University of Southern California

EE511 Simulation Methods for Stochastic Systems

Project #2 – Samples and Statistics

BY

Mohan Krishna Thota

USC ID: 6683486728

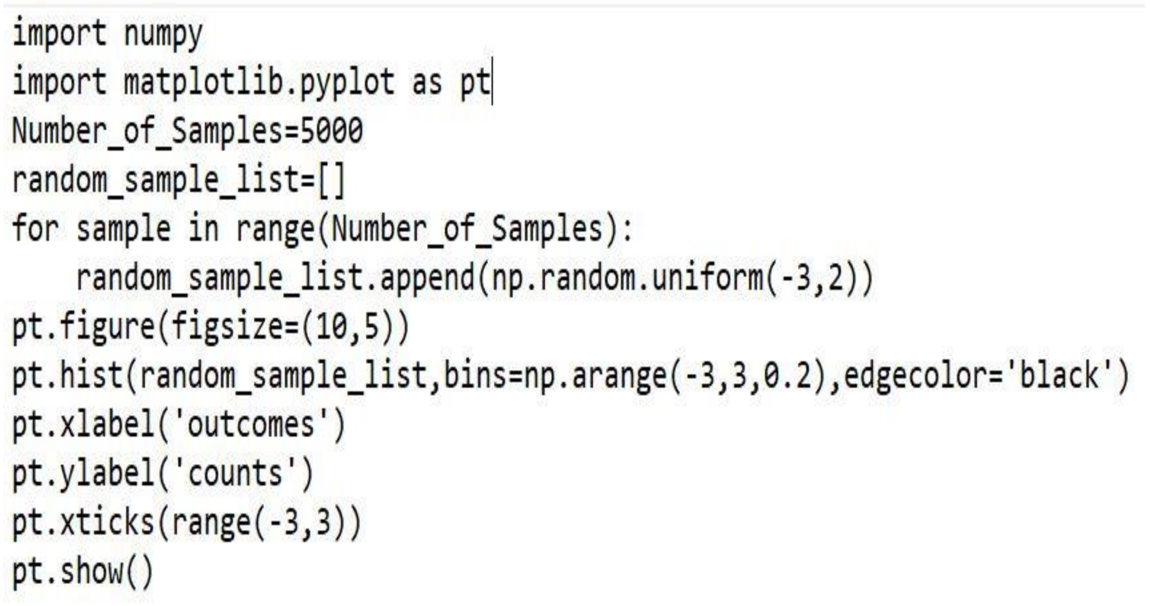
mthota@usc.edu

Question 1:

Simulate sampling uniformly (how many?) on the interval [-3,2]
a. Generate a histogram of the outcomes.
b. Compute the sample mean and sample variance for your samples. How do these values compare to the theoretical values? If you repeat the experiment will you compute a different sample mean or sample variance?
c. Compute the bootstrap confidence interval (what width?) for the sample mean and sample standard deviation.

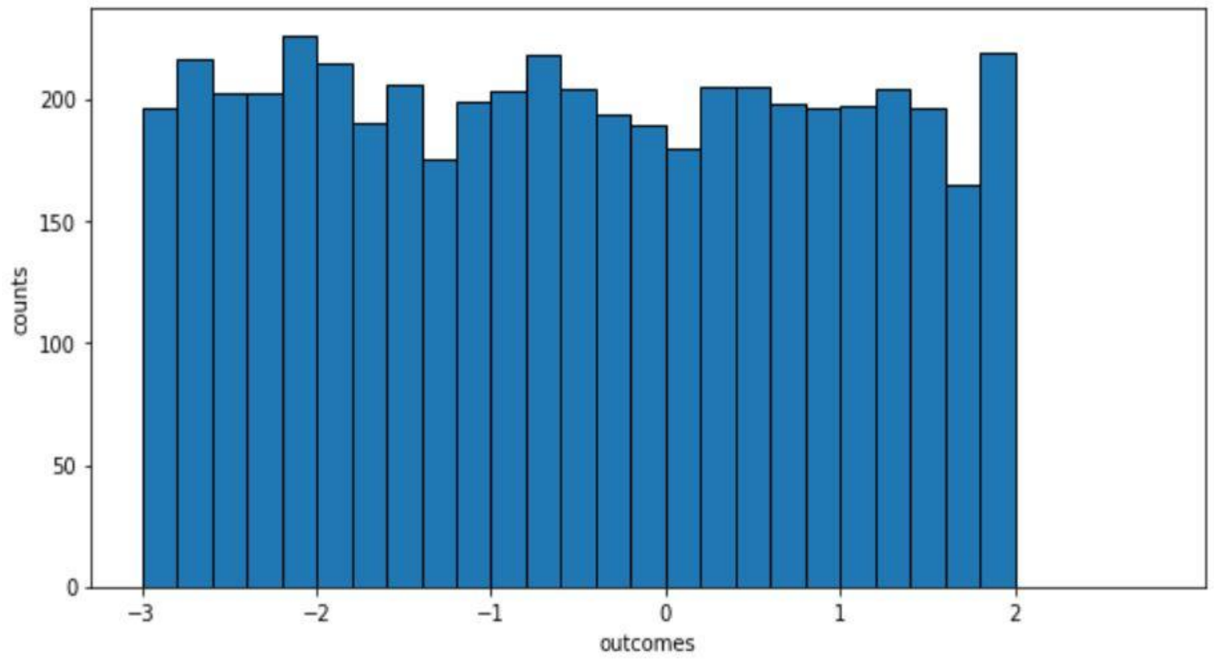
- We know that for a Uniform distribution (a, b). Theoretical computation of mean and variance are as follows:
- Mean= M= $a + b/2$,variance=$(b - a)^2/2$.
- Here in python, the samples are generated using NumPy <np. random. uniform> function which gives us uniform sample in a given distribution.
- Here a total of 5000 values are sampled, where the sampled values are taken to a list.
- From the histogram picture, we could see that with width=0.1, there is fluctuation in the bins of the histogram.
- Fluctuation here is due to scarcity of data. The same is proved when smoother bins are obtained for a total of 50000 samples.

