Problem Set 1

Research method Problem Set 1 due Wed 10thth Oct, 17:00

LIU BOWEN (1004028)

For Problem Set 1, I use the following packages:

```
import matplotlib.pyplot as plt

import numpy as np

import scipy as sp

import scipy.stats as ss
```

Problem 1

Answer:

1) Each dice has six possible values which are equal probability and I firstly write a function calProb() to calculate the probability of occurrence of doubles for two dice.

As each throw is identical and independent, the outcome should meet Binomial Distribution. The expectation value should be N * probability

```
def calProb(n):

    if n <= 0:

        return -1;

        print "n is not allowed input 0 and minus!"

    return n * 1.0/n * 1.0/n;

def calExpec(n):

    prob = calProb(6);

    return n * prob;
```

2） Discussed above, the Variance should be n * probability * (1 probability)

```
    def calVar(n):

        prob = calProb(6);

        return n * prob * (1 - prob);
```

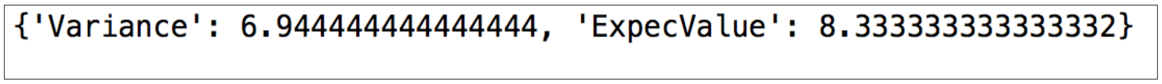
3) Given by the specific throw times, I use a dictionary retValue to store expectation value and variance. This key-value data structure is convenience for output.

```
def calValueForNthrows(n):

    retValue = {};

    retValue['ExpecValue'] = calExpec(n);

    retValue['Variance'] = calVar(n);

    return retValue;
```

Test case:

```
times_of_throw = 50;

print calValueForNthrows(times_of_throw)
```

The result for Problem 1-3 is shown in Figure 1.



```
{'Variance': 6.944444444444444, 'ExpecValue': 8.333333333333332}
```

Figure 1. Problem 1-3 Result

4)I define plotFigure() which is used to plot histogram. For test case, I input three value of n from 1k, 10k and 100k.

```
def plotFigure():

        times = [1000, 10000, 100000];

        for i in range(3):

                testTimes = times[i];

                n, p = testTimes, 1/6.0

                s = np.random.binomial(n, p, testTimes)

                plt.figure()

                plt.xlabel('Occurrence')

                plt.ylabel('Probability')

                plt.hist(s, color='red', edgecolor='black', density=True,
bins='auto')



plotFigure();
```

The histograms to demonstrate the occurrence of a double changes with n throws are shown from Figure-2 to Figure 4.

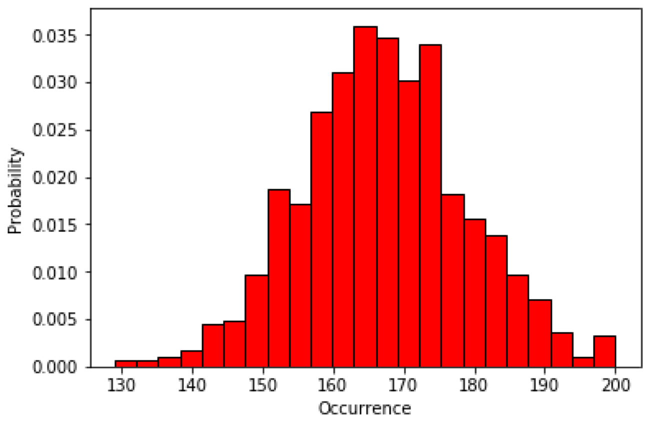
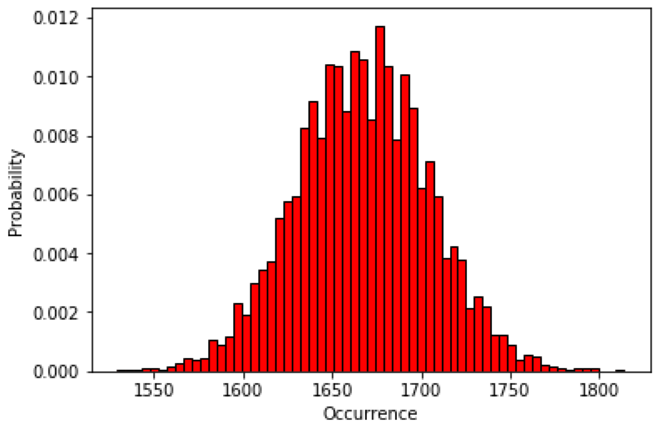


Figure 2 Problem 1-4 n=1000



s

Figure 3 Problem 1-4 n=10k

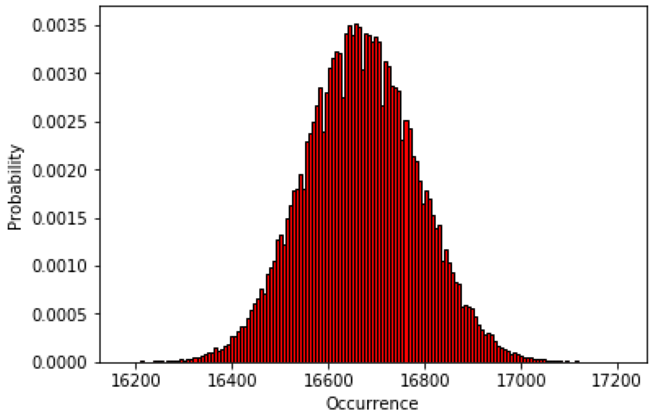


Figure 4 Problem 1-4 n=100k

Problem 2

Answer:

The mean number of patients arriving can meet Poisson Ditribution. The probability function is shown in Figure 5.

$$\boxed{p(x,\mu) \equiv \frac{\mu^x}{x!}e^{-\mu}}$$

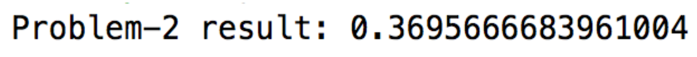
Figure 5 Poisson Probability function

For the Problem-2, I leverage np.power() np.exp() as well as np.math.factorial() to define Poisson function. As I need to calculate the value from 11:00-12:00, the mean value should be half(6.5/2=3.25)

```
def probFunc (x, mean):

    return np.power(mean, x) * 1.0 / np.exp(mean) / np.math.factorial(x);



mean_value = 6.5 / 2;

print 'Problem-2 result:', probFunc (0,mean_value)+ probFunc (1, mean_value)+ probFunc (2, mean_value)
```

Result:

When two or fewer patients arrived, the result should contain 0, 1, 2 circumstances and the result is shown Figure 6.



```
Problem-2 result: 0.3695666683961004
```

Figure 6 Result of Problem 2

Problem 3

Answer:

1)The uncertainty is defined as following:

```
def calUncert(n, p):

    var = n * p * (1-p);

    return var;
```

2)The standard deviation is defined as following:

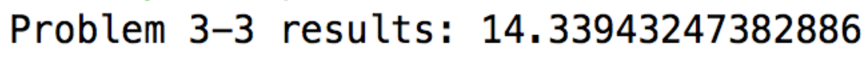
```
def standardVar(n, p):

    variance = calUncert(n, p);

    return np.sqrt(variance);
```

3)For the standard deviation on the last Wednesday, I need to re-calculate the total sample number and the probability.

```
n_sample = 752 + 283

p_sample = 752.0 / n_ sample

def calstandVarWed(n, p):

    return standardVar(n, p);



print calstandVarWed(n_sample, p_sample)
```

Result:

Standard variance is shown in Figure 7.



```
Problem 3-3 results: 14.33943247382886
```

Figure 7 Result of Problem 3-3

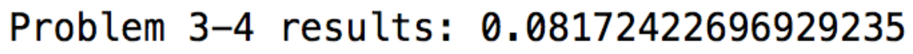
4) The mean for the last Wednesday should be 752 and for this sub-question the probability should be 0.75 as normal distribution is applicable. For this case, I use ss.norm().cdf() to calculate the accumulated probability and then should be doubled as there are two symmetrical part for the result.

```
def calFarProb(p, number, sample_mean):

    mean = p * number;
```

```
        variance = standardVar(number, p);
        return 2 * ss.norm(mean, variance).cdf(sample_mean);


n_sample = 752 + 283
p = 0.75
mean_wedn = 752
print calFarProb(p, n_sample, mean_wedn)
```

Result:

The probability of obtaining a result is shown in Figure 8.



```
Problem 3-4 results: 0.08172422696929235
```

Figure 8 Result of Problem 3-4

<u>Problem 4</u>

Answer:

For this question, I use probability plotting which is used to check whether experimental data is accurately modelled by a particular distribution. There are three procedures to do this:

1)Sorted the random sample data.

2)convert to normal distribution

3)plot the data to observe it.

In addition, R-square is a significant parameter to be used as a measure of goodness of fit. The more R-square is close to 1, the more precise the measurement.

```
def plotR2():
    r_square = [];
    r_number = [];
    for i in range(1, 81):
        sample = ss.poisson(i).rvs(1000)
        def destinationFunc(x, m, c):
            return m * x + c;
        sorted_sample = np.sort(sample)
        std_norm_dis =[]
        for index in range(len(sorted_sample)):
            score=((index + 1) - 0.5) / len(sorted_sample)
            std_norm_dis.append(ss.norm(0,1).ppf(score))
        [m, c] = sp.optimize.curve_fit(destinationFunc, std_norm_dis,
sorted_sample)[0]
        dest_fit = []
        for z in std_norm_dis:
            dest_fit.append(destinationFunc(z,m,c));
        def calR2(sorted_sample, destination_linear):
            counter = 0
            SStot =[]
            SSres =[]
```

```
                for _sample in sorted_sample:

                    SStot.append((_sample - np.mean(sorted_sample))
** 2)

                    SSres.append((_sample -
destination_linear[counter]) ** 2)

                    counter = counter + 1

                return (1-(sum(SSres)/sum(SStot)))

            r_square.append(calR2(sorted_sample, dest_fit));

            r_number.append(np.mean(sample));


        plt.figure()

        plt.ylabel('R-square')

        plt.xlabel('Mean')

        plt.plot(r_number, r_square)


plotR2();
```

Firstly, I generate the number of 1000 Poisson random variables from the mean of 1 to the mean of 80. Then, define the destination plot function. Before converting to the normal distribution, I need to sort the sample. Next, I use the scipy.stats.optimize.curve_fit to generate the slope and intercept of destination function. At last, calculate the value of linear line and I use the r_square and r_number array to store each R-square value and the mean of each sample.
The R-square change with the mean from 1 to 80 is shown in Figure 9.

