HOMEWORK 1: Exercises for Monte Carlo Methods
March 5, 2019

Exercise 1.

The Monte Carlo method can be used to generate an approximate value of pi. The figure below shows a unit square with a quarter of a circle inscribed. The area of the square is 1 and the area of the quarter circle is pi/4. Write a script to generate random points that are distributed uniformly in the unit square. The ratio between the number of points that fall inside the circle (red points) and the total number of points thrown (red and green points) gives an approximation to the value of pi/4. This process is a Monte Carlo simulation approximating pi. Let N be the total number of points thrown. When N=50, 100, 200, 300, 500, 1000, 5000, what are the estimated pi values, respectively? For each N, repeat the throwing process 100 times, and report the mean and variance. Record the means and the corresponding variances in a table.

蒙特卡洛方法可以用于产生接近 pi 的近似值。图 1 显示了一个带有 1/4 内切圆在内的边长为 1 的正方形。正方形的面积是 1，该 1/4 圆的面积为 pi/4。通过编程实现在这个正方形中产生均匀分布的点。落在圈内（红点）的点和总的投在正方形（红和绿点）上的点的比率给出了 pi/4 的近似值。这一过程称为使用蒙特卡洛方法来仿真逼近 pi 实际值。令 N 表示总的投在正方形的点。当投点个数分别是 20, 50, 100, 200, 300, 500, 1000, 5000 时，pi 值分别是多少？对于每个 N，每次实验算出 pi 值，重复这个过程 20 次，并在表中记下均值和方差。

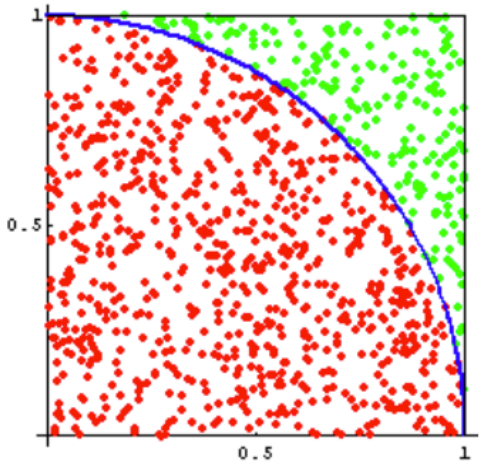


Figure 1 蒙特卡洛方法求解 pi

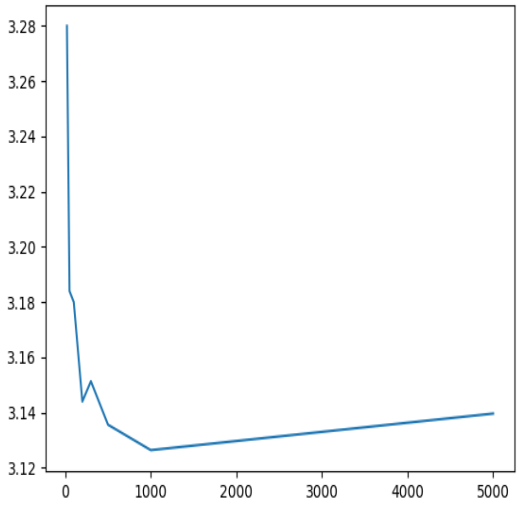
```
import math
import random
import matplotlib.pyplot as plt
import pandas as pd
k = 0
arrpim = [0]*8
```

