EE511 Project2

Name: Yijing Jiang

Email: yijingji@usc.edu

USC ID: 5761477714

Question 1. (a)

Answer:

- Sample on the interval [-3,2] uniformly will give a random floating point number. Record all these outcome samples to plot a histogram. The histogram will show the counting number of samples on each small unit intervals of [-3,2], which can be seen as a simulation of U[-3,2].
- To find how many samples can simulate U[-3,2] well, try multiple sampling times on the interval[-3,2]: 10, 100, 1000, and 10,000 times respectively.

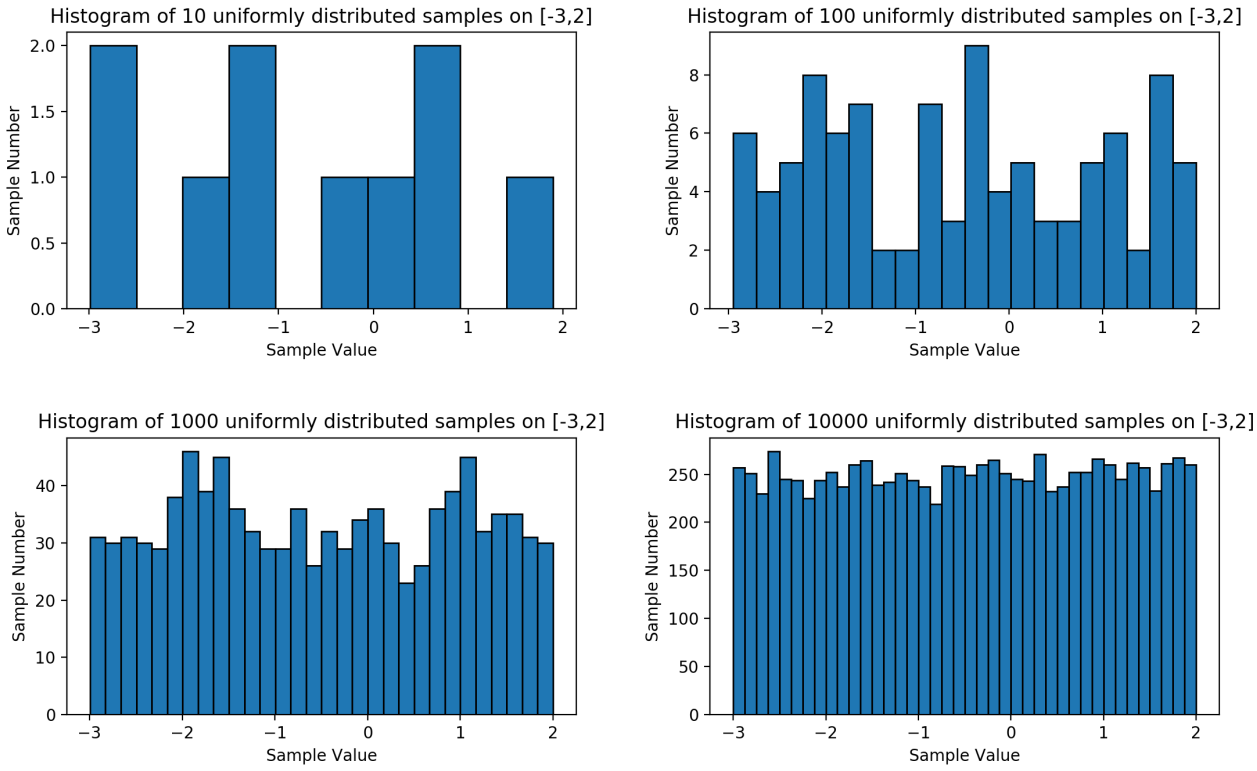

Fig 1.1 Histograms of sampling by 10, 100, 1000, 10000 times

- When the number of samplings is small, the simulation results fluctuate greatly and are much different from the uniform probability distribution. When setting more number of samplings, the outcome is closer to the uniform probability distribution. Therefore, in order to better simulate

uniform or other probability distributions, we always sample as much as possible when conditions permit. Here I choose 10000 samples to simulate [-3,2] uniformly.

