EE 511 Homework 4

Yijing Jiang
yijingji@usc.edu

Feburary 24, 2020

1 Question 1

1.1 Part a

Explanation:
We can translate the original integral to some kind of expectation form.
Here x is the uniform distribution on $[-2, 2]$ with $f(x) = \frac{1}{4}$. The function of x is e^{x+x^2}.
So the integral can seen as the expectation value of the function of x:

$$\int_{-2}^{2} e^{x+x^2} dx = 4 \int_{-2}^{2} e^{x+x^2} \frac{1}{4} dx = 4E[e^{x+x^2}] \tag{1}$$

Then using Monte Carlo simulation method:

- randomly generate x_i from uniform distribution $U[-2, 2]$
- compute $e^{x_i+x_i^2}$
- repeat above two steps for n times
- according to SLLN, for large n:

$$4E[e^{x+x^2}] \approx 4\frac{1}{n}\sum_{i=1}^{n} e^{x+x^2} \tag{2}$$

The exact value of this interval can be compute by using *scipy.integrate.quad()*.

Code:

```
import numpy as np
from scipy import integrate

#simulate the integral
N = 10000 #sample size
x = np.random.uniform(-2,2,N) #sample from U[-2,2]
g_x = np.zeros((N,))
```

```
for i in range (N):
    g_x[i] = np.exp( x[i] + x[i] * x[i] )
integral_simu = 4 * np.mean(g_x) #compute sample mean
print ("Estimate value: ",integral_simu)

#compute exact value
func = lambda x: np.exp( x + x * x )
integral_theo = integrate.quad(func, -2, 2)[0]
print("Exact value: ",integral_theo)
```

Results:

- **Estimate value: 91.37481855768252**
- **Exact value: 93.16275329244199**

The estimate value is close to the exact value, thus this estimation approach can be used to estimate the value of the interval.

1.2 Part b

Explanation:
Separate the integral into two parts:

$$\int_{-\infty}^{\infty} e^{-x^2} dx = \int_{-\infty}^{0} e^{-x^2} dx + \int_{0}^{\infty} e^{-x^2} dx \tag{3}$$

For the positive parts,
let $y = \frac{1}{1+x}$, $dy = y^2 dx$, $y \in (0, 1]$, thus

$$\begin{aligned} \int_{0}^{\infty} e^{-x^2} dx &= \int_{0}^{1} e^{-\frac{1}{y}-1^2} \frac{1}{y^2} dy & (4) \\ &= \int_{0}^{1} h(y) dy & (5) \\ &= E[h(y)] & (6) \end{aligned}$$

then using Monte Carlo simulation method:

- randomly generate y_i from uniform distribution $U[0, 1]$
- compute $h(y_i)$
- repeat above two steps for n times
- according to SLLN, for large n:

$$E[h(y)] \approx \frac{1}{n} \sum_{i=1}^{n} h(y_i) \tag{7}$$

For the negative parts,
let $y = \frac{1}{x-1}$, $dy = y^2 dx$, $y \in [-1, 0)$, thus

$$\begin{aligned} \int_0^{\infty} e^{-x^2} dx &= \int_{-1}^{0} e^{-\frac{1}{y}+1^2} \frac{1}{y^2} dy \quad (8) \\ &= \int_{-1}^{0} h(y) dy \quad (9) \\ &= E[h(y)] \quad (10) \end{aligned}$$

then using Monte Carlo simulation method:

- randomly generate y_i from uniform distribution $U[-1, 0]$
- compute $h(y_i)$
- repeat above two steps for n times
- according to SLLN, for large n:

$$E[h(y)] \approx \frac{1}{n} \sum_{i=1}^{n} h(y_i) \quad (11)$$

The estimation value is the sum of the positive(7) and the negative(11) part.
The exact value of this interval is $\sqrt{\pi}$. It can also be computed by using *scipy.integrate.quad()*.

Code:

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

N = 1000 #sample size

#positive part, y1~U[0,1]
y1 = np.random.uniform(0,1,N)
h_y1 = np.zeros((N,))
for i in range (N):
    y1i = y1[i]
    h_y1[i] = np.exp( -(1/y1i - 1)**2 ) * 1/(y1i**2)
positive = np.mean(h_y1)
#negative part, y2~U[-1,0]
y2 = np.random.uniform(-1,0,N)
h_y2 = np.zeros((N,))
for i in range (N):
    y2i = y2[i]
    h_y2[i] = np.exp( -(1/y2i + 1)**2 ) * 1/(y2i**2)
negative = np.mean(h_y2)
#total integral simulation
integral_simu = positive + negative
```

