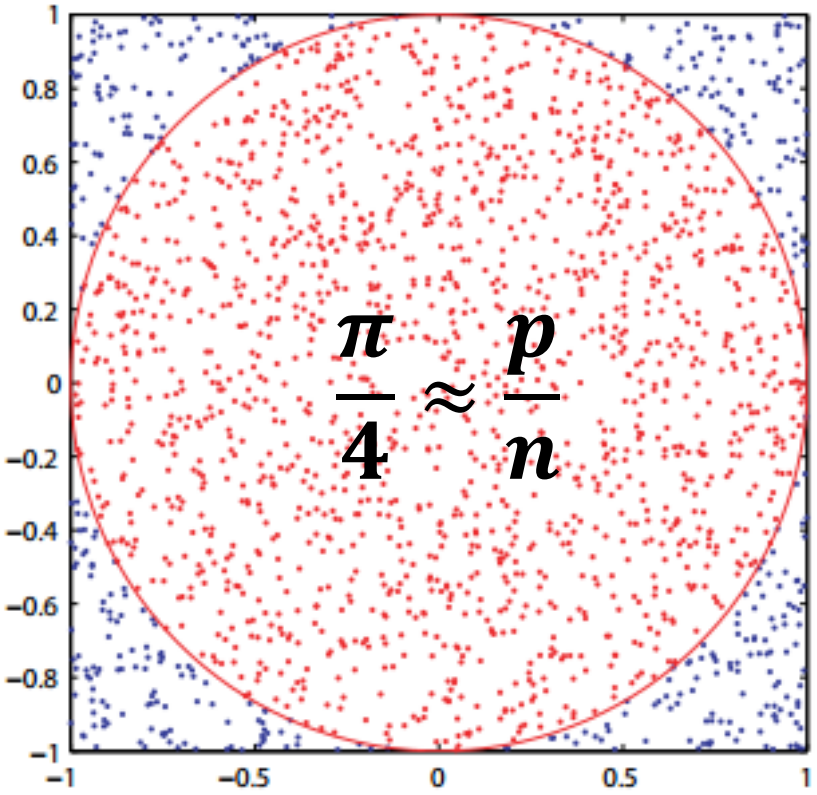
《计算机模拟》

第3讲 - 随机数的生成与检验

胡贤良

浙江大学数学科学学院

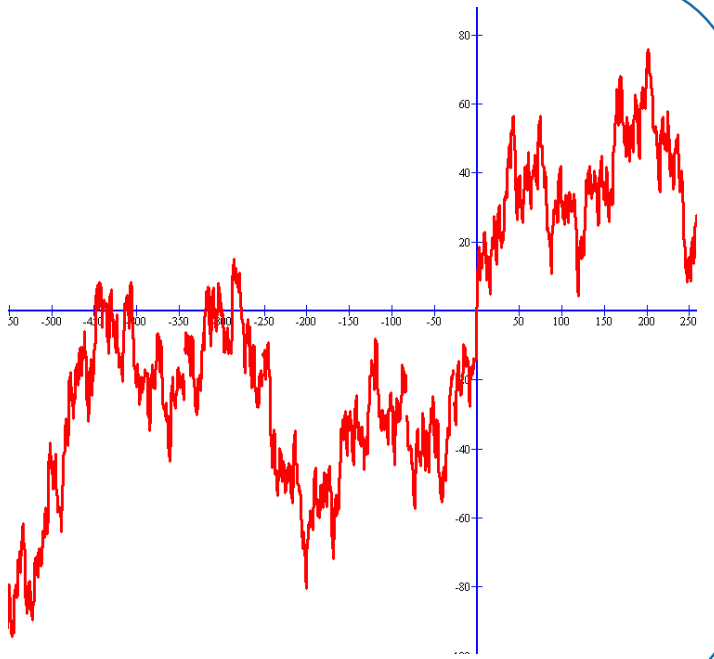
无处不在的随机数

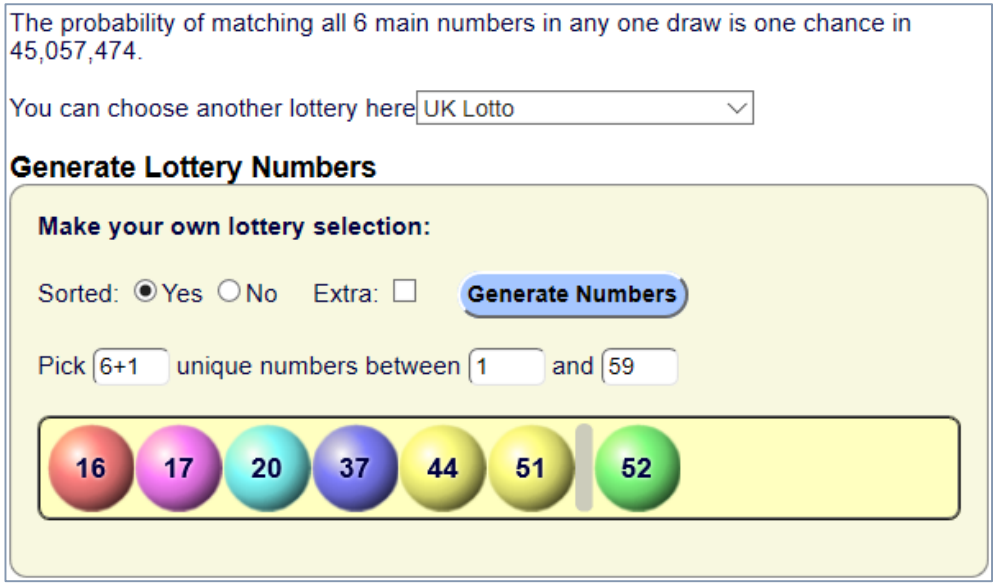


https://www.compadre.org/osp/items/detail.cfm?ID=7311

运行(需java运行环境)：ejs_stp_MonteCarloPi.jar

$$
\begin{aligned}
f(x) &= 2\left(\sin(\pi x) + \sqrt{2}\sin\left(\frac{\pi}{2}x\right)\right. \\
&+ 2\sin\left(\frac{\pi}{4}x\right) + 4\sin\left(\frac{\pi}{16}x\right) \\
&+ 16\sin\left(\frac{\pi}{256}x\right) \\
&\left.+ 16^2\sin\left(\frac{\pi}{256^2}x\right)\right) \\
&+ 2\left(\sin(\pi x) + \sqrt{3}\sin\left(\frac{\pi}{3}x\right)\right. \\
&+ 3\sin\left(\frac{\pi}{9}x\right) + 9\sin\left(\frac{\pi}{81}x\right) \\
&+ 81\sin\left(\frac{\pi}{6561}x\right) \\
&\left.+ 81^2\sin\left(\frac{\pi}{6561^2}x\right)\right)
\end{aligned}
$$





截图来源：http://www.randomnumbergenerator.com/lottery.asp

随机数生成 – 发展概览

- 早期，人们用一个“充分转动起来”的坛子中抓球或摇骰子、分牌之类的办法。
- 1927年，L.H.C.Tippett发布超过40000个随机数表，“从人口统计调查报告中随机地取得的”。
- 1939年，M.G.Kendall和B.Babington-Smith使用机器造出了有100000个随机数字的一张表。
- 1949年，D.H.Lehmer提出线性同余法（LCG）的算法方案。
- 1951年首次安装的Ferranti Mark Ⅰ计算机有一个内置指令，它使用一个电阻噪声生成器将20个随机位放入累加器中，这一功能受到了A.M.Turing的推荐。
- 1955年，RAND公司利用一台取名为ERNIE的随机数机器得到有100万个随机数字的表，后被用于产生英国政府有奖债券的中奖号码。
- 90年代计算机的发展，在CDROM上可以分布（存储）1GB的经过测试的随机数。
- 1995年，George Marsaglia 将一个噪声二极管的电流输出与确定的经过扰频的rap音乐叠加在一起生成了650MB的随机值（称之为黑白噪声）并把它们做成演示光盘，促进了随机数表的复兴。
- 2014年，计算机PRNG算法-PCG诞生，它以更小的，快速的代码和小的状态量实现了出色的统计性能。
- 2017年，量子随机数发生器。。。

随机数表及其读取方法

1. **统一编号:**即将总体中的所有研究对象进行统一编号，然后充分混合，目的是使各样本编号**均匀分布**，符合**机会均等**的原则。
2. **确定抽样起点:**根据需要或意愿，在表上选择一数字编号，由该数字决定抽样的**起点**。
3. **确定抽样顺序:**根据需要或意愿，选择一定顺序方向，使用该种**顺序方向**进行抽取。
4. **录取号码:**根据抽样起点和抽样顺序进行依次录取号码，直至录取到所需抽取的**样本数满为止。**