Code:

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

#Question 1 part(a)

#set different numbers of samples
sample_number = np.array([10,100,1000,10000])
#get these 4 results and plot 4 histograms
for i in range(0,4):
    #get the samples on interval[-3,2]
    samples = np.random.uniform(0,5,(sample_number[i],))-3.0
    #plot the histogram at the (i+1)th position
    #(2,2) put these 4 histogram into 2 rows and 2 columns
    plt.subplot(2,2,i+1)
    #plot samples as a histogram
    #bins = 10,20,30,40 for deifferent sample numbers
    plt.hist(samples, bins = (int)(math.log10(sample_number[i]))*10, edgecolor = "black")

    plt.title('Histogram of {} uniformly distributed samples on [-3,2]'.format(sample_number[i]))
    plt.xlabel('Sample Value')
    plt.ylabel('Sample Number')

plt.show()
```

Question 1. (b)

Answer:

- After sampling the uniform 10000 times, compute the sample mean and sample variance:
  - **sample mean = -0.4948**
  - **sample variance = 2.0710**
- Also theoretically, the mean and the variance of U[-3,2] will be:
  - mean = (a + b) / 2 = 0.5
  - variance = (b - a) ^ 2 / 12 = 2.083
- **For 10000 samples case:** the differences between the sample values and the theoretial values are all on the order of 0.001, which is relatively small. Thus we can use the values of samples to approximately measure a given distribution.
- For 10, 100, 1000 samples cases: if the number of samples is smaller, the sample mean or sample value will obtain larger difference to the theoroetical value, which means small number of samples has worse ability to simulate the outcome. The result is just the same as part(a).

```
Theoretical values for [-3,2]: mean =  -0.5  variance =  2.0833333333333335
sample  10  times: mean =  -1.4294710305799825  variance =  1.1931631641007923
sample  100  times: mean =  -0.6006035974800504  variance =  1.9559063386923448
sample  1000  times: mean =  -0.46601904213061573  variance =  2.0568194925717016
sample  10000  times: mean =  -0.4948164073850088  variance =  2.07101516104259
```

Fig 1.2 Output of sample means and sample variances

- **Repeat the experiment 10 times:** the sample values obtained are slightly different around the theoretical value. This is because the sample data obtained from each experiment will not be the same, so their sample values are also different.

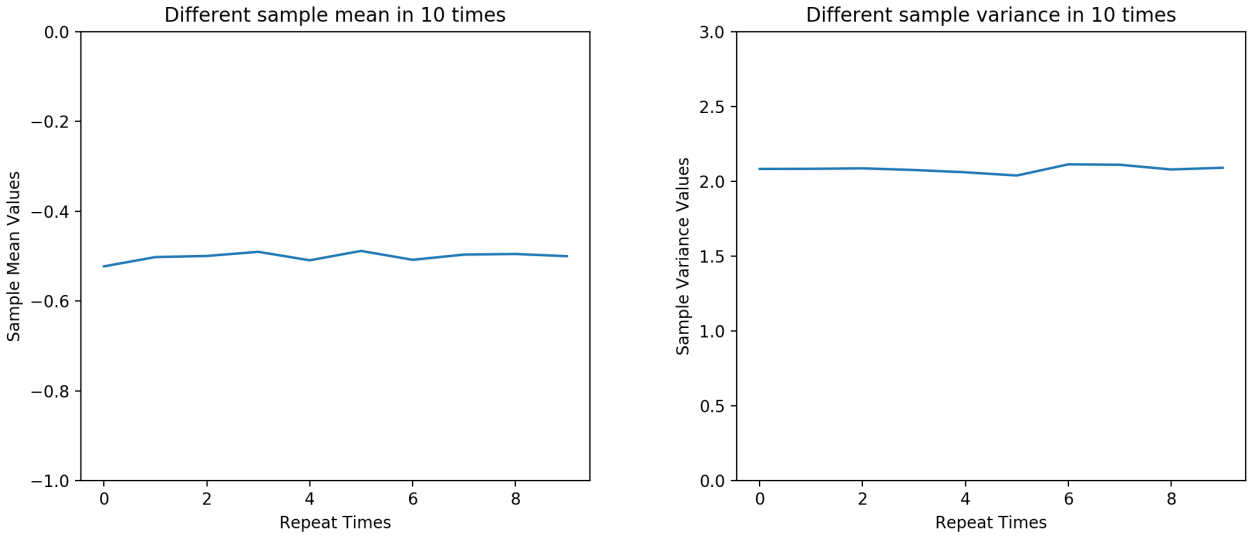

Fig 1.3 Sample means and sample variance of repeating 10 times

Code:

