

# 计算物理 A 第五题

杨旭鹏 PB17000234

2019 年秋季

## 1 题目描述

对于球面上均匀分布的随机坐标点，给出它们在  $(x, y)$  平面上投影的几率分布函数。并由此验证 Marsaglia 抽样方法  $x = 2u\sqrt{1 - r^2}, y = 2v\sqrt{1 - r^2}, z = 1 - 2r^2$  确为球面上均匀分布的随机抽样。

## 2 算法

### 2.1 16807 产生器

16807 产生器属于线性同余法产生器的特例。而线性同余法方法为：

$$\begin{aligned} I_{n+1} &= (aI_n + b) \bmod m \\ x_n &= I_n / m \end{aligned} \tag{1}$$

其中整数  $I_i \in [0, m - 1]$ ,  $a, b, m$  为算法中的可调参数，其选取直接影响产生器的质量。选取参数：

$$\begin{cases} a = 7^5 = 16807 \\ b = 0 \\ m = 2^{31} - 1 = 2147483647 \end{cases} \tag{2}$$

即为所谓的 16807 产生器。由于直接利用 1 编写程序时计算  $(aI_n \bmod m)$  时很容易造成数据溢出，故采取 Schrage 方法进行具体编程的实现：

$$aI_n \bmod m = \begin{cases} a(I_n \bmod q) - r[I_n/q], & if \geq 0 \\ a(I_n \bmod q) - r[I_n/q] + m, & otherwise \end{cases} \tag{3}$$

其中  $m = aq + r$ , 即  $q = m/a = 127773$ ,  $r = m \bmod a = 2836$ 。即可利用此方法产生伪随机数序列。

## 2.2 Fibonacci 延迟产生器: Marsaglia 1 号产生器

其思想是用序列中的两个整数进行操作得到后续的整数, 较线性同余法的优势在于它的周期非常长:

$$I_n = I_{n-p} \otimes I_{n-q} \bmod m \quad (4)$$

其中操作符  $\otimes$  可以是: “+”, “-”, “ $\times$ ”, “XOR”。整数对  $[p, q]$  表示延迟, 取值并非按 Fibonacci 数序列, 而是根据统计验证后确认。

在此程序中, 我们使用的是 Fibonacci 延迟产生器的一个特例——Marsaglia 1 号产生器:

$$I_n = \begin{cases} I_{n-p} - I_{n-q}, & if \geq 0 \\ I_{n-p} - I_{n-q} + 1, & otherwise \end{cases} \quad (5)$$

其中的  $[p, q]$  整数对的值选为  $[97, 33]$ , 因此其算法要求存储所有前面的 97 个随机数值。在本程序中, 前 97 个随机数值由在 Linux 系统下访问 “/dev/random” 一次性来得到<sup>1</sup>, <sup>2</sup> 并存储为文件以便后期读取使用 (此文件的原始数据见附录 B 和 C)。

## 2.3 Marsaglia 抽样方法

Marsaglia 抽样方法本质上为一种舍选法, 其思路大致为:

- 1 随机抽样一对均匀分布的随机数,  $(u, v) \in [-1, 1]$  ;
- 2 计算  $r^2 = u^2 + v^2$ , 如果  $r^2 > 1$  则重新抽样直至  $r^2 \leq 1$  ;
- 3 则  $x = 2u\sqrt{1-r^2}, y = 2v\sqrt{1-r^2}, z = 1-2r^2$  即为三维球面上均匀分布的随机点。

---

<sup>1</sup>对于 Linux 下的“/dev/random”文件在 Linux manual page 上有如下解释: The random number generator gathers environmental noise from device drivers and other sources into an entropy pool. The generator also keeps an estimate of the number of bits of noise in the entropy pool. From this entropy pool, random numbers are created. 可以看出此种方法产生的随机数并非来自于算法, 而来自于热噪声。

<sup>2</sup>便于结果的可重复性, 方便调试程序

### 3 在 $(x, y)$ 平面上投影的几率分布函数

在本问题中，需要得到在球面上均匀分布的随机点，即在球面上任意位置，单位面积上点出现的概率为一定值。在某一球面上，半径  $\rho$  值是固定的，自变量仅为角度  $\theta$  和  $\phi$ 。设概率密度函数为  $p(\theta, \phi)$ ，则对于单位面积元上点出现的概率  $dP$  有：

$$const = dP = p(\theta, \phi) d\theta d\phi dS^2 \quad (6)$$

当  $dS^2 = \sin(\theta)d\theta d\phi$  为定值时，我们有  $d\theta \propto \frac{1}{\sin(\theta)}$ 。则为了满足上式，我们有  $p(\theta, \phi) \propto \sin(\theta)$ 。设  $p(\theta, \phi) = k \sin(\theta)$ ，根据密度函数的归一性有

$$1 = \int_0^{2\pi} \int_0^\pi k \sin(\theta) d\theta d\phi \quad (7)$$

得到  $k = \frac{1}{4\pi}$ 。即有  $p(\theta, \phi) = p_\theta(\theta)p_\phi(\phi) = \frac{1}{4\pi} \sin(\theta)$ 。