随 机 数 表

03 47 43 73 86	36 96 47 36 61	46 98 63 71 62	33 26 16 80 45	60 11 14 10 95
97 74 24 67 62	42 81 14 57 20	42 53 32 37 32	27 07 36 07 51	24 51 79 89 73
16 76 62 27 66	56 50 26 71 07	32 90 79 78 53	13 55 38 58 59	88 97 54 14 10
12 56 85 99 26	96 96 68 27 31	05 03 72 93 15	57 12 10 14 21	88 26 49 81 76
55 59 56 35 64	38 54 82 46 22	31 62 43 09 90	06 18 44 32 53	23 83 01 30 30
16 22 77 94 39	49 54 43 54 82	17 37 93 23 78	87 35 20 96 43	84 26 34 91 64
84 42 17 53 31	57 24 55 06 88	77 04 74 47 67	21 76 33 50 25	83 92 12 06 76
63 01 63 78 59	16 95 55 67 19	98 10 50 71 75	12 86 73 58 07	44 39 52 38 79
33 21 12 34 29	78 64 56 07 82	52 42 07 44 38	15 51 00 13 42	99 66 02 79 54
57 60 86 32 44	09 47 27 96 54	49 17 46 09 62	90 52 84 77 27	08 02 73 43 28
18 18 07 92 45	44 17 16 58 09	79 83 86 19 62	06 76 50 03 10	55 23 64 05 05
26 62 38 97 75	84 16 07 44 99	83 11 46 32 24	20 14 85 88 45	10 93 72 88 71
23 42 40 64 74	82 97 77 77 81	07 45 32 14 08	32 98 94 07 72	93 85 79 10 75
52 36 28 19 95	50 92 26 11 97	00 56 76 31 38	80 22 02 53 53	86 60 42 04 53
37 85 94 35 12	83 39 50 08 30	42 34 07 96 88	54 42 06 87 98	35 85 29 48 39

(真)随机数获取方法

利用机器的噪音生成随机数。噪音源包括各种硬件运行时速，用户和计算机交互时速。比如：击键的间隔时间、鼠标移动速度、特定中断的时间间隔等。

- Java可用java.security.SecureRandom 产生真随机数
- Linux系统有/dev/random,/dev/urandom提供真随机数
- Windows系统有CryptGenRandom 函数生成真随机数

- 优点：装置能产生几乎完美的真随机数
- 缺点：产生效率不高、装置成本昂贵
- 折衷方法：记录真随机数表，提供查询服务

Integers	Size of Range	Price
50,000	10	$5.00
250,000	1,000	$5.00
1,000,000	100	$6.66
5,000,000	1,000	$36.17
10,000,000	1,000	$61.09

网站 https://www.random.org 提供的随机整数表的价格

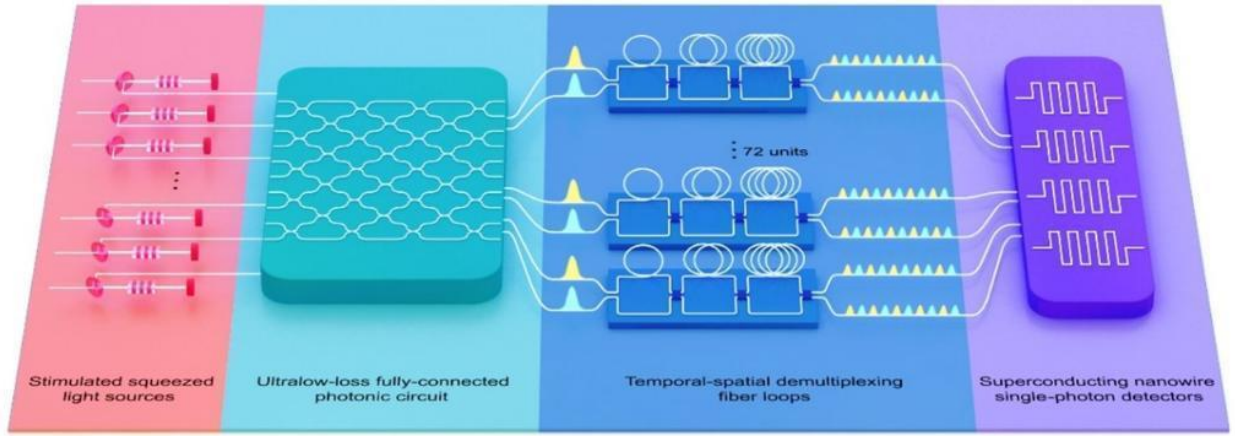
(真)随机数发生器

- 基于电路(震荡/噪声/混沌)的随机数发生器（**真随机数**发生器，TRNG）

- 基于其他物理过程，即客观世界中的真实随机现象，如：分子热运动、量子效应、激光脉冲等。基于量子原理的随机数发生器是最热门的

九章量子计算机

- 2020年版本：玻色子采样GBS。利用热状态、可区分的光子和均匀分布，对获得的样品进行可信假设的验证。比使用最先进的模拟策略和超级计算机快大约 10^{14} 倍的采样率(GBS**200秒 v.s. 神威**$8x10^{14}$秒/25亿年)



- 2021年：
  - 祖冲之二号，56比特量子计算原型机
  - 九章二号，113个光子，可相位编程
- 2023年版本：
  - 图论问题：揭示了高斯玻色取样和图论之间的数学联系，模拟速度比经典计算机快1.8亿倍
  - 九章三号： Gaussian Boson Sampling with Pseudo-Photon-Number-Resolving Detectors and Quantum Computational Advantage. 255个光子，比上一代“九章二号”提升一百万倍，当前全球最快的超级计算机Frontier约需200亿年。

➢ 随机数生成器

1. 平方取中法
2. 经典Fibonacci产生器
3. 线性同余法
4. 非线性同余法
5. 多步线性递推
6. **梅森旋转**

…

(伪)随机数

- 由算法生成，产生速度足够快
- 便宜！便宜！便宜！
- 缺点：不是真随机数，一般具有某种规律
- 折衷的方法：

一些优质算法产生的伪随机序列，在**统计学指标**上表现得和真随机数**几乎一致**！如：高质量的伪随机数生成算法完全可胜任Monte Carlo方法模拟。

```cpp
//rand02.cpp
#include <iostream>
using namespace std;
int main()
{
    unsigned int seed=5;
    srand(seed);
    unsigned int r=rand();
    cout<<"r = "<<r<<endl;
    return 0;
}
```

```cpp
//rand01.c
#include <stdlib.h>
static unsigned int RA
unsigned int random(v
{
    RAND_SEED=(RAND_SEED*123+59)%65536;
    return(RAND_SEED);
}
void random_start(void)
{
    int temp[2];
    movedata(0x0040,0x006c,FP_SEG(temp),FP_OFF(temp),4);
    RAND_SEED=temp[0];
}
main()
{
    unsigned int i,n;
    random_start();
    for(i=0;i<10;i++)
        printf("%u\t",random());
    printf("\n");
}
```

1. 平方取中法(midsquare method)

产生[0,1]上均匀分布随机数, 也称**冯·诺伊曼取中法**.

1. 取一个2s位的整数，称为**种子**，将其平方，得4s位整数(不足4s位时高位补0)

$$x_{i+1} = \left[\frac{x_i^2}{10^s}\right] mod\ 10^{2s},$$

2. 取此4s位的中间2s位作为下一个种子数，并对此数执行归一化运算得到[0,1]之间的数值

$$u_{i+1} = \frac{x_{i+1}}{10^{2s}}$$

【例1】取$s=2, x_0=1234$，模为10 000，则有

$$x_0^2=01522756 \quad x_1=5227 \quad u_1=0.5227$$
$$x_1^2=27321529 \quad x_2=3215 \quad u_2=0.3215$$
$$x_2^2=10336225 \quad x_3=3362 \quad u_3=0.3362$$
$$x_3^2=11303044 \quad x_4=3030 \quad u_4=0.3030$$
$$x_4^2=09180900 \quad x_5=1809 \quad u_5=0.1809$$
$$x_5^2=03272481 \quad x_6=2724 \quad u_6=0.2724$$
$$\cdots\cdots$$

【例2】取$s=2, x_0=6500$，则有

$$x_0^2=42250000 \quad x_1=2500 \quad u_1=0.2500$$
$$x_1^2=06250000 \quad x_2=2500 \quad u_2=0.2500$$