```
1   import numpy as np
2   import matplotlib.pyplot as plt
3
4   #Question 1 part(b)
5
6   #theoretial values for [-3,2]
7   a = -3
8   b = 2
9   theo_mean = (a+b)/2
10  theo_var = (b-a)*(b-a)/12
11  print("Theoretical values for [-3,2]: mean = ",
12      theo_mean," variance = ",theo_var)
13  #sample values when getting different number of samples
14  sample_number = np.array([10,100,1000,10000])
15  for i in range(0,4):
16      samples = np.random.uniform(0,5,
    (sample_number[i],))-3.0
17      sample_mean = np.mean(samples)
18      sample_var = np.var(samples)
19      print("sample ",sample_number[i]," times: mean = ",
20          sample_mean," variance = ",sample_var)
21
22
23  #repeate the experiment 10 times: number of samples = 10000
24  sample_number = 10000
25  means_list = [] #different sample mean in 10 times
26  var_list = [] #different sample variance in 10 times
27  for j in range(0,10): #repeat the experiment 10 times
28      #get 10000 samples
29      samples = np.random.uniform(0,5,(sample_number,))-3.0
30      #compute the sample mean and sample variance
31      sample_mean = np.mean(samples)
32      sample_var = np.var(samples)
33      #record the values in lists
34      means_list.append(sample_mean)
35      var_list.append(sample_var)
36  repeat_times = [x for x in range (10)] #x ray of the plot
37  #create the plot of sample mean and repeating times
38  plt.subplot(121)
39  plt.ylim(-1.0,0.0)
40  plt.title("Different sample mean in 10 times")
41  plt.xlabel("Repeat Times")
42  plt.ylabel("Sample Mean Values")
43  plt.plot(repeat_times, means_list)
44  #create the plot of sample variance and repeating times
45  plt.subplot(122)
46  plt.ylim(0.0,3.0)
47  plt.title("Different sample variance in 10 times")
48  plt.xlabel("Repeat Times")
49  plt.ylabel("Sample Variance Values")
```

```
50 plt.plot(repeat_times, var_list)
51
52 plt.show()
```

Question 1. (c)

Answer:

- Bootstrap: first do the sampling 10,000 times and recorde them as the original sample. Then resample the original sample 10,000 times with the same scale as the original sample each time. Also, calculate and record the sample mean and variance for resample data each time. Get the bootstrap distribution of sample mean and variance as below, which are like Gussian distribution.

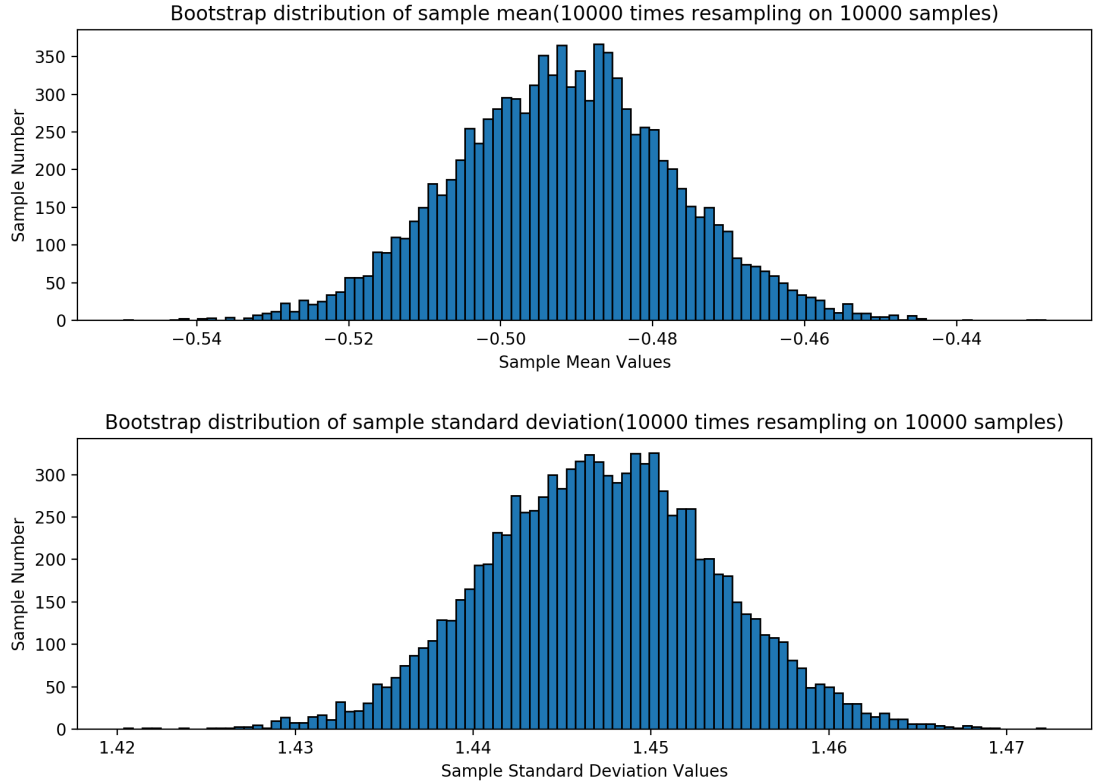

Fig 1.4 Histograms of bootstrap sample mean and sample standard deviation

- **Bootstrap confidence interval = 95%**: After bootstrap, a lot of data will be obtained, and the distribution of these data is close to the Gaussian distribution. Thus we can analyze in which range most data is distributed, which is called confidence interval (C.I.). Here we take C.I. = 95%, which means our sample values have the probability of 0.95 to be in this region. The results obtained are as follows:

	Theorotical	Origin Sample	Bootstrap	Bootstrap C.I.	Length of C.I
Mean	-0.5	-0.4915	-0.4916	[-0.5198, -0.4628]	0.0569
Std	1.4433	1.4473	1.4472	[1.4348, 1.4599]	0.0250

Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.utils import resample
import math

#get the origin sample array
a = -3
b = 2
theo_mean = (a+b)/2
theo_std = math.sqrt((b-a)*(b-a)/12)
#get the origin samples
sample_number = 10000 #10,100,1000,10000
samples = np.random.uniform(0.0,5.0,(sample_number,))-3.0
sample_mean = np.mean(samples)
sample_std = np.std(samples)
#confidence interval = 95%
ci = 95

#bootstrap
resample_times = 10000
mean_list = []
std_list = []
#resample the origin sample for 10000 times
for i in range(resample_times):

    resample_samples = resample(samples, n_samples = sample_number, replac
e = True)
    #sample mean and sample variance
    resample_mean = np.mean(resample_samples)
    resample_std = np.std(resample_samples)
    #combine the values into list
    mean_list.append(resample_mean)
    std_list.append(resample_std)
mean_array = np.array(mean_list)
std_array = np.array(std_list)

#bootstrap confidence interval for sample mean
mean_per1 = np.percentile(mean_array,(100-ci)/2,interpolation='nearest')
mean_per2 = np.percentile(mean_array,(100+ci)/2,interpolation='nearest')
print("Theoretical mean = ",theo_mean)
print("10000 times sample mean = ",sample_mean)
print("Bootstrap confidence interval for sample mean: ")
print("The ",(100-ci)/2,"% percentile is: ",mean_per1)
print("The ",(100+ci)/2,"% percentile is: ",mean_per2)
print("Length of the C.I.(95%): ",mean_per2 - mean_per1)
print("Mean of the bootstrap distribution of sample mean: ",
    np.mean(mean_array))
print(" ")
#bootstrap confidence interval for sample std
std_per1 = np.percentile(std_array,(100-ci)/2,interpolation='nearest')
std_per2 = np.percentile(std_array,(100+ci)/2,interpolation='nearest')
print("Theoretical std = ",theo_std)
print("10000 times sample std = ",sample_std)
print("Bootstrap confidence interval for sample standard deviation: ")
print("The ",(100-ci)/2,"% percentile is: ",std_per1)
print("The ",(100+ci)/2,"% percentile is: ",std_per2)
print("Length of the C.I.(95%): ",std_per2-  std_per1)

print("Mean of the bootstrap distribution of sample standard deviation: ",

    np.mean(std_array))

