计算物理 A 第一题

杨旭鹏 PB17000234

2019 年秋季

1 题目描述

用 Schrage 方法编写随机数子程序，用连续两个随机数作为点的坐标值绘出若干点的平面分布图。再用 $< x^k >$ 测试均匀性 (取不同量级的 N 值，讨论偏差与 N 的关系)、$C(l)$ 测试其 2 维独立性 (总点数 $N > 10^7$)。

2 算法及程序思路

2.1 算法

利用均匀随机数产生器中最简单的线性同余法实现随机数的产生，其方法为：

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

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。本程序使用的为具有“最低标准”的 16807 产生器，即：

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

由于直接利用1编写程序时计算 $(aI_n\ mod\ m)$ 时很容易造成数据溢出，故采取作业要求的 Schrage 方法进行具体编程的实现：

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

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

2.2 程序思路

首先我们需要产生一个随机数种子，最简单的方法便是人工赋值。也可读取相应的时间做种子例如在 C 语言下，可读取时间放入 srand() 函数中，并用 rand() 函数产生种子；在 Linux 系统下可读取"/dev/random" 文件来得到相对"真"的随机数作为种子等等。[1]在此程序中，我们采用最为简单的人工赋值法，这样做的好处是数据具有可重复性，方便后期的 Debug。[2]

产生随机种子后，利用递推公式：

$$I_{n+1} = aI_n\ mod\ m = \begin{cases} a(I_n\ mod\ q) - r[I_n/q], & if \geq 0 \\ a(I_n\ mod\ q) - r[I_n/q] + m, & otherwise \\ a = 7^5 = 16807 & \\ q = m/a = 127773 & \\ r = m\ mod\ a = 2836 & \end{cases} \tag{4}$$

计算随机数列。

得到随机数列后，将奇数项放入一个数组作为坐标 x 的值，偶数项放入另一个数组作为坐标 y 的值，然后将两个数组的数据输出到.dat 文件中，以便后期画图处理。[3]数据的 $< x^k >$ 测试以及 $C(l)$ 测试，也相应的进行，详情请见代码。

3 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。然后经过计算给出所有随机数点的线性相关系数 $C(l)$ 计算结果并输出随机数.dat 文件至当前文件夹。然后按照程序提示，输入所需计算的 $< x^k >$ 中 k 的最大值，按回车继续程序，

[1]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。

[2]在原版程序中，采取的是读取"/dev/random"文件方案

[3]本程序利用 python 脚本，使用 matplotlib 绘制图样

会自动在屏幕上显示从总点数次方量级的 10% 到 100% ，从 $k = 1$ 到您输入的最大值的 $< x^k >$ 和 $< y^k >$ 的计算值及均匀分布的理论值。程序输出完这些后会自动退出。

```
请输入您所需要的总点数: 20000000
请输入您计算线性相关系数C(l)时参数l的最大值: 5
您输入的参数已接受，正在计算请稍等片刻~
所产生的 20000000 个随机数的线性相关系数 C(1) = 6.452935E-05
所产生的 20000000 个随机数的线性相关系数 C(2) = -1.839874E-04
所产生的 20000000 个随机数的线性相关系数 C(3) = 2.321942E-04
所产生的 20000000 个随机数的线性相关系数 C(4) = 2.054347E-04
所产生的 20000000 个随机数的线性相关系数 C(5) = 5.802567E-05
Please input the max value of k in <x^k>:4
以下是<x^k>的均匀性测试:
N = 10       <x^1> = 0.351409  with theoretical number of 0.500000,the difference is 0.148591
N = 10       <x^2> = 0.216174  with theoretical number of 0.333333,the difference is 0.117159
N = 10       <x^3> = 0.152751  with theoretical number of 0.250000,the difference is 0.097249
N = 10       <x^4> = 0.117402  with theoretical number of 0.200000,the difference is 0.082598
N = 100      <x^1> = 0.493705  with theoretical number of 0.500000,the difference is 0.006295
N = 100      <x^2> = 0.327029  with theoretical number of 0.333333,the difference is 0.006304
N = 100      <x^3> = 0.243376  with theoretical number of 0.250000,the difference is 0.006624
N = 100      <x^4> = 0.193030  with theoretical number of 0.200000,the difference is 0.006970
N = 1000     <x^1> = 0.491958  with theoretical number of 0.500000,the difference is 0.008042
N = 1000     <x^2> = 0.323331  with theoretical number of 0.333333,the difference is 0.010002
N = 1000     <x^3> = 0.239747  with theoretical number of 0.250000,the difference is 0.010253
N = 1000     <x^4> = 0.189955  with theoretical number of 0.200000,the difference is 0.010045
N = 10000    <x^1> = 0.500045  with theoretical number of 0.500000,the difference is 0.000045
N = 10000    <x^2> = 0.333121  with theoretical number of 0.333333,the difference is 0.000212
N = 10000    <x^3> = 0.249802  with theoretical number of 0.250000,the difference is 0.000198
N = 10000    <x^4> = 0.199907  with theoretical number of 0.200000,the difference is 0.000093
N = 100000   <x^1> = 0.499777  with theoretical number of 0.500000,the difference is 0.000223
N = 100000   <x^2> = 0.332932  with theoretical number of 0.333333,the difference is 0.000401
N = 100000   <x^3> = 0.249505  with theoretical number of 0.250000,the difference is 0.000495
N = 100000   <x^4> = 0.199445  with theoretical number of 0.200000,the difference is 0.000555
N = 1000000  <x^1> = 0.500341  with theoretical number of 0.500000,the difference is 0.000341
N = 1000000  <x^2> = 0.333583  with theoretical number of 0.333333,the difference is 0.000249
N = 1000000  <x^3> = 0.250158  with theoretical number of 0.250000,the difference is 0.000158
N = 1000000  <x^4> = 0.200097  with theoretical number of 0.200000,the difference is 0.000097
N = 10000000 <x^1> = 0.499880  with theoretical number of 0.500000,the difference is 0.000120
N = 10000000 <x^2> = 0.333173  with theoretical number of 0.333333,the difference is 0.000161
N = 10000000 <x^3> = 0.249832  with theoretical number of 0.250000,the difference is 0.000168
N = 10000000 <x^4> = 0.199835  with theoretical number of 0.200000,the difference is 0.000165

进程已结束，退出代码 0
```