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

void main(){
    int x,i=0; double m; char str[25];
    scanf("%d", &x);
    while(x < 0){
        printf("error! please enter
            a positive number");
        scanf("%d", &x);
    }
    itoa(x, str, 10);
    m = strlen(str);
    while(i <= 99){
        x = (int)(x*x/pow(10.0, (m/2)))
            % (int)pow(10.0, m);
        printf("x%d = %d \n", ++i, x);
    }
}
```

可见，若取种子为6 500，则周期为1。若在某一随机数列中出现6 500，则以后的数都是2 500，即进入退化状态

平方取中法

【例3】取$s=2, x_0=12345$, 有

0.399000 0.920100 0.658400 0.349000 0.180100 0.243600 0.934000 0.235600 0.550700 0.327000 0.692900 0.011000 0.012100 0.014600

0.021300 0.045300 0.205200 0.210700 0.439400 0.307200 0.437100 0.105600 0.115100 0.324800 0.549500 0.195000 0.802500 0.400600

0.048000 0.230400 0.308400 0.511000 0.112100 0.256600 0.584300 0.140600 0.976800 0.413800 0.123000 0.512900 0.306600 0.400300

0.024000 0.057600 0.331700 0.002400 0.000500 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

平方取中法 - 推广形式

1. **乘法取中法**(mid-product method)

$$x_{i+2} = \left[\frac{x_i \times x_{i+1}}{10^s}\right] \ mod\ 10^{2s},$$

2. **常数乘子法**(constant multiplier method)

$$x_{i+1} = \left[\frac{k \times x_i}{10^s}\right] \ mod\ 10^{2s},$$

- 平方取中法及其两种推广的成功与否与种子及常数的选取关系很大。

  一般说来，s值很大时，种子取值也很大，可使退化推迟，周期加长

2. 经典Fibonacci产生器

输入	X0, X1	初值
	M	最大整数值
	N	随机数个数
输出		N 个服从 $U(0,1)$ 分布的随机浮点数序列

```
1 def rand_add_mod(X0, X1, M, N):
2     X = np.zeros(N)
3     X[0] = X0
4     X[1] = X1
5     for i in range(2, N):
6         X[i] = (X[i - 2] + X[i - 1]) % M
7     return X / (M - 1)
```

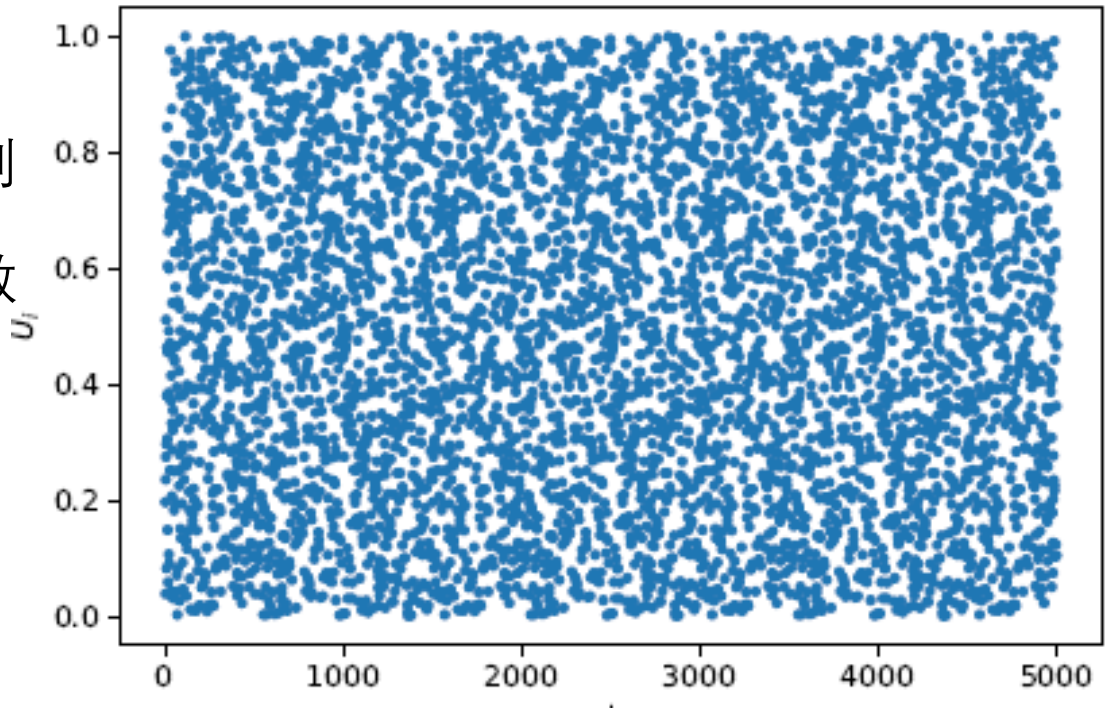
Taussky和Todd(1956)用”加法同余”:

$$X_i = (X_{\{i-2\}} + X_{\{i-1\}}) \mod M, \quad i > 2.$$

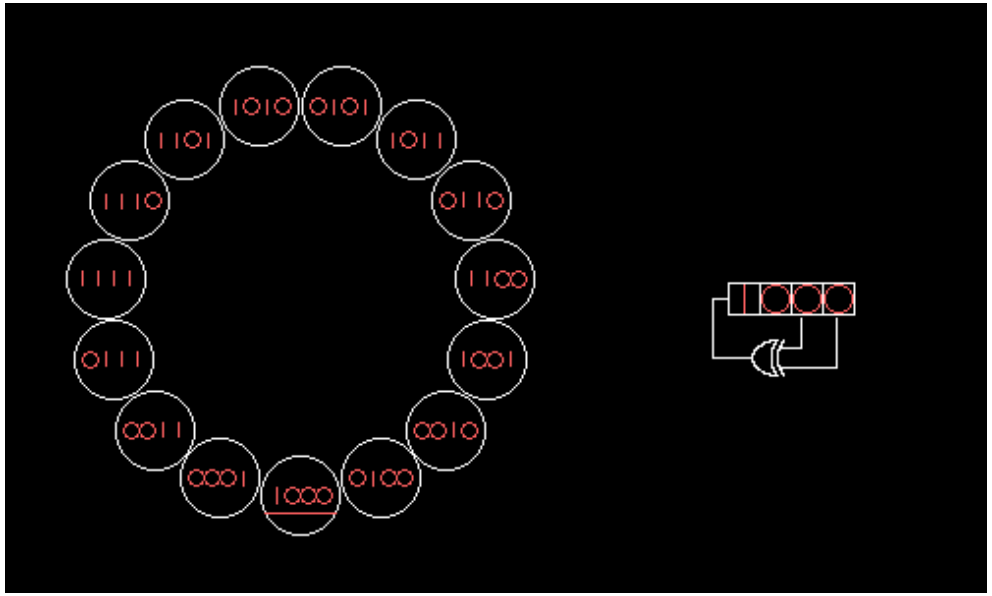
产生在[0,M-1]上的均匀分布伪随机整数序列。

1. 当$X_0 = X_1 = 1$时小于M的部分即为Fibonacci数列
2. 取$U_i = X_i/(M-1)$即得相应$U(0,1)$分布的随机数

参考代码 rand_add_mod.m



Fibonacci-LFSR



- F-LFSR参考斐波那契序列思想生成随机序列
  - 例：一个4bit反馈移位寄存器法（如右图）
- n位的LFSR最多可以产生出2^n-1个不同状态
- 以下代码片断示例了用C语言实现LFSR，代码完成对LFSR的模拟，并得到周期：

```c
#include <stdint.h>
uint16_t lfsr = 0xACE1u; //初始状态
unsigned bit;            //反馈输入
unsigned period = 0;     //生成序列的周期
do {
  /*反馈位： 16、14、13、11*/
 bit  = ((lfsr >> 0) ^ (lfsr >> 2) ^ (lfsr >> 3) ^ (lfsr >> 5) ) & 1; //计算反馈输入（s16^s14^s13^s11）
 lfsr =  (lfsr >> 1) | (bit << 15); //移位，将反馈至输入位
  ++period;
} while(lfsr != 0xACE1u);
```

3. 线性同余法（LCG）

- D.H.Lehmer 1949年提出一般方案，Linear Congruential Generators, LCG.
- 在此基础上，Lehmer 1951和Rotenberg 1960分别研究了**乘同余**和混合同余产生器，

$$X_i = (\mathrm{a} * X_{i-1} + c) \quad mod \ m$$

其中M,A和C分别是模、乘子和增量。

- $c = 0$即为**乘同余**产生器
- $c > 0$ 称为**混合同余产生器**
- 取$U_i = X_i/(m-1)$即可获得[0,1]上的随机数

混合同余产生器
rand_mul_mod.m

输入	X0	初值
	M	最大整数值
	A，C	人工参数
	N	随机数个数
输出		N 个服从 $U(0,1)$ 分布的随机浮点数序列

