hw13

January 6, 2021

1 Q1

1.1 Following the example shown in the lecture slides, computing the value of π using Monte Carlo methods. Then evaluate the effectiveness of bounds generated by inequalities.

1.1.1 we will use python to compute π and show the simulation case

[1]:
```
import random as rd
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
from math import *
from scipy import stats
```

[2]:
```
result = np.array([])

In,Total = 0 , 100000
for i in range(0,5000):
    In = 0
    for j in range(0,Total):
        x , y = rd.uniform(-1,1) , rd.uniform(-1,1)
        if sqrt(x**2+y**2) <= 1 :
            In += 4
    result = np.append(result,In/Total)
print(result.mean())
```

```
3.141556984
```

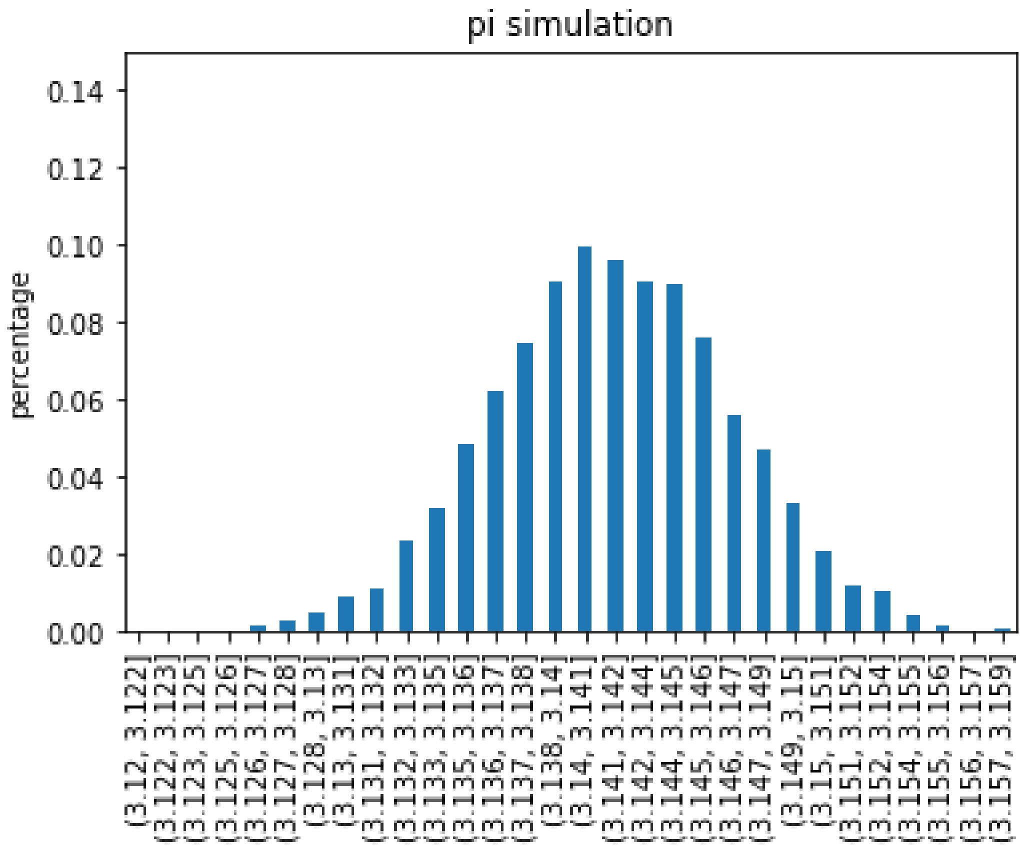
[3]:
```
p = pd.DataFrame({'res':result,'std_error':np.std(result, ddof=1) / np.
 ↪sqrt(len(result))})
counts = p['res'].value_counts(bins = 30,sort = False , normalize = True)
print(counts)
```

```
(3.12, 3.122]     0.0004
(3.122, 3.123]    0.0000
(3.123, 3.125]    0.0004
```

```
(3.125, 3.126]    0.0002
(3.126, 3.127]    0.0014
(3.127, 3.128]    0.0032
(3.128, 3.13]     0.0050
(3.13, 3.131]     0.0090
(3.131, 3.132]    0.0114
(3.132, 3.133]    0.0234
(3.133, 3.135]    0.0318
(3.135, 3.136]    0.0482
(3.136, 3.137]    0.0622
(3.137, 3.138]    0.0744
(3.138, 3.14]     0.0904
(3.14, 3.141]     0.0994
(3.141, 3.142]    0.0964
(3.142, 3.144]    0.0904
(3.144, 3.145]    0.0898
(3.145, 3.146]    0.0758
(3.146, 3.147]    0.0560
(3.147, 3.149]    0.0468
(3.149, 3.15]     0.0332
(3.15, 3.151]     0.0206
(3.151, 3.152]    0.0118
(3.152, 3.154]    0.0108
(3.154, 3.155]    0.0044
(3.155, 3.156]    0.0018
(3.156, 3.157]    0.0004
(3.157, 3.159]    0.0010
Name: res, dtype: float64
```