图 1: 一个典型程序的运行过程示例

4 程序结果与分析

当设定初始种子值为“1”时，利用 python 脚本我们得到如下可视化后的数据:

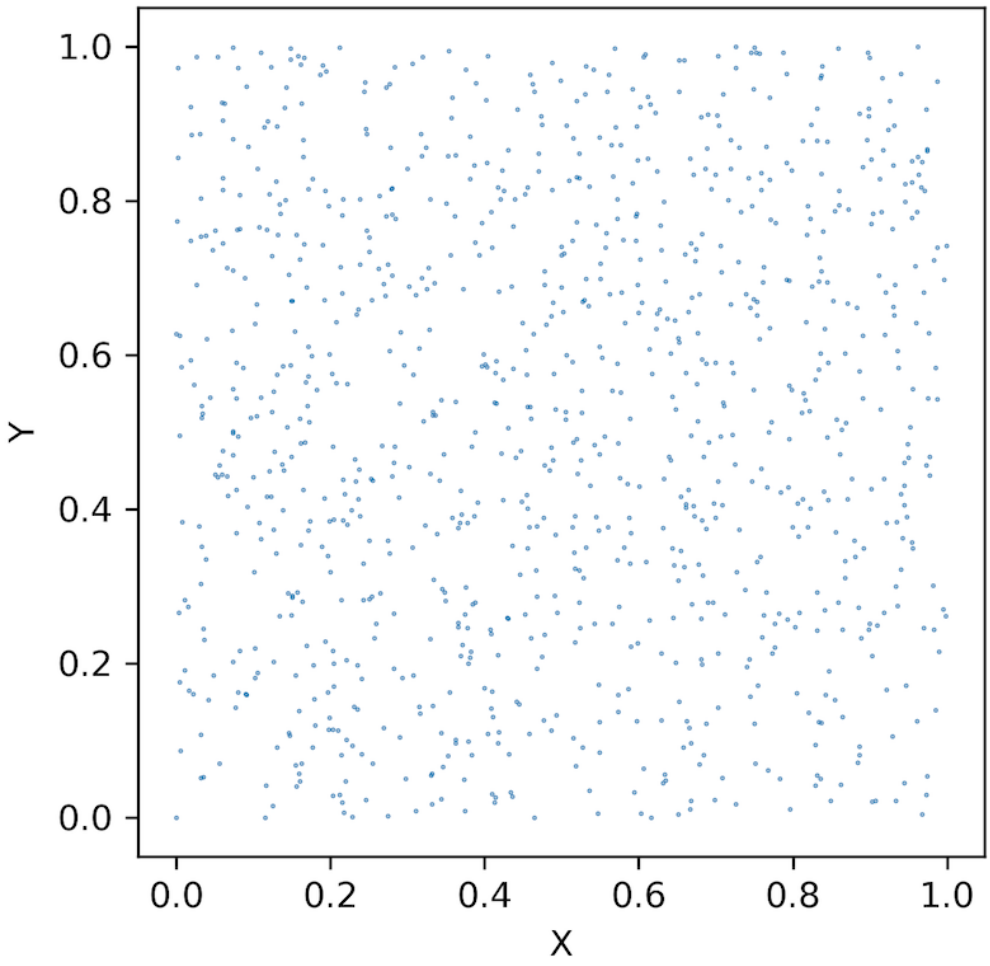

图 2: 由种子“1”生成的 1×10^3 个均匀分布随机点

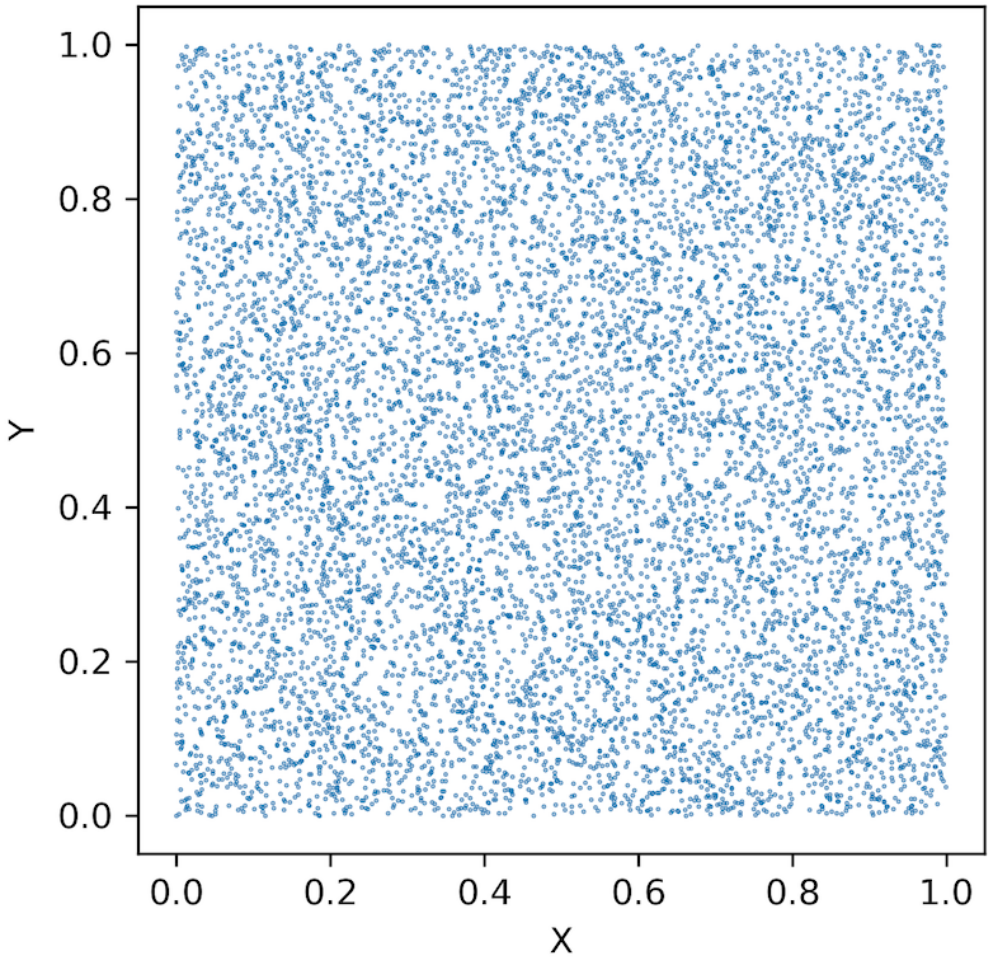

图 3: 由种子“1”生成的 1×10^4 个均匀分布随机点

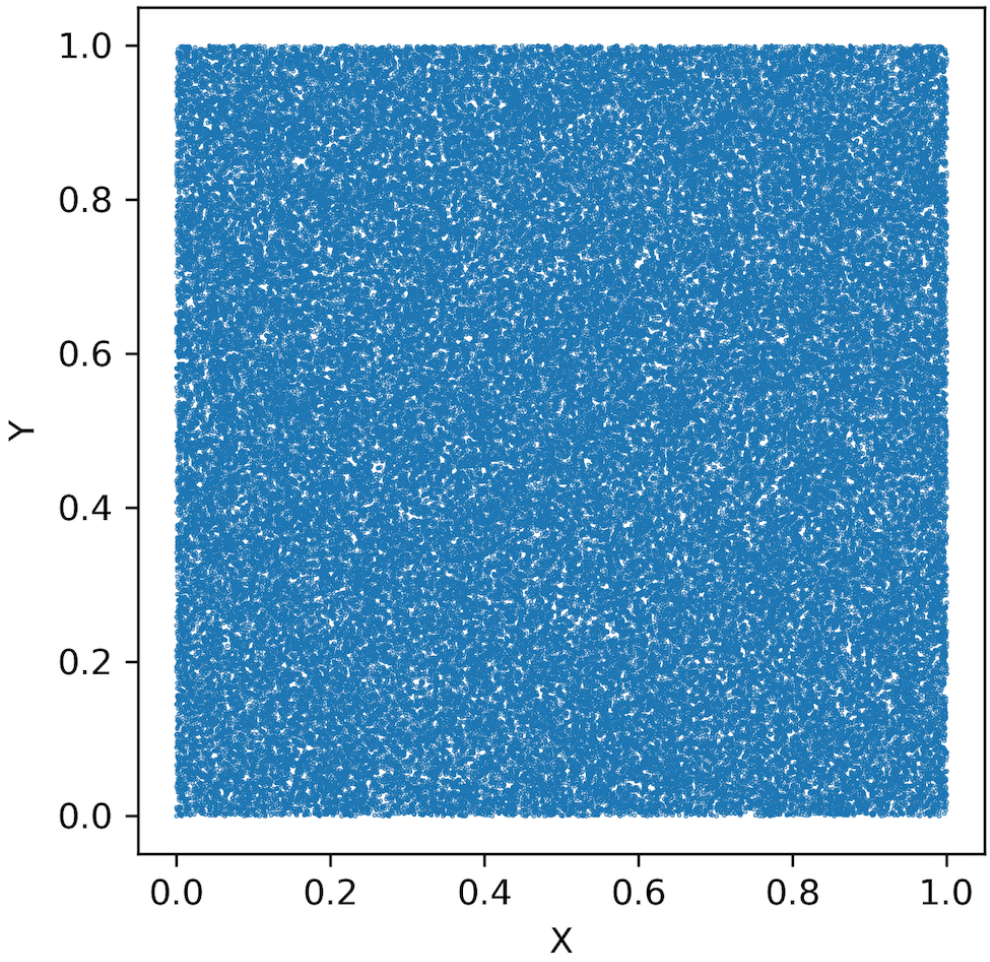

图 4: 由种子“1”生成的 1×10^5 个均匀分布随机点

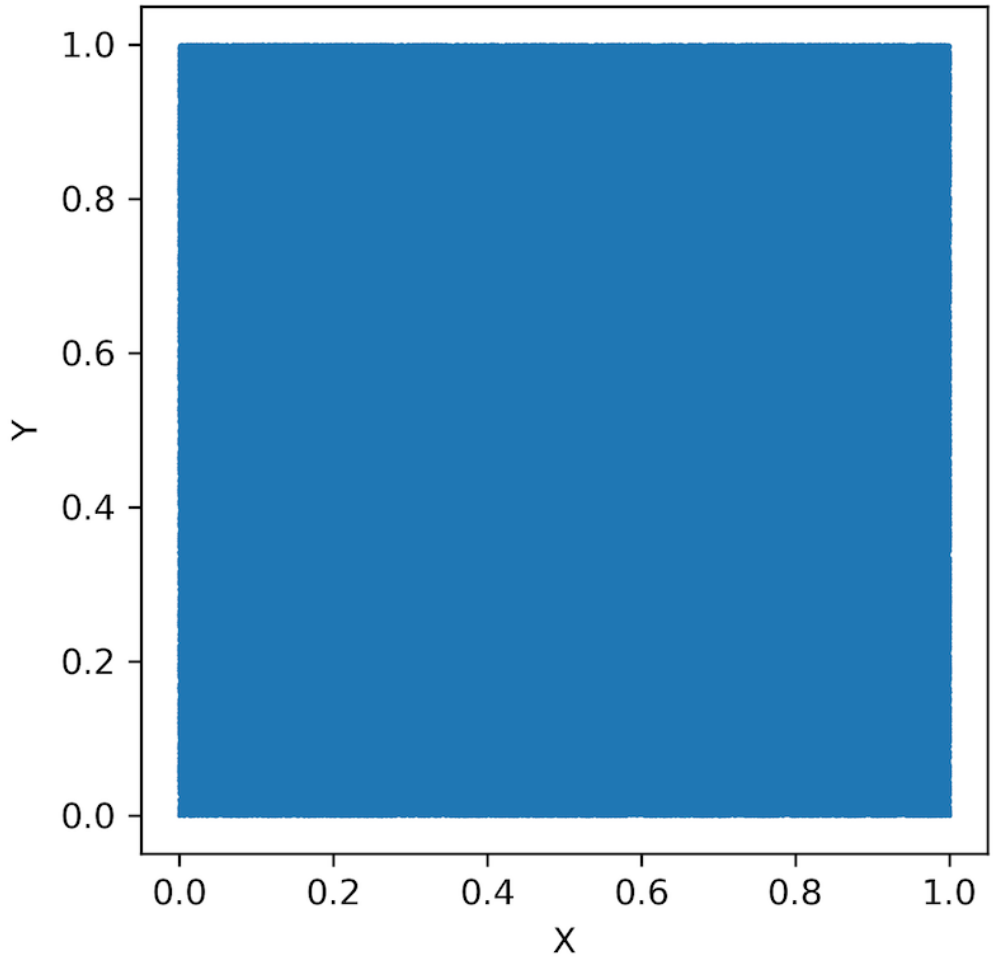

图 5: 由种子“1”生成的 1×10^6 个均匀分布随机点

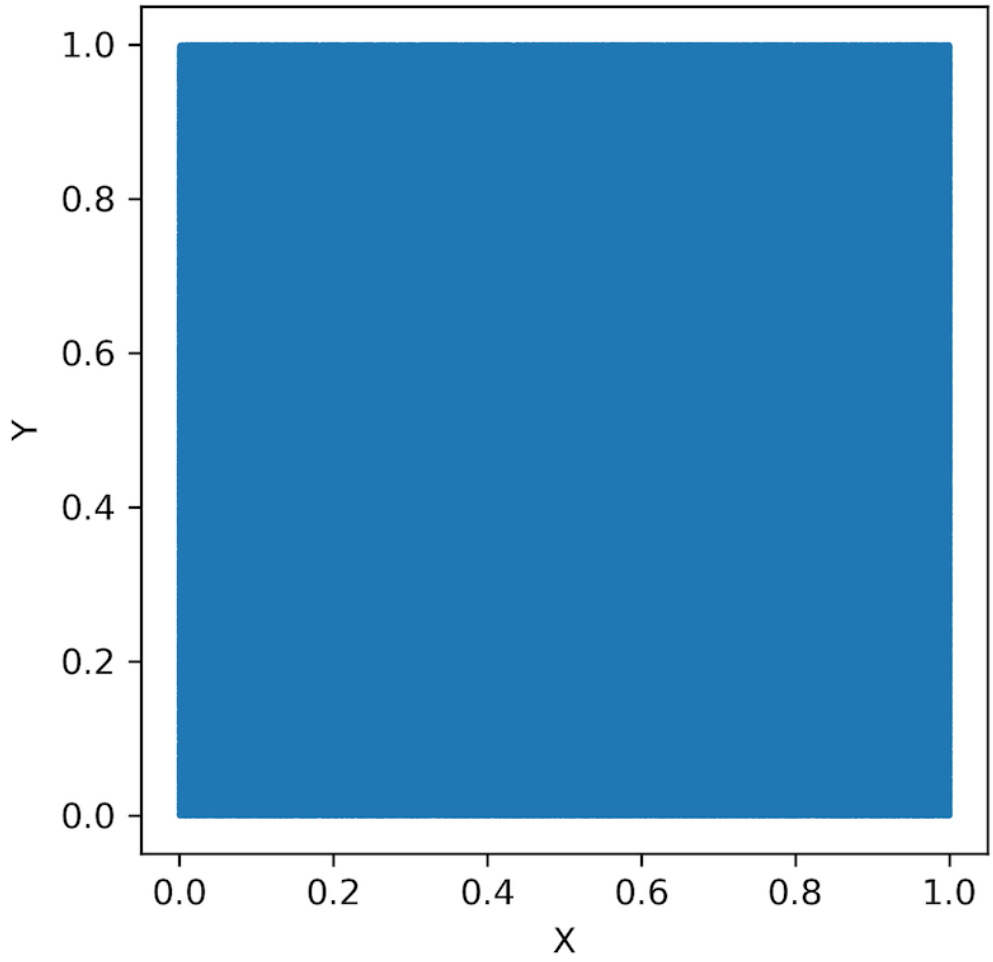

图 6: 由种子“1”生成的 1×10^7 个均匀分布随机点

