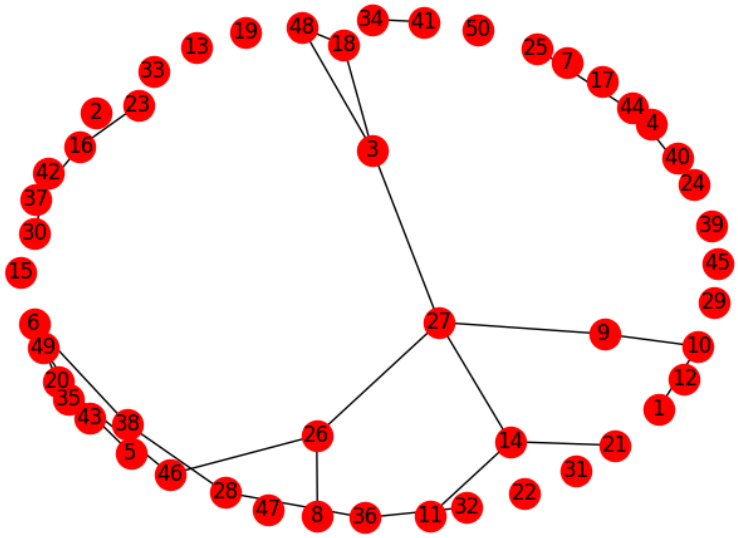


Figure 1

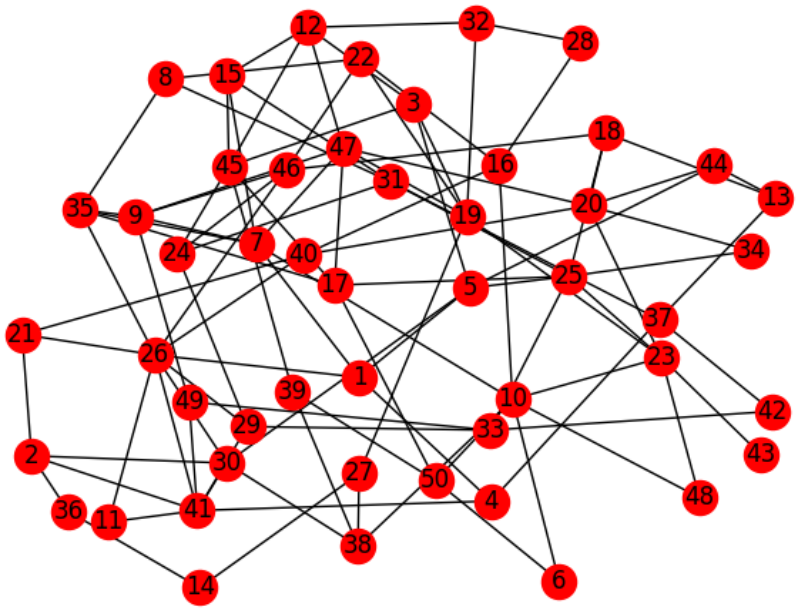


Figure 2

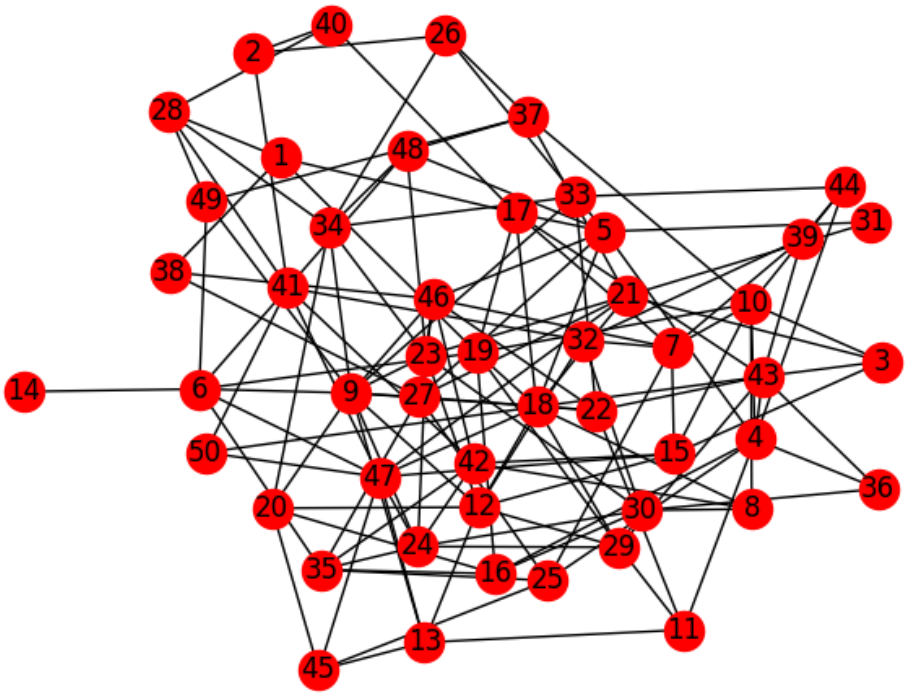


Figure 3

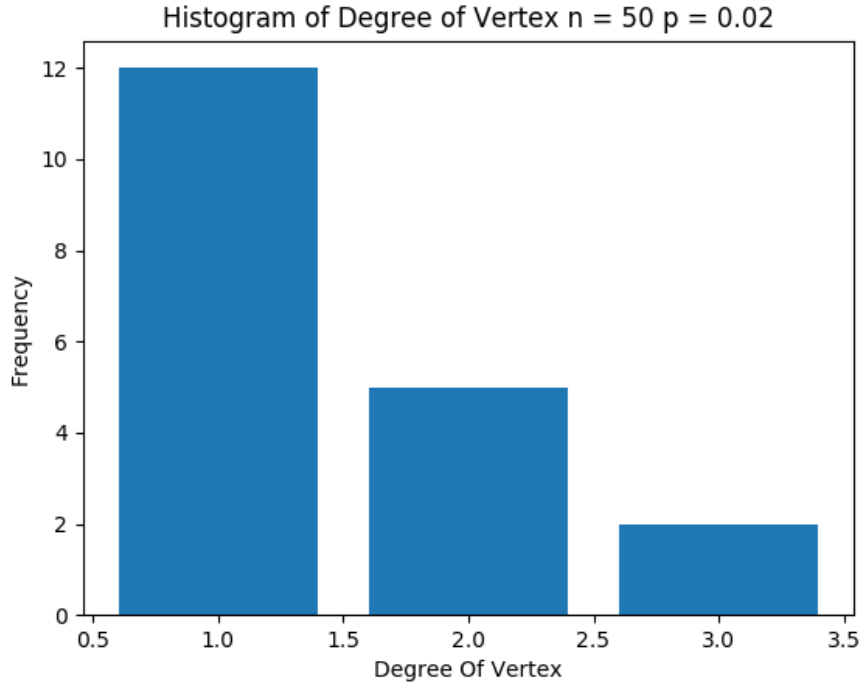