为了得到投影到  $(x, y)$  平面上的几率，我们设关于  $(x, y)$  的概率密度函数为  $g(x, y)$ ，则有：<sup>3</sup>

$$\begin{cases} g(x, y) dx dy = p(\theta, \phi) d\theta d\phi = \frac{\sin(\theta)}{4\pi} d\theta d\phi \\ x = \sin(\theta) \cos(\phi) \\ y = \sin(\theta) \sin(\phi) \end{cases} \quad (8)$$

我们利用 jacobi 行列式将上式进行换元操作有：

$$\begin{aligned} \frac{\sin(\theta)}{4\pi} d\theta d\phi &= g(x, y) \left| \frac{\partial(x, y)}{\partial(\theta, \phi)} \right| d\theta d\phi \\ &= g(x, y) |\cos(\theta)\sin(\theta)| d\theta d\phi \end{aligned} \quad (9)$$

继续化简上式得到：

$$g(x, y) = \frac{1}{4\pi \sqrt{1 - x^2 - y^2}} \quad (10)$$

### 4 验证 Marsaglia 抽样方法确为球面上均匀分布的随机抽样

我们有：

$$h(u, v) du dv = g(x, y) dx dy = g(x, y) \left| \frac{\partial(x, y)}{\partial(u, v)} \right| du dv \quad (11)$$

---

<sup>3</sup>这里不妨假设球面半径为 1，方便计算

带入  $x, y$  与  $u, v$  的关系式：

$$\begin{cases} x = 2u\sqrt{1-r^2} \\ y = 2v\sqrt{1-r^2} \\ r^2 = u^2 + v^2 \end{cases} \quad (12)$$

得到：

$$g(x, y) = \frac{h(u, v)}{|4 - 8u^2 - 8v^2|} = \frac{1}{4\pi\sqrt{1-x^2-y^2}} \quad (13)$$

其中  $h(u, v) = \frac{1}{\pi}$ 。可知其表达式与球面均匀分布点在  $(x, y)$  平面上的投影的概率密度函数相等。

我们又有：

$$h(u, v)dudv = g(x, z)dx dz = g(x, z) \left| \frac{\partial(x, z)}{\partial(u, v)} \right| dudv \quad (14)$$

带入关系式：

$$\begin{cases} x = 2u\sqrt{1-r^2} \\ z = 1 - 2r^2 \\ r^2 = u^2 + v^2 \end{cases} \quad (15)$$

有：

$$g(x, z) = \frac{h(u, v)}{|8v\sqrt{1-u^2-v^2}|} = \frac{1}{4\pi\sqrt{1-x^2-z^2}} \quad (16)$$

其中  $h(u, v) = \frac{1}{\pi}$ 。可知其表达式与球面均匀分布点在  $(x, z)$  平面上的投影的概率密度函数相等（由于点在球面上均匀分布，则在  $(x, z)$  平面上的投影的概率密度函数与在  $(x, y)$  平面上的投影的概率密度函数相等）。

根据 Marsaglia 抽样方法中  $x, y$  的对称性，球面的对称性，得到 Marsaglia 抽样方法在  $(x, y), (x, z), (y, z)$  平面上的投影的概率密度函数与球面上均匀分布的点在  $(x, y), (x, z), (y, z)$  平面上的投影的概率密度函数相等。则证明了利用 Marsaglia 抽样方法得到的随机点确为球面上均匀分布的随机点。

## 5 程序思路

对于 Marsaglia 抽样方法，其随机数对  $(u, v)$  由 16807 产生器产生。对于 16807 产生器，其程序需要一个种子，在本程序中需要产生两组互不相关的随机数，则需要 2 个“互不相关”的初始种子值。这两个种子值通过一次性读取“/dev/random”文件得到，分别为 34028207 和 1677078722<sup>4</sup>。

---

<sup>4</sup>在这里我们这么做是为了结果的可重复性

产生随机种子后，各自利用递推公式：

$$I_{n+1} = aI_n \bmod m = \begin{cases} a(I_n \bmod q) - r[I_n/q], & if \geq 0 \\ a(I_n \bmod q) - r[I_n/q] + m, & otherwise \\ a = 7^5 = 16807 \\ q = m/a = 127773 \\ r = m \bmod a = 2836 \end{cases} \quad (17)$$

计算随机数列。

对于 Marsaglia 1 号产生器，所需要 97 个初始随机数，得到方式在上文已经提及。

之后将产生的随机数组对  $(u, v)$  使用上文提到的舍取方法得到我们需要的随机点的  $(x, y, z)$  坐标值，存入文件，利用 python 可视化脚本画图。详情请见代码。

## 6 程序使用方法

在运行程序后，会看到请求输入所需总随机点数的提示，按照提示在后面输入所需要的总随机点数，摁回车继续。然后按照程序提示，输入 Marsaglia 1 号产生器所需延迟整数值  $[p, q]$ （最大值不可超过 97，Marsaglia 1 号产生器简易使用默认延迟整数值为 [97, 33]，建议按此输入），按回车继续程序。然后经过计算给出 16807 方法和 Marsaglia 方法产生的随机坐标  $(\theta, \phi)$  分别存为两个文件。程序输出完这些后会自动退出。

```
|请输入您所需要的总点数: 1000000
请输入Marsaglia方法的延迟整数对[p,q](建议输入97,33)(英文逗号隔开): 97,33
您输入的参数已接受，正在计算请稍后~
Program ended with exit code: 0
```

