计算物理 · 物理模拟II

COMPUTATIONAL PHYSICS · COMPUTER SIMULATION
MONTE CARLO METHOD

蒙特卡洛

- Monte Carlo is the largest city in Monaco
- The area is 5 km^2, with around 400 people living there
- also the center of gambling, politics, fireworks and culture
- it has lots of casinos, hotels and historical buildings

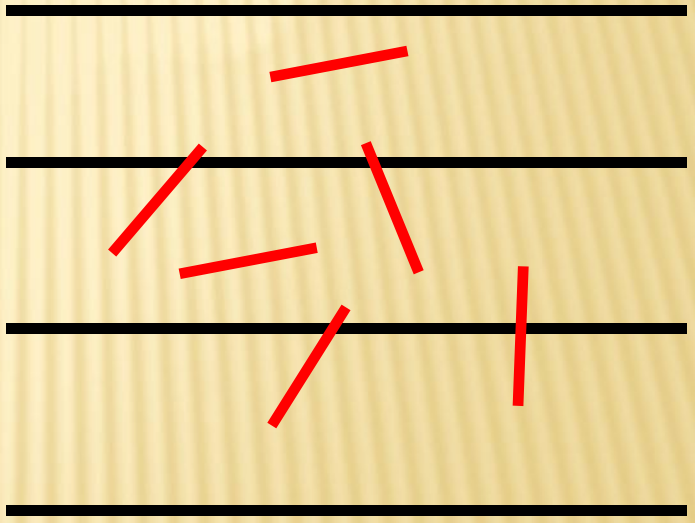
蒙特卡洛（Monte Carlo）方法是一种应用随机数来进行计算机模拟的方法．此方法对研究的系统进行随机观察抽样，通过对样本值的观察统计，求得所研究系统的某些参数．

