EE 511 Project #1: Sampling and Waiting
Name: Yang Liu
Student ID number:5847572002
Due: Sep 16

Summary

In project 1, I used python along with the mathematical modules including numpy and matploblib to simulate random variables distribution and plot histograms of frequency. I also compared the outcomes with the theoretical probability.

In the first problem—adding coins, the main point is to simulate repeated Bernoulli trials or binominal trials. There is a function called numpy.random.binomial, which can stimulate the distribution of repeated Bernoulli trials according to the input parameters, including trial times n, probability p and sampling times. Then I used the function called numpy.unique to count the frequency of different outcomes of repeated Bernoulli trials. In the question 3 in problem 1, It's a little different because we need to count the number of failed trials until the first successful trial, so I used a for loop and nest the binomial function in this loop to calculate the number of failed trials. At last, I used plot functions from matploblib module to draw the histograms, set up block colors, x y labels and so on.

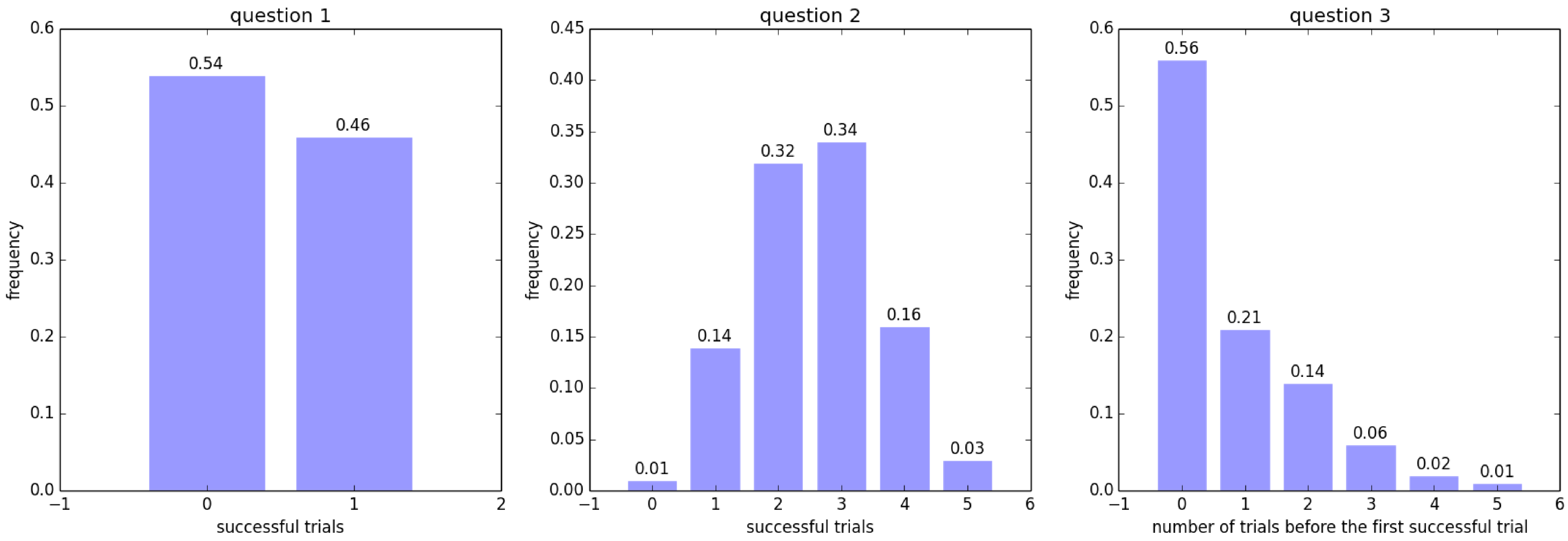
In the second problem, I also used binomial function in numpy module to simulate repeated Bernoulli trials. The only difference is that I used a parameter k to control the number of repeated Bernoulli trials. Moreover, I used normal distribution approximation to get the corresponding normal distribution of binomial distribution with different numbers of trial times, and then put them in the same diagram, which can help to check whether or not the repeated Bernoulli trials are reasonable, especially when k becomes bigger.

The last problem is to get the mean value interval with the confidence of 95% by bootstrap resampling the given data from NJGAS.dat. I used numpy.loadtxt funxtion to read NJGAS.dat and print them out . Then I used a bootstrap resample function to resample the given data 100000 times, every time I resampled, I keep the size of resampling sequences as the same to the original data sequence. After that I calculated and stored the mean values of the resampling data to get a mean value list composed of 100000 mean values. Finally I used mean confidence interval function to calculate the upper limit and lower limit of the 95% bootstrap confidence interval.

Discussion of the results

Problem #1

Question 1
According to the diagram the outcome of a single Bernoulli trial is approximately equal. This is reasonable because the given probability is fair, and each one of these trials in different samples is independent to each other.



