第10章 随机数

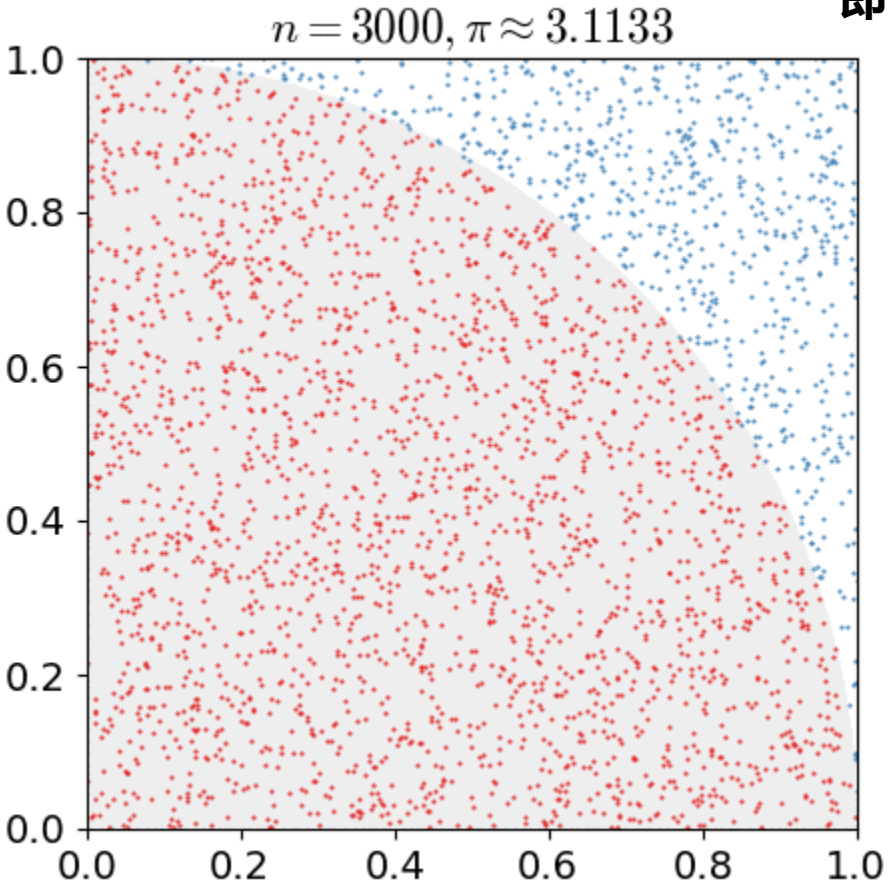
苏 芮

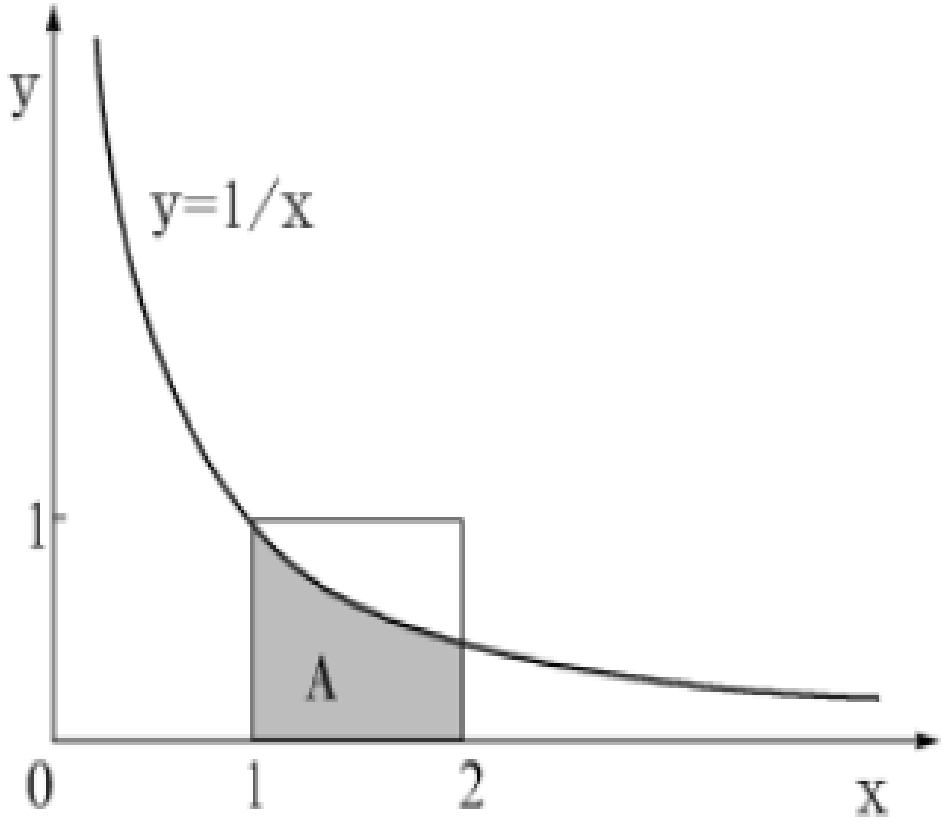
srhello@zju.edu.cn

开物苑4-202

蒙特卡洛法

取正方形，边长为1，以1为半径，做一个1/4的圆，一个人随机向正方形中投掷飞镖，总共投了n次，进行计数，在圆内的就是Sqrt(x^2+y^2)≤1，假如圆点数为m，圆外的就是sqrt(x^2+y^2)>1的，点数就为n-m，总的点数是n。投中的点数和其面积是存在比例关系的：
1/4圆的面积为1/4*π*1^2=π/4，正方形的面积为1
即(π/4)/1=m/n，即π=4*m/n





9.1 伪随机数

```
format long
rand
ans =
   0.814723686393179
```

计算机本质上是确定性的机器，如果没有接入γ射线计数器或者时钟之类的外部设备，那么的确只能算出伪随机数(Pseduorandom)。

9.2 均匀分布

#随机数生成过程（相乘取模算法）

有三个整数参数 a, c, m，以及一个称为种子的初始值 x0,

$$x_{k+1} = (ax_k + c) \bmod m$$

其中操作 mod m 的意思是除以 m 后取余数

当 a=13, c=0, m=31, x_0=1 时

$$x_1 = (ax_0 + c) \bmod m = (13 \times 1 + 0) \bmod 31 = 13$$

