EE511_Project2

February 28, 2019

In [6]:
```
from scipy.stats import beta
from scipy.stats import binom
from scipy.stats import poisson
import networkx as nx
import numpy as np
import random
import matplotlib.pyplot as plt
import math

%matplotlib inline


def function_b(x, y):
    return math.factorial(x - 1) * math.factorial(y - 1)
/ math.factorial(x + y - 1)


class MyBeta:

    def __init__(self, a, b):
        self.a = a
        self.b = b

    def pdf(self, x):
        return math.pow(x, self.a-1) * math.pow(1-x, self.b-1)

    def rv(self):
        mode = (self.a - 1) / (self.a + self.b - 1)
        maximum = self.pdf(mode)
        while True:
            x1 = random.uniform(0, 1)
            y1 = random.uniform(0, maximum)
            if self.pdf(x1) < y1:
                continue
            else:
                return x1
```

```
class BimodalPdf:

    def __init__(self, a, b, size=1000):
        self.a = a
        self. b = b
        self.size = size

    def left_modal(self):
        mode = (self.a - 1) / (self.a + self.b - 1)
        maximum = 0.5 * beta.pdf(mode, self.a, self.b)
        rej = 0
        while True:
            x1 = random.uniform(0, 1)
            y1 = random.uniform(0, maximum)
            if 0.5 * beta.pdf(x1, self.a, self.b) < y1:
                rej += 1
                continue
            else:
                return x1, rej

    def right_modal(self):
        def cdf_2(x):
            if x <= 5:
                return 0.5 * (x - 4)
            elif 5 < x <= 6:
                return -0.5 * (x - 6)
            else:
                return 0

        maximum2 = 0.5
        rej2 = 0
        while True:
            x2 = random.uniform(4, 6)
            y2 = random.uniform(0, maximum2)
            if cdf_2(x2) < y2:
                rej2 += 1
                continue
            else:
                return x2, rej2

    def rv(self):
        u = random.uniform(0, 1)
        if u <= 0.5:
            return self.left_modal()
        else:
            return self.right_modal()
```

```
def plot_pdf(self):
    size = self.size
    left_rej = 0
    right_rej = 0
    out_list = []
    while size:
        x1, rej = self.left_modal()
        size -= 1
        out_list.append(x1)
        left_rej += rej
    p, t = np.histogram(out_list, bins=100, range=(0, 1))
    ind = np.arange(0, 1, 0.01)
    width = 0.01
    plt.bar(ind, p, width)

    size = self.size
    out_list = []
    while size:
        x2, rej2 = self.right_modal()
        size -= 1
        out_list.append(x2)
        right_rej += rej2

    p, t = np.histogram(out_list, bins=200, range=(4, 6))
    ind = np.arange(4, 6, 0.01)
    width = 0.01
    plt.bar(ind, p, width)
    plt.title(' A bimodal distribution')
    plt.show()
    print("The number of rejected candidates in first
          envelope is", left_rej)
    print("The number of rejected candidates in second
          envelope is", right_rej)
    print("The The rejection rate is", (right_rej+left_rej)/2000.0)
```

Implement rejection sampling routines for X. Generate 1000 samples of the random variable using each envelope. Track the rejection rate of your rejection sampling RNGs. The rejection rate is the average number of rejected candidates per sample. This is a measure of the efficiency of your RNG. 1.Simulation Methodology

I create a class named BimodalPdf to generate a random variable X has a bimodal distribution via rejection method. This bimodal distribution in ourexperiment is made up of an equally weighted, convex summation of a beta and a triangle distribution. To be specific, I initiate the class with parameter a and b which specify the beta distribution and size which denotes the numbers of sampling. Since the area of beta distribution equals to the area of triangle distribution which are 1/2. I use uniform function to generate a random number between 0 and 1. If the number less than 0.5, I assume the routine generate a beta distribution. On the other hand, if the number is not less than 0.5, I assume it generate a triangle distribution. And I count the number of points rejected in each envelope.

2.Theoretical Exploration or Analysis

Suppose that we want to generate a random variable X with pdf as the bimodal distribution as followed.