Question 2

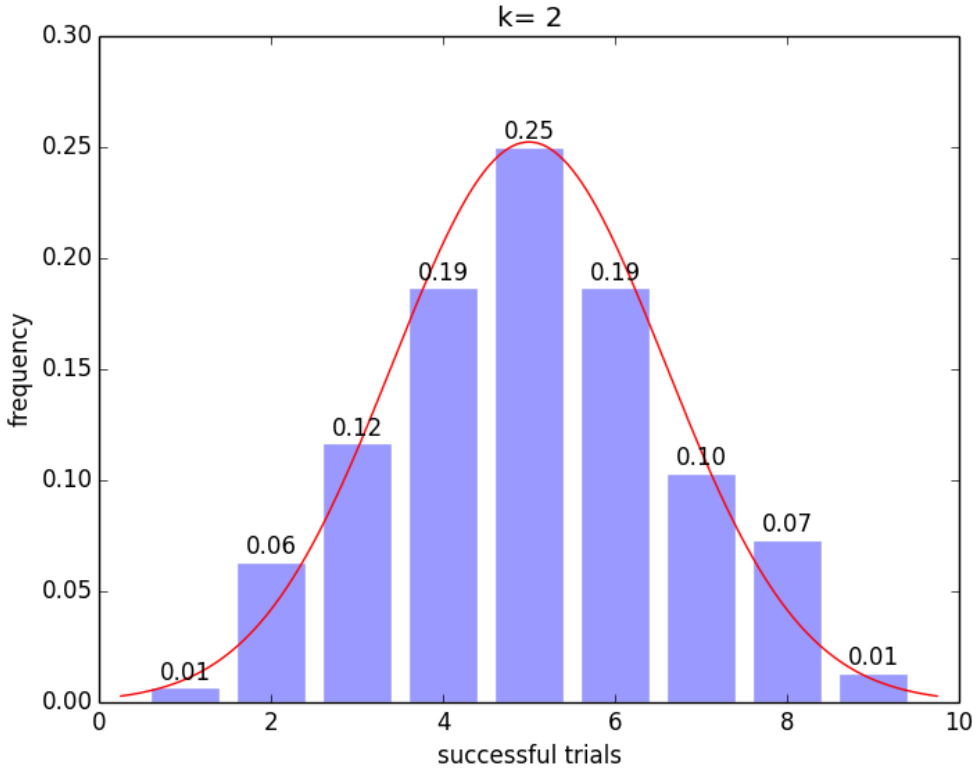
Theoretically, the outcome of question 2 obeys binomial distribution, the probability mass function should be $P(X = k) = \binom{n}{k}p^k(1-p)^{n-k}$, and n is 5, p is 0.5. So I calculated the theoretical probability of the outcomes should be 0.03, 0.15, 0.3125, 0.3125, 0.15, 0.03. If we check the diagram with the theoretical probability, they are very close, so I can consider that the routine is correct.

