

Project #2 – Samples and statistics

EE 511: Fall 2019

Qiong Wang Id: 5906740674

Tool: PyCharm

1. Simulate uniformly distribution

Requirement: Simulate sampling uniformly on the interval $[-3, 2]$, generate a histogram of the outcomes. Compute the sample mean and sample variance and compare to the theoretical values. ---- $Q1(a,b)$

Analysis: For a uniform distribution $U \sim (a, b)$, mean = $(a + b)/2$, variance = $(b - a)^2/12$. So, in this question, I simulate $U \sim (-3, 2)$, by using these two functions, I get that: mean = -0.500, var = 2.083. In order to simulate $U \sim (-3, 2)$ in PyCharm, I use the function `<np.random.uniform(-3, 2)>` to generate a random real number between -3 and 2. To get a relative good result, I decide to sample 5000 times, and save the data to a list, and plot a histogram and calculate the mean and variance. Moreover, uniform distribution is a kind of continues distribution, but for computer, I can only get some discrete value. So, when I plot the histogram, I decide to change the width of bins. Width = [1, 0.5, 0.2, 0.1], wider the bins, smoother the histogram.

Code:

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

number_samples = 5000
outcomes = []
width = [1, 0.5, 0.2, 0.1]
for i in range(number_samples):
    outcomes.append(np.random.uniform(-3, 2))
plt.figure(1)
plt.title('histogram of outcome')
for i in range(4):
```

```

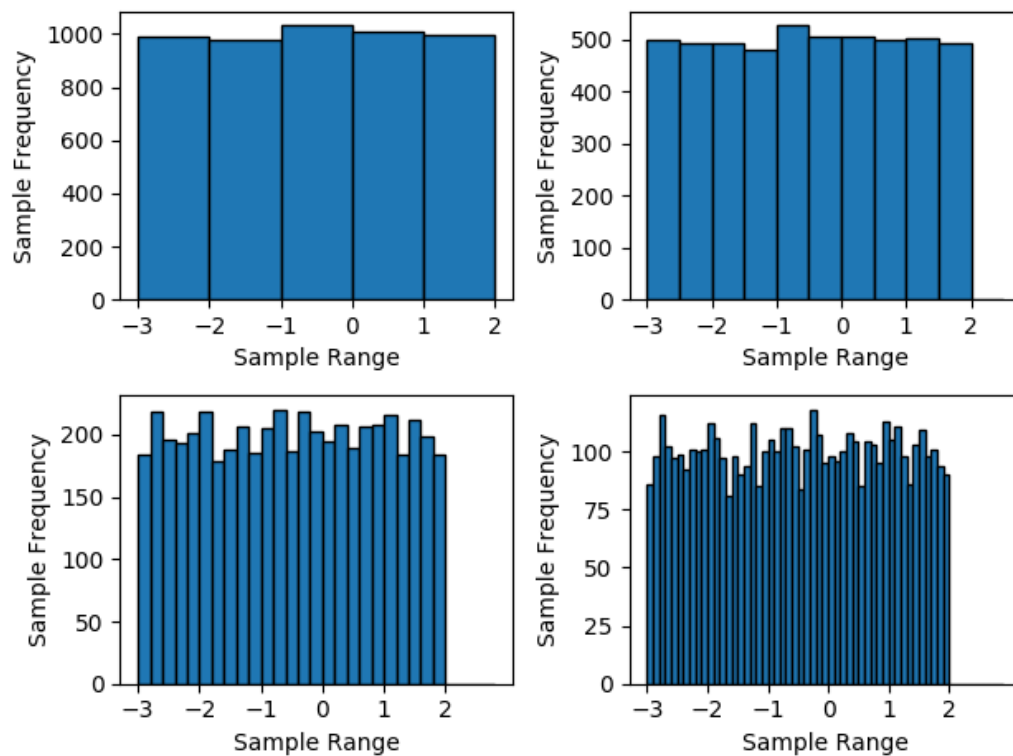
plt.subplot(2, 2, i+1)
plt.hist(outcomes, bins=np.arange(-3,3,width[i]),
edgecolor='black')
plt.xticks(range(-3,3))
plt.xlabel('Sample Range')
plt.ylabel('Sample Frequency')
plt.show()

print('Sample Mean:', np.mean(outcomes))
print('Sample Variance:', np.var(outcomes, ddof=1))

