# Project #4 – Integrals and Intervals


EE 511: Fall 2019

Qiong Wang    Id: 5906740674


Tool: PyCharm

## 1. Question 1-a

**Requirement: Simulate $\int_{-2}^{2} e^{x+x^2}\,dx$.**

**Code:**

```
import numpy as np


result = []
for i in range(10000):
    u = np.random.uniform()
    result.append(4*np.exp(16*(u**2)-12*u+2))
print(np.mean(result))
```

**Result:**

As we all know, the theoretical result is 93.1628, and I use Monte Carlo method to simulate this integral 20 times, and got 20 different result, I compare the experiment result with theoretical value. 20 different result were shown below, the maximum experiment error is 3.22% when the simulation value is 16.169, so the Monte Carlo method is good to simulate the result of integral.

```
/Users/qiongwang/PycharmProjects/511/pr
simulation result = 94.07843359395714
simulation result = 90.0431808565173
simulation result = 91.65713282690692
simulation result = 90.68857002838952
simulation result = 92.9648837248866
simulation result = 92.83611438388306
simulation result = 96.169434375346
simulation result = 91.57582021821193
simulation result = 90.76637792319664
simulation result = 90.76748804344645
simulation result = 95.91420394418981
simulation result = 92.99606725682173
simulation result = 91.79787423834686
simulation result = 92.40415341760853
simulation result = 94.20585779234263
simulation result = 91.29450117690646
simulation result = 91.08312602813034
simulation result = 93.38043088234923
simulation result = 94.9095645875663
simulation result = 90.50422197960584

Process finished with exit code 0
```

## 2. Question 1-b

**Requirement: Simulate $\int_{-\infty}^{+\infty} e^{-x^2} dx$.**

**Code:**

```
import numpy as np

ans = []
for times in range(20):
    result = []
    for i in range(10000):
        u = np.random.uniform()
        result.append(2*(np.exp(-(1/u-1)**2)/u**2))
```

```
    print('simulation result =', np.mean(result))
    ans.append(np.mean(result))

print('maximum value =', max(ans))
print('minimum value =', min(ans))
print('mean of value =', np.mean(ans))
print('variance of value =', np.var(ans))
```

**Result:**

As we all know, the theoretical result is 1.7725, and I use Monte Carlo method to simulate this integral 20 times, and got 20 different result, I recorded the maximum value of the result, the minimum value of the result, the mean of 20 result, and the variance of this 20 value, I compare the experiment result with theoretical value. 20 different result were shown below, the maximum experiment error is 1.6% when the simulation value is 1.744, and the variance is very small, so the Monte Carlo method is good to simulate the result of integral.

maximum value = 1.7919990935515493

minimum value = 1.7440729092585203

mean of value = 1.768549100812382

variance of value = 0.00018586313123015015

```
/Users/qiongwang/PycharmProjects/511/project4
simulation result = 1.777746870219599
simulation result = 1.7777549212963792
simulation result = 1.7609619118789415
simulation result = 1.7615837057413541
simulation result = 1.772530154703337
simulation result = 1.75646772313382
simulation result = 1.7701626248869307
simulation result = 1.7919990935515493
simulation result = 1.78418281238801
simulation result = 1.7512184807935534
simulation result = 1.7798325552030732
simulation result = 1.7541135242874233
simulation result = 1.7878034511699226
simulation result = 1.7617317668460892
simulation result = 1.7440729092585203
simulation result = 1.7444798402279507
simulation result = 1.7745249625731845
simulation result = 1.7659425835513156
simulation result = 1.771094473371658
simulation result = 1.7827776511650293
maximum value = 1.7919990935515493
minimum value = 1.7440729092585203
mean of value = 1.768549100812382
variance of value = 0.00018586313123015015

Process finished with exit code 0
```

## 3. Question 1-c

**Requirement: Simulate** $\int_0^1 \int_0^1 e^{-(x+y)^2} dydx$

**Code:**

```python
import numpy as np

ans = []
for times in range(20):
```

```
    result = []
    for i in range(10000):
        x = np.random.uniform()
        y = np.random.uniform()
        result.append(np.exp(-(x+y)**2))
    print('simulation result =', np.mean(result))
    ans.append(np.mean(result))

print('maximum value =', max(ans))
print('minimum value =', min(ans))
print('mean of value =', np.mean(ans))
print('variance of value =', np.var(ans))
```

**Result:**

As we all know, the theoretical result is 0.4118, and I use Monte Carlo method to simulate this integral 20 times, and got 20 different result, I recorded the maximum value of the result, the minimum value of the result, the mean of 20 result, and the variance of this 20 value, I compare the experiment result with theoretical value. 20 different result were shown below, the maximum experiment error is 1.2% when the simulation value is 0.407, and the variance is very small, so the Monte Carlo method is good to simulate the result of integral.