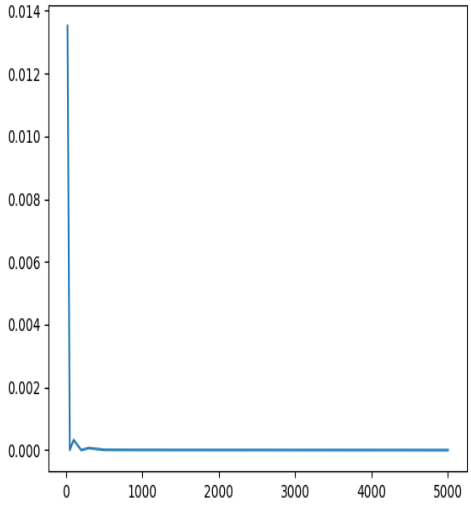
```
arrvar = [0]*8
s = [20, 50, 100, 200, 300, 500, 1000, 5000]
for n in s:
    arrpi = [0] * 20
    pim = 0
    var = 0
    k+=1
    for m in range(20):
        total = 0

        for i in range(n):
            x, y = random.random(), random.random()
            if math.sqrt(x * x + y * y) < 1.0:
                total += 1
                pi = 4 * total / n
        arrpi[m] = pi
        pim += arrpi[m]/20    #20 次随机过程，计算获得 pi 平均值
    arrpim[k - 1] = pim
    print(arrpi)
    var += (arrpi[m]-pim)*(arrpi[m]-pim)/n
    arrvar[k - 1] = var
print('pim', arrpim)
print('var', arrvar)
plt.plot(s, arrpim)
plt.savefig(r"1.png")
plt.show()
data = pd.DataFrame(arrpim)
data.to_csv('data.csv')

plt.plot(s, arrvar)
plt.savefig(r"2.png")
plt.show()
data1 = pd.DataFrame(arrvar)
data1.to_csv('data1.csv')
```

```
/Users/shuufuyou/PycharmProjects/untitled/venv/bin/python "/Users/shuufuyou/PycharmProjects/untitled/Monte Carlo.py"
[2.8, 3.4, 2.4, 3.0, 3.0, 3.2, 3.4, 3.4, 2.4, 3.4, 3.6, 2.8, 3.6, 2.4, 3.8, 3.6, 3.0, 3.0, 3.8, 3.4]
[3.2, 2.96, 3.04, 3.04, 3.04, 3.04, 3.12, 2.96, 3.28, 3.2, 3.2, 3.12, 3.12, 3.04, 3.04, 3.04, 2.96, 3.12, 3.52, 2.96]
[3.48, 3.12, 3.16, 3.24, 2.84, 3.24, 3.2, 2.92, 3.0, 3.16, 3.2, 3.2, 3.2, 3.16, 3.32, 3.04, 3.0, 3.32, 3.0, 3.44]
[3.24, 3.26, 3.3, 3.3, 3.18, 3.24, 3.28, 3.06, 3.04, 3.26, 3.24, 2.9, 3.08, 3.24, 3.04, 3.06, 3.22, 3.08, 3.3, 3.08]
[3.1333333333333333, 3.36, 3.12, 3.08, 3.2133333333333334, 3.1466666666666665, 3.1866666666666665, 3.3066666666666666, 3.12, 3.2266666666666666, 3.12, 3.3733333333333335, 3.08, 3.2533333333
[3.128, 3.272, 3.112, 3.088, 3.12, 3.096, 3.128, 3.176, 3.152, 3.08, 3.192, 3.136, 3.2, 3.176, 3.248, 3.088, 3.048, 3.08, 3.128, 3.208]
[3.228, 3.116, 3.14, 3.176, 3.148, 3.212, 3.14, 3.256, 3.192, 3.112, 3.18, 3.176, 3.152, 3.124, 3.132, 3.036, 3.176, 3.208, 3.068, 3.248]
[3.14, 3.132, 3.144, 3.1336, 3.1504, 3.12, 3.144, 3.1576, 3.1088, 3.1752, 3.14, 3.0968, 3.1832, 3.1664, 3.1792, 3.156, 3.1824, 3.1424, 3.1272, 3.1328]
pim [3.17, 3.1000000000000005, 3.1619999999999995, 3.17, 3.1873333333333333, 3.1428, 3.1610000000000005, 3.1456]
var [0.0026449999999999993, 0.0003920000000000032, 0.00077284000000000025, 4.0499999999999870-05, 2.1690370370370370-05, 8.502080000000096-06, 7.5689999999995550-06, 3.276799999999960-08]
```

	方差
0	0.002645
1	0.000392
2	0.00077284
3	4.05E-05
4	2.17E-05
5	8.50E-06
6	7.57E-06
7	3.28E-08

	平均值
0	3.17
1	3.1
2	3.162
3	3.17
4	3.187333333
5	3.1428
6	3.161
7	3.1456





Exercise 2.