$$x_2 = (ax_1 + c) \bmod m = (13 \times 13 + 0) \times \bmod 31 = 14$$

$$x_3 = (ax_2 + c) \bmod m = (14 \times 13 + 0) \times \bmod 31 = 27$$

$$x_4 = (ax_3 + c) \bmod m = (27 \times 13 + 0) \times \bmod 31 = 10$$

…

这个序列以 m-1 为一个周期，若一个在 0-m 间取值的伪随机数整数系列除以 m，结果就是在[0,1]区间均匀分布的浮点数，简单的浮点系列就是：

1/31=0.032258065

13/31=0.419354839

14/31=0.451612903

27/31=0.870967742

10/31=0.322580645

…

在20世纪60年代，IBM大型计算机的科学子程序包中有一个RND或者RANDU的随机数发生器，它是参数a=65539, c=0, m=2^31的乘同余算法程序，用32比特位的字长，2^31的模运算可以非常快，a=2^16+3, 所以乘a运算可以通过一次位移和一次加法实现，对于那个时代的计算机，这些特殊设置显得很重要，然而导致产生的随机数系列很不理想。

9.2 均匀分布

$$x_{k+2}=\left(2^{16}+3\right)x_{k+1}=\left(2^{16}+3\right)^2 x_k=\left(2^{32}+6\cdot 2^{16}+9\right)x_k$$