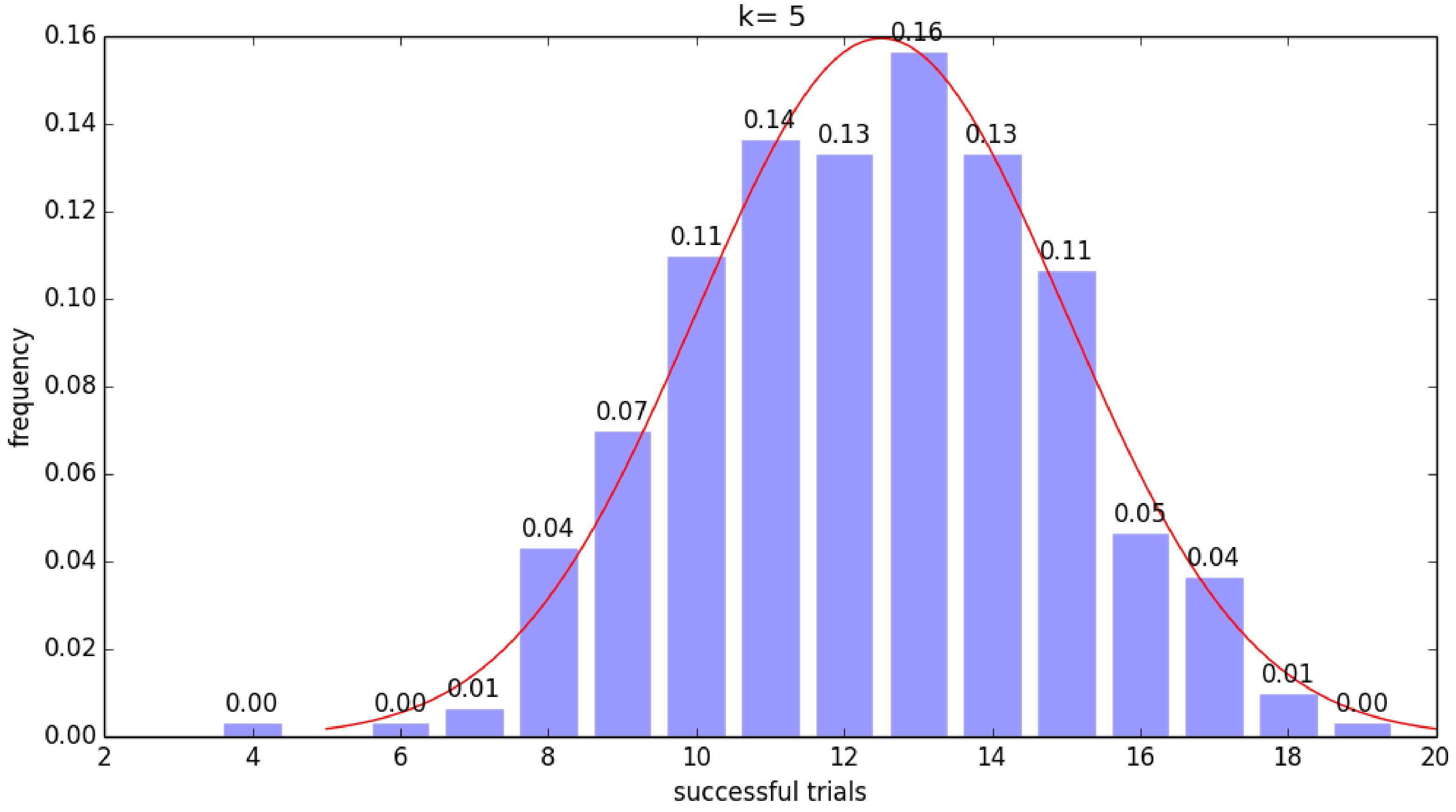
Question 3

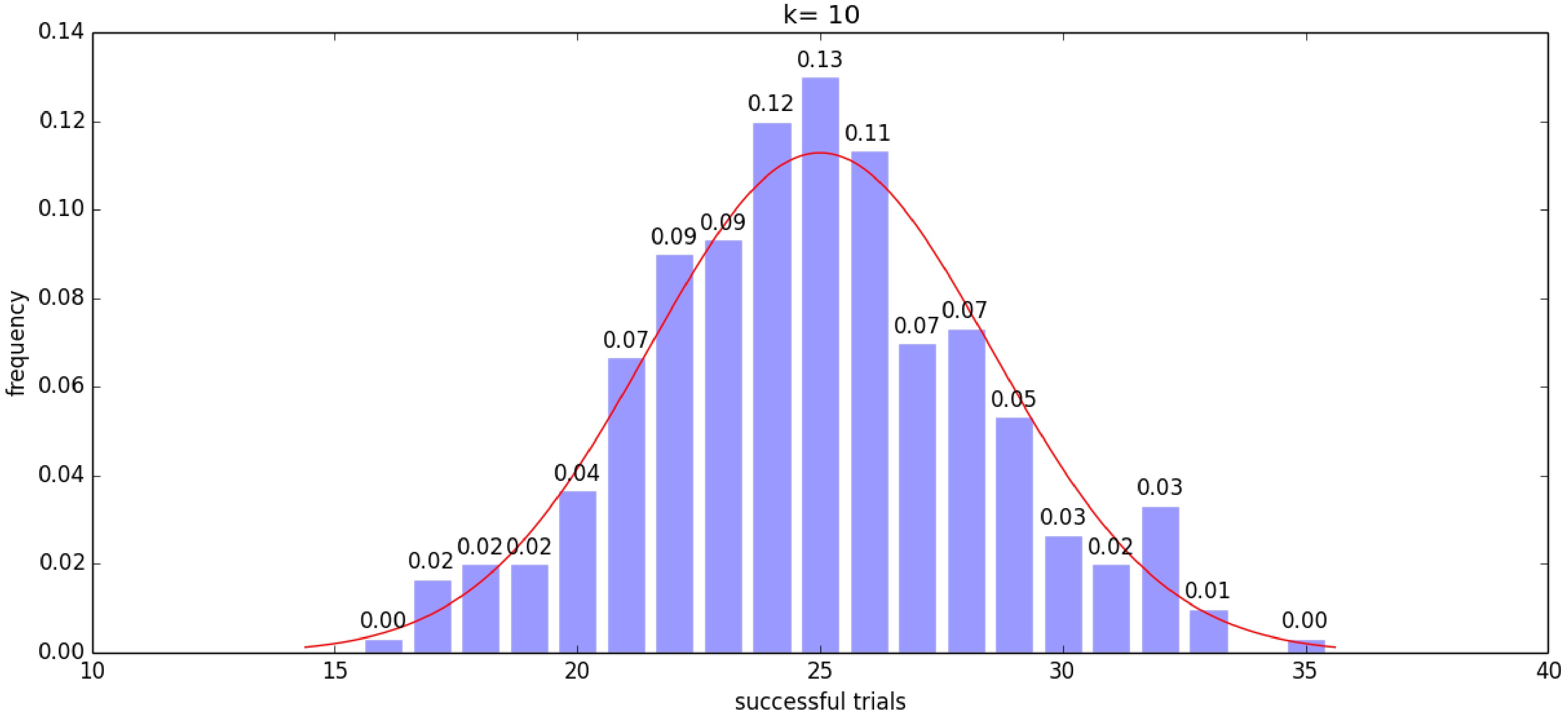
In terms of this question, if we draw a tree diagram, we could find that each time we conduct a trial, the probability to succeed is 0.5, and to fail is equal, so the theoretical probability to succeed in 1st trial, 2nd trial, 3rd trial and 4th trial is 0.5, 0.25, 0.125, 0.0625. For the 5th trial, the probability should have been 0.03125, but actually we do not have 6th trial here, so no matter whether or not the 5th trial is successful, I just assign the remaining probability 0.03125 to it. Here we can see the diagram is consistent with theoretical probability in general.

