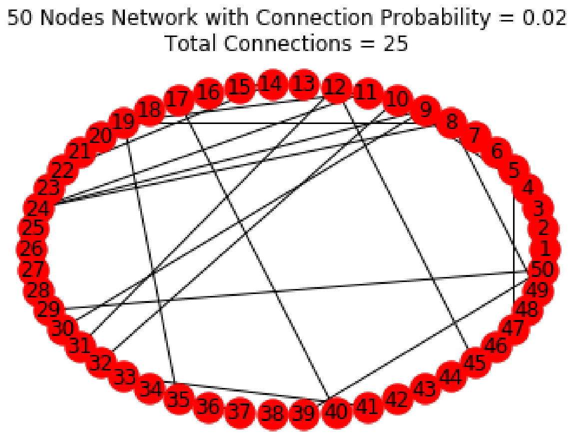
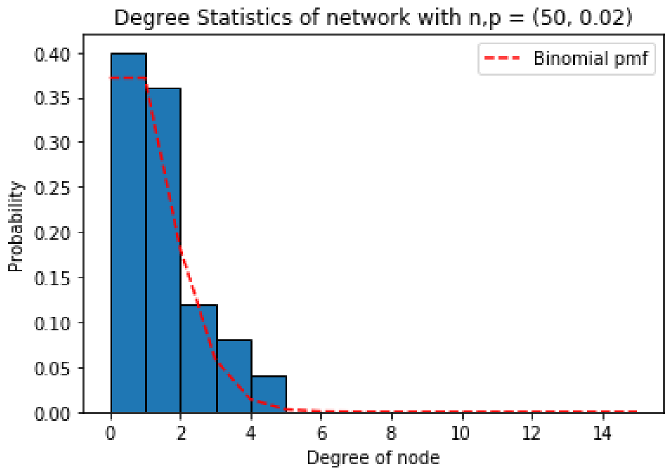
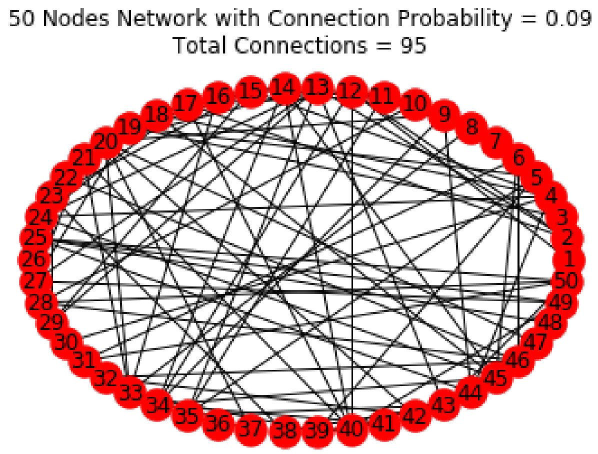
EE511 Project 2

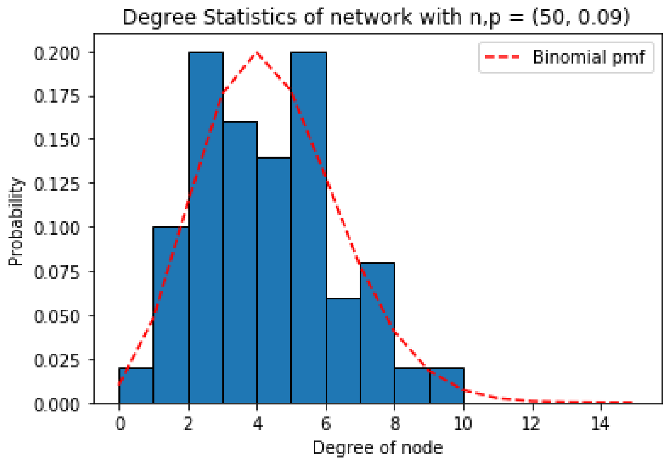
[Networking Again]