[4]:
```
counts.plot(kind = 'bar',ylim = (0,0.15),title='pi␣
↪simulation',ylabel='percentage')
```

[4]:
```
<AxesSubplot:title={'center':'pi simulation'}, ylabel='percentage'>
```



Step 1: Z_i : indication of the i^{th} point chosen uniformly landing and $P(Z_i = 1) = \frac{\pi}{4}$, Z_i are i.i.d. ,$E(Z_i) = \frac{\pi}{4} = \mu$ where $0 \leq Z_i \leq 1$

Step 2:we run the experiment for $n = 100000$ times,$w = \frac{1}{n}\sum_{i=1}^{n} Z_i$, $E(w) = \mu = \frac{\pi}{4}$, Hence $\hat{\pi} = 4w$ is a estimation of π

Step 3:$P(\mid \hat{\pi} - \pi \mid \geq \varepsilon) = P\left(\left|w - \frac{\pi}{4}\right| \geq \frac{\varepsilon}{4}\right) = P\left(\left|\frac{1}{n}\sum_{i=1}^{n} Z_i - \mu\right| \geq \frac{\varepsilon}{4}\right) \leq 2e^{\frac{-2n(\frac{\varepsilon}{4})^2}{(1-0)^2}} = 2e^{-\frac{n\varepsilon^2}{8}}$ By Hoeffding Bound

Step 4:Let $\delta = 2e^{-\frac{1}{8}n\varepsilon^2}$, Hence $\varepsilon = \sqrt{\frac{8\ln\frac{2}{\delta}}{n}}$

Step 5: Let $n = 100000$ and $\delta = 0.05$

[5]:
```
epsilon = sqrt(8*log(2/0.05)/100000)
print(epsilon)
```

```
0.0171787763338695
```

[6]:
```
mean = result.mean()
pr = 1 - (sum((result<pi-epsilon)+(result>pi+epsilon))/8000)
```

```
print(pr)
print(pr>1-0.05)
```

```
0.999625
True
```

In short, Monte Carlo methods will estimate π well.

2 Q2

2.1 Please generate samples of random variables satisfying standard Normal distribution by both Box-Muller method and the Acceptance-Rejection Method.Discuss the pros and cons of both methods.

2.1.1 First, we will use Box-Muller method to generate samples of random variables to satisfy standard Normal distribution.

[7]:
```
U1,U2 = np.random.random(100000),np.random.random(100000)#U1 U2
X = np.cos(2*pi*U1)*np.sqrt(-2*np.log(U2))
Y = np.sin(2*pi*U1)*np.sqrt(-2*np.log(U2))
Data = pd.DataFrame({'X':X,'Y':Y,'U1':U1,'U2':U2})
counts_U1 = Data['U1'].value_counts(bins = 30,sort = False , normalize = True)
counts_U2 = Data['U2'].value_counts(bins = 30,sort = False , normalize = True)
counts_X = Data['X'].value_counts(bins = 30,sort = False , normalize = True)
counts_Y = Data['Y'].value_counts(bins = 30,sort = False , normalize = True)
```

[8]:
```
print(Data['X'])
```

```
0         0.184640
1         0.016962
2         0.196154
3        -0.813094
4         1.213821
           …
99995    -1.620485
99996     0.351321
99997    -0.759046
99998    -1.526640
99999     0.686683
Name: X, Length: 100000, dtype: float64
```