进而有：

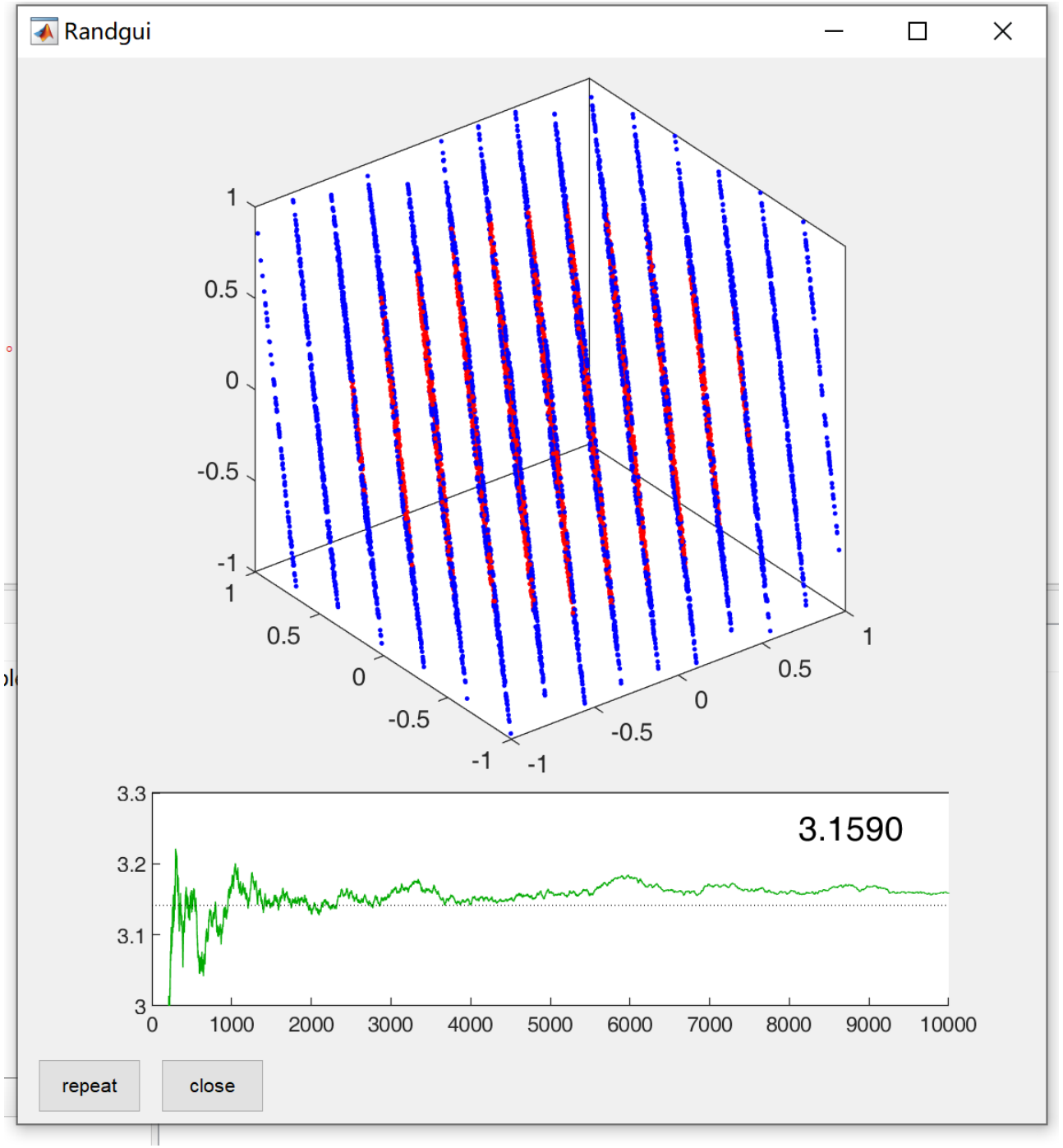
$$x_{k+2}=6x_{k+1}-9x_k$$

这样，由 RANDU 产生的系列中三个接续随机整数间，存在着特别高的相关性。

用 M-文件 randssp 实现了这个有缺陷的发生器，演示程序 randgui 试图通过在立方体内生成随机点和实际位于内球中的点数，计算 π 。可以看出所生成的点的分布模式与随机相差较远。