Figure 1, 2 and 3 are the network graphs for p = [0.02, 0.09 and 0.12] respectively

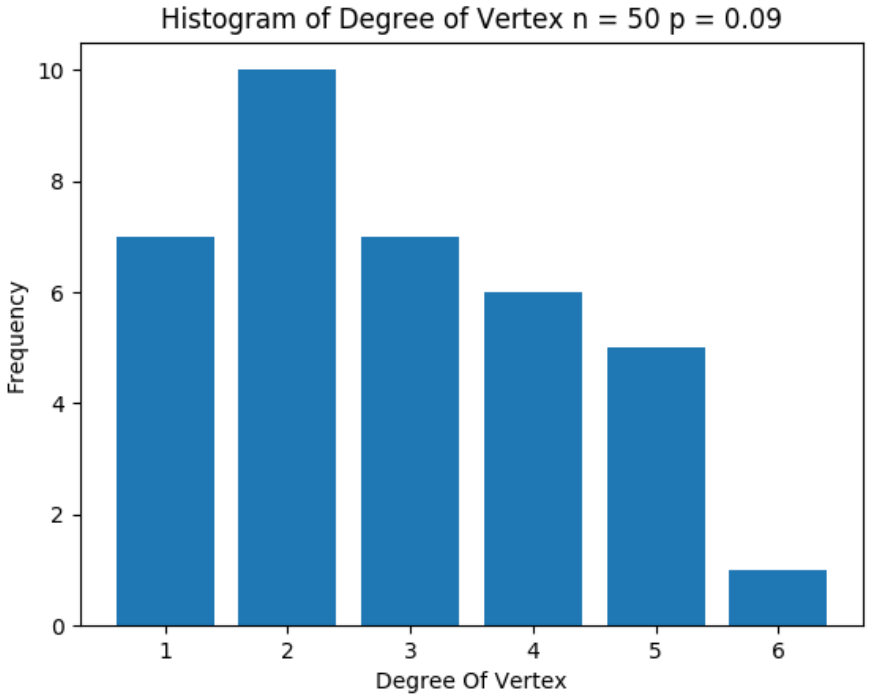
The following observations can be made from the above graph network structures:

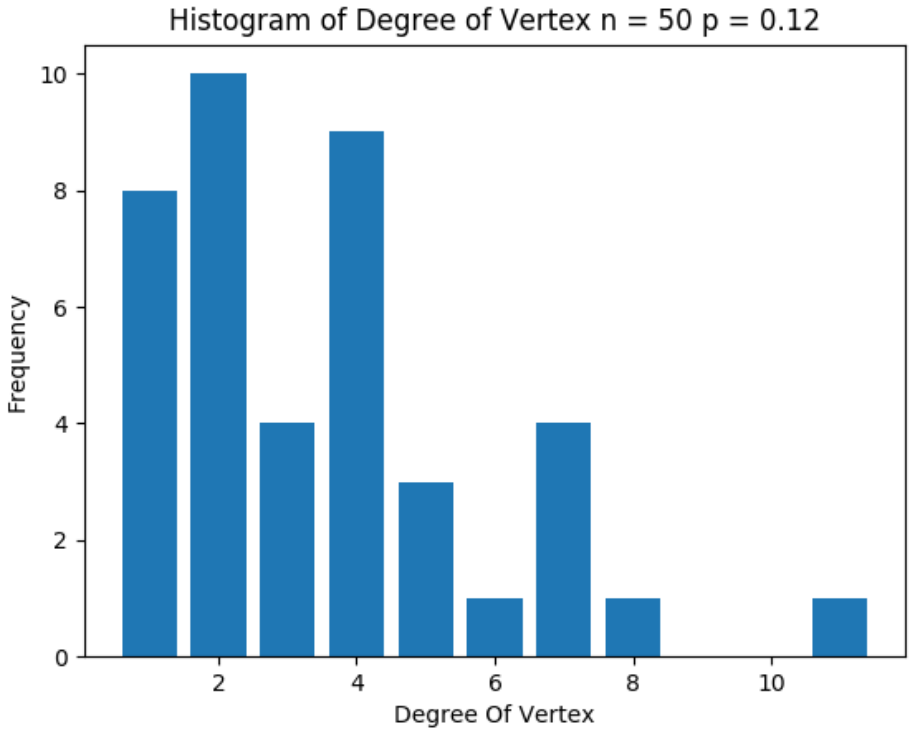
- When the probability is less, the connections between the nodes are less. Very few nodes are connected to each other in the first figure, when probability p=0.02
- Similarly, as the probability value increases, the connections grow stronger. We can observe that from figure 2 and 3. The number of edges is seen increasing very obviously.
- The structure of the graph also depends on the number of nodes connected, indirectly on the probability.
- Figure 1 graph nodes are arranged in such a way that the nodes connected are grouped together. Since less number of nodes are connected, the graph is beautifully structured elliptically.
- Since in figure 2 and 3, the number of nodes connected are high, the graph nodes are oddly placed to suffice the connections.

PLOT OF HISTOGRAMS OF DEGREE OF VERTEX:

Degree of Vertex is the number of connections each node or vertex is connected to. Degree of vertect would again depend of the probability of edge selection. The graphs shown in the next page correspond to the histogram of degree of vertex for the above 3 graphs. Figure 4, figure 5 and figure 6 are for p [0.02, 0.09 and 0.12] respectively.

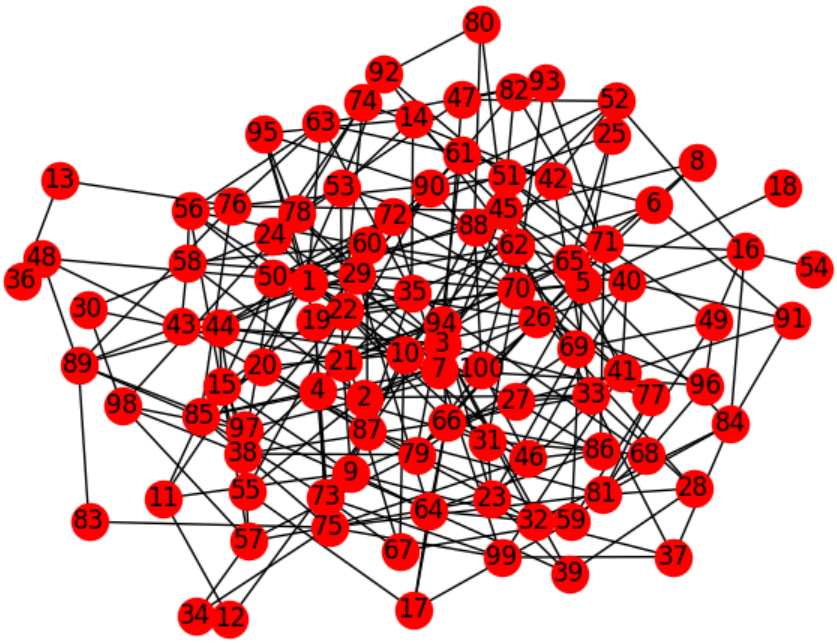

Histogram of Degree of Vertex n = 50 p = 0.02
Frequency
Degree Of Vertex
0
2
4
6
8
10
12
0.5
1.0
1.5
2.0
2.5
3.0
3.5