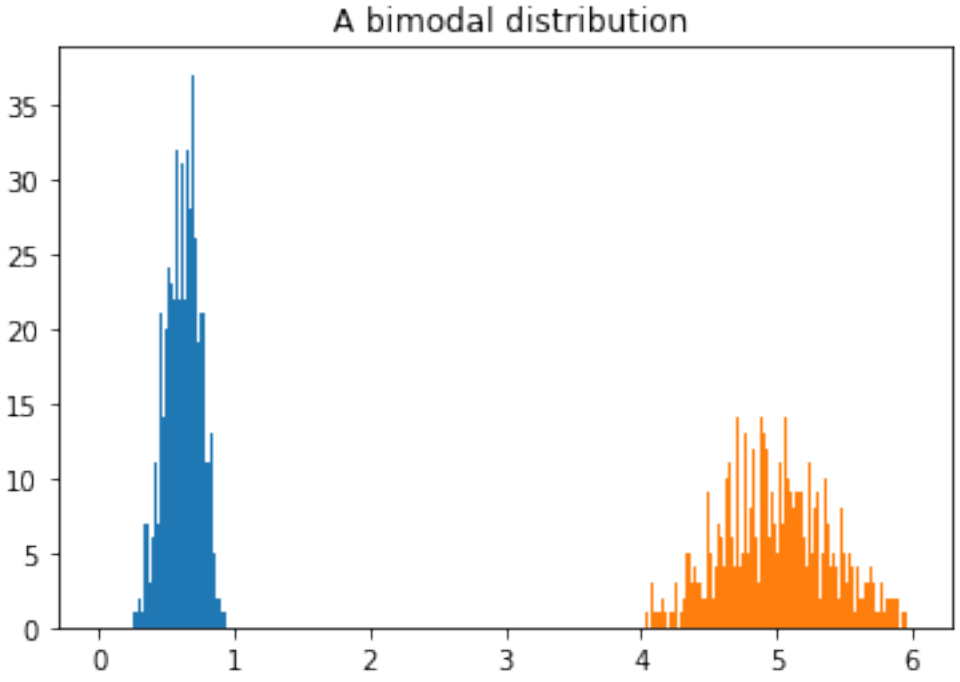
$$f(x) = \begin{cases} 0.5 * Beta(8,5), & \text{if } 0 \leq x \leq 1 \\ 0.5 * (x-4), & \text{if } 4 \leq x \leq 5 \\ -0.5 * (x-6), & \text{if } 5 \leq x \leq 6 \\ 0, & \text{else} \end{cases} \tag{1}$$

In particular, we assume that: (1) the pdf is nonzero only in the interval [a, b], where [a, b] is the abscissa of each envelope. and (2) the pdf takes on values in the range [0, c], where c is the maximum of the desired pdf in each envelope. The rejection method in this case works as follows:

1. Generate uniform in the interval[a, b].
2. Generate Y uniform in the interval[0, c].
3. If Y $\leq f_x(X_1)$ then output Z = X1 ; else, reject X1 and return to step 1.

Apparently, in the first envelope, a=0, b=1, c=$\frac{\alpha-1}{\alpha+\beta-1}$ and in the second envelope a=4, b=6, c=0.5.

```
In [12]: a = 8
         b = 5
         bimodal = BimodalPdf(a, b)
         bimodal.plot_pdf()
```



```
The number of rejected candidates in first envelope is 1750
The number of rejected candidates in second envelope is 1086
The The rejection rate is 1.418
```

```
In [15]: def covariance_statistic():
             size = 1000
             outlist2 = []
             while size:
                 outlist2.append(bimodal.rv()[0])
                 size -= 1

             X_k = np.array(outlist2[0: 995])
             X_k5 = np.array(outlist2[5: 1000])
             X_k_mean = X_k.mean()
             X_k5_mean = X_k5.mean()

             sample_covariance = np.sum(((X_k-X_k_mean)*(X_k5-X_k5_mean)))/(995-1)
             print('The sample convariance is', sample_covariance)

         covariance_statistic()
         covariance_statistic()
         covariance_statistic()
```

```
The sample convariance is 0.04359126084520724
The sample convariance is -0.08114230478641239
The sample convariance is -0.056210804042538316
```

Independence: Internally and Externally 1. Take 1000 samples of the bimodal distribution above. Use the covariance statistic to test the independence between X_k and the lagged version X_{k+5}

1.Theoretical Exploration or Analysis

I use the routine of the bimodal distribution above to generate 1000 samples and then label the first 995 samples as the first random variable and the last 995 samples as the second random variable. FinallyI calculate the sample covariance of these two group via

$$\frac{1}{N-1}\sum_{i=1}^{N}(x_{i1}-\overline{x}_1)(x_{i2}-\overline{x}_2) \tag{2}$$

where N=995 and x_{i1} denotes the ith sample in the first random variable, x_{i2} denotes the ith sample in the seond random variable. If the two variables are independentthe covariance will be 0.