9.2 均匀分布

randgui randssp



9.2 均匀分布

Matlab 均匀随机数函数 rand 也是一个乘法同余生成器
采用的参数是：

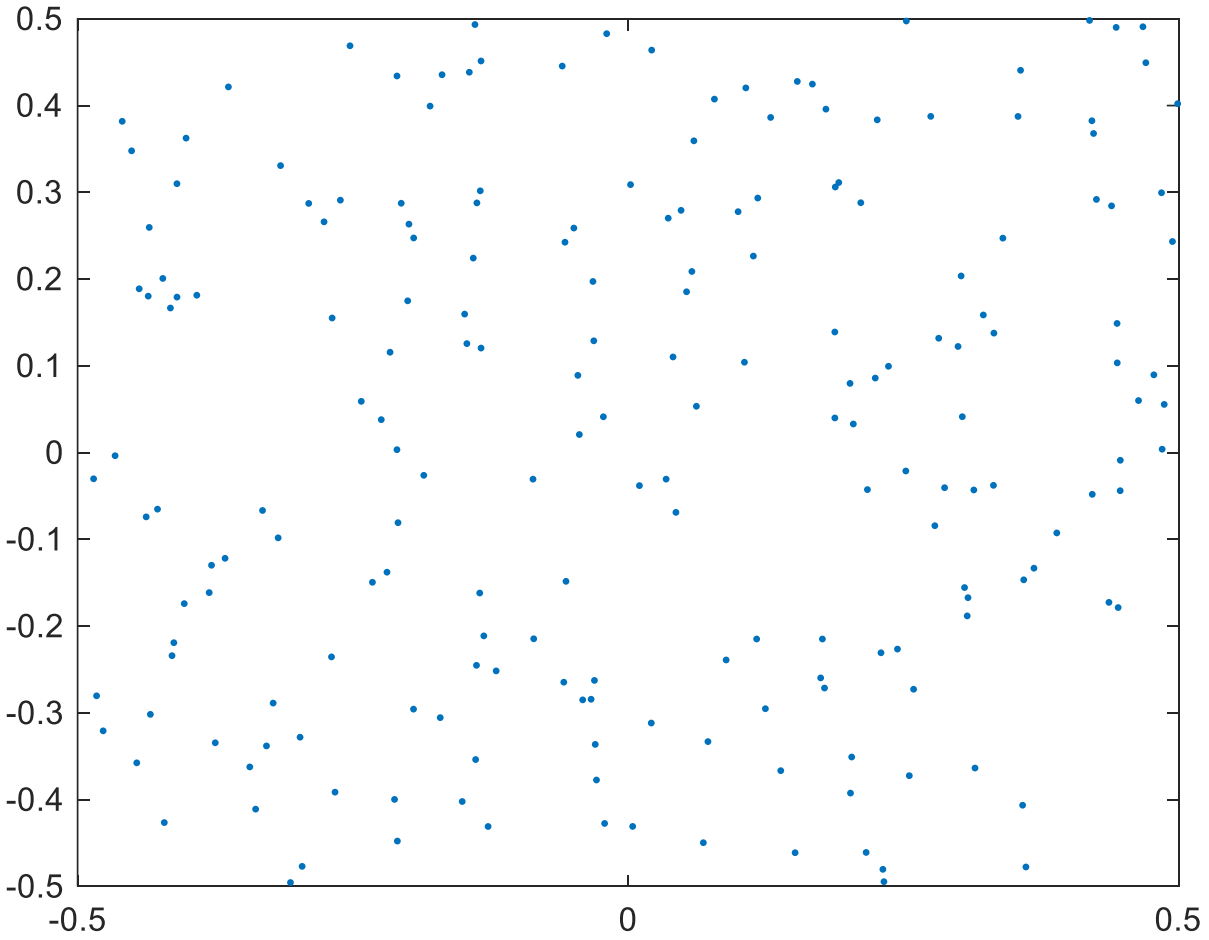
$$a = 7^5 = 16807$$

C=0

$$m = 2^{31} - 1 = 2147483647$$

#产生二维随机的点

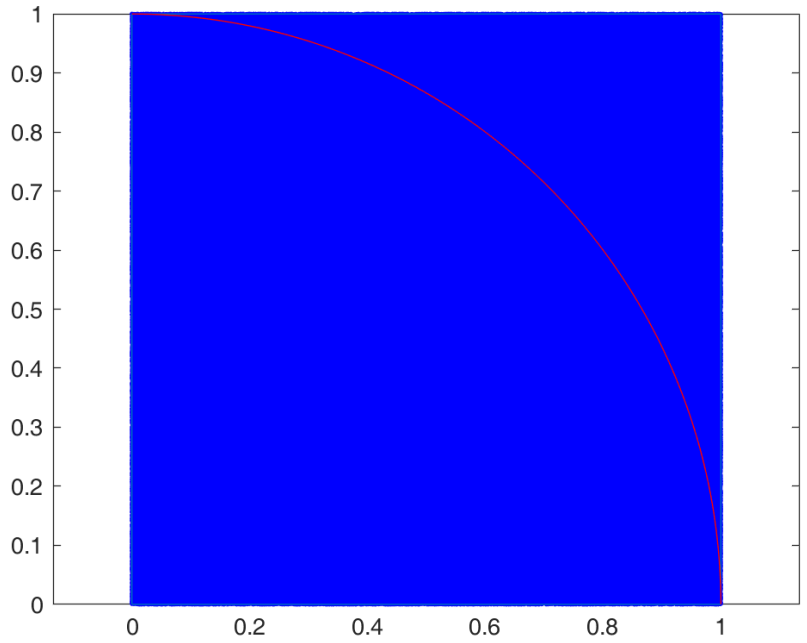
```
n=200
s=0.02
x=rand(n,1)-0.5;
y=rand(n,1)-0.5;
h=plot(x,y,'.');
```