```

```
61  #show the bootstrap distribution of sample mean
62  plt.subplot(211)
63  plt.hist(mean_array, bins = 100, edgecolor = "black")
64  plt.title("Bootstrap distribution of sample mean")
65  plt.xlabel("Sample Mean Values")
66  plt.ylabel("Sample Number")
67  #show the bootstrap distribution of sample std
68  plt.subplot(212)
69  plt.hist(std_array, bins = 100, edgecolor = "black")
70  plt.title("Bootstrap distribution of sample standard deviation")
71  plt.xlabel("Sample Standard Deviation Values")
72  plt.ylabel("Sample Number")
73  plt.show()
```

Question 2.(a)

Answer:

- **X(k) and X(k+1) are uncorrelated,** which means they are not linearly dependent. Xk is created from sampling at k time, and X(k+1) is created from sampling at k+1 time. Therefore, X(k+1) can be obtained by shifting X(k) one unit to the left. Get the covariance: **cov = 0.0006285**, which is very close to 0. Thus there is almost no linear relationship between X(k) and X(K+1).
- **X(k) and X(k+1) are indepentent.** This result is not concluded from the uncorrelation. In most cases, two uncorrelated random variables does not mean they are totally independent to each other. Being independent means that the value of one of them will not have any effect on the value of the other. Since the values between the two moments are sampled seperately. No matter what value is taken at X(k), the value at X(k+1) can still be taken from the entire interval with the probability unchanged. Therefore, the theoretical conclusion is that the two are independent of each other.
- There are two other ways to explicitly prove this conclusion:
  - Show the plot of X(k) and X(k+1). If the two are completely independent, the value range of the latter will not change after the former is set, thus the entire plane will be covered. For example, if X(k) takes the value of 0.4, then X(k+1) can still take all the points in the range of [0.0, 1.0].