maximum value = 0.41540930511375906

minimum value = 0.4070998054058483

mean of value = 0.41135324529975204

variance of value = 6.344008254757384e-06

```
/Users/qiongwang/PycharmProjects/511/project4/
simulation result = 0.4104897222656736
simulation result = 0.4082443213900286
simulation result = 0.4133299006655634
simulation result = 0.40790237471674556
simulation result = 0.4079673485300047
simulation result = 0.4129338623639669
simulation result = 0.4118253304244915
simulation result = 0.4102368272735853
simulation result = 0.4097776276988812
simulation result = 0.4070998054058483
simulation result = 0.41066071233812773
simulation result = 0.41246787420513464
simulation result = 0.41406625738079184
simulation result = 0.41230788584921346
simulation result = 0.40750637085009744
simulation result = 0.41368292980082644
simulation result = 0.4134397444704661
simulation result = 0.41459023971845854
simulation result = 0.4131264655333773
simulation result = 0.41540930511375906
maximum value = 0.41540930511375906
minimum value = 0.4070998054058483
mean of value = 0.41135324529975204
variance of value = 6.344008254757384e-06

Process finished with exit code 0
```

## 4. Question 2

**Analysis:** In this question, I simulate the empirical distribution of $\chi^2(4)$, and compare the result with theoretical result. When I simulate $\chi^2(4)$, I design 4 identical independent distribution random variable, $Z_1$ $Z_2$ $Z_3$ $Z_4$. For each Z ~ N (0, 1). Then I sum these four variables as a new random variable x. In the end, I plot the empirical distribution and the theoretical CDF distribution of $\chi^2(4)$ in the same figure and compare the result.