```
1 def rand_mul_mod(X0, M, A, C, N):
2     X = np.zeros(N)
3     X[0] = X0
4     for i in range(1, N):
5         X[i] = (A * X[i - 1] + C) % M
6     return X / (M - 1)
```

- 序列具有最大的周期m，当且仅当：
  ① **c**和**m**互质；
  ② **m**所有质因子的乘积能够整除**(a-1)；**
  ③ **(a-1)**是4的倍数，若**m**是4的倍数
- 产生的随机序列对a、c、m十分敏感，选择不恰当产生的随机数质量会很差！

线性同余发生器例子

例5.2 考虑线性同余发生器

$$x_n = 7x_{n-1} + 7 \pmod{10}.$$

取初值$x_0 = 7$, 数列为: $(7, 6, 9, 0, 7, 6, 9, 0, 7, \ldots)$, 周期为$T = 4 < M = 10$。

例5.3 考虑线性同余发生器

$$x_n = 5x_{n-1} + 1 \pmod{10}$$

取初值$x_0 = 1$。 数列为: $(1, 6, 1, 6, 1, \ldots)$, 显然周期$T = 2$。

例5.4 考虑线性同余发生器

$$x_n = 5x_{n-1} + 1 \pmod{8}$$

取初值$x_0 = 1$。 数列为$(1, 6, 7, 4, 5, 2, 3, 0, 1, 6, 7, \ldots)$, $T = 8 = M$达最大周期。

- 标准C语言中**rand()**函数的参数设定为： m = 2^{31}， a = 1103515245， c = 12345。

关于参数选择

原根(*Primitive Root*): 称线性同余产生器参数乘子A是模M的原根，当且仅当：

(1) $\left(A^{\{M-1\}}-1\right)\ \ mod\ \ M=0;$

(2) 对于任意 $I<M-1,(A^I-1)/M$不是整数。

可以证明：当模M是素数时，当且仅当乘子A是其原根时，乘同余产生器的周期是$M-1$。

故：

1. 乘同余: $M=2^\beta-1$，即梅森数，β是机器字长

2. 混合同余:$M=2^\beta$；当且仅当$A=4k+1,M$与k互质时，周期为M。

- 《计算机程序设计艺术（第二卷）》3.2.1.1推荐：m=2^31 - 1，a=0.01m~0.99m，且其二进制/十进制表示不应有一个简单的正规模式，推荐值7^5=16807

KISS(Keep It Simple, Stupid)组合产生器

- KISS84:

$$X_i = (69069X_{i-1} + 12345) mod\ 2^{32}$$

- Kiss90:

$$X_i = (X_{i-1} - 362436069) mod\ 2^{32}$$

- KISS93:

$$X_i = (69069X_{i-1} + 23606797) mod\ 2^{32}$$

- KISS99($X_i = (A_i \gg 16) + B_i$) with:

$$\mathrm{A}_i = 36969(\mathrm{A}_{i-1}\&65535) + (A_{i-1} \ll 16)$$

$$B_i = 36969(B_{i-1}\&65535) + (B_{i-1} \ll 16)$$

4. 非线性同余产生器

这种生成器的思想是当 M 是素数时，通过构建某种排列多项式，将 $\{0, 1, \cdots, M-1\}$ 作为一个 Galois 域重排成 $\{g(0), g(1), \cdots, g(M-1)\}$，然后在此基础上构建递推生成器。以继承这种基于重排的均匀性。

比如逆同余产生器：

$$X_i = (AX_{i-1}^{-1} + C) \mod M, i \geq 1. \tag{11}$$

这里的 $g(x) = x^{-1}$ 是一种非线性整函数，定义为：

定义 3.1. 乘法逆（*multiplicative inverse*）称 z^{-1} 是 z 关于 M 的乘法逆，如果

$$z \cdot z^{-1} \mod M = 1, z^{-1} \in \mathbb{N}, z^{-1} < M. \tag{12}$$

并且规定 0 的乘法逆为 0。

可以验证，对 $M = 11$，$A = 8$，$C = 1$ 和 $X_0 = 0$ 产生的前 9 个数是

$$0, 1, 9, 8, 2, 5, 7, 10, 4, 3,$$

基本上就是 $0 \sim 10$ 的重排（漏掉了 6 以外）。这种产生器可以通过很高维的栅格检验，但是它的生成效率远低于线性同余。关于如何高效求乘法逆涉及数论、群论和计算机算法的高深内容，有兴趣的同学可参见 [7].

此外，常用的非线性同余产生器还有二次或高次同余，二次同余递推公式为：

$$X_i = (AX_{i-1}^2 + BX_{i-1} + C) \mod M. \tag{13}$$

[*] Rudolf Lidl and Harald Niederreiter. *Introduction to finite fields and their applications*. Cambridge University Press, 1986.

5. 多步线性递推

即增加线性递推的递推层数，一般性的多步线性递推公式为：

$$X_i = (A_1 X_{i-1} + A_2 X_{i-2} + \cdots + A_j X_{i-j}) \mod M, i \geq j.$$

为提高效率，只保留其中两个系数A_j和A_l不为零，衍生出多种方案：

1. FMRG $(A_1 = 1,\ A_l = B)$,

$$X_i = (X_{i-1} + BX_{i-l}) \mod M;$$

2. DX-k-2 $(A_1 = A_l = B)$,

$$X_i = B(X_{i-1} + X_{i-l}) \mod M;$$

3. DX-k-3 $(A_1 = A_{[l/2]} = A_l = B)$,

$$X_i = B(X_{i-1} + X_{i-[1/2]} + X_{i-l}) \mod M;$$

4. DX-k-4 $(A_1 = A_{l/3} = A_{[2l/3]} = A_l = B)$,

$$X_i = B(X_{i-[l/3]} + X_{i-[2l/3]} + X_{i-l}) \mod M.$$

6. 梅森旋转(Mersenne Twister)

- 是一种目前最广泛使用的随机数产生器
- 由四位日本数学家持续改进：
  1. Matsumoto（松本） & Kurita（栗田）：92, 94
  2. Matsumoto & Nishimura（西村）：98, 00
  3. Matsumoto & Saito （斋藤）：08

  一个重要的实例MT19937：
  - 基于32 位字长机器运算；
  - 几乎所有重要编程平台所采纳；
  - 周期长达$2^{219937}-1$；
  - 能通过目前几乎所有的随机数检验程序；
  - 在**至多623** 维空间分布均匀。

马林·梅森（Marin Mersenne, 1588-1648）17世纪法国著名的数学家和修道士，入选100位在世界科学史上有重要地位的科学家。最早系统深入研究$M_p = 2^p - 1$型的数(梅森数)。如果梅森数为素数，则称之为梅森素数。

6.1 初始化工作区

- 工作区：长624的数组，每个元素均为32 位整数
- 初始化本身也是一个伪随机数产生过程：

$$M_i = f * (M_{i-1} \oplus (M_{i-1} >> (w-2))) + i, i = 1, 2, \cdots, 624. \qquad (19)$$

这里 32 位整数向量 M 存放的就是工作区，M_0 由用户作为种子给出，f 是参数，在 MT19937 中取 1812433253（十六进制：6C078965）。运算符 $\oplus$ 表示异或运算 XOR，w 表示机器字长，在 MT19937 中取 32。具体的算法代

输入	seed	随机数种子
	M	存储梅森工作区的空间
输出		生成的梅森工作区

```
1 # 转换成32位整数。
2 def inter(t):
3     return(0xFFFFFFFF & t) #截取最后32位
4
5 def mainset(seed, M):
6     M[0] = seed
7     for i in range(1,624):
8         M[i] = inter(1812433253 * (M[i - 1] ^ M[i - 1] >> 30) + i
                  )
9     return M
```

6.2 梅森旋转算法MT19937

输入 梅森工作区的空间
输出 旋转后的梅森工作区

```
1  def twister(M):
2      for i in range(624):
3          # 截取M[i]高位和M[i+1](越界就返回M[0])低位，用普通加法合
               并，对齐32位
4          # 这里高位取了1位，低位取了31位。
5          y = inter((M[i] & 0x80000000) +(M[(i + 1) % 624] & 0
               x7fffffff))
6          yA = y >> 1
7          if y & 1 == 1: #取最低位
8              yA = yA ^ 0x9908b0df
9          M[i] = M[(i + 397) % 624] ^ yA
10     return M
```

接下去通过梅森旋转，直接从二进制层面在整个工作区重新分布 0 和 1，并且用旋转和异或操作使二进制数分布均匀。由于工作区的长度是 624，故每产生 624 个数，都要重新执行一次梅森旋转。

$$M_i = M_{i+m} \oplus ((\text{upper\_mask}(M_i)||\text{lower\_mask}(M_{i+1}))A) \tag{20}$$

此处，$\text{upper\_mask}(M_i)$ 和 $\text{lower\_mask}(M_{i+1})$ 分别表示取 M_i 和 M_{i+1} 的高 $w-r$ 位和低 r（在 MT19937 中 $w=32$，$r=31$），符号 $||$ 表示将两边的高位和低位连接成一个 32 位的数。下一步 xA 运算的定义为：

$$xA = \begin{cases} x >> 1, & x_0 = 0, \\ (x >> 1) \oplus a, & x_0 = 1. \end{cases} \tag{21}$$

这里 x_0 表示 x 的最低位，a 是给定参数，在 MT19937 中设置为 0x9908B0DF（十六进制）。最后将 i 后第 m 个工作区数（在 MT19937 中 $m=397$），即 M_{i+m} 拿来和连接成的数做异或，注意上述下标 $i+1$ 和 $i+m$ 在实际实现中都要和 624 取模，因此实际上整个工作区是循环边界的。这也是梅森旋转的名字中旋转的来历。完整的算法过程见算法 4.2。

完整MT19937算法

输入	seed	随机数种子
	num	随机数个数
输出		num 个服从 $U(0,1)$ 分布的随机浮点数

