19335030_陈至雪 _HW1

Exercise 1 蒙特卡洛法计算 pi

利用随机函数`random.random()`，生成范围在[0,1]之间的随机数，作为随机点的 x 和 y 坐标。每次产生 n 个随机点，然后对落在圆内的点进行计算，最后通过公式 $\pi = 4 * \frac{落在圆内的点数}{总的点数}$ 估算 π 的值。对每个 n 计算100次，取均值并计算方差。

代码：

```python
import random
import numpy as np
#n is the total number
def count_pi(n):
    times = 0
    pi = []
    while times < 100:
        i = 0
        count = 0
        while i < n:
            #产生一个0-1的随机数
            x = random.random()
            y = random.random()
            if (pow(x,2) + pow(y,2)) < 1:
                count += 1
            i += 1
        pi.append(4 * (count / n))
        times += 1
    pi = np.array(pi)
    mean = np.mean(pi)
    var = np.var(pi)
    return mean, var

N = [50, 100, 200, 300, 500, 1000, 5000]
MEAN = []
VAR = []
print(type(VAR))
for n in N:
    mean, var = count_pi(n)
    MEAN.append(mean)
    VAR.append(var)

print("maen is")
print(MEAN)
print("Var is ")
print(VAR)
```

结果：

N	均值	方差
20	3.1220000000000003	0.16791599999999998
50	3.1328000000000005	0.03708415999999999
100	3.1452	0.026932959999999995
200	3.1602000000000006	0.008155959999999995
300	3.1349333333333336	0.008143217777777774
500	3.1444	0.005176800000000005
1000	3.1469600000000004	0.0026686784000000013
5000	3.1439439999999994	0.0006626976639999997

可以看到 n 越大，均值越接近 π 的真实值，方差越小。

Exexcise 2 蒙特卡洛法算定积分

原理：

定积分其实就是一个面积，将其设为 I，现在我们就是要求出这个 I。我们的想法是通过在包含定积分的面积为 S 的区域（通常为矩形）内随机产生一些随机数，其数量为 N，再统计在积分区域内的随机数，其数量为 i，则产生的随机数在积分区域内的概率为 $\frac{i}{N}$，这与积分区域与总区域面积的比值 $\frac{I}{S}$ 应该是近似相等的，我们利用的就是这个关系，即

$$\frac{I}{S} \approx \frac{i}{N}$$

最后即得所求定积分算式为：

$$I = \frac{i}{N} S$$

代码实现：

```
#蒙特卡洛求积分[0,1]x^3
def cal_x_3(n):
    times = 0
    res = []
    while times < 100:
        sum = 0
        i = 0
        j = 0
        while i < n:
            y = random.random()
            if (y < pow(random.random(), 3)):
                #sum += y
                j += 1
            i += 1
```

```
        sum = j/n
        res.append(sum)
        times += 1
    res = np.array(res)
    mean = np.mean(res)
    var = np.var(res)
    return mean, var
```

