HOMEWORK 1: Exercises for Monte Carlo Methods

Student ID:18329015 Student Name:郝裕玮

Lectured by 梁上松, Sun Yat-sen University

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 100 次，并在表中记下均值和方差。

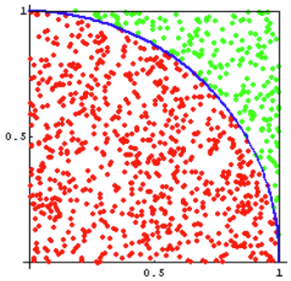


Figure 1 蒙特卡洛方法求解 pi

解答过程：

具体代码及注释如下：

```
import numpy as np
import math

#判断随机点是否在 1/4 圆范围内
def is_in(location):
    distance = location[0] ** 2 + location[1] ** 2
    distance = distance ** 0.5
    if distance < 1:
        return True
    else:
        return False
```

```
#投点个数
sizes = [20, 50, 100, 200, 300, 500, 1000, 5000]
#记录不同投点个数下的均值
mean = [0.0] * 8
#记录不同投点个数下的方差
var = [0.0] * 8

#对不同投点个数的情况进行实验
for num in range(0, 8):
    size = sizes[num]
    #每种投点个数的实验重复 100 次
    result = [0.0] * 100

    #每种投点个数的实验重复 100 次
    for time in range(0, 100):
        #生成随机坐标点 size 个(size 代表投点个数)
        points = np.random.rand(size, 2)
        #判断每个点是否在 1/4 圆内
        for one in points:
            if is_in(one):
                result[time] += 1
        result[time] /= size
        result[time] *= 4

    #计算均值和方差
    results = np.array(result)
    mean[num] = results.mean()
    var[num] = results.var()
mean = np.array(mean)
var = np.array(var)

#打印结果
for i in range(0,8):
    print("投点个数为：%d，均值为：%.6f，方差为：%.8f，相对误差
为：%.6f%%" %(sizes[i],\
                                                decimal.Decimal("%.6f" %floa
t(mean[i])),\
                                                decimal.Decimal("%.8f" %
float(var[i])),\
                                                100*abs(decimal.Decimal("%.8
f" %float(math.pi))-
decimal.Decimal("%.6f" %float(mean[i])))/(decimal.Decimal("%.8f" %float(math.p
i)))))
```

结果如下图所示：

PS：π 准确值为 3.1415926…

```
投点个数为：20，均值为：3.162000，方差为：0.11175600，相对误差为：0.649586%
投点个数为：50，均值为：3.172800，方差为：0.04918016，相对误差为：0.993361%
投点个数为：100，均值为：3.126400，方差为：0.02642304，相对误差为：0.483597%
投点个数为：200，均值为：3.138400，方差为：0.01458944，相对误差为：0.101625%
投点个数为：300，均值为：3.117200，方差为：0.01087038，相对误差为：0.776442%
投点个数为：500，均值为：3.135840，方差为：0.00566653，相对误差为：0.183113%
投点个数为：1000，均值为：3.146040，方差为：0.00268784，相对误差为：0.141564%
投点个数为：5000，均值为：3.144408，方差为：0.00046422，相对误差为：0.089615%
```

由结果可知，均值的相对误差和方差在大体趋势上均随投点个数的增加而减小。

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 100 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。

解答过程：

具体代码及注释如下：

```python
import numpy as np

#投点个数
sizes = [5, 10, 20, 30, 40, 50, 60, 70, 80, 100]
#记录不同投点个数下的均值
mean = [0.0] * 10
#记录不同投点个数下的方差
var = [0.0] * 10

#对不同投点个数的情况进行实验
for num in range(0, 10):

    size = sizes[num]
    #每种投点个数的实验重复 100 次
    result = [0.0] * 100
```

```
    #每种投点个数的实验重复 100 次
    for time in range(0, 100):
        #生成随机数据点的横坐标 x
        locations = np.random.rand(size, 1)
        #将生成的随机点横坐标代入函数进行累加积分
        for one in range(0, size):
            #该操作会使得使得随机数生成数组点变成 0-1 内的均匀有序随机分布
            x = locations[one]/size + one*1.0/size
            #进行累加积分
            result[time] += pow(x, 3)

        result[time] /= size

    #计算均值和方差
    results = np.array(result)
    mean[num] = results.mean()
    var[num] = results.var()

#打印结果
for i in range(0,8):
    print("投点个数为：%d，均值为：%.6f，方差为：%.8f，相对误差
为：%.6f%%" %(sizes[i],\
                                                decimal.Decimal("%.6f" %floa
t(mean[i])),\
                                                decimal.Decimal("%.8f" %
float(var[i])),\
                                                100*abs(decimal.Decimal("%.2
f" %float(0.25))-
decimal.Decimal("%.6f" %float(mean[i])))/(decimal.Decimal("%.2f" %float(0.25))
)))
```