```
print ("Estimate value: ",integral_simu)

#Exact value
func = lambda x: np.exp(-x*x)
integral_theo = integrate.quad(func, -np.inf, np.inf)[0]
print("Exact value: ",integral_theo)
print("Theoretical value: ",math.sqrt(math.pi))
```

Results:

- **Estimate value: 1.7926703740147678**
- **Exact value: 1.7724538509055159**
- **Theoretical value: 1.7724538509055159**

The estimate value is close to the exact value, thus this estimation approach can be used to estimate the value of the interval.
Also the exact value is equal to the theoretical value, so to compute the exact value by using *scipy.integrate.quad()* is correct.

1.3 Part c

Explanation:
Random variable $X \sim U[0,1]$, $Y \sim U[0,1]$Regard this integral as the expectation of the function $g(x,y)$ on jointly distribution (x,y):

$$\int_0^1 \int_0^1 e^{-(x+y)^2} dydx = \int_0^1 \int_0^1 g(x,y)dydx = E[g(x,y)] \tag{12}$$

Then using Monte Carlo simulation method:

- randomly generate $x_i \sim U[0,1]$, $y_i \sim U[0,1]$
- compute $g(x,y)$
- repeat above two steps for n times
- according to SLLN, for large n:

$$E[g(x,y)] \approx \frac{1}{n}\sum_{i=1}^{n} g(x_i,y_i) \tag{13}$$

The exact value of this interval can be computed by using *scipy.integrate.dblquad()*.

Code:

```
import numpy as np
from scipy import integrate

N = 1000 #sample size

#simulate the integral
x = np.random.uniform(0,1,N)
y = np.random.uniform(0,1,N)
g_xy = np.zeros((N,))
for i in range (N):
    g_xy[i] = np.exp( -( x[i] + y[i] ) **2 )
integral_simu = np.mean(g_xy)
print ("Estimate value: ",integral_simu)

#compute exact value
func = lambda x, y: np.exp( -(x + y) **2)
integral_theo = integrate.dblquad(func, 0, 1, lambda y: 0, lambda y: 1)[0]
print("Exact value: ",integral_theo)
```

Results:

- **Estimate value: 0.4043011758654545**
- **Exact value: 0.4117928941729141**

The estimate value is close to the exact value, thus this estimation approach can be used to estimate the value of the interval.

2 Question 2

Random variable X is defined as $X = Z_1^2 + Z_2^2 + Z_3^2 + Z_4^2$, where $Z_i \sim N(0,1)$, $i = 1, 2, 3, 4$, Z_i are i.i.d. Thus $X \sim \chi^2(4)$.

Explanation:
(1) Get the samples of X. X is a chisquare random variable with 4 degrees:

- sample each $Z_i \sim N(0,1)$ for $i = 1, 2, 3, 4$
- compute one sample $X = Z_1^2 + Z_2^2 + Z_3^2 + Z_4^2$
- do the sampling for N times, N = 10, 100, 1000

(2) Plot the empirical distribution of $F_N^*(x)$. Empirical distribution of an i.i.d sequence of X_i is defined as:

$$F_N^*(x) = \frac{1}{n} \sum_{i=1}^{n} I_{[X_i,\infty)}(x) = P(X_i \leqslant x) \tag{14}$$

where I_C is the indicator function of set C. There are the steps to compute empirical distribution:

- sort the N samples in ascending orde: $x_{(1)} \leqslant x_{(2)} \leqslant ... \leqslant x_{(N)}$
- for each $x_{(i)}$, $F_N^*(x_{(i)}) = \frac{i}{n}$
- plot $x_{(i)}$ and $F_N^*(x_{(i)})$

(3) Plot the theoretical distribution $F(x)$:

- set x in range of [0,20], with $step = 10^{-3}$
- use *scipy.stats.chi2.cdf(x,df)* to get $F(x)$, where df = 4
- plot x and $F(x)$

(4) Estimate a lower bound for $\| F_N^*(x) - F(x) \|_\infty$, by computing $max \mid F_N^*(x_i) - F(x_i) \mid$. According to Glivenko–Cantelli theorem, $F_N^*(x)$ will converge to $F(x)$ pointwise by the SLLN, that is:

$$\| F_N^*(x) - F(x) \|_\infty \longrightarrow 0 \quad almost \quad surely \tag{15}$$

So we can estimate the degree of convergence by compute the maximum difference on each x_i.

(5) Find the 25th, 50th, 90th percentiles on empirical distribution and theoretical $\chi^2(4)$.

- use *numpy.percentile(x,q)* to get the percentiles on empirical distribution, where q is the percentile range $[0, 100]$

- use *scipy.stats.chi2.ppf(q,df)* to get the theretical value on each percentiles, where q is the percentile range $[0, 1]$, df is the *degree* $= 4$

Code:

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

N = 10 #sample number: 100, 1000

#get sample value of X
z1 = np.random.normal(0,1,N)
z2 = np.random.normal(0,1,N)
z3 = np.random.normal(0,1,N)
z4 = np.random.normal(0,1,N)
x = z1*z1 + z2*z2 + z3*z3 + z4*z4

#empirical distribution
x = np.sort(x, axis=None)
Fx_N = np.arange(0,1,1/N)

#theoretical distribution
x_theo = np.arange(0,20,10**(-3))
Fx_theo = chi2.cdf(x_theo, 4)

#compute the lower bound
Fx_theo_N = chi2.cdf(x, 4)
diff = abs(Fx_N - Fx_theo_N)
max = np.max(diff)
print("The maximum of difference is: ", max)

#compute the 25th, 50th, 90th percentile of empirical distribution
print("Percentiles from empirical distribution:")
print("25%: ", np.percentile(x, 25, interpolation = 'nearest'))
print("50%: ", np.percentile(x, 50, interpolation = 'nearest'))
print("90%: ", np.percentile(x, 90, interpolation = 'nearest'))
#compute the 25th, 50th, 90th percentile of theoretical distribution
print("Percentiles from theoretical distribution:")
print("25%: ", chi2.ppf(0.25, 4))
print("50%: ", chi2.ppf(0.50, 4))
print("90%: ", chi2.ppf(0.90, 4))
#ppf(q = percent, df = degree, loc=0, scale=1): Percent point function
    (inverse of cdf-percentiles).

#show the image
fig = plt.figure(figsize=(8,6),dpi=100)
plt.step(x,Fx_N, label='Empirical CDF')
plt.plot(x_theo, Fx_theo, label='Theoretical CDF')
```

```
plt.ylim((0,1.05))
plt.xlabel("X")
plt.ylabel("F(X)")
plt.legend()
plt.show()
```