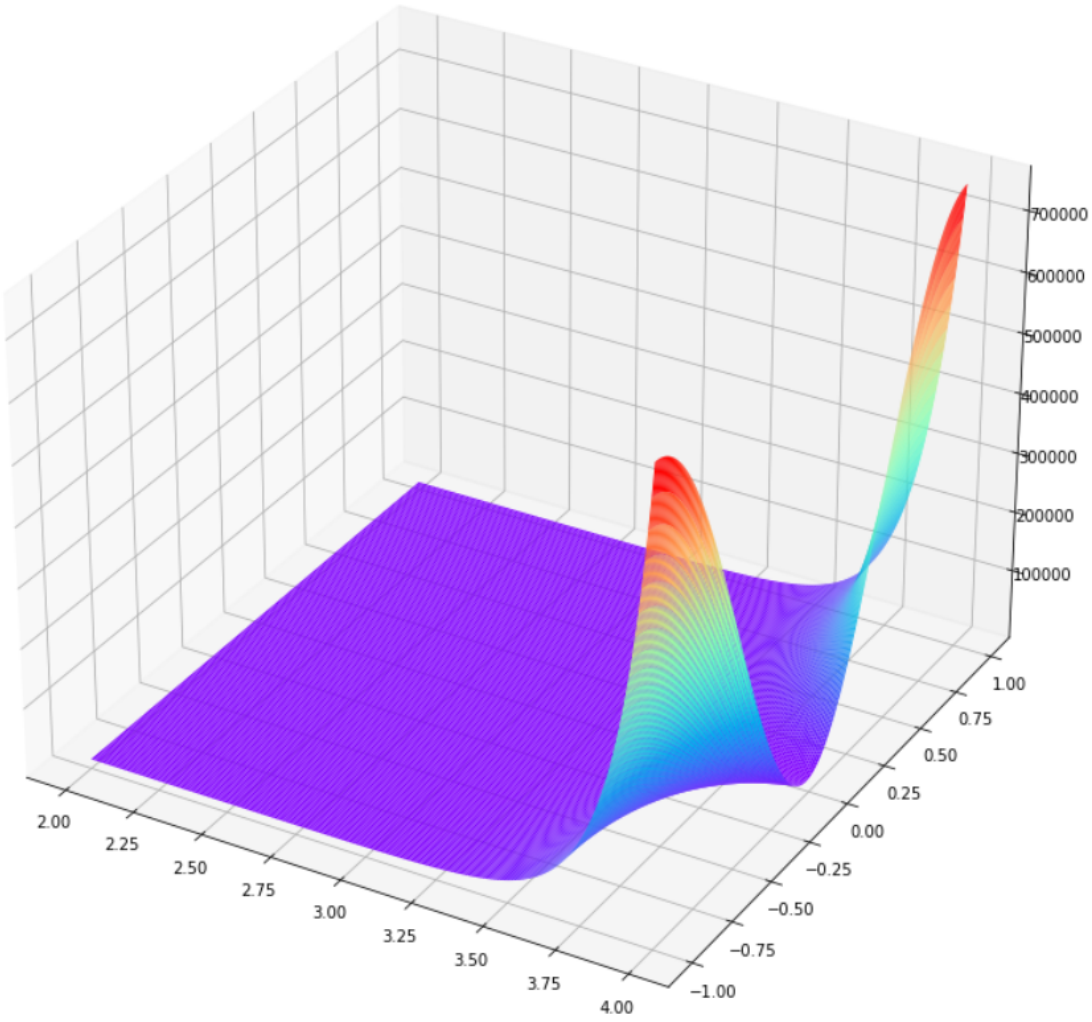
输入的n为要取的随机数的个数，对每个n计算100次，取均值和方差进行返回。其中，用`random.random()`函数取范围为 [0,1] 的随机数。结果进行输出，如下：

n	均值	方差
5	0.24332561039280257	0.0182676396566898
10	0.26186777107589554	0.009777931499985227
20	0.24533034093481398	0.0034570137972379333
30	0.2530499759295687	0.002422745465419641
40	0.2457312025694085	0.0017808163686956846
50	0.25479135301809475	0.0015288628252355413
60	0.24968121087164544	0.0013736424718419806
70	0.24762524467093996	0.000983544372908745
80	0.2573600102549105	0.0010121911394057061
100	0.25261811320093935	0.0007867323150617048

可以看到，对每个 n 进行 100 次计算并取均值后，结果都接近于 0.25，但是 n 越大，方差越小。

Exercise 3 蒙特卡洛求积分[1]