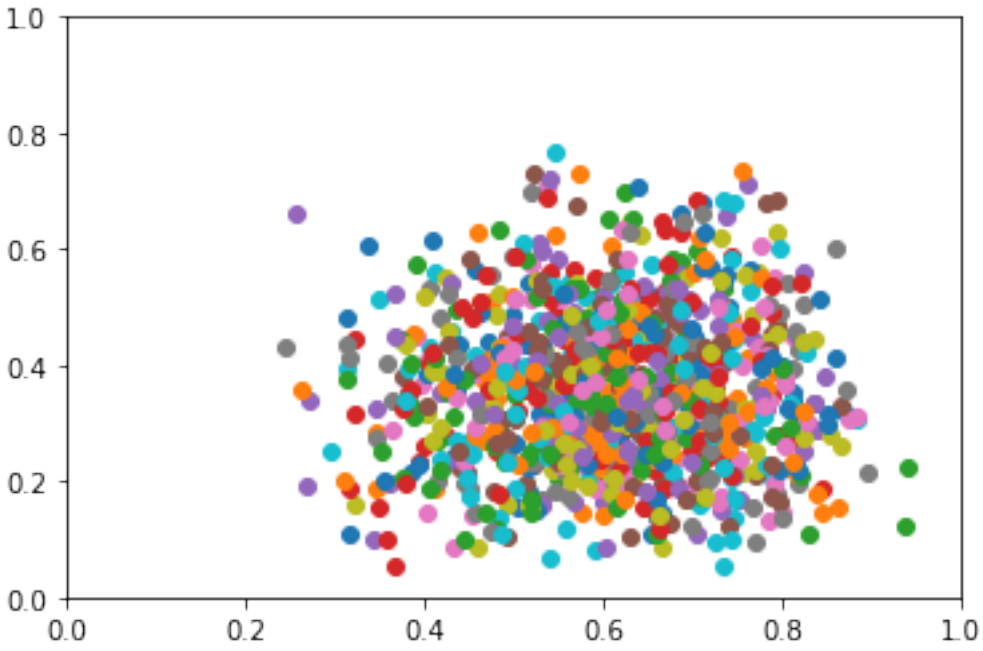
$$\begin{aligned} COV(X,Y) &= E((X-E(X)(Y-E(Y)) &(3)\\ &= E(X-E(X))E(Y-E(Y)) &(4)\\ &= 0 &(5)\end{aligned}$$

2.Experiments and Results

The result of experiment shows that the three sample covariance are 0.04359126084520724, -0.08114230478641239, -0.056210804042538316, which accord with the theoretical value and are close to 0.

```
In [17]: outlist3 = []
         bin_matrix = np.zeros((4, 4))
         for n in range(0, 1000):
             x = beta.rvs(8, 5)
             y = beta.rvs(4, 7)
             outlist3.append((x, y))
             plt.scatter(x, y)
             i = int(x/0.25)
             j = int(y/0.25)
             bin_matrix[i][j] += 1

         plt.xlim(0, 1)
         plt.ylim(0, 1)
         plt.show()

         estimated_bin_matrix = np.zeros((4, 4))
         for i in range(0, 4):
             for j in range(0, 4):
                 estimated_bin_matrix[i][j] = 1000 * (beta.cdf(0.25 * (i+1), 8, 5)
                                                      - beta.cdf(0.25 * i, 8, 5)) * \
                                              (beta.cdf(0.25 * (j+1), 4, 7)
                                               - beta.cdf(0.25 * j, 4, 7))


         res = 0
         for i in range(0, 4):
             for j in range(0, 4):
                 res += np.power((bin_matrix[i][j]
                                  - estimated_bin_matrix[i][j]), 2)
                 /estimated_bin_matrix[i][j]


         def check_answer(res, k=4):
             degree_of_freedom = k - 1
             # The threshold at a 5% significance level of different degree_of_freedom
             chi_squared_dict = {
                 1: 3.84,
                 2: 5.99,
                 3: 7.82,
                 4: 9.49,
                 5: 11.07,
                 6: 12.59,
                 7: 14.07,
```

```
        8: 15.51,
        9: 16.92,
        10: 18.31,
    }
    if res <= chi_squared_dict[degree_of_freedom]:
        probability_of_truth = 0.95
    else:
        probability_of_truth = 0.05
    return probability_of_truth


print('the probability that X and Y are independent is',
      check_answer(res, (4-1)*(4-1)))
```




```
the probability that X and Y are independent is 0.95
```

