Project 4

Qidi Han

February 25, 2020

Question 1

Approximate the following integrals using a Monte Carlo simulation. Compare your estimates with the exact values (if known):

Theory

In this question, I will use Monte Carlo simulation to do the integral. Then I will use integrate.quad to do the theoretical result. I will explain the result in the explaining part. There are several equation that we mentioned in the class

Algorithm 1: Code for Monte Carlo simulation

/* Equation 1: Compute Monte Carlo */

sum =

$$\int_a^b g((1/y)-1)/y^2 dy$$

/* Equation 2: Compute multiple integrals by using Monte Carlo */

sum = E[g(u1,u2...un))]

/* code for experiment result */

for *i from* 0 *to* 10000 **do**

    u = np.random.rand();

    sum=sum+4*math.exp((4*u-2) + ((4*u-2)**2))

answer = sum/totalsample;

```
import numpy as np
import math
from scipy import integrate

total_sample=10000;
sum=0;

for i in range(0,total_sample):
    u = np.random.rand();
    sum=sum+4*math.exp((4*u-2) + ((4*u-2)**2))

answer = sum/total_sample;

invexp = lambda x: np.exp(x+x**2)
theo = integrate.quad(invexp, -2, 2)

print('Experiment result:',str(answer),'Theoretical result:',str(theo))
```

```
Experiment result: 93.6328413362832 Theoretical result: (93.16275329244199, 1.6178564393058
```

Figure 1: Code and results for part a

```
import numpy as np
import math
total_sample=10000;
sum=0;

for i in range(0,total_sample):
    u = np.random.rand();
    sum=sum+2*math.exp(-((1/u)-1)**2)*((1/u)**2)

answer = sum/total_sample;

invexp = lambda x: np.exp(-x**2)
theo = integrate.quad(invexp, -np.inf, np.inf)

print('Experiment result:',str(answer),'Theoretical result:',str(theo))
```

```
Experiment result: 1.7741771311742773 Theoretical result: (1.7724538509055159, 1.4202636781830878e-08)
```

Figure 2: Code and results for part b

```
import numpy as np
import math
total_sample=10000;
sum=0;

for i in range(0,total_sample):
    u = np.random.rand();
    u2 = np.random.rand();
    sum=sum+math.exp(-(u+u2)**2)

answer = sum/total_sample;

f = lambda y, x: np.exp(-(x+y)**2)
theo = integrate.dblquad(f, 0, 1, lambda x: 0, lambda x: 1)

print('Experiment result:',str(answer),'Theoretical result:',str(theo))
```

```
Experiment result: 0.4133790590656576 Theoretical result: (0.4117928941729141, 8.276155368153834e-15)
```

Figure 3: Code and results for part c

Explanation

For the part a, I make the random number u range from -2 to 2. Therefore, we have 4*u-2 for our y. Then I summed up all the number and divided by the number of sample. The experiment result is similar with our theoretical result. For part to I use the equation 1 proved above. Repeated the same process with part a. The experiment result is similar with the theoretical result. The part c use the equation 2 proved above. And do the same steps with last two examples. The experiment result is similar with the theoretical result. Therefore, the experiment is successful.

Question 2

Define the random variable X=Z1*Z1 + Z2*Z2+Z3*Z3+Z4*Z4 where Zk N(0,1). Then X x2(4)Generate 10 samples from X by first sampling Zi for i=2, 3, 4 and then computing X Plot the empirical distribution F10(x) for your samples and overlay the theoretical distribution F(x) Estimate a lower bound for ‖F10(x)-F(x) ‖computing the maximum difference at each of your samples: max ‖F10(x)-F(x) ‖.Then find the 25th, 50th, and 90th percentiles using your empirical distribution and compare the value to the theoretical percentile values for x2(4) Repeat the above using 100 and 1000 samples from X