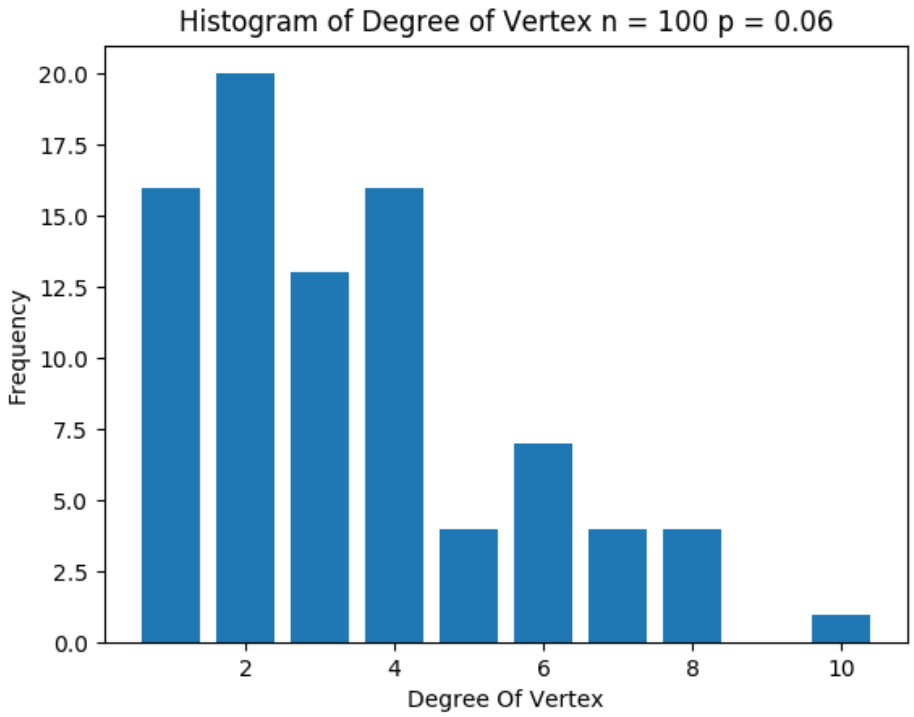

Histogram of Degree of Vertex n = 50 p = 0.09
Frequency
Degree Of Vertex
0
2
4
6
8
10
1
2
3
4
5
6



NETWORK FOR 100 NODES WITH PROBABILITY 0.06:

Degree of vertex is said to be binomially distributed for small network sizes. We would look into what happens when the number of nodes increases. The network graph and histogram of vertex of nodes are given below.





As the number of nodes 'n' increases, the binomial distribution approaches to **Poisson**. We can see that clearly from the above histograms shown.

PROBLEM – 2 WAITING

INVERSE CDF METHOD FOR EXPONENTIAL RANDOM VARIABLES:

The procedure followed for inverse CDF Method to generate the exponentially distributed random variables is given below:

- Given theta – 0.2 (0.2 time units is the average waiting time)
- Generate uniformly distributed random variables
- Exponential random variables are generated by using the given formula:

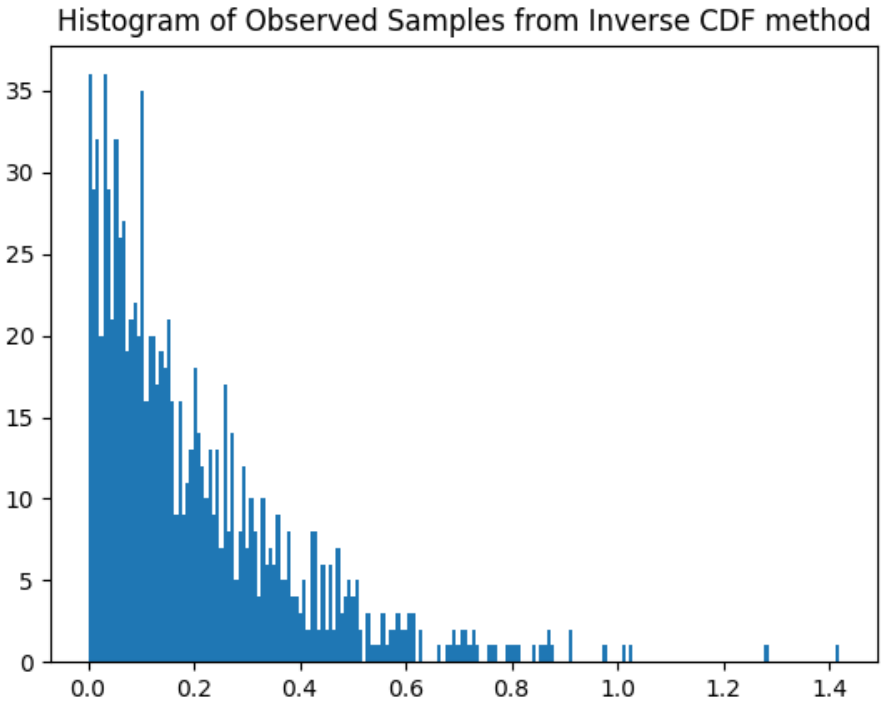
$$y = (- \text{theta}) * \log (1 – x)$$

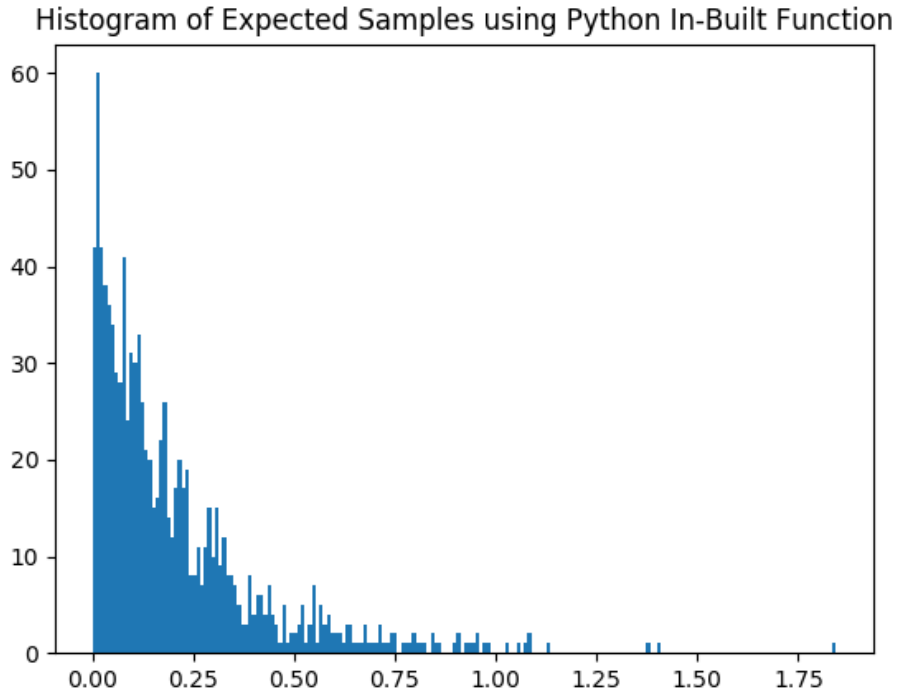
where, x is the uniformly distributed random values

y is the exponentially distributed random values

The quality of this method can be tested using various Goodness of Fit tests.

- Chi- square test
- KS test





The histogram of Observed exponential samples (using Inverse CDF method) and expected samples are given above. The range of the values generated, and frequency of occurrence is very similar when compared. This can be validated by various of Goodness of Fit Test.

- **chi-square Goodness of Fit test:**

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

where

χ^2 = Pearson's cumulative test statistic, which asymptotically approaches a χ^2 distribution.

O_i = the number of observations of type *i*.

N = total number of observations

$E_i = Np_i$ = the expected (theoretical) frequency of type *i*, asserted by the null hypothesis that the fraction of type *i* in the population is p_i

n = the number of cells in the table.

Since this test is sensitive on the number of bins we choose, we can choose bins in such a way that the desired chi-square value is obtained. Compared with critical region.

Bins = 5, chi-square = 41.2634 (> 18.467 for p value = 0.000023)

Bins = 20, chi-square = 76.0842 (> 43.820 for p value = 0.000008)

Bins = 30, chi-square = 101.7821 (> 58.301 for p value = 0.000005)

- **KS-Test:**

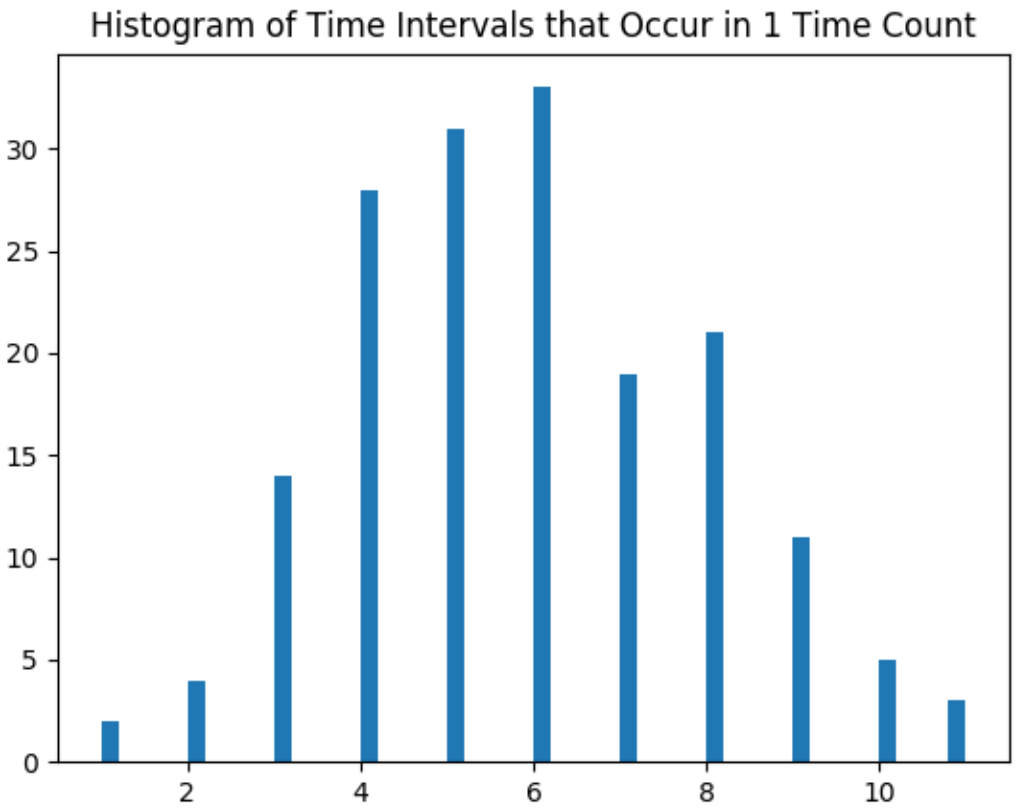
Kolmogorov–Smirnov test (K–S test or KS test) is a nonparametric test of the equality of continuous, one-dimensional probability distributions that can be used to compare a sample with a reference probability distribution (one-sample K–S test), or to compare two samples (two-sample K–S test).

