Assignment-1: Exercises for Monte Carlo Methods

21307289 刘森元

Exercise. 1

根据蒙特卡罗方法，在平面内抽取随机点，根据概率有公式

$$\hat{\pi} = \frac{\text{落在圆内的点数}}{\text{总的点数}}$$

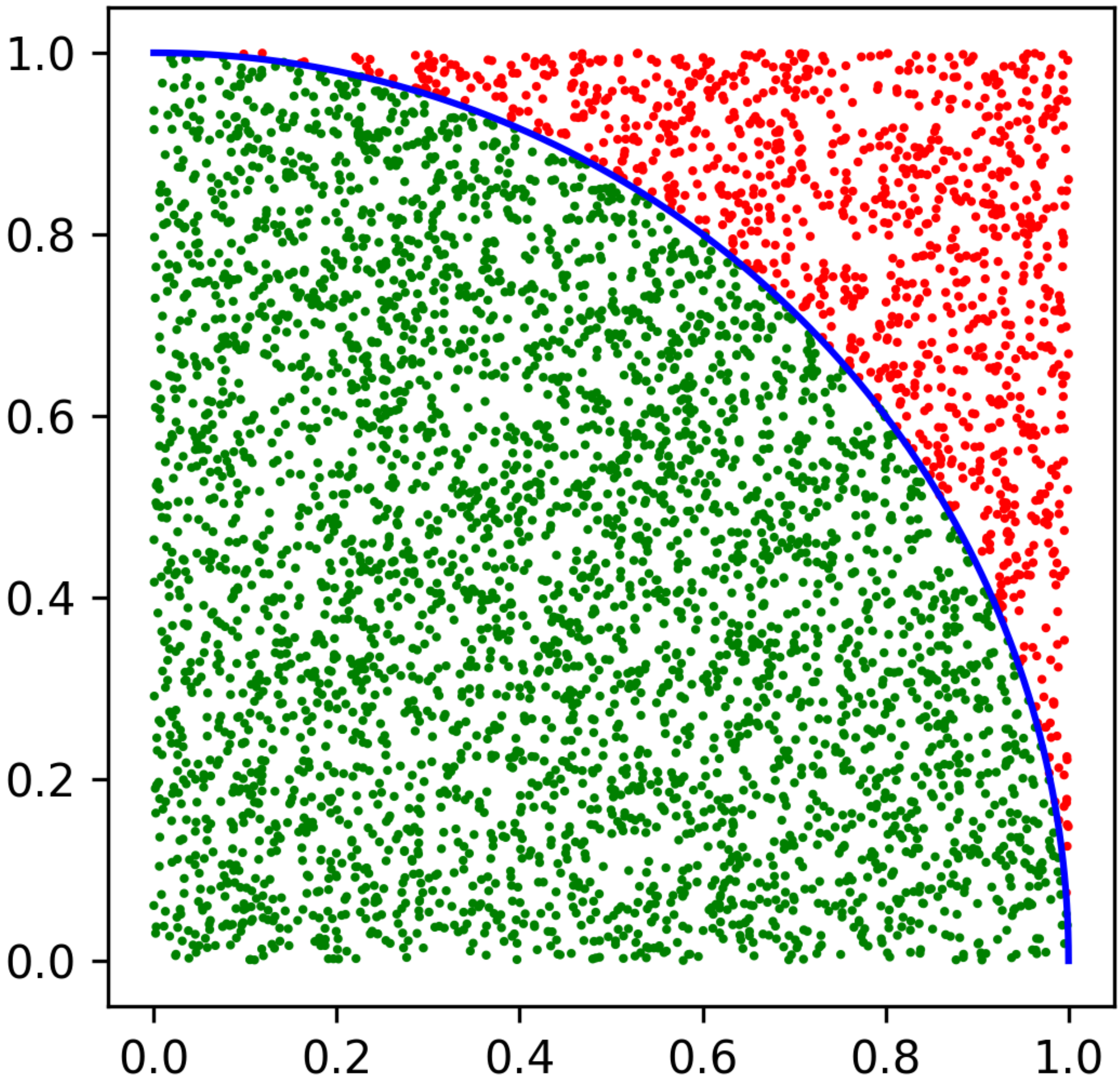
为了方便，采取 1/4 圆进行模拟，故有

$$\hat{\pi} = \frac{\text{落在圆内的点数}}{\text{总的点数}} \times 4$$

可写出核心代码（完整代码见 *Exercise-1.py* ）

```
X = np.random.rand(N)
Y = np.random.rand(N)
d = np.sqrt(np.square(X) + np.square(Y))
inplace = np.where(d < 1, 1, 0).sum()  # Calculating random points inplace