把物理问题变换成几率问题

普丰投针法

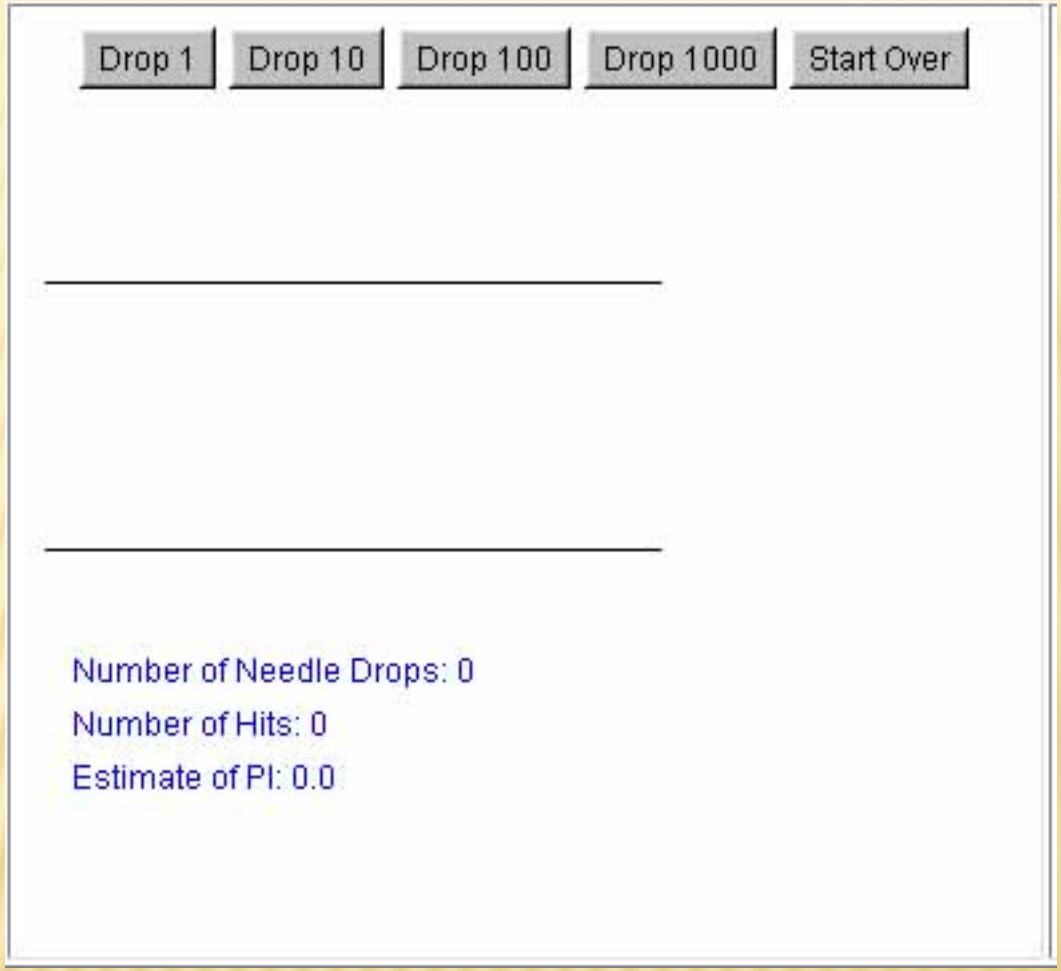
Buffon's needle for **π**

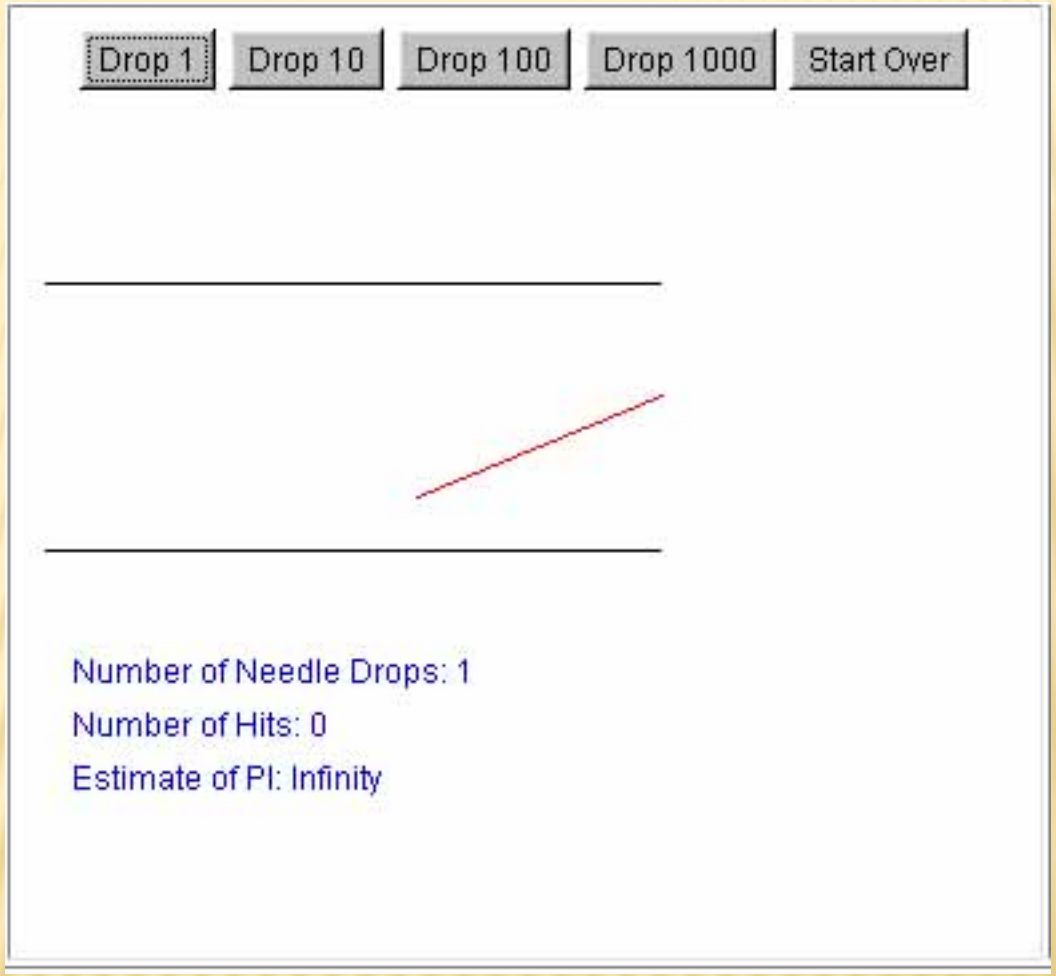
- 针的长度 ***l***
- 线间距离 ***d>l***

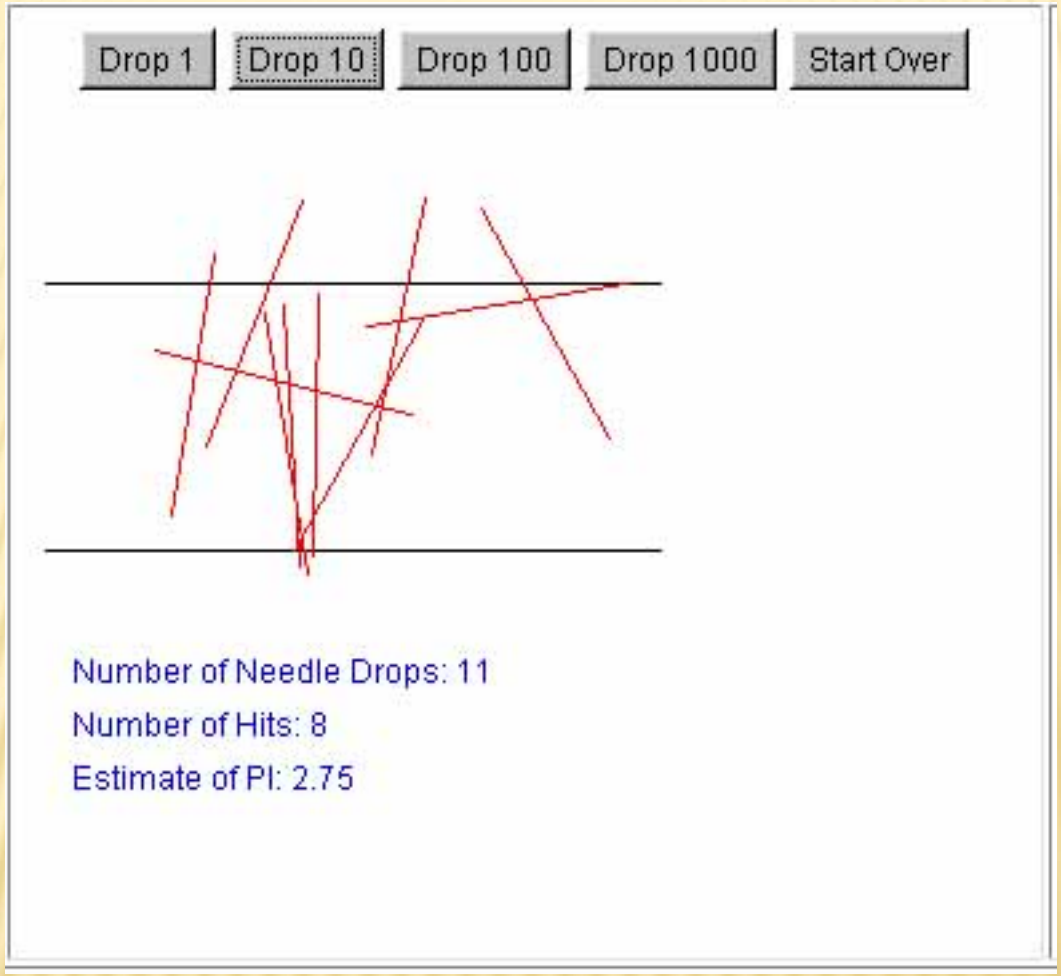


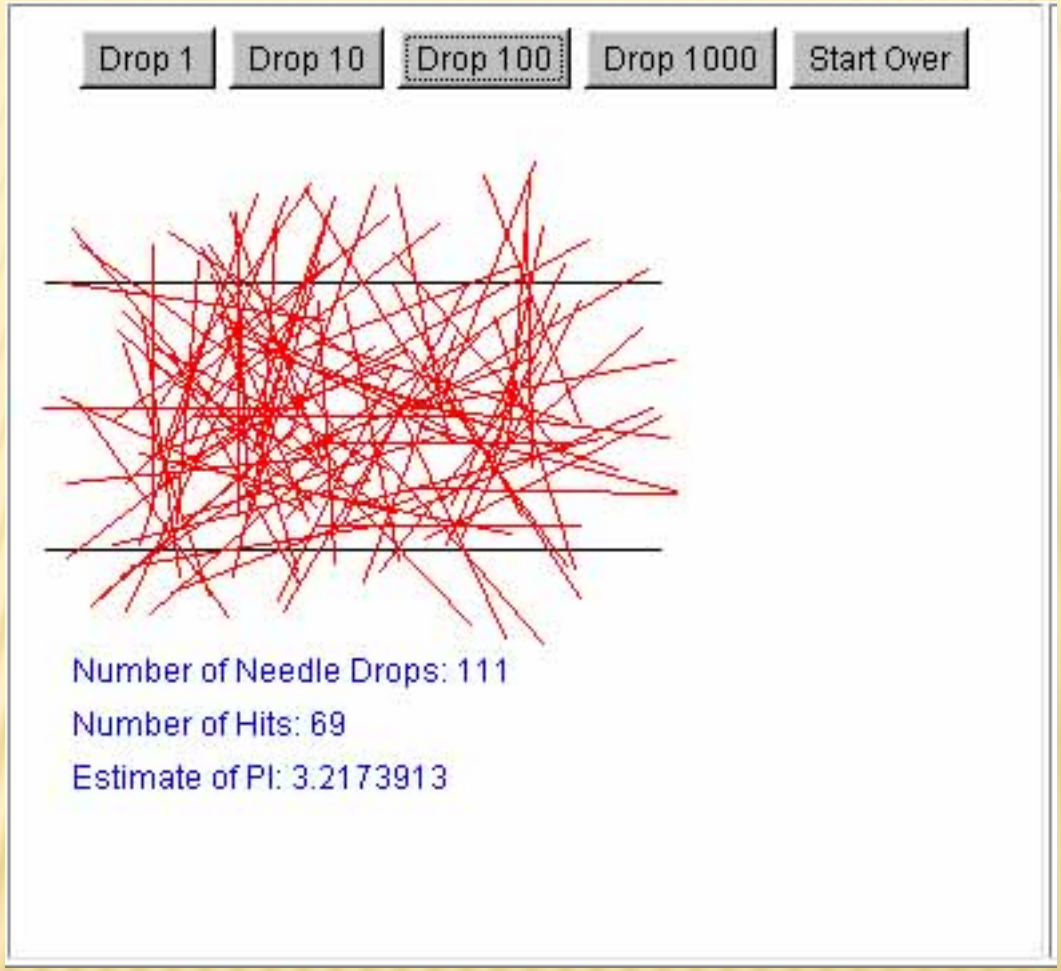
***P* (needle crosses a line) =**

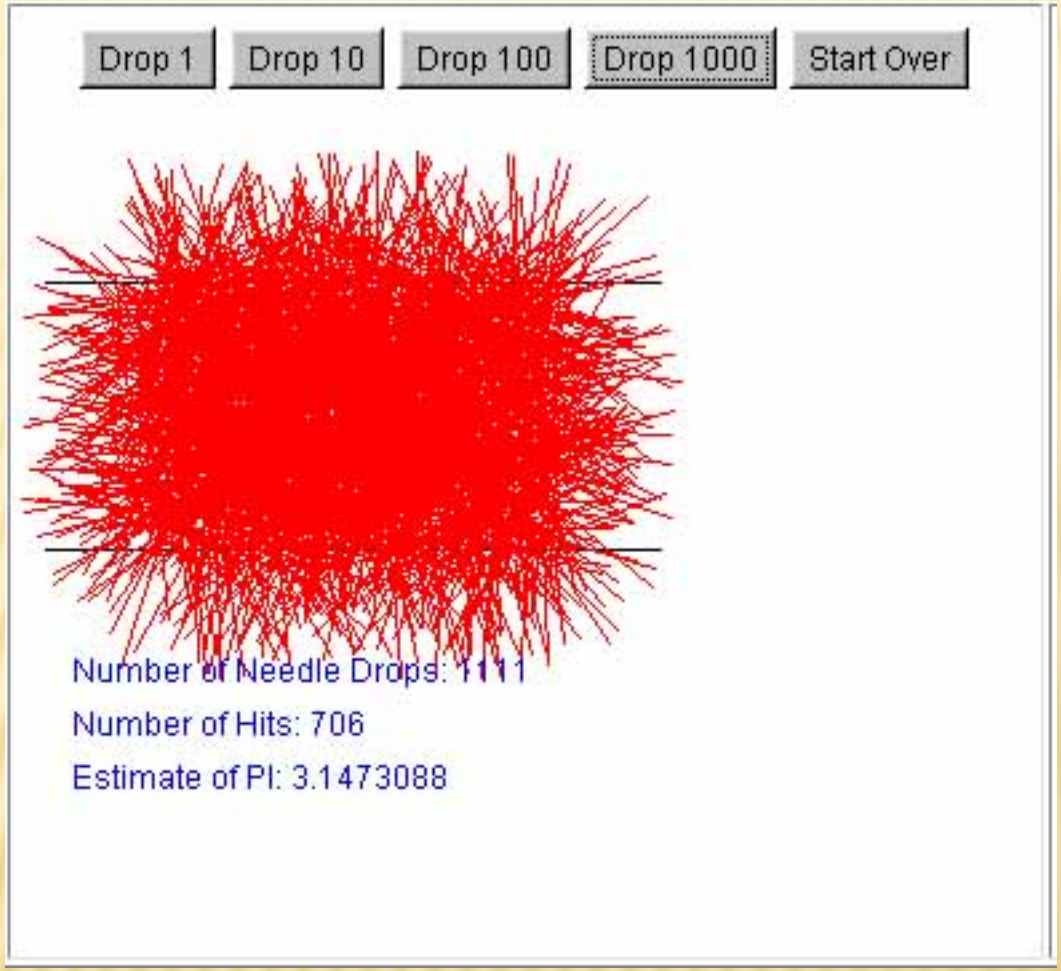
$$2\cdot\int_{-\pi/2}^{\pi/2}\frac{l\cos\theta}{d}\frac{d\theta}{2\pi}=\frac{2l}{\pi d}$$


Drop 1
Drop 10
Drop 100
Drop 1000
Start Over
Number of Needle Drops: 0
Number of Hits: 0
Estimate of PI: 0.0


Drop 1
Drop 10
Drop 100
Drop 1000
Start Over
Number of Needle Drops: 1
Number of Hits: 0
Estimate of PI: Infinity