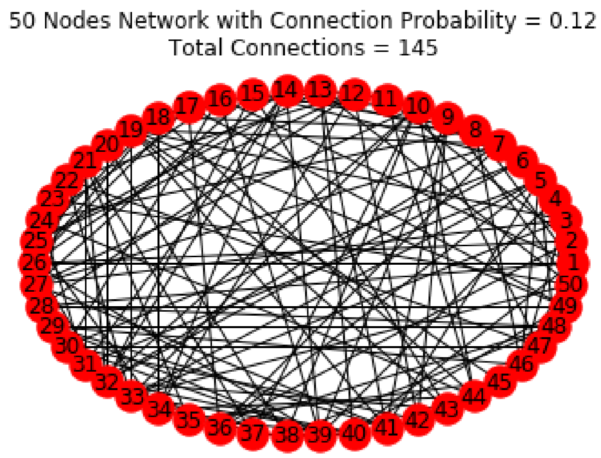
The following three groups of plots. The left column are undirected networks with n = 50 and p = {0.02, 0.09, 0.12}. The right column is histograms of vertex degrees, corresponding to networks in left column. We can clearly see that connections go denser as p increased.

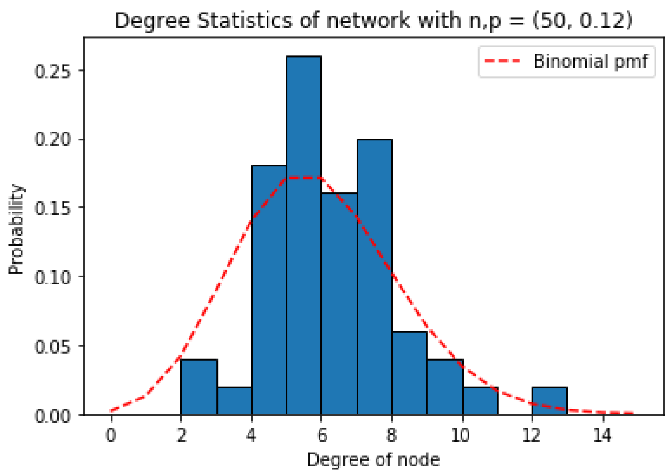


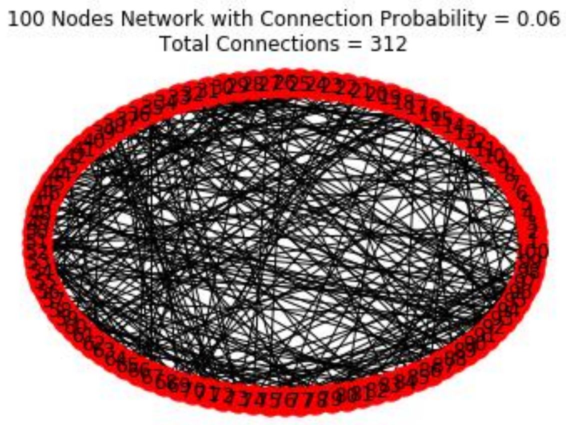


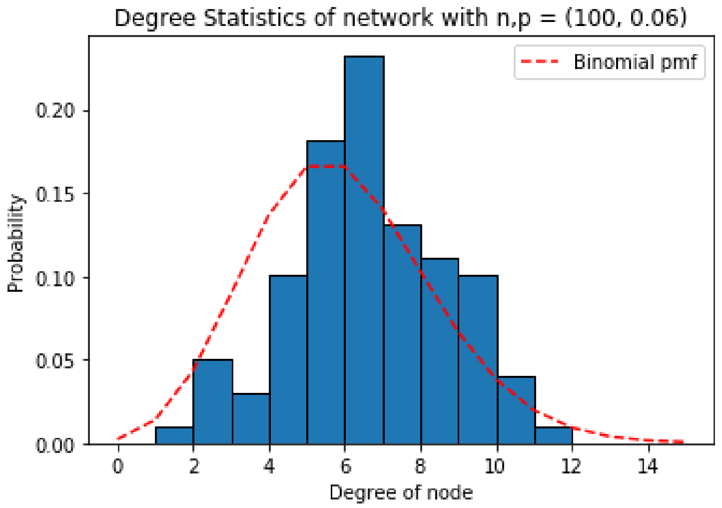












The above plot is the network with n = 100, p = 0.06. The right side is the statistics of degrees of all nodes (normalized to 1).