Results:

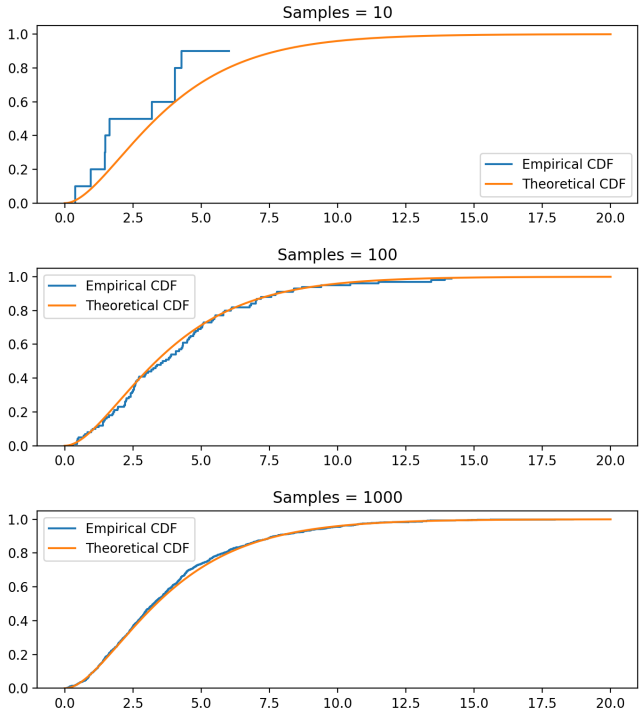

Figure 1: cdf of empirical and theoretical distribution on 10,100,1000 samples. (x,y) = (x, $F(x)$)

- **Sample 10 times:**
  The maximum of difference is: 0.2022
  25th,50th,90th percentiles: empirical / theoretical
  25th: 1.4763 / 1.9225
  50th: 1.6363 / 3.3566
  90th: 4.2865 / 7.7794

- **Sample 100 times:**
  The maximum of difference is: 0.0686
  25th,50th,90th percentiles: empirical / theoretical
  25th: 2.2152 / 1.9225
  50th: 3.7042 / 3.3566
  90th: 7.7301 / 7.7794

- **Sample 1000 times: The maximum of difference is: 0.0370**
  25th,50th,90th percentiles: empirical / theoretical
  25th: 1.9122 / 1.9225
  50th: 3.2609 / 3.3566
  90th: 7.6695 / 7.7794

As sample numbers increasing, the convergence of $F_N^*(x)$ to $F(x)$ almost surely can be shown in the following:

- Most intuitive, in the image, the plot of $F_N^*(x)$ is much similar to $F(x)$.