9.2 均匀分布

蒙特卡罗方法

取正方形，边长为1，以1为半径，做一个1/4的圆，一个人随机向正方形中投掷飞镖，总共投了n次，进行计数，在圆内的就是Sqrt(x^2+y^2)≤1，假如圆点数为m，圆外的就是sqrt(x^2+y^2)>1的，点数就为n-m，总的点数是n。投中的点数和其面积是存在比例关系的：1/4圆的面积为1/4*π*1^2=π/4，正方形的面积为1,即(π/4)/1=m/n，即π=4*m/n



PI = 3.1417

```
#通过随机数求圆周率π
k=10000000;
n=0;
for i=1:k
      Rx=rand(1);
      Ry=rand(1);
      R1(i)=Rx;
      R2(i)=Ry;
      plot(Rx,Ry,’r.’);
       if (Rx^2+Ry^2)<1
       n=n+1;
       end
end
```

```
plot(R1,R2,'b.');
hold on
line([0  0],[1 0]);
line([1  0],[1 1]);
line([1  1],[0 1]);
line([0  1],[0 0]);
aplha=0:pi/200:pi/2;
r=1;
x=r*cos(aplha);
y=r*sin(aplha);
plot(x,y,'r-');
axis equal
n
PI=4*n/k
```

9.2 均匀分布

#有各种各样的随机数发生器，

RANDSSP：

Multiplicative congruential（乘法求余的）uniform random number generator. Based on the parameters used by IBM's Scientific Subroutine Package.

The statement

r = randssp

generates a single uniformly distributed random number.

The statement

r = randssp(m,n)

generates an m-by-n random matrix.

RANDMCG

Multiplicative congruential（乘法求余的）uniform random number generator.

Based on the parameters used by MATLAB version 4.

The statement

r = randmcg

generates a single uniformly distributed random number.

The statement

r = randmcg(m,n)

generates an m-by-n random matrix.