Drop 1
Drop 10
Drop 100
Drop 1000
Start Over
Number of Needle Drops: 11
Number of Hits: 8
Estimate of PI: 2.75


Drop 1
Drop 10
Drop 100
Drop 1000
Start Over
Number of Needle Drops: 111
Number of Hits: 69
Estimate of PI: 3.2173913

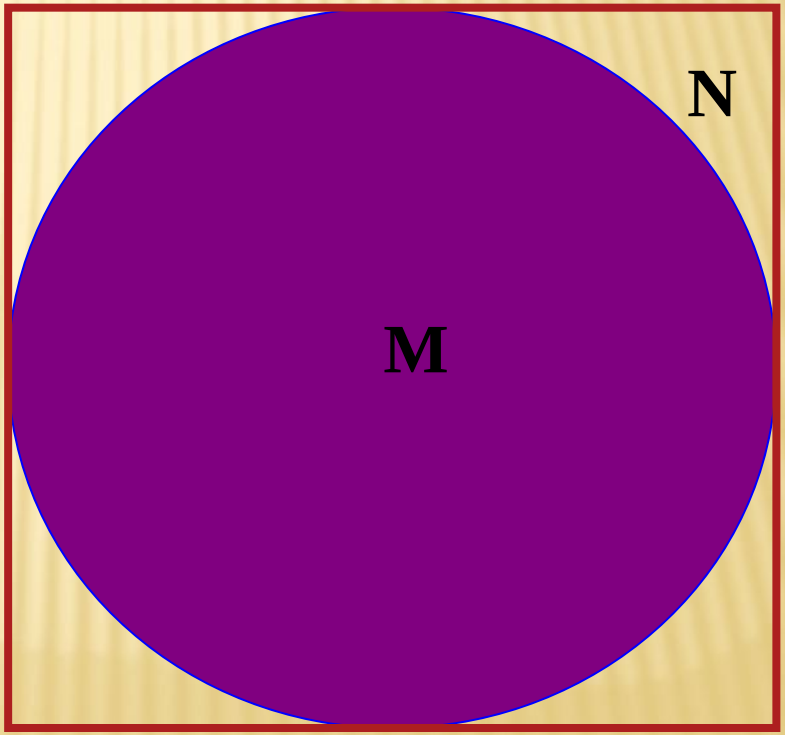

Drop 1
Drop 10
Drop 100
Drop 1000
Start Over
Number of Needle Drops: 1111
Number of Hits: 706
Estimate of PI: 3.1473088

求“PI”

- 射击“Shooting”
  - 方框
  - 内切圆

$$M / N = \pi R^2 / (2R)^2$$

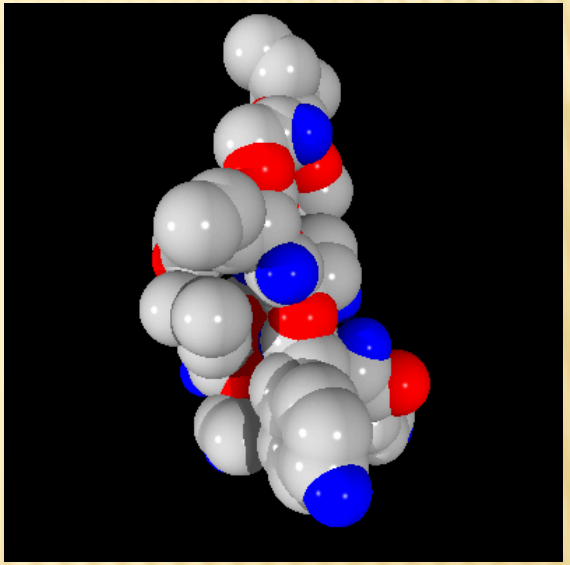
$$\pi = M / N \cdot (2R)^2 / R^2 = 4M / N$$

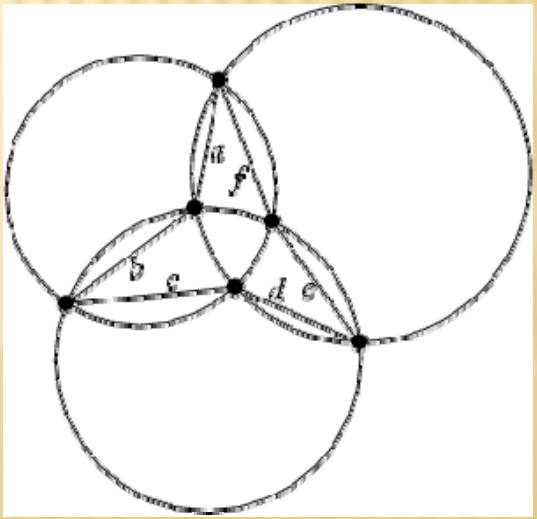


求复杂分子的面积、体积

- 球面(内)均匀打点
- 判据

  哪些点位于球面外(内)





思考：如何生成均匀分布于球面的点？

-- Ask Google

蒙特卡洛方法的特点

- 误差 error~ 1/sqrt(N)
- $N=10^6$ → accuracy of 0.1%, → 慢!
- 形式简单
- 收敛有保障，more is better
- 对高维空间有优越性
  - 一维积分，梯度法→ error ~ $1/N^2$
  - 高维(d维)积分，梯度法→ error ~ $1/N^{2/d}$
  - MC → 始终为 1/sqrt(N)
- 对实际问题，空间的维数可以极其巨大!

随 机 数

- 真随机数
- 伪随机数

真随机数

- 真随机数数列是不可预计的，因而也不可能重复产生两个相同的真随机数数列。
- 真随机数只能用某些随机物理过程来产生。例如：放射性衰变、电子设备的热噪音、宇宙射线的触发时间等等。
- 如果采用随机物理过程来产生蒙特卡洛计算用的随机数，理论上不存在什么问题。但在实际应用时，要做出速度很快（例如每秒产生上百个浮点数），而又准确的随机数物理过程产生器是非常困难的。

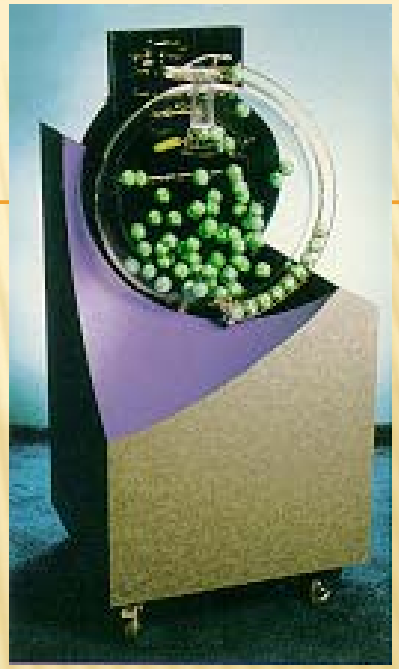
伪随机数

实际应用的随机数通常都是通过某些数学公式计算而产生的伪随机数。这样的伪随机数从数学意义上讲已经一点不是随机的了。但是，只要伪随机数能够通过随机数的一系列的统计检验，我们就可以把它当作真随机数而放心地使用。这样我们就可以很经济地、重复地产生出随机数。

对物理问题的计算机模拟所需要的伪随机数应当满足如下的标准：

理论上，要求伪随机数产生器具备以下特征：良好的统计分布特性、高效率的伪随机数产生、伪随机数产生的循环周期长，产生程序可移植性好和伪随机数可以重复产生。其中满足良好的统计性质是最重要的。

随机数生成



- 线性同余算法:
  - http://en.wikipedia.org/wiki/Linear_congruential_generator
- Mersenne twister
  - http://en.wikipedia.org/wiki/Mersenne_twister
- Linear feedback shift register
  - http://en.wikipedia.org/wiki/Linear_feedback_shift_register

线性同余算法 (Linear congruential generator)

$$X_{n+1} = (aX_n + c) \bmod m$$

The period of a general LCG is at most m, and for some choices of a much less than that. Provided that c is nonzero, the LCG will have a full period for all seed values if and only if:

- c and m are relatively prime (no common factors other than 1),
- a-1 is divisible by all prime factors of m,
- a-1 is a multiple of 4 if m is a multiple of 4.

Advantages & Disadvantages of LCG

- Fast, minimal memory
- LCG should not be used for applications where high-quality randomness is critical (such as in Monte Carlo and cryptographic applications)
- serial correlation between LCG generated numbers
- The lower-order bits of the generated sequence have a far shorter period than the sequence as a whole if m is set to a power of 2.

The random generators using LCG in some popular compilers

Source	m	a	c	output bits of seed in *rand()* / *Random(L)*
Numerical Recipes	2^{32}	1664525	1013904223	
Borland C/C++	2^{32}	22695477	1	bits 30..16 in *rand()*, 30..0 in *lrand()*
glibc (used by GCC)[4]	2^{32}	1103515245	12345	bits 30..0
ANSI C: Watcom, Digital Mars, CodeWarrior, IBM VisualAge C/C++	2^{32}	1103515245	12345	bits 30..16
Borland Delphi, Virtual Pascal	2^{32}	134775813	1	bits 63..32 of *(seed * L)*
Microsoft Visual/Quick C/C++	2^{32}	214013	2531011	bits 30..16
Apple CarbonLib	$2^{31} - 1$	16807	0	see Park-Miller RNG
MMIX by Donald Knuth	2^{64}	6364136223846793005	1442695040888963407	
VAX's MTH$RANDOM,[5] old versions of glibc	2^{32}	69069	1	
Random class in Java API	2^{48}	25214903917	11	bits 48...17

As shown above, LCGs do not always use all of the bits in the values they produce. The Java implementation produces 48 bits with each iteration but only returns the 32 most significant bits from these values. This is because the higher-order bits have longer periods than the lower order bits (see below). LCGs that use this technique produce much better values than those that do not.

