第一章：Monte Carlo 方法概述

本章主要概述 Monte Carlo 的一些基础知识，另外包括一个最简单的用 Monte Carlo 方法计算数值积分的例子。

一、Monte Carlo 历史渊源

Monte Carlo 方法的实质是通过大量随机试验，利用概率论解决问题的一种数值方法，基本思想是基于概率和体积间的相似性。它和 Simulation 有细微区别。单独的 Simulation 只是模拟一些随机的运动，其结果是不确定的；Monte Carlo 在计算的中间过程中出现的数是随机的，但是它要解决的问题的结果却是确定的。

历史上有记载的 Monte Carlo 试验始于十八世纪末期（约 1777 年），当时布丰（Buffon）为了计算圆周率，设计了一个“投针试验”。（后文会给出一个更加简单的计算圆周率的例子）。虽然方法已经存在了 200 多年，此方法命名为 Monte Carlo 则是在二十世纪四十年，美国原子弹计划的一个子项目需要使用 Monte Carlo 方法模拟中子对某种特殊材料的穿透作用。出于保密缘故，每个项目都要一个代号，传闻命名代号时，项目负责人之一 von Neumann 灵犀一点选择摩洛哥著名赌城蒙特卡洛作为该项目名称，自此这种方法也就被命名为 Monte Carlo 方法广为流传。

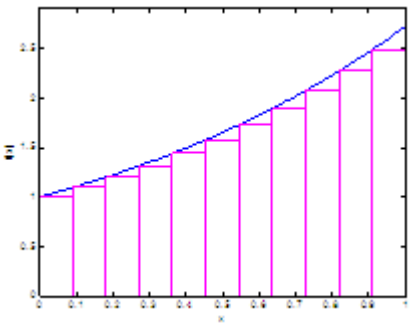
十一、Monte Carlo 方法适用用途

（一）数值积分

$$\int_{x0}^{x1} f(x)dx$$

计算一个定积分，如 ，如果我们能够得到 f(x)的原函数 F(x)，那么直接由表达式: F(x1)-F(x0)可以得到该定积分的值。但是，很多情况下，由于 f(x)太复杂，我们无法计算得到原函数 F(x)的显示解，这时我们就只能用数值积分的办法。如下是一个简单的数值积分的例子。

数值积分简单示例



如图，数值积分的基本原理是在自变量 x 的区间上取多个离散的点，用单个点的值来代替该小段上函数 f(x)值。

常规的数值积分方法是在分段之后，将所有的柱子（粉红色方块）的面积全部加起来，用这个面积来近似函数 f(x)（蓝色曲线）与 x 轴围成的面积。这样做当然是不精确的，但是随着分段数量增加，误差将减小，近似面积将逐渐逼近真实的面积。

Monte Carlo 数值积分方法和上述类似。差别在于，Monte Carlo 方法中，我们不需要将所有方柱的面积相加，而只需要随机地抽取一些函数值，将他们的面积累加后计算平均值就够了。通过相关数学知识可以证明，随着抽取点增加，近似面积也将逼近真实面积。

在金融产品定价中，我们接触到的大多数求基于某个随机变量的函数的期望值。考虑一个欧式期权，假定我们已经知道在期权行权日的股票服从某种分布（理论模型中一般是正态分布），那么用期权收益在这种分布上做积分求期望即可。

（五）随机最优化

Monte Carlo 在随机最优化中的应用包括：模拟退火(Simulated Annealing)、进化策略(Evolution strategy)等等。一个最简单的例子是，已知某函数，我们要求此函数的最大值，那么我们可以不断地在该函数定义域上随机取点，然后用得到的最大的点作为此函数的最大值。这个例子实质也是随机数值积分，它等价于求此函数的无穷阶范数（∞-Norm）在定义域上的积分。

由于在金融产品定价中，这部分内容用的相对较不常见，所以此课程就不介绍随机最优化方法了。

十二、Monte Carlo 形式与一般步骤

（一）积分形式

做 Monte Carlo 时，求解积分的一般形式是：

$$\int_{x0}^{x1} f(x)\psi(x)dx$$

X 为自变量，它应该是随机的，定义域为(x0, x1)，f(x)为被积函数，ψ(x)是 x 的概率密度。在计算欧式期权例子中，x 为期权到期日股票价格，由于我们计算期权价格的时候该期权还没有到期，所以此时 x 是不确定的（是一随机变量），我们按照相应的理论，假设 x 的概率密度为ψ(x)、最高可能股价为 x1(可以是正无穷）、最低可能股价为 x0（可以是 0），另外，期权收益是到期日股票价格 x 和期权行权价格的函数，我们用 f(x)来表示期权收益。

（二）一般步骤

我将 Monte Carlo 分为三加一个步骤：

1．依据概率分布ψ(x)不断生成随机数 x, 并计算 f(x)

由于随机数性质，每次生成的 x 的值都是不确定的，为区分起见，我们可以给生成的 x 赋予下标。如 x_i 表示生成的第 i 个 x。生成了多少个 x，就可以计算出多少个 f(x)的值

2．将这些 f(x)的值累加，并求平均值

例如我们共生成了 N 个 x，这个步骤用数学式子表达就是

$$\frac{\sum_{i=1}^{N} f(x_i)}{N}$$

3．到达停止条件后退出

常用的停止条件有两种，一种是设定最多生成 N 个 x，数量达到后即退出，另一种是检测计算结果与真实结果之间的误差，当这一误差小到某个范围之内时退出。

有趣的类比：积分表达式中的积分符合类比为上式中累加符号，dx 类比为 1/N（数学知识告诉我们积分实质是极限意义下的累加；f(x)还是它自己，积分中的ψ(x)可类比为依据ψ(x)生成随机数

4．误差分析

Monte Carlo 方法得到的结果是随机变量，因此，在给出点估计后，还需要给出此估计值的波动程度及区间估计。严格的误差分析首先要从证明收敛性出发，再计算理论方差，最后用样本方差来替代理论方差。在本课程中我们假定此方法收敛，同时得到的结果服从正态分布，因此可以直接用样本方差作区间估计。详细过程在例子中解释。

这个步骤的理论意义很重要，但在实际应用中，它的重要性有所淡化，倘若你的老板不太懂这些知识，你报告计算结果时可以只告诉他点估计即可。

注意，前两大步骤还可以继续细分，例如某些教科书上的五大步骤就是将此处的前两步细分成四步。

十三、最简单的例子

举个例子:

计算从e^x函数从 0 到 2 的定积分值$\int_0^2 e^x dx$。

数学方法：我们已知e^x的原函数是e^x，那么定积分值就是：e^2-e^0=6.38905609893065 。

计算这个数值可以在 Matlab 中输入代码：

```
exp(2)-exp(0)
```

上面得到的值是此不定积分的真实值。

常规数值积分：在$x\in(0,2)$区间内取 N 个点，计算各个点上的函数值，然后用函数值乘以每个区间宽度，最后相加。Matlab 代码：

```
N=100;x=linspace(0,2,N);sum(exp(x).*2/N)
```

试着调大 N 的值，你会发现，最后的结果将更接近真实值。

Monte Carlo 数值积分法：在$x\in(0,2)$内随机取 N 个点，计算各个点上的函数值，最后求这些函数值的平均值再乘以 2（为何要乘以 2 在后面小节详细讲）。看 Matlab 代码：

```
N=100;x=unifrnd(0,2,N,1);mean(2*exp(x))
```

同样的，通过增大 N，这种方法得到的结果也将越来越接近真实值。

解释

这个例子要求的积分形式是：$\int_0^2 e^x dx$，还不完全是$\int_{x0}^{x1} f(x)\psi(x)dx$形式，我们先做变换，$\int_0^2 (2e^x)(\frac{1}{2})dx$，这里$2e^x$是 f(x)；1/2 是ψ(x)，它表示，在取值范围(0,2)区间内，x 服从均匀分布。

前一例子共三条语句，逐句解释如下：

```
N=100;
```

设定停止条件，共做 N 次 Monte Carlo 模拟。

```
x=unifrnd(0,2,N,1);
```

按照(0,2)区间均匀分布概率密度对 x 随机抽样，共抽取 N 个 x_i。此句相当于第一个步骤中的前半部分。

```
mean(2*exp(x))
```

2*exp(x)作用是对每个 x_i 计算 $f(x_i)$的值,共可得到 N 个值，这个相当于第一个步骤后半部分；Mean()函数的作用是将所有的 $f(x_i)$加起来取平均值，相当于第二个步骤。

这段代码中的停止条件隐含于 N 值设定中，它一次性生成 N 个 x 值，完成此次计算后整个程序就结束了。

十四、Monte Carlo 方法的优点

对比前面常规数值积分和 Monte Carlo 数值积分代码，同样数量的 N 值——也就意味这几乎相同的计算量——常规数值积分结果的精确度要高于 Monte Carlo 数值积分的结果。那么，我们为何还需要用 Monte Carlo 来算数值积分呢？

答案的关键在于，常规数值积分的精度直接取决于每个维度上取点数量，维度增加了，但是每个维度上要取的点却不能减少。在多重积分中，随着被积函数维度增加，需要计算的函数值数量以指数速度递增。例如在一重积分 $\int_{x0}^{x1} f(x)\psi(x)dx$ 中，只要沿着 x 轴取 N 个点；要达到相同大小的精确度，在 s 重积分 $\iint\cdots\int f(x_1,x_2,\cdots,x_s)\psi(x_1,x_2,\cdots,x_s)d(x_1,x_2,\cdots,x_s)$ 中，仍然需要在每个维度上取 N 个点，s 个纬度的坐标相组合，共需要计算 N^s 个坐标对应的 f()函数值。取点越多，会占用计算机大量内存，也需要更长运算时间，最终导致这种计算方法不可行！

Monte Carlo 方法却不同，不管是积分有多少重，取 N 个点计算的结果精确度都差不多。因此，即使在一重积分的情形下，Monte Carlo 方法的效率比不过常规数值积分，但随着积分维度增加，常规数值积分的速度呈指数下降，Monte Carlo 方法的效率却基本不变。经验表明，当积分重数达到 4 重积分甚至更高时，Monte Carlo 方法将远远优于常规数值积分方法。

现在回到金融产品定价，欧式期权理论定价公式只需要一重积分，此时 Monte Carlo 方法的效果不明显，但是如果我们考虑一个亚式期权：期限为 1 年期，期权价格基于此 1 年内每天某个时点时的价格，全年共 252 个交易日，这样此亚式期权理论定价公式是一个 252 重积分。常规的数值积分方法，需要取 N^{252} 个点，这个数有多大，你自己去计算一下就知道了（注意：N 取值要远远大于 2），常规数值积分方法不可行，只能用 Monte Carlo。

综上，如果计算高维度多重积分，如路径依赖的 exotic options（奇异期权）等金融产品定价，我们一般用的方法都是 Monte Carlo。

十五、Monte Carlo 方法原理(选读)

Monte Carlo 方法计算的结果收敛的理论依据来自于大数定律，且结果渐进地（Asymptotically）服从正态分布的理论依据是中心极限定理。

以上两个属性都是渐进性质，要进行很多次抽样，此属性才会比较好地显示出来，如果 Monte Carlo 计算结果的某些高阶距存在，即使抽样数量不太多，这些渐进属性也可以很快地达到。

这些原理在理论上意义重大，但由于我们一般遇上的 Monte Carlo 问题都是收敛的、结果也都是渐进正态分布，所以工作中使用时可以不加考虑。

详细推导见相关书籍。

第二章：随机数的生成

本章第一节会简要复习随机变量的一些概念，但学习本章最好要有一定的数学基础。第二节主要介绍如何生成一维概率分布的随机数，第三节介绍如何生成高维分布的随机数。最后略提及伪随机数问题的应对策略。

由前文可知，Monte Carlo 积分解决的问题形如 $\int_{x0}^{x1} f(x)\psi(x)dx$，f(x)值只需由 x 值决定，因此此处最重要的就是如何生成服从 ψ(x)概率分布的随机数。可以说，正确生成随机数，Monte Carlo 方法就做完了一半。

一、随机变量基本概念

（一）随机变量

现实世界中有很多可以用数字来衡量的事物，站在当前时间点来看，它们在未来时刻的值是不确定的。例如，我们掷一骰子，在它停稳前，我们不可能知道掷出多少点（传说中的赌王除外，哈哈）；例如某只股票在明天的股价，没有人能准确知晓第二天股票的价格（不然他就发惨了！）。但是，我们却可以描述这些事物未来各种值的可能性。