Theory

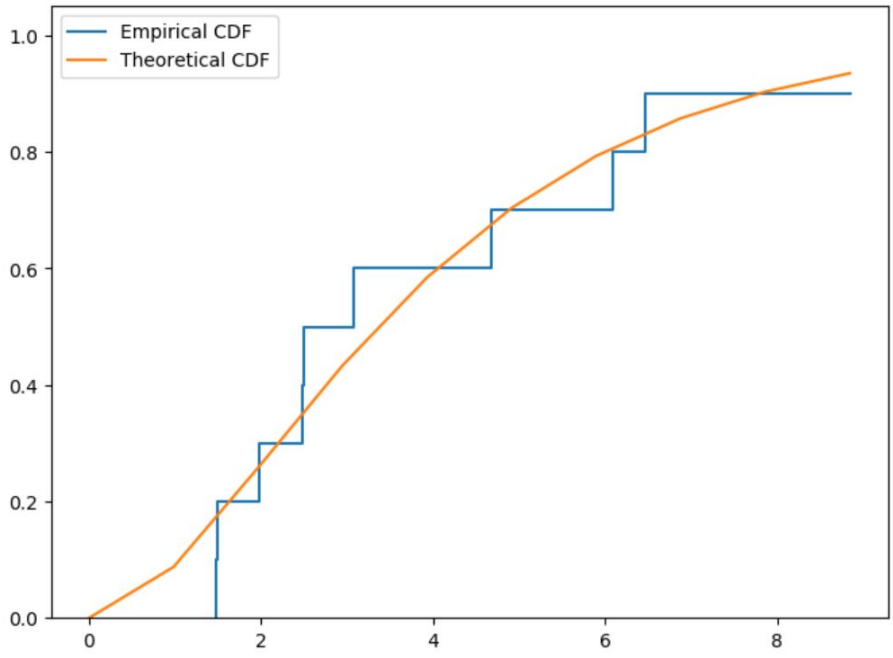
In this question, I will use randn() function to generate four numbers for Z1, Z2, Z3 and Z4 where Zk is normal distributed in range 0 to 1. Then I will sum them together to become X. I will generate 10, 100 and 1000 X to do three different experiment, Then I will plot the empirical distribution for my samples, and I will overlay the theoretical distribution F(x). For each CDF data point, I will compute the maximum different between the experiment and theoretical result. Finally, I will use np.percentile() function to find the 25th, 50th and 90th percentiles of my empirical distribution and theoretical result.

Algorithm 2: Code and formula for empirical distribution.

```
   /* Code for generate sample of empirical distribution                    */
 1 for i from 0 to 100 do
 2     z1= np.random.randn();
 3     z2= np.random.randn();
 4     z3= np.random.randn();
 5     z4= np.random.randn();
 6     x.append(z1**2+z2**2+z3**2+z4**2)
 7     x = np.sort(x, axis=None)
   /* Code for theoretical result                                           */
 8 t = np.linspace(0,max(x),n)
 9 y = chi2.cdf(t, 4) /* Plot for experiment and theoretical result        */
10 plt.step(x,np.arange(0,1,1/n), label='Empirical CDF')
11 plt.plot(t,y, label='Theoretical CDF')
```