结果如下图所示：

PS：该积分准确值为 1/4 = 0.25

```
投点个数为：5，均值为：0.250629，方差为：0.00130345，相对误差为：0.251600%
投点个数为：10，均值为：0.247770，方差为：0.00014571，相对误差为：0.892000%
投点个数为：20，均值为：0.249872，方差为：0.00001951，相对误差为：0.051200%
投点个数为：30，均值为：0.249901，方差为：0.00000602，相对误差为：0.039600%
投点个数为：40，均值为：0.249987，方差为：0.00000238，相对误差为：0.005200%
投点个数为：50，均值为：0.250053，方差为：0.00000124，相对误差为：0.021200%
投点个数为：60，均值为：0.250101，方差为：0.00000075，相对误差为：0.040400%
投点个数为：70，均值为：0.250019，方差为：0.00000048，相对误差为：0.007600%
```

由结果可知，均值的相对误差和方差在大体趋势上均随投点个数的增加而减小。

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：

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

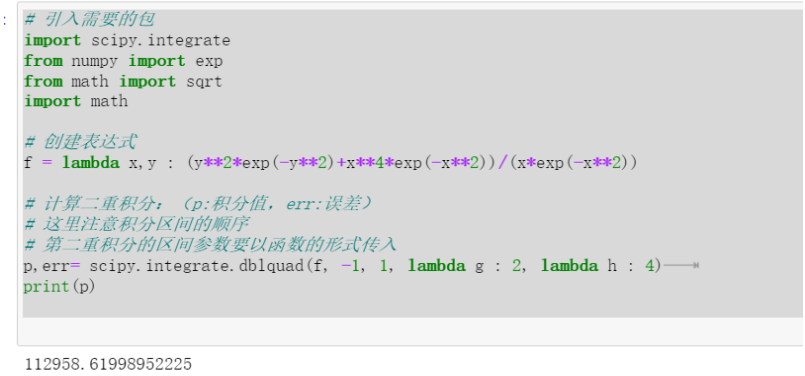
你能够通过公式直接求解上述的积分吗？如果你用蒙特卡洛的方法求解该积分，你认为(x, y)可以通过什么分布采样获得？如果点（x, y）的采样次数是分别是 N = 10, 20, 30, 40, 50, 60, 70, 80, 100, 200, 500, 积分结果有多好？对于每个采样次数 N，重复蒙特卡洛过程 100 次，求出均值和方差，然后在表格中记录对应的均值和方差。

解答过程：

首先我们先计算该积分的准确值，代码如下：

```
# 引入需要的包
import scipy.integrate
from numpy import exp
from math import sqrt
import math

# 创建表达式
f = lambda x,y : (y**2*exp(-y**2)+x**4*exp(-x**2))/(x*exp(-x**2))

# 计算二重积分：（p:积分值，err:误差）
# 这里注意积分区间的顺序
# 第二重积分的区间参数要以函数的形式传入
p,err= scipy.integrate.dblquad(f, -1, 1, lambda g : 2, lambda h : 4)
print(p)
```

运行可知积分准确值为：112958.61998952225



之后我们再进行蒙特卡洛模拟，代码如下：

```
import numpy as np
import math

#积分函数公式
def calculate(x1, y1):
    cal_result = []
    for index in range(0, len(x)):
        temp1 = y1[index] ** 2 * math.exp(-(y1[index] ** 2))
        temp2 = x1[index] ** 4 * math.exp(-(x1[index] ** 2))
        temp3 = x1[index] * math.exp(-(x1[index] ** 2))
        cal_result.append((temp1 + temp2) / temp3)

    return cal_result

#投点个数
sizes = [10, 20, 30, 40, 50, 60, 70, 80, 100, 200, 500]
#记录不同投点个数下的均值
mean = []
#记录不同投点个数下的方差
var = []
floor_space = 2.0 * 2.0

for num in range(0, len(sizes)):
    size = sizes[num]
    result = []
    #每种投点个数的实验重复 100 次
    for time in range(0, 100):

        #生成随机数据点
        x = np.random.rand(size, 1)
        x *= 2
        x += 2

        y = np.random.rand(size, 1)
        y *= 2
        y -= 1
        temp = calculate(x, y)
        ones = np.array(temp)
        one = ones.mean() * floor_space
        result.append(one)

    results = np.array(result)
    mean.append(results.mean())
    var.append(results.var())
    #积分准确值
    true_value=112958.61998952225
```

```
#打印结果
for i in range(0,8):
    print("投点个数为：%d，均值为：%.6f，方差为：%.8f，相对误差
为：%.4f%%" %(sizes[i],\
                                                decimal.Decimal("%.6f" %floa
t(mean[i])),\
                                                decimal.Decimal("%.8f" %
float(var[i])),\
                                                100*abs(decimal.Decimal("%.6
f" %float(true_value))-
decimal.Decimal("%.6f" %float(mean[i])))/(decimal.Decimal("%.6f" %float(true_v
alue)))))
```