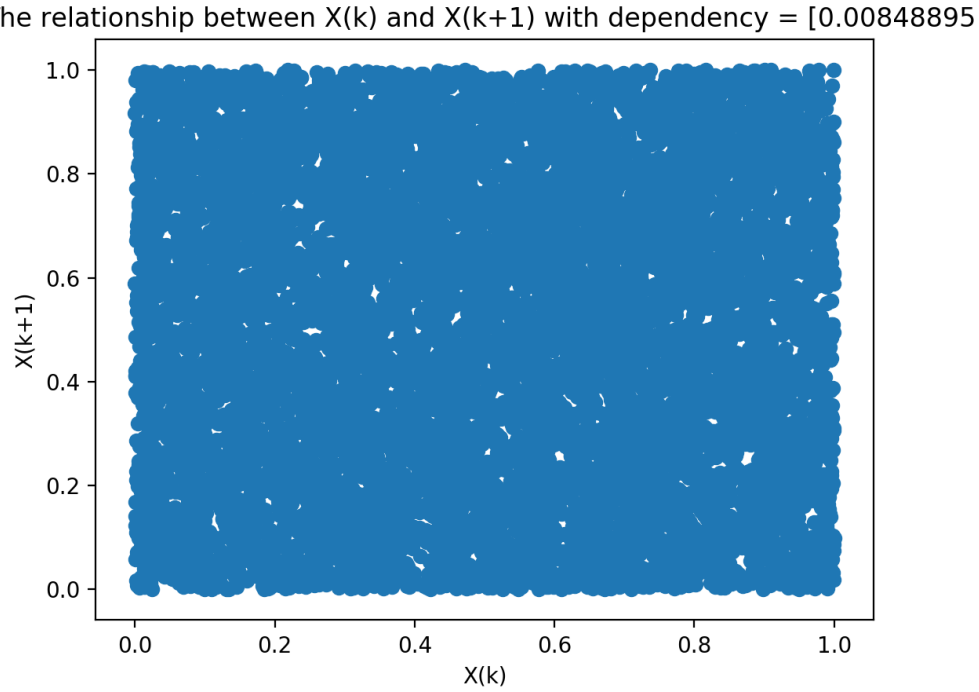

Fig 2.1 Plots of X(k) and X(k+1)

- Using the function: **mutual_info_regression**. This function can compute the mutual influence between two random variables. The more dependent of the two, the larger value it will return. So if two random variables are completely independent, then the value obtained from this function will approach 0. Here we get **dependency = 0.00848895**, which is small enough to show that the two are independent of each other.

Code:

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

from sklearn.feature_selection import mutual_info_regression


length = 10000
#X(k)
X = np.random.uniform(0.0,1.0,(length,))
#X(k+1): shift X to left by 1
X_left1 = np.roll(X,-1)
X_left1[length-1] = 0.0
#compute covariance of X(k) and X(k+1)
cov = np.cov(X,X_left1)
print("Cov[X(k),X(k+1)] = ",cov)