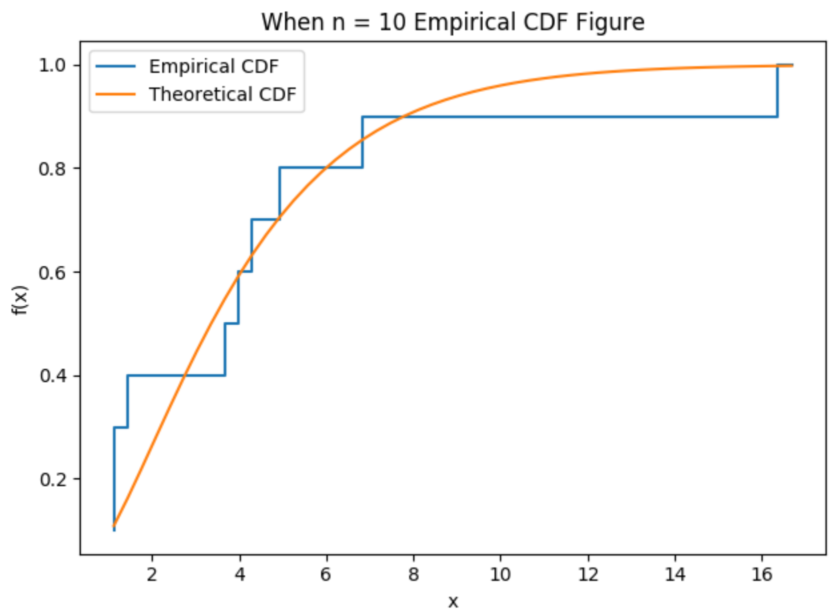
**Code:**

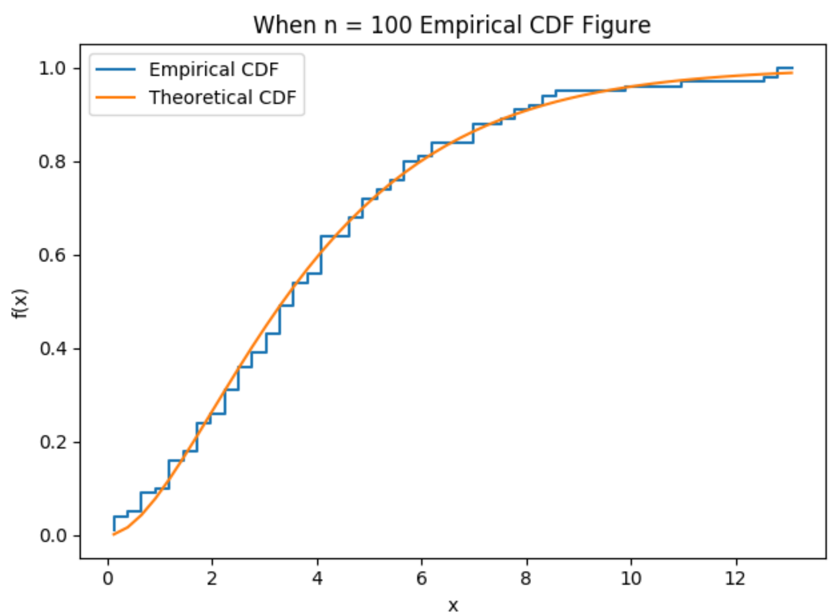
```
import numpy as np
import scipy.stats as stats
import statsmodels.api as sm
import matplotlib.pyplot as plt


for n in [10, 100, 1000]:
    sample = []
    for i in range(n):
        z1 = np.random.standard_normal()
        z2 = np.random.standard_normal()
        z3 = np.random.standard_normal()
        z4 = np.random.standard_normal()
        sample.append(z1 ** 2 + z2 ** 2 + z3 ** 2 + z4 ** 2)
    x_cdf = sm.distributions.ECDF(sample)
    x_line = np.linspace(min(sample), max(sample))
    cdf = stats.chi2.cdf(x_line, df=4)
    plt.figure()
    plt.title('When n = ' + str(n) + ' Empirical CDF Figure')
    plt.step(x_line, x_cdf(x_line), label='Empirical CDF')
    plt.plot(x_line, cdf, label='Theoretical CDF')
    plt.legend()
    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.show()

    print('When n =', n)
    print('maximum difference =', max([abs(list(cdf.data)[k]-
x_cdf(x_line)[k]) for k in range(len(x_cdf(x_line)))]))
    print('Empirical 25% percentiles =', np.percentile(x_cdf(x_line),
25))
    print('Theoretical 25% percentiles =',
np.percentile(list(cdf.data), 25))
    print('Empirical 50% percentiles =', np.percentile(x_cdf(x_line),
50))
    print('Theoretical 50% percentiles =',
np.percentile(list(cdf.data), 50))
    print('Empirical 90% percentiles =', np.percentile(x_cdf(x_line),
90))
    print('Theoretical 90% percentiles =',
```

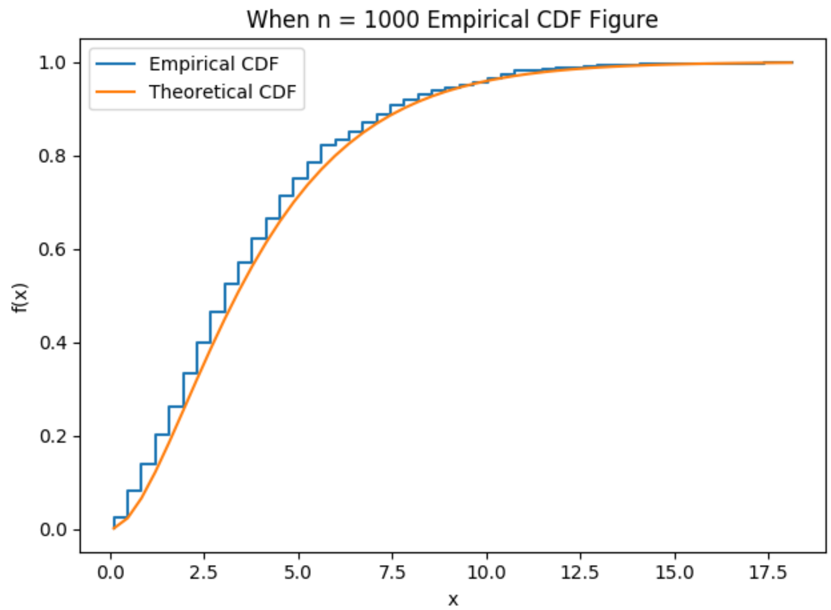
```
np.percentile(list(cdf.data), 90))
    print('')
```