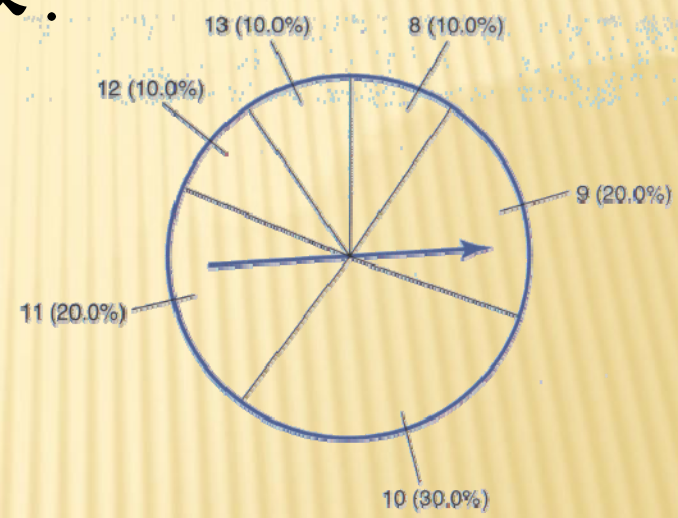
LCG产生的是均匀分布的伪随机数。

如何产生具有某确定分布的随机数？

一、离散分布的随机变量抽样

例：要求按下表产生随机数：

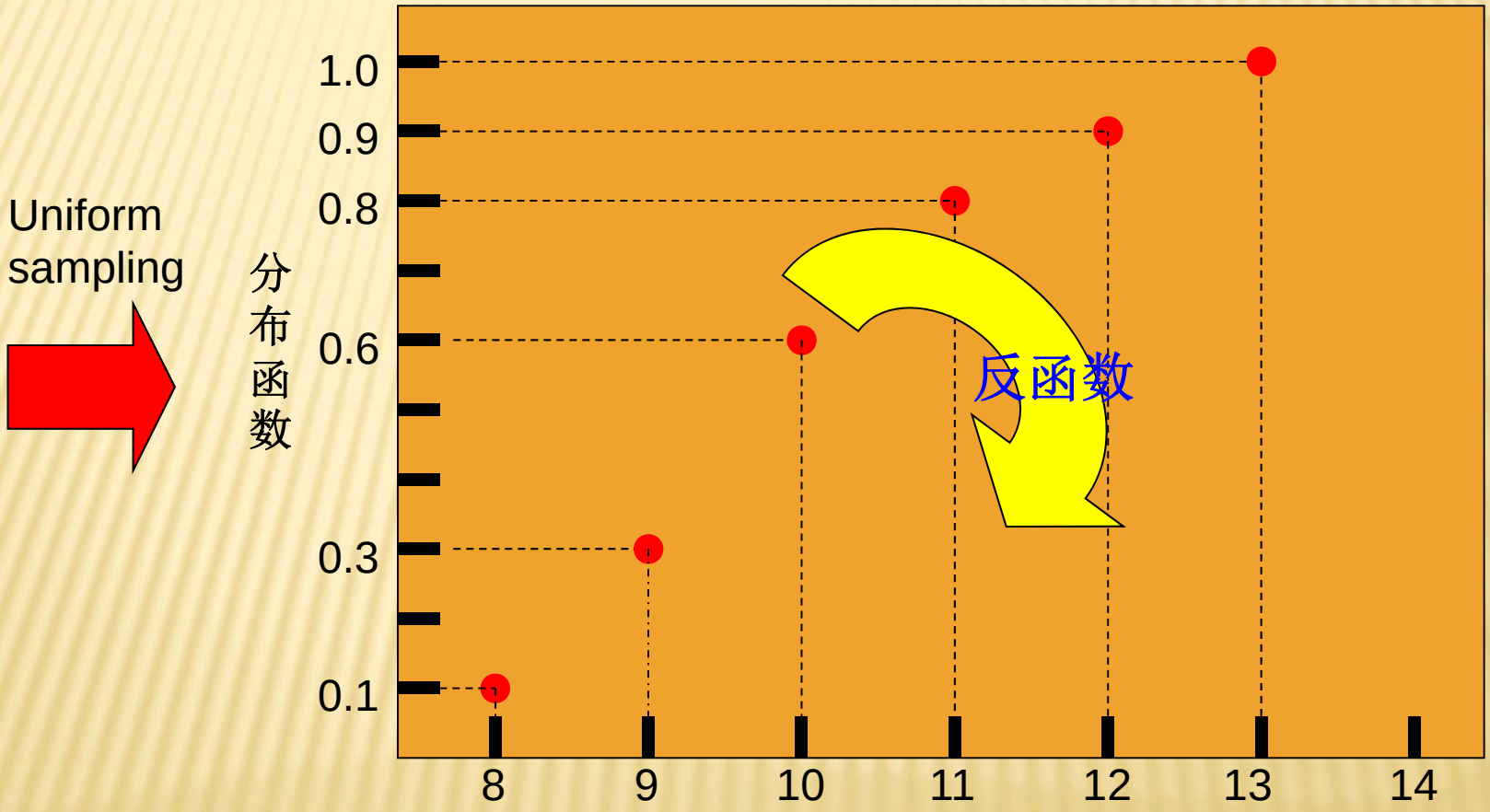
	A	B
1	Demand	Probability
2	8	10%
3	9	20%
4	10	30%
5	11	20%
6	12	10%
7	13	10%
8		100%



- 10% of the interval (0.0 to 0.09999) is mapped (assigned) to a demand d= 8.
- 20% of the interval (0.1 to 0.29999) is mapped to d =9.
- 30% of the interval (0.3 to 0.59999) is mapped to d =10.
- etc., etc.

Random Number	Demand
0.0-0.09999	8
0.1-0.29999	9
0.3-0.59999	10
0.6-0.79999	11
0.8-0.89999	12
0.9-0.99999	13

一、离散分布的随机变量抽样



一、离散分布的随机变量抽样

如果离散型随机变量 x 以概率 p_i 取值 $x_i(i=1,2,\ldots)$，则其分布函数为：

$$F(x)=\sum_{x_i\le x} p_i \quad .$$

其中 p_i 应满足归一化条件：$\sum_i p_i=1$。该随机变量的直接抽样方法如下：首先选取在[0，1]区间上的均匀分布的随机数 ξ，然后判断满足如下不等式

$$F(x_{j-1})\le \xi < F(x_j)$$

的 j 值，与 j 对应的 x_j 就是所抽子样的一个抽样值，即 $\eta = x_j$。该子样具有分布函数 $F(x_j)$。

二、连续分布的随机变量抽样

直接抽样法(反函数法)

直接抽样法又称为反函数法。设连续型随机变量 η 的分布密度函数为 $f(x)$，在数学上它的分布函数应当为

$$F(x)=\int_{-\infty}^{x} f(x)dx \quad .$$

得到的 $\eta=F^{-1}(\xi)$ 即为满足分布密度函数 $f(x)$ 的一个抽样值。

例　对指数分布的直接抽样。

解　指数分布的问题可用于描述粒子运动的自由程，粒子衰变寿命或射线与物质作用长度等许多物理问题。它的分布密度函数为

$$f(x)=\begin{cases}\lambda e^{-\lambda x}, x>0, \lambda>0\\ 0, \text{其它.}\end{cases}$$

它的分布函数为

$$F(x)=\int_{-\infty}^{x} f(t)dt=\int_{0}^{x}\lambda e^{-\lambda t}dt=1-e^{-\lambda x}\quad .$$

设ξ是[0，1]区间上的均匀分布的随机数，令$\xi=F(\eta)=1-e^{-\lambda\eta}$，解此方程得到

$$\eta = -\frac{1}{\lambda}\ln(1-\xi) \quad .$$

注意到$1-\xi$和ξ同样服从[0，1]区间的均匀分布，故有

$$\eta = -\frac{1}{\lambda}\ln\xi \quad .$$

例　对如下的分布密度函数抽样

$$f(x)=\left(\frac{\gamma-1}{x_0^{\gamma-1}}\right)x^{-\gamma}，\quad x_0\le x,\gamma>1.$$

解　（　　上式的分布密度函数的对应分布函数为

$$F(x)=\int_{x_0}^{x}f(x)dx\Big/\int_{x_0}^{+\infty}f(x)dx=1-\left(\frac{x_0}{x}\right)^{\gamma-1}.$$

在[0，1]区间上的随机抽取均匀分布的随机数ξ，令

$\xi=F(\eta)=1-\left(\frac{x_0}{x}\right)^{\gamma-1}$，解此方程，并考虑到到$1-\xi$和$\xi$都是[0，1]

区间的均匀分布的伪随机数，得到

$$\eta=x_0\xi^{-1/(\gamma-1)}.$$

三、舍选抽样法

基本思想：按照给定的分布密度函数 $f(x)$，对均匀分布的随机数序列 $\{\xi_n\}$ 进行舍选。舍选的原则是在 $f(x)$ 大的地方，保留较多的随机数 ξ_i；在 $f(x)$ 小的地方，保留较少的随机数 ξ_i，使得到的子样中 ξ_i 的分布满足分布密度函数 $f(x)$ 的要求。