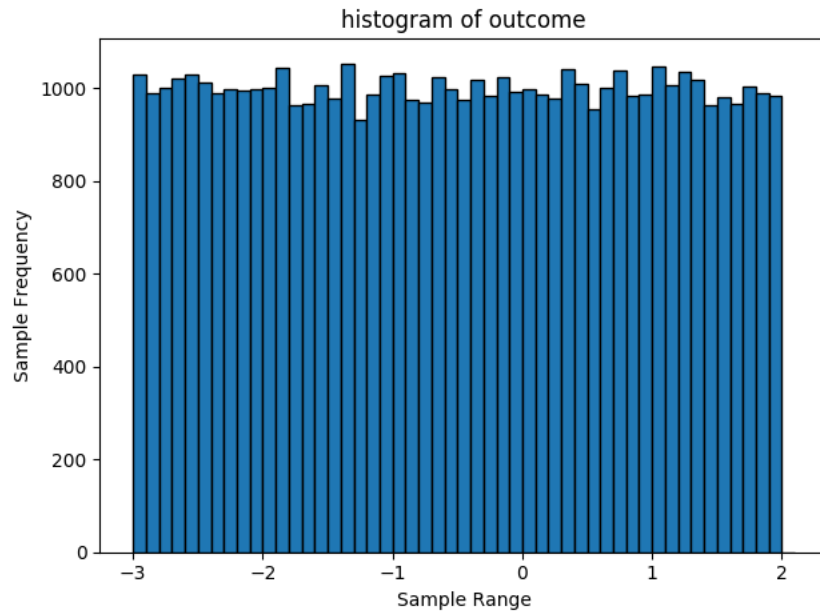
```

Result:

the result is: Sample Mean: -0.4949; Sample Variance: 2.0653. Compare to the theoretical value, the deviation of mean is 1.03%, the deviation of variance is 0.87%.



From the histogram, I find that when width of bin = 0.1, the histogram become fluctuated. I think the reason is I don't have enough data, so I sample this distribution 50000 times, and plot the histogram with bin=0.1.



When I sample 50000 times, the histogram become smoother and the result become much better: sample Mean = -0.504, Sample Variance: 2.085. I think more sample times could eliminate the system noise and experiments deviation. So, I change the sample times so get different results:

sample times	mean	var
10.00	-0.88	2.26
100.00	-0.45	2.33
500.00	-0.55	2.21
1000.00	-0.52	2.07
5000.00	-0.49	2.06
10000.00	-0.49	2.11
50000.00	-0.51	2.08
100000.00	-0.50	2.09

Then I fixed the sample times = 5000 and repeat, the experiment 6 times:

No.	mean	var
1	-0.50	2.05
2	-0.48	2.06
3	-0.47	2.13
4	-0.50	2.06
5	-0.47	2.06
6	-0.50	2.09

So, when I repeat the experiment, I will compute a different sample mean or sample variance, however, even though I get different outcomes, the result is very close to theoretical value.

2. Bootstrap uniformly distribution ---- Q1(c)

Requirement: Compute the bootstrap confidence interval for the sample mean and sample standard deviation.

Analysis: Theoretically, mean = 0.5, std = 1.443. In this question I set 95% bootstrap confidence interval, each time I simulate 50 sampling, and resample 10000 times, calculate each group of sample's mean and std. Store the data in list and sort these 2 list in non-decreasing order, print out the 250th and 9750th element in the list.

Code:

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

std = []
mean = []
samples = []
number_samples = 50
number_resample = 10000

for i in range(number_samples):
    samples.append(np.random.uniform(-3, 2))

for i in range(number_resample):
    resample = []
```

```
for j in range(number_samples):
    resample.append(random.sample(samples, 1))
std.append(np.std(resample, ddof=1))
mean.append(np.mean(resample))

plt.figure(1)
plt.subplot(2, 1, 1)
plt.title('mean')
plt.hist(mean, bins=np.arange(min(mean)-0.1, max(mean)+0.2, 0.05),
         edgecolor='black')

plt.subplot(2, 1, 2)
plt.title('std')
plt.hist(std, bins=np.arange(min(std)-0.1, max(std)+0.2, 0.02),
         edgecolor='black')

plt.show()

mean.sort()
std.sort()
print('2.5% mean:', mean[249])
print('97.5% mean:', mean[9749])
print('2.5% std:', std[249])
print('97.5% std:', std[9749])
```

Result:

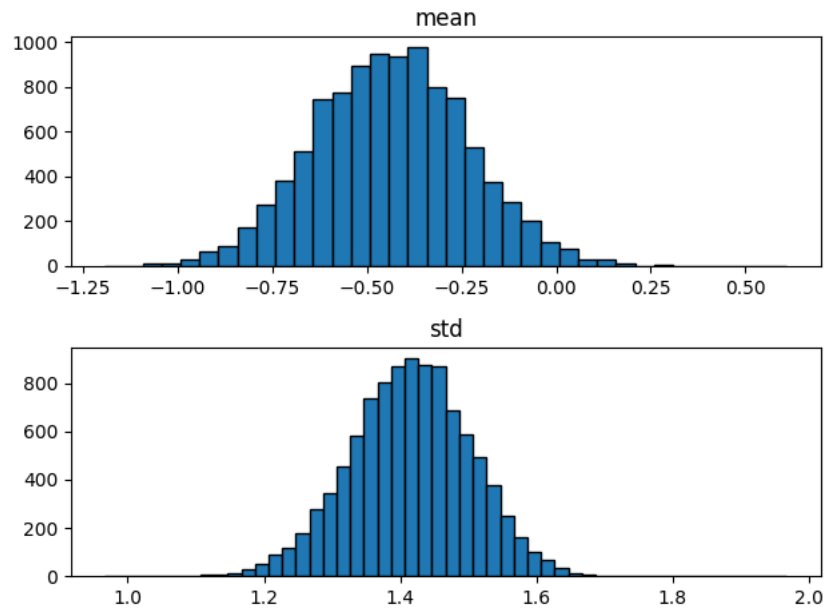
the result is:

2.5% mean: -0.8275314415637202

97.5% mean: -0.041027978954332696

2.5% std: 1.2377743059553485

97.5% std: 1.5833498836041249



3. Produce a sequence X by drawing samples from a standard uniform random variable. ---- Q2(a)

Requirement: Compute $Cov[X_k, X_{k-1}]$. Are X_k and X_{k-1} uncorrelated? What can you conclude about the independence of X_k and X_{k-1} ?

Analysis: In this question, I generate a list of data which is obey $U \sim (0, 1)$. Each pair of data is i.i.d, so, $Cov[X_k, X_{k-1}] = 0$, and for standard uniform random variable $Cov[X_k, X_k] = \text{var}(X) = 0.0833$. When I use PyCharm to simulate this question, I change the length of dataset from 1000 to 1000000 and calculate the covariance of X_k and X_{k-1} . With the increasing number of data, the variance will converge to 0.083, and covariance will converge to 0.

Code:

```
import random
import numpy as np
```

```
X = []
X_k = []
X_k1 = []
```

```
length_k = 1000

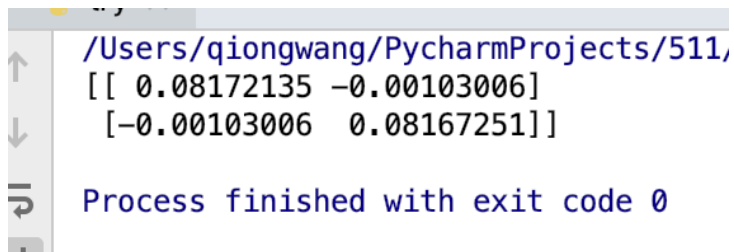
for i in range(length_k+1):
    X.append(random.uniform(0,1))

X_k = X[:length_k]
X_k1 = X[1:]

print(np.cov([X_k, X_k1]))
```

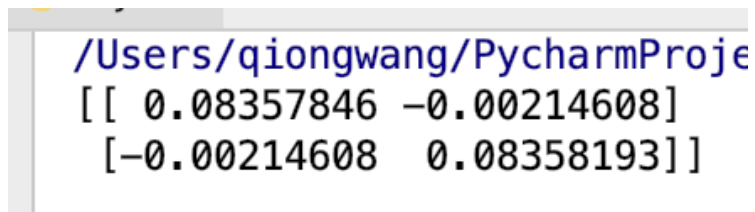
Result:

length = 1000, cov:



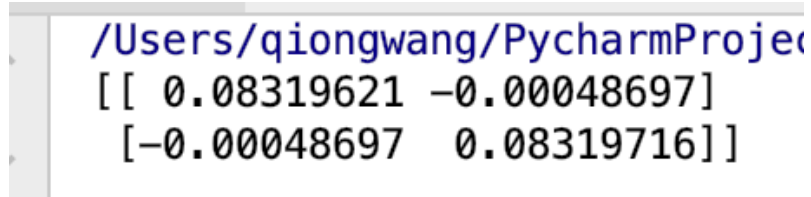
```
/Users/qiongwang/PycharmProjects/511,
[[ 0.08172135 -0.00103006]
 [-0.00103006  0.08167251]]
Process finished with exit code 0
```

length = 10000, cov =



```
/Users/qiongwang/PycharmProje
[[ 0.08357846 -0.00214608]
 [-0.00214608  0.08358193]]
```

length = 100000, cov =



```
/Users/qiongwang/PycharmProje
[[ 0.08319621 -0.00048697]
 [-0.00048697  0.08319716]]
```

from the result, X_k , X_{k-1} are uncorrelated.