图 1：一个典型程序的运行过程示例

## 7 程序结果与讨论

当输入一些不同的点数时，得到如下结果：

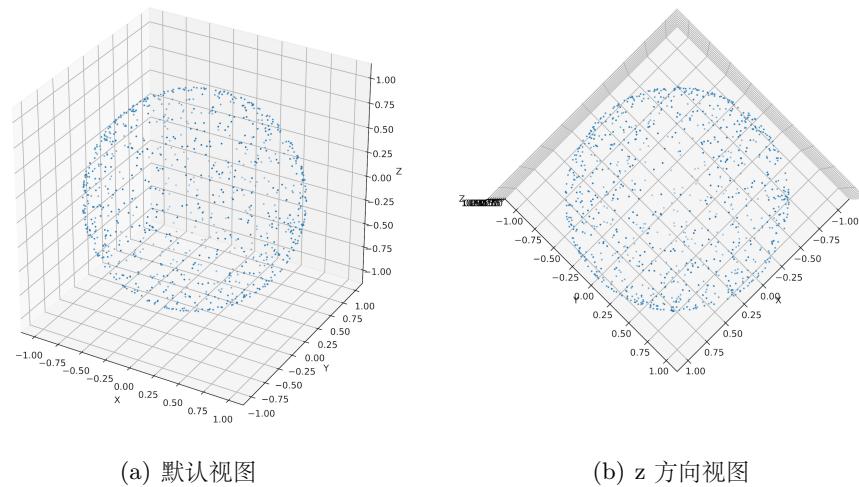


图 2：球面上均匀分布的 1000 随机点

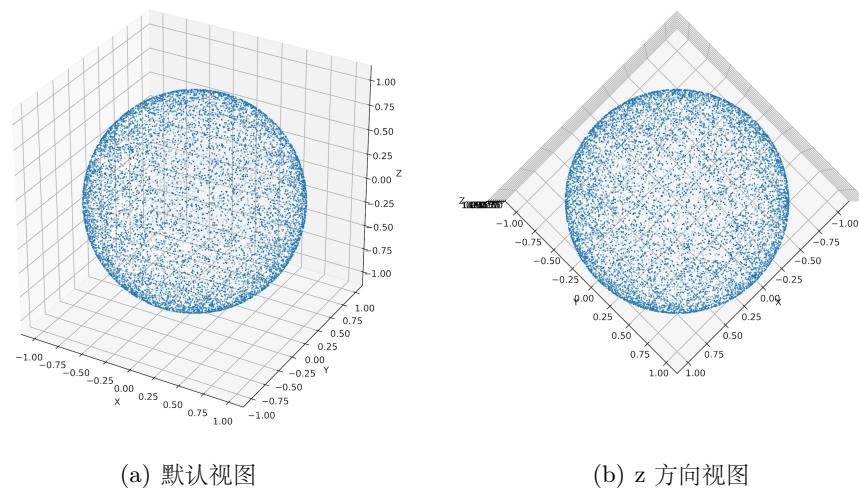


图 3：球面上均匀分布的  $10^4$  随机点

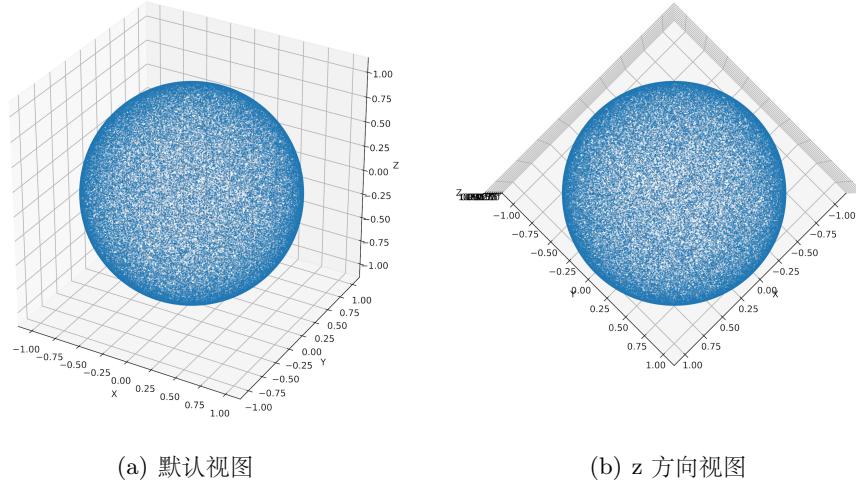


图 4: 球面上均匀分布的  $10^5$  随机点

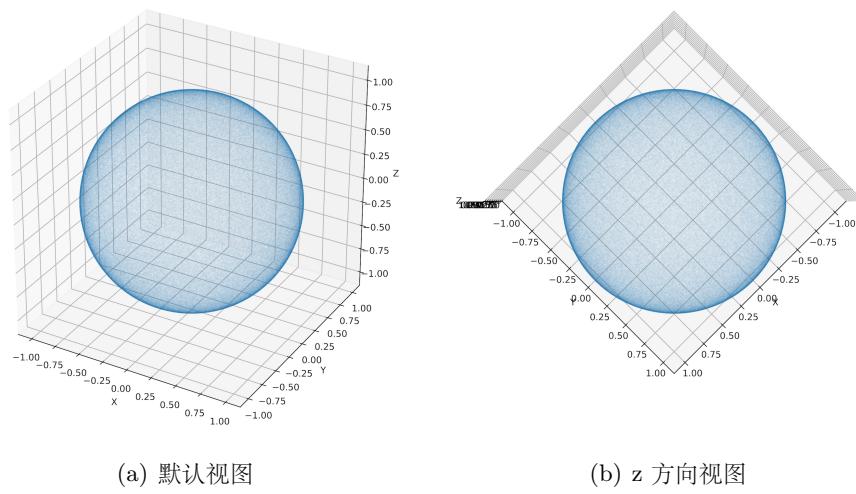
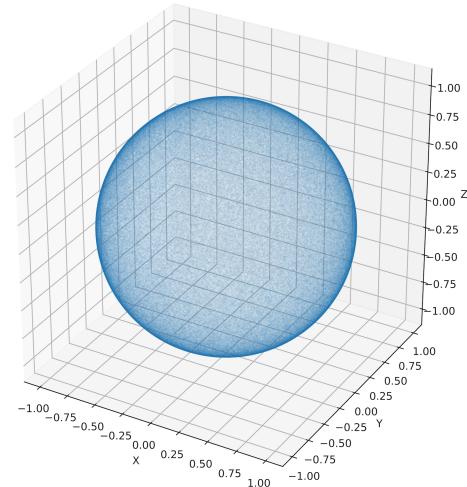
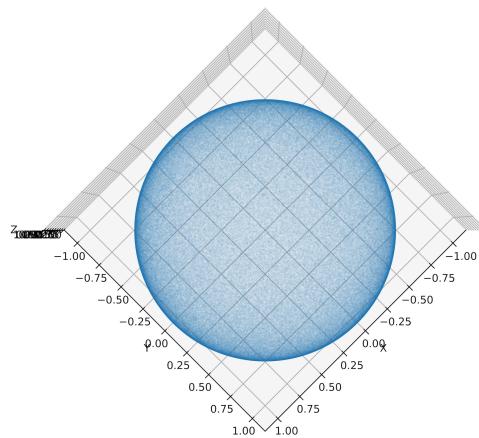


图 5: 球面上均匀分布的  $10^6$  随机点

由上述结果可以看出当  $N \leq 10^6$  时, 利用 16807 和 Marsaglia 抽取相结合的方法产生的随机点在球面上的分布还算均匀, 几乎看不出什么条带结构和规则网格结构等有明显规律性的结构。我们换取 Marsaglia 1 号产生器和 Marsaglia 抽取法相结合的方式进行试验, 得到如下结果:



(a) 默认视图



(b) z 方向视图

图 6: Marsaglia 1 号产生器产生的球面上均匀分布的  $10^6$  随机点

可以看出利用 marsaglia 1 号产生器与 Marsaglia 抽取法相结合的方式产生的球面上均匀分布的随机点在  $N = 10^6$  时看不出什么有规律性的结构，仍可认为是比较均匀的。

将  $(x, y)$  平面上的区域  $[-1, 1] \times [-1, 1]$  划分为  $50 \times 50$  个小格，统计数值抽样后，每个小格内出现点的概率得到如下结果<sup>5</sup>：

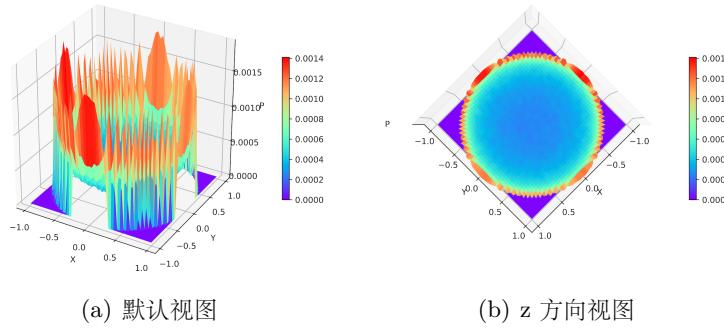


图 7：16807 号产生器产生的球面上均匀分布的  $10^6$  随机点在  $(x, y)$  平面上的投影概率统计

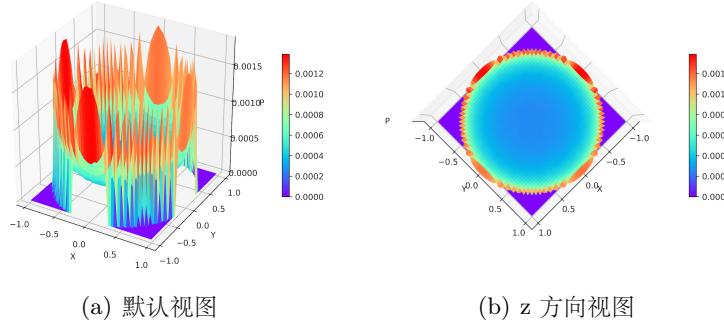


图 8：16807 号产生器产生的球面上均匀分布的  $10^7$  随机点在  $(x, y)$  平面上的投影概率统计

---

<sup>5</sup>可视化结果通过 python 脚本得到，由于点数相对于剖分区域还是比较少，故在靠近边缘处出现“毛刺”