这种方法对分布密度函数 $f(x)$ 在抽样范围内有界，且其上界是容易得到的情况，是可以采用的。它使用起来十分灵活，计算也较简单，因而使用也比较广泛。

这种方法，对 $f(x)$ 在抽样范围内函数值变化很大的时候，效率是很低的，因为大量的均匀分布抽样点被舍弃了。

设随机变量η在[a, b]上的分布密度函数为$f(x)$， $f(x)$的在区间[a, b]上的最大值存在，并等于

$$L = \max f(x)$$

则： $f(x)/L$ 在**[a,b]**区间的取值为**[0,1]**

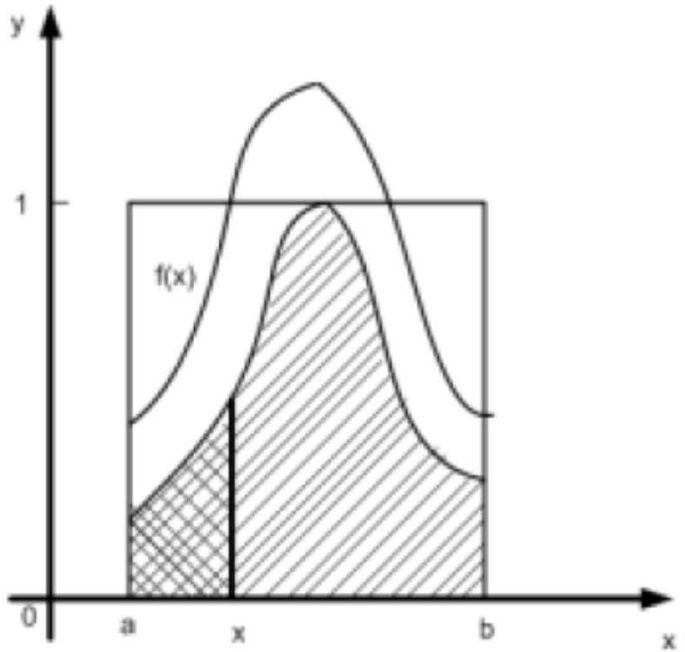
采用舍选法的步骤为：

(a) 选用均匀的[0，1]区间的随机数ξ_1，构造出[a, b]区间上的均匀分布的随机数$\delta = a + (b-a)\xi_1$。

(b) 再选取独立的均匀分布于[0，1]区间上的随机数ξ_2，判断$\xi_2 \le f(\delta)/L$是否满足。如满足上面不等式，则执行(c)；如不满足，则返回到步骤(a)。

(c) 选取$\eta = \delta$作为一个抽样值。

重复上述步骤可得到满足f(x)的随机数序列



例　对随机变量η抽样。它的分布密度函数为

$$f(x)=\begin{cases}2x, 0\le x\le 1,\\ 0, \text{其它}.\end{cases}$$

解　如果用直接抽样法，首先求出分布函数

$$F(x)=x^2 \quad .$$

抽取在[0，1]区间上的均匀分布的随机数ξ。令

$$\xi = x^2 \quad .$$

则有

$$x=\sqrt{\xi} \quad .$$

x 为η的子样的一个个体。但是开方运算量较大，可改用舍选法来做。

$$L \equiv \max_{x\in[0,1]} f(x) = \max_{x\in[0,1]} 2x = 2 \quad .$$

依照第一类舍选法步骤：

1. 依次产生独立的[0，1]区间上的均匀分布的随机数 ξ_1, ξ_2，
2. 判断 $\xi_2 \le \frac{1}{L} f(\xi_1) = \xi_1$ 是否成立。
3. 若成立，则取 $x = \xi_1$；
4. 若上面不等式不成立，可以再产生一组 ξ_1, ξ_2 进行重复试验。

多维随机向量抽样舍选法

设随机向量变量 $\vec{\eta}$ 的各分量为 $\eta_1, \eta_2, \cdots, \eta_n$ ，

$$\vec{\eta} = (\eta_1 \eta_2 \eta_3 \cdots \eta_n)^T .$$

它的联合分布密度函数为 $f(x_1, x_2, \cdots, x_n)$， 抽样范围在平行多面体

$$\{a_1 \le x_1 \le b_1, a_2 \le x_2 \le b_2, \cdots, a_n \le x_n \le b_n\}$$

内。令在该范围内，

$$L = \sup f(x_1, x_2, \cdots, x_n) < +\infty$$

我们将一维舍选法推广到这里，得到 n 维舍选法的做法如下：

（1）产生 n+1 个[0，1]上的均匀分布随机数 $\xi_1, \xi_2, \cdots, \xi_{n+1}$，

（2）然后判断如下不等式

$$\xi_{n+1} < \frac{1}{L} F[(b_1 - a_1)\xi_1 + a_1, (b_2 - a_2)\xi_2 + a_2, \cdots, (b_n - a_n)\xi_n + a_n] \quad \cdot$$

是否成立。

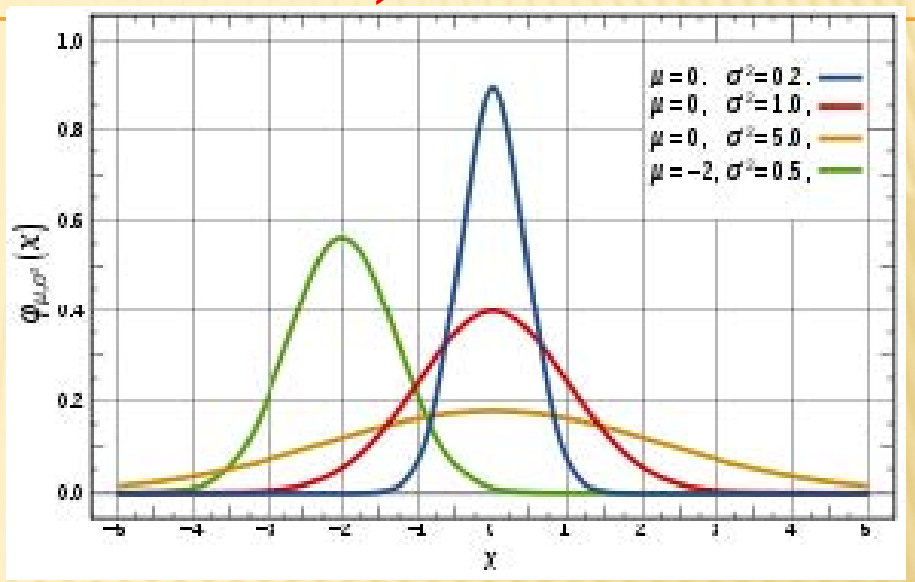
（3）若不等式成立，则得到 $\bar{\eta}$ 的一个抽样值，该向量的各个分量值为 $\eta_i = (b_i - a_i)\xi_i + a_i$，　　$(i = 1,2,\cdots,n)$

（4） 若不等式不成立，再重新产生 n+1 个随机数 ξ_i，

（5） 重复上面的步骤，直至该不等式成立。

四、生成高斯分布(Gaussian Distribution)

$$\frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$



Normalized Gaussian distribution

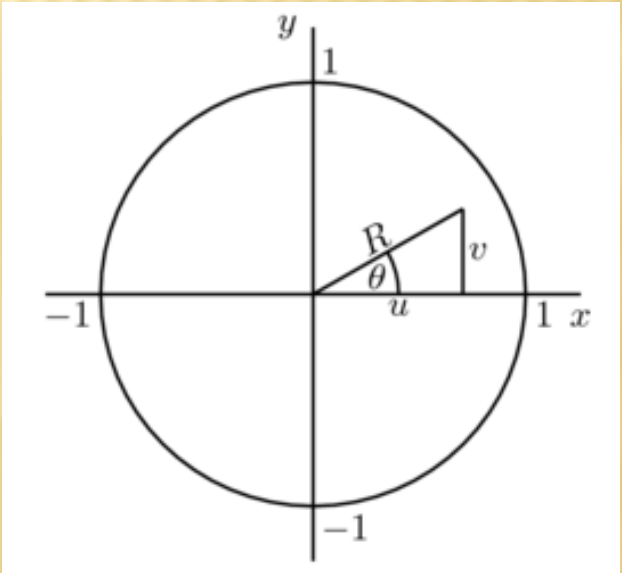
$$\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2},$$

- **Box-Muller**

$$z_0 = \sqrt{-2\ln s}\left(\frac{u}{\sqrt{s}}\right) = u\sqrt{\frac{-2\ln s}{s}}$$

$$z_1 = \sqrt{-2\ln s}\left(\frac{v}{\sqrt{s}}\right) = v\sqrt{\frac{-2\ln s}{s}}$$

$$s = u^2 + v^2$$



http://en.wikipedia.org/wiki/Box_Muller_transform

具体程序

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

    double gaussrand()
    {
        static double V1, V2, S;
        static int phase = 0;
        double X;

        if(phase == 0) {
            do {
                double U1 = (double)rand() / RAND_MAX;
                double U2 = (double)rand() / RAND_MAX;

                V1 = 2 * U1 - 1;
                V2 = 2 * U2 - 1;
                S = V1 * V1 + V2 * V2;
            } while(S >= 1 || S == 0);