从图中我们基本上看不出有条带结构和规则网格结构等有明显规律性的结构[4]，可以大致认为随机点见关联性较小。从线性相关系数上看，有下表：

	$N=1\times10^7$	$N=2\times10^7$	$N=3\times10^7$	$N=4\times10^7$	$N=5\times10^7$
$C(1)$	3.443596×10^{-4}	6.452935×10^{-5}	9.313238×10^{-5}	-1.120625×10^{-4}	7.062223×10^{-5}
$C(2)$	3.922429×10^{-5}	-1.839874×10^{-4}	-1.082820×10^{-4}	-3.138071×10^{-4}	-3.202328×10^{-4}
$C(3)$	3.617954×10^{-4}	2.321942×10^{-4}	1.555808×10^{-4}	2.003216×10^{-4}	9.992192×10^{-5}
$C(4)$	-4.454325×10^{-5}	2.054347×10^{-4}	3.752631×10^{-5}	3.822061×10^{-5}	1.845382×10^{-5}
$C(5)$	-6.123060×10^{-5}	5.802567×10^{-5}	6.715514×10^{-5}	6.617884×10^{-5}	1.476232×10^{-4}
$\frac{1}{\sqrt{N}}$	3.162278×10^{-4}	2.236068×10^{-4}	1.825742×10^{-4}	1.581139×10^{-4}	1.414214×10^{-4}
	$N=6\times10^7$	$N=7\times10^7$	$N=8\times10^7$	$N=9\times10^7$	$N=1\times10^8$
$C(1)$	3.834345×10^{-5}	9.639918×10^{-5}	6.261886×10^{-5}	4.522615×10^{-5}	4.801942×10^{-5}
$C(2)$	-1.925323×10^{-4}	-1.556041×10^{-4}	-1.938455×10^{-4}	-1.909826×10^{-4}	-1.506461×10^{-4}
$C(3)$	6.037514×10^{-5}	6.153215×10^{-5}	3.273687×10^{-5}	7.904676×10^{-5}	1.029654×10^{-4}
$C(4)$	3.925839×10^{-5}	6.243648×10^{-5}	6.87037×10^{-5}	6.874031×10^{-5}	7.159278×10^{-5}
$C(5)$	1.117007×10^{-4}	9.083208×10^{-5}	1.036038×10^{-5}	-1.098901×10^{-5}	-3.705724×10^{-5}
$\frac{1}{\sqrt{N}}$	1.290994×10^{-4}	1.195229×10^{-4}	1.118034×10^{-4}	1.054093×10^{-4}	1.000000×10^{-4}