**Result:** I totally simulate 3 times in the question, I change the sample size of x, and print all statistical result as below. Then I plot the figure of empirical distribution of x and the theoretical CDF distribution of $\chi^2(4)$ in the same figure. From the picture, it is obviously to see that, with the amount of n increased, the empirical distribution of x will converge to the theoretical CDF distribution of $\chi^2(4)$, so it is a good simulation.

```
/Users/qiongwang/PycharmProjects/511/project4/venv/bi
When n = 10
maximum difference = 0.180487929605313
Empirical 25% percentiles = 0.7250000000000001
Theoretical 25% percentiles = 0.7135793583986807
Empirical 50% percentiles = 0.9
Theoretical 50% percentiles = 0.9362693757306335
Empirical 90% percentiles = 0.9
Theoretical 90% percentiles = 0.9955498882100294

When n = 100
maximum difference = 0.060635033392930804
Empirical 25% percentiles = 0.445
Theoretical 25% percentiles = 0.500821717243552
Empirical 50% percentiles = 0.8400000000000001
Theoretical 50% percentiles = 0.8410779873277969
Empirical 90% percentiles = 0.97
Theoretical 90% percentiles = 0.9808664124835854

When n = 1000
maximum difference = 0.022517052959192707
Empirical 25% percentiles = 0.6765
Theoretical 25% percentiles = 0.6690156597915606
Empirical 50% percentiles = 0.9430000000000001
Theoretical 50% percentiles = 0.9413892062768086
Empirical 90% percentiles = 0.997
Theoretical 90% percentiles = 0.9973753334379253


Process finished with exit code 0
```


When n = 10 Empirical CDF Figure
Empirical CDF
Theoretical CDF
f(x)
x
1.0
0.8
0.6
0.4
0.2
2
4
6
8
10
12
14
16


When n = 100 Empirical CDF Figure
Empirical CDF
Theoretical CDF
f(x)
x
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
12




## 5. Question 3

**Analysis:** In this question, I read the waiting data set first, and store all data in a list. Then, I design 2 functions to calculate the data. The first one is to compute a 95% statistical confidence interval for the mean waiting time, $Z_{0.025}$ = 1.96, so, the lower bound is $mean - Z_{0.025} \times \frac{std}{\sqrt{n}}$, the upper bound is $mean + Z_{0.025} \times \frac{std}{\sqrt{n}}$. I calculate the sample mean and variance and according to the size of data I can get the upper bound and. Lower bound. The second method is to calculate 95% bootstrap confidence interval, in this function I will resample 10000 times, for each times, I calculate and store the sample mean, and then, I sorted the sample mean by non-decreasing order, and find out the 250$^{th}$ and 9750$^{th}$ value.

**Code:**

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

```python
def main():
    waiting = read_txt('faithful.dat.txt')

    run_15_data(waiting[:15])

    run_all_data(waiting)


def run_15_data(data_set):
    print('95% statistical confidence for 15 data:')
    lower, upper = statistical(data_set)
    print('lower bound:', lower)
    print('upper bound:', upper)
    print('')

    print('95% bootstrap confidence for 15 data:')
    lower, upper = bootstrap(data_set)
    print('lower bound:', lower)
    print('upper bound:', upper)
    print('')


def run_all_data(data_set):
    print('95% statistical confidence for all data:')
    lower, upper = statistical(data_set)
    print('lower bound:', lower)
    print('upper bound:', upper)
    print('')

    print('95% bootstrap confidence for all data:')
    lower, upper = bootstrap(data_set)
    print('lower bound:', lower)
    print('upper bound:', upper)
    print('')


def statistical(data_set):
    mean = np.mean(data_set)
    std = np.std(data_set, ddof=1)
    z = 1.96
```

```
    return [mean-z*(std/math.sqrt(len(data_set))),
mean+z*(std/math.sqrt(len(data_set)))]


def bootstrap(data_set):
    length = len(data_set)
    result = []
    for i in range(10000):
        candidate = []
        for j in range(length):
            candidate.append(random.sample(data_set, 1))
        result.append(np.mean(candidate))
    return [np.percentile(result, 2.5), np.percentile(result, 97.5)]


def read_txt(address):
    with open(address, 'r') as f:
        data = f.readlines()

    line = 0
    while data[line][0] != '1':
        line += 1

    waiting = []

    for i in range(line, line + 272):
        waiting.append(int(data[i].split()[-1]))

    return waiting


if __name__ == '__main__':
    main()
```

**Result:** From the result, I find that the result of bootstrap confidence interval is very similar to statistical confidence interval. The difference between the two value is very small. However, the result for using first 15 data have a big difference with whole data set. So, there exit some deviation when I use only a small part of the data.

```
/Users/qiongwang/PycharmProjects/511/proje
95% statistical confidence for 15 data:
lower bound: 63.278770830413194
upper bound: 78.58789583625348

95% bootstrap confidence for 15 data:
lower bound: 63.531666666666666
upper bound: 78.13333333333334

95% statistical confidence for all data:
lower bound: 69.28139874542947
upper bound: 72.51271890162934

95% bootstrap confidence for all data:
lower bound: 69.26470588235294
upper bound: 72.42656249999999


Process finished with exit code 0
```