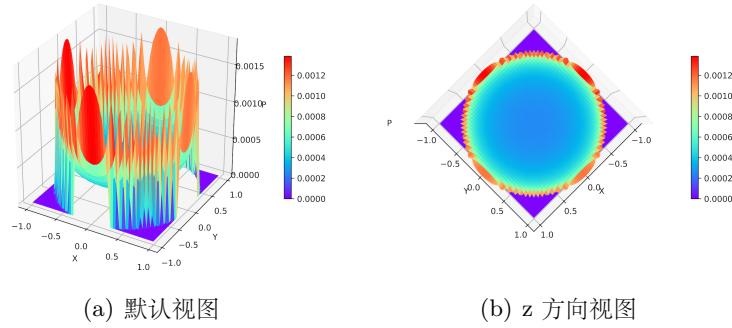


图 9: 16807 号产生器产生的球面上均匀分布的  $10^8$  随机点在  $(x, y)$  平面上的投影概率统计

对于  $N = 10^8$  的情况, 我们还做了其在  $(x, y)$  平面上的投影概率统计与理论概率统计的差值的图像:

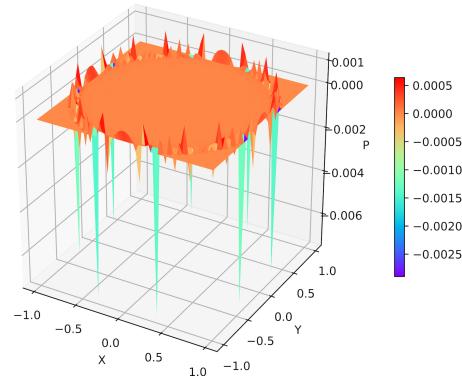


图 10:  $N = 10^8$  时其  $(x, y)$  平面上的投影概率统计与理论概率统计的差值

其中理论概率统计由概率密度函数在剖分小区域中心的值并归一化得到，其为近似值。可看出两者的差除了在边缘附件比较大之外，其他地方差值基本为 0. 由于此概率密度分布在边缘处发散，故很难用数值的方法得到与理论值相差很小的结果，故此结果已经能够在一定程度上说明抽样得到的概率密度与理论密度一致。

通过上节的理论推导与本节产生的图片，都能说明 Marsaglia 抽样方法产生的为球面上均匀分布的随机点。

## 8 心得与体会

通过此次作业结果，验证了 Marsaglia 抽样方法确实能抽取球面上均匀分布的点。

通过编程作业，也更加熟悉了一些 C 语言和 L<sup>A</sup>T<sub>E</sub>X。

## 9 附录

### A 产生随机数种子的 C 语言程序

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 //利用/dev/random 产生随机种子
5
6 int my_randomseed(int seed[],int n){
7
8 //seed为存储随机种子的数据, n为所需要种子的个数
9
10 FILE * fp1 = fopen("/dev/random","r");
11
12 for(int i=0;i<n;i++){
13     fread(&seed[i], 1, sizeof(int), fp1);
14 }
15
16 fclose(fp1);
17
18 return 0;
19 }
20
21 int main(){
22 int ranI[97];
23
24 my_randomseed(ranI,97);
25
26 FILE * fp;
27
28 fp = fopen("ranI.txt","w+");
29
30 for(int i=0;i<96;i++){
31     fprintf(fp,"%d,",ranI[i]);
32 }
```