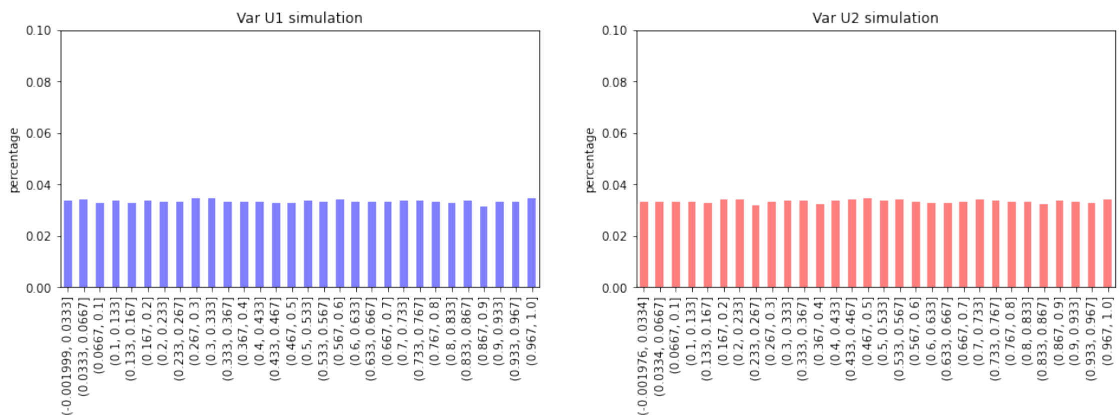
[9]:
```
print(Data['Y'])
```

```
0         1.090765
1        -0.059573
2        -1.818510
3        -0.204970
4         0.069421
```

```
            …
99995   -1.505420
99996   -0.599688
99997    0.237789
99998   -0.174864
99999   -2.258668
Name: Y, Length: 100000, dtype: float64
```

[10]:
```
fig1, ax1 = plt.subplots(1,2,figsize=(16,4))
counts_U1.plot(ax=ax1[0], color='b', alpha=0.5,kind = 'bar',ylim = (0,0.
↪1),title='Var U1 simulation',ylabel='percentage')
counts_U2.plot(ax=ax1[1], color='r', alpha=0.5,kind = 'bar',ylim = (0,0.
↪1),title='Var U2 simulation',ylabel='percentage')
```

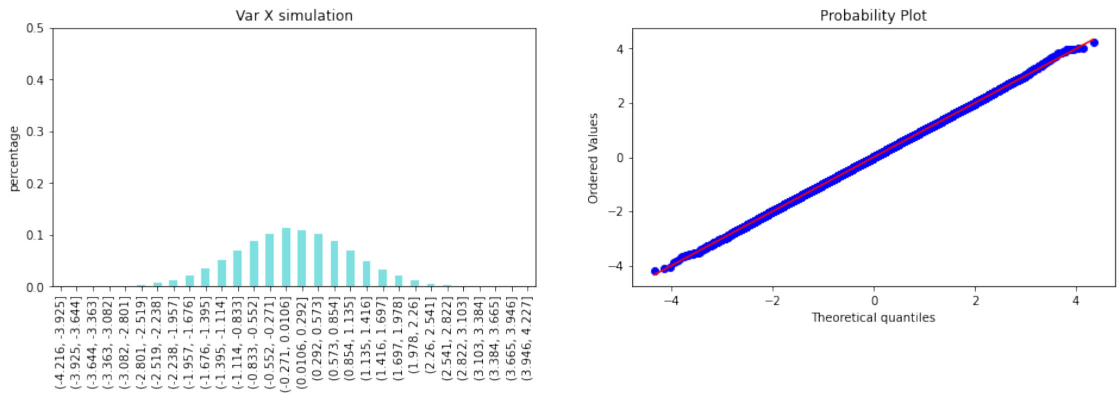
[10]:
```
<AxesSubplot:title={'center':'Var U2 simulation'}, ylabel='percentage'>
```



We can determine that U1 and U2 are uniformly distributed .(Actually, random function will produce the random vars which are uniformly distributed)