```
1  def exnum(M, index):
2      y = M[index]
3      y = y ^ y >> 11
4      y = y ^ y << 7 & 2636928640
5      y = y ^ y << 15 & 4022730752
6      y = y ^ y >> 18
7      index = index + 1
8      return inter(y)
9
10 def MT19937(seed, num):
11     U = [0]*num
12     M = [0]*624
13     M = mainset(seed, M)
14     twister(M)
15     for i in range(num):
16         index = i % 624
17         U[i] = exnum(M, index) / (2**32 - 1)
18         if (index == 623):
19             twister(M)
20     return U
```

最后我们可以利用工作区，产生梅森旋转的递推公式：

$$y = x \oplus ((x >> u)\&d), \quad (22)$$
$$y = y \oplus ((y << s)\&b), \quad (23)$$
$$y = y \oplus ((y << t)\&c), \quad (24)$$
$$z = y \oplus (y >> l). \quad (25)$$

这里 x 依次从 M 中取，取完 624 个则重新执行梅森旋转。y 是中间变量，z 是返回的下一个随机数。其他 u，d，s，b，t，c 和 l 均为人工参数。在 MT19937 中，$l = 18$，其余参数分别取作

$$(u, d) = (11, 0\text{xFFFFFFFF}_{16}),$$
$$(s, b) = (7, 0\text{x9D2C5680}_{16}),$$
$$(t, c) = (15, 0\text{xEFC60000}_{16}),$$

- 各种语言的实现:
  http://en.wikipedia.org/wiki/Mersenne_twister#Implementations_in_various_languages
- 官方版本:
  http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html
- 基于SIMD计算优化的实现:
  http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/SFMT/index.html

MT19937实用性

- 是目前最通用和最受欢迎的伪随机数生成算法
- 需要634个字长的内存空间（在某些硬件下，这可能是一个缺点）
- 梅森旋转中大量运算是位运算，效率极高、便于机器底层实现
- 运行速度与C语言的rand函数几乎一样快，也有64 和128 位版本
- 作者还在不断提高其计算效率和随机数质量
- 一个缺点是启动慢(当MT的种子各位含有较多的0时)，一般用LCG等初始化。
- 可以找到改进一些具体实现:

  http://en.wikipedia.org/wiki/Well_Equidistributed_Long-period_Linear
- 最新的进展可参见广岛大学提供的网站:

  http://www.math.sci.hiroshima-u.ac.jp/ m-mat/MT/emt.html

资料：梅森与法国数学

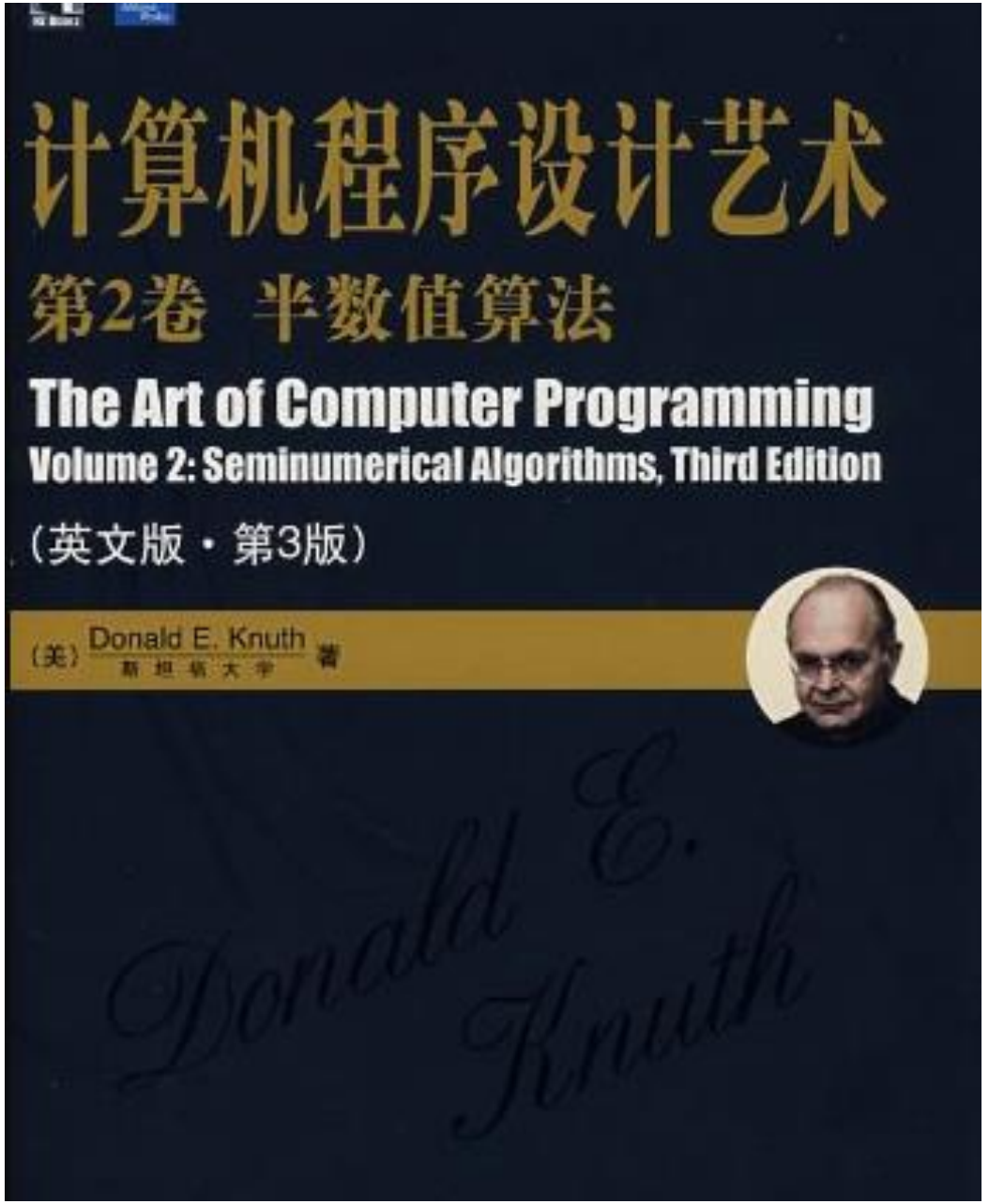
- 法国数学崛起的历史原因: 一个学者(马兰·梅森)、两大君主(路易十四和拿破仑)、一个机构(法国科学院)。法国数学始于17世纪中叶修道院里的数学家马兰·梅森寓所。马兰·梅森少年时毕业于耶稣会学校，是笛卡尔的学长。
- 梅森才华横溢、平易近人，他由于个人的魅力与全欧洲的科学家都建立起良好的联系，在梅森身边也聚集着一大批学者，他们定期在梅森的寓所讨论科学问题。后来，梅森寓所这些科学家沙龙聚会，被称作梅森学院，在当时是全欧洲的学术交流中心。历史上，就有大名鼎鼎的神童帕斯卡，当年他仅十四岁，但已经显出了非凡的数学天分。
- 梅森把帕斯卡接纳进梅森学院，并且鼓励帕斯卡在托里拆利的基础上进一步展开研究，由此帕斯卡不负众望提出了著名的帕斯卡定律。梅森的朋友费马，与帕斯卡同时开拓了概率论的数学分支，他被后人称誉为最杰出的业余数学家，因为很多不懂数学的人，也曾经听过费马定理。