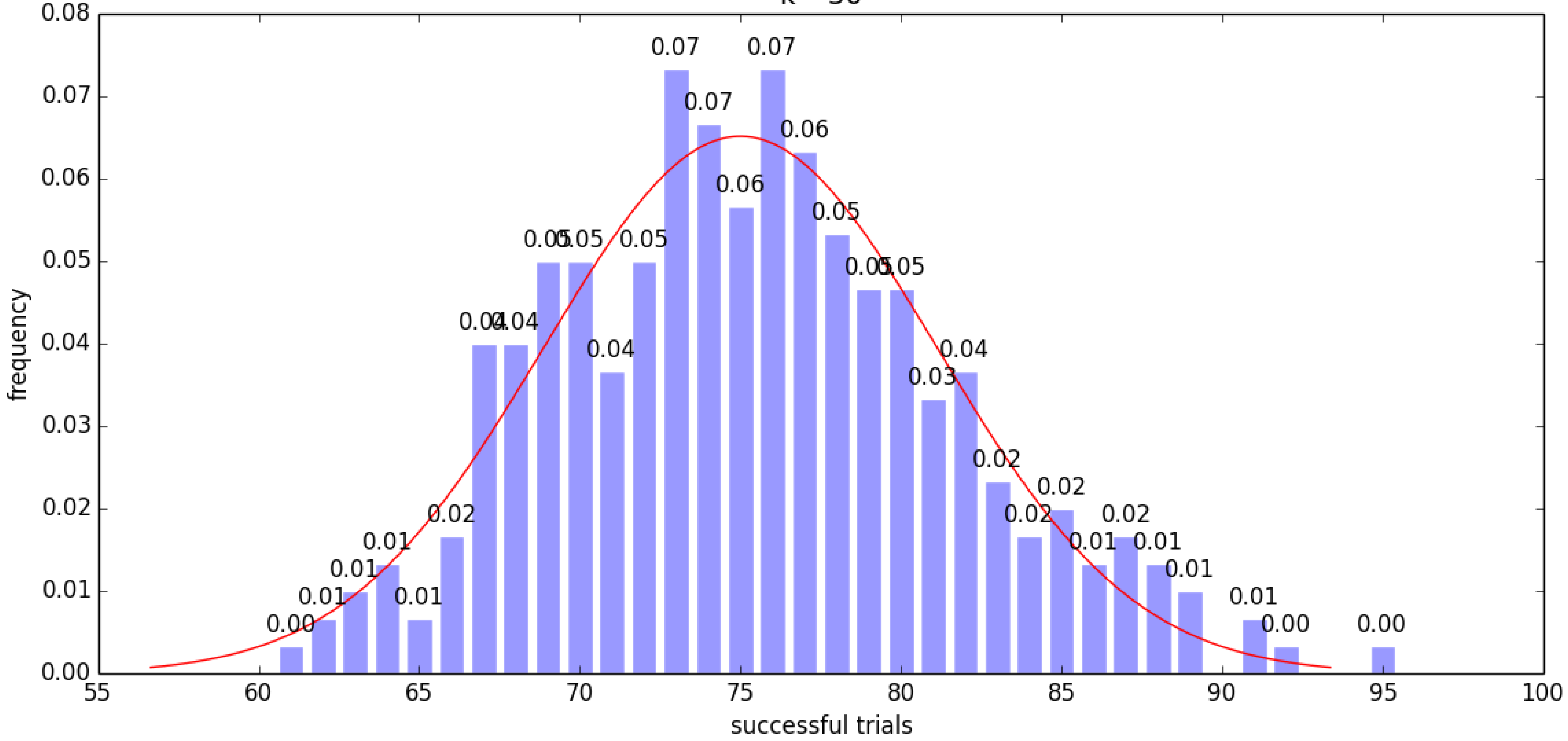
Problem #2

Problem #2 is similar to the 2nd question in problem 1, the only difference is that the total number of repeated Bernoulli trials is no longer 5--it changes to 5k, while k could be 2, 5, 10, 30, 50. What's more, we know that in binomial distribution, if n is large enough and p is not neat to 0 or 1, then the skew of the distribution is not too great, in this case a reasonable approximation to B (n, p) is given by the normal distribution N (np, np(1-p)). According to this theory, I generate the bell shaped curve of corresponding normal distribution N (2.5k, 1.25k) according to the theoretical binomial distribution B (5k, 0.5). I can justify the histograms by comparing it to the bell shaped curve because they are generally consistent.