```
33
34     fprintf(fp, "%d", ranI[96]); //最后一个数据后不加 ","
35
36     fclose(fp);
37
38     return 0;
39
40 }
```

## B 由上述程序产生的随机数文件 1

809576131,-1025892587,558213681,2056216731,-1652403892,-849346475,  
-947398527,-2082766203,131013461,753913035,-634671236,995067675,-1025241712,  
-57745909,187349022,-1487194468,174292363,-1523076266,-1381700026,1751317799,  
812085654,-384904015,935241924,-2120102602,-1176252493,2084189454,-1308162651,  
-721701972,1776417975,-1953284964,-1385173859,-1156931024,-272454405,-1527712783,  
1040918716,966408491,-899150905,-1102248190,363327930,1940215723,924796768,861925965,  
1030548881,1694050903,-1841641721,-5770120,448493076,1650846131,1012895949,  
-2135767741,1924774890,-885687700,-219340479,-1417922044,-1101149761,1198019956,  
-1522975896,842839847,2023490906,-1533950707,1267038558,1823349522,-1476843016,  
786347740,1101867560,110498268,-1018391091,-1677263454,2117579346,-306068839,  
569112568,-1031628950,1399939025,1102660725,-526224915,212410442,-968658785,  
1808251768,166339465,-347114238,-1130541052,450232354,-684115741,-1313290066,  
140916175,-1221648810,-1079940150,2079743593,414264974,-1639098679,142289346,  
-2080694919,-345845707,-1201999899,1632261567,98605000,-1843368371

## C 由上述程序产生的随机数文件 2

-753424879,1872953453,608893860,597180613,2049110889,2140413464,1070133919,  
-256067828,-1191457009,-574707053,-1410977199,1133706033,-404358309,1141209606,  
-1604908044,2123698944,1542004389,894557832,-1972430293,-1544149044,-155439154,  
2108522693,1220970092,697871479,-1502727159,1427890000,-1629207648,-132267497,  
-749675637,1645721388,-1858854966,-2111533027,-151887203,-1412609570,-876296650,  
-1227304997,701135804,1224292810,1805574955,-955336658,619132236,1948713847,

1737807285,845890383,-1491155579,-650075247,-1329866879,1910267507,-1879619968,  
289013714,-24743716,1328343531,656220109,2045662973,-137884045,-1218521402,  
1439903068,1203175289,-2098367843,1830857371,106846260,-1413985256,584445562,  
-1141203261,-1584804986,-2109822000,-1573941759,1728136137,-54161293,-1017405251,  
1652078157,1750018552,1594371093,1725375480,1445249495,-1114348371,-237895730,  
1481242551,-1581295506,-970746651,397491742,462545489,-436989135,-1245824880,  
-945627835,773430348,429365809,-108076855,620101426,1362478058,-923198353,  
-1447637852,-1914066341,-1674823883,824439501,-461869274,-634359412

## D C 语言程序源码

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #include <math.h>
5 #define a 16807
6 #define b 0
7 #define m 2147483647
8 #define r (m%a)
9 #define q (m/a)
10 #define Pi 3.1415926
11
12
13
14 int my_filereader_int(char str[],int num[],int n){
15     FILE * fp;
16     fp = fopen(str,"r");
17
18     for(int i=0;i<n-1;i++)
19     {
20         fscanf(fp,"%d,",&num[i]);
21
22     }
23     fscanf(fp,"%d",&num[n-1]); //最后一个数据后不加 ","
24     fclose(fp);
```

```

25     return 0;
26 }
27
28 int my_filewriter(char str[],double num[],int n){
29     FILE * fp;
30     fp = fopen(str,"w+");
31
32     for(int i=0;i<n;i++)
33     {
34         if (i == (n-1)){
35             fprintf(fp,"%lf",num[i]);
36             break;
37         }
38         fprintf(fp,"%lf,",num[i]);
39     }
40     fclose(fp);
41     return 0;
42 }
43
44
45
46
47
48 int my_marsaglia_16807_sphere(int seed[],double ranu[],double
49     ranv[],int n){
50     for (int j = 0; j <= n; ) {
51         if(seed[0] == m-1){
52             if(a >= b){ //由于Schrage方法只对z in
53                 (0,m-1)成立，故这里要讨论z == m-1的情况
54                 seed[0] = m + (b-a) % m;
55             }
56             else seed[0] = (b-a) % m;
57         }
58         if(seed[1] == m-1){
59             if(a >= b){ //由于Schrage方法只对z in

```

```

60     }
61     else  seed[1] = (b-a) % m;
62
63 }
64 seed[0] = ((a * (seed[0] % q) - r * (seed[0] / q)) + b % m )
65             % m;
66 seed[1] = ((a * (seed[1] % q) - r * (seed[1] / q)) + b % m )
67             % m;
68 if (seed[0] >= 0) {
69     ranu[j] = 2*(seed[0] / (double)m) -1;
70 }
71 else{
72     ranu[j] = 2*((seed[0] + m) / (double)m)-1 ;
73 }
74 if (seed[1] >= 0) {
75     ranv[j] = 2*(seed[1] / (double)m)-1;
76 }
77 else{
78     ranv[j] = 2*((seed[1] + m) / (double)m)-1;
79 }
80 if( pow(ranu[j],2) + pow(ranv[j],2) < 1 ) j++;
81 //第j+1个元素满足舍选法条件后可继续向后写入数据，否则下一循环重新覆盖此数据
82 }
83 return 0;
84 }
85
86
87 //Fibonacci延迟器
88 int my_fibonacci_sphere(double ranu[],double ranv[],int ranI1[],int
89 ranI2[],int n,int o,int p){
90     int temp1;
91     int temp2;
92     int j = 0;
93     for (int i = p; j < n; i++) { //ranI[i % p]存放的为第i项

```

```

94     temp1 = ranI1[i % p] - ranI1[(i % p+(p-o)) % p]; //递推式
95     temp2 = ranI2[i % p] - ranI2[(i % p+(p-o)) % p]; //递推式
96
97     if(temp1 >= 0) ranI1[i % p] = temp1 ;
98     else ranI1[i % p] = temp1 + 1;
99         //递推得到的新结果放入ranI中i%p一项
100
101    if(temp2 >= 0) ranI2[i % p] = temp2 ;
102    else ranI2[i % p] = temp2 + 1;
103        //递推得到的新结果放入ranI中i%p一项
104
105    if (ranI1[i % p] >= 0) { //计算ran
106        ranu[j] = 2*(ranI1[i % p] / (double) m)-1;
107    } else {
108        ranu[j] = 2*((ranI1[i % p] + m) / (double) m)-1;
109    }
110
111    if (ranI2[i % p] >= 0) { //计算ran
112        ranv[j] = 2*(ranI2[i % p] / (double) m)-1;
113    } else {
114        ranv[j] = 2*((ranI2[i % p] + m) / (double) m)-1;
115    }
116
117    if ( pow(ranu[j],2) + pow(ranv[j],2) < 1 ){
118        j++;
119    }
120
121    return 0;
122
123
124
125 int my_count(double ranx[],double rany[],double nk[][51],int n,int
126 k){
127     int x;
128     int y;