2 Generate 1000 samples of the random variables X~Beta(8,5) and Y~Beta(4,7). Use 2-way contingency tables to argue for the independence of X and Y.

1.Theoretical Exploration or Analysis

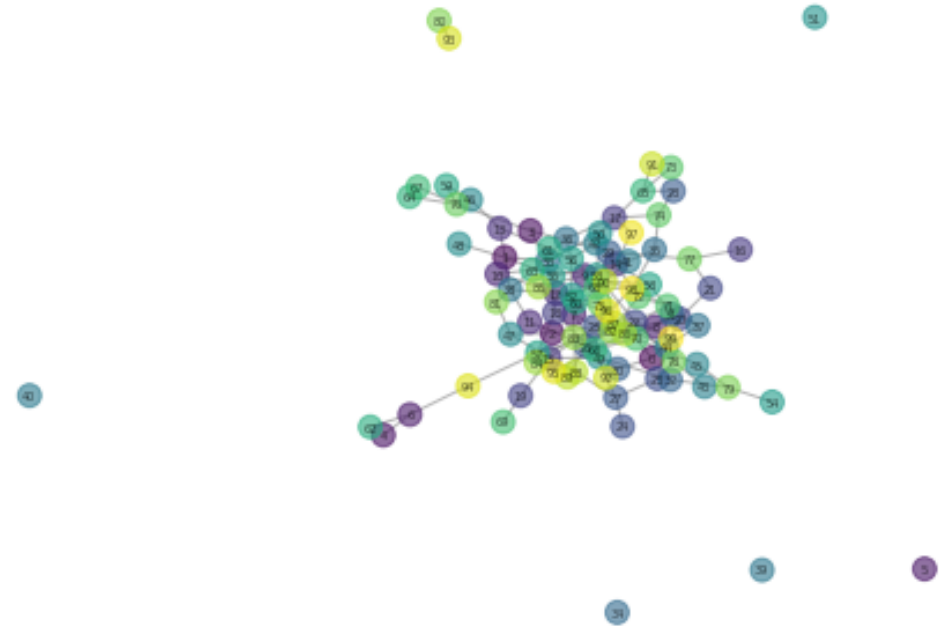
In statistics, a contingency table is a type of table in a matrix format that displays the (multivariate) frequency distribution of the variables. They provide a basic picture of the interrelation between two variables and can help find interactions between them. We combine 2-way contingency tables and chi-squared test to argue for the independence of X and Y. Firstlywe generate 1000 samples of X and Yand then partition the X-axis and Y axis into 4 equally piecesso that we