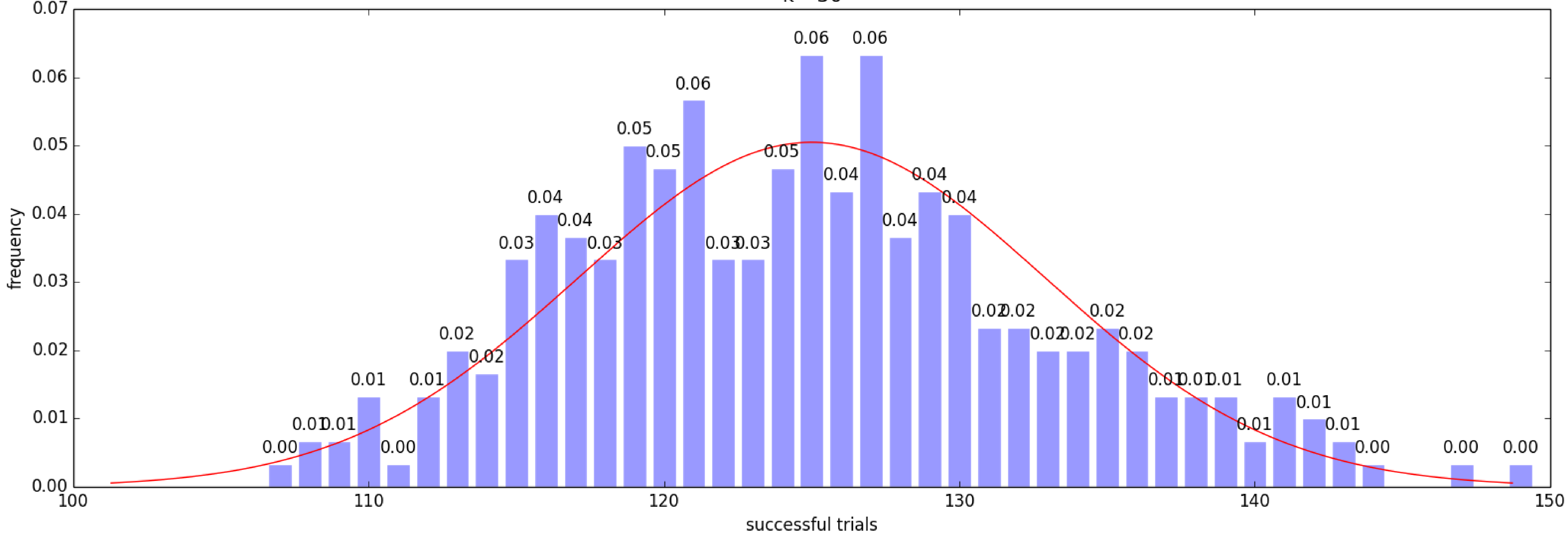






k= 30

frequency
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
successful trials
55
60
65
70
75
80
85
90
95
100
0.00
0.01
0.01
0.01
0.01
0.02
0.04
0.04
0.05
0.05
0.04
0.05
0.07
0.07
0.06
0.07
0.06
0.05
0.05
0.05
0.03
0.04
0.02
0.02
0.02
0.01
0.02
0.01
0.01
0.01
0.00
0.00