表 1: $C(l)$ 一览表 1

[4]由于图片分辨率限制，故点数很多时实际上所有 $(0,1)$ 的像素点比总点数要小很多，故此时形成 $(0,1)$ 是一“整块色块”的结果。

可以看出对于随机数总数 $N = i \times 10^7 (N = 1, 3, 4, 5, 6, 7, 8, 9, 10)$，随机数列的相关系数较大 ($C(l)$ 与 $\frac{1}{\sqrt{N}}$ 量级有时相当)，且能看出其线性相关系数随 N 的增大有减小的趋势，不能肯定 x_n 和 x_{n+l} 间没有线性关系。而通过比较发现相关系数与 l 之间也没有明显的关系。为了进一步探究 $C(l)$ 是否和 N，l 有关，我们又做了如下测试：[5]

	$N = 1 \times 10^1$	$N = 1 \times 10^2$	$N = 1 \times 10^3$	$N = 1 \times 10^4$	$N = 1 \times 10^5$	$N = 1 \times 10^6$
$C(1)$	2.76606×10^{-1}	8.989166×10^{-2}	4.054796×10^{-2}	-7.021954×10^{-6}	2.4258476×10^{-3}	-2.708991×10^{-4}
$C(2)$	-3.208741×10^{-1}	-6.171155×10^{-2}	-2.748282×10^{-2}	-1.144587×10^{-2}	-2.674438×10^{-3}	-1.717263×10^{-3}
$C(3)$	-1.575569×10^{-1}	1.522378×10^{-2}	1.006404×10^{-2}	-5.431764×10^{-3}	3.505449×10^{-3}	8.817028×10^{-4}
$C(4)$	2.964212×10^{-3}	5.501791×10^{-2}	3.450233×10^{-2}	1.437336×10^{-4}	3.865893×10^{-3}	-3.776349×10^{-4}
$C(5)$	5.247579×10^{-1}	-1.43132×10^{-2}	-7.097997×10^{-3}	-5.586901×10^{-3}	-7.990500×10^{-4}	-3.313229×10^{-4}
$\frac{1}{\sqrt{N}}$	3.162278×10^{-1}	1.000000×10^{-1}	3.162278×10^{-2}	1.000000×10^{-2}	3.162278×10^{-3}	1.000000×10^{-3}

表 2: $C(l)$ 一览表 2

由上表可以看出，$C(l)$ 和 l 确实没有明显的关系，但确实随着 N 的增大而明显的减小。但在不同的 N 的量级下，也总会有 $C(l)$ 比 $\frac{1}{\sqrt{N}}$ 大的情况，所以也不能说 16807 产生器产生的随机数，其顺序相关较小。

当我们设定 $k \in [1, 10]$ 来计算 $< x^k >$ 时可得到一下结果：

	$N = 10^1$	$N = 10^2$	$N = 10^3$	$N = 10^4$	$N = 10^5$	$N = 10^6$	$N = 10^7$	理论值
$k = 1$(计算值)	0.351409	0.493705	0.491958	0.500045	0.499777	0.500341	0.499880	0.500000
$k = 2$(计算值)	0.216174	0.327029	0.323331	0.333121	0.332932	0.333583	0.333173	0.333333
$k = 3$(计算值)	0.152751	0.243376	0.239747	0.249802	0.249505	0.250158	0.249832	0.250000
$k = 4$(计算值)	0.117402	0.193030	0.189955	0.199907	0.199445	0.200097	0.199835	0.200000
$k = 5$(计算值)	0.095795	0.159362	0.156989	0.166714	0.166079	0.166725	0.166507	0.166667
$k = 6$(计算值)	0.081586	0.135182	0.133601	0.143057	0.142260	0.142893	0.142704	0.142857
$k = 7$(计算值)	0.071592	0.116897	0.116170	0.125347	0.124409	0.125022	0.124853	0.125000
$k = 8$(计算值)	0.064114	0.102531	0.102685	0.111595	0.110536	0.111125	0.110970	0.111111
$k = 9$(计算值)	0.058209	0.090916	0.091945	0.100606	0.099447	0.100010	0.099865	0.100000
$k = 10$(计算值)	0.053334	0.081314	0.083188	0.091620	0.090382	0.090917	0.090779	0.090909