KS test value = 0.544618237379

The above obtained values for Chi-square goodness of fit test and KS test align with good distributions. There are distortions because, the exponential random numbers were generated using Inverse CDF Method. This method is just an approximation for random numbers of a particular distribution that have invertible CDF. These values can be improved with other random number generation methods.

TIME INTERVALS THAT OCCUR IN 1 TIME UNIT:

- Exponential random variables are generated using Inverse CDF Method
- We count the number of steps we have to take to get 1 time unit if we cumulatively add the random variable.
- This was done 1000 times and obtained the counts
- The distribution is given below and looks **Normally distributed**.
- Since the average waiting time is 0.2 time units, it approximately takes 5 steps to get 1 time unit (0.2 *5 = 1)
- Hence the below graph is normally distributed with **mean around 5**



PROBLEM 3 DOUBLE REJECTION

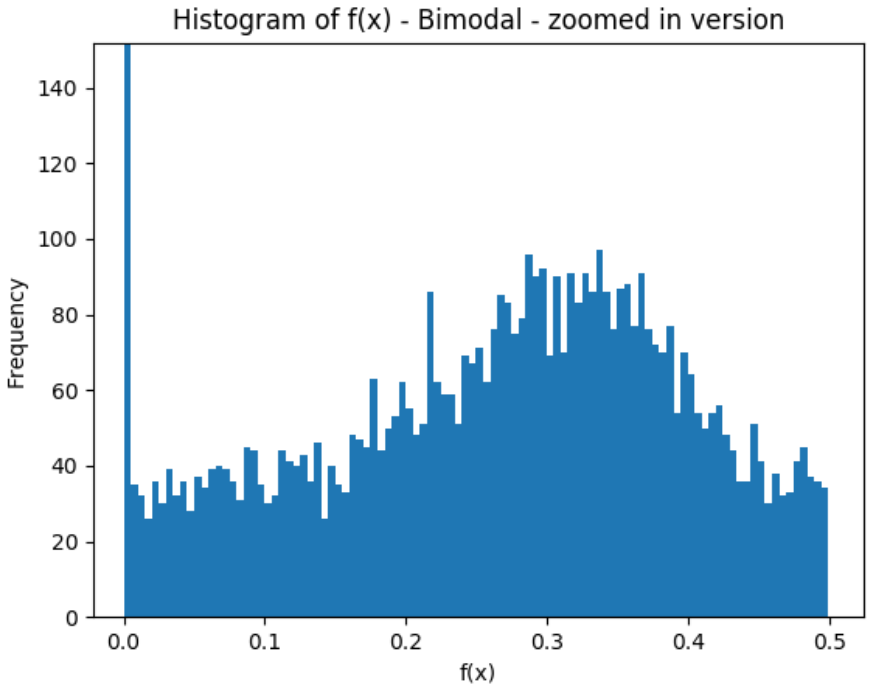
The algorithm to implement rejection sampling routines for X is given below:

- Generate x – uniform random number between 0 and 6: U(0,6)
- Calculate f(x) using the given distribution
- Generate y – uniform random number between 0 and max(f(x)) = approximately 1.46
- Accept the sample if y <= f(x)
- Otherwise reject
- Track the number of samples to rejected to get 1 acceptance
- Repeat this number of times to get 1000 accepted samples
- Calculate the rejection rate = number of rejected samples / total number of iterations
- This is the measure of efficiency