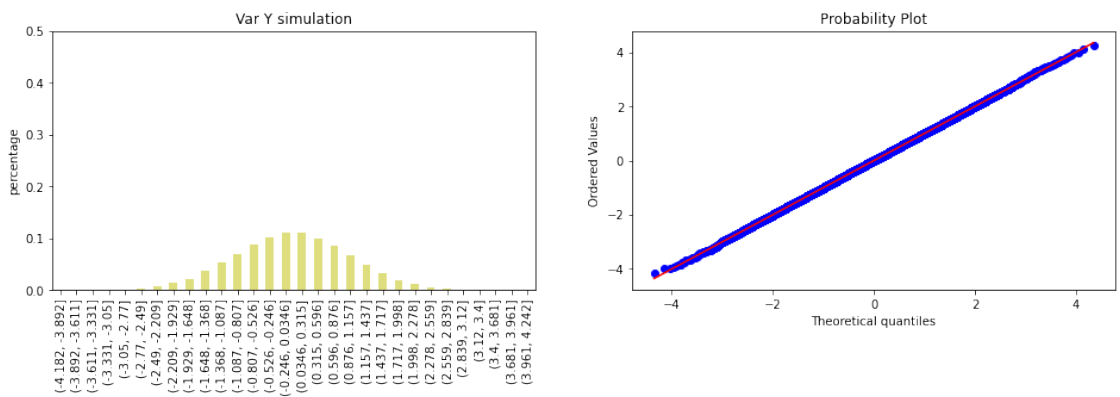
[11]:
```
fig_X, ax_X = plt.subplots(1,2,figsize=(16,4))
counts_X.plot(ax=ax_X[0], color='c', alpha=0.5,kind = 'bar',ylim = (0,0.
↪5),title='Var X simulation',ylabel='percentage')
stats.probplot(Data['X'], dist="norm", plot=plt)
```

[11]:
```
((array([-4.34602155, -4.14724578, -4.03913009, …,  4.03913009,
          4.14724578,  4.34602155]),
  array([-4.20615027, -4.11938899, -4.04883556, …,  4.00730552,
          4.02341064,  4.22733388])),
 (0.9982684637787755, -8.934958522480965e-05, 0.999989136072598))
```



[12]:
```python
fig_Y, ax_Y = plt.subplots(1,2,figsize=(16,4))
counts_Y.plot(ax=ax_Y[0], color='y', alpha=0.5,kind = 'bar',ylim = (0,0.
 ↪5),title='Var Y simulation',ylabel='percentage')
stats.probplot(Data['Y'], dist="norm", plot=plt)
```

[12]:
```
((array([-4.34602155, -4.14724578, -4.03913009, …,  4.03913009,
          4.14724578,  4.34602155]),
  array([-4.17234596, -3.99385517, -3.96749315, …,  4.0098114 ,
          4.12947287,  4.24163893])),
 (1.0009852336100467, 0.000960231795845812, 0.9999901435846327))
```



[13]:
```python
print(stats.kstest(Data['X'], 'norm',(Data['X'].mean(),Data['X'].std())))
print(stats.kstest(Data['Y'], 'norm',(Data['Y'].mean(),Data['Y'].std())))
```

```
KstestResult(statistic=0.0016640569140452532, pvalue=0.9442233350664352)
KstestResult(statistic=0.002630951549332572, pvalue=0.4922703727630263)
```

By results of Q-Q graph and ks-test, X and Y follow normal distributions well.

2.1.2 Then we will use the Acceptance-Rejection Method to generate samples of random variables to satisfy standard Normal distribution.¶

Step 1: the PDF of Normal distribution $f(X=x,\mu=0,\sigma=1)=\frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}, x\in(-\infty,+\infty)$