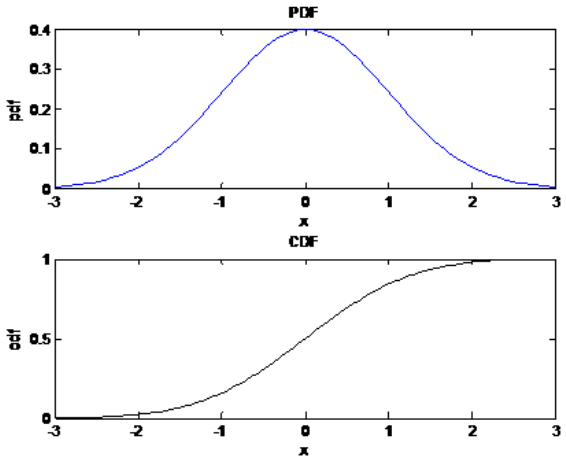
（二）离散型随机变量

离散型随机变量最重要的是分布律，即每个取值的概率是多少。例如掷骰子，我们认为扔出任何一个点的概率都是 1/6。那么掷骰子得到的点数的分布律如下表：

骰子点数	1	2	3	4	5	6
概率	1/6	1/6	1/6	1/6	1/6	1/6

（三）连续性随机变量

连续型随机变量有两个重要的概念。概率密度函数(PDF)和累积概率分布函数(CDF)，具体定义见数学书籍。



PDF 函数本身不是概率，只有对 x 的某段区间中的 PDF 积分得到的数值才有概率的含义。CDF 是概率的意思，点 x 上 CDF 的值表示该随机变量可能取值小于 x 的概率的大小。

如图是正态分布的 PDF 和 CDF

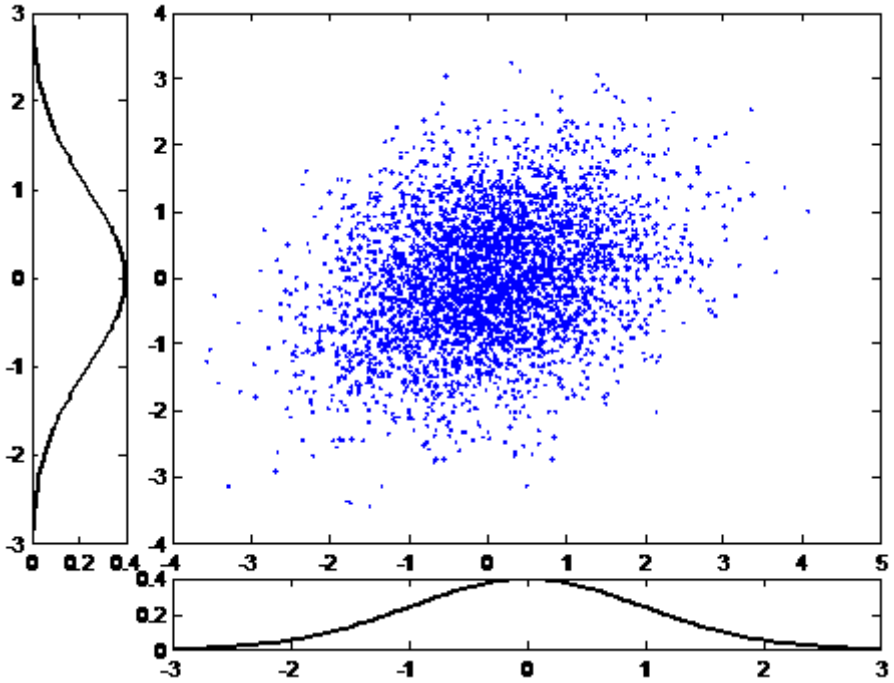
（四）多元分布

在这个课程里面，我不将多维的随机变量拆分成多个一维的随机变量来表述，而是将各个维度组合成一个随机变量向量。多元随机变量分布需要掌握联合分布、边缘分布和条件分布。

联合分布和单变量 PDF 类似，如果将其对随机变量某个取值范围做积分就可以得到随机变量最终取值落在该区间内的概率。

边缘分布是只考虑多维随机变量中的某一维，其他维度不考虑情况下的 PDF

条件分布是固定随机变量在其他维度的取值，再考虑剩余那个维度上的 PDF。



如图是一个二维正态分布(两个维度间相关系数为 0.3)的示意图。两个小图分别是第一和第二维度的边缘分布 PDF 图（都是标准正态分布 PDF）。右上角的大图是依据此二维正态联合分布生成的随机数。从随机数的疏密程度可以看出联合分布 PDF 函数在该区域的大小。

研究这些分布不是我们的目的，只是达到目的的手段。我们所需要做的事情是生成符合各种分布的随机数。由分布的不同类型——连续型和离散型，常规型（Matlab 中有内置函数）和特殊型（Matlab 中无内置函数），一维分布和多维分布——我们接下来就各种情况分别讲述如何在 Matlab 中生成各种各样的随机数。

十六、一维随机数

两个注意事项：

1．生成随机数有两种选择，可以每次只生成一个随机数，直接用此数计算 f(x),然后循环重复此过程，最后求平均值；另一种方法是每次生成全部循环所需的随机数，利用 Matlab 矩阵运算语法计算 f(x)，不需要写循环，直接即可求平均值。前一种方法代码简单，但速度慢；后一种方法代码相对更难写。此处一定要掌握如何生成随机数组成的向量和矩阵（单个随机数就是一个 1*1 的矩阵），在后面章节的例子里面一般会有两个版本的代码分别讲述这两种生成方法。

2．生成了一维的随机数后，可以用 hist()函数查看这些数服从的大致分布情况。

（一）Matlab 内部函数

a. 基本随机数

Matlab 中有两个最基本生成随机数的函数。

1．rand()

生成（0,1）区间上均匀分布的随机变量。基本语法：

```
rand([M,N,P ...])
```

生成排列成 M*N*P... 多维向量的随机数。如果只写 M，则生成 M*M 矩阵；如果参数为[M,N]可以省略掉方括号。一些例子：

```
rand(5,1) %生成 5 个随机数排列的列向量，一般用这种格式
rand(5) %生成 5 行 5 列的随机数矩阵
rand([5,4]) %生成一个 5 行 4 列的随机数矩阵
```

生成的随机数大致的分布。

```
x=rand(100000,1);
hist(x,30);
```

由此可以看到生成的随机数很符合均匀分布。(视频教程会略提及 hist()函数的作用)

2．randn()

生成服从标准正态分布（均值为 0，方差为 1）的随机数。基本语法和 rand()类似。

```
randn([M,N,P ...])
```

生成排列成 M*N*P... 多维向量的随机数。如果只写 M，则生成 M*M 矩阵；如果参数为[M,N]可以省略掉方括号。一些例子：

```
randn(5,1) %生成 5 个随机数排列的列向量，一般用这种格式
randn(5) %生成 5 行 5 列的随机数矩阵
randn([5,4]) %生成一个 5 行 4 列的随机数矩阵
```

生成的随机数大致的分布。

```
x=randn(100000,1);
hist(x,50);
```

由图可以看到生成的随机数很符合标准正态分布。

b. 连续型分布随机数

如果你安装了统计工具箱（Statistic Toolbox)，除了这两种基本分布外，还可以用 Matlab 内部函数生成符合下面这些分布的随机数。

3．unifrnd()

和 rand()类似，这个函数生成某个区间内均匀分布的随机数。基本语法

```
unifrnd(a,b,[M,N,P,...])
```

生成的随机数区间在(a,b)内，排列成 M*N*P... 多维向量。如果只写 M，则生成 M*M 矩阵；如果参数为[M,N]可以省略掉方括号。一些例子：

```
unifrnd(-2,3,5,1) %生成 5 个随机数排列的列向量，一般用这种格式
unifrnd(-2,3,5) %生成 5 行 5 列的随机数矩阵
unifrnd(-2,3,[5,4]) %生成一个 5 行 4 列的随机数矩阵
%注：上述语句生成的随机数都在(-2,3)区间内.
```

生成的随机数大致的分布。

```
x=unifrnd(-2,3,100000,1);
hist(x,50);
```

由图可以看到生成的随机数很符合区间(-2,3)上面的均匀分布。

4．normrnd()

和 randn()类似，此函数生成指定均值、标准差的正态分布的随机数。基本语法

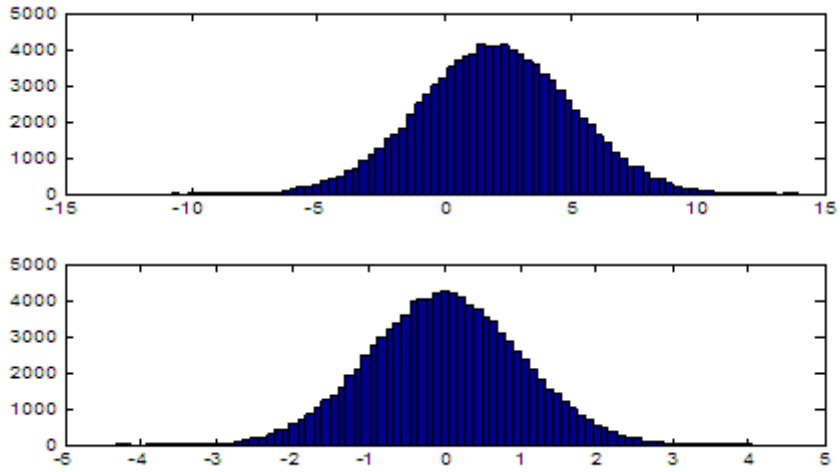
```
normrnd(mu,sigma,[M,N,P,...])
```

生成的随机数服从均值为 mu，标准差为 sigma（注意标准差是正数）正态分布，这些随机数排列成 M*N*P... 多维向量。如果只写 M，则生成 M*M 矩阵；如果参数为[M,N]可以省略掉方括号。一些例子：

```
normrnd(2,3,5,1) %生成 5 个随机数排列的列向量，一般用这种格式
normrnd(2,3,5) %生成 5 行 5 列的随机数矩阵
normrnd(2,3,[5,4]) %生成一个 5 行 4 列的随机数矩阵
%注：上述语句生成的随机数所服从的正态分布都是均值为 2，标准差为 3.
```

生成的随机数大致的分布。

```
x=normrnd(2,3,100000,1);
hist(x,50);
```



如图，上半部分是由上一行语句生成的均值为 2，标准差为 3 的 10 万个随机数的大致分布，下半部分是用小节“randn()”中最后那段语句生成 10 万个标准正态分布随机数的大致分布。

注意到上半个图像的对称轴向正方向偏移（准确说移动到 x=2 处），这是由于均值为 2 的结果。

而且，由于标准差是 3，比标准正态分布的标准差（1）要高，所以上半部分图形更胖(注意 x 轴刻度的不同)。

5．chi2rnd()

此函数生成服从卡方（Chi-square)分布的随机数。卡方分布只有一个参数：自由度 v。基本语法

```
chi2rnd(v,[M,N,P,...])
```

生成的随机数服从自由度为 v 的卡方分布，这些随机数排列成 M*N*P... 多维向量。如果只写 M，则生成 M*M 矩阵；如果参数为[M,N]可以省略掉方括号。一些例子：

```
chi2rnd(5,5,1) %生成 5 个随机数排列的列向量，一般用这种格式
chi2rnd(5,5) %生成 5 行 5 列的随机数矩阵
chi2rnd(5,[5,4]) %生成一个 5 行 4 列的随机数矩阵
%注：上述语句生成的随机数所服从的卡方分布的自由度都是 5
```

生成的随机数大致的分布。

```
x=chi2rnd(5,100000,1);
hist(x,50);
```

6．frnd()

此函数生成服从 F 分布的随机数。F 分布有 2 个参数：v1, v2。基本语法

```
frnd(v1,v2,[M,N,P,...])
```

生成的随机数服从参数为(v1,v2)的卡方分布，这些随机数排列成 M*N*P... 多维向量。如果只写 M，则生成 M*M 矩阵；如果参数为[M,N]可以省略掉方括号。一些例子：

```
frnd(3,5,5,1) %生成 5 个随机数排列的列向量，一般用这种格式
frnd(3,5,5) %生成 5 行 5 列的随机数矩阵
frnd(3,5,[5,4]) %生成一个 5 行 4 列的随机数矩阵
%注：上述语句生成的随机数所服从的参数为(v1=3,v2=5)的 F 分布
```