结果如下图所示：

```
投点个数为：10，均值为：97881.468492，方差为：11621862805.98944283，相对误差为：13.3475%
投点个数为：20，均值为：123924.376086，方差为：7751647862.52150726，相对误差为：9.7078%
投点个数为：30，均值为：106406.483989，方差为：4256323601.65790462，相对误差为：5.8005%
投点个数为：40，均值为：115066.064218，方差为：2623979371.18007612，相对误差为：1.8657%
投点个数为：50，均值为：114234.772145，方差为：2385613775.56985283，相对误差为：1.1298%
投点个数为：60，均值为：108219.124066，方差为：2261148663.05091429，相对误差为：4.1958%
投点个数为：70，均值为：112112.791914，方差为：1906100502.12455535，相对误差为：0.7488%
投点个数为：80，均值为：112354.833725，方差为：1400758893.88661504，相对误差为：0.5345%
```

由结果可知，均值的相对误差和方差在大体趋势上均随投点个数的增加而减小。

Exercise 4.
An ant is trying to get from point A to point B in a grid. The coordinates of point A is (1,1) (this is top left corner), and the coordinates of point B is (n,n) (this is bottom right corner, n is the size of the grid).

Once the ant starts moving, there are four options, it can go left, right, up or down (no diagonal movement allowed). If any of these four options satisfy the following:

(a) The new point should still be within the boundaries of the n×n grid

(b) Only the center point (4, 4) is allowed to be visited zero, one or two times, while the remainder points should not be visited previously (are allowed to be visited zero or one time).

If P is the probability of the ant reaching point B for a 7×7 grid, use Monte Carlo simulation to compute P. Pick the answer closest to P in value (assume 20,000 simulations are sufficient enough to compute P).

一只蚂蚁试图从网格中的 A 点到达 B 点。 A 点的坐标是（1,1）（这是左上角）·B 点的坐标是（n·n）（这是右下角·n 是网格的大小）。

一旦蚂蚁开始移动·有四个选项·它可以向左、向右、向上或向下移动（不允许对角线移动）。 如果这四个选项中的任何一个满足以下条件：

(a) 新点仍应在 n×n 网格的边界内

(b) 只允许中心点(4, 4)被访问 0 次、1 次或 2 次，其余点不应该被访问过(允许访问 0 次或 1 次)。

如果 P 是 7×7 网格中蚂蚁到达点 B 的概率，则使用蒙特卡罗模拟计算 P。选择值最接近 P 的答案（假设 20,000 次模拟足以计算 P）。

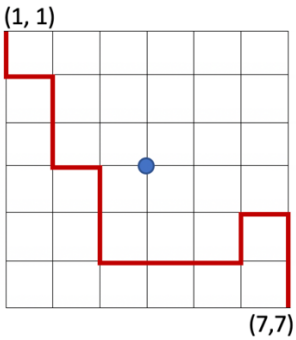


Figure 2 An ant is trying to get from point A (1,1) to point B (7,7) in a grid.