```

```

129  for(int j=0;j<=k;j++){
130      for(int e=0;e<=k;e++){
131          nk[j][e] = 0; //初始化数组
132      }
133  }
134 }
135 for(int i=0;i<n;i++) {//计算各个实际频数
136     x = floor((ranx[i]+1)/2*k);
137     y = floor((rany[i]+1)/2*k);
138     nk[x][y] += (double)1/n;
139 }
140 return 0;
141 }
142
143
144
145
146 int main() {
147     int N; //产生总点数
148     char str[50];
149     printf("请输入您所需要的总点数: ");
150     while (!scanf("%d",&N)){ //简单的输入检查
151         gets(str);
152         printf("\nInput error,please try again\n");
153         printf("请输入您所需要的总点数: ");
154     }
155
156     int o,p;
157     inputpq:printf("请输入Marsaglia方法的延迟整数对[p,q](建议输入97,33)(英文逗号隔开): ");
158     while (!scanf("%d,%d",&o,&p)) { //相对简单的输入检查
159         gets(str);
160         printf("\nInput error,please try again\n");
161         printf("请输入Marsaglia方法的延迟整数对[p,q](建议值97,33)(英文逗号隔开): ");
162     }
163
164     int temp;
165     if(o > p){ //使 p > o

```

```

167     temp = o;
168     o = p;
169     p=temp;
170 }
171 if( N-p < 3 ) { //排除N相对于p,q太小的情况发生
172     printf("输入的p,q值相对于总点数太大了, 请重试!\n");
173     goto inputpq;
174 }
175
176
177 if(N > 100000) printf("您输入的参数已接受, 正在计算请稍后~\n");
178
179 int seed[2] = {34028207,1677078722};
180
181 double * ranu = malloc(sizeof(double)*(N+1));
182 double * ranv = malloc(sizeof(double)*(N+1));
183 double * ranx = malloc(sizeof(double)*N);
184 double * rany = malloc(sizeof(double)*N);
185 double * ranz = malloc(sizeof(double)*N);
186
187 my_marsaglia_16807_sphere(seed, ranu, ranv, N);
188
189 for (int i = 0;i<N;i++){ //计算x,y,z坐标值
190     ranx[i] = 2*ranu[i]*pow(1-pow(ranu[i],2)-pow(ranv[i],2),0.5);
191     rany[i] = 2*ranv[i]*pow(1-pow(ranu[i],2)-pow(ranv[i],2),0.5);
192     ranz[i] = 1 - 2*( pow(ranu[i],2) + pow(ranv[i],2) );
193 }
194
195 double nk[51][51];
196 my_count(ranx,rany,nk,N,50);
//统计在(x,y)平面上的y投影的分布, 将[-1,1]*[-1,1]区间剖分为50*50个小区间统计
197
198 FILE * fp;
199 fp = fopen("hist.txt","w+"); //存放在(x,y)平面上的投影的分布
200
201 for(int i=0;i<50;i++){
//统计16807方法产生的点在(x,y)平面上的投影的分布
202     for(int j=0;j<50;j++){

```

```

203         if( i==49 && j==49 ){
204             fprintf(fp,"%lf",nk[j][i]);
205             break;
206         }
207         fprintf(fp,"%lf ",nk[j][i]);
208     }
209     fprintf(fp,"\n");
210 }
211 fclose(fp);
212
213
214 my_filewriter("ranx_16807.dat", ranx, N);
215 my_filewriter("rany_16807.dat", rany, N);
216 my_filewriter("ranz_16807.dat", ranz, N);
217 /* 还可利用fibonacci产生器产生随机数方法抽样，此处略去
218 int *ranI1 = malloc(sizeof(int) * p); //用来存放递推数列
219 int *ranI2 = malloc(sizeof(int) * p);
220 my_filereader_int("ranI-1.txt",ranI1,p);
221         //读取之前利用"/dev/random"产生的初始随机数组
222 my_filereader_int("ranI-2.txt",ranI2,p);
223
224 my_fibonacci_sphere(ranu,ranv,ranI1,ranI2,N+1,o,p);
225 for (int i = 0;i<N;i++){
226     ranx[i] = 2*ranu[i]*pow(1-pow(ranu[i],2)-pow(ranv[i],2),0.5);
227     rany[i] = 2*ranv[i]*pow(1-pow(ranu[i],2)-pow(ranv[i],2),0.5);
228     ranz[i] = 1 - 2*( pow(ranu[i],2) + pow(ranv[i],2) );
229 }
230 my_filewriter("ranx_fibo.dat", ranx, N);
231 my_filewriter("rany_fibo.dat", rany, N);
232 my_filewriter("ranz_fibo.dat", ranz, N);
233 */
234 return 0;
235 }
```