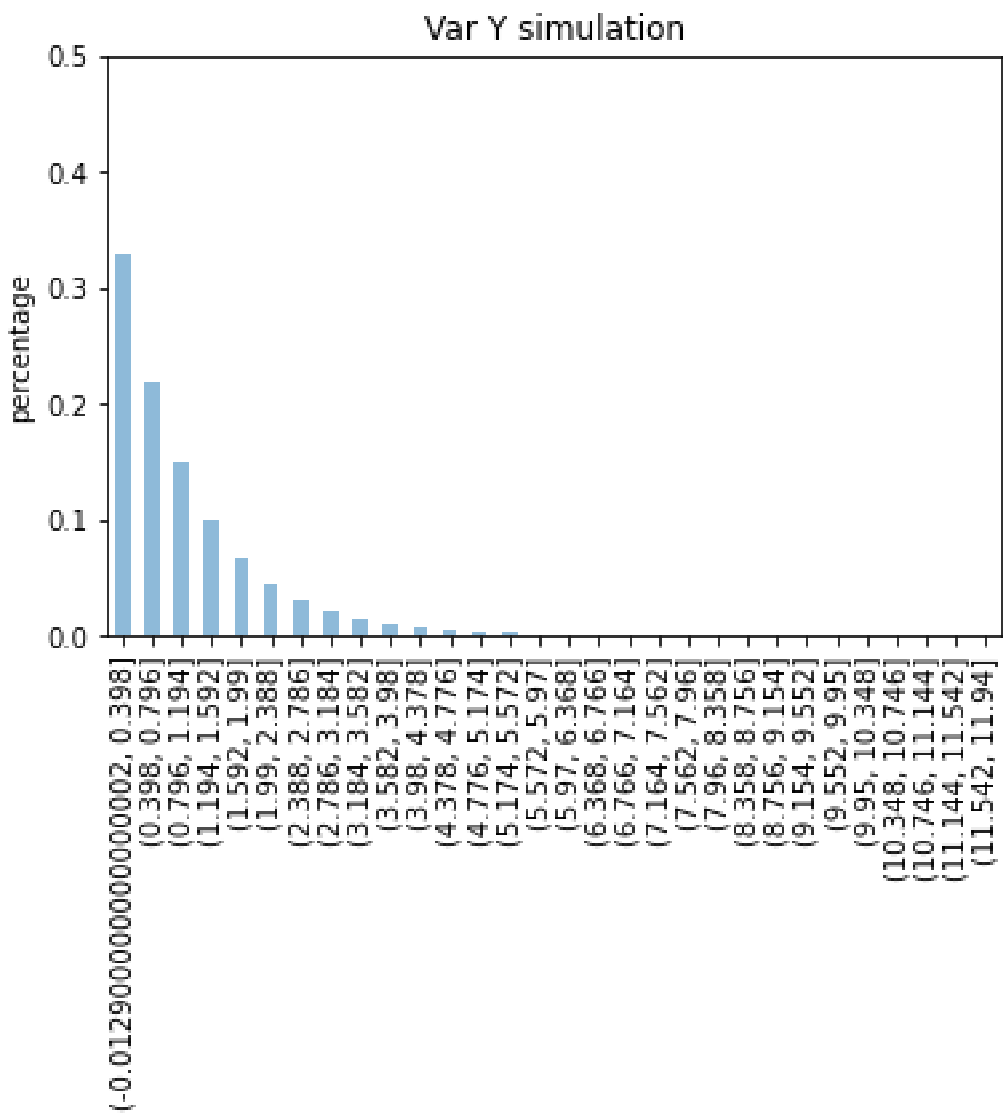
Step 2: To simplify , we choose $Y\sim Expo(\frac{1}{2})$ and $g(Y=y)=e^{-y}, y\in(0,+\infty)$ and Y is geneeate by $U1\sim Unif(0,1)$

Step 3: the constant var $c\geq max(\frac{\hat{f}(x)}{g(x)})=max(\sqrt{\frac{2}{\pi}}\cdot e^{x-\frac{x^2}{2}})=\sqrt{\frac{2e}{\pi}}$. Hence $\frac{\hat{f}(x)}{cg(x)}=e^{-\frac{(x-1)^2}{2}}$

Step 4: Let $U2\sim Unif(0,1)$ and $U3\sim Unif(0,1)$, if $U2\leq\frac{f(x)}{cg(x)}$,then if $U3>=0.5$, $X=Y$, otherwise $X=-Y$

[14]:
```
size = 100000
U1 = np.random.random(size)
U2 = np.random.random(size)
U3 = np.random.random(size)
X = []
Y = -np.log(U1)
data = pd.DataFrame({'Y':Y})
counts_Y = data['Y'].value_counts(bins = 30,sort = False , normalize = True)
counts_Y.plot(alpha=0.5,kind = 'bar',ylim = (0,0.5),title='Var Y␣
↪simulation',ylabel='percentage')
```

[14]:
```
<AxesSubplot:title={'center':'Var Y simulation'}, ylabel='percentage'>
```



[15]:
```
for i in range(0,size-1):
    if U2[i] <= np.exp((-(Y[i]-1)**2)/2):
        if U3[i] >= 0.5:
            X.append(Y[i])
        else:
            X.append(-Y[i])
X = np.array(X)
data = pd.DataFrame({'X':X})
counts_X = data['X'].value_counts(bins = 30,sort = False , normalize = True)
print(data['X'])
```

```
0        -0.347456
1        -0.206205
```