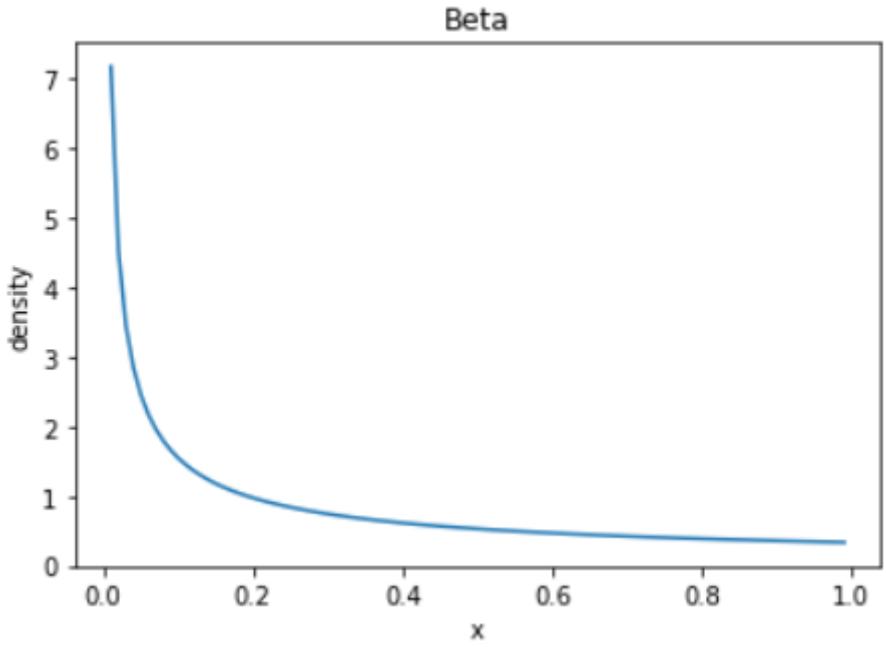
画出函数图像：



可以发现当 x 趋近于4时，函数值很大，而且呈指数上涨趋势。对 y，可以发现当 y 在 -1 和 1 附近时，函数值比较大，在 0.0 附近函数值比较小，为了让估算时误差不要太大，对 x 和 y 进行重要性采样。

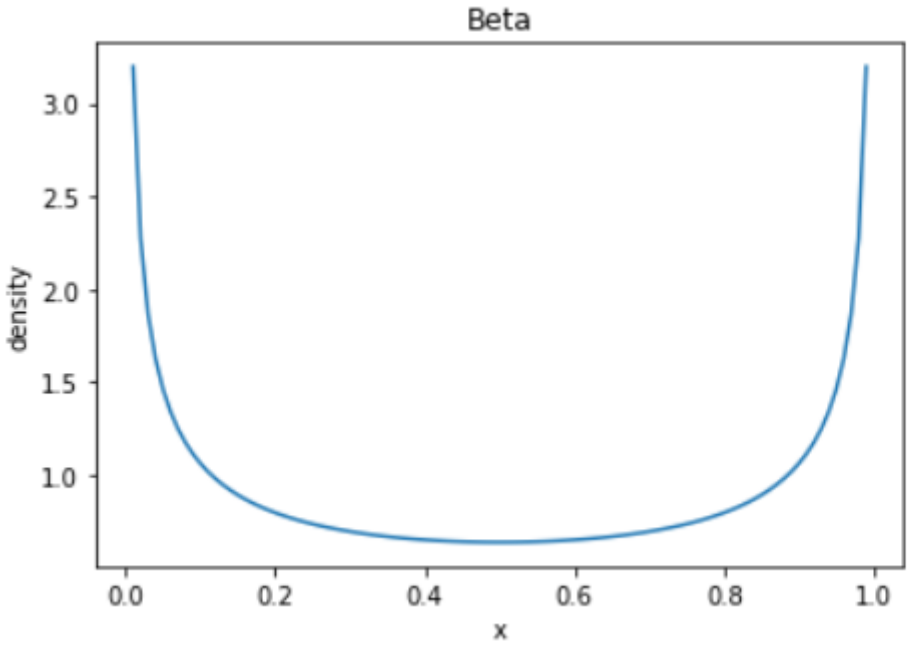
对 x 采用 beta(1/3, 1) 分布进行采样，这个采样偏向于靠近 4 的值，实现如下：

```
x = beta.rvs(0.33333, 1, size=1)
x = 1-x
x = x*2
x += 2
```



对 y 采用 beta 分布进行采样，这个采样偏向于处在 1 和 -1 附近的值，实现如下：

```
y = beta.rvs(0.5, 0.5, size=1) #生成0-1之间的随机数
y *= 2
y -= 1
```