- 1648年梅森去世，人们在他的遗产中发现梅森与欧洲78位学者的十分珍贵的信函，对很多科学领域均有涉猎，其中就包括很多数学大师费马、伽利略、托里拆利、笛卡尔、惠更斯。法国最珍贵的遗产——梅森学院，成为了现如今的巴黎皇家科学院，1666年，巴黎皇家科学院建立。
- 此后，法国年轻的路易十四决定建设一所官方科学院来推动法国科学的发展，巴黎皇家科学院被正式定名，路易十四提供了大量资金的赞助，目的是打消科学家的生活及研究压力。路易十四的财务大臣柯尔贝尔虽然平常精打细算，但也开始大笔资金投入迅速收拢一大批国内外杰出的人才。

- 随机数检验

“当今使用的大部分随机数生成器都不够优秀，而且开发者倾向于拿来就用，不去了解具体的生成策略。以至于我们常常发现一些略有瑕疵、年代久远的随机数生成器会被盲目地用在一个又一个的程序中，而对于它们的局限性，却无人问津。”

--- 《计算机程序设计艺术（第二卷）》



随机数的“好坏”

德国联邦信息安全办公室：

1. 包含相同随机数序列的概率很低.
2. 根据指定的统计检验区别于“真随机数”的数字序列.
3. 不可能被任何人能够通过计算得到，或以其他的方法进行猜测，得到任何给定的一个子序列，以及任何工作中产生的一切序列中的值，以及迭代器的工作状态.
4. 对于任何目的而言，任何人不能从随机数生成器的内部状态计算或猜测得到序列中的任何一个之前的数字／序列以及任何之前的随机数的状态.

- 解决一个模拟问题时，要反复确认所用的随机数在自己的问题中是好的，没有明显缺陷的。例如：
  - 一个经典的称为RANDU的随机数发生器用在一维积分时效果很好，但连续三个一组作为三维均匀分布则很不均匀。
  - 为了验证随机数的效果，找一个和当前问题很接近但是有已知答案的问题，用随机模拟计算并考察其精度。

选自：李东风《统计计算》讲稿 5.4 节

伪随机数的检验

随机模拟广泛地应用于计算各种实际问题，关键之处在于产生[0,1)区间上均匀分布的随机数序列，所生成随机数的质量直接影响计算结果。

1. 理论检验：
   - 主要针对乘同余产生器，没有普适性。如避免出现降维现象、稀疏栅格现象等
2. 统计检验：
   - 一般检验：十进制(0,1)实随机数序列。很多同余法产生的伪随机数都能通过该项检验。检验侧重：自相关性、均匀性、顺序统计量、组合规律、奇偶序列和前后序列间的K-S检验等多项指标
   - 严格检验：针对整数随机数的一种检验方法，按照二进制数字串进行检验，比十进制严格得多。

DieHard检验程序

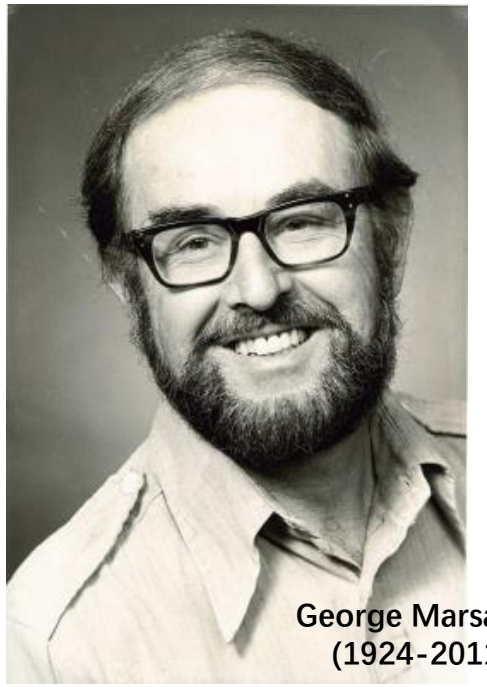
- 由George Marsaglia 教授提出(95,08,10)并维护至2011年:

  http://www.stat.fsu.edu/pub/diehard/

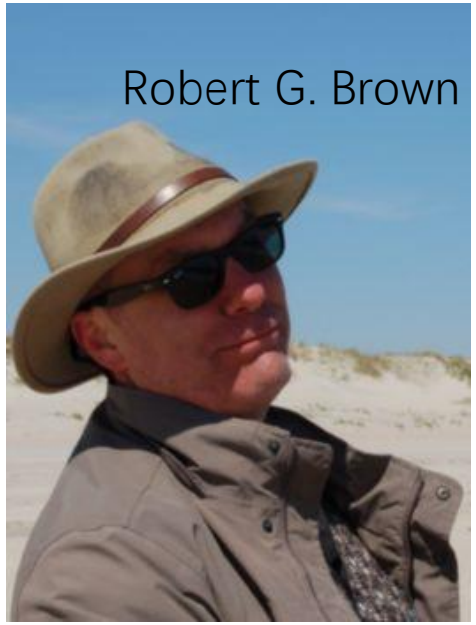
- 引入GPL 协议（开源、**免费**），仍有人继续维护:

  http://webhome.phy.duke.edu/~rgb/General/dieharder.php

- 其中包括:
  - **一般检验方法**：重叠排列检验、停车场检验、最小举例检验、三维随机球检验、挤压检验、重叠求和检验、升连检验、降连检验和掷骰子检验等
  - **严格检验方法**：2进制检验、**猴子检验**、计数1检验、生日间隔检验、最大公因数检验和**大猩猩检验**等



George Marsaglia
(1924-2011)

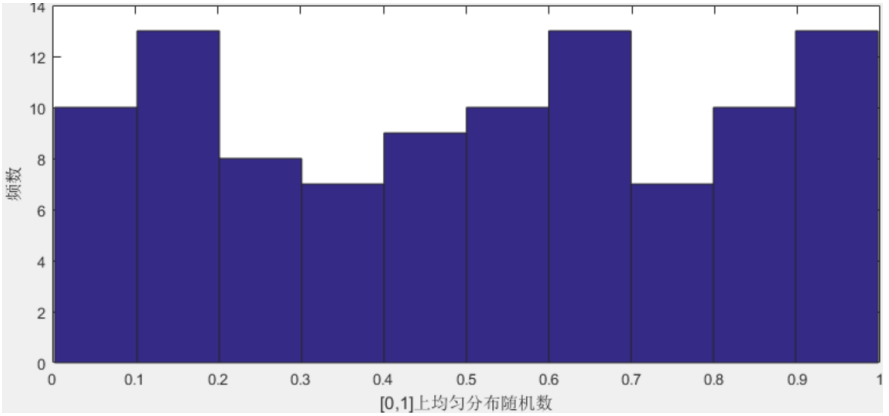


理论检验

均匀随机数：从服从均匀分布U(0,1)的随机变量中

抽样得到的简单子样，其概率密度函数为：

$$f(x) = 1, 0 \le x \le 1.$$

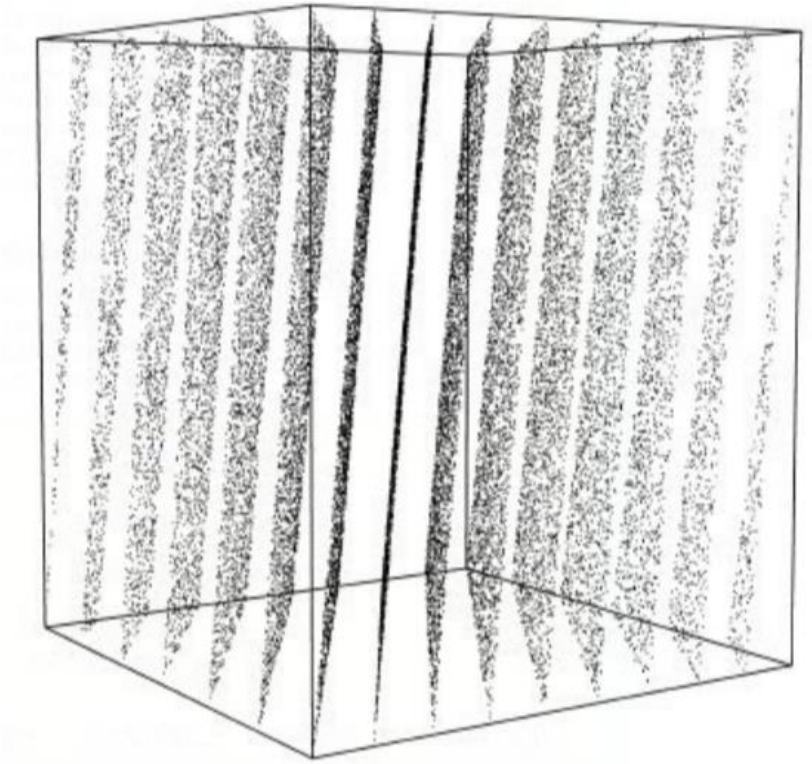


- 随机变量的抽样序列称为**随机数列**：
  - 若随机变量是均匀分布的，则的抽样序列称为**均匀随机数(序)列**
  - 如果是正态分布的随机变量，则称其抽样序列为**正态随机数(序)列**
- 产生均匀分布随机数的算法叫做**均匀分布随机数发生器**
- 若序列$\{U_1, U_2, \ldots\}$表示一个独立同分布随机序列，定义

  高维分布均匀性：由s个随机数组成的s维空间上的点

  $$(U_{\{n+1\}}, U_{\{n+2\}}, \cdots, U_{\{n+s\}})$$

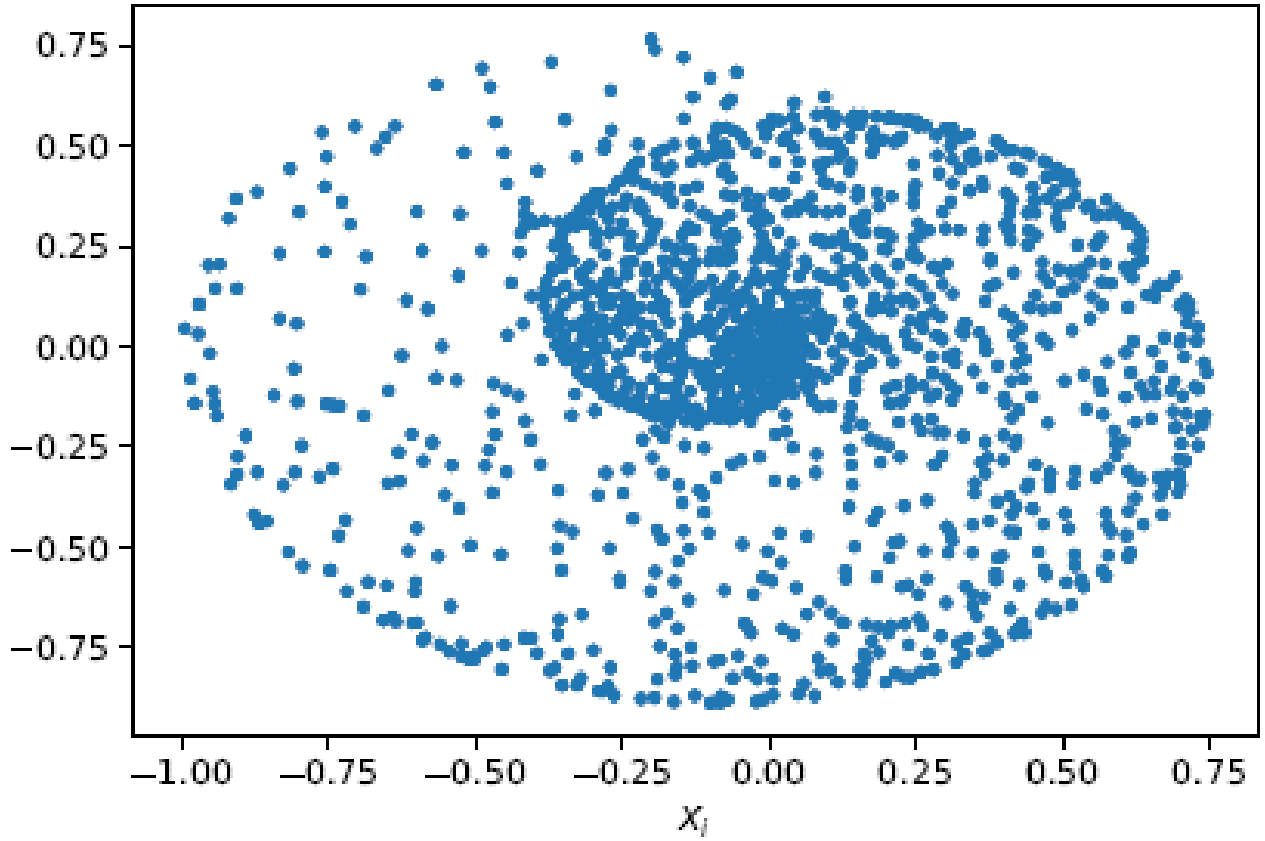