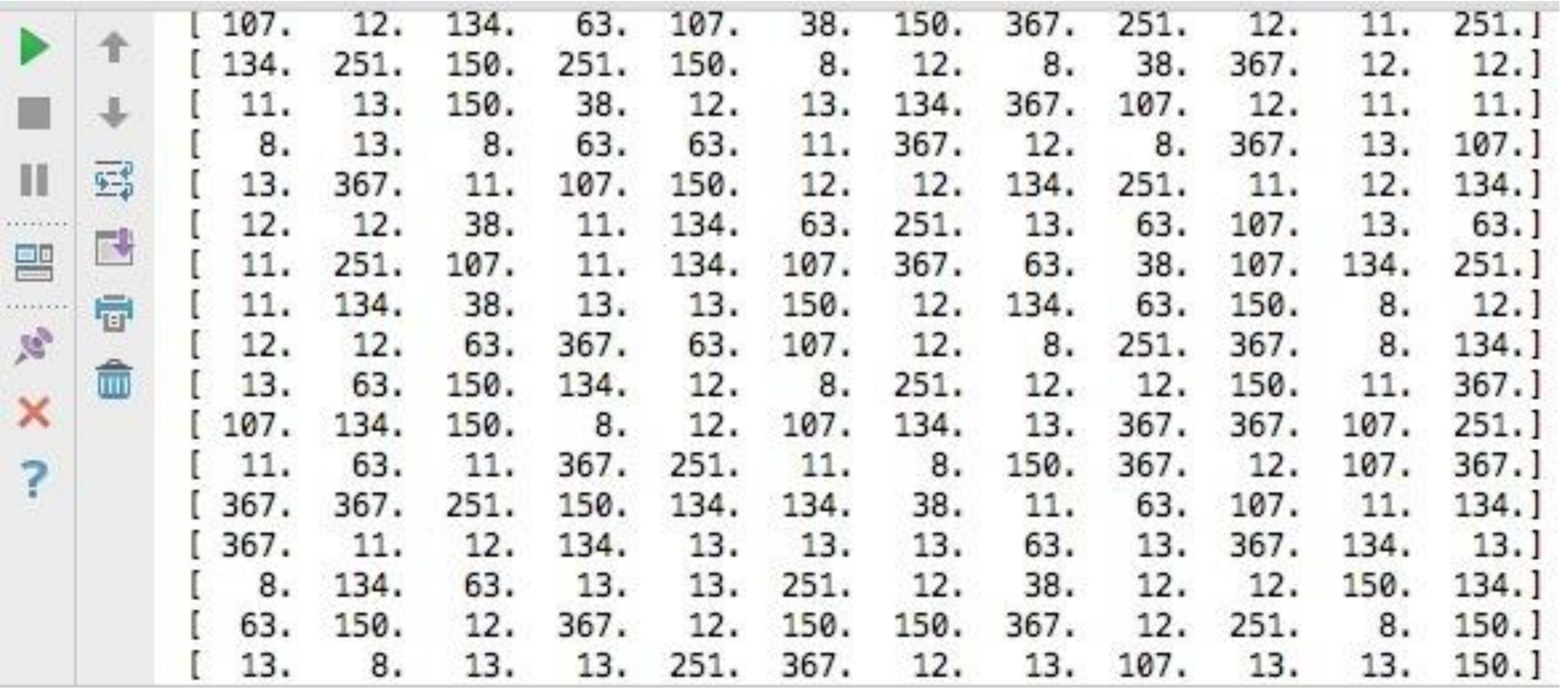


k= 50

frequency
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
successful trials
100
110
120
130
140
150
0.00
0.01
0.01
0.01
0.00
0.01
0.02
0.02
0.03
0.04
0.04
0.03
0.05
0.05
0.06
0.03
0.03
0.05
0.06
0.04
0.06
0.04
0.04
0.04
0.02
0.02
0.02
0.02
0.02
0.02
0.01
0.01
0.01
0.01
0.01
0.01
0.01
0.00
0.00
0.00