表 3: $< x^k >$ 一览表

[5]本人仅测试了 $l = 1, 2, 3, 4, 5$，至少这些结论对于这些 l 的取值是成立的

	$N=10^1$	$N=10^2$	$N=10^3$	$N=10^4$	$N=10^5$	$N=10^6$	$N=10^7$
$k=1$(计算值)	0.148591	0.006295	0.008042	0.000045	0.000223	0.000341	0.000120
$k=2$(计算值)	0.117159	0.006304	0.010002	0.000212	0.000401	0.000249	0.000161
$k=3$(计算值)	0.097249	0.006624	0.010253	0.000198	0.000495	0.000158	0.000168
$k=4$(计算值)	0.082598	0.006970	0.010045	0.000093	0.000555	0.000097	0.000165
$k=5$(计算值)	0.070872	0.007304	0.009677	0.000048	0.000587	0.000059	0.000160
$k=6$(计算值)	0.061271	0.007675	0.009256	0.000199	0.000597	0.000036	0.000153
$k=7$(计算值)	0.053408	0.008103	0.008830	0.000347	0.000591	0.000022	0.000147
$k=8$(计算值)	0.046997	0.008580	0.008426	0.000484	0.000575	0.000014	0.000141
$k=9$(计算值)	0.041791	0.009084	0.008055	0.000606	0.000553	0.000010	0.000135
$k=10$(计算值)	0.037575	0.009595	0.007721	0.000711	0.000527	0.000008	0.000130
$\frac{1}{\sqrt{N}}$	0.316228	0.10000	0.031623	0.010000	0.003162	0.001000	0.000316

表 4: $< x^k >$ 与理论值差别的绝对值一览表

由上表看出，当 N 越大时，随机点的 $< x^k >$ 越接近于理想均匀分布的理论值。在 $N \leq 10^7$ 时与理论值的差别控制在 $\frac{1}{\sqrt{N}}$ 以内，可认为此方法产生的随机点分布较为均匀。

5 心得与体会

通过此次作业结果，发现比较简单的 16807 随机数产生器比预想的要好一点，之前以为画出的随机点的图像上会有明显的规律性结构，在此问题下此随机数产生器还是有可用性的。但是其顺序相关性还是不能够认为是比较小的，故在某些需要顺序相关性比较小的情况下，还是不能够使用此产生器产生随机数。

另外也是第一次编程作业的缘故，也有大致复习了 (重学) 了一遍 C 语言的作用。另外也发现要给数组开辟较大的空间时，要采用 malloc() 函数，从堆上开辟空间，直接按静态数组定义，会造成栈空间的不够从而报错 (全局数组例外) 的 trick。还有就是也熟悉了一下 LaTeX。

6 附录

A C 语言程序源码

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#define a 16807
#define b 0
#define m 2147483647
#define r (m % a)
#define q (m / a)


//写文件子程序，输入写成文件名称字符串str，数据来源于数组num，数据总数n
int my_filewriter(char str[],double num[],int n){
    FILE * fp;
    fp = fopen(str,"w+");

    for(int i=0;i<(n-1);i++)
    {
        fprintf(fp,"%lf,",num[i]);

    }
    fprintf(fp,"%lf",num[n]); //最后一个数据后不加 ","
    fclose(fp);
    return 0;
}



// Schrage 方法产生随机数
int my_schrage(double ran[],int ranI[],int n){
    for (int i = 0; i < n - 1; i++) {
        if (ranI[i] >= 0) {
```

```
33            ran[i] = ranI[i] / (double) m;
34        } else {
35            ran[i] = (ranI[i] + m) / (double) m;
36        }
37        if(ranI[i] == m-1){
38            if(a >= b){   //由于Schrage方法只对z in
                  (0,m-1)成立，故这里要讨论z == m-1的情况
39                ranI[i+1] = m + (b-a) % m;
40            }
41            else  ranI[i+1] = (b-a) % m;
42
43        }
44        else ranI[i+1] = ((a * (ranI[i] % q) - r *( ranI[i] / q) ) +
              b % m ) % m; //递推式
45    }
46    if (ranI[n-1] >= 0) {
47        ran[n-1] = ranI[n-1] / (double) m;
48    } else {
49        ran[n-1] = (ranI[n-1] + m) / (double) m;
50    }
51    return 0;
52 }
53
54
55
56
57
58 //线性相关检验
59 int my_cl_check(double ranx[],double rany[],int n){
60    long double xy = 0; //sum(x*y)
61    long double x = 0; //sum(x)
62    long double x2 = 0; //sum(x^2)
63    double cl = 0;
64    for(int i = 0;i<n;i++){
65        xy += ranx[i] * rany[i];
66        x += ranx[i];
67        x2 += pow(ranx[i],2);
68    }
```