pi = inplace / N * 4
```

可得出结果




```
% python3 Exercise-1.py
        N   Average  Variance
0    20.0  3.114000  0.167404
0    50.0  3.165600  0.058913
0   100.0  3.152800  0.030444
0   200.0  3.133000  0.012539
0   300.0  3.148133  0.010339
0   500.0  3.143760  0.004490
0  1000.0  3.142520  0.001899
0  5000.0  3.137544  0.000556
```

可见随着 N 的值增大，答案均值愈发逼近真实值，方差逐渐减小

Exercise.2

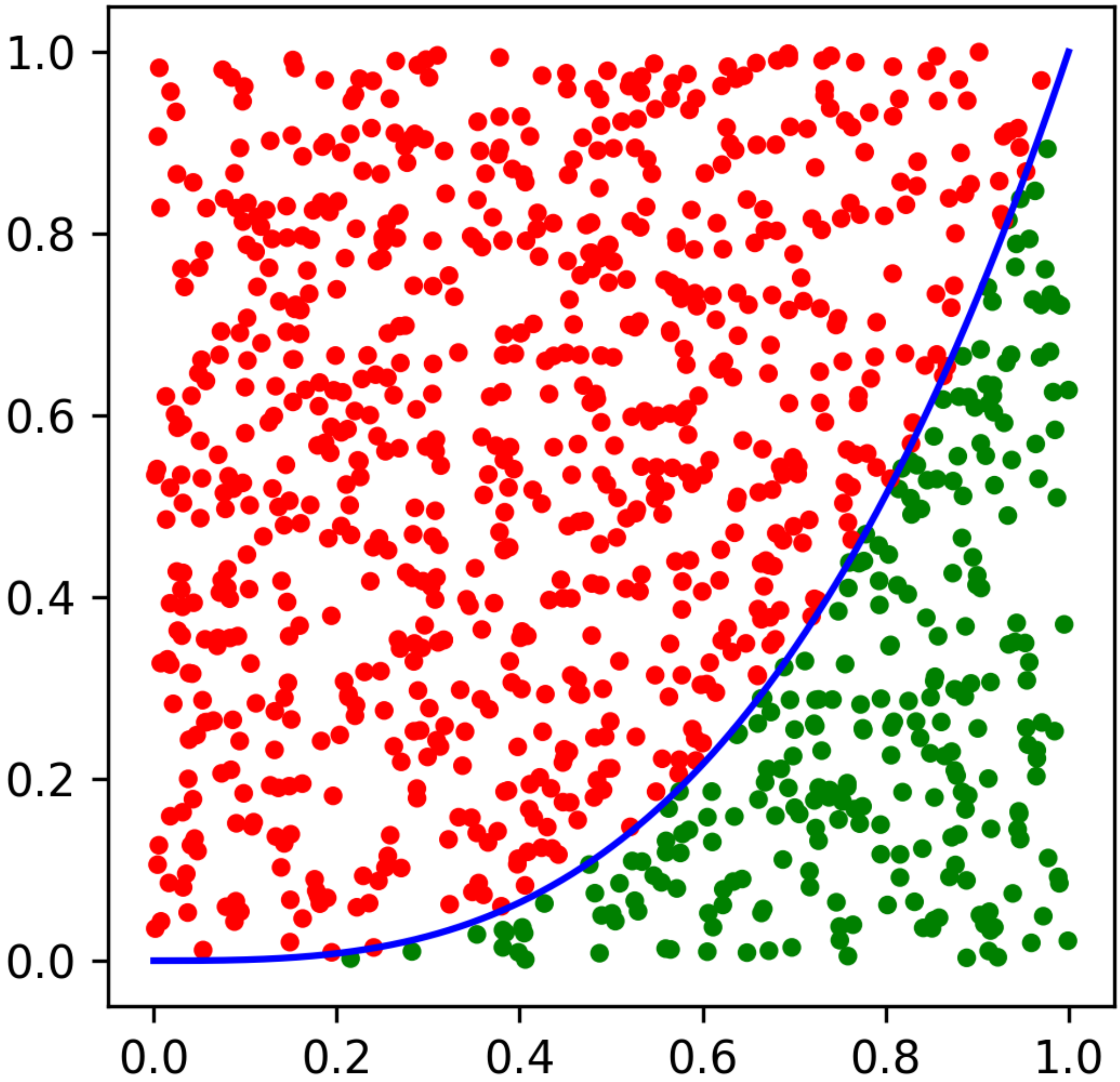
由积分的几何意义：曲边梯形的有向面积，结合蒙特卡罗方法，在平面内取随机点，根据概率有公式