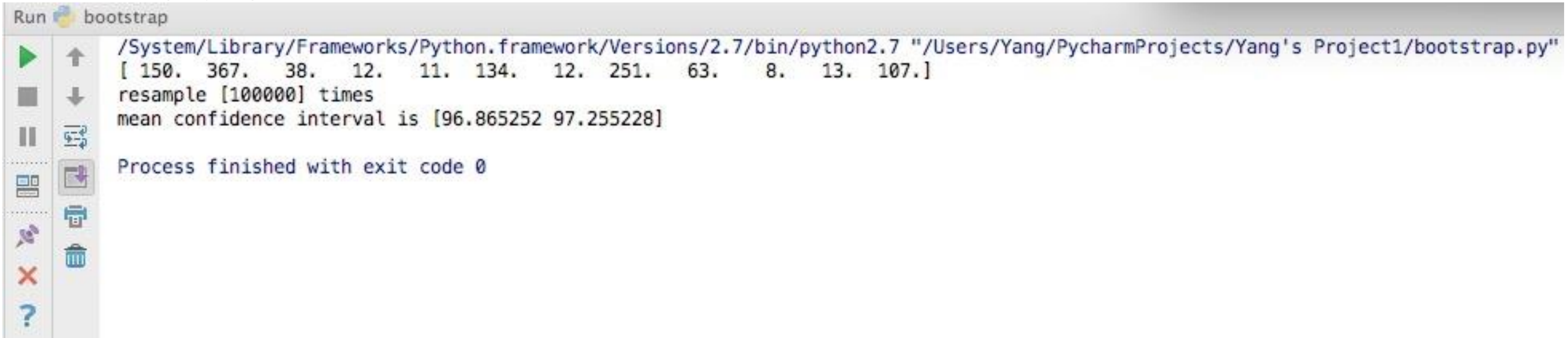
Problem #3

In problem 3, the main point is to bootstrap resample and calculate the mean confidence interval of the resampling sequences. I can set any number of bootstrap resampling times, here I used 100000. Actually in programming, I printed all of the resampling sequences for checking, the screenshot is as follows.



```
[ 107.   12.  134.   63.  107.   38.  150.  367.  251.   12.   11.  251.]
[ 134.  251.  150.  251.  150.    8.   12.    8.   38.  367.   12.   12.]
[  11.   13.  150.   38.   12.   13.  134.  367.  107.   12.   11.   11.]
[   8.   13.    8.   63.   63.   11.  367.   12.    8.  367.   13.  107.]
[  13.  367.   11.  107.  150.   12.   12.  134.  251.   11.   12.  134.]
[  12.   12.   38.   11.  134.   63.  251.   13.   63.  107.   13.   63.]
[  11.  251.  107.   11.  134.  107.  367.   63.   38.  107.  134.  251.]
[  11.  134.   38.   13.   13.  150.   12.  134.   63.  150.    8.   12.]
[  12.   12.   63.  367.   63.  107.   12.    8.  251.  367.    8.  134.]
[  13.   63.  150.  134.   12.    8.  251.   12.   12.  150.   11.  367.]
[ 107.  134.  150.    8.   12.  107.  134.   13.  367.  367.  107.  251.]
[  11.   63.   11.  367.  251.   11.    8.  150.  367.   12.  107.  367.]
[ 367.  367.  251.  150.  134.  134.   38.   11.   63.  107.   11.  134.]
[ 367.   11.   12.  134.   13.   13.   13.   63.   13.  367.  134.   13.]
[   8.  134.   63.   13.   13.  251.   12.   38.   12.   12.  150.  134.]
[  63.  150.   12.  367.   12.  150.  150.  367.   12.  251.    8.  150.]
[  13.    8.   13.   13.  251.  367.   12.   13.  107.   13.   13.  150.]
```

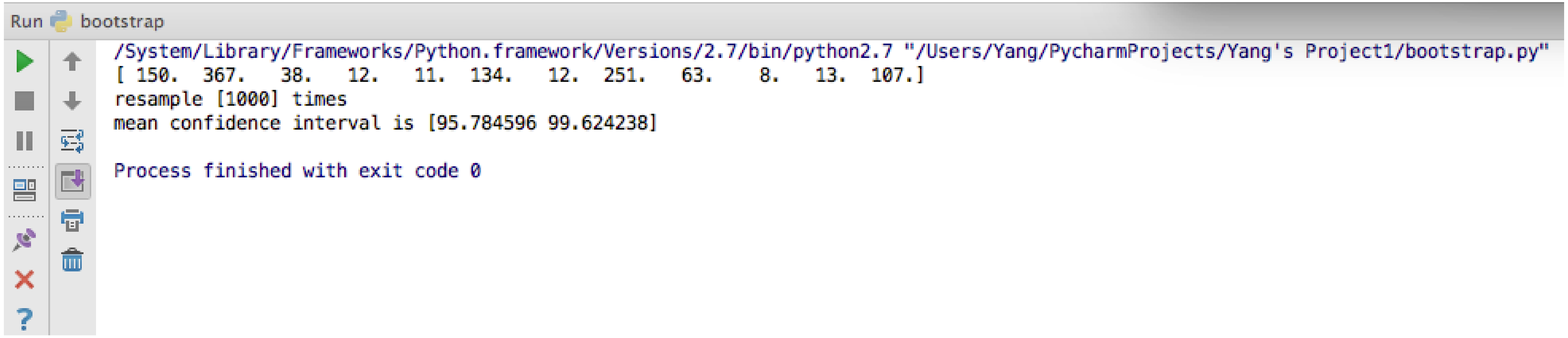
To make it more clear to see the outcome of the mean confidence interval, I finally commented the code responsible for showing the resampling sequences. About the confidence interval, the final outcome is always stable near 97 when I ran the codes many times. The upper limit is slightly smaller than 97 while the lower limit is slightly larger.



```
/System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 "/Users/Yang/PycharmProjects/Yang's Project1/bootstrap.py"
[ 150.  367.   38.   12.   11.  134.   12.  251.   63.    8.   13.  107.]
resample [100000] times
mean confidence interval is [96.865252 97.255228]

Process finished with exit code 0
```

If I changed the number of bootstrap resampling times, the interval may change. For example, if I set resampling number to be 1000, the outcome of mean confidence interval with the same confidence 95% shows as following.



```
/System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 "/Users/Yang/PycharmProjects/Yang's Project1/bootstrap.py"
[ 150.  367.   38.   12.   11.  134.   12.  251.   63.    8.   13.  107.]
resample [1000] times
mean confidence interval is [95.784596 99.624238]

Process finished with exit code 0
```

we can see that with larger number of resampling times, we can get narrower mean confidence intervals, but it also takes more time in simulation.