The statement

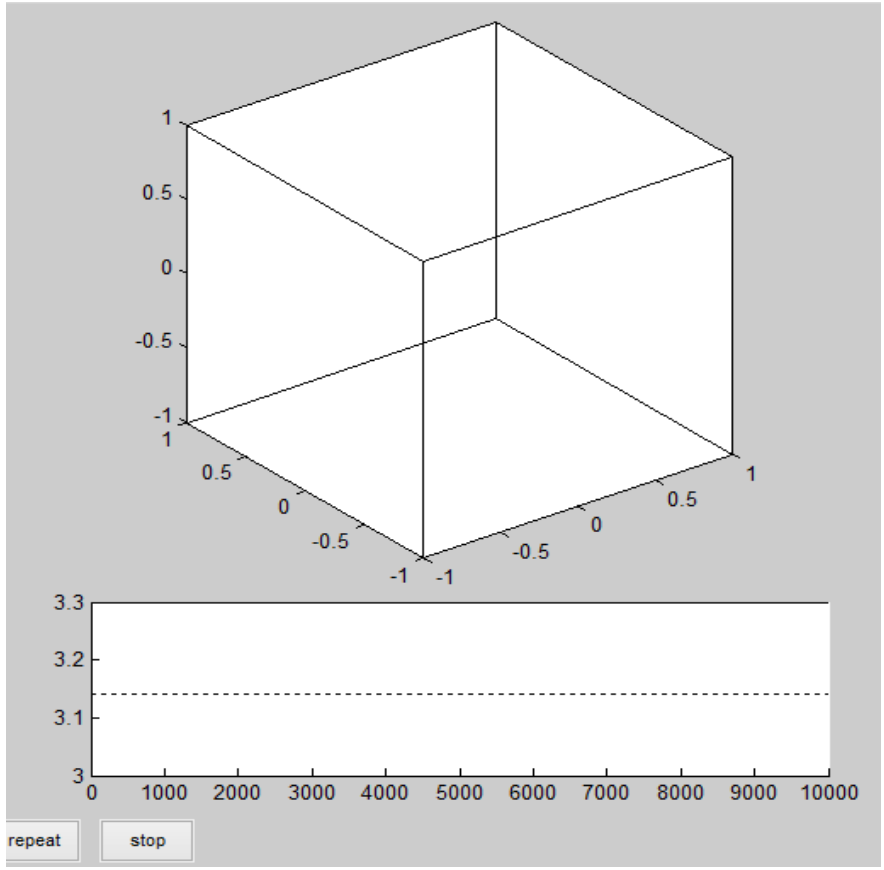
clear randmcg

will cause the generator to reinitialize itself. The function can not accept any other starting seed.

9.2 均匀分布

#rand gui
help randgui
RANDGUI Monte Carlo computation of pi. Generate random points in a cube and count the portion that are also in the inscribed sphere. The ratio of the volume of the sphere to the volume of the cube is pi/6. RANDGUI with no arguments or RANDGUI('rand') uses MATLAB's built-in random number generator. RANDGUI('randtx') uses our textbook version of the built-in generator. RANDGUI('randmcg') and RANDGUI('randssp') use different Lehmer congruential generators, one with good parameters and one with the parameters used years ago by IBM's "RANDU" function.

得到的显示就是π的值。



9.3 正态分布

rand(10,10,12)

ans(:,:,1) =

0.5038	0.5864	0.4035	0.9300	0.3037	0.2859	0.5039	0.0216	0.9437	0.8295
0.4896	0.6751	0.1220	0.3990	0.0462	0.5437	0.6468	0.5598	0.5492	0.8491
0.8770	0.3610	0.2684	0.0474	0.1955	0.9848	0.3077	0.3008	0.7284	0.3725
0.3531	0.6203	0.2578	0.3424	0.7202	0.7157	0.1387	0.9394	0.5768	0.5932
0.4494	0.8112	0.3317	0.7360	0.7218	0.8390	0.4756	0.9809	0.0259	0.8726
0.9635	0.0193	0.1522	0.7947	0.8778	0.4333	0.3625	0.2866	0.4465	0.9335
0.0423	0.0839	0.3480	0.5449	0.5824	0.4706	0.7881	0.8008	0.6463	0.6685
0.9730	0.9748	0.1217	0.6862	0.0707	0.5607	0.7803	0.8961	0.5212	0.2068
0.1892	0.6513	0.8842	0.8936	0.9227	0.2691	0.6685	0.5975	0.3723	0.6539
0.6671	0.2312	0.0943	0.0548	0.8004	0.7490	0.1335	0.8840	0.9371	0.0721

rand产生的随机数是在[0,1]区间，中间值为0.5，

9.3 正态分布

sum(rand(10,10,12),3)

相当于12个这样的矩阵(10x10矩阵)产生，比如对于第一个元素a11,这样就有12个a_{11}相加，再进行统计，可以看出，大量的数是出现在中值6（0.5*12=6）附近的，而在接近0和12的两端的数是非常少的。

6.659183831109826	6.934094616614512	6.904192144267246	7.233773238567934	6.058082958513223	8.183075105725422	7.512765879950106
4.517332524336851	4.487886713868327	5.148318325832814	4.251567316430137	7.288385565335308	5.815007263627963	6.806180085450174
8.065022055240423	5.214416224240884	5.451646049207861	5.670986824160620	7.460440327791865	5.331122094916143	5.578034920648124
7.311306874115938	5.432382033937075	3.646477109637095	5.674723126048631	6.293475856264219	5.620630836216390	6.681314591696142
4.441505196787316	6.629708940113408	6.759389522197200	4.180739628392453	5.436263451700198	6.962005230390943	6.001915880984126
5.741047201838313	6.644165924425623	5.509134158861755	6.140718709219993	7.631746596152198	6.139088306556457	7.796732393958655
5.704628194340191	6.961882679735798	5.707398622676520	6.322283353871290	6.456348458592885	5.093067246403186	6.520060279262913
6.868164078495877	5.690341585877375	6.656948151064507	4.718964356307923	6.468859592383787	6.048953948663819	7.069776742486853
5.106158575791468	6.925646537540262	7.439937790747368	4.677475539566274	6.708270191562927	5.465721664639327	4.365915393610285
4.978396254892249	7.298657677548459	5.650836863754434	5.534855585321583	6.670754492241358	5.492538758600110	6.634394700258396