We are now trying to integrate the another function by Monte Carlo method:

$$\int_0^1 x^3$$

A simple analytic solution exists here: $\int_{x=0}^{1} x^3 = 1/4$. If you compute this integration using Monte Carlo method, what distribution do you use to sample x? How good do you get when N = 5, 10, 20, 30, 40, 50, 60, 70, 80, 100, respectively? For each N, repeat the Monte Carlo process 20 times, and report the mean and variance of the integrate in a table.

我们现在尝试通过蒙特卡洛的方法求解如下的积分：

$$\int_0^1 x^3$$

该积分的求解我们可以直接求解，即有$\int_{x=0}^{1} x^3 = 1/4$。如果你用蒙特卡洛的方法求解该积分，你认为 x 可以通过什么分布采样获得？如果采样次数是分别是 N = 5, 10, 20, 30, 40, 50, 60, 70, 80, 100，积分结果有多好？对于每个采样次数 N，重复蒙特卡洛过程 100 次，求出均值和方差，然后在表格中记录对应的均值和方差。

我认为可以通过均匀分布采样获得

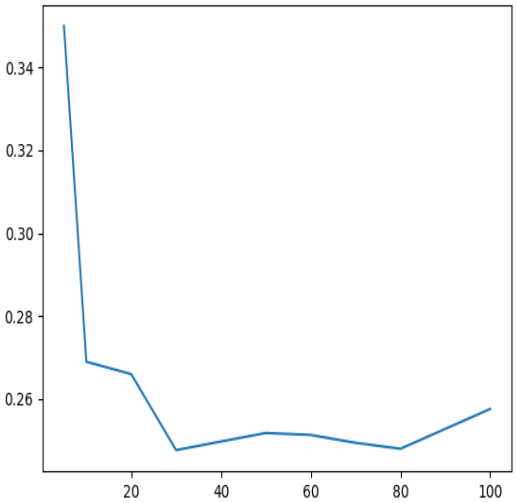
计算一重积分可以通过求面积方法来实现

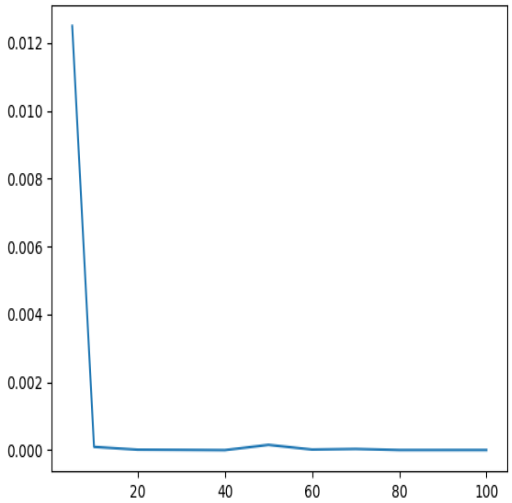
$f_0^1 x^3 dx$ 表示直线 x=0, x=1,y=0, $y = x^3$ 所围成的曲形的面积，基本思路与第一题相同

```
import math
import random
import matplotlib.pyplot as plt
import pandas as pd

k = 0
arrpim = [0]*10
arrvar = [0]*10
s = [5, 10, 20, 30, 40, 50, 60, 70, 80, 100]
for n in s:
    arrpi = [0] * 100
    pim = 0
    var = 0
    k+=1
    for m in range(100):                          # 100 次蒙特卡洛过程
        total = 0

        for i in range(n):
            x, y = random.random(), random.random()
            if y < (x * x * x):
                total += 1
                pi = total / n
        arrpi[m] = pi
        #print('F(X)', arrpi[m])
        pim += arrpi[m]/100
    arrpim[k - 1] = pim
    print(arrpi)
    var += (arrpi[m] - pim) * (arrpi[m] - pim) / n
    arrvar[k - 1] = var
print('pim', arrpim)
print('pim', arrpim)
print('var', arrvar)
plt.plot(s, arrpim)
plt.savefig(r"1.png")
plt.show()
data = pd.DataFrame(arrpim)
data.to_csv('data.csv')

plt.plot(s, arrvar)
plt.savefig(r"2.png")
plt.show()
data1 = pd.DataFrame(arrvar)
data1.to_csv('data1.csv')
```

```
/Users/shuufuyou/PycharmProjects/untitled/venv/bin/python /Users/shuufuyou/PycharmProjects/untitled/Monte.py
[0.2, 0.4, 0.4, 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.2, 0.2, 0.2, 0.6, 0.4, 0.4, 0.4, 0.8, 0.2, 0.2, 0.4, 0.4, 0.4, 0.4, 0.4, 0.6, 0.4, 0.4, 0.2, 0.2, 0.2, 0.4,