#see independence by function: mutual_info_regression(X,Y)
#measures the dependency of two variables
#return 0 or extremely small value if independent
#X: shape(n_samples,n_features)
#Y: shape(n_samples)
mi = mutual_info_regression(X.reshape(length, 1),X_left1)
print("The dependency between X(k) and X(k+1) is ", mi)
#see independence through plot
plt.scatter(X,X_left1)
plt.xlabel("X(k)")
plt.ylabel("X(k+1)")
plt.title("The relationship between X(k) and X(k
+1) with dependency = {}".format(mi))
plt.show()
```

Question 2. (b)

Answer:

- **X(k) and Y(k) are uncorrelated**. X(k-i) is samples obtained at k-i time, which can be produced by shifting X(k) one unit to the right and set all the values between X(0) to X(i) as 0.0. The calculated cov of X and Y is: **cov = 0.08253031**. This value is an order of magnitude larger than the value in part(a). It can be concluded that X(k) and Y(k) have some correlation, that is, there is a certain linear relationship between the two.
- This correlation can also be reflected in mapping. Unlike the graph in part(a), when the value of X changes here, the value range of Y changes approximately linearly. This is the reflection of the linear correlation between the two variables.

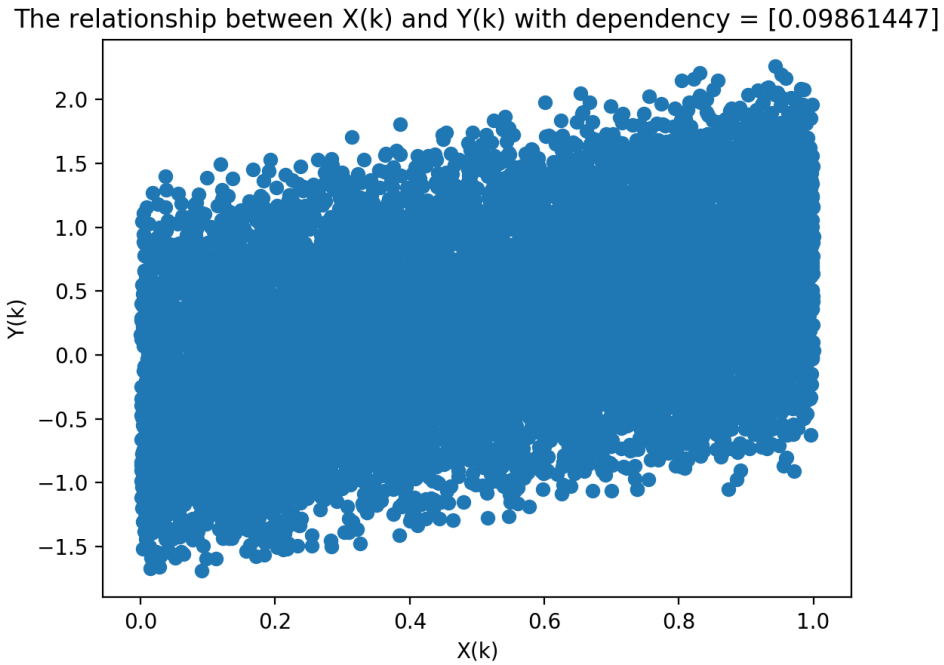

Fig 2.2 Plots of X(k) and Y(k)

- In addition, if two variables are uncorrelated, they must not be independent of each other. The independence parameter obtained by mutual_info_regression is: **dependency = 0.09861447**, not close to 0, which also proves that the two random variables are **not independent** of each other.

Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.feature_selection import mutual_info_regression

length = 10000
#X(k)
X = np.random.uniform(0.0,1.0,(length,))
#X(k-1): shift X to right by 1
X_right1 = np.roll(X,1)
X_right1[0] = 0.0
#X(k-2)
X_right2 = np.roll(X,2)
X_right2[0:2] = 0.0
#X(k-3)
X_right3 = np.roll(X,3)
X_right3[0:3] = 0.0
#Y(k)=X(k)-2*X(k-1)+0.5*X(k-2)-X(k-3)
Y = X - 2 * X_right1 + 0.5 * X_right2 + X_right3

#compute covariance of X(k) and Y(k)
cov = np.cov(X,Y)
print("Cov[X(k),Y(k)] = ",cov)

#see independence through function: mutual_info_regression
 #measure dependency of 2 r.v.: if independent, close to 0
#X: shape(n_samples,n_features)
#Y: shape(n_samples)
mi = mutual_info_regression(X.reshape(length, 1),Y)
print("The dependency between X(k) and X(k+1) is ", mi)
#see independence through plot
plt.scatter(X,Y)
plt.xlabel("X(k)")
plt.ylabel("Y(k)")

plt.title("The relationship between X(k) and Y(k) with depend
ency = {}".format(mi))
plt.show()
```

Question 3.(a)

Answer:

- Sample 10,000 times on the outcomes 0, 1, 2, 3, …, 9 to get the histogram as follows. The cumulative number of each outcome is almost the same with a slight amount of floating. The distribution of histogram is generally close to the uniform ditribution of outcomes.