5.993369803155398	5.677218672990740	4.556550698185593
5.857008042925397	5.166794639092593	4.274815239559230
7.552417800363925	4.841417869085396	7.746259473492658
6.089215778750011	5.344461953137081	4.640013484003480
5.666526713883860	4.649694832020137	7.592585264911508
5.951420554711067	5.936499983942772	6.812609160374124
5.425433035081498	6.956944010484107	8.321681177120931
6.086628106842046	3.650868432793284	5.413254466696539
6.162033502711143	7.394252521719729	7.447760763906707
6.560166176367596	5.524134743141197	7.322565111809404

9.3 正态分布

从 MATLAB 第 5 版开始，正态分布随机数生成器 randn 使用了一个复杂的查表算法，这个算法也是 George Marsaglia 提出的。Marsaglia 称之为金字形神塔算法。金字形神塔是古代美索不达米亚的叠层式塔，数学上称之为二维阶跃函数。一维的金字形神塔算法是 Marsaglia 提出的算法的基础。

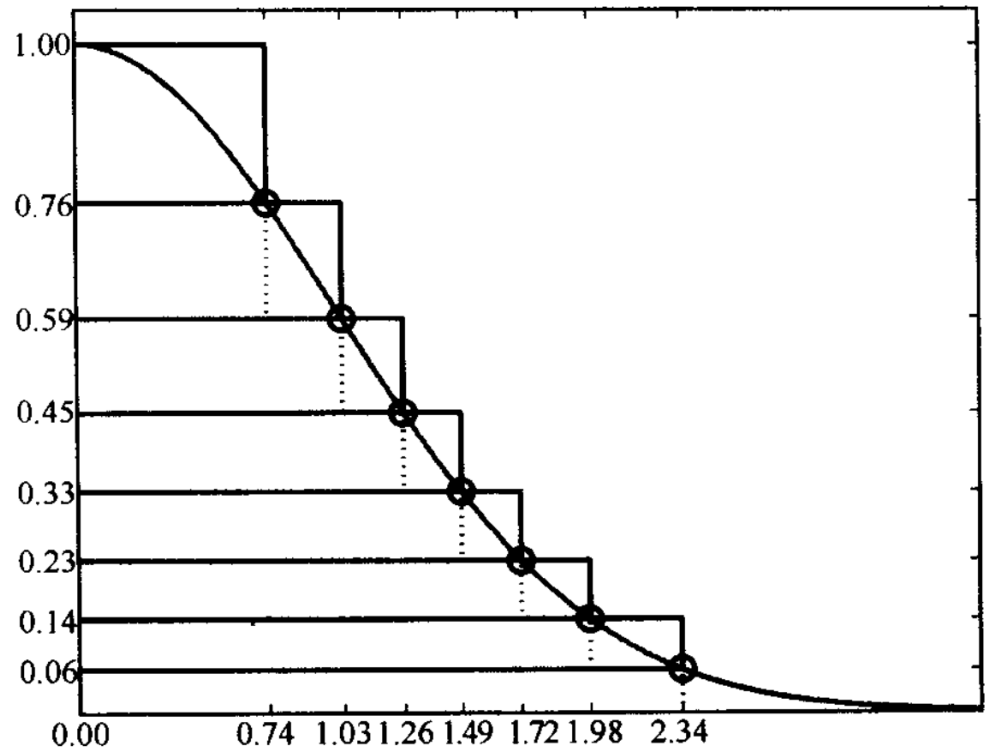


图 9-2　金字形神塔算法

9.4 randtx, randntx

对于均匀分布随机数生成器 randtx，状态变量 s 为含 35 个元素的向量。其中 32 个分量为 2^{-53}到 $1-2^{-53}$之间的浮点数，另外 3 个分量为 eps 的较小的整数倍。randtx 状态中所有可能的位模式的总数为 $2\cdot 32\cdot 2^{32}\cdot 2^{32\cdot 52}$，即 2^{1702}，虽然按默认的初始设置未必都能达到。

对于正态分布随机数生成器 randntx，状态 s 为两个 32 位的整数元素构成的向量，因此其总的状态数目为 2^{64}。

两个生成器程序在第一次使用或复位后都需要做一些启动运算。对于 randtx，启动过程生成状态向量中的初始浮点数，每次生成一位。对于 randntx，设置计算金字形神塔阶跃函数中的转折点。

感谢聆听,欢迎讨论!