生成的随机数大致的分布。

```
x=frnd(3,5,100000,1);
hist(x,50);
```

从结果可以看出来， F 分布集中在 x 正半轴的左侧，但是它在极端值处也很可能有一些取值。

7．trnd()

此函数生成服从 t(Student's t Distribution，这里 Student 不是学生的意思，而是 Cosset.W.S. 的笔名)分布的随机数。t 分布有 1 个参数：自由度 v。基本语法

```
trnd(v,[M,N,P,...])
```

生成的随机数服从参数为 v 的 t 分布，这些随机数排列成 M*N*P... 多维向量。如果只写 M，则生成 M*M 矩阵；如果参数为[M,N]可以省略掉方括号。一些例子：

```
trnd(7,5,1) %生成 5 个随机数排列的列向量，一般用这种格式
trnd(7,5) %生成 5 行 5 列的随机数矩阵
trnd(7,[5,4]) %生成一个 5 行 4 列的随机数矩阵
%注：上述语句生成的随机数所服从的参数为(v=7)的 t 分布
```

生成的随机数大致的分布。

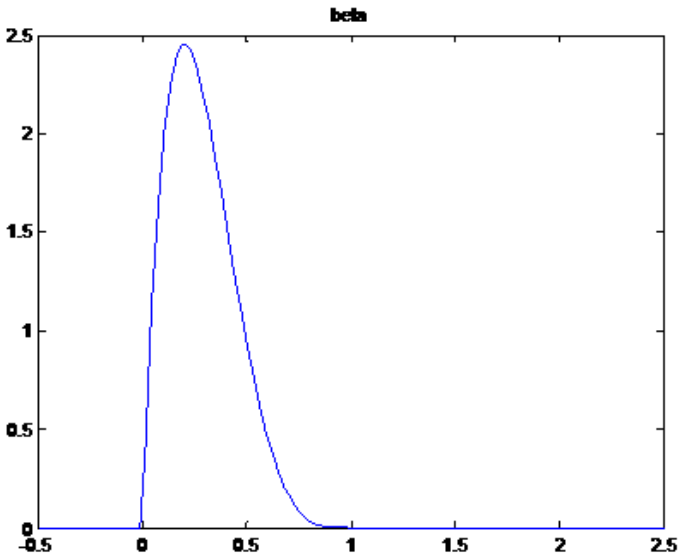
```
x=trnd(7,100000,1);
hist(x,50);
```

可以发现 t 分布比标准正太分布要“瘦”，不过随着自由度 v 的增大，t 分布会逐渐变胖，当自由度为正无穷时，它就变成标准正态分布了。

接下来的分布相对没有这么常用，同时这些函数的语法和前面函数语法相同，所以写得就简略一些——在视频中也不会讲述，你只需按照前面那几个分布的语法套用即可，应该不会有任何困难——时间足够的话这是一个不错的练习机会。

8．betarnd()

此函数生成服从 Beta 分布的随机数。Beta 分布有两个参数分别是 A 和 B。下图是 A=2,B=5 的 beta 分布的 PDF 图形。

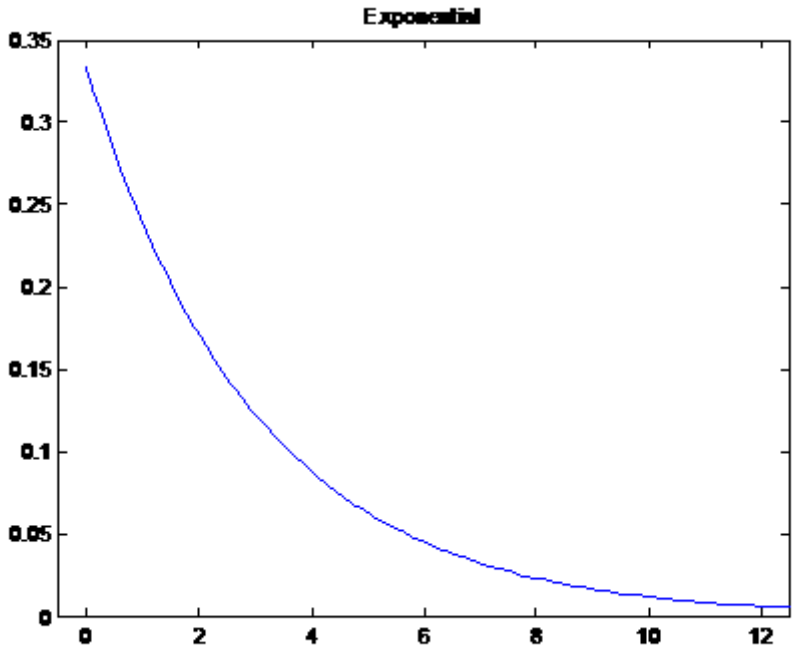


生成 beta 分布随机数的语法是：

```
betarnd(A,B,[M,N,P,...])
```

9．exprnd()

此函数生成服从指数分布的随机数。指数分布只有一个参数: mu，下图是 mu=3 时指数分布的 PDF 图形

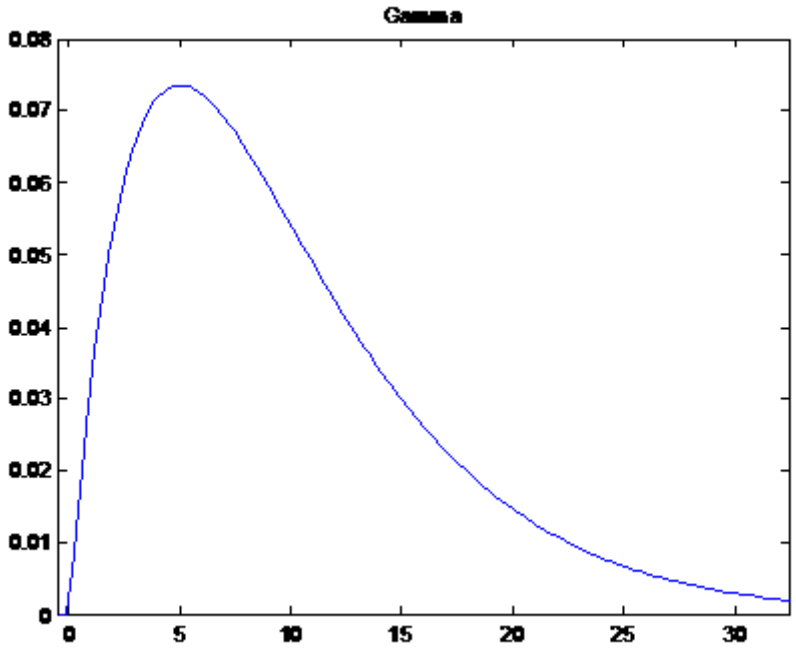


生成指数分布随机数的语法是：

```
betarnd(mu,[M,N,P,...])
```

10．gamrnd()

生成服从 Gamma 分布的随机数。Gamma 分布有两个参数：A 和 B。下图是 A=2,B=5 Gamma 分布的 PDF 图形

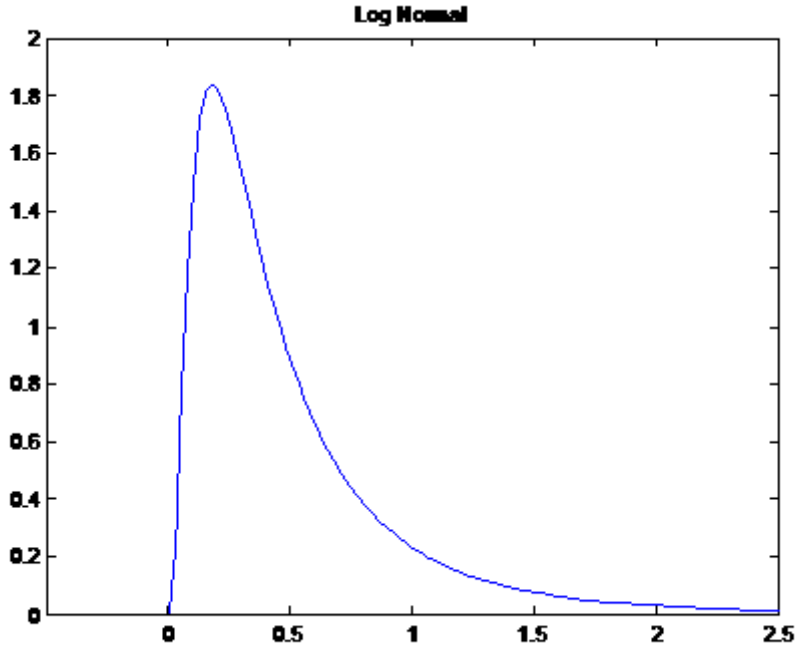


生成 Gamma 分布随机数的语法是：

```
gamrnd(A,B,[M,N,P,...])
```

11．lognrnd()

生成服从对数正态分布的随机数。其有两个参数：mu 和 sigma，服从这个这样的随机数取对数后就服从均值为 mu，标准差为 sigma 的正态分布。下图是 mu=-1, sigma=1/1.2 的对数正态分布的 PDF 图形。

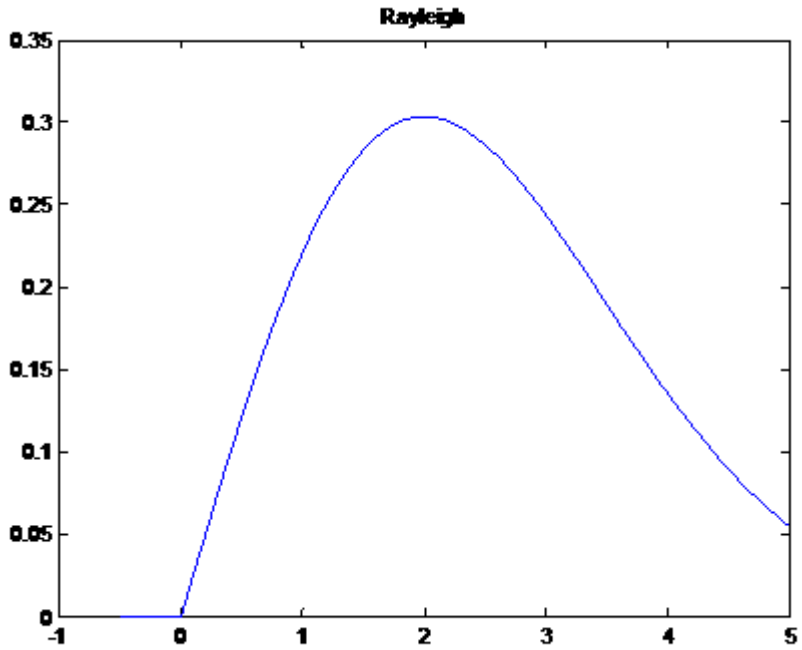


生成对数正态分布随机数的语法是：

```
lognrnd(mu,sigma,[M,N,P,...])
```

12．raylrnd()

生成服从瑞利（Rayleigh）分布的随机数。其分布有 1 个参数：B。下图是 B=2 的瑞利分布的 PDF 图形。

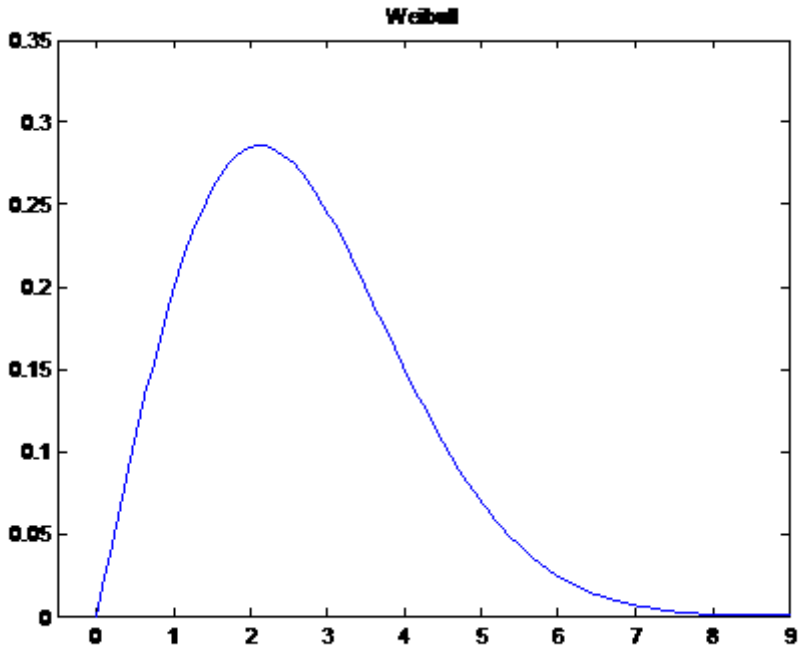


生成瑞利分布随机数的语法是：

```
raylrnd(B,[M,N,P,...])
```

13．wblrnd()

