EE 511 Project #2

Name: Yang Liu

Student ID number:5847572002

Due: Oct 7

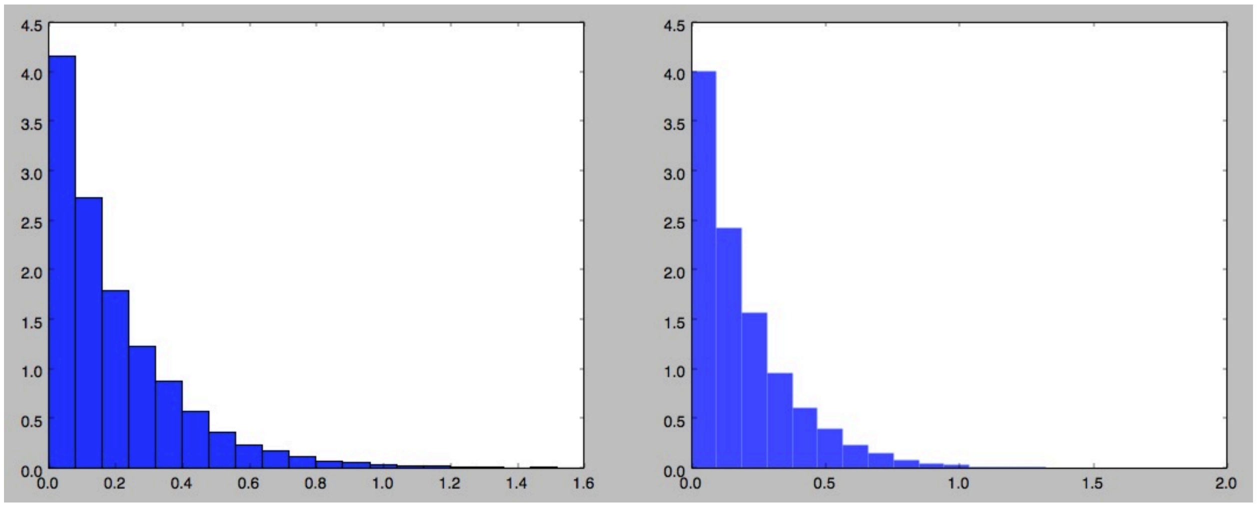
problem 1

The main point in problem 1 is to generate exponential random variables using the inverse CDF method. There are 3 steps in this problem.

1. Generate u~U(0,1).
2. Calculate X= inverse F(u)

in this case the formula I used is E=-(1/lam)*log(1-u)

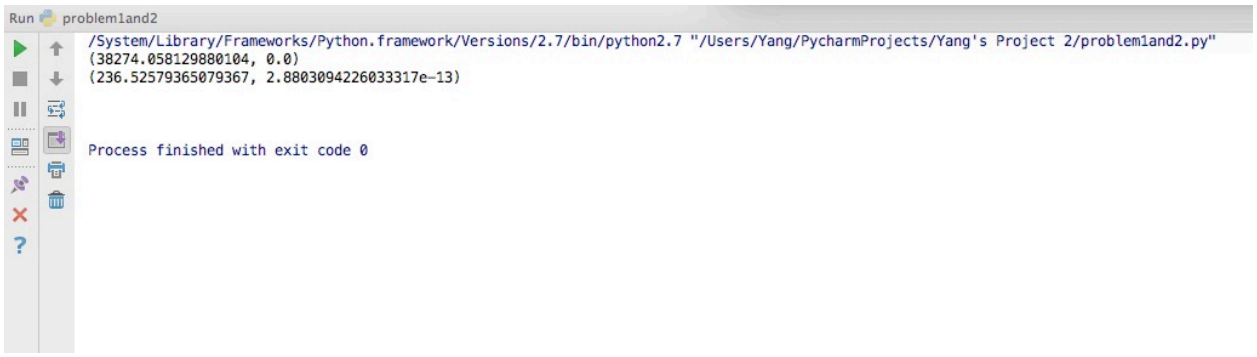
3. Compared the exponential random variable we just generated with expected exponential random variable. I plot both of them in one figure.



the left one is the exponential samples I produced using the inverse CDF method while the right one is from the function of np.random.exponentia.