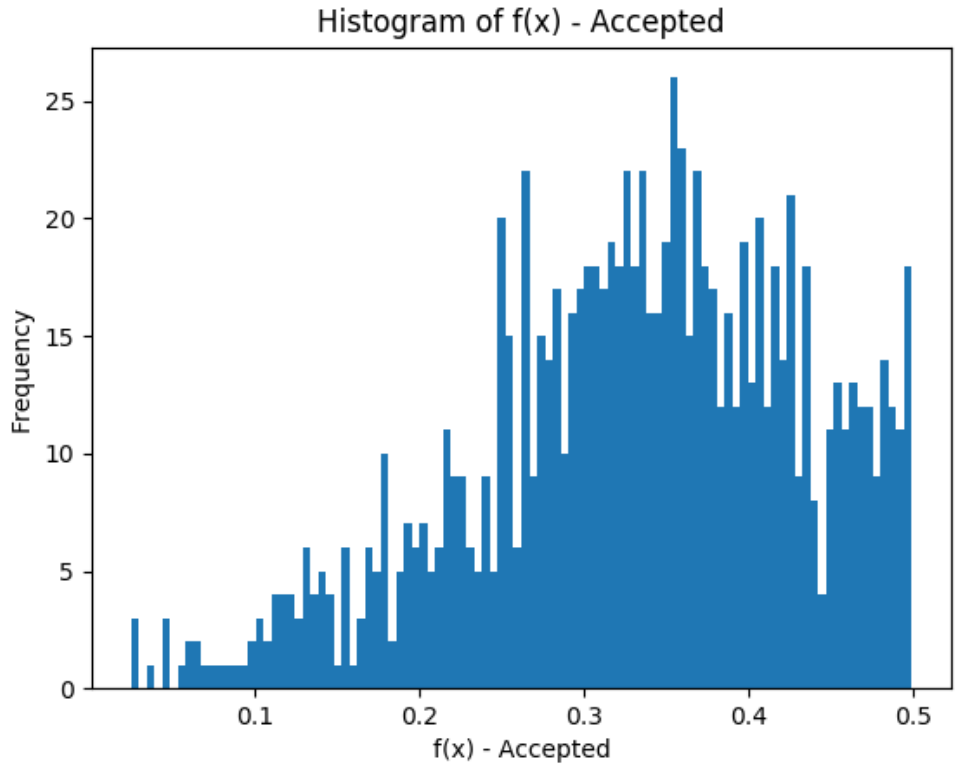
The results are shown below:

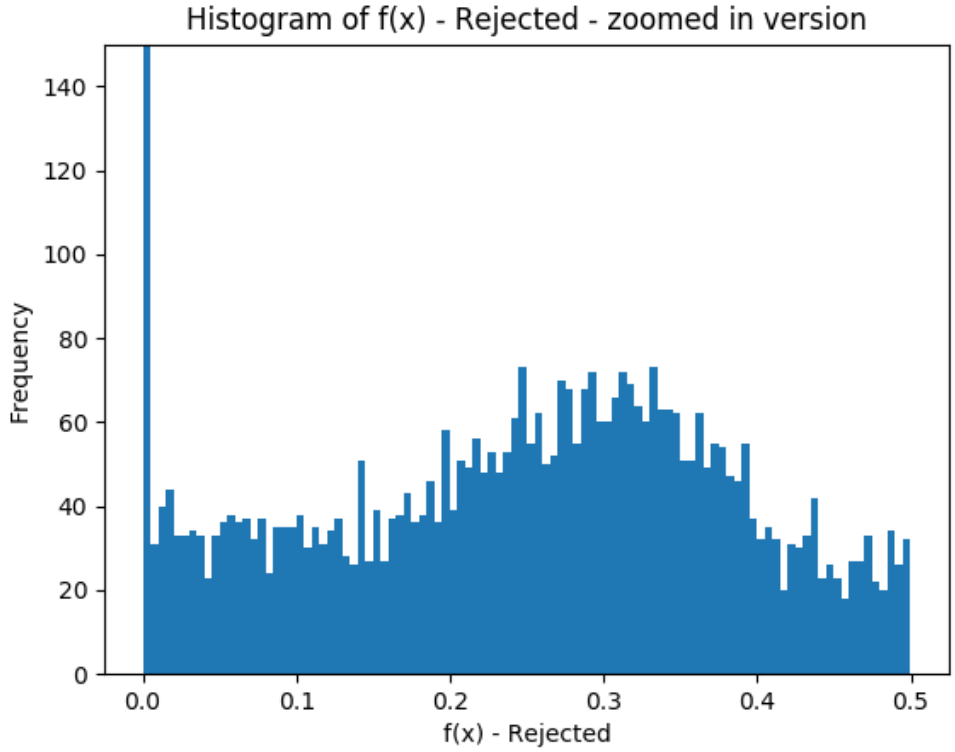
- Average number of samples rejected **9.864**
- Rejected rate **0.907 (90.7%)**

```
f(x):  0
 -- Reject
f(x):  0
 -- Reject
f(x):  0
 -- Reject
f(x):  0
 -- Reject
f(x):  0.272863411267
 -- Reject
f(x):  0
 -- Reject
f(x):  0.236252603659
 -- Reject
f(x):  0.197254258642
 -- Reject
f(x):  0.240528495805
 -- Reject
f(x):  0
 -- Reject
f(x):  0
 -- Reject
f(x):  0
 -- Reject
f(x):  [ 0.24480605]
 -- Reject
f(x):  [ 0.10142169]
 -- Reject
f(x):  0
 -- Reject
f(x):  0
 -- Reject
f(x):  0
 -- Reject
f(x):  0
 -- Reject
f(x):  0.280485445009
 -- Accept
Total Number of Iterations Took:  10864
Average number of samples rejected 9.864
Rejected rate 0.90795287187
```



Thus, the double rejection routine was implemented. The distribution of f(x) generated is shown and is seen to be bi modal. The two modes are 0 and approximately 0.32. The histogram and iteration results are shown above. The distribution of accepted and rejected f(x) samples are also given below.