对随机采样的 x 和 y 值代入函数，计算对应的函数值 $f(x,y)$ ，在范围 (0, 700000) 范围内均匀采样，将数值赋给 z , 若 $z < f(x,y)$， 则将计数值 times 加一，进行多次采样后最后计算出 $z < f(x,y)$ 的概率，最后的积分约等于此区域的体积与该概率之积。此区域的体积
$V = (4-2) * [1-(-1)] * 700000$. 实现代码如下：

```
import math
import random
import numpy as np
from scipy.stats import beta
import matplotlib.pyplot as plt
def cal_fx(n):
    times = 0
```

```
    res = []

    while times < 100:
        i = 0
        fx = 0
        fy = 0
        j = 0
        temp = 0
        time = 0
        while i < n:
            x = beta.rvs(0.33333, 1, size=1)
            x = 1-x
            x = x*2
            x += 2
            #x /= 4.0
            #x = x + 2
            y = beta.rvs(0.5, 0.5, size=1) #生成0-1之间的随机数
            y *= 2
            y -= 1
            fy = (pow(y,2) * np.exp(-pow(y,2))+pow(x,4)*np.exp(-
pow(x,2)))/(x * np.exp(-pow(x,2)))
            z = random.uniform(0,700000)
            #print(fy)
            #print(z)
            if( z < fy ):
                temp += z
                time += 1
            i += 1
            #if( time == 0 ):
            #    time = 1
            #print("x:")
            #print(y)
            #print("y:")
            #print(y)
        #temp /= time #算高的均值
        p = time/ n
        res.append(p * 4 * 700000) #算体积
        times += 1
    res = np.array(res)
    mean = np.mean(res)
    var = np.var(res)
    return mean, var, res

N = [10,20,30,40,50,60,70,80,100,200,500]
MEAN = []
VAR = []
RES = []
for n in N:
    mean,  var , res = cal_fx(n)
    MEAN.append(mean)
    VAR.append(VAR)
    RES.append(res)
print("MEAN is:")
print(MEAN)
print("VAR is: ")
print(VAR)
```

输出结果如下表：

n	均值	方差
10	747600.0	nan
20	761600.0	nan
30	771866.6666666664	nan
40	760900.0	nan
50	780080.0	nan
60	765333.3333333333	nan
70	736000.0	nan
80	750050.0	nan
100	760200.0	nan
200	760480.0	nan
500	759360.0	nan

可以看到对每个采样次数，积分结果在75万左右，但是方差很大。导致的原因可能是采样的分布函数选择不当。

Exercise 4

思想：

- 状态的记录

  使用一个二维数组`canVisti[N][N]`记录每个点是否可以访问，可以访问的化，为true，否则为false。使用一个一维数组`next_step[4]`记录当前位置可以移动的方向，四个方向中，若该方向可以移动，则记录为`true`，否则记录为`false`.

- 状态的更新

  假设蚂蚁的当前状态在（x0，y0）。若（x0，y0）不是中心点，则在蚂蚁进入下一状态后`canVisit[x0][y0]`更新为 false；若 (x0,y0) 为中心点，进一步判断中心点之前是否被访问过，若被访问过，则将`canVisit[x0][y0]`更新为 false，否则更新为 true。

- 状态的检查

  对当前状态，检查其邻居结点有几个是允许访问的。该功能通过函数`int check(int i, int j)`实现。其中（i, j）是当前的状态。返回值是可以访问的邻居结点个数。注意返回值一定小于等于4。

  代码实现如下：

```
int check( int i, int j ){
    int res = 0;
    if( i - 1 >= 0 && canVisit[i-1][j] ){
        res ++;
        next_step[0] = true;
    }else next_step[0] = false;
```

```
    if( i + 1 < N && canVisit[i+1][j] ){
        res ++;
        next_step[1] = true;
    }else next_step[1] = false;
    if( j - 1 >= 0 && canVisit[i][j-1] ){
        res ++;
        next_step[2] = true;
    }else next_step[2] = false;
    if( j + 1 < N && canVisit[i][j+1] ){
        res ++;
        next_step[3] = true;
    }else next_step[3] = false;
    return res;
}
```

- 方向的选择

  当前状态（x0,y0）下，能选取往哪个方向走取决于（x0,y0）可被访问的邻居在哪个方向。处在边界的状态邻居个数小于4，同时，如果邻居不允许被访问，则可以选择的方向更少。只要对此进行判断即可知道在当前状态下可以选择什么方向进入下一个状态。在代码中，此功能用函数`int choice(int n)`函数实现，其中 n 是可以选择的1方向维度，返回方向的标号。

  代码实现如下：

```
//传进去的n一定大于等于1
int choice( int n ){
    if( n == 1 ) {
        for( int i = 0; i < 4; i++ ){
            if( next_step[i] )
                return i;
        }
    }
    int n_rand = rand() % n + 1;
    int i = 0;
    for( int k = 0; i < 4 && k < n_rand; i++ ){
        if( next_step[i] ) k++;
    }
    return i-1;
}
```