[0.2, 0.3, 0.3, 0.3, 0.1, 0.1, 0.2, 0.3, 0.4, 0.1, 0.5, 0.2, 0.1, 0.4, 0.3, 0.4, 0.2, 0.2, 0.1, 0.2, 0.3, 0.3, 0.2, 0.2, 0.2, 0.2, 0.3, 0.2, 0.3, 0.2, 0.2, 0.2, 0.2, 0.4, 0.6, 0.5, 0.2, 0.5,
[0.15, 0.2, 0.35, 0.3, 0.3, 0.2, 0.25, 0.35, 0.5, 0.2, 0.05, 0.35, 0.3, 0.35, 0.2, 0.25, 0.15, 0.2, 0.25, 0.2, 0.25, 0.2, 0.2, 0.15, 0.25, 0.2, 0.3, 0.25, 0.35, 0.2, 0.2, 0.3, 0.25, 0.25, 0.
[0.36666666666666664, 0.26666666666666666, 0.2, 0.3, 0.23333333333333334, 0.26666666666666666, 0.26666666666666666, 0.2, 0.16666666666666666, 0.23333333333333334, 0.26666666666666666, 0.2666
[0.225, 0.225, 0.225, 0.25, 0.3, 0.1, 0.325, 0.1, 0.225, 0.25, 0.275, 0.225, 0.4, 0.15, 0.125, 0.275, 0.25, 0.275, 0.15, 0.2, 0.325, 0.375, 0.175, 0.25, 0.15, 0.25, 0.35, 0.225, 0.275, 0.25,
[0.24, 0.26, 0.34, 0.16, 0.2, 0.3, 0.22, 0.2, 0.22, 0.22, 0.3, 0.2, 0.24, 0.14, 0.24, 0.26, 0.28, 0.28, 0.26, 0.22, 0.42, 0.16, 0.18, 0.24, 0.3, 0.32, 0.3, 0.24, 0.26, 0.26, 0.3, 0.36, 0.28,
[0.16666666666666666, 0.2, 0.25, 0.23333333333333334, 0.2, 0.26666666666666666, 0.2833333333333333, 0.26666666666666666, 0.18333333333333332, 0.2833333333333333, 0.3, 0.26666666666666666, 0.
[0.2571428571428571, 0.2571428571428571, 0.24285714285714285, 0.21428571428571427, 0.2714285714285714, 0.3, 0.2857142857142857, 0.12857142857142856, 0.24285714285714285, 0.3, 0.1142857142857
[0.2, 0.275, 0.2375, 0.325, 0.2125, 0.2125, 0.225, 0.25, 0.2875, 0.2375, 0.2625, 0.3125, 0.225, 0.3125, 0.275, 0.25, 0.325, 0.2625, 0.1875, 0.325, 0.25, 0.3, 0.325, 0.2625, 0.2125, 0.2, 0.26
[0.22, 0.25, 0.31, 0.27, 0.31, 0.26, 0.29, 0.24, 0.14, 0.32, 0.2, 0.24, 0.23, 0.24, 0.25, 0.22, 0.23, 0.25, 0.22, 0.23, 0.23, 0.24, 0.32, 0.21, 0.27, 0.2, 0.21, 0.35, 0.28, 0.26, 0.3, 0.2, 0
pim [0.3360000000000024, 0.270000000000002, 0.2515000000000017, 0.2400000000000019, 0.2540000000000017, 0.2566000000000001, 0.2571666666666668, 0.2429999999999999, 0.2557499999999998, 0
var [0.0008191999999999944, 0.0004900000000000024, 1.1250000000002518e-07, 1.4814814814815621e-06, 0.0001560250000000072, 5.511200000000076e-06, 8.560185185185554e-07, 1.1778717201166118e-0
```