  在s维空间的单位正方体上是均匀分布的(右图为一个反例)



检验 - Fibonacci产生器☹

考察抽样公式（构造）

$$
\begin{aligned}
X_i &= \sqrt{U_i}\cos(2\pi U_{i+1})\sin(\pi U_{i+2}) \\
Y_i &= \sqrt{U_i}\sin(2\pi U_{i+1})\sin(\pi U_{i+2})
\end{aligned}
$$



得到具有“某种规律/相关性”的随机序列(X,Y)，如上图所示.

- 参考main_4-1.m

检验 - 线性同余发生器 ☹

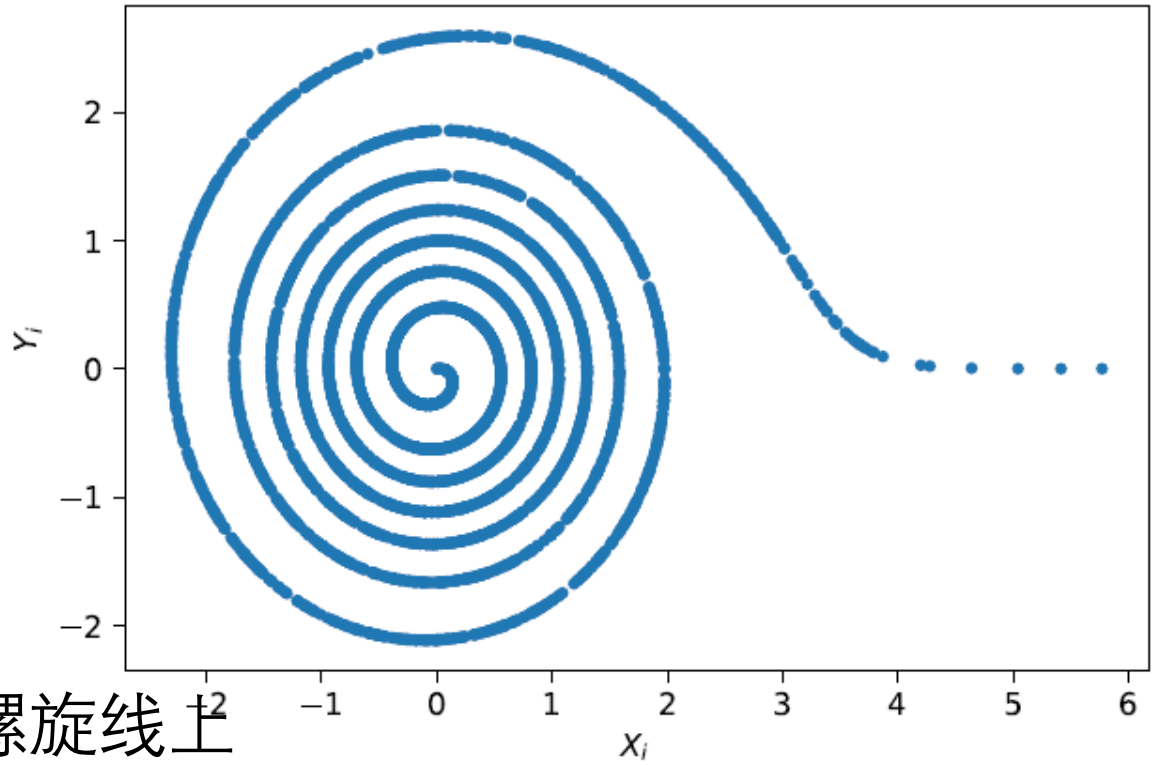
不恰当地选择参数也会有有类似加同余的异常相关现象。

如取

$$A = 7, M = 2^{\{31\}} - 1,$$

则抽样

$$\begin{aligned} X_i &= \sqrt{-2\ln U_i}\cos(2\pi U_{i+1}) \\ Y_i &= \sqrt{-2\ln U_i}\sin(2\pi U_{i+1}) \end{aligned}$$



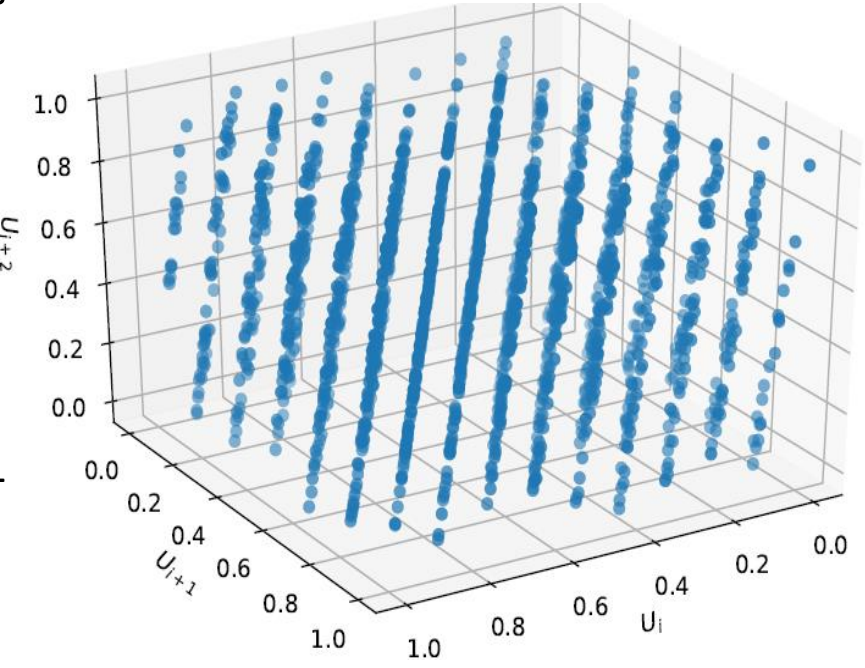
的二维分布（即样本点）会落在一条螺旋线上

- 参考 main_4-2.m

检验 - 稀疏栅格现象 – 降维☹

Florida大学的G. Masaglia教授(1968,1972)首次发现线性同余方法产生的随机数具有很强的相关性:

- 相继的s个伪随机数作为s维空间一个点时，这些点只分布在s维空间的少数几个彼此平行的超平面上，如当M=2^{32}时, 有可能落在不超过$(s!\,M^{1/s})$个低维超平面上!
  - s=1, 5, 10, 50, 100, 500, 1000，低维超平面个数为2^{32}, 84, 9.2, 5.6, 1.3, 1.1
  - 1000维以后，之落在1个超平面上



例：随机数产生器RANDU:

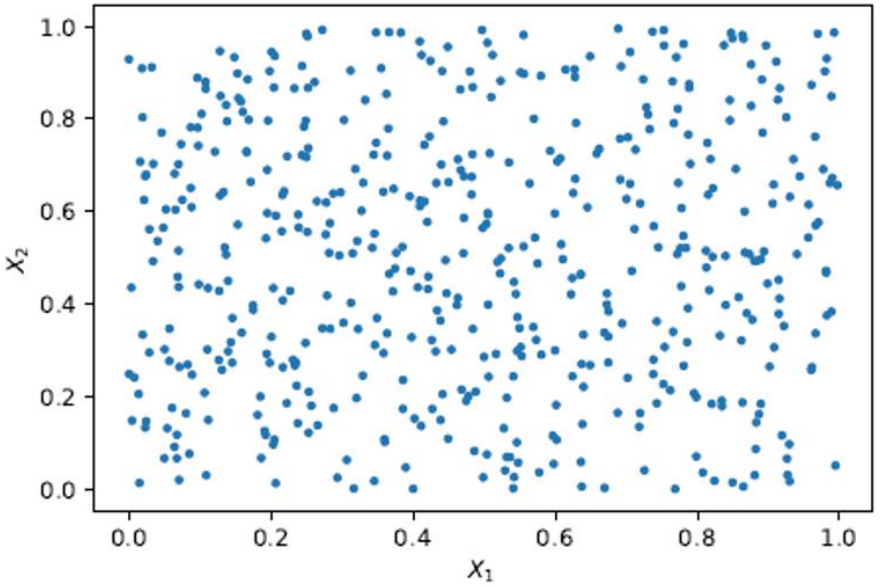
$$A = 65536, M = 2^{31}, C = 0,$$

其三重分布(U_i, U_{i+1}, U_{i+2}) 只分布在15 个空间平面上: (main_4-2.m)

- https://en.wikipedia.org/wiki/RANDU
- 在进行类似Monte Carlo 模拟时，要格外警惕类似的问题！！！

检验 - MT19937

用MT19937 产生一维50 万个随机数，并按定理次序重新组织成500 个1000维的随机向量。观察第(1,2)维和第(999,1000)维组成的平面随机数分布：



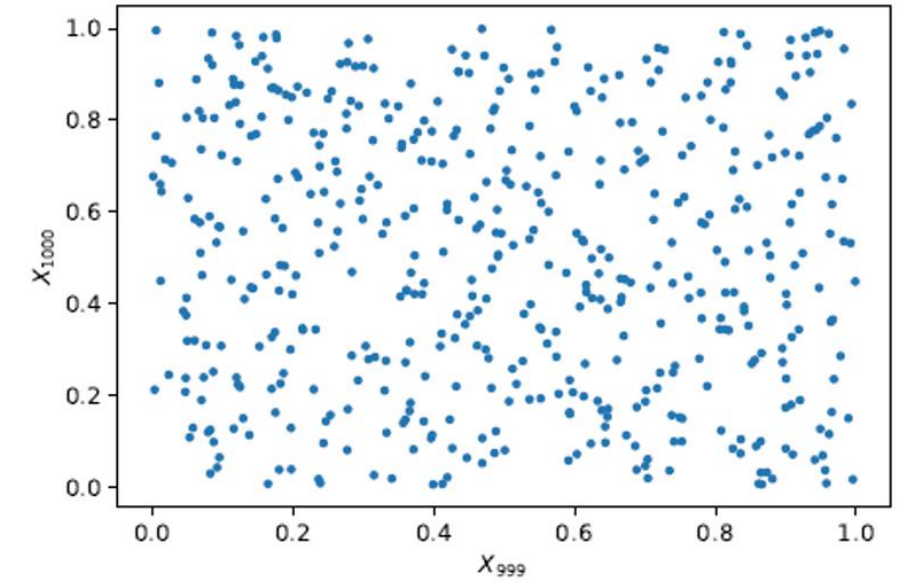
定理 5.1. 若 $\{A_i, i = 1, 2, \cdots, ns\}$ 是独立同分布的随机变量序列，则

多维随机数 $\{(A_{(i-1)s+1}, A_{(i-1)s+2}, \cdots, A_{is}), i = 1, 2, \cdots\}$

必是 s 维独立同分布的随机向量序列。而且每个分量序列

$$\begin{aligned} \mathrm{U}_1 &= \{U_1, U_{s+1}, \cdots, U_{(n-1)s+1}\}, \\ \mathrm{U}_2 &= \{U_2, U_{s+2}, \cdots, U_{(n-1)s+2}\}, \\ &\cdots \\ \mathrm{U}_s &= \{U_s, U_{2s}, \cdots, U_{ns}\}, \end{aligned}$$

在各自的维度中也是独立同分布的。



小结

伪随机数产生器的数学结构:

- 定义伪随机数的状态空间S为有限域
- 必须有初始状态S_0作为伪随机数初值
- 包含一个转换函数作为数学结构主体:

$$S_t = f(S_{t-1})$$

- 定义伪随机数的输出空间U通常是常数
- 包含一个输出函数把整随机数 --> (0,1):

$$U = g(S_t)$$

好的伪随机数特点(Tezuka,1995)

① 算法有数学理论支撑

② 可重复产生，不用存储

③ 产生速度快、占用内存少

④ 周期长，至少有10^{50}，如果问题需要N个随机数，则周期需要$2N^2$

⑤ 不产生0或1

⑥ 能通过统计检验

Homework 3

推荐收看报告： Gradient Based Online Learning for Queueing Systems: A Gentle Introduction