$$\hat{I} = \frac{\text{落在区域内的点数}}{\text{总的点数}} \times \text{平面总面积}$$

可写出核心代码（完整代码见 *Exercise-2.py* ）

```
X = np.random.rand(N)
Y = np.random.rand(N)
inplace = np.where(Y < X * X * X, 1, 0)  # Calculating random points inplace
I = inplace.sum() / N
```

可得出结果




```
% python3 Exercise-2.py
          N   Average  Variance
0       5.0  0.224000  0.031424
0      10.0  0.246000  0.015284
0      20.0  0.254000  0.010984
0      30.0  0.242000  0.007458
0      40.0  0.243750  0.005330
0      50.0  0.260400  0.003376
0      60.0  0.249833  0.002758
0      70.0  0.241429  0.002688
0      80.0  0.255000  0.002113
0     100.0  0.246500  0.001661
0    1000.0  0.248460  0.000155
```

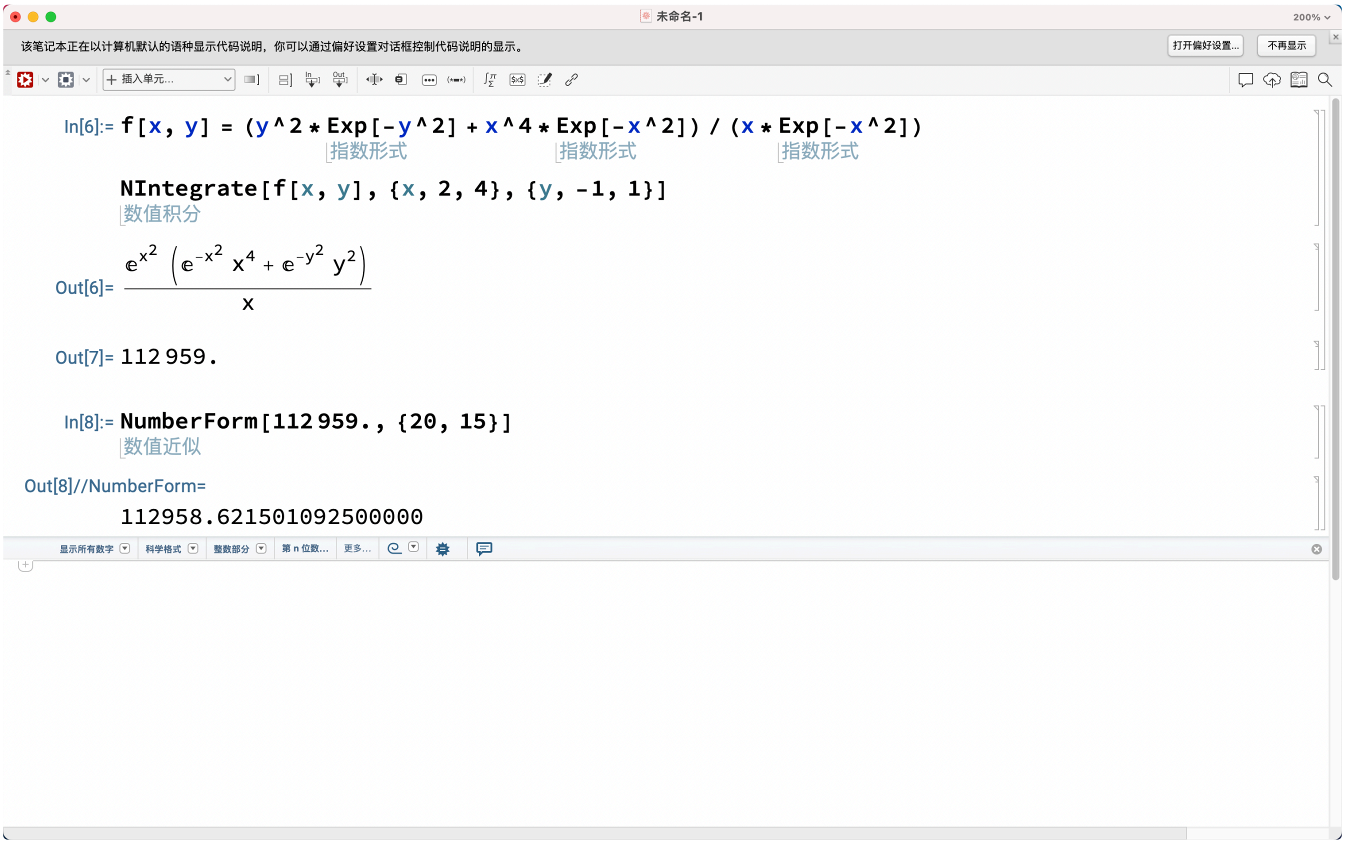
可见随着 N 的值增大，答案均值愈发逼近真实值，方差逐渐减小