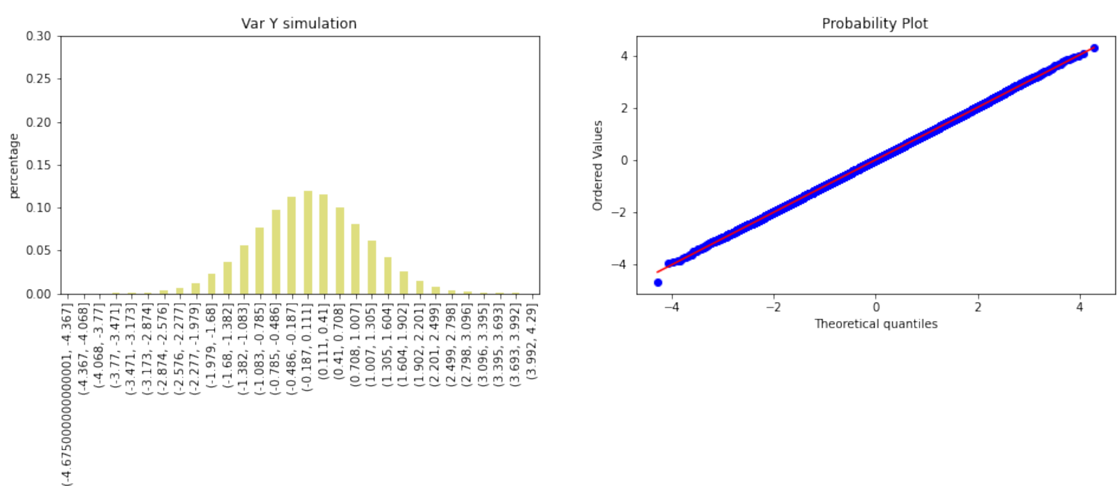
```
2         1.020818
3        -0.086596
4         1.387774
            …
76122    -0.004819
76123    -0.376571
76124    -0.544608
76125    -1.027802
76126    -0.709377
Name: X, Length: 76127, dtype: float64
```

[16]:

```python
fig_X, ax_X = plt.subplots(1,2,figsize=(16,4))
counts_X.plot(ax=ax_X[0], color='y', alpha=0.5,kind = 'bar',ylim = (0,0.
 ↪3),title='Var Y simulation',ylabel='percentage')
stats.probplot(data['X'], dist="norm", plot=plt)
```

[16]:

```
((array([-4.28577647, -4.08434016, -3.97467981, …,  3.97467981,
          4.08434016,  4.28577647]),
  array([-4.66526482, -3.95090571, -3.89445904, …,  3.97131042,
          4.06041138,  4.29030042])),
 (1.0027795842972136, 0.0006298808443084742, 0.9999791182526))
```



[17]:

```python
print(stats.kstest(data['X'], 'norm',(data['X'].mean(),data['X'].std())))
```

```
KstestResult(statistic=0.0029813153631731426, pvalue=0.5069273627459023)
```

By results of Q-Q graph and ks-test, X follows normal distributions well.

2.1.3 cons and pros between two methods:

For Box-Muller method:It easy to generate var in normal distribution , but only generate var in normal disttabution,There are limitations.

For Acceptance-Rejection Method, it can generate any var in all kinds of distribution with its PDF, however PDF and const c is not easy to find.

3 Q3

3.1 Given a random variable Y with the standard Normal distribution, evaluate $c = P(Y > 8)$ by Monte Carlo methods with & without importance sampling.Discuss the pros and cons of importance sampling.

[18]:
```
Size = 1000000
N = np.random.normal(0,1,Size)
```

First, we choose Monte Carlo methods without importance sampling.

[19]:
```
P_without = sum(N>8)/Size
print(sum(N>8))
print(P_without)
```

```
0
0.0
```

Then, we choose Monte Carlo methods with importance sampling.

[20]:
```
N_add = N+8
P_with = sum((N_add>8)*np.exp((0.5*8**2)-8*N_add))/Size
print((N_add>8)*np.exp((0.5*8**2)-8*N_add))
print(sum((N_add>8)*np.exp((0.5*8**2)-8*N_add)))
print(P_with)
```

```
[0.00000000e+00 3.94359501e-15 2.78345933e-16 … 3.70170876e-18
 3.93244366e-16 1.44213531e-17]
6.206058372262149e-10
6.206058372262149e-16
```

3.1.1 cons and pros between two methods:

Monte Carlo methods without importance sampling: using for a large amount of data and easy to use. it's main usage is judge the distribution of data. But it hard to find the specific distribution in the case of small amount of data.

Monte Carlo methods with importance sampling: we can use a new distribution to find the specific distribution in the case of small amount of data.Nut it is hard to find a good new distribution.