图 2 一只蚂蚁试图从网格中的 A (1,1) 点到达 B (7,7) 点。

解答过程：

具体代码及注释如下：

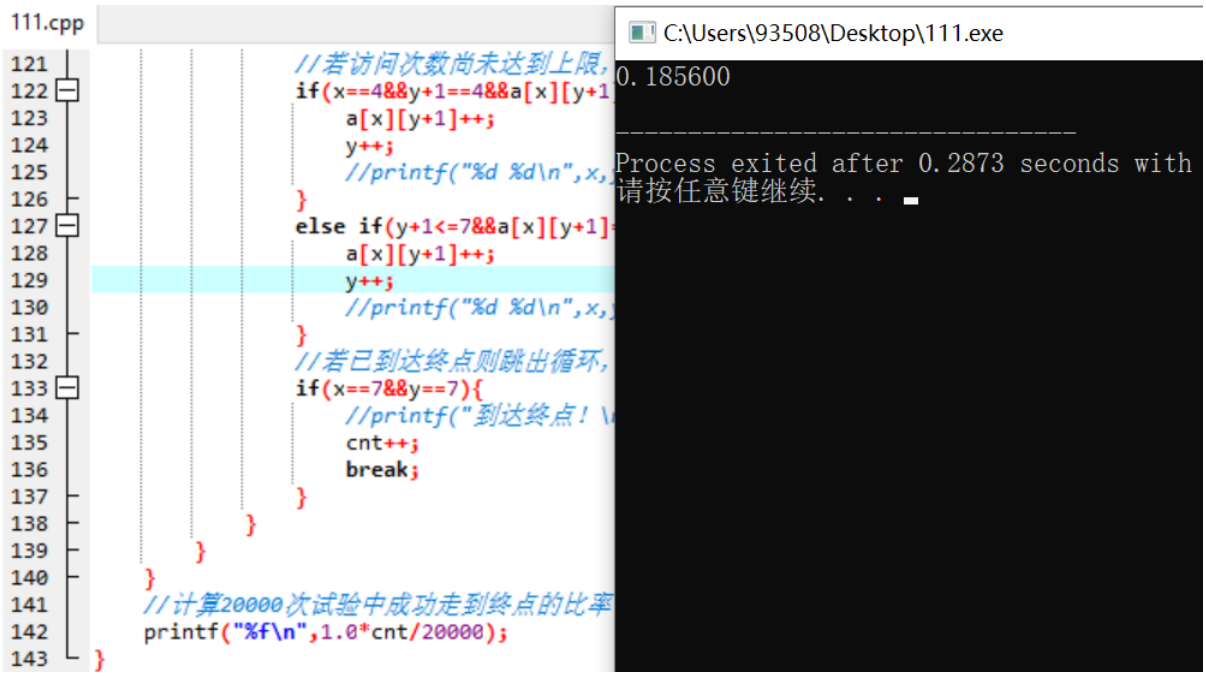
```c
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

int main()
{

    srand((int)time(0));
    int i,j,k,cnt=0;
    int temp=20000;//temp 代表 20000 次重复试验
    int a[8][8];//地图数组

    //蚂蚁的起点横纵坐标(1,1)
    int x=1;
    int y=1;
    while(temp--){
        //printf("第%d 次尝试\n",temp);
        //每次试验初始化
        //起点坐标重置为(1,1)
        //a[i][j]为非 0 值代表访问过，为 0 代表尚未访问
        x=1;
        y=1;
        for(i=0;i<=7;i++){
            for(j=0;j<=7;j++){
```

```
            if(i==0||j==0){
                //不属于地图的边缘值设置为 1，禁止访问
                a[i][j]=1;
            }
            else{
                a[i][j]=0;
            }
        }
    }
    //开始本次试验的随机探路，直至到达终点或无法再走时跳出循环
    while(1){
        //若该点上下左右有点(4,4)且已访问 2 次，则跳出循环
        if(x-1==4&&y==4){
            if(a[x-1][y]==2){
                break;
            }
        }
        else if(x+1==4&&y==4){
            if(a[x+1][y]==2){
                break;
            }
        }
        else if(x==4&&y-1==4){
            if(a[x][y-1]==2){
                break;
            }
        }
        else if(x==4&&y+1==4){
            if(a[x][y+1]==2){
                break;
            }
        }
        //若无点(4,4)但上下左右均访问过，也跳出循环
        else if(a[x-1][y]!=0&&a[x+1][y]!=0&&a[x][y-1]!=0&&a[x][y+1]!=0){
            break;
        }
        //选择随机方向
        int step=rand()%4+1;
        if(step==1){//往上走
            //若访问次数尚未达到上限，则更新当前坐标，并设置新坐标访问次数+1
            if(x-1==4&&y==4&&a[x-1][y]<=1){
                a[x-1][y]++;
                x--;
                //printf("%d %d\n",x,y);
            }
            else if(x-1>=1&&a[x-1][y]==0){
                a[x-1][y]++;
                x--;
```

```
                //printf("%d %d\n",x,y);
            }
            //若已到达终点则跳出循环，并设置成功次数+1
            if(x==7&&y==7){
                //printf("到达终点！\n");
                cnt++;
                break;
            }
        }
        if(step==2){//往下走
            //若访问次数尚未达到上限，则更新当前坐标，并设置新坐标访问次数+1
            if(x+1==4&&y==4&&a[x+1][y]<=1){
                a[x+1][y]++;
                x++;
                //printf("%d %d\n",x,y);
            }
            else if(x+1<=7&&a[x+1][y]==0){
                a[x+1][y]++;
                x++;
                //printf("%d %d\n",x,y);
            }
            //若已到达终点则跳出循环，并设置成功次数+1
            if(x==7&&y==7){
                //printf("到达终点！\n");
                cnt++;
                break;
            }
        }
        if(step==3){//往左走
            //若访问次数尚未达到上限，则更新当前坐标，并设置新坐标访问次数+1
            if(x==4&&y-1==4&&a[x][y-1]<=1){
                a[x][y-1]++;
                y--;
                //printf("%d %d\n",x,y);
            }
            else if(y-1>=1&&a[x][y-1]==0){
                a[x][y-1]++;
                y--;
                //printf("%d %d\n",x,y);
            }
            //若已到达终点则跳出循环，并设置成功次数+1
            if(x==7&&y==7){
                //printf("到达终点！\n");
                cnt++;
                break;
            }
        }
        if(step==4){//往右走
```

```
                //若访问次数尚未达到上限，则更新当前坐标，并设置新坐标访问次数+1
                if(x==4&&y+1==4&&a[x][y+1]<=1){
                    a[x][y+1]++;
                    y++;
                    //printf("%d %d\n",x,y);
                }
                else if(y+1<=7&&a[x][y+1]==0){
                    a[x][y+1]++;
                    y++;
                    //printf("%d %d\n",x,y);
                }
                //若已到达终点则跳出循环，并设置成功次数+1
                if(x==7&&y==7){
                    //printf("到达终点！\n");
                    cnt++;
                    break;
                }
            }
        }
    }
    //计算 20000 次试验中成功走到终点的比率
    printf("%f\n",1.0*cnt/20000);
}
```