            X = V1 * sqrt(-2 * log(S) / S);
        } else
            X = V2 * sqrt(-2 * log(S) / S);

        phase = 1 - phase;

        return X;
    }
```

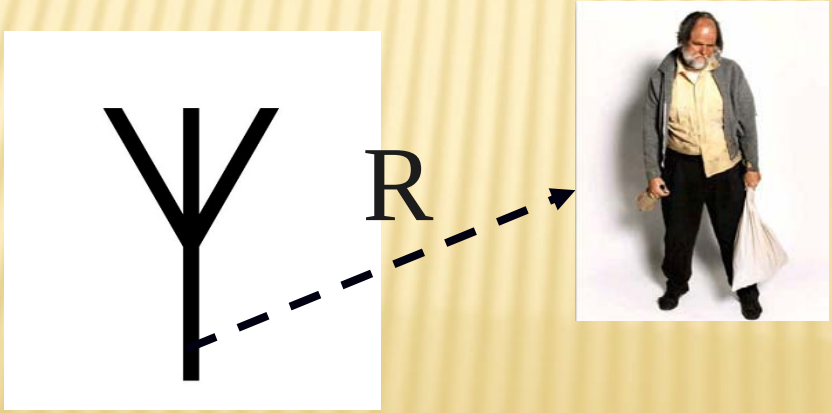
直接Monte-Carlo模拟(I)

随机游走(random walk)

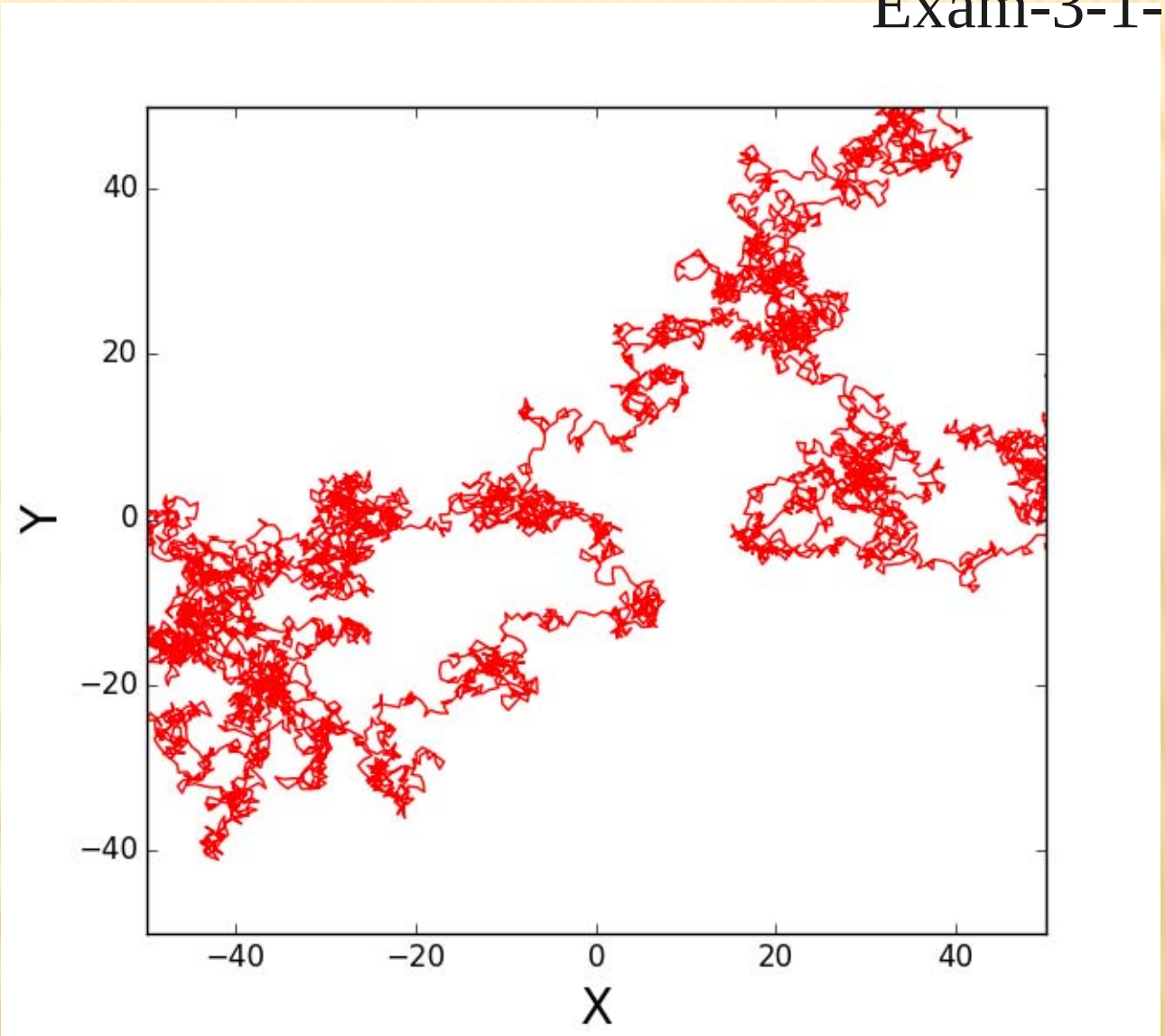
醉汉开始从一根电线杆出发，坐标为(0.0,0.0), 醉汉的步长为1，他走的每一步是随机的。

1）N步后醉汉相对电线杆位置的分布；

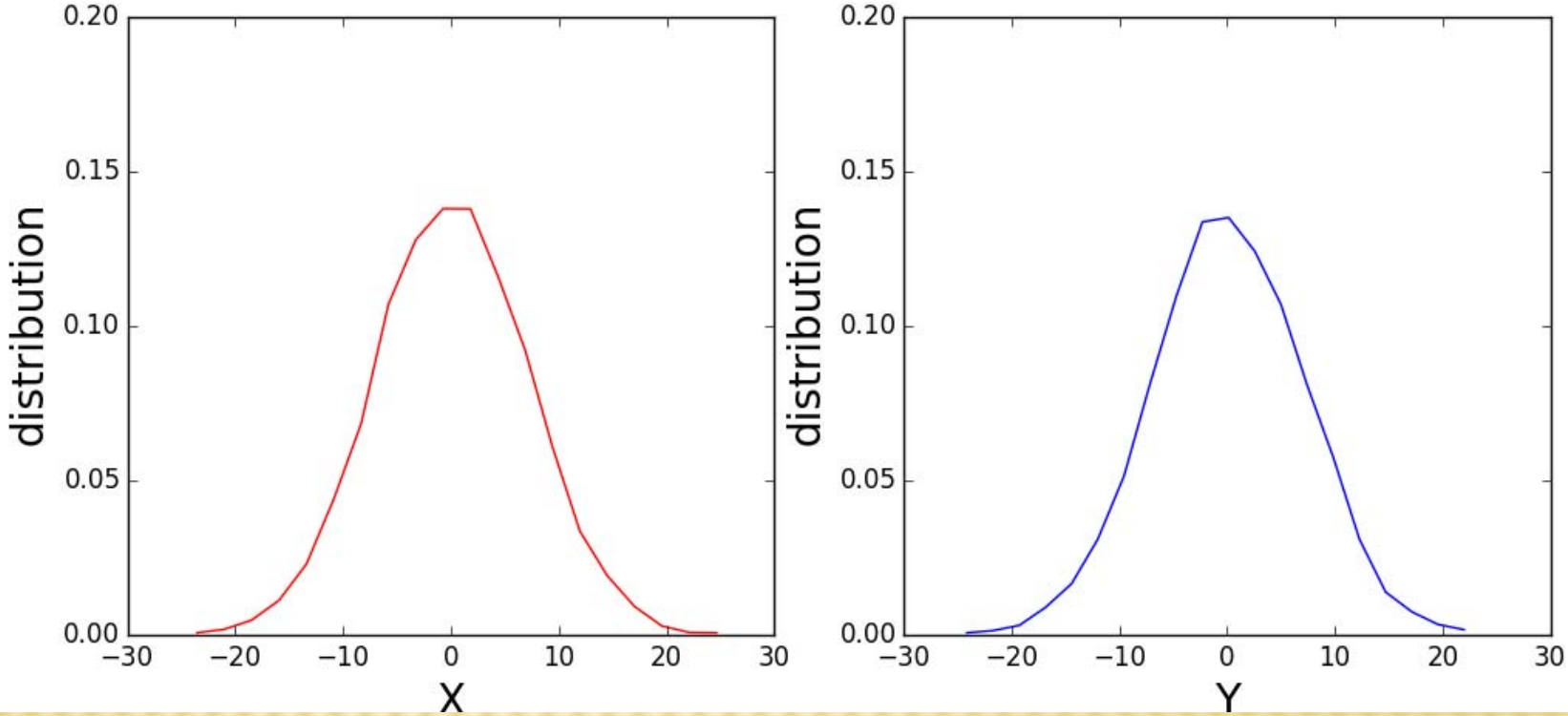
2）醉汉与电线杆均方距离随N的变化关系。



Exam-3-1-5.py

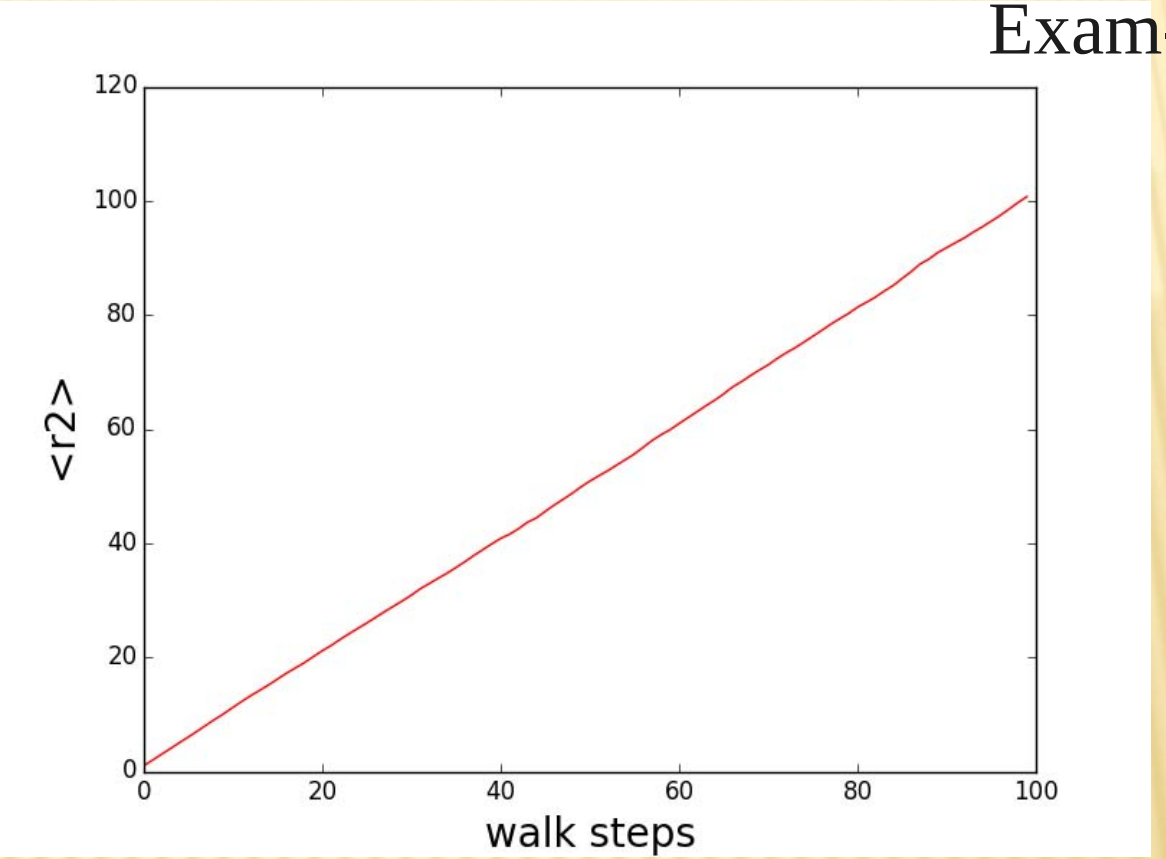


Exam-3-1-6.py



高斯分布！

Exam-3-1-7.py



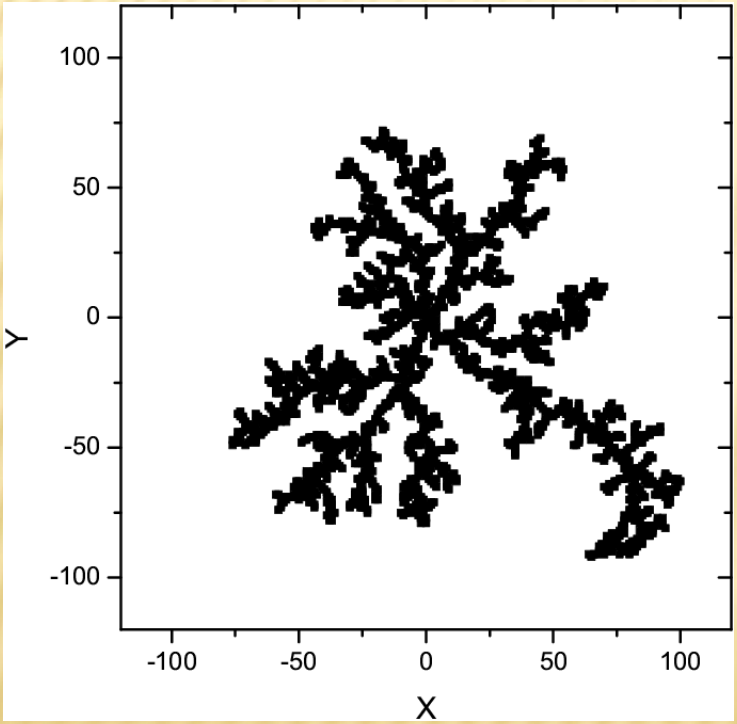
$$< r^2 >= \alpha * \text{step}$$

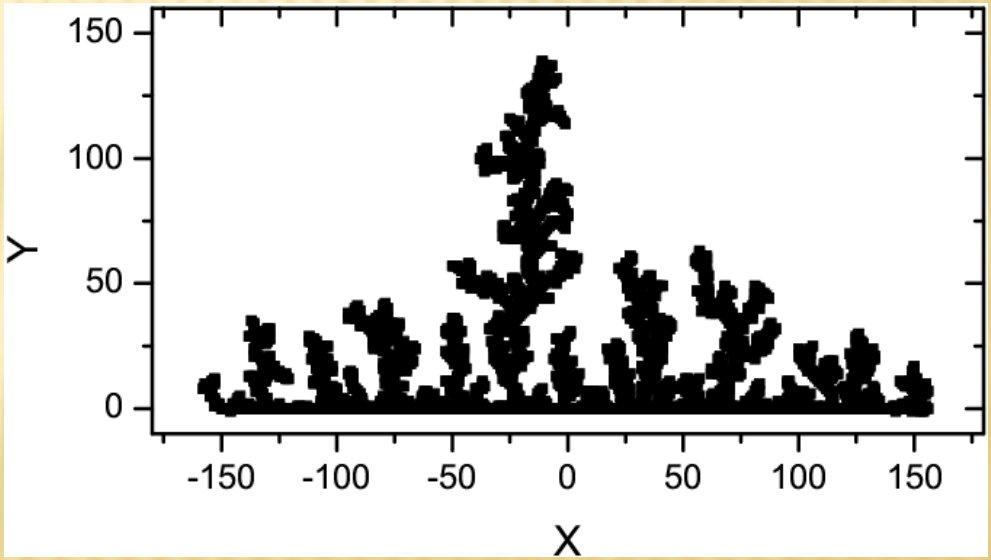
$$< r^2 >= \frac{6k_B T}{\xi} = 6Dt$$

Diffusion limited aggregation(DLA):

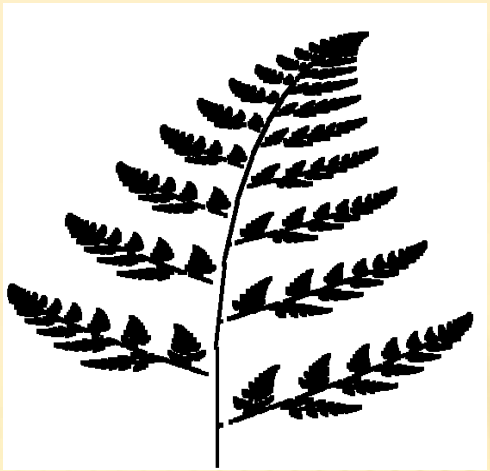
Diffusion limited aggregation(DLA):

Exam-3-1-8.py





初值：x0 = 0.5; y0 = 0.0



$$x^{(k+1)} = 0.5$$
$$y^{(k+1)} = 0.27*y^{(k)}$$

0.0 < rand < 0.02

$$x^{(k+1)} = -0.139x^{(k)} + 0.263*y^{(k)} + 0.57$$