4. Compute a new sequence Y. ---- Q2(b)

Requirement: $y[k] = x[k] - 2x[k-1] + 0.5x[k-2] - x[k-3]$, Compute $Cov [X_k, Y_k]$. Are $X[k]$ and $Y[k]$ uncorrelated?

Analysis: Because the symbol before $x[k]$ is positive, and $y[k]$ is uncorrelated with $x[k-1]$, $x[k-2]$, $x[k-3]$, so, the Cov between $y[k]$ and $x[k]$ should be positive. When I use PyCharm to simulate this question, I change the length of dataset from 1000 to 1000000 and calculate the covariance of X_k and Y_k .

Code:

```
import random
import numpy as np

X = [0, 0, 0]
X_k = []
Y_k = []

length_k = 1000

for i in range(length_k):
    X.append(random.uniform(0,1))

for i in range(3,length_k+3):
    X_k.append(X[i])
    Y_k.append(X[i] - 2 * X[i-1] + 0.5 * X[i-2] - X[i-3])

print(np.cov([X_k, Y_k]))
```


Result:

length = 1000, cov:

```
/Users/qiongwang/PycharmProjects/51
[[0.08381265 0.08798751]
 [0.08798751 0.55339879]]

Process finished with exit code 0
```

length = 10000, cov:

```
/Users/qiongwang/PycharmProjects/51
[[0.08398646 0.08233139]
 [0.08233139 0.51964494]]

Process finished with exit code 0
```

length = 100000, cov:

```
/Users/qiongwang/PycharmProjects/51
[[0.08330796 0.08223342]
 [0.08223342 0.51680222]]

Process finished with exit code 0
```

length = 1000000, cov:

```
/Users/qiongwang/PycharmProjects/51
[[0.08319496 0.0829048 ]
 [0.0829048  0.51912725]]

Process finished with exit code 0
```

From the result, I find that with the increasing number of data length, the output will not change too much.

5. Let $M = 10$. Simulate (uniform) sampling with replacement from the outcomes 0, 1, 2, 3, ..., $M-1$. ---- Q3(a,b)

Requirement: Let $M = 10$. Simulate (uniform) sampling with replacement from the outcomes 0, 1, 2, 3, ..., $M-1$. Generate a histogram of the outcomes. Perform a statistical goodness-of-fit test to conclude at the 95% confidence level if your data fits samples from a discrete uniform distribution 0, 1, 2, ..., 9.

Analysis: In this problem, I simulate sampling 1000 times and I set bin = [0,1,2,3,4,5,6,7,8,9], so, for each bin, expected value is 100, degree freedom = 9, for 95% confidence level, the threshold is 16.91898, and then I calculate chi-square, if the value less than threshold, it's a good fit.

Code:

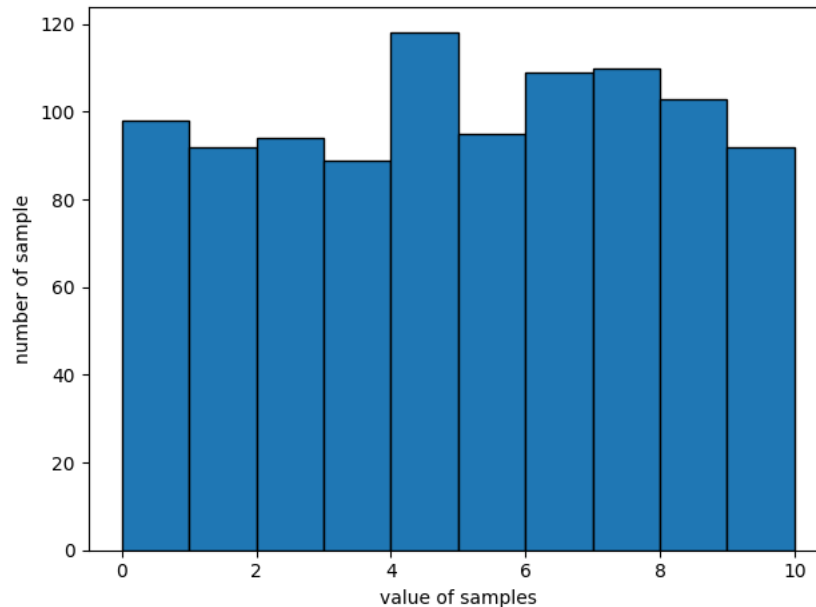
```
import random
import matplotlib.pyplot as plt

for times in range(20):
    X2 = 0
    num = 1000
    degree = 9
    sample = []
    for i in range(num):
        sample.extend(random.sample(range(10), 1))
    for i in range(degree+1):
        X2 += ((sample.count(i) - num/(degree + 1)) ** 2) /
    (num/(degree + 1))
    print('In the ' + str(times) + ' times calculation, chi-Square
    =', X2)

plt.figure()
plt.hist(sample, bins=range(11), edgecolor='black', )
plt.xlabel('value of samples')
plt.ylabel('number of sample')
plt.show()
```