4. Use chi2 test method to accurately check the samples of the exponential random variable and print the result. The first returned value of the chi2 function in python is the χ^2 statistic, and the second one is p-value of the test. If the p-value is equal to 0, then we could consider the samples of random variable we

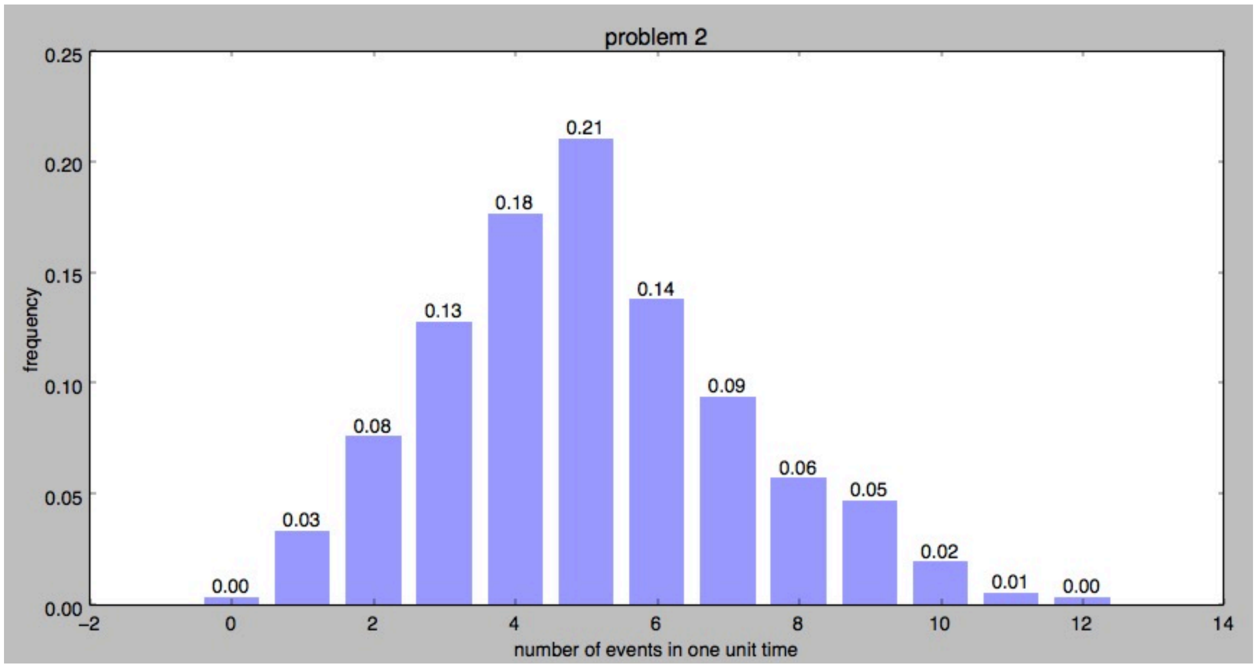
generated are exponential random variables. The p-value of the result in of problem is equal to zero, so I have generated exponential random variable correctly.