$$y^{(k+1)} = 0.246*x^{(k)} + 0.224*y^{(k)} - 0.036$$

0.02 < rand < 0.17

$$x^{(k+1)} = 0.17x^{(k)} - 0.215*y^{(k)} + 0.408$$
$$y^{(k+1)} = 0.222*x^{(k)} + 0.176*y^{(k)} + 0.0893$$

0.17 < rand < 0.3

$$x^{(k+1)} = 0.781x^{(k)} - 0.034*y^{(k)} + 0.1075$$
$$y^{(k+1)} = -0.032*x^{(k)} + 0.739*y^{(k)} + 0.27$$

0.3 < rand < 1.0

Exam-3-1-9.py

初值：x0 = 0.5; y0 = 0.0

$$x^{(k+1)} = 0.05*x^{(k)}$$
$$y^{(k+1)} = 0.6*y^{(k)}$$

0.0 < rand <= 0.1

$$x^{(k+1)} = 0.05*x^{(k)}$$
$$y^{(k+1)} = -0.5*y^{(k)} + 1.0$$

0.1 < rand <= 0.2

$$x^{(k+1)} = 0.46x^{(k)} - 0.32*y^{(k)}$$
$$y^{(k+1)} = 0.39*x^{(k)} + 0.38*y^{(k)} + 0.6$$

0.2 < rand <= 0.4

$$x^{(k+1)} = 0.47x^{(k)} - 0.15*y^{(k)}$$
$$y^{(k+1)} = 0.17*x^{(k)} + 0.42*y^{(k)} + 1.1$$

0.4 < rand <= 0.6

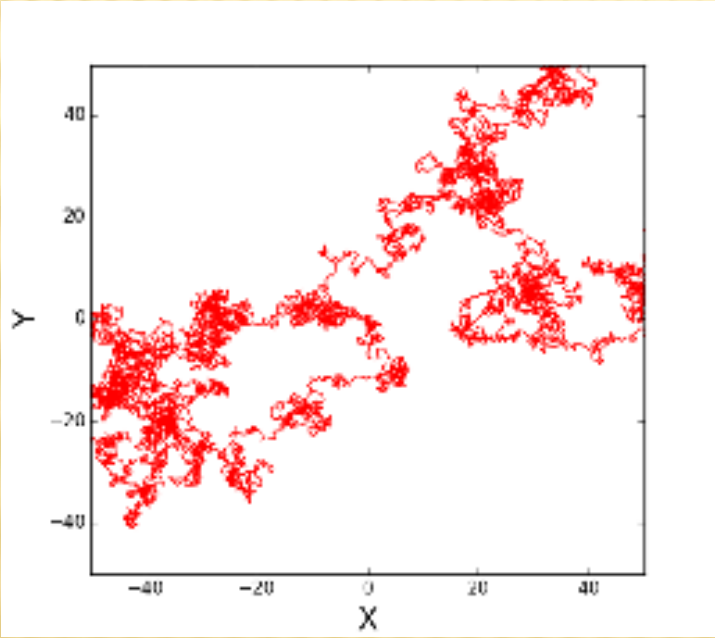
$$x^{(k+1)} = 0.43x^{(k)} + 0.28*y^{(k)}$$
$$y^{(k+1)} = -0.25*x^{(k)} + 0.45*y^{(k)} + 1.0$$

0.6 < rand <= 1.0

Exam-3-1-10.py

课 堂 练 习

醉汉模拟：
1）画成轨迹
2）计算x和y的分布
3）计算均方位移和步长的关系