- Maximum difference $\mid F_N^*(x_i) - F(x_i) \mid$ is decreasing and much closer to 0, which means the lower bound for $\parallel F_N^*(x) - F(x) \parallel_\infty$ will converge to 0.

- Percentiles of emirical distribution are much closer to the theretical distribution, which also shows the two distributions will be much similar as n growing.

3 Question 3

Explanation:
(1) Statistical confidence interval of 95%, with unknown mean and unknown standard deviation:

- compute the sample mean μ
- compute the sample variance s^2, standart deviation s
- if samples $n < 50$, use t-distribution to get the confidence interval:

$$\mu \pm \frac{s}{\sqrt{n}} t_{\frac{\alpha}{2}} \tag{16}$$

- if samples $n \geq 50$, use z-distribution to get the confidence interval: $Z \sim N(0, 1)$

$$\mu \pm \frac{s}{\sqrt{n}} z_{\frac{\alpha}{2}} \tag{17}$$

(2) Bootstrap confidence interval of 95%:

- resample n samples with replacement from the original n samples, compute sample mean of the new n samples.
- repeate the first step 10000 times, get a sequence of sample mean: $\mu_1, \mu_2, ..., \mu_{10000}$
- confidence interval = $(\mu_{2.5}, \mu_{97.5})$

Code:

```
import numpy as np
from scipy import stats
from sklearn.utils import resample

#read data from a txt file (only contain the datas of eruptions: 272*3)
data = np.loadtxt('/Users/mac-pro/Desktop/20SPRING/511/project
    4/code/data.txt')

#get how many erptions, or 272
N = 15
wait_time = data[0:N, 2]
ci = 95

#statistical CI
mean_s = np.mean(wait_time)
print("----sample number = {}----".format(N))
print("The population mean is:", mean_s)
std_s = np.std(wait_time)
t_ci = stats.t.ppf((100+ci)/200, N-1) #get t(0.975), degree=14/271
E_s = std_s * t_ci / np.sqrt(N)
```

```
ci_s = np.array([mean_s - E_s, mean_s + E_s])
width_s = 2 * E_s
print("Statistical 95% C.I. of ", N," eruptions is: ", ci_s, width_s)
test_s = 'bad'
if mean_s >= ci_s[0] and mean_s <= ci_s[1]: test_s = 'good'
print("This C.I is ", test_s, ".")

#bootstrap CI
R = 10000 #resample times
mean_resample = np.zeros((R,))
for i in range (R):
    wait_resample = resample(wait_time, n_samples = N, replace = True)
    mean_resample[i] = np.mean(wait_resample)
ci_l = np.percentile(mean_resample, (100-ci)/2, interpolation = 'nearest')
ci_h = np.percentile(mean_resample, (100+ci)/2, interpolation = 'nearest')
ci_b = np.array([ci_l, ci_h])
width_b = ci_h - ci_l
print("Bootstrap 95% C.I. of", N, " eruptions is: ", ci_b, width_b)
test_b = 'bad'
if mean_s >= ci_b[0] and mean_s <= ci_b[1]: test_b = 'good'
print("This C.I is ", test_b, ".")
```

Results:

- **Sample number = 15**
  The population mean: 70.9333
  Statistical 95% C.I. and width: [62.8411 79.0255] 16.1844
  This C.I is good .
  Bootstrap 95% C.I. and width: [63.4666 78.1333] 14.6666
  This C.I is good .

- **Sample number = 272**
  The population mean: 70.8970
  Statistical 95% C.I. and width: [69.2771 72.5169] 3.2397
  This C.I is good .
  Bootstrap 95% C.I. and width: [69.2316 72.4926] 3.2610
  This C.I is good .

- All the confidence interval above are good(all include the population mean).

- When using the first 15 eruptions, the width of confidence interval on bootstrap method is slightly smaller than on statistical method, without much difference.

- When using all the 272 eruptions(more datas than 15 eruptions), the width of confidence interval is much smaller than using first 15 eruptions. And the confidence interval of statistical and the bootstrap method is close to each other.

Therefore, the width of confidence interval is somehow related to the original number of samples. When the sample number is increasing, the width of confidence interval

will be decreasing. With more datas, the confidence interval region will be smaller, which means the datas are more concentrated to some region. The difference between statistical method and bootstrap method will not be too large.

4 Reference

- https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.quad.html
- https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.dblquad.html
- https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.chi2.html
- https://en.wikipedia.org/wiki/Glivenko%E2%80%93Cantelli_theorem
- https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.t.html