生成服从威布尔（Weibull）分布的随机数。其分布有 2 个参数：scale 参数 A 和 shape 参数 B。下图是 A=3，B=2 的 Weibull 分布的 PDF 图形。



生成 Weibull 分布随机数的语法是：

```
wblrnd(A,B,[M,N,P,...])
```

还有非中心卡方分布(ncx2rnd)，非中心 F 分布(ncfrnd)，非中心 t 分布（nctrnd)， 括号中是生成服从这些分布的函数，具体用法用：

```
help 函数名
```

查找。

c. 离散型分布随机数

离散分布的随机数可能的取值是离散的，一般是整数。

14．unidrnd()

此函数生成服从离散均匀分布的随机数。Unifrnd 是在某个区间内均匀选取实数（可为小数或整数），Unidrnd 是均匀选取整数随机数。离散均匀分布随机数有 1 个参数：n, 表示从{1, 2, 3, ... N}这 n 个整数中以相同的概率抽样。基本语法：

```
unidrnd(n,[M,N,P,...])
```

这些随机数排列成 M*N*P... 多维向量。如果只写 M，则生成 M*M 矩阵；如果参数为[M,N]可以省略掉方括号。一些例子：

```
unidrnd(5,5,1) %生成 5 个随机数排列的列向量，一般用这种格式
unidrnd(5,5) %生成 5 行 5 列的随机数矩阵
unidrnd(5,[5,4]) %生成一个 5 行 4 列的随机数矩阵
%注：上述语句生成的随机数所服从的参数为(10,0.3)的二项分布
```

生成的随机数大致的分布。

```
x=unidrnd(9,100000,1);
hist(x,9);
```

可见，每个整数的取值可能性基本相同。

15．binornd()

此函数生成服从二项分布的随机数。二项分布有 2 个参数：n,p。考虑一个打靶的例子，每枪命中率为 p，共射击 N 枪，那么一共击中的次数就服从参数为（N,p）的二项分布。注意 p 要小于等于 1 且非负，N 要为整数。基本语法：

```
binornd(n,p,[M,N,P,...])
```

生成的随机数服从参数为(N,p)的二项分布，这些随机数排列成 M*N*P... 多维向量。如果只写 M，则生成 M*M 矩阵；如果参数为[M,N]可以省略掉方括号。一些例子：

```
binornd(10,0.3,5,1) %生成 5 个随机数排列的列向量，一般用这种格式
binornd(10,0.3,5) %生成 5 行 5 列的随机数矩阵
binornd(10,0.3,[5,4]) %生成一个 5 行 4 列的随机数矩阵
%注：上述语句生成的随机数所服从的参数为(10,0.3)的二项分布
```

生成的随机数大致的分布。

```
x=binornd(10,0.45,100000,1);
hist(x,11);
```

我们可以将此直方图解释为，假设每枪射击命中率为 0.45，每论射击 10 次，共进行 10 万轮，这个图就表示这 10 万轮每轮命中成绩可能的一种情况。

16．geornd()

此函数生成服从几何分布的随机数。几何分布的参数只有一个：p。几何分布的现实意义可以解释为，打靶命中率为 p，不断地打靶，直到第一次命中目标时**没有击中次数之和**。注意 p 是概率，所以要小于等于 1 且非负。基本语法：

```
geornd(p,[M,N,P,...])
```

这些随机数排列成 M*N*P... 多维向量。如果只写 M，则生成 M*M 矩阵；如果参数为[M,N]可以省略掉方括号。一些例子：

```
geornd(0.4,5,1) %生成 5 个随机数排列的列向量，一般用这种格式
geornd(0.4,5) %生成 5 行 5 列的随机数矩阵
geornd(0.4,[5,4]) %生成一个 5 行 4 列的随机数矩阵
%注：上述语句生成的随机数所服从的参数为(0.4)的二项分布
```

生成的随机数大致的分布。

```
x=geornd(0.4,100000,1);
hist(x,50);
```

17. poissrnd()

此函数生成服从泊松(Poisson)分布的随机数。泊松分布的参数只有一个：lambda。此参数要大于零。基本语法：

```
geornd(p,[M,N,P,...])
```

这些随机数排列成 M*N*P... 多维向量。如果只写 M，则生成 M*M 矩阵；如果参数为[M,N]可以省略掉方括号。一些例子：

```
poissrnd(2,5,1) %生成 5 个随机数排列的列向量，一般用这种格式
poissrnd(2,5) %生成 5 行 5 列的随机数矩阵
poissrnd(2,[5,4]) %生成一个 5 行 4 列的随机数矩阵
%注：上述语句生成的随机数所服从的参数为(2)的泊松分布
```

生成的随机数大致的分布。

```
x=poissrnd(2,100000,1);
hist(x,50);
```

其他离散分布还有超几何分布(Hyper-geometric, 函数是 hygernd)等，详细见 Matlab 帮助文档。

（六）特殊连续分布

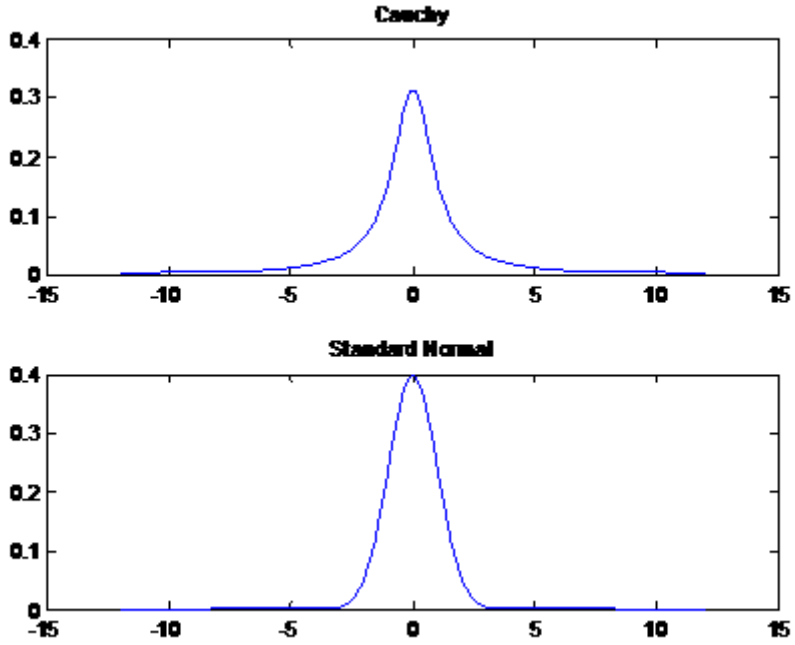
这里我将 Matlab 中没有对应函数的分布称为特殊分布。有多种方法可以用于生产服从这些分布的随机数。这里主要介绍两种最常见的。

1. 逆 CDF 函数法

如果我们已知某特定一维分布的 CDF 函数，经过如下几个步骤即可生成符合该分布的随机数。（其中数学推导等在此处略去，详见相关数学书籍）

1. 计算 CDF 函数的反函数：$F^{-1}(x)$
2. 生成服从(0,1)区间上均匀分布的初始随机数 a
3. 令 x=$F^{-1}(a)$，则 x 即服从我们需要的特定分布的随机数。

为了更形象解说这种方法，这里选取柯西（Cauchy）分布作为例子。有时也称其为洛仑兹分布或者 Breit-Wigner 分布。柯西分布有一大特点就是，它是肥尾（Fat-tail，又译作胖尾）分布。在金融市场中，肥尾分布越来越受到重视，因为在传统的正态分布基本不考虑像当前次贷危机等极端情况，而肥尾分布则能很好地将很极端的情形考虑进去。



上图是 Cauchy 分布和标准正态分布 PDF 图对比，看看是不是 Cauchy 分布的尾巴（x 轴两端）更“胖”一点？

柯西分布的 PDF 函数是：

$$f(x) = \frac{\gamma}{\pi[\gamma^2 + (x - x_0)^2]}$$

简化起见我们只考虑 $x_0=0$， $\gamma=1$ 情形。此时 PDF 函数是：

$$f(x) = \frac{1}{\pi(1 + x^2)}$$

PDF 函数对 x 作积分，就得到 CDF 函数（推导过程略）：

$$F(x) = \frac{1}{2} + \frac{\arctan(x)}{\pi}$$

现在我们套用这三个步骤来生成服从 Cauchy 分布的随机数：

$$F^{-1}(x) = \tan[(x - \frac{1}{2})\pi]$$

1. 计算得到 Cauchy 分布 CDF 函数的反函数为：
2. 使用 rand()函数生成（0，1）区间上均匀分布的初始随机数。我习惯一次生成一堆这种随机数。

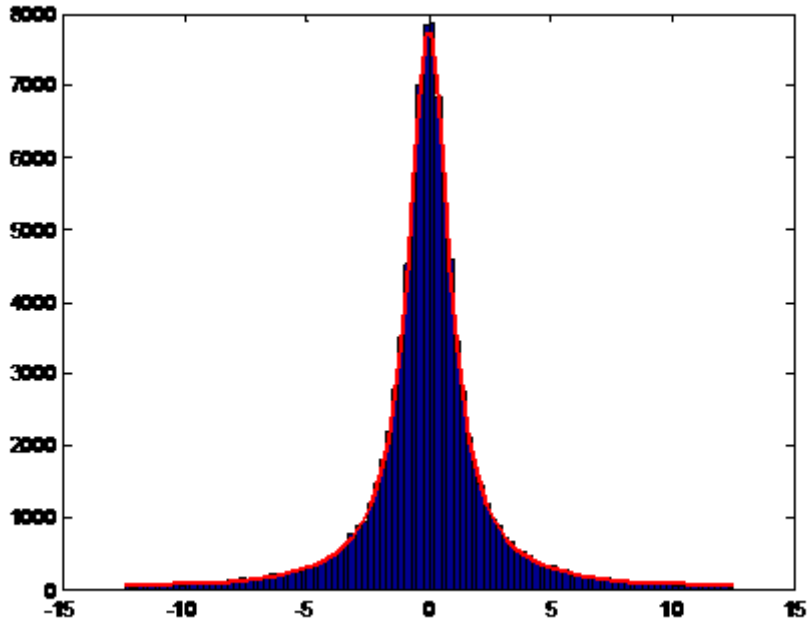
```
original_x=rand(1,100000);
```

3. 将初始随机数代入 CDF 反函数即可得到我们需要的 Cauchy 随机数。。

```
cauchy_x=tan((original_x-1/2)*pi);
```

上面这两句代码结合起来就生成了 10 万个服从参数为($x_0=0$， $\gamma=1$)Cauchy 分布的随机数。

这种方法生成随机数与 Cauchy 分布有多大相似之处呢？这里有一个图可以说明：



蓝色的图形就是用 hist 画出的随机数的样本分布情况，红色线条是 Cauchy 分布理论的 PDF 图形。由此可看出生成的随机数挺符合 Cauchy 分布。

注意：上图中，我略去了 x 轴小于-12.5 和大于 12.5 部分的图形——因为 Cauchy 是胖尾分布，会生成出的不少取值很大的随机数，而那些很大的值使得我们不方便用 hist 函数来画随机数分布图。

注意，这种方法本身虽然很简单，效率也很高，但有如下受限之处：

1．它有个可能会出错的地方，有的 CDF 函数的反函数在 0 或者 1 处的值是正/负无穷，例如此处的 Cauchy 分布就是这样，倘若用(0,1)均匀分布产生的初始随机数中包含 0 或者 1，那么这个程序会出错。幸运的是，迄今为止，我用 Matlab 的 rand() 函数生成的随机数中还没有出现过 0 或者 1。但不同版本的 Matlab 的这种情况也许会改变。此处提醒你，如果程序出错，不要忘记检查是否是这个错误。

2．CDF 函数必须严格单调递增，这也就意味着，PDF 函数在 x 定义域内必须处处严格大于零，否则 CDF 的反函数不存在。

3．即使 CDF 函数存在，如果它太复杂，可能导致计算速度太慢，甚至无法计算的后果。

2．接受/拒绝法 Acceptance-Rejection Method

Accelptence-Rejection 方法的精髓在于“形似”，可以形象地将其比喻为制作冰雕——二者相同之处在于都要首先堆砌出雏形，然后再用将多出的部分削去。用此法生成服从 f(x)分布的随机数，分为如下几大步骤：