Comparing with theoretical binomial distribution, the degree approximately fit the binomial PMF with n = (100 - 1) and p = 0.06. I use Chi Square Test to verify goodness of fit. The result show that my assumption could be accepted.

```
Power_divergenceResult(statistic=0.11957846457028917, pvalue=0.9999999999994857)
```

[Waiting]

From exponential distribution CDF, the inverse CDF is

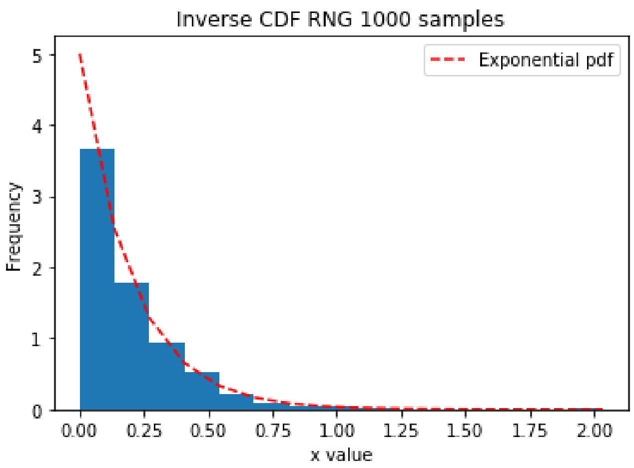
$$f^{-1}(u) = \ln(1-u)\,/-\lambda$$

Hence, the r.v. generator could be designed as:
Step1: Generate U in (0,1)
Step2: Calculate Y = $\ln(1-U)\,/-\lambda$, then set X = Y

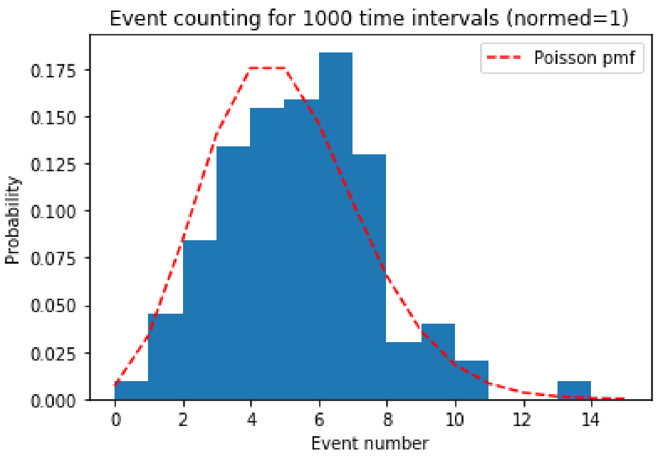
The following plot 1 is the histogram of the generated 1000 samples and theoretical exponential distribution It looks like that the 1000 samples are exponential distribution. I use Kolmogorov–Smirnov test to evaluate the goodness of fit. Here is the statistic result:

```
KstestResult(statistic=0.01763419523941629, pvalue=0.9149363140540462)
```

Since P-Value is far bigger than statistical value, then we can accept the hypothesis that the generated r.v. is an exponential distribution.



Plot 1



Plot 2

Plot 2 is the histogram of event counting result. It looks like a Poisson distribution which descripts the number of event occurred in given time. I use Chi Square test to verify this hypothesis. The result is:

```
Power_divergenceResult(statistic=0.11353000645493878, pvalue=0.9999999999996414)
```

This shows that the hypothesis could be accepted since P-Value is far bigger than statistic value.