结果如下图所示：



Exercise 5.
Given a system made of discrete components with known reliability, what is the reliability of the overall system? For example, suppose we have a system that can be described with the following high-level diagram:
给定一个由已知可靠性的分立元件组成的系统，整个系统的可靠性是多少？ 例如，假设我们有一个可以用下面的高级图来描述的系统：

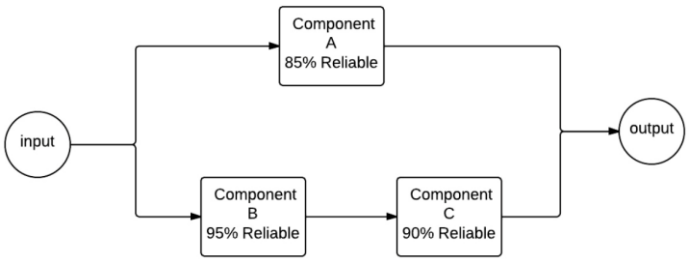


Figure 3 A system made of discrete components.

图 3 由分立元件组成的系统。

When given an input to the system, that input flows through component A or through components B and C, each of which has a certain reliability of correctness. Probability theory tells us the following:
当给定系统输入时，该输入流经组件 A 或组件 B 和 C，每个组件都具有一定的正确性可靠性。 概率论告诉我们：

$$reliability_{BC} = 0.95 * 0.90 = 0.855$$

$$reliability_A = 0.85$$

And the overall reliability of the system is:
系统的整体可靠性为：

$$\begin{aligned} reliability_{sys} &= 1.0 - [(1.0 - 0.85) * (1.0 - 0.855)] \\ &= 0.97825 \end{aligned}$$

Create a simulation of this system where half the time the input travels through component A. To simulate its reliability, generate a number between 0 and 1. If the number is 0.85 or below, component A succeeded, and the system works. The other half of the time, the input would travel on the lower half of the diagram. To simulate this, you will generate two numbers between 0 and 1. If the number for component B is less than 0.95 and the number for component C is less than 0.90, then the system also succeeds. Run many trials to see if you converge on the same reliability as predicted by probability theory.

创建该系统的模拟，其中输入的一半时间通过组件 A。为了模拟其可靠性，生成一个介于 0 和 1 之间的数字。如果数字为 0.85 或更低，则组件 A 成功，系统工作。 另一半时间，输入将在图表的下半部分传播。 为了模拟这一点，您将生成 0 和 1 之间的两个数字。如果组件 B 的数字小于 0.95，组件 C 的数字小于 0.90，则系统也成功。 进行多次试验，看看你是否收敛到与概率论预测的相同的可靠性。

解答过程：

具体代码及注释如下：

```
import random
cnt=0
for i in range(0,10000):
```

```
    a=random.random()
    b=random.random()
    c=random.random()
    if a<=0.85 or (b<=0.95 and c<=0.9):
        cnt=cnt+1
ans=cnt/10000
print(ans)
```

结果如下图所示：

可知模拟结果与概率论理论值 0.97825 几乎一致。