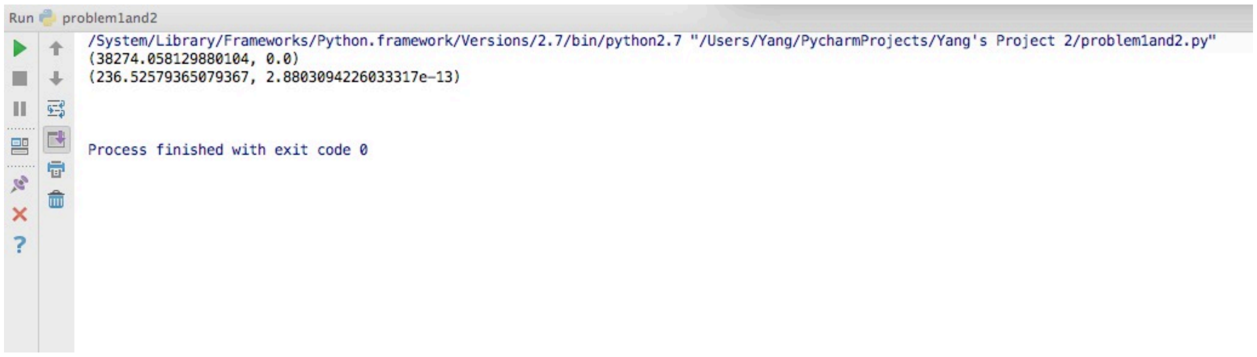
```
/System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 "/Users/Yang/PycharmProjects/Yang's Project 2/problem1and2.py"
(38274.058129880104, 0.0)
(236.52579365079367, 2.8803094226033317e-13)


Process finished with exit code 0
```

problem 2

The most difficult point in problem 2 is to write the algorithm to count the numbers of events in unit time. I mainly used ‘for’ and ‘while’ loop to do the counting, and used some variables as cache to store the timer, the cumulative number and the decimal time. In the loop, I compared the decimal time with 1 to determine whether to continue counting or get out of the loop.



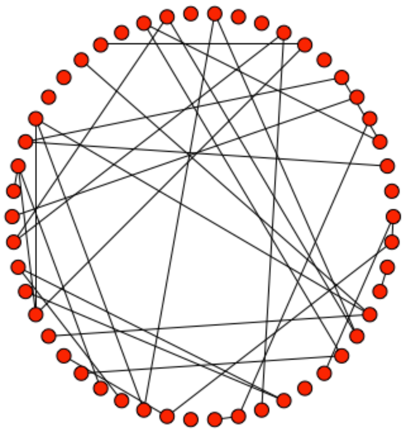
After the counting, I calculated the statistics and show it in a histogram. According to observation and discussion with classmates, the most possible distribution of this histogram is Poisson distribution. So I used chi2 test again to check whether this distribution belongs to Poisson distribution, and I used np.random.poisson fuction to generate a passion distribution for comparison. The result shows that the distribution is quite likely belongs to Poisson distribution because the p-value returned as follows is very close to zero.