## E 计算理论概率分布的 C 程序源码

```
1 #include <stdio.h>
2 #include <math.h>
3 #define PI 3.1415926535
4
5
6 int my_filewriter_double(char str[],double num[],int n){
7     FILE * fp;
8     fp = fopen(str,"w+");
9
10    for(int i=0;i<n;i++){
11        if (i == (n-1)){
12            fprintf(fp,"%lf",num[i]);
13            break;
14        }
15        fprintf(fp,"%lf ",num[i]);
16    }
17
18    fclose(fp);
19    return 0;
20}
21
22
23
24
25 int main(){
26     int i = 0;
27     double sum = 0;
28     double ran[2500];
29     int flag = 0;
30     for (double y=-0.98;y<1;y += 0.04){
31         for(double x = -0.98;x <1;x += 0.04){
32             flag++;
33             if(x*x+y*y >=1){
34                 ran[i] = 0;
35             }
36         }
37     }
38     my_filewriter_double("out.txt",ran,2500);
39 }
```

```

36     else{
37         ran[i] = 1/(4*PI*sqrt(1-x*x-y*y));
38         sum += 1/(4*PI*sqrt(1-x*x-y*y));
39     }
40     i++;
41
42 }
43 }
44 printf("sum = %lf flag = %d\n",sum,flag);
45 for(int i =0;i<2500;i++){
46     ran[i] = ran[i]/sum;
47 }
48 my_filewriter_double("op.txt", ran, 2500);
49 return 0;
50 }
```

## F 可视化绘图 python 程序源码

```

1
2 import matplotlib.pyplot as plt
3 from mpl_toolkits.mplot3d import Axes3D
4 import numpy as np
5 from IPython.core.pylabtools import figsize # import figsize
6 figsize(7, 7) # 设置 figsize
7 plt.rcParams['savefig.dpi'] = 1000 #图片像素
8 plt.rcParams['figure.dpi'] = 1000 #分辨率
9 # 默认的像素: [6.0,4.0], 分辨率为100, 图片尺寸为 600&400
10
11 fig = plt.figure()
12 ax = Axes3D(fig)
13 X = []
14 Y = []
15 Z = []
16
17 with open('problem 5/ranx_fibo.dat', 'r') as f:
```

```

18     while True:
19         lines = f.readline() # 整行读取数据
20         if not lines:
21             break
22             pass
23         tmp = [float(i) for i in lines.split(',')]] #
24             将整行数据分割处理。
25         X.append(tmp) # 添加新读取的数据
26             pass
27         X = np.array(X) # 将数据从list类型转换为array类型。
28             pass
29
30     with open('problem 5/rany_fibo.dat', 'r') as f:
31         while True:
32             lines = f.readline() # 整行读取数据
33             if not lines:
34                 break
35                 pass
36             tmp = [float(i) for i in lines.split(',')]] #
37                 将整行数据分割处理。
38             Y.append(tmp) # 添加新读取的数据
39             pass
40         Y = np.array(Y) # 将数据从list类型转换为array类型。
41             pass
42     with open('problem 5/ranz_fibo.dat', 'r') as f:
43         while True:
44             lines = f.readline() # 整行读取数据
45             if not lines:
46                 break
47                 pass
48             tmp = [float(i) for i in lines.split(',')]] #
49                 将整行数据分割处理。
50             Z.append(tmp) # 添加新读取的数据
51             pass
52         Z = np.array(Z) # 将数据从list类型转换为array类型。
53             pass

```

```

53
54
55
56
57 # 绘制散点图
58 ax.scatter(X, Y, Z,s=0.0001) #画散点图
59 ax.view_init(elev=90,azim=45)
60 ax.set_aspect('equal') #设置三个坐标轴比例尺相同
61 ax.set_xlabel('X')
62 ax.set_ylabel('Y')
63 ax.set_zlabel('Z')
64 plt.savefig("1.png")
65
66 Z = []
67 with open('problem 5/hist.txt', 'r') as f:
68     while True:
69         lines = f.readline() # 整行读取数据
70         if not lines:
71             break
72         temp = [float(i) for i in lines.split()] #
    将整行数据分割处理，如果分割符是空格，括号里就不用传入参数，如果是逗号，  

    则传入 '，' 字符。
73         Z.append(temp)
74     Z = np.array(Z) # 将数据从list类型转换为array类型。
75 pass
76
77
78 Z = Z.reshape(50,50)
79
80
81 X = np.linspace(-0.99, 0.99, 50)
82 Y = np.linspace(-0.99, 0.99, 50)
83 x, y = np.meshgrid(X, Y)
84 surf = ax.plot_surface(x, y, Z, rstride=1, cstride=1,
    cmap='rainbow')
85
86 ax.set_aspect('equal') #设置三个坐标轴比例尺相同
87 ax.set_xlabel('X')

```

```
88 ax.set_ylabel('Y')
89 ax.set_zlabel('P')
90 ax.set_zticks([0, 0.0005, 0.001, 0.0015])
91 #ax.set_zticks([])
92 ax.set_yticks([-1, -0.5, 0, 0.5, 1])
93 ax.set_xticks([-1, -0.5, 0, 0.5, 1])
94 plt.colorbar(surf, shrink = 0.5)
95 #ax.view_init(elev=90,azim=45)
96 plt.savefig("p.png")
```