1. 首先，选用某个分布，如 pdf 为 g(x)的分布，此时要计算一个常数 c，使得 $f(x) \le cg(x)$，对 x 定义域内任意的 x 都成立——这相当于使 cg(x)图形完全“覆盖”住 f(x)图形，容易理解，做冰雕时，最初堆出来的那堆冰块要比最终得到的雕塑大。

2．生成服从 pdf 为 g(x)分布的随机数，假设生成的随机数为 x_0。

3．再生成一个服从（0，1）间的均匀分布的随机数 y

4．如果 $y > \frac{f(x)}{cg(x)}$，丢弃生成的 x0；反之，生成的 x0 就是我们需要的、服从 f(x)分布的随机数。

下面用一个例子结合图形解释这种方法，假设我们要生成的分布是: $f(x)=\frac{(x-0.5)^2}{2.4}, x\in(0,2)$，此 pdf 图形如下图的蓝色曲线。

1．我们选用（0，2）之间的均匀分布作为原始分布，即 g(x)=0.5，此分布的 pdf 图见下图中的绿色线。由条件：无论哪个 x，$f(x)\le cg(x)$都要成立，我们计算得到 c 要大于等于 10.8。这种情况下，我们一般选择 c=1.875。因为 c 选得越大，意味着我们堆砌的原始雏形越大，需要削去的部分越多，效率越低，所以我们要使得 c 尽量地小。

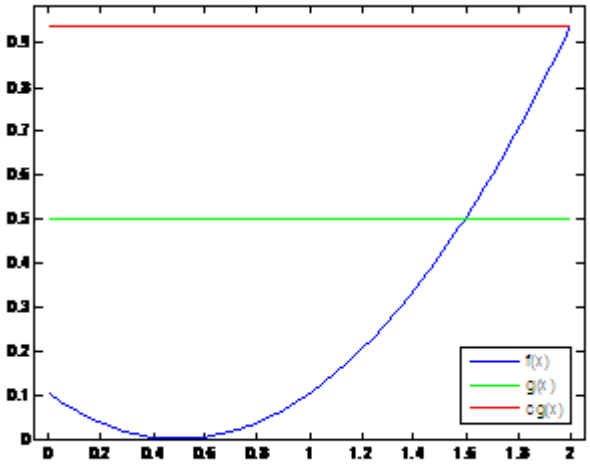
2．生成服从（0，2）之间的均匀分布的随机数，设它为 x0

```
X0=unifrnd(0,2);
```

3．然后再生成一个服从（0，1）间的均匀分布的随机数 y

```
Y=rand;
```

4．如果$y>\frac{f(x)}{cg(x)}$，丢弃生成的 x0，重新生成；反之，生成的 x0 就是我们需要的、服从 f(x)分布的随机数，用于做后续计算。



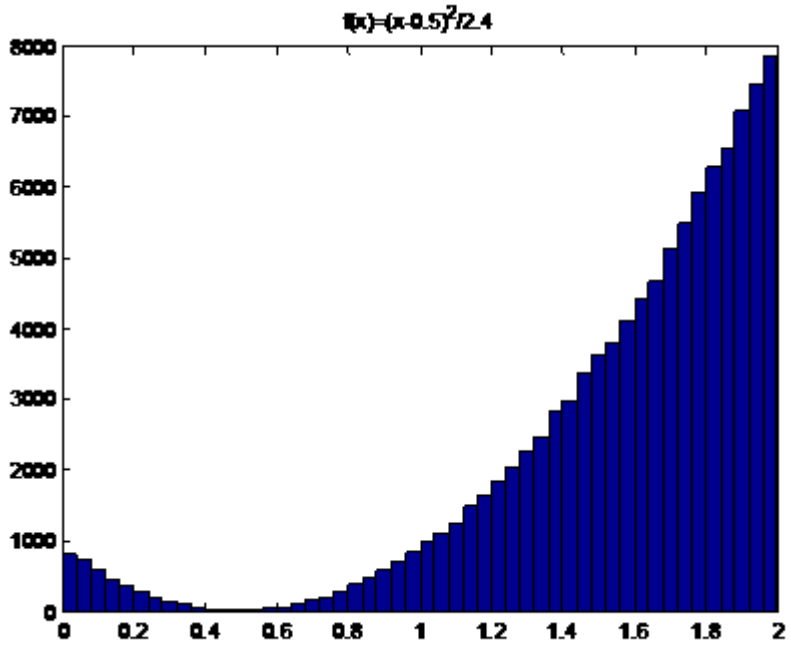
以上步骤每次只能处理一个随机数，效率较低，下面这段代码可以一次性生成一堆随机数。

```
N=400000;c=1.875;gx=0.5
x0=unifrnd(0,2,1,N);
y=rand(1,N);
fx0=(x0-0.5).*(x0-0.5)/2.4;
final_x=x0(y<=fx0./c/gx);
```

在视频教程中我会逐句解释每句含义，如果没听懂，一般是因为你对 Matlab 向量运算不熟悉，请参照 Matlab 基础教程学习此部分的内容，后面章节会有很多地方用得上。

这段语句生成的变量 final_x 即为服从 f(x)分布的随机数组成的一个行向量。我们可以用 hist 查看这些随机数大致的分布。

```
hist(final_x,50);title('f(x)=(x-0.5)^2/2.4');
```



如图所示，生成的随机数挺符合 f(x)分布。

这种方法很简单，也不需要计算 CDF 函数的反函数，但它也有如下受限之处：

1．由于我们用随机数 y 来控制是否削去某个随机数 x0，所以我们无法准确预知最终得到的随机数数量多少。

2．选择合适的 g(x)分布是此方法最关键的技巧所在。g(x)的选择原则是在完全覆盖 f(x)的前提下尽可能与 f(x)形似，二者形状越相似，需要削去的部分就越少，这种方法的效率就越高。需要记住的是：很多时候，人们不选用这种方法的原因几乎都在于它的效率过低。

（七）特殊离散分布

离散分布关键在获得它的分布律，有了分布律我们计算骰子掷出点数小于等于某个数字的累积概率分布。一个简单的例子，假设我们有一个不均匀的骰子，获得六个点数的概率分别是：

点数	1	2	3	4	5	6
概率	0.1	0.2	0.1	0.2	0.2	0.2
累积点数	≤1	≤2	≤3	≤4	≤5	≤6
累积概率	0.1	0.3	0.4	0.6	0.8	1

生成符合该分布随机数的步骤是：

1．生成一个（0，1）间均匀分布的随机数 x0。

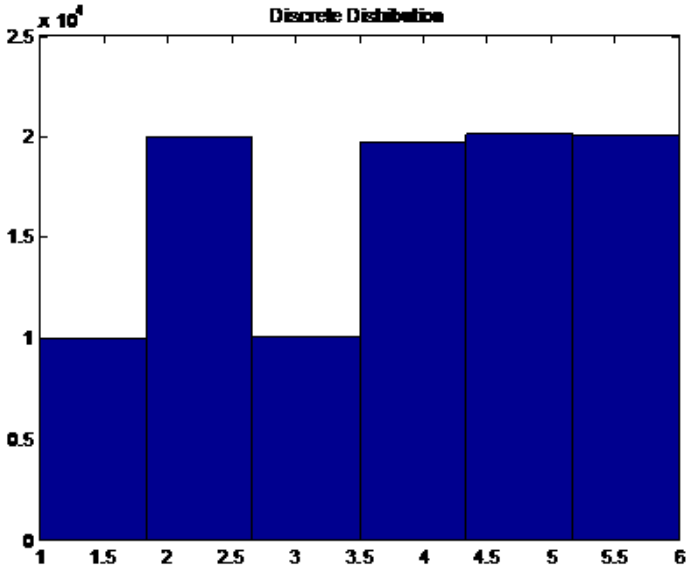
2．依据 x0 介于累积概率哪个区间来决定掷出骰子的点数 x。如 0<x0<=0.1,则点数 x 为 1，...., 0.8<x0<=1,点数 x 为 6。

代码是

```
x0=rand;
if x0<0.1
x=1;
```

```
    elseif x0<0.3
    x=2;
    elseif x0<0.4
    x=3
    elseif x0<0.6
    x=4
    elseif x0<0.8
    x=5
    else
    x=6
    end
```

这段语句能生成一个服从上表中离散分布的随机数 x，如果生成多个 x，可以用循环语句，也可以考虑将上述代码向量化。下图是我用上述代码生成 10 万个随机数所画出的分布直方图，可见这些随机数很符合上表中的分布律。



十七、生成多维联合分布随机数

一维随机变量是标量（也就是指单独的一个数字），而多维随机变量是一个向量。一个 n 维随机变量 x 是有 n 个分量的向量，$(X_0, X_1, \ldots, X_n)$，用 $f(X_0, X_1, \ldots, X_n)$ 表示联合分布，用 $f_k(X_k)$ 表示第 k 维的边缘分布，用 $f_k(X_k|X_1=x_1, X_2=x_2, \ldots X_{k-1}=x_{k-1}, X_{k+1}=x_{k+1}, \ldots, X_n=x_n)$ 表示当分量 $X_1=x_1, X_2=x_2, \ldots X_{k-1}=x_{k-1}, X_{k+1}=x_{k+1}, \ldots, X_n=x_n$ 时第 k 个分量 x_k 的分布。这里大写 X 表示随机变量某个维度上的分量，小写 x 表示具体的数值。关于边缘分布、条件分布、联合分布一定要明白，这些都是基础数学知识，非本课程内容。如果手头没有书，通过 google 搜索或上维基百科临阵磨枪也是可以的。

各种生成多维分布随机数的方法一般步骤都是，逐个维度生成随机数分量，最后将这些分量依次组合起来——如先生成 x_0，再 $x_1, \ldots$，最后 x_n，最终写成 $(x_0, x_1, \ldots, x_n)$。

在详细讲如何生成这些分量前，我们讲讲如何储存生成的随机数。

如果一次生成一个 n 维的随机数向量，可以用 n 变量来储存这个随机数的 n 个分量，也可以将这 n 个分量按照次序（次序不能乱）存于一个 1*n 的行向量中。如果一次生成随机数的数量很多，例如 N 个随机数，前面两种办法都可以用，即可用 n 个变量来储存这些随机数的每个分量，此时每个变量是 N*1 的列向量；也可以只用一个 N*n 矩阵储存随机数所有

分量，这个矩阵每一行是一个服从规定的联合分布的随机数，共有 N 行即表示共储存 N 个这样的随机数，矩阵的每一列表示这 N 个随机数的一个维度上的分量，共有 n 个维度。

（一）最简单的——各维度独立

各维度独立的联合分布随机数的生成最为方便，由于联合分布函数就是每个维度边缘分布函数的直接乘积，所以只要分别生成每个维度的随机数分量，然后组合成随机数向量即可得到服从该联合分布的随机数。

例子 1，生成一个在 0≤x≤2,0≤y≤2，正方形区域上的二维均匀分布。二维均匀分布在每个维度上都是均匀分布（即两个维度的边缘分布都是(0,2)上的均匀分布），且两个维度互相独立。

用第一种存储方法，

```
x=unifrnd(0,2);
y=unifrnd(0,2);
```

则每个维度上分别生成一个服从(0,2)均匀分布并分别储存在 x,y 这两个变量中。如果一次生成多个随机数，如 N 个，可用

```
N=400;
x=unifrnd(0,2,N,1);
y=unifrnd(0,2,N,1);
```

这里 x，y 都是 N*1 大小的列向量，分布存储着这 N 个随机数的第一维和第二维两个分量。我们看看这些随机数是否很好得符合二维均匀分布特性。

```
scatter(x,y);
```

接着，我们看看用上述第二种存储方法，

```
X=[x,y];
```

紧接着第一种存储方法中的语句，我们将生成的两个分量组合起来储存到一个变量中。当然这里还有一种取巧的办法，由于两个维度的边缘分布都相同且独立，我们只需用 unifrnd 函数一次性生成一个 N*n 大小的矩阵就可以了。

```
X=unifrnd(0,2,N,2);
```

例子 2，我们要生成的随机数服从一个三维联合分布，其第一维边缘分布服从标准正太，第二维边缘分布是自由度为 4 的 t 分布，第三维边缘分布是自由度为(7,8)的 F 分布，各个维度边际的边缘分布之间相互独立。我们只要用如下代码：

```
x1=rand
x2=trnd(4)
x3=frnd(7,8)
x=[x1,x2,x3]
```

x1,x2,x3 分布储存三个维度的分量（第一种存储方法），将这些分量组合起来存入 x 中（第二种存储方法）。

如果要一次就能生成一堆这样的随机数。可以用如下的代码：

```
N=1000;
x=[rand(N,1),trnd(4,[N,1]),frnd(7,8,[N,1])];
```