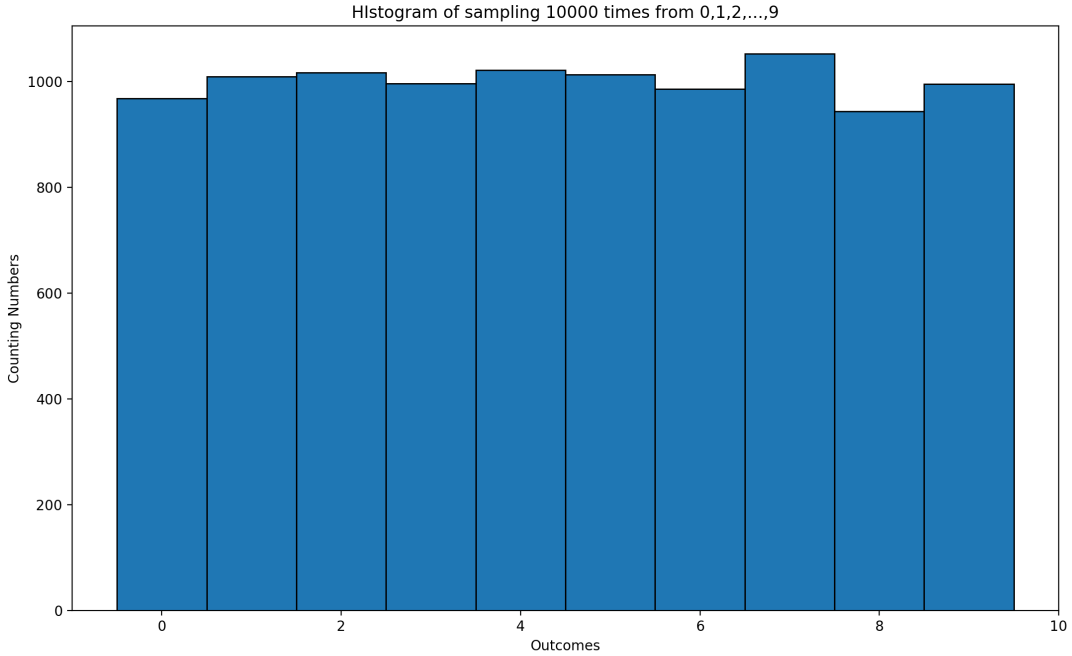

Fig 3.1 Histogram of the origin 10000 samples from 0,1,2,…,9

Code:

```
1   import numpy as np
2   import matplotlib.pyplot as plt
3
4   #number of outcomes: 0,1,2,3...,9
5   M = 10
6   #number of sampling
7   number = 10000
8   samples = np.random.randint(0,M,(number,))
9   #plot thte samples
10  plt.hist(samples, range = (0,M), edgecolor = "black", align =
     "left")
11  plt.title("HIstogram of sampling 10000 times from 0,1,2,...,9
    ")
12  plt.xlabel("Outcomes")
13  plt.ylabel("Counting Numbers")
14  plt.show()
```

Question 3. (b)

Answer:

```
MacProdeMacBook-Pro:project 2 mac-pro$ /usr/local/bin/python3 "/Users/mac-pro/Deskt
10000  Samples from the outcomes:  [0 1 2 3 4 5 6 7 8 9]
Expected counts:  [1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000.]
Sample counts:  [1047  937  982  996 1029  998  989 1005  969 1048]
Chi square test statistic is:  10.774
Chi square test p-value is:  0.29151880066170355
The 95 percentile of chi square dist with  9 degree is:  16.918977604620448
These data fits the uniform distribution 0,1,2,3...,9
```

Fig 3.2 Output of goodness-of-fit test on 95% confidence interval between the origin samples and expected outcomes on 0,1,2,3,…,9

- **Chi square test statistic** : it is used to measure the difference between the obtained distribution and the expected distribution. Chi square test value is obtained by accumulating the difference between the obtained counting number and the expected number of each outcomes:
  - Formula of chi square test statistic: $$\chi_c^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$
  - **Chi square test satistic value of this experiment = 10.774**
- **P-value:** the probability of larger value than the obtained chi square test statistic value. In this experiment, get **p-value = 0.2915**. According to the blue notation in the table delow, as the chi square value is 10.774, the obtained p-value is satisfied in the expected p-value region from 0.25 to 0.50.

Percentage Points of the Chi-Square Distribution

Degrees of Freedom	Probability of a larger value of x^2								
	0.99	0.95	0.90	0.75	0.50	0.25	0.10	0.05	0.01
1	0.000	0.004	0.016	0.102	0.455	1.32	2.71	3.84	6.63
2	0.020	0.103	0.211	0.575	1.386	2.77	4.61	5.99	9.21
3	0.115	0.352	0.584	1.212	2.366	4.11	6.25	7.81	11.34
4	0.297	0.711	1.064	1.923	3.357	5.39	7.78	9.49	13.28
5	0.554	1.145	1.610	2.675	4.351	6.63	9.24	11.07	15.09
6	0.872	1.635	2.204	3.455	5.348	7.84	10.64	12.59	16.81
7	1.239	2.167	2.833	4.255	6.346	9.04	12.02	14.07	18.48
8	1.647	2.733	3.490	5.071	7.344	10.22	13.36	15.51	20.09
9	2.088	3.325	4.168	5.899	8.343	11.39	14.68	16.92	21.67
10	2.558	3.940	4.865	6.737	9.342	12.55	15.99	18.31	23.21
11	3.053	4.575	5.578	7.584	10.341	13.70	17.28	19.68	24.72

Fig 3.3 Table of the probability of chi-square distribution

- **95% confidence level**: means if we sample from the expected distribution, then there is 0.95 probability that the chi square value will smaller than a certain value. The larger chi square is, the less similar the obtained and the expected distribution will be . The table above shows what is the

probability that a chi square value is greater than a certain value. Thus 95% confidence level corresponds to 0.05 probability greater than the certain value. And the degree here is 9 (there are 10 classes in total, but you only need to know the 9 of them to decide the 10th result). Therefore, the corresponding value is **16.92** (red box in the figure). That is, the probability of the obtained chi value greater than 16.92 is 0.05. As long as the obtained chi value is less than 16.92, then this distribution is within 95% confidence level. We say the obtained distribution fits the expected distribution.