Result:

I repeat the simulation 20 times, and I print all of the value of chi-square. All of the value is smaller than the threshold, so I think it's a good fit.



```
/Users/qiongwang/PycharmProjects/511/project2/venv/bin/python /l
```

```
In the 0 times calculation, chi-Square = 5.199999999999999
In the 1 times calculation, chi-Square = 9.600000000000001
In the 2 times calculation, chi-Square = 6.200000000000001
In the 3 times calculation, chi-Square = 12.600000000000001
In the 4 times calculation, chi-Square = 10.8
In the 5 times calculation, chi-Square = 11.4
In the 6 times calculation, chi-Square = 6.000000000000001
In the 7 times calculation, chi-Square = 10.0
In the 8 times calculation, chi-Square = 10.4
In the 9 times calculation, chi-Square = 10.4
In the 10 times calculation, chi-Square = 3.4000000000000004
In the 11 times calculation, chi-Square = 16.8
In the 12 times calculation, chi-Square = 6.0
In the 13 times calculation, chi-Square = 4.8
In the 14 times calculation, chi-Square = 8.4
In the 15 times calculation, chi-Square = 10.600000000000001
In the 16 times calculation, chi-Square = 3.6
In the 17 times calculation, chi-Square = 12.200000000000001
In the 18 times calculation, chi-Square = 6.6
In the 19 times calculation, chi-Square = 1.6
```

```
Process finished with exit code 0
```

6. Let $M = 10$. Simulate (uniform) sampling with replacement from the outcomes $0, 1, 2, 3, \dots, M-1$. ---- Q3(c)

Requirement: Repeat (b) to see if your data (the same data from b) instead fit an alternate uniform distribution $1, 2, 3, \dots, 10$.

Analysis: In this problem, I simulate sampling 1000 times and I set bin = [$\leq 1, 2, 3, 4, 5, 6, 7, 8, 9, 10$]. The reason I put 0 and 1 in the first bin is that the expected value of $M=0$ is 0, so, I put 0 and 1 to the same bin. for each bin, expected value is 100, degree freedom = 9, for 95% confidence level, the threshold is 16.91898, and then I calculate chi-square, if the value less than threshold, it's a good fit.

Code:

```
import random

for times in range(20):
    X2 = 0
    num = 1000
    degree = 9
    sample = []
    for i in range(num):
        sample.extend(random.sample(range(10), 1))
    X2 += ((sample.count(0)+sample.count(1) - num / (degree + 1)) **
2) / (num / (degree + 1))
    for i in range(2,degree+2):
        X2 += ((sample.count(i) - num/(degree + 1)) ** 2) /
(num/(degree + 1))
    print('In the ' + str(times) + ' times calculation, chi-Square
=', X2)
```

Result:

All of the chi-square value is larger than threshold, so, it's not a good fit.

```
/Users/qiongwang/PycharmProjects/511/project2/venv/bin/python  
In the 0 times calculation, chi-Square = 188.23999999999998  
In the 1 times calculation, chi-Square = 249.88000000000002  
In the 2 times calculation, chi-Square = 263.29999999999995  
In the 3 times calculation, chi-Square = 176.86  
In the 4 times calculation, chi-Square = 240.02000000000004  
In the 5 times calculation, chi-Square = 167.94  
In the 6 times calculation, chi-Square = 193.77999999999997  
In the 7 times calculation, chi-Square = 246.68000000000004  
In the 8 times calculation, chi-Square = 270.74  
In the 9 times calculation, chi-Square = 212.76  
In the 10 times calculation, chi-Square = 213.74  
In the 11 times calculation, chi-Square = 202.08  
In the 12 times calculation, chi-Square = 337.94  
In the 13 times calculation, chi-Square = 206.74  
In the 14 times calculation, chi-Square = 216.0  
In the 15 times calculation, chi-Square = 217.24  
In the 16 times calculation, chi-Square = 237.23999999999998  
In the 17 times calculation, chi-Square = 217.45999999999998  
In the 18 times calculation, chi-Square = 202.0  
In the 19 times calculation, chi-Square = 168.46
```

```
Process finished with exit code 0
```