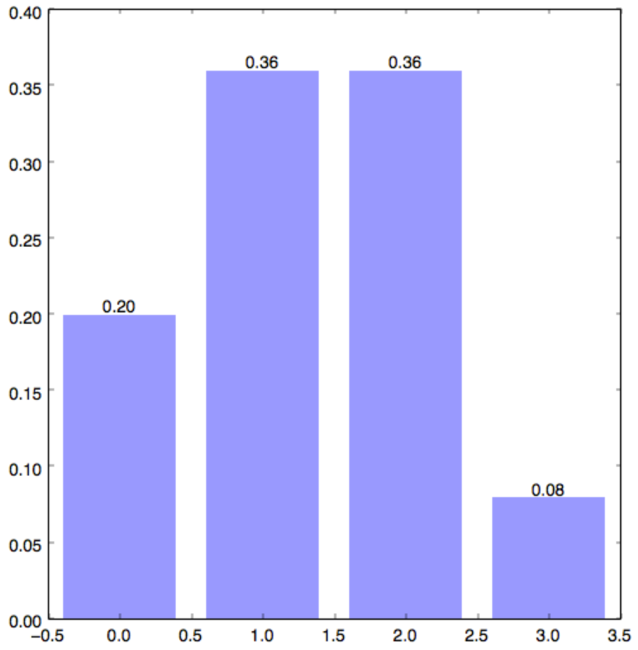


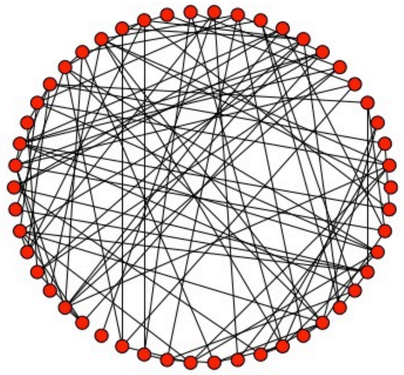
```
/System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 "/Users/Yang/PycharmProjects/Yang's Project 2/problem1and2.py"
(38274.058129880104, 0.0)
(236.52579365079367, 2.8803094226033317e-13)


Process finished with exit code 0
```

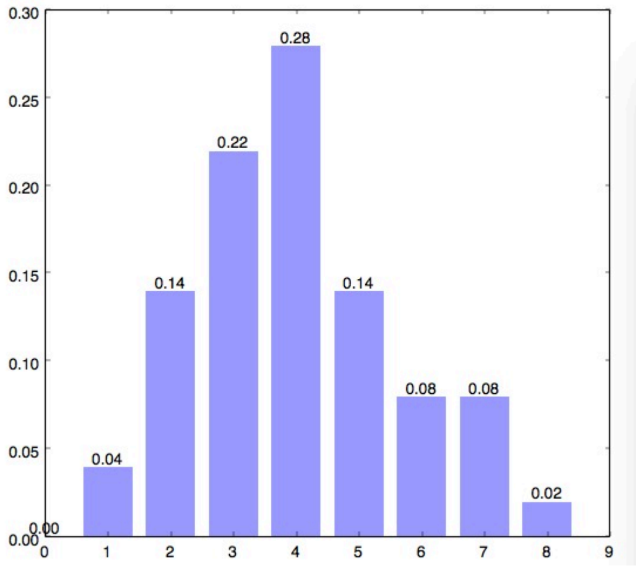
problem 3

Problem 3 is the easiest one in the assignment if we become familiar with the module called networkx. The problem actually asks us to draw two ER graphs with the parameter n and p equal to (50,0.02) and (50,0.09).
I used the fuction nx.random_graphs.erdos_renyi_graph to get the ER graphs, and I used function nx.degree_histogram to count the degrees of the nodes. I also used a shell layout to show the structure of the network. Moreover, we could learn the structure from the histogram in problem 4 as well.