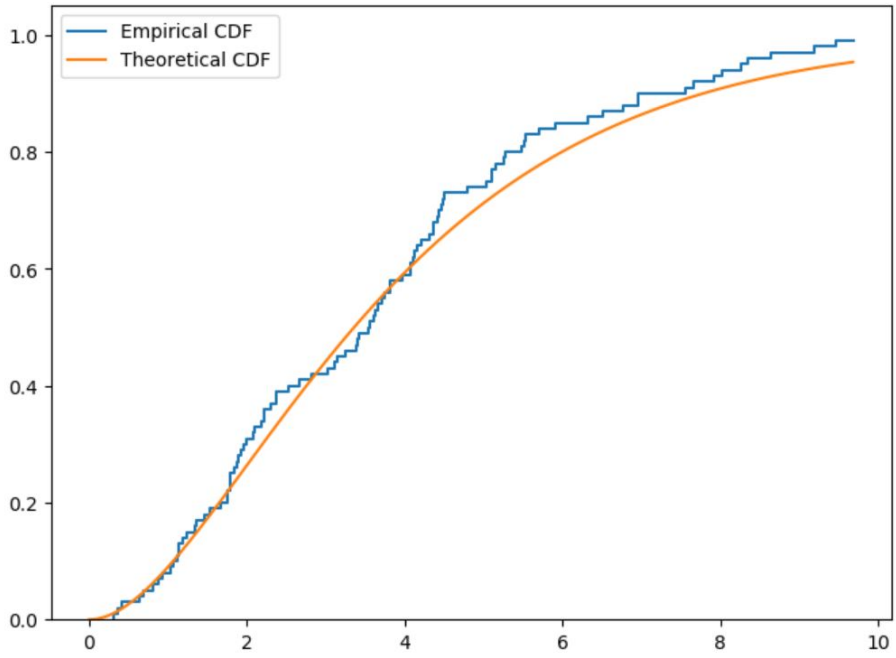
```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import chi2


def ecdf(data):
    x = np.sort(data)
    n = x.size
    y = np.arange(1, n+1) / n
    return(y)

n=1000;
x=[];

for ten_range in range(0,n):
    z1= np.random.randn();
    z2= np.random.randn();
    z3= np.random.randn();
    z4= np.random.randn();
    x.append(z1**2+z2**2+z3**2+z4**2)

x = np.sort(x, axis=None)
t = np.linspace(0,max(x),n)
y = chi2.cdf(t, 4)

fig = plt.figure(figsize=(8,6),dpi=100)
plt.step(x,np.arange(0,1,1/n), label='Empirical CDF')
plt.plot(t,y, label='Theoretical CDF')
plt.ylim((0,1.05))
plt.legend()
plt.show()
f=ecdf(x)
lower_bound_value=[];

y1 = np.zeros(n)
h=np.arange(0,1,1/n);

lower_bound = max(abs(h-chi2.cdf(x, 4)))
y_output =chi2.cdf(x, 4);

print('The lower bound is: ',str(lower_bound))
print('The 25% percentile empirical value is: ',str(np.percentile(f,25)))
print('The 50% percentile empirical value is: ',str(np.percentile(f,50)))
print('The 90% percentile empirical value is: ',str(np.percentile(f,90)))
print('The 25% percentile theoretical value is: ',str(np.percentile(y_output,25)))
print('The 50% percentile theoretical value is: ',str(np.percentile(y_output,50)))
print('The 90% percentile theoretical value is: ',str(np.percentile(y_output,90)))
```

Figure 4: Code for this empirical distribution



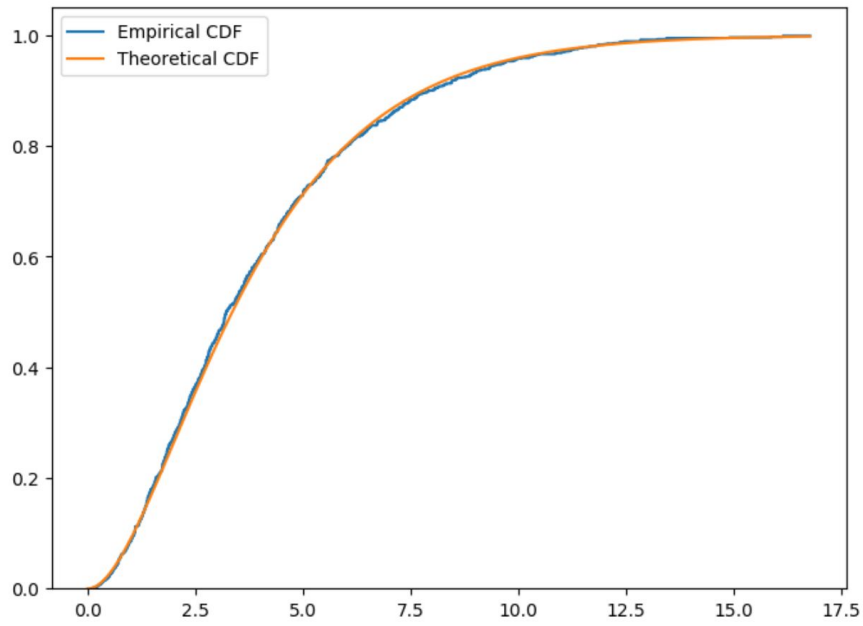

```
The lower bound is:  0.1683262926363713
The 25% percentile empirical value is:  0.32499999999999996
The 50% percentile empirical value is:  0.55
The 90% percentile empirical value is:  0.91
The 25% percentile theoretical value is:  0.2841910482547384
The 50% percentile theoretical value is:  0.40501471432669084
The 90% percentile theoretical value is:  0.8433270660388457
```