CODE:



```
import numpy
import matplotlib.pyplot as pt
Number_of_Samples=5000
random_sample_list=[]
for sample in range(Number_of_Samples):
    random_sample_list.append(np.random.uniform(-3,2))
pt.figure(figsize=(10,5))
pt.hist(random_sample_list,bins=np.arange(-3,3,0.2),edgecolor='black')
pt.xlabel('outcomes')
pt.ylabel('counts')
pt.xticks(range(-3,3))
pt.show()
```

Histogram:

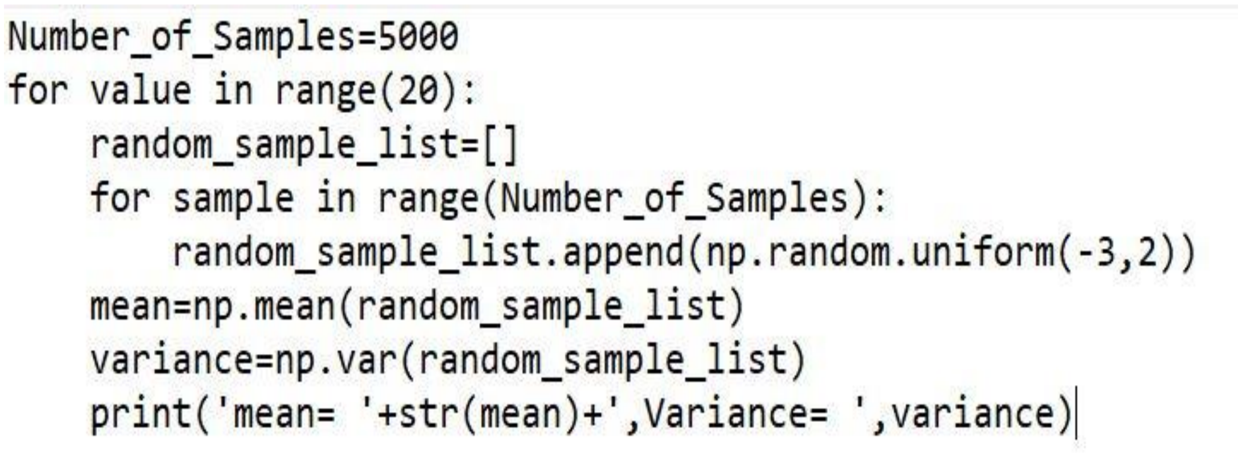


(b).

- In order to check the variations in the mean and variance of both theoretical and obtained values.
- Mean and variance are calculated each different time keeping the number of samples constant.

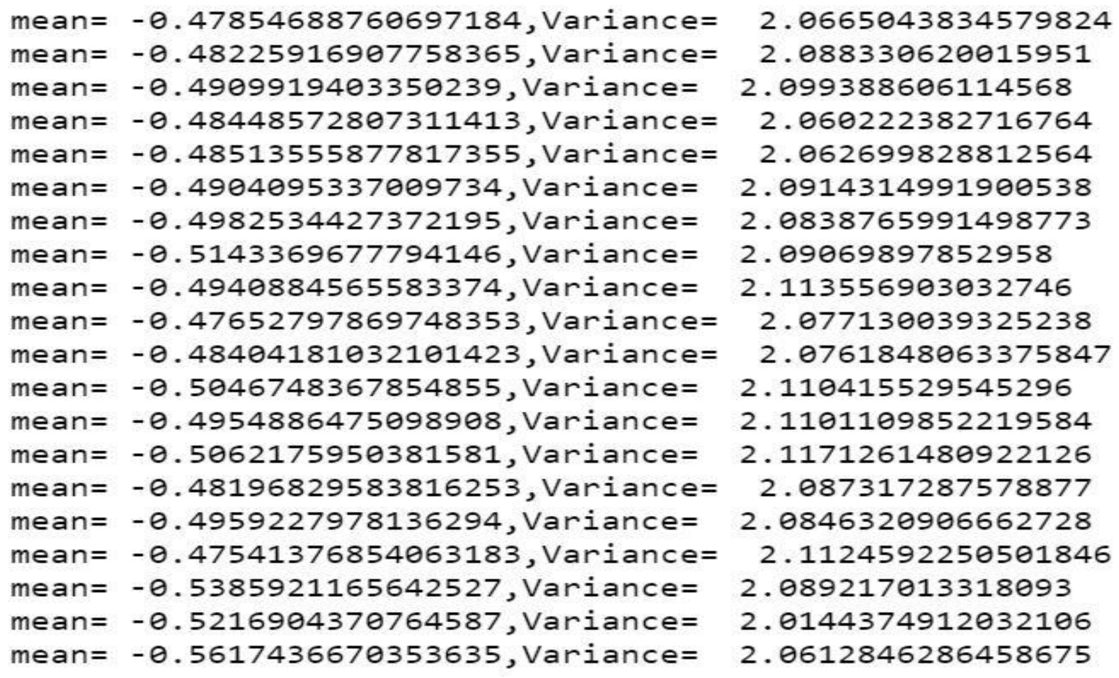
- We could see that values are not same each time as the number are generated randomly each different time.

CODE:



```
Number_of_Samples=5000
for value in range(20):
    random_sample_list=[]
    for sample in range(Number_of_Samples):
        random_sample_list.append(np.random.uniform(-3,2))
    mean=np.mean(random_sample_list)
    variance=np.var(random_sample_list)
    print('mean= '+str(mean)+',Variance= ',variance)
```

Output:



```
mean= -0.47854688760697184,Variance=  2.0665043834579824
mean= -0.4822591690775836,Variance=  2.08833062001595l
mean= -0.490991940335023,Variance=  2.09938860611456
mean= -0.48448572807311413,Variance=  2.060222382716764
mean= -0.48513555877817355,Variance=  2.062699828812564
mean= -0.4904095337009734,Variance=  2.0914314991900538
mean= -0.4982534427372195,Variance=  2.0838765991498773
mean= -0.514336967779414,Variance=  2.0906989785295
mean= -0.4940884565583374,Variance=  2.113556903032746
mean= -0.47652797869748353,Variance=  2.077130039325238
mean= -0.48404181032101423,Variance=  2.0761848063375847
mean= -0.5046748367854855,Variance=  2.110415529545296
mean= -0.4954886475098908,Variance=  2.1101109852219584
mean= -0.5062175950381581,Variance=  2.117126148092212
mean= -0.48196829583816253,Variance=  2.087317287578877
mean= -0.4959227978136294,Variance=  2.0846320906662728
mean= -0.47541376854063183,Variance=  2.1124592250501846
mean= -0.538592116564252,Variance=  2.089217013318093
mean= -0.5216904370764587,Variance=  2.0144374912032106
mean= -0.561743667035363,Variance=  2.0612846286458675
```

Experiment:
When the number of samples are increased, this difference between the calculated and obtained values are reduced and they are almost same

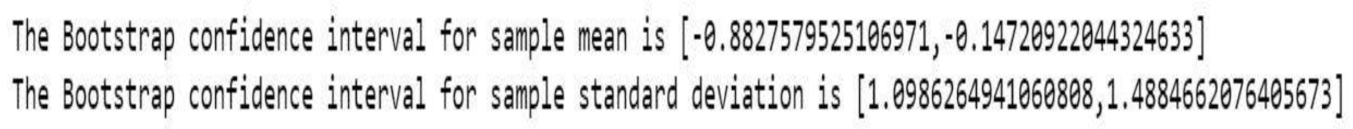
.