Histogram of f(x) - Accepted
Frequency
0
5
10
15
20
25
0.1
0.2
0.3
0.4
0.5
f(x) - Accepted


Histogram of f(x) - Rejected - zoomed in version
Frequency
0
20
40
60
80
100
120
140
0.0
0.1
0.2
0.3
0.4
0.5
f(x) - Rejected

CODES

PROBLEM 1:

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Thu Feb 15 18:22:35 2018

@author: abinaya
"""

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from itertools import groupby, combinations
import networkx as nx

def returnBernoulliTrials(k, p):
    uniform_trials= np.random.uniform(size = k)
    bernoulli_trials = np.copy(uniform_trials)
    bernoulli_trials[uniform_trials <= p] = 1
    bernoulli_trials[uniform_trials > p] = 0
    return bernoulli_trials

def returnGraph(nodes_list,edges_list):
    G = nx.Graph()
    G.add_nodes_from(nodes_list)
    G.add_edges_from(edges_list)
    return G


def networking(n,p):
    ### Number of edges that can be selected
    N = (n * (n - 1)) / 2
    edge_df = pd.DataFrame(list(combinations(range(1,n+1) , 2)), columns=['Node1','Node2'])
    edge_df['Edges'] = edge_df[['Node1', 'Node2']].apply(tuple, axis=1)

    bernoulli_trials = returnBernoulliTrials(N, p)
    edge_df['Selection'] = bernoulli_trials

    ### Form Nodes and Edges
    nodes_list = range(1,n+1)
    edge_df_edgepresent = edge_df.loc[edge_df['Selection'] == 1]
```

```
    ### Form Graphs
    G = returnGraph(nodes_list,list(edge_df_edgepresent['Edges']))

    ### Graph Visualization
    plt.figure()
    nx.draw(G,with_labels = True)

    ### Get Degree of Vertex
    vertex_df = pd.DataFrame(nodes_list, columns=['Vertex'], index=nodes_list)
    vertex_df['DOV'] = edge_df_edgepresent['Node1'].value_counts().sort_index()

    ### Get Hisogram of DOV
    hist_dov = vertex_df['DOV'].value_counts().sort_index()
    plt.figure()
    plt.bar(hist_dov.index, hist_dov.values)
    plt.title('Histogram of Degree of Vertex n = '+ str(n)+' p = '+str(p))
    plt.xlabel('Degree Of Vertex')
    plt.ylabel('Frequency')


### Given parameters
networking(50,0.02)
networking(50,0.09)
networking(50,0.12)
networking(100,0.06)
```

PROBLEM 2:

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Thu Feb 15 23:13:05 2018

@author: abinaya
"""
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from itertools import groupby
from scipy.stats import *

k = 1000
```

```
theta = 0.2
no_of_trials = 1

### Generate Uniform samples
uniform_trials= np.random.uniform(size = k)

### Generate Exponential samples from Uniform samples -- Observed
exponential_trials = - (theta) * np.log(1 - uniform_trials)
plt.figure()
observed_bin_hist = plt.hist(exponential_trials, bins=20)[0]
plt.title('Histogram of Observed Samples from Inverse CDF method')

### Generate Exponential samples using inbuilt functions -- Expected
expected_exponential_trials = np.random.exponential(scale = theta, size=k)
plt.figure()
expected_bin_hist = plt.hist(expected_exponential_trials, bins=20)[0]
plt.title('Histogram of Expected Samples using Python In-Built Function')

### Chisquare Goodness of Fit Test
chisq, p = chisquare(observed_bin_hist[:5], expected_bin_hist[:5])
print chisq, p

ks, p = kstest(exponential_trials, cdf='expon')
print ks, p

### Histogram of count
count = []
temp_sum = 0
temp_count = 0
for i in exponential_trials:
   temp_sum += i
   temp_count += 1
   if temp_sum > 1:
      count.append(temp_count)
      temp_sum = 0
      temp_count = 0

plt.figure()
plt.hist(count, bins=50)
plt.title('Histogram of Time Intervals that Occur in 1 Time Count')
```

PROBLEM 3:

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Sun Feb 18 23:03:31 2018

@author: abinaya
"""

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)


### f(x) generation using x
def 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
   return function_of_x

### 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 (sum(accept_reject) < 1000):

   x = np.random.uniform(low=a, high=b, size=1)
   function_of_x = returnFunctionOfx(x)
   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 Took: ", len(accept_reject)
print "Average number of samples rejected",np.mean(number_of_rejections)
print "Rejected rate", float(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 - zoomed in version")
plt.ylim([0,150])
function_of_x_array = np.array(function_of_x_list)
accept_reject_array = np.array(accept_reject)
plt.figure()
plt.hist(function_of_x_array[accept_reject_array==1], bins=100)
plt.xlabel("f(x) - Accepted")
plt.ylabel("Frequency")
plt.title("Histogram of f(x) - Accepted")

plt.figure()
```

```
plt.hist(function_of_x_array[accept_reject_array==0], bins=100)
plt.xlabel("f(x) - Rejected")
plt.ylabel("Frequency")
plt.title("Histogram of f(x) - Rejected - zoomed in version")
plt.ylim([0,150])
```