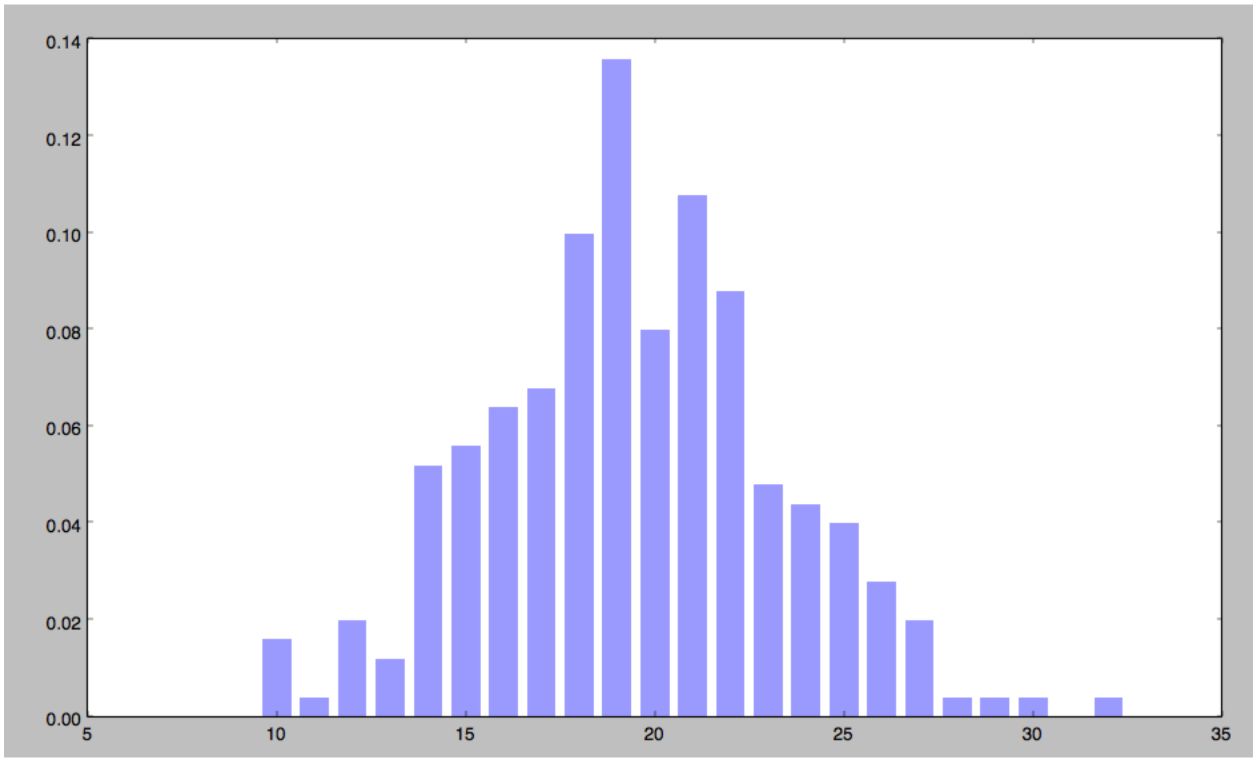






Problem 4

This problem is about larger scale of network.Firstly I plot the histogram of the two degrees of (which I plot in the same graph in problem 3). Then I do the same generation and counting of the network with the parameter n and p equal to (250,0.08)



It has been given that Vertex degree is a binomialy distributed statistic for small networks. So I check it with my histogram. which really looks like binomial distribution. However, the first one is no as stable as others, I think the reason is that the parameters of possibility and numbers are both too small. And the last one in problem 4, which has the lager n and p looks likely to not only binominal but also Poisson distribution as well. The theoretical proof is given as follow.

An important property of the Poisson random variable is that it may be used to approximate a binomial random variable when the binomial parameter n is large and p is small. To see this, suppose that X is a binomial random variable with parameters (n, p), and let $\lambda = np$. Then

$$\begin{aligned}
P\{X = i\} &= \frac{n!}{(n-i)!\,i!} p^i (1-p)^{n-i} \\
&= \frac{n!}{(n-i)!\,i!} \left(\frac{\lambda}{n}\right)^i \left(1 - \frac{\lambda}{n}\right)^{n-i} \\
&= \frac{n(n-1)\cdots(n-i+1)}{n^i} \frac{\lambda^i}{i!} \frac{(1-\lambda/n)^n}{(1-\lambda/n)^i}
\end{aligned}$$

Now, for n large and p small

$$\left(1 - \frac{\lambda}{n}\right)^n \approx e^{-\lambda}, \qquad \frac{n(n-1)\cdots(n-i+1)}{n^i} \approx 1, \qquad \left(1 - \frac{\lambda}{n}\right)^i \approx 1$$