- **Results:**
  - **Chi square = 10.774;**
  - **95% confidence level with 9 degree =16.92:**
  - **10.774< 16.92**, thus the obtained data **fits** samples from uniform distribution 0, 1, 2, 3, …, 9

Code:

```
1   import numpy as np
2   from scipy.stats import chisquare
3   from scipy.stats import chi2
4   import math
5   import matplotlib.pyplot as plt
6
7   #Question 3 Part (b)
8
9   #number of outcomes: 0,1,2,3...,9
10  M = 10
11  #number of sampling
12  number = 10000
13  samples = np.random.randint(0,M,(number,))
14  #count the number of each outcomes
15  unique, count = np.unique(samples,return_counts = True)
16  #compute the expected number of each outcomes
17  exp = np.full((M,),number/M)
18  #compute the test value
19  chisq, p_value= chisquare(count, exp)
20  #compute the 95% confidence interval
21  ci = chi2.ppf(0.95, M-1)
22  #print the results to terminal
23  print(number," Samples from the outcomes: ", unique)
24  print("Expected counts: ",exp)
25  print("Sample counts: ",count)
26  print("Chi square test statistic is: ", chisq)
27  print("Chi square test p-value is: ",p_value)
28
    print("The 95 percentile of chi square dist with ", M-1, "deg
    ree is: ",ci)
29  if chisq <= ci :
```

```
30
    print("These data fits the uniform distribution 0,1,2,3..,9")
31  else:
32
    print("These data dose not fit the uniform distribution 0,1,2,3...,9")
```

Question 3. (c)

Answer:

- The samples obtained from part(b) is sampled on the outcomes 0,1,2, ..., 9, which means the counting number at outcome 10 in these samples is 0. Also, the expected distribution is on the outcomes 1, 2, 3, ..., 10. Thus the expected value on outcome 0 is 0. When coding, both obtained and expected counting arrays are expanded to 11 elements, that is, the counting values of outcomes from 0 to 10 are compared. To make the calculating work, we cannot directly set 0 to the value of counting numbers. Instead, we set an extremely small number, 1, to the value of counting number, which is small enough to be seen as not occur.

Results:

```
10000  Samples from the outcomes:  [ 1  2  3  4  5  6  7  8  9 10]
Expected counts:  [   1. 1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000.]
Sample counts:  [1047  937  982  996 1029  998  989 1005  969 1048    1]
Chi square test statistic is:  1095122.5659999999
Chi square test p-value is:  0.0
The 95 percentile of chi square dist with  10 degree is:  18.307038053275146
These data dose not fit the uniform distribution 1,2,3...,9,10
```

Fig 3.2 Output of goodness-of-fit test on 95% confidence interval between the origin samples and expected outcomes on 1,2,3,…,9,10

- **Chi square = 1095122.566**
- **95% confidence level with 10 degree = 18.3070:**
- **1095122.566 >> 18.3070,** thus the obtained data from part (b) **does not fit** samples from uniform distribution 1, 2, 3, …, 9, 10

- Chi square value is much greater than the 95% confidence level. This large chi square value also leads to pvalue = 0.0 , which means the probability of the obtained data sampling from expected distribution is zero. This satisfy the fact that our obtained data is sampled from outcomes of 0 to 9, not from expected outcomes of 1 to 10.
- The reason of large chi square value is the big difference of counting values on two outcomes:
  - expected value on outcome 0 is 1, but obtained value on outcome 0 is 1047
  - expected value on outcome 10 is 1000, but obtained value on outcome 10 is 1

Code: **continue after part(b)**

```
#Question 3 Part (c)

#extend count array to 0,1,2,3,...,10
#set the counting number of 10 as 1
count_c = np.insert(count, M, 1)
#extend expected array to 0,1,2,3,...,10
#set the counting number of 0 as 1
exp_c = np.insert(exp, 0, 1)
#compute the test value
chisq_c, p_value_c= chisquare(count_c, exp_c)
#compute the 95% confidence interval
ci_c = chi2.ppf(0.95, M)
#print the results
print(number," Samples from the outcomes: ", unique+1)
print("Expected counts: ",exp_c)
print("Sample counts: ",count_c)
print("Chi square test statistic is: ", chisq_c)
print("Chi square test p-value is: ",p_value_c)

print("The 95 percentile of chi square dist with ", M, "degre
e is: ",ci_c)
if chisq_c <= ci :

    print("These data fits the uniform distribution 1,2,3...,
9,10")
else:

    print("These data dose not fit the uniform distribution 1
,2,3...,9,10")
```