	均值		方差
0	0.336	0	0.0008192
1	0.27	1	0.00049
2	0.2515	2	1.13E-07
3	0.24	3	1.48E-06
4	0.254	4	0.000156
5	0.2566	5	5.51E-06
6	0.2571667	6	8.56E-07
7	0.243	7	1.18E-05
8	0.25575	8	4.03E-05
9	0.256	9	3.14E-05





Exercise 3:
We are now trying to integrate a more difficult function by Monte Carlo method that may not be analytically computed:

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

Can you compute the above integration analytically? If you compute this integration using Monte Carlo method, what distribution do you use to sample (x,y)? How good do you get when the sample sizes are N = 5, 10, 20, 30, 40, 50, 60, 70, 80, 100, 200 respectively? For each N, repeat the Monte Carlo process 100 times, and report the mean and variance of the integrate.

我们现在尝试通过蒙特卡洛的方法求解如下的更复杂的积分：

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

你能够通过公式直接求解上述的积分吗？如果你用蒙特卡洛的方法求解该积分，你认为(x, y)可以通过什么分布采样获得？如果点（x, y）的采样次数是分别是 N = 10, 20, 30, 40, 50, 60, 70, 80, 100, 200, 500, 积分结果有多好？对于每个采样次数 N，重复蒙特卡洛过程 100 次，求出均值和方差，然后在表格中记录对应的均值和方差。

不能直接通过公式求解上述积分，采用蒙特卡洛的方法求解该积分，我认为(x, y)可以通过均匀分布采样获得

二重积分在几何意义上是代表曲顶柱体的体积，$\int_a^b\int_c^d f(x)dxdy$ 的值等于以 D 为底，曲面 z=f(x,y)为顶的曲顶柱体, 计算曲顶柱体的体积时，可以用“平行截面面积”法来计算

采用最大似然法估计平均值：

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

$$f(x,y) = \frac{y^2}{x} * e^{-y^2+x^2} + x^3$$

$$\int_{x=2}^{4}\int_{y=-1}^{1} x^3 dxdy = 480$$

$\frac{y^2}{x} * e^{-y^2+x^2}$ 随 y^2,x 增大而增大，$\frac{y^2}{x} * e^{-y^2+x^2}$ 当 y 确定时接近呈指数增长，$f(x,y)$ 的均值受最大值影响较大，$f(x,y)_{mean}$ 约等于 $\frac{1}{4}e^{13}$ =110600, 远大于 $\int_{x=2}^{4}\int_{y=-1}^{1} x^3 dxdy = 480$，预计最终计算结果应于 111000 接近。