(C).

– **Bootstrap**
  - A statistical technique employed to estimate the variation of the statistics by performing the computations on the data itself. It is sampling with replacement.
  - A confidence interval is the interval which is likely to contain the parameter of interest.
  - 95% of confidence interval would mean that, if the same data is resampled number of times and intervals are estimated each time, then the resulting interval would contain the parameter of our interest 95% of chances.

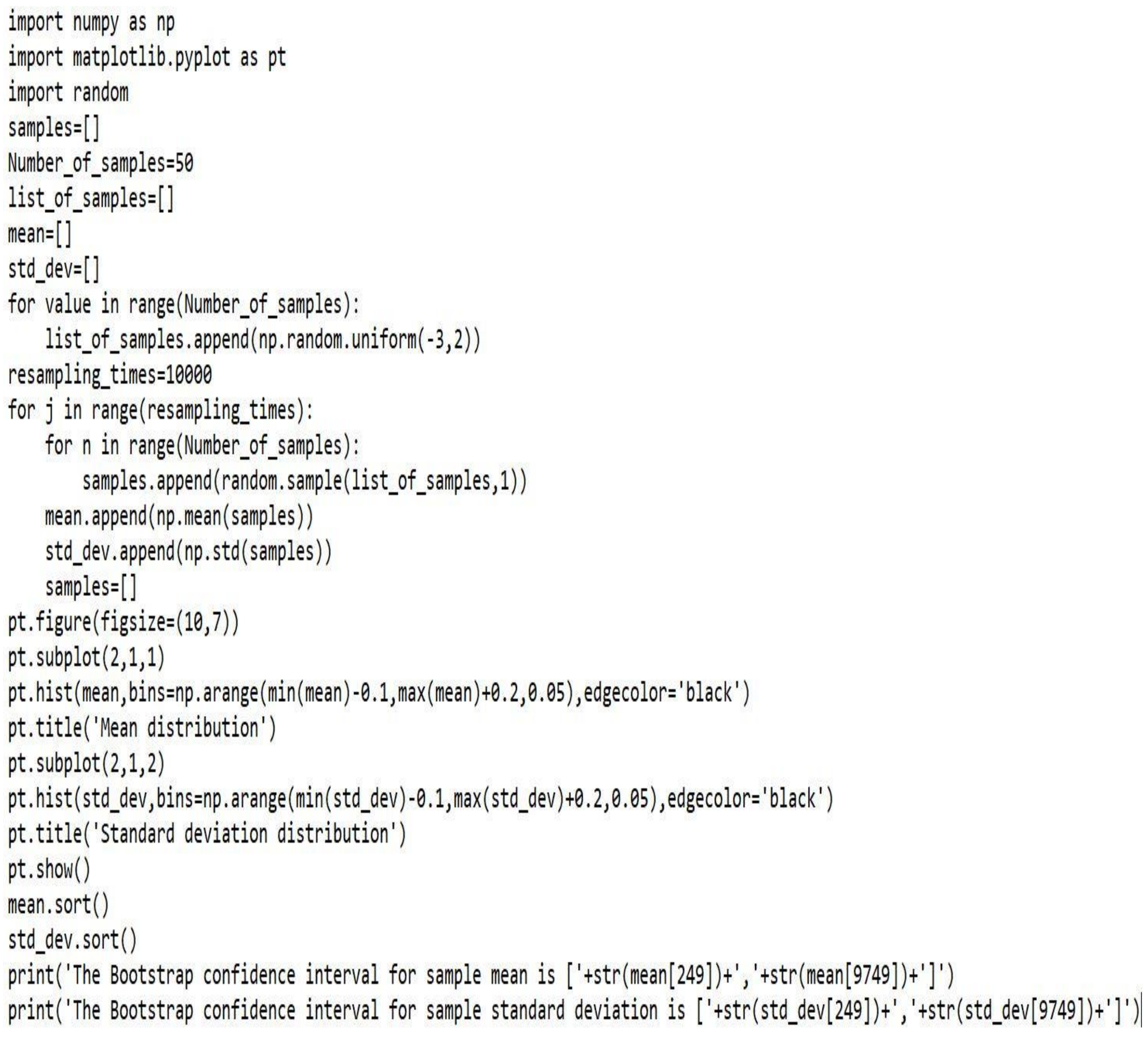
– **Procedure:**
  - Here in the problem, NumPy's <np.random.uniform> function is used to resample the data.
  - Resampling is done on each occasion of 1000 times.
  - For each resampled data, it is stored in a list and corresponding mean and standard deviation of the list is calculated.
  - As a result, at the end of all occasions (1000), a list of all means and standard deviations are obtained, and they are sorted using sort () function.
  - Since the 95% of confidence level is set, the confidence interval is calculated between 25th and 975th mean value.

Output:



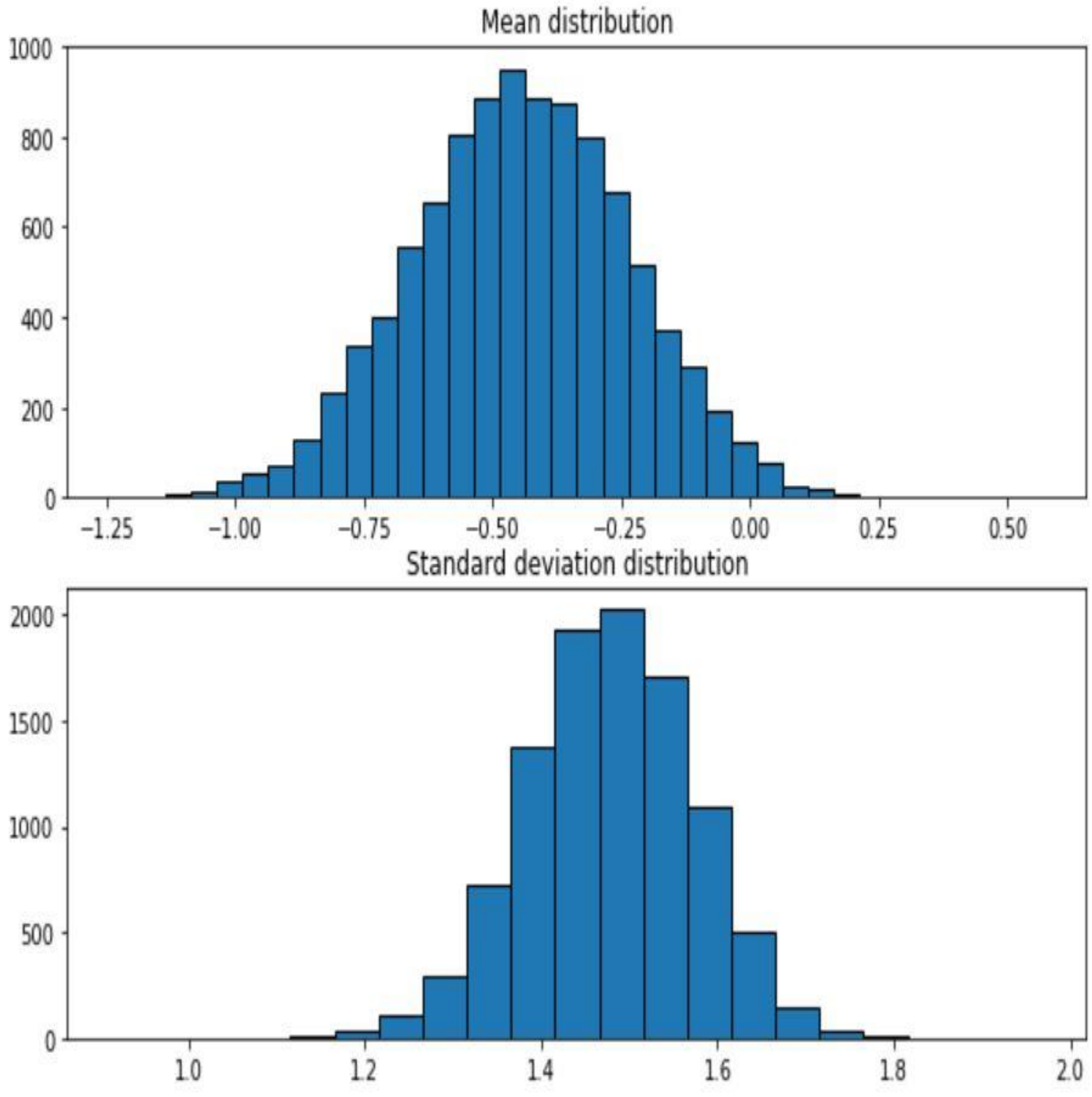
```
The Bootstrap confidence interval for sample mean is [-0.8827579525106971,-0.14720922044324633]
The Bootstrap confidence interval for sample standard deviation is [1.0986264941060808,1.4884662076405673]
```

Code:



```
import numpy as np
import matplotlib.pyplot as pt
import random
samples=[]
Number_of_samples=50
list_of_samples=[]
mean=[]
std_dev=[]
for value in range(Number_of_samples):
    list_of_samples.append(np.random.uniform(-3,2))
resampling_times=10000
for j in range(resampling_times):
    for n in range(Number_of_samples):
        samples.append(random.sample(list_of_samples,1))
    mean.append(np.mean(samples))
    std_dev.append(np.std(samples))
    samples=[]
pt.figure(figsize=(10,7))
pt.subplot(2,1,1)
pt.hist(mean,bins=np.arange(min(mean)-0.1,max(mean)+0.2,0.05),edgecolor='black')
pt.title('Mean distribution')
pt.subplot(2,1,2)
pt.hist(std_dev,bins=np.arange(min(std_dev)-0.1,max(std_dev)+0.2,0.05),edgecolor='black')
pt.title('Standard deviation distribution')
pt.show()
mean.sort()
std_dev.sort()
print('The Bootstrap confidence interval for sample mean is ['+str(mean[249])+','+str(mean[9749])+']')
print('The Bootstrap confidence interval for sample standard deviation is ['+str(std_dev[249])+','+str(std_dev[9749])+']')
```

Histogram:


Mean distribution
1000
800
600
400
200
0
−1.25
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
Standard deviation distribution
2000
1500
1000
500
0
1.0
1.2
1.4
1.6
1.8
2.0

Question 2:

Produce a sequence X by drawing samples from a standard uniform random variable.

a) Compute Cov[Xk,Xk+1]. Are Xk and Xk+1 uncorrelated? what can you conclude about the independence of Xk and Xk+1?
b) Compute a new sequence Y where: Y[k]= X[k]-2.X[k-1]+0.5.X[k-2]-X[k-3]. Assume X[k]=0 for k<=0. Compute Cov(Xk,Yk). Are they uncorrelated?

Solution:

– **Covariance:**

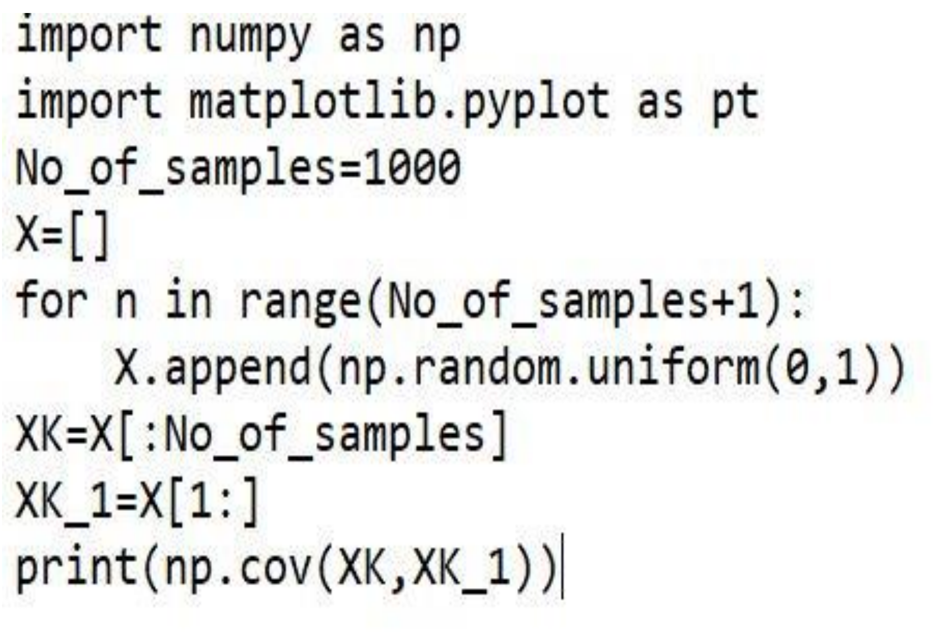
- Covariance helps us to determine the relation between two sets of data by statistical calculations.
- It acts as a measure of how changes in one variable can influence another variable. It helps in determining the degree to which variables are linearly associated.
- The covariance of *X* and *Y*, denoted *Cov*(*X*,*Y*) or $\boldsymbol{\sigma_{XY}}$**,** is defined as:

Cov(X,Y)==E[(X−μX)(Y−μY)]

– **Solution:**

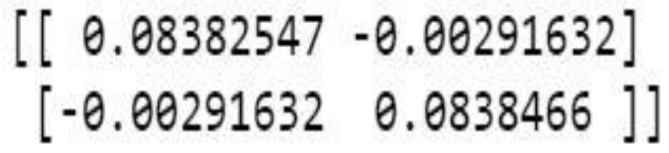
- A total of N (1000 samples) are taken as a sequence, and stored as X. To get the time shifted sequence is obtained by taking first sequence from 0 to (N-1) and Xk+1 sequence is obtained by taking [1:N]. Covariance is calculated by using inbuilt <np.conv()> function. And printed as follows
- We know if COV(X,Y)=0 then X,Y are uncorrelated.
- Here the covariance obtained is 0.00122 which is almost equal to zero. Therefore, the two sequences are uncorrelated.
- With respect to independence, if two variables are independent the covariance is zero but the vice versa is not true.
- Therefore, even though the covariance is zero, here the variables are not independent since one sequence is time shifted version of other.

Code:



```
import numpy as np
import matplotlib.pyplot as pt
No_of_samples=1000
X=[]
for n in range(No_of_samples+1):
    X.append(np.random.uniform(0,1))
XK=X[:No_of_samples]
XK_1=X[1:]
print(np.cov(XK,XK_1))
```

Output:

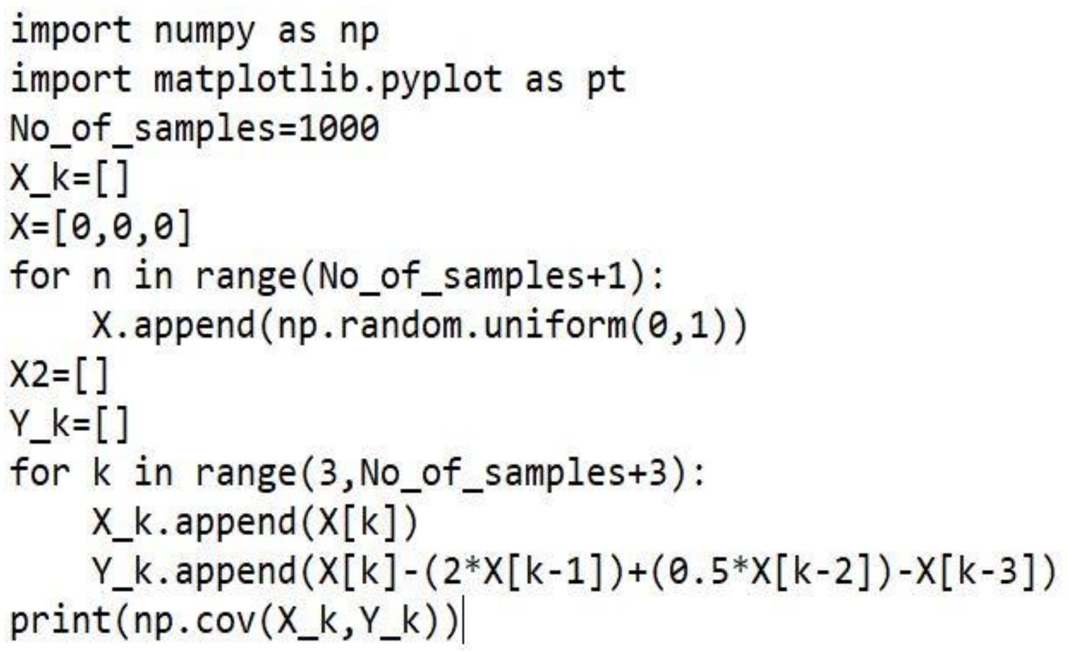


```
[[ 0.08382547 -0.00291632]
 [-0.00291632  0.0838466 ]]
```

(b).

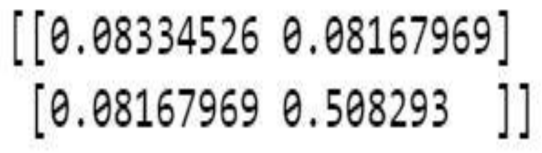
- Here the second sequence Y[k] is obtained with the formula as given in the question
- The covariance is calculated with inbuilt python function, we could see that it is not equal to zero.
- Therefore, they are correlated as the new sequence is a linear combination of the X[k] sequence.

Code:



```
import numpy as np
import matplotlib.pyplot as pt
No_of_samples=1000
X_k=[]
X=[0,0,0]
for n in range(No_of_samples+1):
    X.append(np.random.uniform(0,1))
X2=[]
Y_k=[]
for k in range(3,No_of_samples+3):
    X_k.append(X[k])
    Y_k.append(X[k]-(2*X[k-1])+(0.5*X[k-2])-X[k-3])
print(np.cov(X_k,Y_k))
```

Output:



```
[[0.08334526 0.08167969]
 [0.08167969 0.508293  ]]
```

Question 3:

Let M = 10. Simulate (uniform) sampling with replacement from the outcomes 0, 1, 2, 3, …, M-1.

a. Generate a histogram of the outcomes.
b. Perform a statistical goodnesfit test to conclude at the 95% confidence level if your data fits samples from a discrete uniform distribution 0, 1, 2, …, 9.
c. Repeat (b) to see if your data(the same data from b) instead fit an alternate u niform distribution 1, 2, 3,.. , 10.