```
import numpy as np
import matplotlib.pylab as pl

np.random.seed(12345) #赋初值
np.random.random()
#返回一个 [0,1) 之间的随机数

np.random.random(100)
#返回100个 [0,1) 之间的随机数
```

课 堂 练 习 2

初值：x0 = 0.5; y0 = 0.0

$x^{(k+1)} = 0.05*x^{(k)}$
$y^{(k+1)} = 0.6*y^{(k)}$

0.0 < rand <= 0.1

$x^{(k+1)} = 0.05*x^{(k)}$
$y^{(k+1)} = -0.5*y^{(k)} + 1.0$

0.1 < rand <= 0.2

$x^{(k+1)} = 0.46x^{(k)} - 0.32*y^{(k)}$
$y^{(k+1)} = 0.39*x^{(k)} + 0.38*y^{(k)} + 0.6$

0.2 < rand <= 0.4

$x^{(k+1)} = 0.47x^{(k)} - 0.15*y^{(k)}$
$y^{(k+1)} = 0.17*x^{(k)} + 0.42*y^{(k)} + 1.1$

0.4 < rand <= 0.6

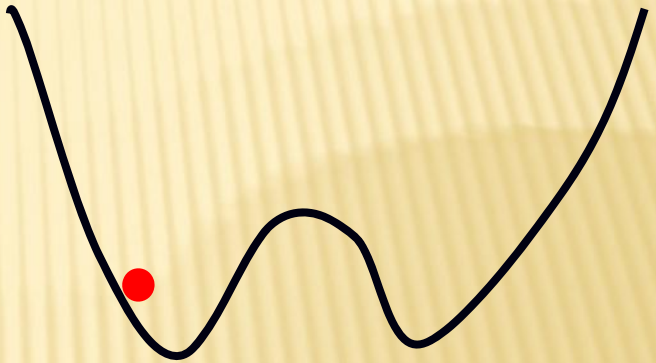
$x^{(k+1)} = 0.43x^{(k)} + 0.28*y^{(k)}$
$y^{(k+1)} = -0.25*x^{(k)} + 0.45*y^{(k)} + 1.0$

0.6 < rand <= 1.0

Exam-3-1-10.py

Langevin Equation: Simulating Brownian motions

$$V(\mathrm{x}) = (\mathrm{x}-1)^2(\mathrm{x}+2)^2$$



$$\mathrm{m}\frac{d^2x}{dt^2} = \mathrm{F}(\mathrm{x}) - \xi\frac{dx}{dt} + F_R$$

小雷诺数情形： $\xi\frac{dx}{dt} = \mathrm{F}(\mathrm{x}) + F_R$

$$\mathrm{F}(\mathrm{x}) = -\frac{\partial}{\partial x}V(\mathrm{x}) = -2(\mathrm{x}-1)(\mathrm{x}+2)(2\,\mathrm{x}+1)$$

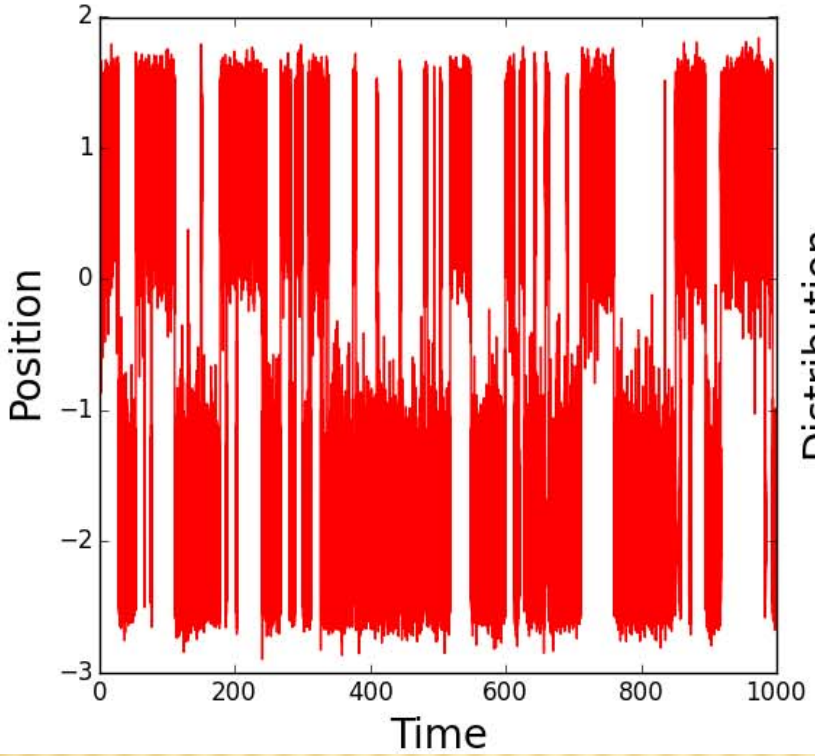
$$\frac{\mathrm{x}(\mathrm{t}+\Delta\mathrm{t})-\mathrm{x}(\mathrm{t})}{\Delta\,\mathrm{t}} = \frac{1}{\xi}F(\mathrm{x}) + \sqrt{\frac{2k_BT}{\xi\cdot\Delta t}}\cdot\Gamma_{normal}$$

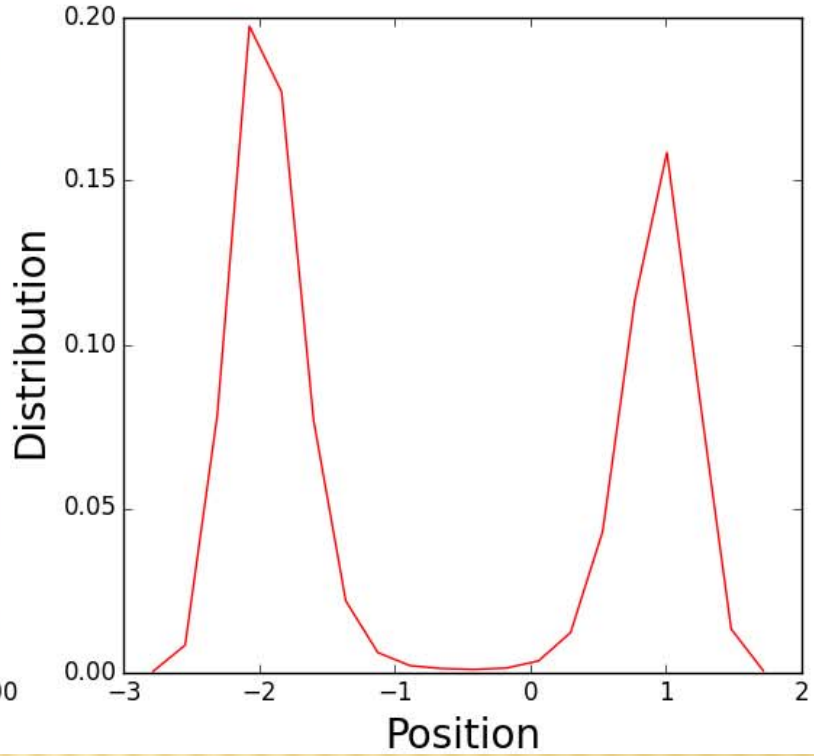
标准正态分布随机数

高斯白噪声： $< F_R(\mathrm{t}) >= 0; \quad < F_R(\mathrm{t})\cdot F_R(\mathrm{t}') >= 2\,\mathrm{k_B T}\,\xi\delta(\mathrm{t}-\mathrm{t}')$

可以验证：当V(x) = 0时， $< \mathrm{x}(\mathrm{t}) >= 0; \quad < x(\mathrm{t})\cdot x(\mathrm{t}) >= 2Dt$

Exam-3-1-11.py


Position
Time
2
1
0
−1
−2
−3
0
200
400
600
800
1000


Distribution
Position
0.20
0.15
0.10
0.05
0.00
−3
−2
−1
0
1
2

动画：Exam-3-1-12.py