Figure 5: Plot and result for 10 samples experiment




```
The lower bound is:  0.06349970695809815
The 25% percentile empirical value is:  0.2575
The 50% percentile empirical value is:  0.505
The 90% percentile empirical value is:  0.9010000000000001
The 25% percentile theoretical value is:  0.23169847436228366
The 50% percentile theoretical value is:  0.5291372544317506
The 90% percentile theoretical value is:  0.8646253438135284
```

Figure 6: Plot and result for 100 samples experiment




```
The lower bound is:  0.02174528036767115
The 25% percentile empirical value is:  0.25075000000000003
The 50% percentile empirical value is:  0.5005
The 90% percentile empirical value is:  0.9001
The 25% percentile theoretical value is:  0.237141171210702
The 50% percentile theoretical value is:  0.4781144096404662
The 90% percentile theoretical value is:  0.9053291645007661
```

Figure 7: Plot and result for 1000 samples experiment

Explanation

From figure 5,6,7, we can see that as the samples number increasing, the theoretical result is overlay the empirical result. The 25th, 50th and 90th percentiles of empirical value is close to theoretical result when we have 1000 samples. Therefore, our experiment is correct. The lower bound for 1000 sample is 0.03617336964709206. I think 1000 sample give us the best result. The sample size 10 and 100 is not perfect as 1000 samples.

Question 3

. A geyser is a hotspring characterized by an intermittent discharge of water and steam. Old Faithful is a famous cone geyser in Yellowstone National Park, Wyoming. It has a predictable geothermal discharge and since 2000 it has erupted every 44 to 125 minutes. Refer to the addendum data file that contains waiting times and the durations for 272 eruptions. Compute a 95 percentage statistical confidence interval for the mean waiting time using data from only the first 15 eruptions. Compare this to a 95 percentage bootstrap confidence interval using the same 15 data samples. Repeat these calculations using all the data samples. Comment

on the relative width of the confidence intervals when using only 15 samples vs using all samples.

Theory

In this question, I will separate the data point to two different txt files. I will input the eruptions data into data1.txt file. I will input the waiting time data into data2.txt file. Then I will use open() function to read the data from the file to our array. Next, I will make a new array to store with first 15 data. I will compute the 95th statistical confidence interval for the mean waiting time by using first 15 data and all data. The way of doing the statistical confidence interval is compute standard error for the data first, then your this error do multiple with PDF by using stats.t.ppf() function. And using those two data to compute confidence interval. Finally, I will compute 95th bootstrap confidence interval for the mean waiting time using data first 15 data and all data. The way of doing this bootstrap confidence is that I will generate 272 random number and use this as index for the data points from the data file. I will record each mean for those task, I will repeat with 1000. Finally, I will sort them by using np.sort() function. In the explanation part, I will discuss the result and the relative width of confidence intervals for both data.

Algorithm 3: .

```
   /* Read data from txt file                                                    */
 1 file = open(data1.txt,r)
 2 file2 = open(data2.txt,r)
 3 a=95
 4 Repeat for each song in the text file
 5 eruptions =[];
 6 waiting=[];
 7 for line in file:
 8 eruptions.append(float(line))
 9 for line2 in file2:
10 waiting.append(float(line2))
11 file.close()
12 file2.close()
   /* Compute statistical confidence interval                                    */
13 standard_error = np.std(waiting)/np.sqrt(len(waiting))
14 scaling = stats.t.ppf(0.025, len(waiting)-1)
15 scaling2 = stats.t.ppf(0.975, len(waiting)-1)
16 confidence_low = np.mean(waiting) + standard_error * scaling;
17 confidence_high = np.mean(waiting) + standard_error *
     scaling2; /* Compute bootstrap confidence interval  */
18 y = np.zeros(1000)
19 for i from 0 to 1000 do
20 |   index=[];  all_number = [];
21 |   value = 272*np.random.rand(272,1);
22 |   for i from 0 to 272 do
23 |   index.append(math.ceil(value[j]))
24 |   for i from 0 to 272 do
25 |   ind = index[j]-1;
26 |   all_number.append(waiting[ind])
27 |   y[i] = np.mean(all_number);
28 |   y = np.sort(y);
```