Exercise.3

对于定积分

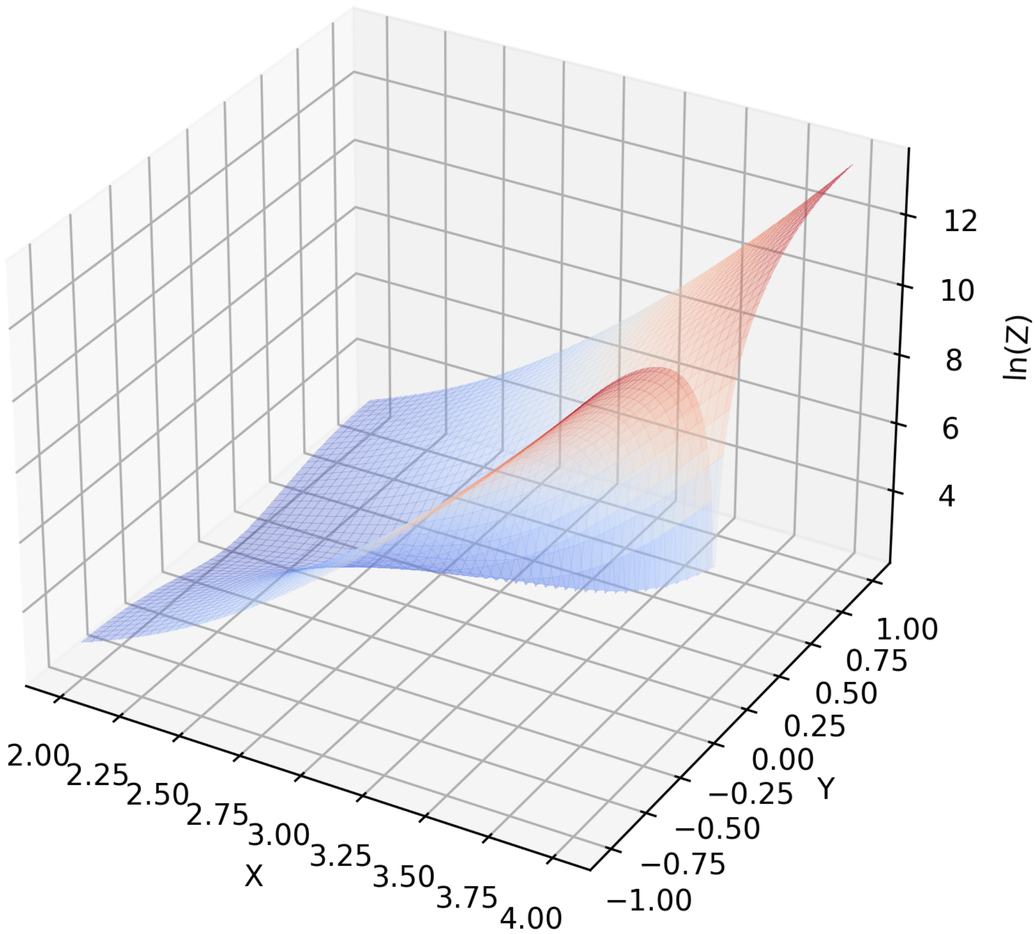
$$\int_{x=2}^{4}\int_{y=-1}^{1} f(x,y) = \frac{y^2\cdot e^{-y^2}+x^4\cdot e^{-x^2}}{x\cdot e^{-x^2}}$$

利用 Mathmatica



可求出其真实值为 `112958.6215010925`

做出该积分图像有



由多重积分的几何意义：曲顶柱体的有向体积，结合蒙特卡罗方法，在空间内取随机点，根据概率有公式

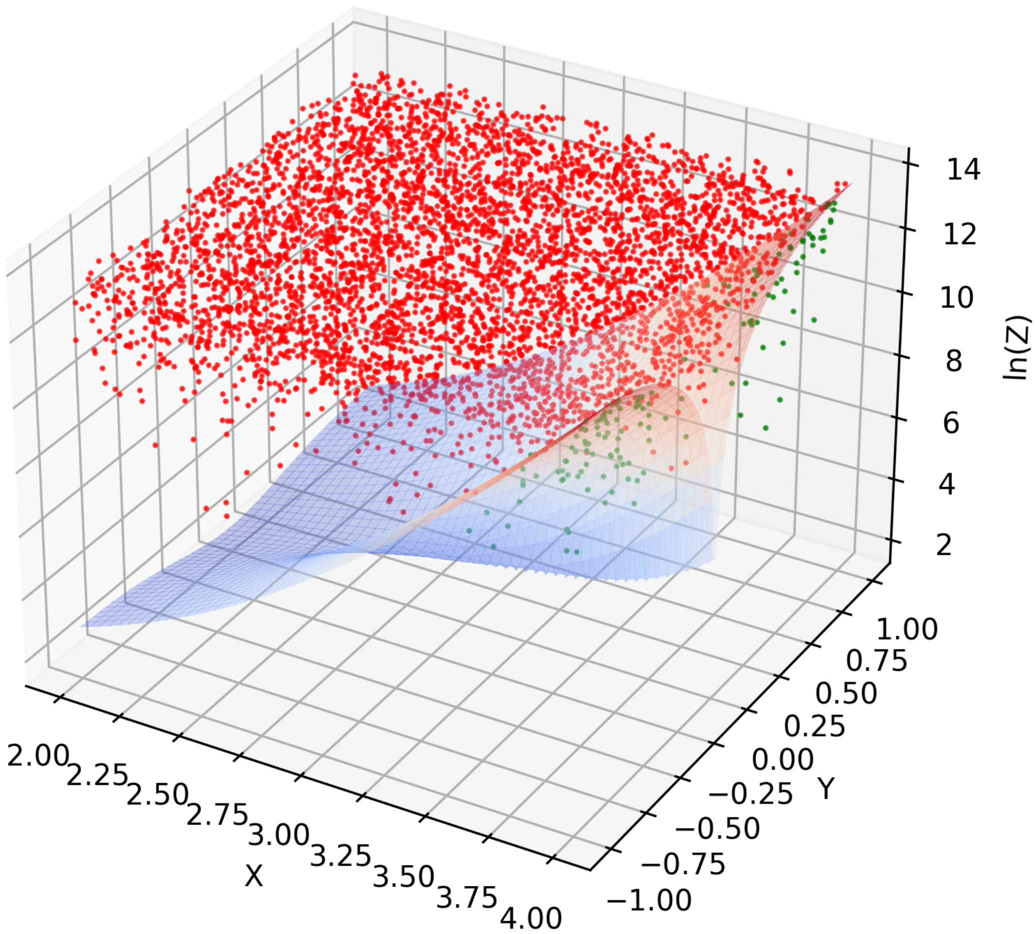
$$\hat{I} = \frac{\text{落在区域内的点数}}{\text{总的点数}} \times \text{空间总体积}$$