这段代码略过了中间过程，直接生成第二种存储方法所说的矩阵，这个矩阵大小为 N*3，我们可以大致观察该联合分布在每个区域内的概率密度的大小。

```
scatter3(x(:,1),x(:,2),x(:,3),'marker','.','sizedata',1);
```

注：x(:,1)表示将 x 矩阵的第一列（也即随机数第一维上的分量）提取出来。

点越密集的地方，该联合分布概率密度函数的值越大。你可以在 Matlab 中旋转图形更细致观察该分布。特别的，分别从 x,y,z 三个轴的角度看次图形的横截面图可以分别看到此三维度边缘分布的大致分布图形。

（二）最常用的——协方差阵生成多元正态分布

一个 n 维的随机变量，其协方差矩阵为一个 n*n 大小的矩阵，该矩阵对角线上的元素是随机变量各个分量的方差，矩阵其他位置的元素是各维分量两两之间的协方差；此随机变量的相关系数矩阵也是一个 n*n 大小的矩阵，该矩阵对角线上的元素都是 1，其他位置的元素是各维分量两两之间的相关系数。这两个矩阵关系非常密切。更详细的内容请参考数学书籍。

协方差矩阵及相关系数矩阵揭示了不同纬度之间的线性相关关系，它只是高维随机变量的部分性质，一般而言，仅仅依据协方差矩阵或相关系数矩阵再加上各维度的边缘分布信息，我们还不能确定此随机变量的联合分布。如下例子：

下表是一个两维的离散型随机变量。两个维度的边缘分布都是（1/4,1/2,1/4），两维间相关系数是 0.

维度 1\维度 2	-1	0	1	此维边缘分布
-1	0	1/4	0	1/4
0	1/4	0	1/4	1/2
1	0	1/4	0	1/4
此维边缘分布	1/4	1/2	1/4	

下表也是一个两维的离散型随机变量。两个维度的边缘分布都是（1/4,1/2,1/4），两维间相关系数是 0。但是这两个表的联合分布是不同的，即二者是两个不同的随机变量。这个例子说明边缘分布加相关系数并不能完全代表多维分布的所有信息。

维度 1\维度 2	-1	0	1	此维边缘分布
-1	1/8	0	1/8	1/4
0	0	1/2	0	1/2
1	1/8	0	1/8	1/4
此维边缘分布	1/4	1/2	1/4	

但是有一类特殊的分布：多元正态分布，它的全部信息可以浓缩成边缘分布加相关系数。多元正态分布的边缘分布都是正太分布，只要我们知道每个维度上的边际正太分布的均值和标准差，再加上相关系数矩阵，我们就可以得到整个联合分布。

此节所述方法的步骤：

1．依照给定的边缘分布的均值和标准差，分别独立地生成各个维度上的符合正态分布的随机数。并依次序组合成一个向量。

2．将相关系数矩阵作 Cholesky 分解

3．用分解得到的矩阵乘以第一步中生成的向量即可得到我们需要的随机数。

例子 1：假定我们要生成一个三维的多元正态分布。各个维度均值标准差如下表：

维度	均值	标准差
1	2	3
2	-1	2
3	0	1

相关系数矩阵如下表：

1	0.3	0.4
0.3	1	0.2
0.4	0.2	1

详细做法：

1．生成各维度上的独立的正态分布随机数。注：此处代码一次性生成 10 万个三维正态分布随机数，这些数组成了一个 100000*3 大小的矩阵。

```
N=100000;
x0=[normrnd(2,3,N,1),normrnd(-1,2,N,1),normrnd(0,1,N,1)];
```

2．将系数矩阵 R 做 Cholesky 分解得到矩阵 L。

```
R=[1,0.3,0.4;0.3,1,0.2;0.4,0.2,1];
L=chol(R);
```

3．计算 x0*L，即可得到 10 万个符合上述二表中条件要求的多元正态分布随机数，这些随机数被存储在一个 100000*3 大小矩阵中。

```
x=x0*L;
```

注意，最后一个语句是矩阵乘法，L 和 x0 的次序不能颠倒，否则会出错。

例子 2：生成本章第一节第三小节“多元分布”中所述相关系数为 0.3 的二维正太分布随机数。

1．这个二维分布在每个维度上的边缘分布都是标准正太。所以我们先生成由标准正太随机数组成的 N*2 矩阵

```
N=1000;
x0=randn(N,2);
```

2．由两个维度间相关系数为 0.3，可以知道其相关系数矩阵，并作 Cholesky 分解

```
R=[1,0.3;0.3,1];
L=chol(R);
```

3．最后计算 x0*L，即可得到 10 万个符合上述二表中条件要求的多元正态分布随机数，这些随机数被存储在一个 100000*2 大小矩阵中。

```
x=x0*L;
```

我们可以将这 10 万个随机数画在二维平面上。用如下语句：

```
scatter(x(:,1),x(:,2),'marker','.','sizedata',1)
```

（三）最一般的——由联合分布生成多维分布随机数（选读）

我们知道，联合分布函数包含多维分布随机数所有信息，所以我们直接从联合分布函数出发，通过相关的技巧生成随机数应该可以解决任意形式分布的问题。的确，只要给出联合分布函数 pdf，无论此联合分布如何诡异，我们用此节所述方法都有可能将服从此分布的随机数向量生成出来。但是这种方法涉及到很多计算，相当麻烦，故使用地很少。

这种方法的总体原则是：对联合分布 PDF 函数积分，计算出某个维度的边缘分布，用其生成随机数；再将已生成的这个维度的随机数代回联合分布函数，得到这个维度分量数值给定条件下的新的联合分布函数，不断重复上述过程，直到所有维度的值都确定为止。最后将各个维度的值组合起来即可得到我们所需的随机数。

假定联合分布的 PDF 函数为：$f(x_1,x_2,\cdots,x_k)$，生成符合此分布的随机数需要如下的步骤

1. 选择一个维度，计算该维度上的边缘分布，例如可以选 x1，计算得边缘分布为 $f_1(x_1)$

2．生成一个服从上述边缘分布的随机数 $\overline{x_1}$

3．令 $x_1=\overline{x_1}$，计算得到剩余维度的概率密度函数 $f(x_2,x_3,\cdots,x_k)$

4．再选择一个维度，依据 $f(x_2,x_3,\cdots,x_k)$ 计算该维度上的边缘分布，例如可以选 x2，计算得边缘分布为 $f_2(x_2)$。

5．生成一个服从上述边缘分布的随机数$\overline{x_2}$

6．重复进行 3，4，5 三个步骤，直到所有维度的随机数都生成。

7．最后将各维上的随机数组合成一个 n 维向量，该向量就是服从$f(x_1,x_2,\cdots,x_k)$分布的随机数向量。

生成服从某边缘分布随机数的方法要用到前一节讲的生成一维分布的方法。如果是常规分布，可以直接用 Matlab 内置函数，如果是特殊的分布，可以用逆 CDF 法或者 Acceptance-Rejection 法。

在视频教程中将以一个相对简单的例子进一步说明此方法。此例子是一个二维的随机数向量，联合密度函数是：

PDF：$f(x,y)=\frac{1}{4}(1+xy)$，定义域：$-1\le x\le 1,-1\le y\le 1$

（八）最流行的——Copula

现在 Copula 很热门，书籍很多，当然它也确实很有用。如果要详细讲 Copula，估计要一本挺厚的书才行。这里我们只简要介绍一点 Copula 的基础知识。需要指出，本课程的简单例子并没有用到 Copula，所以这个部分你完全可以跳过。

我们知道，多维随机变量联合分布一般都很复杂，为了将这个复杂的问题简化，我们常常考虑，各个维度上的边缘分布。现在的问题在于，联合分布与边缘分布之间的关系也很复杂。

最简单的情形是各个维度独立，则联合分布函数就是各个维度上边缘分布函数的直接乘积

稍微复杂一点是多元正态分布，我们知道各维度正态分布的均值方差（也即相当于知道了边缘分布），再加上两两维度之间相关系数的信息，我们就可以得到整个多元正态分布的联合分布函数。

但是，除此之外，我们就很难找到联合分布与边缘分布间比较简单的关系了。这种困难孕育了 Copula。Copula 是一种函数，这种函数揭示了联合分布与各维度边缘分布的关系。下面式子是一个二维分布的 Copula 函数：

$$\Psi(x,y)=C(\Psi_x(x),\Psi_y(y))$$

左侧是联合累积概率分布(多维随机变量的 CDF 函数)，右侧的 C 表示 Copula 函数，C()内部分布是 x 维度上的边缘分布和 y 维度上的边缘分布。这个式子的意义在于，我们只要知道边缘分布和 Copula 函数，整个联合分布就可以计算出来。

Copula 的名称也随着 C()函数形式的不同而改变。常见的有高斯 Copula，t-Copula，Archimedean Copula 等等。

Copula 的一个作用在于：我们从现实中能直接观察到的信息主要是边缘分布，利用这些边缘分布，然后选择合适的 Copula 函数类型，用这些信息去拟合现实数据，最后可以确定 Copula 函数中待定参数。至此，我们就能得到联合分布函数。

高版本 Matlab 附带的统计工具箱中有几个常用的 Copula 命令，使用很方便。这里我也顺带提一下如何自写代码生成 n 维的 Gauss-Copula。

1．Gauss-Copula 的参数是一个 n*n 相关系数矩阵，将此矩阵做 Cholesky 分解得到 L

2．生成一个 n 维、各维为独立标准正太的多元正态分布随机数向量 x0

3．计算 x0*L，并将结果的各个分量求其标准正态分布 CDF 函数的值

4．用各个分量分别代换到各维的 CDF 函数的反函数中，计算得到各维上的数值

5．最后将这些数值按照顺序组合成一个向量，此向量即为我们要的随机数向量。

代码及 Matlab 自带 Copula 命令见视频教程的 PPT。

（九）最熟悉的——Acceptance-Rejection Method

方法原理和前面一维的情况下完全一样。只要将那一小节中的所有步骤中的 x 改为 x 向量，就变成了多维情形下的 Acceptance-Rejection Method。此方法请自行研究。

多维情形下这种方法效率过低的弱点更加明显，所以选择一个好的 g(x)尤为重要。但问题在于，维数越高，联合分布越复杂，选择 g(x)就越困难。

十八、伪随机数的诅咒

很多书介绍了计算机生成的随机数都不是真正意义上的随机数这个问题，而且还详细讲明伪随机数会带来很多可能的危害。这个问题的确不可忽视，但我们也不要杞人忧天。以我们现在的技术水平，要想生成比 Matlab 更“真”的随机数基本不可能，所以，不要花太多时间在此方面。我建议注意以下两点：

1．由于同一台机器生成的随机数的模式固定，且有一定的规律，所以可以在一台机器上长时间运行同一个程序，以此来测试此程序的稳健性。因为程序中可能有些很微小的错误需要运行很长一段时间、落入某个特殊“陷阱”时才能显现。

2．不同机器随机数生成的机制会有差别，所以可以将同一套程序在多台配置不同的机器上运行，如果不同机器上得到的结果差不多，说明此程序质量很好，不受特定的随机数生成机制影响；否则，如果结果悬殊，则证明你的程序很可能有问题。

第三章：随机过程模拟

我们不可能确切知道未来经济变量——如，股票价格、收益率、债券利率、汇率等等——确切走势，一般我们采用与布朗运动有关的随机过程来描述这些变量的未来趋势。这里我们遇上的问题就是如何用计算机代码来描述这些随机的运动。

注意：这里只讲基于布朗运动的一维随机过程，它是学习其他深层次知识的基础。

基于布朗运动的随机过程是连续的，我们如何使用它们呢？依据计算任务的不同，一般分为如下这些情况：如果金融产品价格只基于到期日时标的资产价格，那么我们只需要由随机过程推导出标的资产在到期日价格的分布，然后用 Monte Carlo 方法，例如欧式期权就是此类；如果金融产品价格基于在到期日前标的资产在离散时间点上的市场价格，我们就用随机过程推导在每个盯市时间点上标的资产的价格的分布，然后用 Monte Carlo 方法计算，例如离散时间盯市的亚式期权；如果金融产品价格基于在到期日前标的资产在连续时间内的市场价格，这种情况比较麻烦，详见下一段。