[Double Rejection]

The top-level envelope design:

The X is binomial distribution of equal weighted Beta and Triangle distribution. The total (integral) probability of Beta is p=0.5 and the same for Triangle. Therefore, the top-level generator could be designed as:

Step 1: Generate U from (0,1)
Step 2: If U <= 0.5, go to Beta envelope to generate. Otherwise go to Triangle envelope to generate.

Beta envelope design:

f(x) = 0.5 Beta(8,5) = $0.5x^7(1-x)^4$ Since this r.v. concentrate on (0,1], we can assume reject function g(x)=1 0<x<=1. To determine c, we need to calculate maximum of f(x).

$$\frac{d}{dx}\left(\frac{f(x)}{g(x)}\right) = 0.5(7x^6(1-x)^4 - 4x^7(1-x)^3)$$

Let this equation = 0, we can get the maximum value c = 0.0003694679 when x = 7/11. Therefore,

$$\left(\frac{f(x)}{cg(x)}\right) = 2706.595 * 0.5x^7(1-x)^4$$

Then the rejection routine will be:
Step 1: Generate U_1 and U_2
Step 2: If $U_2 \le 2706.595 * 0.5U_1^7(1-U_1)^4$, accept U_1 and set X = U_1. Otherwise reject U_1 and return to step1

Triangle envelope design:

Let g(x)=(x-4)/2 such that 0<g(x)<=1.

$$\left(\frac{f(x)}{g(x)}\right) = \begin{cases} 1 & 4 < x \le 5 \\ (6-x)/(x-4) & 5 < x \le 6 \end{cases} \le m = 1$$

Then rejection routine will be:
Step 1: Generate U1 and set Y=2U1 + 4
Step 2: Generate U2
Step 3: If U2 $\le \begin{cases} 1 & 4 < Y \le 5 \\ (6-Y)/(Y-4) & 5 < Y \le 6 \end{cases}$ set X=Y. Otherwise, go to step1