https://www.math.pku.edu.cn/kxyj/xsbg/tlb/informationsciences/153120.htm

1. 阅读相关文献，叙述一类新型随机数生成原理：如

   [1] BOX-MULLER法

   [2] https://github.com/Reputeless/Xoshiro-cpp

   [3] https://doi.org/10.1103/PhysRevLett.131.150601

   [4] …… （不仅限于上述所列资料中提及的方法）

3. 上机作业：调研如下任一随机数检验方法或工具

   - NIST测试方法
   - TestU01测试工具

   作业本上写一份1-2页篇幅的调研报告。

机器学习与数据科学博士生系列论坛（第五十九期）—— Gradient Based Online Learning for Queueing Systems: A Gentle Introduction

报告人：Jiadong Liang (PKU)
时间：2023-10-12 16:00-17:00
地点：腾讯会议 551-1675-5419

摘要：
Queueing systems find diverse applications across various domains. They are commonly used in telecommunications, transportation, computer resource allocation, and healthcare settings. These applications highlight the significance of queueing systems in optimizing processes and resource utilization. Within these systems, we often aim to achieve specific objectives by adjusting parameters, such as maximizing our net profit, minimizing the mean steady-state system time, or identifying Nash equilibrium points in queueing games. In highly streaming and non-iid data environments like queueing systems, our primary concern is how to efficiently and promptly identify target parameters. To tackle these issues, (stochastic) gradient descent is the top choice.
In this talk, we will explore classical queueing model problems, along with several mainstream algorithms for addressing them. These algorithms are all adaptations of SGD. Specially, we will focus on the dynamic pricing and capacity sizing problem in a G/G/1 service system, emphasizing the application of the SAMCMC algorithm. Leveraging SAMCMC, we can achieve the quickest statistical estimation of target parameters, minimizing our cumulative regret. Ultimately, we will explore how SAMCMC enables online statistical inference for target parameters in queueing systems, a capability that previous methods could not offer.
论坛简介：该线上论坛是由张志华教授机器学习实验室组织，每两周主办一次（除了公共假期）。论坛每次邀请一位博士生就某个前沿课题做较为系统深入的介绍，主题包括但不限于机器学习、高维统计学、运筹优化和理论计算机科学。