由于我们不可能将时间无限细分，所以我们无法用计算机直接模拟连续情况下的随机过程，一般的做法都是将连续时间近似看做离散时间的版本。例如要用 Monte Carlo 方法计算连续时间盯市的亚式期权，我们只能将其转化为离散时间盯市的亚式期权。这一步转化必然要带来误差，这一误差不可避免，但随着离散化时将盯市时间间隔区段越短，所得结果越精确，当然计算时间也越长。具体将区段划分到何种程度，以及转化的误差有多大，可以用复杂的数学方法进行分析，这里推荐的一个简单的办法是，用不同长度的时间间隔区段分别做几次，将所得结果做一比较，即可大略知道区段划分会带来多大的误差影响，只要这个误差在可以接受范围之内，则一般可以将区段划分到该程度即可。

下面介绍三个简单的一维随机过程的模拟，更复杂的内容不太适宜在这个入门级别的课程中详细讲述，如果你想深入学习，可以参考相关书籍。

讲义和 PPT 的内容部分重叠，但表述方法不同，侧重点各异，故先简要讲解讲义，再详细讲解 PPT 内容。

一、标准布朗运动

标准布朗运动，又称维纳过程(Weiner Process)，用 W(t)表示。它的性质比较独特：

1．W(0)=0， 即我们定义初始时刻的点为 0 点。

2．W(t)~N(0,t)，在 t 时刻的位置服从均值为 0，**方差**为 t 的正态分布

3．W(s)-W(t)~N(0,s-t)，从时刻 t 走到时刻 s（s 要大于 t），位置的变化服从均值为 0，方差为 s-t 的正态分布，且该分布与 W(t)独立，但不与 W(s)独立，因为 W(s)=W(t)+(W(s)-W(t))。

有了这些性质，我们可以模拟标准布朗运动（准确说是一维的标准布朗运动）。分两种情况：

如果我们只要看终点时刻 T 时的位置，则有性质 2，我们可知，其在时刻 T 服从 N(0,T)分布，直接生成服从该分布的随机数即可。

如果我们需要模拟此运动从时间 0 到时点 T 整个过程中的运动路径。首先需要明确一点，我们只能模拟离散时间点的位置。此处我们用 $t_1, t_2, \ldots t_n$ 表示即将用于模拟的时间点。由上面的性质 2 和 3，我们有:

- W(t1)~N(0,t1)
- W(t2)-W(t1)~N(0,t2-t1)
- W(t3)-W(t2)~N(0,t3-t2)
- .....
- W(T)-W(tn)~N(0,T-tn)

所以我们要做的是分别生成服从 N(0,t1)，N(0,t2-t1)，N(t3-t2)，……，N(0,T-tn)这些随机数，在第 i 个时间点 ti 时此运动的位置 X(ti)就是将服从 N(0,$t_i - t_{(i-1)}$)以及在这个分布之前的所有分布的随机数累加起来。

详细例子见视频教程中展示的 PPT

十九、带漂移的布朗运动

标准布朗运动的表述是：$X(t) = W(t)$

一般形式的布朗运动的表述是：$X(t) = \mu t + \sigma W(t)$，其微分表述是：$dX(t) = \mu dt + \sigma dW(t)$。这里 μ 是漂移系数，σ 表示波动率。我们通过如下转化：$Y(t) = \frac{X(t) - \mu t}{\sigma}$ 就得到了标准布朗运动。

这个转化告诉我们，如果要生成一般形式的布朗运动，我们只要先生成一个标准布朗运动，然后由 $X(t) = \sigma Y(t) + \mu t$ 即可得到所需的一般形式。

具体例子见 ppt

二十、几何布朗运动

几何布朗运动的作用是用来模拟股价的变动。它的好处在于，一般形式布朗运动中取值可能为负数，而几何布朗运动取值永远不小于 0，这一点符合**股价**永远不为负的特征。

几何布朗运动微分形式的表述。或者称 SDE（随机微分方程）形式：

$$\frac{dS(t)}{S(t)} = \mu dt + \sigma dW(t)$$

其中的 S(t)可以理解为股价。

几何布朗运动函数形式表述：

$$S(t) = S(0)e^{[(\mu - \frac{1}{2}\sigma^2)t + \sigma W(t)]}$$

上述式子告诉我们，可以先生成一服从 $X(t) = (\mu - \frac{1}{2}\sigma^2)t + \sigma W(t)$ 的一般形式布朗运动，然后求其指数函数，最后乘以 S(0)，即期初的股价，就可以得到几何布朗运动。

补充：为何这里 t 的系数多出一项？具体可以参考伊藤公式。

第四章：例子

太多理论纸上谈兵，例子才是王道。每个例子有两个代码文件，前一种代码用循环语句编写，后一种代码用向量化语句编写。购买视频教程并确认成功后，我将把代码文件随 PDF 讲义文档发送至你的邮箱。代码文件命名规则：如 eg31.m 表示第三个例子第一种代码文件。

一共五个例子，前两个是纯粹的数学实例，后三个是三种金融产品理论定价模型的例子。需要注意，为简化起见，这里的例子是理论定价模型，附加了很多假设，所以不要生搬硬套直接拿到市场里面用，否则后果很严重。

另外，每个例子我都给出两套程序，第一套用循环方法做，速度慢，但简单易懂；第二套用向量做，速度快，但代码很复杂。这个课程要求两种方法里面掌握一种即可，尤其是第二套方法，我不能保证你一定能懂。不过即使不懂，多看看视频教程和讲义，多练习练习对提高你的 Matlab 编程水平还是很有益处的。

一、关于停止条件

在开始讲例子前，先回顾第一章所述关于 Monte Carlo 方法基本步骤的内容：

- ✓ 依据概率分布 ψ(x)不断生成随机数 x，并计算 f(x)

由于随机数性质，每次生成的 x 的值都是不确定的，为区分起见，我们可以给生成的 x 赋予下标。如 xi 表示生成的第 i 个 x。生成了多少个 x，就可以计算出多少个 f(x)的值

- ✓ 将这些 f(x)的值累加，并求平均值

例如我们共生成了 N 个 x，这个步骤用数学式子表达就是

$$\frac{\sum_{i=1}^{N} f(x_i)}{N}$$

- ✓ 到达停止条件后退出

常用的停止条件有两种，一种是设定最多生成 N 个 x，数量达到后即退出，另一种是检测计算结果与真实结果之间的误差，当这一误差小到某个范围之内时退出。

- ✓ 误差分析

Monte Carlo 方法得到的结果是随机变量，因此，在给出点估计后，还需要给出此估计值的波动程度及区间估计。严格的误差分析首先要从证明收敛性出发，再计算理论方差，最后用样本方差来替代理论方差。在本课程中我们假定此方法收敛，同时得到的结果服从正态分布，因此可以直接用样本方差作区间估计

这里比较难以处理的是“停止条件”，我们说过，Monte Carlo 方法计算结果都是随机变量，其与真实值之间必然有误差。从工作应用角度来讲，我们直接的目标当然是先设定好我们能够忍受的误差幅度，接着编程用 Monte Carlo 模拟，不断地重复生成随机数并计算，直到结果误差符合要求。所以，基于这种模式的 Monte Carlo 程序，每次或每隔一定次数，就要加上一句判断语句，直到判定结果的误差已经符合要求了才退出。

这种模式最大的问题在于，判断语句是需要成本的，为了得到一个结果，我们做 Monte Carlo 模拟的次数可能在几万次甚至更高，若每次模拟都要判断一次，那整个程序会慢很多，因此，如果你想写这种模式的程序，最好设定一个较大的值 M，使得只在每做完 M 此模拟后才判断一次误差是否满足。

话说回来，即使上面那种办法，也只比较适合 C，C++等程序语言，对于 Matlab 语言来说，它最大优势在向量运算，与其边做模拟边加判断，我不如设定一个大得多的 N，用向量运算的语法一次性将结果算出来，说不定时间还更短。

所以这里所有的例子里面，我都直接设定 Monte Carlo 模拟的次数为 N，然后根据计算出的结果以及经验来决定选择多大的 N 值。**一般而言，N 每增加为原来值的 100 倍，计算结果的标准差就减小为原来值的 1/10、精确度提高 10 倍、有效数字增加一位。**

二十一、计算 $\int_0^2 e^x dx$

这个例子在第一章已经讲过，这里用更严谨的步骤复习一遍，以便开展后面的内容。

回顾，我们要解决的 Monte Carlo 问题都是对服从 ψ(x)概率密度函数分布的随机数 x 多对应的取值函 f(x) 做数值积分的形式，即 $\int_{x0}^{x1} f(x)\psi(x)dx$ 这种形式，标题中的积分表达式不是这种形式，所以我们先改写其为 $\int_0^2 (2e^x)(\frac{1}{2})dx$，这里 ψ(x)= $(\frac{1}{2})$，意味着 x 服从（0，2）区间的均匀分布。

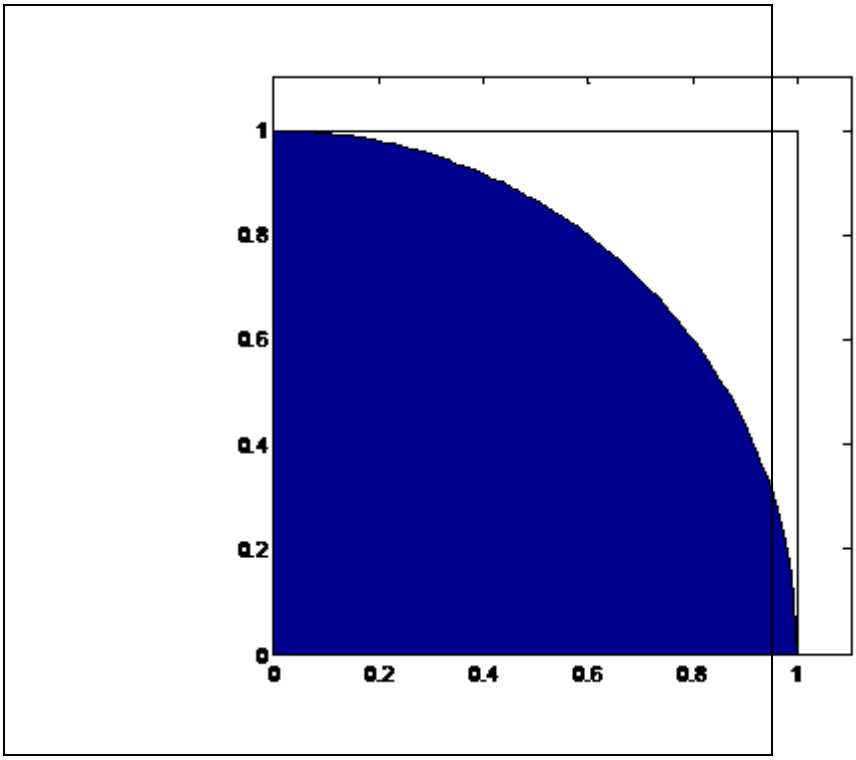
接下来请分别打开 eg11.m 和 eg12.m 两个代码文件，跟随视频教程的解说，详细了解这个例子的代码编写技术。其中前一个 m 文件用循环的办法，每次只生成一个随机数，优点是方便对 Matlab 向量编程技术不太熟悉的网友阅读，缺点是这种方法比较低级，速度慢。后一个文件对代码彻底优化，全部采用向量化代码，优点是速度快，且你可以从中学到 Matlab 向量化编程的很多小技巧，缺点是这些技巧不是一天两天就能熟练地运用，刚开始一般都很难适应，甚至很多都看不懂——看不懂没关系，结合视频教程，多听多看，多在 Matlab 上练习即可。

二十二、计算圆周率

这个例子主要演示如何生成并使用服从联合分布的随机数。

我们用这种办法计算 π。如图，一个边长为 1 的正方形内接一个半径为 1 的半圆。正方形面积是 1，半圆面积是 π/4。在正方形区域内生成一些均匀分布的随机数，随着生成的随机数数量增加，这些随机数落在半圆内的比例会趋近 π/4。

由这个几何图形我们可得到计算 π 的 Monte Carlo 方法：先生成 N 个正方形区域内均匀分布的二维随机数，然后计算这 N 个随机数中有多少个落在半圆区域内，设其为 M 个，最后 M/N 就等于半圆面积除以正方形面积——此比例是 π/4。