The result of this RNG is:

```
Total Rejection Rate: 1.146
Beta Acceptance: 497
Beta Rejections: 939
Beta Rejection rate: 1.8893360161
Triangle Acceptance: 503
Triangle Rejections: 207
Triangle Rejection rate: 0.411530815109
```

```python
#Author: Yong Wang <yongw@usc.edu>
#Copyright reserved
#https://github.com/uscwy/EE511Project2/blob/master/project2.py


import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
import scipy.stats


def generate_network(n, p):
    net = nx.Graph()
    net.clear()
    net.add_nodes_from(range(1, n+1))
    for i in range(1,n+1):
        for j in range(i+1,n+1):
            #randomly make connection between peoples
            if np.random.random() <= p:
                net.add_edge(i, j)
    return net


def draw_network(n, p):

    net = generate_network(n, p)

    plt.subplot(1,1,1)
    nx.draw_circular(net, with_labels=True)
    plt.title(str(n) + " Nodes Network with Connection Probability = " + str(p)
              + "\nTotal Connections = " + str(net.number_of_edges()))
    plt.show()

    d=[]
    for i in range(1, n+1):
        d.append(net.degree(i))



    o, bins, patches = plt.hist(d, range(0,16), normed=1, edgecolor='black')
    plt.xlabel('Degree of node')
    plt.ylabel('Probability')
    plt.title('Degree Statistics of network with '
             + 'n,p = ('+str(n)+', '+str(p)+')')

    y = scipy.stats.binom.pmf(bins, n-1, p)
    plt.plot(bins, y, 'r--', label='Binomial pmf')
    plt.legend()
    plt.show()
```

```
    print scipy.stats.chisquare(o,y[0:15])


    return d


def inver_CDF_RNG(num, lam=5.0):
    xs = np.empty(num)
    us = np.random.uniform(size=num)
    for i in range(0, num):
        #caculate x from inverse CDF of exponential distribution (lambda=5)
        xs[i] = np.log(1.0 - us[i])/-lam
        #print us[i]
    return xs


def count_number_of_intervals(intervals):
    count = []
    t = 0
    c = 0
    for i in range(0, len(intervals)):
        t = t + intervals[i]
        if t < 1.0:
            c = c + 1
        else:
            count.append(c)
            t = t - 1
            c = 1
            while(t > 1):
                t = t - 1
                count.append(0) #0 event occured



    return count


def question_1():


    draw_network(50, 0.02)
    draw_network(50, 0.09)
    draw_network(50, 0.12)
    draw_network(100, 0.06)

def question_2():
```

```
    lam = 5.0

    xs = inver_CDF_RNG(1000)

    ob, bins, pactches = plt.hist(xs, 15, normed=1)
    plt.ylabel("Frequency")
    plt.xlabel("x value")
    plt.title("Inverse CDF RNG 1000 samples")
    y = scipy.stats.expon.pdf(bins, 0, 1/lam)
    plt.plot(bins, y, 'r--', label='Exponential pdf')
    plt.legend()
    plt.show()

    ef = scipy.stats.expon(loc=0, scale=0.2)
    print scipy.stats.kstest(xs, ef.cdf)

    count = count_number_of_intervals(xs)

    o, bins, patches = plt.hist(count, 15, range=(0,15), normed=1)
    plt.ylabel("Probability")
    plt.xlabel("Event number")
    plt.title("Event counting for 1000 time intervals (normed=1)")

    y = scipy.stats.poisson.pmf(bins,5.0)
    plt.plot(bins, y, 'r--', label='Poisson pmf')
    plt.legend()
    plt.show()


    print scipy.stats.chisquare(o,y[0:15])


def beta_envelope():
    c = 0.5*(7.0/11)**7*(1-7.0/11)**4
    retry = 0
    rej = 0


    while(retry<10000):
        retry = retry + 1
        u = np.random.rand(2)
        fx = (1/c)*0.5*(u[0]**7)*(1-u[0])**4
        if u[1] <= fx :
            #print u[0], u[1], fx*c;
            return u[0], rej
        else:
            rej = rej + 1
```

```
    return -1, rej


def triangle_envelope():
    rej = 0
    retry = 0

    while(retry<10000):
        retry = retry + 1
        u = np.random.rand(2)
        y = 2*u[0] + 4
        if u[1] <= 1 and y >4 and y<=5 :
            return u[0], rej
        elif u[1] < (6-y)/(y-4):
            return u[0], rej
        else:
            rej = rej + 1


    return -1,rej


def question_3():
    xs = np.empty(1000)
    i = 0
    beta_rej = 0
    tri_rej = 0
    beta_ac = 0
    tri_ac = 0



    while(i<1000):
        #Generate r.v. using Beta and triangle envelope with equal weight
        if(np.random.rand()) < 0.5:
            y, rej = beta_envelope()
            beta_rej = beta_rej + rej #count rejection
            if y >= 0:
                xs[i] = y
                i = i + 1
                tri_ac = tri_ac + 1
        else:
            y, rej = triangle_envelope()
            tri_rej = tri_rej + rej #count rejection
            if y >= 0:
                xs[i] = y
                i = i + 1
                beta_ac = beta_ac + 1
```

```
    print "Total Rejection Rate:", float(beta_rej+tri_rej)/(tri_ac+beta_ac)

    print "Beta Acceptance:", beta_ac
    print "Beta Rejections:", beta_rej
    print "Beta Rejection rate:", float(beta_rej)/beta_ac


    print "Triangle Acceptance:", tri_ac
    print "Triangle Rejections:", tri_rej
    print "Triangle Rejection rate:", float(tri_rej)/tri_ac



if __name__ == "__main__":
    question_1()
    question_2()
    question_3()
```