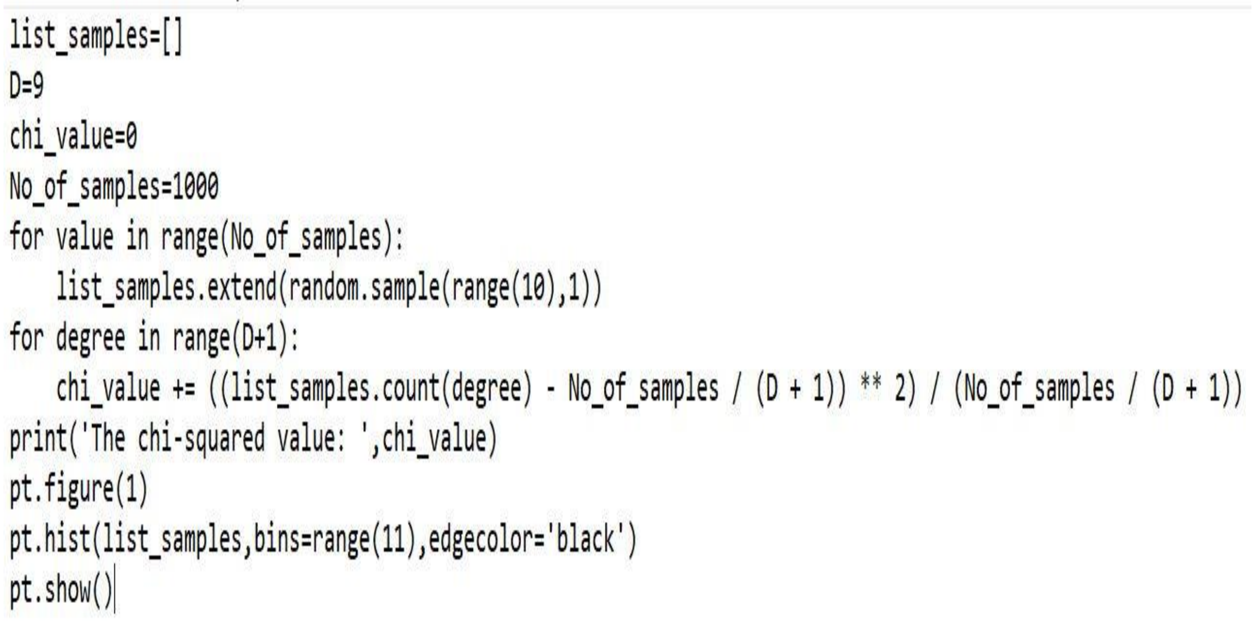
Description:

Goodness of fit test:

- If we have an observed set of data and want to know how well it reflects the data, that is, how close are the observed values to those which would be expected, we can use the chi-square goodness of fit test.
- Chi-square goodness-of-fit is very appropriate when the sampling method is random sampling.
- For a random variable X it is given by
    - X^2= (sum (observed values-expected values) ^2)/expected
    - values
- To perform statistical goodness-of-fit, the expected value is calculated considering equal probability.
- So 1/M is multiplied with N samples and (1,M) array with ones to get the expected value sequence.

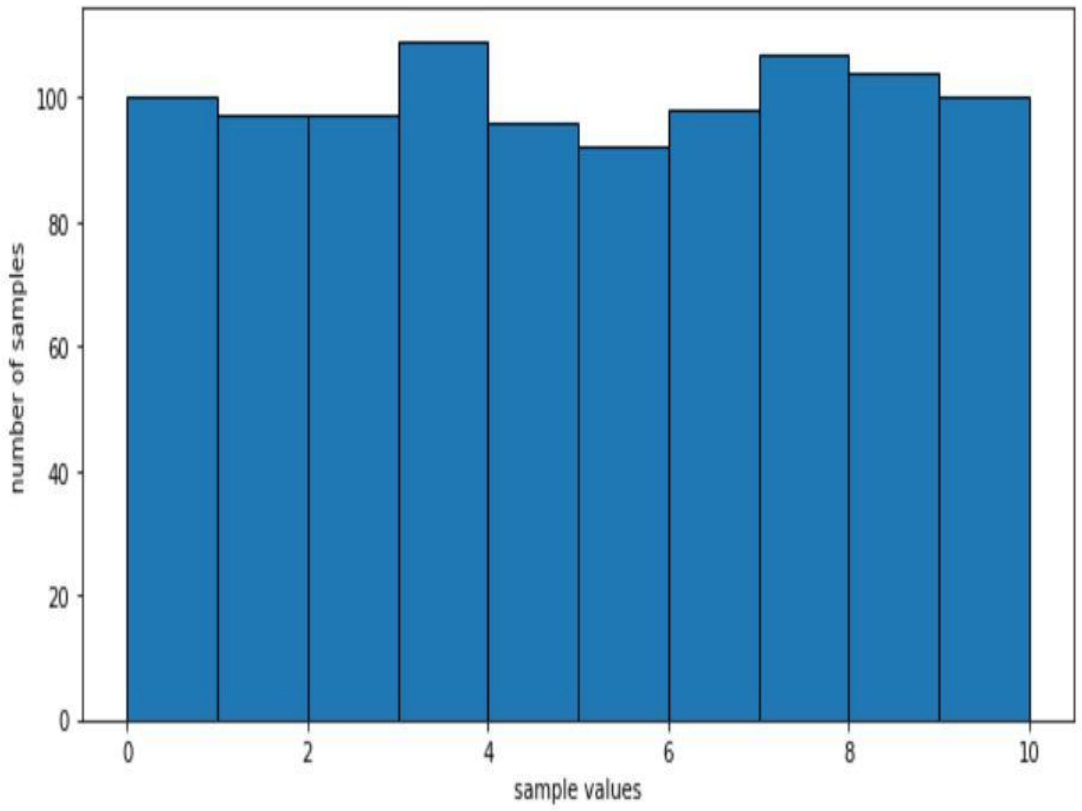
(a).

Code:



```
list_samples=[]
D=9
chi_value=0
No_of_samples=1000
for value in range(No_of_samples):
    list_samples.extend(random.sample(range(10),1))
for degree in range(D+1):
    chi_value += ((list_samples.count(degree) - No_of_samples / (D + 1)) ** 2) / (No_of_samples / (D + 1))
print('The chi-squared value: ',chi_value)
pt.figure(1)
pt.hist(list_samples,bins=range(11),edgecolor='black')
pt.show()
```