- 单次迭代

  每次从 (0, 0) 点出发，对 (0, 0) 状态进行检查，获取可以访问的邻居结点个数，然后随机选取一个方向进入下一个状态。当当前状态下还有可以访问的邻居结点时，就继续执行，直到到达一个不能继续往下走的状态。最后判断终止点是否是终点 (7, 7)。该功能用函数`bool loop()`实现，当到达的是终点 (7, 7) 时返回 true，否则返回true。

  代码实现如下：

```
bool loop(){
    for( int i = 0; i < N; i++ ){
        for (int j = 0; j < N; j++){
            canVisit[i][j] = true;
        }
    }
    int x = 0, y = 0;
    int times_cen = 0;
    int num_choice;
```

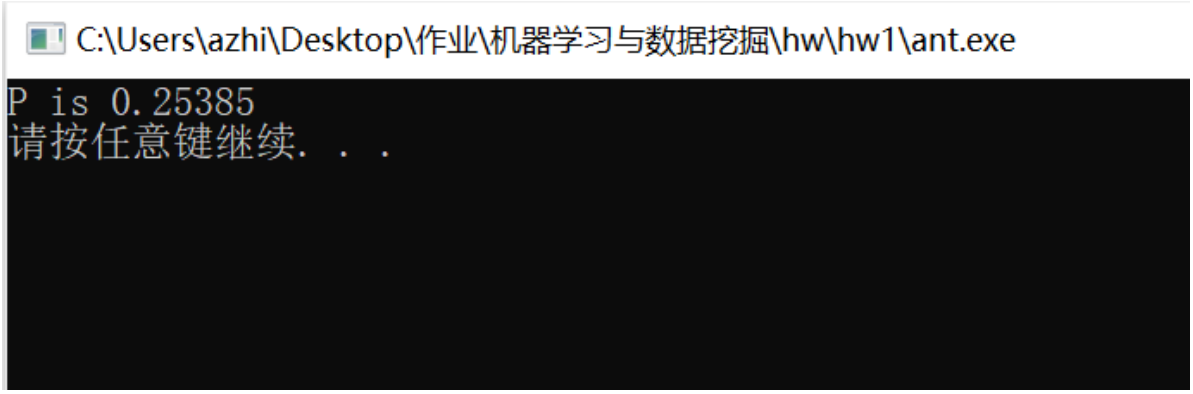
```
    num_choice = check(x,y);
    while( num_choice > 0 ){
        if( !( x == 3 && y == 3 ) ) { //if the point is not the center
            canVisit[x][y] = false;
        }
        else{
            times_cen ++;
            if( times_cen == 2 ) canVisit[x][y] = false;
        }
        int act;
        act = choice(num_choice);
        switch( act ){
            case 0:
                x--;
                break;
            case 1:
                x++;
                break;
            case 2:
                y--;
                break;
            case 3:
                y++;
                break;
        }
        if( x == N-1 && y == N-1 ) return true;
        num_choice = check(x,y);
    }
    return false;
}
```

- 概率计算

  调用 20000 次 loop 函数，用变量 res 记录成功到达终点的次数，P = res/ 20000。最后对结果进行输出。代码实现如下：

```
int main(){
    int runTimes = 20000;
    int res = 0;
    for ( int i = 0; i < runTimes; i++){
        if( loop() ) res ++;
    }
    cout << "P is " << (float) res / runTimes << endl;
    system("PAUSE");
    return 0;
}
```

实验结果：



可以看到，蚂蚁能到达终点的概率为 0.25385.

Exercise 5

实现代码如下：

```
N = 5000
res = 0
for i in range(N):
    pa = random.random()
    pb = random.random()
    pc = random.random()
    if( pa < 0.85 or ( pb < 0.95 and pc < 0.90 ) ):
        res += 1
print( res / N )
```

产生三个随机数，范围在 [0,1]，作为判别零件A、B、C是否能正常工作的值，按照题目逻辑，进行多次实验，对系统的 Reliability 进行估计，结果如下：

采样次数	Reliability
5000	0.9754
10000	0.9785
20000	0.9774
50000	0.9777
100000	0.9785

可以看到当采样次数增多时，概率收敛于整体可靠性。

参考：

1. https://blog.csdn.net/star_of_science/article/details/118820313