Hence, for n large and p small,

$$P\{X = i\} \approx e^{-\lambda} \frac{\lambda^i}{i!}$$

source code

problem 1 and 2

```
import matplotlib.pyplot as plt
import numpy as np
import scipy
from scipy.stats import chisquare
import scipy.stats.mstats as mst

plt.figure(1)
plt.subplot(1,2,1)
u = np.random.uniform(0,1,10000)
np.all(u >= 0)
np.all(u < 1)

lam=5.0
E=-(1/lam)*np.log(1-u)
count, bins, ignored = plt.hist(E, 20, normed=True)

plt.subplot(1,2,2)
Y = np.random.exponential(1/lam,10000)
plt.hist(Y,20, normed=True,lw=0,alpha=.8)
plt.show()


observed=E
expected=Y
test0=chisquare(observed,expected)
print(test0)

plt.figure(2)
i=0
timer=0
t=0
count=0

qq= []
for t in range(0,1000,1):

    while (timer <= 1):
        timer = timer + E[i]
        i=i+1

        count=count+1
    qq.insert(t, count-1)

    #print t
    #print len(qq)

    if (timer > 2):

        qq.append(0)
        timer = timer - 1
    if (1<timer<=2):
        count=1
        timer=timer-1
    #print qq[t-1]
    #print qq[t]
l=len(qq)
pp=np.random.poisson(lam,l)
#print(qq)
#test1=chisquare(qq,pp)
test0=mst.chisquare(np.array(qq), np.array(pp))

print(test0)


unique,counts=np.unique(qq,return_counts=True) #numpy version no lower than 1.9
#print unique
#print counts/1000.0
plt.bar(unique-0.4,+counts/float(l),facecolor='#9999ff',edgecolor='white')
for x, y in zip(unique, counts/float(l)):
    plt.text(x , y + 0.002, '%.2f' % y, ha='center', va='bottom')
plt.ylim(0,0.25)
plt.xlabel('number of events in one unit time')
plt.ylabel('frequency')
plt.title('problem 2')
plt.show()
print "\n"
```

problem 3 and 4

```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import chisquare
import scipy.stats.mstats as mst

plt.figure(1)
plt.subplot(1,2,1)

ER = nx.random_graphs.erdos_renyi_graph(50,0.02)
pos = nx.circular_layout(ER)
nx.draw(ER,pos,with_labels=False,node_size = 100)
plt.subplot(1,2,2)
degree = nx.degree_histogram(ER)
a = range(len(degree))
b = [z / float(sum(degree)) for z in degree]
from numpy import array
x= array(a)
y= array(b)
plt.bar(x-0.4 ,y,facecolor='#9999ff',edgecolor='white')
for x, y in zip(a, b):
    plt.text(x , y + 0.002, '%.2f' % y, ha='center', va='bottom')
plt.show()


plt.figure(2)
plt.subplot(1,2,1)
```

```
ER2 = nx.random_graphs.erdos_renyi_graph(50,0.09)
pos = nx.shell_layout(ER2)
nx.draw(ER2,pos,with_labels=False,node_size = 100)
plt.subplot(1,2,2)
degree = nx.degree_histogram(ER2)
a = range(len(degree))
b = [z / float(sum(degree)) for z in degree]
from numpy  import array
x= array(a)
y= array(b)
plt.bar(x-0.4 ,y,facecolor='#9999ff',edgecolor='white')
for x, y in zip(a, b):
    plt.text(x , y + 0.002, '%.2f' % y, ha='center', va='bottom')
plt.show()


plt.figure(3)

ER3 = nx.random_graphs.erdos_renyi_graph(250,0.08)
#pos = nx.shell_layout(ER3)
#nx.draw(ER3,pos,with_labels=False,node_size = 10)
#plt.show()
#plt.figure(4)
degree = nx.degree_histogram(ER3)
a = range(len(degree))
b = [z / float(sum(degree)) for z in degree]
from numpy  import array
x= array(a)
y= array(b)
plt.bar(x-0.4 ,y,facecolor='#9999ff',edgecolor='white')
#for x, y in zip(a, b):
   # plt.text(x , y + 0.002, '%.2f' % y, ha='center', va='bottom')
plt.show()

kk=[]
i=0
for i in range(0,250,1):
    t= ER3.degree(i)
    kk.append(t)
print kk
Y=np.random.binomial(1,0.08,250)
observed=kk
expected=Y
test0=mst.chisquare(np.array(observed), np.array(expected))
#chisquare(observed,expected)
print(test0)
```