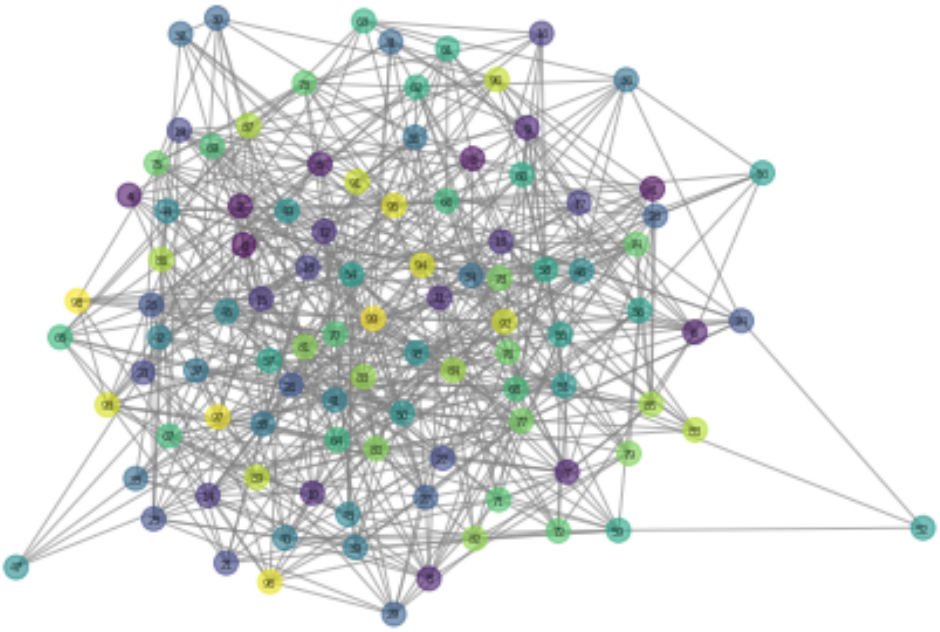
get 16 square. Secondlywe count the numbers of pointsX, Yin each sqaure. If X and Y are independentwe could calculate expected numbers of points in each square. Finallywe could use chi-squared test we used in project1 to verify their independency.

2.Experiments and Results

The test statistic has approximately a chi-squared distribution whose number of degrees of freedom are (4 - 1)(4 - 1)=9, and then we check the the p-value in Table of χ^2 values vs p-values. We could conclude that the probability that X and Y are independent is 0.95.

```
In [18]: def network(p):
             network_dict = {
             }
             for i in range(0, 100):
                 network_dict[i] = []
                 for j in range(i, 100):
                     u = random.uniform(0, 1)
                     if u <= p:
                         network_dict[i].append(j)
             return network_dict


         network_dict1 = network(0.03)
         network_dict2 = network(0.12)


         def draw_network(network_dict):
             G = nx.Graph()
             for i in range(0, 100):
                 G.add_node(i)
             for i in range(0, 100):
                 for j in network_dict[i]:
                     G.add_edge(i, j)
             nx.draw(G, node_size=100, node_color=range(0, 100), edge_color='gray',
                     font_size=5, with_labels=True, alpha=0.6)
             plt.show()


         draw_network(network_dict1)
         draw_network(network_dict2)
```





Network Fit Given n people in a social network. Suppose any given unordered pair of two people are connected at random and independently with probability p Generate and plot three network samples for each value of p=0.03 and p=0.12. Briefly discuss the structure of these sample graphs.

1.Theoretical Exploration or Analysis

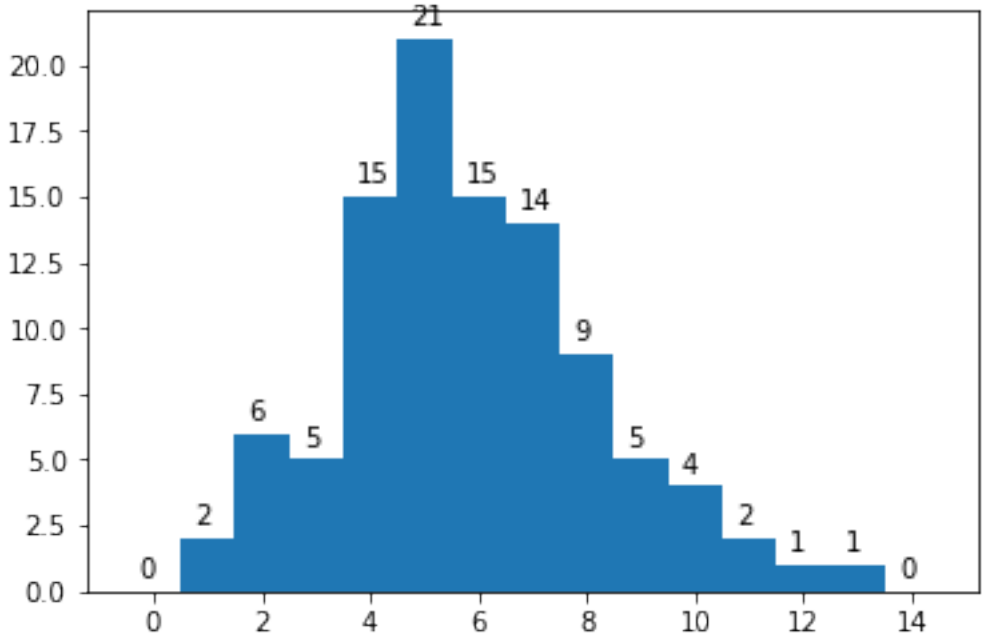
The event that any given unordered pair of two people are connected at random is independently with probability p. To be specific, we use Bernoulli to simulate the event. There are total of $\binom{n}{2}$ links. That is, we should repeat the event $\binom{n}{2}$ times and record each successful links. I save all the records in a dictionary. Intuitively the p is larger, the number successful links are larger.