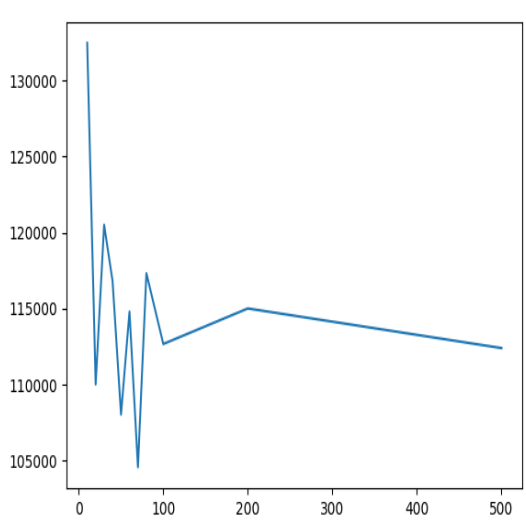
```
import math
import random
import matplotlib.pyplot as plt
import pandas as pd
k = 0
arrpim = [0]*10
arrvar = [0]*10
```

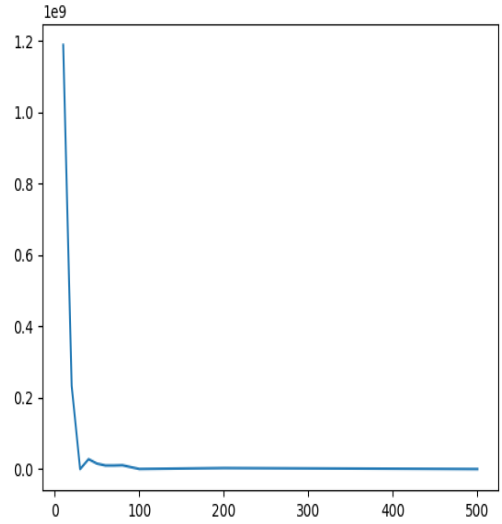
```
s = [5, 10, 20, 30, 40, 50, 60, 70, 80, 100]
for n in s:
    arrpi = [0] * 100
    pim = 0
    var = 0
    k+=1
    for m in range(100):                        # 100 次蒙特卡洛过程
        total = 0

        for i in range(n):
            x, y = random.random(), random.random()
            if y < (x * x * x):
                total += 1
                pi = total / n
        arrpi[m] = pi
        #print('F(X)', arrpi[m])
        pim += arrpi[m]/100
    arrpim[k - 1] = pim
    print(arrpi)
    var += (arrpi[m] - pim) * (arrpi[m] - pim) / n
    arrvar[k - 1] = var
print('pim', arrpim)
print('var', arrvar)
plt.plot(s, arrpim)
plt.savefig(r"1.png")
plt.show()
data = pd.DataFrame(arrpim)
data.to_csv('data.csv')

plt.plot(s, arrvar)
plt.savefig(r"2.png")
plt.show()
data1 = pd.DataFrame(arrvar)
data1.to_csv('data1.csv')
```

```
/Users/shuufuyou/PycharmProjects/untitled/venv/bin/python /Users/shuufuyou/PycharmProjects/untitled/S3.py
pim [135092.75026778193, 103344.96456696988, 111824.5541532504, 109493.97599148245, 121271.83049517855, 118001.73242851907, 109794.51355050916, 108934.01686450689, 109976.829320698, 111221.6
var [1357426387.3231456, 39893292.587871775, 152070305.03607273, 153239854.1269769, 94336700.60557383, 31571363.028356127, 519846.67950125615, 2915410.7461012388, 1550241.565306499, 3167730.
```

	均值
0	135092.75
1	103344.96
2	111824.55
3	109493.98
4	121271.83
5	118001.73
6	109794.51
7	108934.02
8	109976.83
9	111221.65
10	114059.94

	方差
0	1357426387
1	39893292.6
2	152070305
3	153239854
4	94336700.6
5	31571363
6	519846.68
7	2915410.75
8	1550241.57
9	3167731
10	42502.0623





通过蒙特卡洛预测分析，当 n 增大时，最终结果在 111000 附近，蒙特卡洛预测分析结果正确。算法与实验思路正确。