[15]:
```
import numpy as np
from scipy import stats
from sklearn.utils import resample
import math
from random import random


file = open("data1.txt","r")
file2 = open("data2.txt","r")
a=95
#Repeat for each song in the text file
eruptions =[];
waiting=[];
for line in file:
    eruptions.append(float(line))
for line2 in file2:
    waiting.append(float(line2))
file.close()
file2.close()
#first 15 number
waiting_15 = waiting[:15]
eruptions_15 = eruptions[:15]
#statistical confidence interval of 15 samples
standard_error = np.std(waiting_15)/np.sqrt(len(waiting_15))
scaling =  stats.t.ppf(0.025, len(waiting_15)-1)
scaling2 =  stats.t.ppf(0.975, len(waiting_15)-1)
confidence_low = np.mean(waiting_15) + standard_error*scaling;
confidence_high = np.mean(waiting_15) + standard_error*scaling2;
print('The ',str((100-a)/2),'% percentile of 15 samples is: ',str(confidence_low))
print('The ',str((100+a)/2),'% percentile of 15 samples is: ',str(confidence_high))

#statistical confidence interval of all samples
standard_error = np.std(waiting)/np.sqrt(len(waiting))
scaling =  stats.t.ppf(0.025, len(waiting)-1)
scaling2 =  stats.t.ppf(0.975, len(waiting)-1)
confidence_low = np.mean(waiting) + standard_error*scaling;
confidence_high = np.mean(waiting) + standard_error*scaling2;
print('The ',str((100-a)/2),'% percentile of all samples is: ',str(confidence_low))
print('The ',str((100+a)/2),'% percentile of all samples is: ',str(confidence_high))

#Bootstrap confidence interval of 15 samples
y1 = np.zeros(1000)
for i in range(1,1000):
    index=[];
    all_number=[];
    value = 272*np.random.rand(15,1);
    for j in range(0,15):
        index.append(math.ceil(value[j]))
    for j in range(0,15):
        ind = index[j]-1;
        all_number.append(waiting[ind])
    y1[i] = np.mean(all_number);
y1 = np.sort(y1);
per_1 = np.percentile(X, (100-a)/2,interpolation='nearest')
per_2 = np.percentile(X, (100+a)/2,interpolation='nearest')
print('The ',str((100-a)/2),'% percentile of 15 samples is: ',str(y1[25]))
print('The ',str((100+a)/2),'% percentile of 15 samples is: ',str(y1[975]))

#Bootstrap confidence interval of all samples
y = np.zeros(1000)
for i in range(1,1000):
    index=[];
    all_number=[];
    value = 272*np.random.rand(272,1);
    for j in range(0,272):
        index.append(math.ceil(value[j]))
    for j in range(0,272):
        ind = index[j]-1;
        all_number.append(waiting[ind])
    y[i] = np.mean(all_number);
y = np.sort(y);
per_1 = np.percentile(X, (100-a)/2,interpolation='nearest')
per_2 = np.percentile(X, (100+a)/2,interpolation='nearest')
print('The ',str((100-a)/2),'% percentile of all samples is: ',str(y[25]))
print('The ',str((100+a)/2),'% percentile of all samples is: ',str(y[975]))
```

Figure 8: Code for this experiment

```
The  2.5 % percentile of 15 samples is:  62.841129828135855
The  97.5 % percentile of 15 samples is:  79.02553683853083
The  2.5 % percentile of all samples is:  69.2771667513403
The  97.5 % percentile of all samples is:  72.51695089571851
The  2.5 % percentile of 15 samples is:  64.13333333333334
The  97.5 % percentile of 15 samples is:  77.86666666666666
The  2.5 % percentile of all samples is:  69.31985294117646
The  97.5 % percentile of all samples is:  72.53308823529412
```

```
1:
```

Figure 9: Result for this experiment

Explanation

From the experiment, the relative width of confidence intervals is wider for first 15 data point [62.84 79.02] compare to all data point[69.277 72.277]. I think the reason is because the number of data point is too small, therefore, the data is not widely separated. The number of data point is too limited for compute the confidence interval when we only have 15 data points. The result for 95th statistical confidence interval is close to the 95th bootstrap confidence interval. I think this experiment is successful.