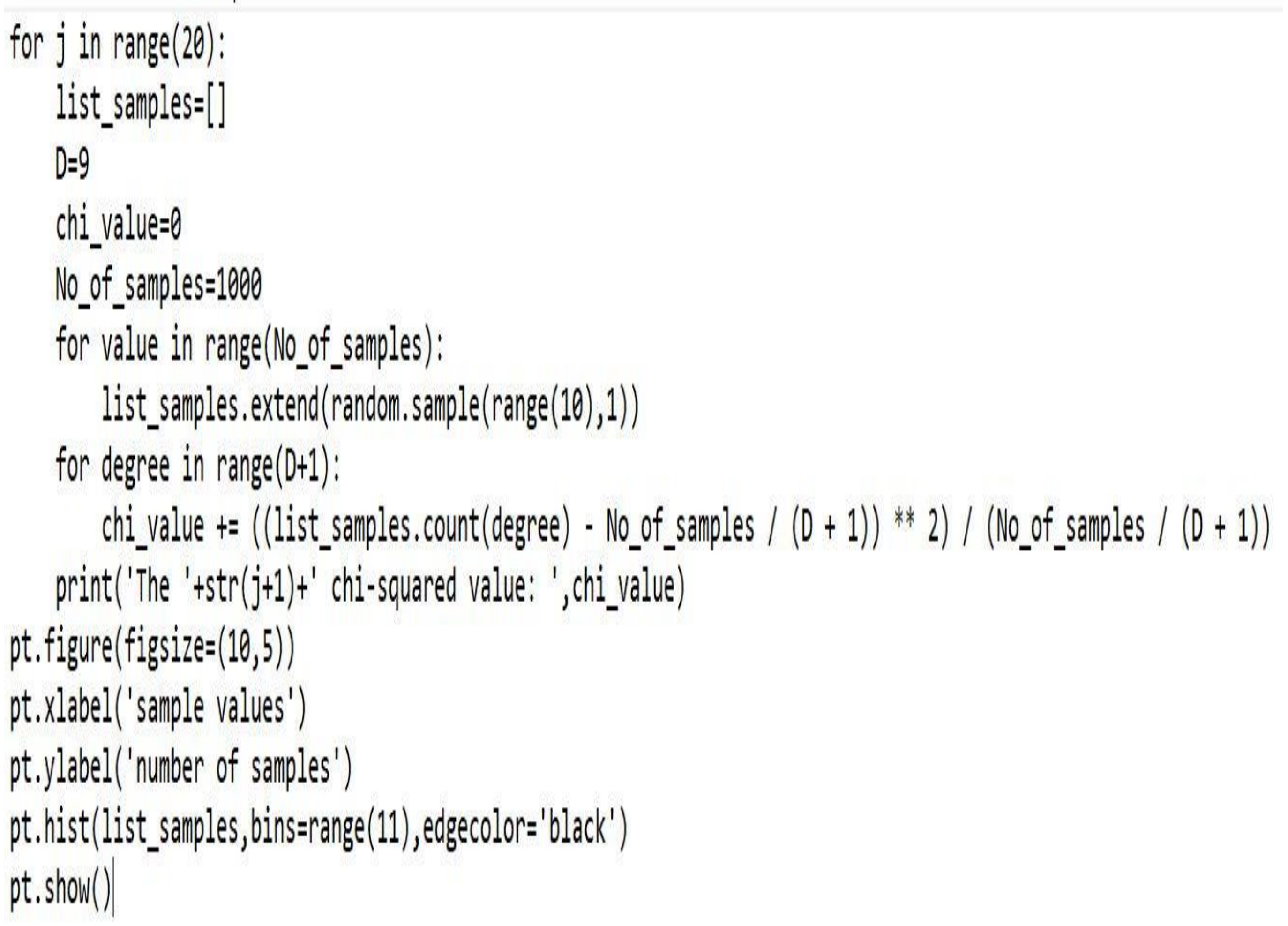
Histogram



(b).

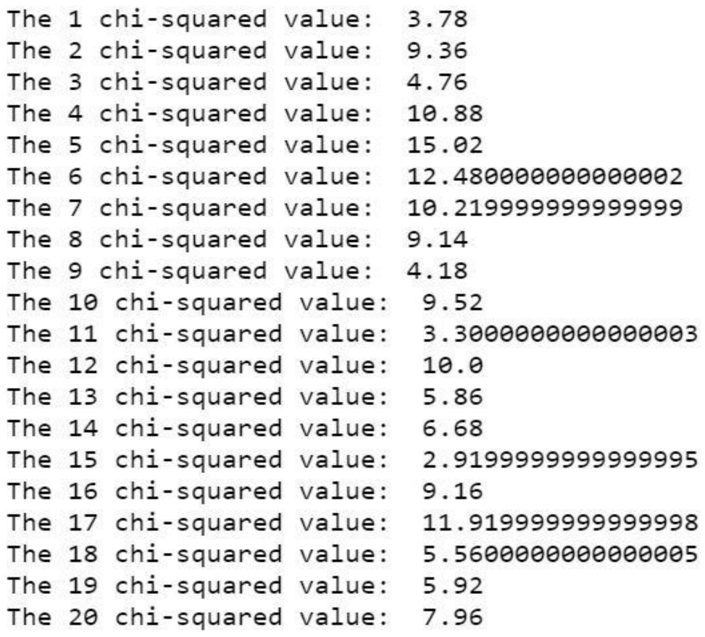
- Here the samples are generated using <random.sample()> function. The bins are set as [<=1,2,3,4,5,6,7,8,9].
- For 95% confidence interval, expected value is 100, and degree freedom is 9. The calculated threshold value is 16.9160. If the obtained value is less than threshold value, then it is a good fit.
- Here all the values obtained are less than the threshold, therefore it is a good fit.

Code:



```
for j in range(20):
    list_samples=[]
    D=9
    chi_value=0
    No_of_samples=1000
    for value in range(No_of_samples):
        list_samples.extend(random.sample(range(10),1))
    for degree in range(D+1):
        chi_value += ((list_samples.count(degree) - No_of_samples / (D + 1)) ** 2) / (No_of_samples / (D + 1))
    print('The '+str(j+1)+' chi-squared value: ',chi_value)
pt.figure(figsize=(10,5))
pt.xlabel('sample values')
pt.ylabel('number of samples')
pt.hist(list_samples,bins=range(11),edgecolor='black')
pt.show()
```

Output:

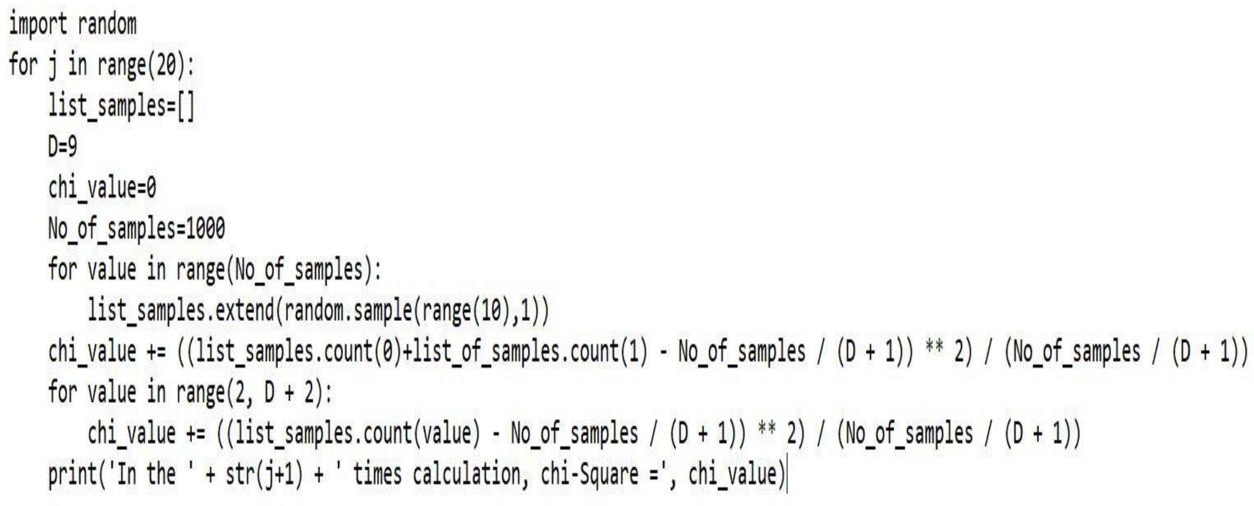


```
The 1 chi-squared value:  3.78
The 2 chi-squared value:  9.36
The 3 chi-squared value:  4.76
The 4 chi-squared value:  10.88
The 5 chi-squared value:  15.02
The 6 chi-squared value:  12.480000000000002
The 7 chi-squared value:  10.219999999999999
The 8 chi-squared value:  9.14
The 9 chi-squared value:  4.18
The 10 chi-squared value:  9.52
The 11 chi-squared value:  3.3000000000000003
The 12 chi-squared value:  10.0
The 13 chi-squared value:  5.86
The 14 chi-squared value:  6.68
The 15 chi-squared value:  2.9199999999999995
The 16 chi-squared value:  9.16
The 17 chi-squared value:  11.919999999999998
The 18 chi-squared value:  5.5600000000000005
The 19 chi-squared value:  5.92
The 20 chi-squared value:  7.96
```

(C).

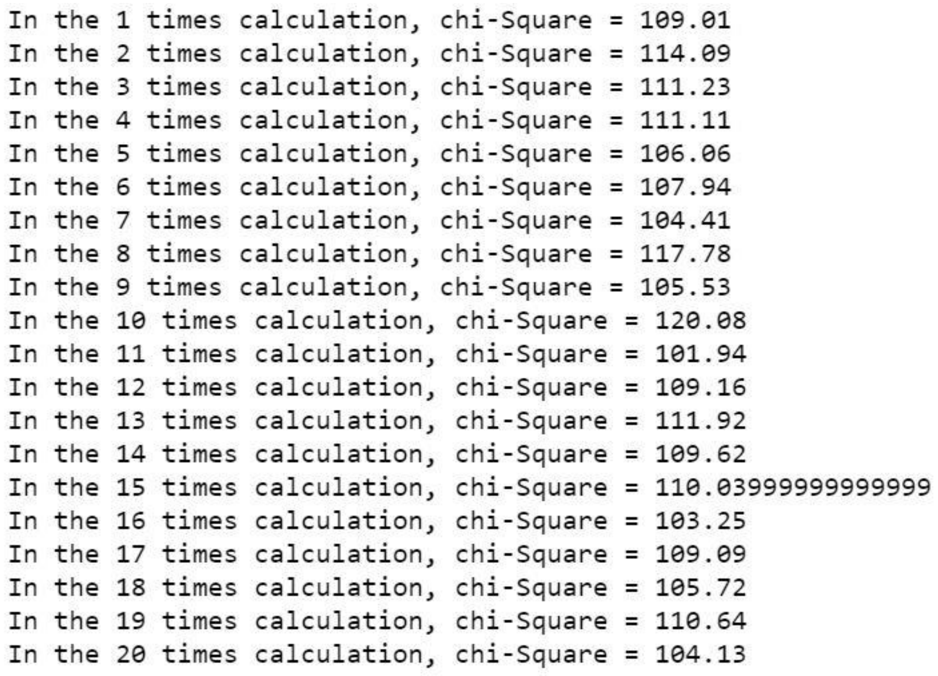
- Here the samples are generated using <random.sample()> function. The bins are set as [<=1,2,3,4,5,6,7,8,9].
- Here since the expected value of M=0, 0 and 1 are put in the first bin.
- For 95% confidence interval, expected value is 100, and degree freedom is 9. The calculated threshold value is 16.9160.
- If the obtained value is less than threshold value, then it is a good fit.
- Here all the values obtained are greater than the threshold, therefore it does not fit an alternate uniform distribution 1,2,3, ……..,10.

Code:



```
import random
for j in range(20):
    list_samples=[]
    D=9
    chi_value=0
    No_of_samples=1000
    for value in range(No_of_samples):
        list_samples.extend(random.sample(range(10),1))
    chi_value += ((list_samples.count(0)+list_of_samples.count(1) - No_of_samples / (D + 1)) ** 2) / (No_of_samples / (D + 1))
    for value in range(2, D + 2):
        chi_value += ((list_samples.count(value) - No_of_samples / (D + 1)) ** 2) / (No_of_samples / (D + 1))
    print('In the ' + str(j+1) + ' times calculation, chi-Square =', chi_value)
```

Output:



```
In the 1 times calculation, chi-Square = 109.01
In the 2 times calculation, chi-Square = 114.09
In the 3 times calculation, chi-Square = 111.23
In the 4 times calculation, chi-Square = 111.11
In the 5 times calculation, chi-Square = 106.06
In the 6 times calculation, chi-Square = 107.94
In the 7 times calculation, chi-Square = 104.41
In the 8 times calculation, chi-Square = 117.78
In the 9 times calculation, chi-Square = 105.53
In the 10 times calculation, chi-Square = 120.08
In the 11 times calculation, chi-Square = 101.94
In the 12 times calculation, chi-Square = 109.16
In the 13 times calculation, chi-Square = 111.92
In the 14 times calculation, chi-Square = 109.62
In the 15 times calculation, chi-Square = 110.03999999999999
In the 16 times calculation, chi-Square = 103.25
In the 17 times calculation, chi-Square = 109.09
In the 18 times calculation, chi-Square = 105.72
In the 19 times calculation, chi-Square = 110.64
In the 20 times calculation, chi-Square = 104.13
```