```
69      cl = ((xy/n)-pow(x/n,2))/(pow(x,2)/n -pow(x/n,2)); //C(l)计算
70      printf("所产生的%d 个随机点的线性相关系数 C(l) = %E\n",n,cl);
71      return 0;
72  }
73
74
75
76  //<x^k>测试均匀性
77  int my_xk_check(double ranx[],double rany[],int n,int k){
78      double sumxk[k]; //sum(x^k)
79      double sumyk[k]; //sum(y^k)
80      for(int j = 0;j < k;j++){ //初始化数组
81          sumxk[j] = 0;
82          sumyk[j] = 0;
83      }
84
85      for(int i = 0;i < n;i++){ //数组赋值
86          for(int j = 0;j < k;j++){
87              sumxk[j] += pow(ranx[i],j+1);
88              sumyk[j] += pow(rany[i],j+1);
89          }
90      }
91      for(int j = 0;j < k;j++){ //计算<x^k>,<y^k>
92          sumxk[j] = sumxk[j]/n;
93          printf("N = %-8d <x^%d> = %lf with theoretical number of
                %lf\n",n,j+1,sumxk[j],1.0/(2+j));
94          sumyk[j] = sumyk[j]/n;
95          //printf("N = %-8d <y^%d> = %lf with theoretical number of
                %lf\n",n,j+1,sumyk[j],1.0/(2+j));
96          //此子程序还可用来计算<y^k>
97      }
98
99
100     return 0;
101 }
102
103
104
```

```
105 int main() {
106     int N;
        //总随机数个数，即x坐标随机数值+y坐标随机数值（即必须为偶数）
107     char str[50];
108     printf("请输入您所需要的总点数: ");
109     while (!scanf("%d",&N)) { //简单的输入检查
110         gets(str);
111         printf("\nInput error,please try again\n");
112         printf("请输入您所需要的总点数: ");
113     }
114     N=2*N;
115     if(N >2000000)
            printf("您输入的参数已接受，正在计算请稍等片刻~\n");
116
117     double *ran = malloc(sizeof(double) * N); //用来存放随机数
118     int *ranI = malloc(sizeof(int) * N); //用来存放递推数列
119     ranI[0]=1; //产生一个种子值放入ranI的第一项
120
121     my_schrage(ran,ranI,N);
122
123     double *ranx = malloc(sizeof(double) * N/2);
124     double *rany = malloc(sizeof(double) * N/2);
125
126     for(int i=0;i<N;i=i+2){
            //将ran[]中相邻的两个随机数分别赋值到ranx[]和rany[]
127         ranx[i/2]=ran[i];
128         rany[i/2]=ran[i+1];
129     }
130     my_filewriter("ranx.dat",ranx,N/2);
131     my_filewriter("rany.dat",rany,N/2);
132     my_cl_check(ranx,rany,N/2);
133
134     int k;
135     printf("Please input the max value of k in <x^k>:");
136     while (!scanf("%d",&k)) { //简单的输入检查
137         gets(str);
138         printf("\nInput error,please try again\n");
139         printf("Please input the max value of k in <x^k>:");
```

```
140     }
141     printf("以下是<x^k>的均匀性测试:\n");
142     for(int j = N/10 ;j >= 1 ;j = j/10){ //进行<x^k>均匀性测试
143         my_xk_check(ranx,rany,N/j,k);
144     }
145 
146     return 0;
147 }
```

B 可视化 python 脚本源程序

```
import matplotlib.pyplot as plt
import numpy as np
from IPython.core.pylabtools import figsize # import figsize
#figsize(12.5, 4) # 设置 figsize
plt.rcParams['savefig.dpi'] = 300 #图片像素
plt.rcParams['figure.dpi'] = 300 #分辨率

fig = plt.figure()
ax = fig.add_axes([0.1, 0.1, 0.8, 0.8])
X = []
Y = []

with open('cmake-build-debug/ranx.dat', 'r') as f:
    #依数据文件所在位置而变
    while True:
        lines = f.readline() # 整行读取数据
        if not lines:
            break
            pass
        tmp = [float(i) for i in lines.split(',')] #
            将整行数据分割处理
        X.append(tmp) # 添加新读取的数据
        pass
    X = np.array(X) # 将数据从list类型转换为array类型。
pass


with open('cmake-build-debug/rany.dat', 'r') as f:
    #依数据文件所在位置而变

    while True:
        lines = f.readline() # 整行读取数据
        if not lines:
            break
            pass
```

```
33          tmp = [float(i) for i in lines.split(',')] #
                将整行数据分割处理
34          Y.append(tmp) # 添加新读取的数据
35          pass
36      Y = np.array(Y) # 将数据从list类型转换为array类型。
37  pass
38
39
40
41  # 绘制散点图
42  ax.scatter(X, Y, s=0.1) #画散点图
43  ax.set_aspect('equal')
44  ax.set_xlabel('X')
45  ax.set_ylabel('Y')
46  plt.savefig("random(x,y).png")
```