可写出核心代码（完整代码见 *Exercise-3.py*）

```
X = np.random.rand(N) * (X_max - X_min) + X_min
Y = np.random.rand(N) * (Y_max - Y_min) + Y_min
Z = np.random.rand(N) * Z_max
position = np.where(Z < f(X, Y), 1, 0)  # Calculating random points inplace
inplace = position.sum()
I = inplace / N * (X_max - X_min) * (Y_max - Y_min) * Z_max
```

可得出结果

可见随着 N 的值增大，答案均值愈发逼近真实值，方差逐渐减小




```
% python3 Exercise-3.py
         N        Average      Variance
0      5.0   91539.654429  6.857516e+10
0     10.0   94808.927802  2.841979e+10
0     20.0  120963.114781  2.010441e+10
0     30.0   99167.958965  1.023568e+10
0     40.0  150386.575134  9.849129e+09
0     50.0  117693.841409  7.438951e+09
0     60.0  123142.630363  6.805975e+09
0     70.0  126567.583420  6.207633e+09
0     80.0  107886.021292  5.762248e+09
0    100.0  119982.332770  4.639832e+09
0    200.0  114588.031705  1.258503e+09
0    500.0  112070.691208  6.960293e+08
0   5000.0  112325.694531  7.298120e+07
```

可见随着 N 的值增大，答案均值愈发逼近真实值，方差逐渐减小

Exercise.4

根据题目可实现函数 `ant()` 来模拟蚂蚁爬行进程，并返回是否能到达终点（ `True` / `False` ）

根据蒙特卡罗方法，可给出核心代码（完整代码见 *Exercise-4.py* ）

```
count = 0
for it in range(N):
    if ant():
        count += 1
P = count / N
```

最终有结果

```
% python3 Exercise-4.py
Possibility = 0.25640000
```

Exercise.5

根据蒙特卡罗方法，可给出核心代码（完整代码见 *Exercise-5.py* ）

```
A = np.random.rand(N)
B = np.random.rand(N)
C = np.random.rand(N)

upper = np.where(A < 0.85, 1, 0)
lower = np.bitwise_and(np.where(B < 0.95, 1, 0), np.where(C < 0.9, 1, 0))
all = np.bitwise_or(upper, lower)

P = all.sum() / N
```

最终运行 20000 次有结果

```
% python3 Exercise-5.py
Average = 0.97820460
Variance = 0.00000020
```

符合理论结果