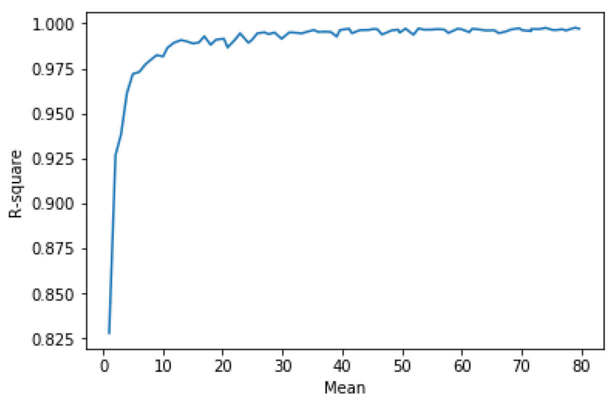


Figure 9 R-square value – Mean

From the Figure 9 shown, the R-square is close to 1 when the mean is larger than around 20(For my test case, 1000 random variables each time). And I use the calR2() function to calculate the R-square.

In addition, I also use the probability plotting to observe.

```
sample = ss.poisson(50).rvs(1000)
def destinationFunc(x, m, c):
    return m * x + c;
def calR2(sorted_sample, destination_linear): #compute R-squared
    counter = 0
    SStot =[]
    SSres =[]
    for _sample in sorted_sample:
        SStot.append((_sample - np.mean(sorted_sample)) ** 2)
        SSres.append((_sample - destination_linear[counter]) ** 2)
        counter = counter + 1
        return (1-(sum(SSres)/sum(SStot)))
sorted_sample = np.sort(sample)
std_norm_dis =[]
for i in range(len(sorted_sample)):
    score=((i + 1) - 0.5) / len(sorted_sample)
    std_norm_dis.append(ss.norm(0,1).ppf(score))
[m, c] = sp.optimize.curve_fit(destinationFunc, std_norm_dis,
sorted_sample)[0]
dest_fit = []
for z in std_norm_dis:
    dest_fit.append(destinationFunc(z,m,c));
plt.figure()
plt.plot(std_norm_dis, sorted_sample, color='red', marker='+',
markerfacecolor='None')
plt.plot(std_norm_dis, dest_fit, color='black')
plt.xlabel('z-statistic mean=30')
plt.ylabel('Value')
```

Above-mentioned code is about probability plotting, I plot the three figures which mean equals 10, 30, 50.

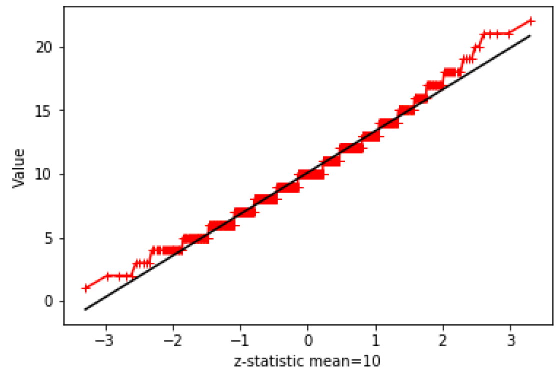


Figure 10 Mean=10, size=1000

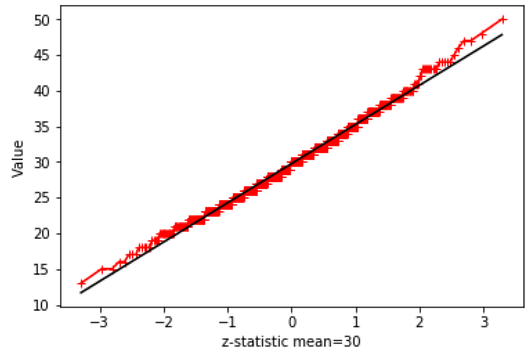


Figure 11 Mean=30, size=1000

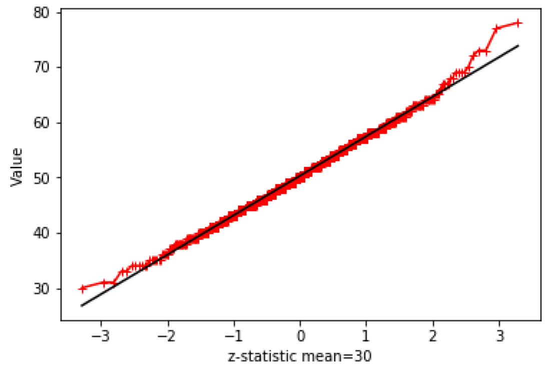


Figure 12 Mean=50, size=1000

From the figure 10-12, when the mean equals 50 the data meet the linear line more.