Source Code

Problem 1: addimg coins.py

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

plt.figure()

#question1-1
plt.subplot(1,3,1)
n11, p11 = 1, .5  # number of trials, probability of each trials
s11 = np.random.binomial(n11, p11, 100) # return number of successes each sample
print'q11'
unique,counts=np.unique(s11,return_counts=True) #numpy version no lower than 1.9
print unique
print counts/100.0
plt.bar(unique-0.4,+counts/100.0,facecolor='#9999ff',edgecolor='white')
for x, y in zip(unique, counts/100.0):
    plt.text(x , y + 0.002, '%.2f' % y, ha='center', va='bottom')
plt.ylim(0,0.6)
new_ticks=np.linspace(-1,2,4)
plt.xticks(new_ticks)
plt.xlabel('successful trials')
plt.ylabel('frequency')
plt.title('question 1')
print "\n"

#question1-2
plt.subplot(1,3,2)
n12, p12 = 5, .5  # number of trials, probability of each trials
s12 = np.random.binomial(n12, p12, 100) # return number of successes each sample
print'q12'
unique,counts=np.unique(s12,return_counts=True) #numpy version no lower than 1.9
print unique
print counts/100.0
plt.bar(unique-0.4,+counts/100.0,facecolor='#9999ff',edgecolor='white')
for x, y in zip(unique, counts/100.0):
    plt.text(x , y + 0.002, '%.2f' % y, ha='center', va='bottom')
plt.ylim(0,0.45)
plt.xlabel('successful trials')
plt.ylabel('frequency')
plt.title('question 2')
print "\n"

#question1-3
plt.subplot(1,3,3)
s13=[]
for i in range(0,100,1):
    flag=0
    fail=0
    n13, p13 = 1, .5  # number of trials, probability of each trials
    while (flag ==0 and fail<5):
        flag = np.random.binomial(n13, p13, 1) # return outcome each time
        if(flag==0):
            fail=fail+1
    s13.append(fail)
print'q13'
unique,counts=np.unique(s13,return_counts=True) #numpy version no lower than 1.9
print unique
print counts/100.0
plt.bar(unique-0.4,+counts/100.0,facecolor='#9999ff',edgecolor='white')
for x, y in zip(unique, counts/100.0):
    plt.text(x , y + 0.002, '%.2f' % y, ha='center', va='bottom')
plt.ylim(0,0.6)
plt.xlabel('number of trials before the first successful trial')
plt.ylabel('frequency')
plt.title('question 3')
plt.show()
print "\n"
```

problem #2 coin limits.py

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

kstring=(2,5,10,30,50)
for ii in range(0,5,1):
    n, p = 5, .5  # number of trials, probability of each trials
    k=kstring[ii]
    s2 = np.random.binomial(n*k, p, 300) # return number of successes each time
    print'q2%i' %(ii+1)
    unique,counts=np.unique(s2,return_counts=True) #numpy version no lower than 1.9
    print unique
    print counts/300.0
    plt.figure(ii)
    plt.bar(unique - 0.4, +counts / 300.0, facecolor='#9999ff', edgecolor='white')
    for x, y in zip(unique, counts / 300.0):
        plt.text(x, y + 0.001, '%.2f' % y, ha='center', va='bottom')
    plt.xlabel('successful trials')
    plt.ylabel('frequency')
    plt.title('k= %i' % (kstring[ii]))
    print "\n"

    mu = n*k*p
    variance = n*k*p*(1-p)
    sigma = math.sqrt(variance)
    x = np.linspace(mu-3*sigma,mu+3*sigma, 500)
    plt.plot(x,mlab.normpdf(x, mu, sigma),color='red',linewidth=1.0)

plt.show()
```

problem #3: bootstrap.py

```
import numpy as np
import scipy as sp
import scipy.stats

arr=np.loadtxt('NJGAS.dat')#read data
# http://docs.scipy.org/doc/numpy/reference/generated/numpy.loadtxt.html
print arr
#print "mean value of orginal data is [%f]" %arr.mean()

def bootstrap_resample(X, n=None):
    """ Bootstrap resample an array_like
    Parameters
    ----------
    X : array_like
      data to resample
    n : int, optional
      length of resampled array, equal to len(X) if n==None
    Results
    -------
    returns X_resamples
    """
    if n == None:
        n = len(X)

    resample_i = np.floor(np.random.rand(n) * len(X)).astype(int)
    X_resample = X[resample_i]
    return X_resample
mstr=[]#mean value string
for i in range(0,100000,1):#times of resample
    resam=bootstrap_resample(arr)
    #print resam
    #print resam.mean()
    mstr.insert(i, resam.mean())
#print mstr
print "resample [%i] times" %len(mstr)

def mean_confidence_interval(data, confidence):
    a = 1.0*np.array(data)
    n = len(a)
    m, se = np.mean(a), scipy.stats.sem(a)
    h = se * sp.stats.t._ppf((1+confidence)/2., n-1)
    return m, m-h, m+h
cen,low,high=mean_confidence_interval(mstr,0.95)
print "mean confidence interval is [%f %f]"  %(low,high)
```