2.Experiments and Results

From the figure, the first figure is less dense than the second onein other words the total of links are less than the second one. The result accord with our expectation. In conclusionthe p is larger, the number successful links are larger.

```
In [20]: def network1(p):
             network_dict = {
             }
             for i in range(0, 100):
                 network_dict[i] = []
             for i in range(0, 100):
                 for j in range(i, 100):
                     u = random.uniform(0, 1)
                     if u <= p:
                         network_dict[i].append(j)
                         network_dict[j].append(i)
             return network_dict


         network1_dict1 = network1(0.06)
         degree_list = []
         for value in network1_dict1.values():
             degree_list.append(len(value))

         p1, t = np.histogram(degree_list, bins=15, range=(0, 15))
         ind = np.arange(0, 15, 1)
         width = 1
         plt.bar(ind, p1, width)
         for i in range(0, 15):
             plt.text(i-0.25, p1[i]+0.5, str(p1[i]), fontsize=10)
         plt.show()


         estimated_degrees_list1 = []
         n, p = 100, 0.06
         for i in range(0, 15):
             estimated_degrees_list1.append(binom.pmf(i, n, p) * n)
```

```
estimated_degrees_list2 = []
for i in range(0, 15):
    estimated_degrees_list2.append(poisson.pmf(i, n*p) * n)


def chi_squared(f, f0, k):
    res = 0
    vol = int(len(f)/k)
    f_new = [sum(f[i*vol: vol*(i+1)]) for i in range(0, k)]
    f0_new = [sum(f0[i*vol: vol*(i+1)]) for i in range(0, k)]
    for i in range(0, k):
        res += np.power((f_new[i] - f0_new[i]), 2)/f0_new[i]
    return res


print('the probability that network degree distribution fits a binomial is',
      check_answer(chi_squared(p1, estimated_degrees_list1, 5), 5))
print('the probability that network degree distribution fits a Poisson is',
      check_answer(chi_squared(p1, estimated_degrees_list2, 5), 5))
```




```
the probability that network degree distribution fits a binomial is 0.95
the probability that network degree distribution fits a Poisson is 0.95
```

Generate a network with (n, p) = (100, 0.06). The number of connections for network node i is called the degree > of the node. Count the degree of each node in the network and plot the

histogram of degrees. Use goodness-of-fit tests to check how well the network degree distribution fits a binomial(n=100, p=0.06) or a Poisson(=np=6). 1.Theoretical Exploration or Analysis

The event that any given unordered pair of two people are connected at random is independently with probability p. To be specific, we use Bernoulli to simulate the event. There are total of $\binom{n}{2}$ links. That is, we should repeat the event $\binom{n}{2}$ times and record the degree of each node. So, the experiment satisfy the binomial modal. And given the fact that the binomial(n, p) probability generating function is close to the Poisson(λ = np) probability generating function in the assumption that n is large and p is small.

$$\binom{n}{k} p^k (1-p)^k \approx \frac{(np)^k e^{-np}}{k!} \tag{6}$$

2.Experiments and Results

I use chi-squared test to test goodness-of-fit. From the result, we know that the probability that network degree distribution fits a binomial is 0.95 and the probability that network degree distribution fits a Poisson is 0.95. In conclusion, it not only fit the binomial distribution but also Possion distributionwhich justifies our theory.