二十三、欧式期权定价

这个例子主要讲述如何生成并使用服从几何布朗运动的随机数

我们假设当前时点为0，股价为S_0，股票波动率σ，无红利，一个欧式看涨期权(call option)的 striking price 为 K，到期日迄今时间长度为 T，市场无风险利率为 r，注意上述所有变量的时间单位要一致。由 Black-Scholes 公式

$$V_0 = S_0\Phi\left[\frac{\ln(S_0/K)+(r+\sigma^2/2)T}{\sigma\sqrt{T}}\right]$$
$$-Ke^{-rT}\Phi\left[\frac{\ln(S_0/K)+(r-\sigma^2/2)T}{\sigma\sqrt{T}}\right]$$

可以计算此欧式期权的价值。

用 Monte Carlo 怎样做呢？这里最重要的是依据 Black-Scholes 公式的假设：股票价格服从几何布朗运动，从而依照在前一章讲述的方法推导出时间 T 时股价的概率分布。

详细推导见视频教程中的 PPT 讲解。

此例子同样也有两个版本的 m 文件——eg31.m 和 eg32.m，请参照视频教程逐句学习这两个代码文件。

二十四、计算亚式期权

这个例子主要讲述如何生成路径。

这里，我们以一个算术平均、离散时间盯市的亚式看涨期权作为例子。参数假设继承自前一个例子：我们假设当前时点为 0，股价为 S_0，股票波动率 σ，无红利，一个亚式看涨期权(call option)的 striking price 为 K，到期日为 T，市场无风险利率为 r，注意上述所有变量的时间单位要一致，在这个例子里面，时间单位为天。如 T=90 表示还有 90 天到期。

和欧式期权到期日计算收益用 max(S(T)-K,0)不同，亚式期权用的是 max(Sa-K,0)。此处的 Sa 来源如下：

在到期前，记录每天的股票价格 S(t), t=1,2,...,T, 取这些价格的算术平均值得到, Sa=(S(1)+S(2)+,...,+S(T))/T。

这么一个简单的变化，直接导致我们在做 Monte Carlo 模拟时，仅仅计算出到期日时股价的分布式不够的，现在我们需要的是模拟出从 1 到 T 每天股价变化的路径。建议先参考前一张关于几何布朗运动路径的模拟，再结合视频教程学习程序文件。

二十五、计算 Accumulator

这个例子主要讲述当基于随机数 x 的函数 f(x)比较复杂时如何操作。

Accumulator(昵称是 I'll kill you later，在港资银行向内地富豪推销时又被冠以打折股票）， 去年内地很多富豪因为购买这类金融产品而血本无归。网络上还有个特别的网站专门申讨此类产品——可以谷歌一下。

按照那个网站的介绍：“打折股票不是股票，而是一项跟股票挂钩的风险极高的复杂金融衍生品”，这句话基本是对的。不过，它的复杂程度也不是太复杂，毕竟它依然可以拆解成一些常规的金融产品。至于那个网站所说“适合对未来一年市场看涨时买进，看跌时千万不能做，否则将导致巨亏”，问题也不太大，但这个产品的实质在于，私人银行和富豪们签订这个产品的合约时，这个产品的内在价值实质是负数，而银行没有花一分钱，这实际就意味着，在签订产品合约时银行就包赚不赔了。

Accumulator 的标的资产不一定就是股票，外汇也行，当然其他类型的资产也有可能，但没那么常见。一般香港私人银行卖的 Accumulator 合约是一个多期的合约，按照时期来拆解，每个时期的合约相当于几个期权的叠加。

由于这一小节例子主旨在于讲怎样处理比较复杂的 f(x)函数，所以，我们就用一个简化的例子——这个例子中只有一期合约。

我们假设当前时点为 0，股价为 S_0，股票波动率 σ，无红利，市场无风险利率为 r。Accumulator 到期日为 T。Accumulator 合约中有三个价格,我们设其为 K，Ka，Kb，其中 K 为交割价，Ka 为第一行权价，Kb 为第二行权价，Kb>K≥Ka。另外 Accumulator 还规定了一个倍数 m，m>1,一般为 2。

设在到期日股票价格为 S(T)，则该合约在到期日的价值分布如下：

S(T)价格区间	S(T)>Kb	Kb≥S(T)≥Ka	Ka>S(T)
合约多头收益 (负则为损失)	0	Max{S(T)-K,0}	(S(T)-K)*m

下面结合这个表及代码文件观看视频教程。从程序计算的结果来看，这个 accumulator 的价值是负数。即持有 Accumulator 多头的富豪们，在最初和银行签订协议的时候就已经亏损了。

关于参数，我举个例子，此例来自“打折股票网”，也即我前面提到大家用谷歌搜索的网站。

假设中国移动的股票价格现为 100 港元，如果一份 Accumulator 合约规定 10%的折让行使价，5%的合约终止价，两倍杠杆，一年有效期，在中国移动 100 港元的股价时，投资人可以在其一年期限内，以九折优惠即 90 港元的行使价每月买入中国移动的股份。如果中国移动的股价涨到了 105 港元，这份合约就自动终止了。但是如果中国移动的股价跌破了 90 港元 Ka，投资者必须以 90(K)港元的价格每天买双份的中国移动股份，直到一年合约期满。

这个例子里面的参数是：S0=100，K=90，Kb=105，Ka=90，以上价格单位均为港元，m=2，T=1 年。

第五章：并行计算

并行计算有两种：共享内存式和消息传递式。Matlab 并行计算工具箱中的函数不少，不过我们这里只需要用上 parfor 这个函数。虽然这里介绍的使用 Matlab 并行计算工具箱的方法可以用于集群系统，但是这里只考虑在双核或多核机器上用此工具箱。

此章代码也可用于单核心 CPU 机器，但速度将很慢；内存低于 1G 请勿尝试此章代码。此章代码文件命名规律：如 pareg4.m 表示第四个例子的代码文件。

一、Matlab 并行计算原理梗概

Matlab 的并行计算实质还是主从结构的分布式计算。当你初始化 Matlab 并行计算环境时，你最初的 Matlab 进程自动成为主节点，同时初始化多个（具体个数手动设定，详见下文）Matlab 计算子节点。Parfor 的作用就是让这些子节点同时运行 Parfor 语句段中的代码。Parfor 运行之初，主节点会将 Parfor 循环程序之外变量传递给计算子节点。子节点运算过程时互不干扰，运算完毕，则应该有相应代码将各子节点得到的结果组合到同一个数组变量中，并返回到 Matlab 主节点。当然，最终计算完毕应该手动关闭计算子节点。

二十六、初始化 Matlab 并行计算环境

这里讲述的方法仅针对多核机器做并行计算的情况。设机器的 CPU 核心数量是 CoreNum 双核机器的 CoreNum2，依次类推。CoreNum 以不等于核心数量，但是如果 CoreNum 于核心数量则核心利用率没有最大化，如果 CoreNum 于核心数量则效率反而可能下降。因此单核机器就不要折腾并行计算了，否则速度还更慢。下面一段代码初始化 Matlab 并行计算环境：

```
%Initialize Matlab Parallel Computing Enviornment by Xaero | macro2.org
CoreNum=2; %设定机器 CPU 核心数量，我的机器是双核，所以 CoreNum=2
if matlabpool('size')<=0 %判断并行计算环境是否已然启动
matlabpool('open','local',CoreNum); %若尚未启动，则启动并行环境
else
disp('Already initialized'); %说明并行环境已经启动。
end
```

运行成功后会出现如下语句：

```
Starting matlabpool using the 'local' configuration ... connected to 2 labs.
```

如果运行出错，按照下面的办法检测：

首先运行：

```
matlabpool size
```

如果出错，说明你没有安装 Matlab 并行工具箱。确认安装了此工具箱后，运行：

```
matlabpool open local 2;
```

如果出错，证明你的机器在开启并行计算时设置有问题。请联系 MathWorks 的售后服务。

二十七、终止 Matlab 并行计算环境

用上述语句启动 Matlab 并行计算环境的话，在你的内存里面有 CoreNum1 个 Matlab 进程存在，每个占用内存都在百兆以上。（可以用 Windows 任务管理器查看），故完成运行计算后可以将其关闭。关闭的命令很简单：

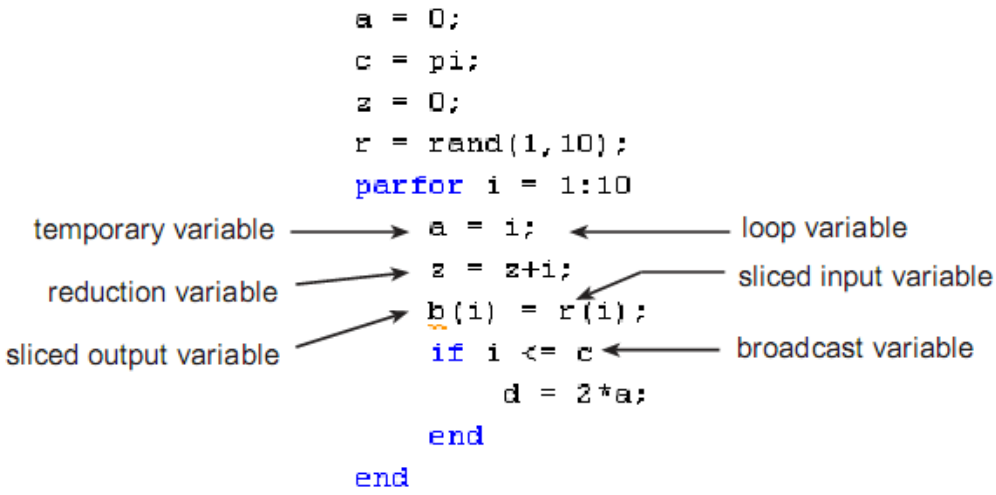
```
matlabpool close
```

二十八、Matlab 做 Monte Carlo 并行的算法

Matlab 并行计算比较特别。下图节选自 Matlab 并行计算工具箱用户手册。这个列表告诉你 Matlab 如何处理 Parfor 并行计算程序段中的各种变量。所以写代码时要注意不少问题，否则写出的并行代码可能还不如非并行的代码快。

Classification	Description
Loop	Serves as a loop index for arrays
Sliced	An array whose segments are operated on by different iterations of the loop
Broadcast	A variable defined before the loop whose value is used inside the loop, but never assigned inside the loop
Reduction	Accumulates a value across iterations of the loop, regardless of iteration order
Temporary	Variable created inside the loop, but unlike sliced or reduction variables, not available outside the loop

Each of these variable classifications appears in this code fragment:



这里我推荐大家用 Matlab 写 Monte Carlo 并行代码时按照以下注意事项来写：

1．将 Monte Carlo 模拟过程中不会改变的参数都写在 Parfor 循环块外面

2．生成随机数、计算 f(x)等过程都写在 Parfor 里面

3．不要将 V0 结果传递出 Parfor，而是直接计算出 V0 的均值、方差传递出 parfor。

4．最后用数学公式将传递出 Parfor 的 V0 的均值方差组合计算成最终结果

这些事项如何体现到程序中请参照示例代码文件并结合视频教程学习。这样的并行办法简单易行，对原始程序没有太大的改动，同时传递变量耗费时间也较少，效率比较高。

另外一个问题就是并行代码做模拟的次数问题。我们要达到用非并行的代码做 N 此模拟所能得到结果的精确程度，在核心为 CoreNum 并行代码中，Parfor 语句段中只要做 N/CoreNum 次即可达到。

二十九、将例子改写为并行代码

附件中的 pareg1.m，……，pareg5.m 五个文件分别是前一章五个例子的并行代码。这里需要提到的是，这五个代码文件都是用向量化的代码编写。原因在于，在前一章大家都看到了，向量化的代码比循环语句代码一般快几十甚至上千倍，所以要提高速度，向量化代码是最重要的优化方法，并行计算倒是其次。

三十、速度实测结果

需要指出的是，虽然有 CoreNum 核心同时工作，并行代码的速度并不能达到非并行代码的 CoreNum。很多时候，如果要做的模拟次数（N）很小，并行代码反而更慢。从下面的表可以看出，并行代码比非并行代码快，前提是 N 很大。

由于我的机器在录制视频教程的同时无法做并行计算，所以在视频教程中，我虽然会像在前一章那样将每个代码文件每行语句详细讲解，但是我没有办法演示运行这些代码文件。这里将我机器的实测结果列表以备大家参考。

我机器大致配置是: E2200@2.42GHz，MSI G31 主板， Winxp32 位，内存 3G，并行运算环境：Matlab2009a。CoreNum=2。机器不同，运行时间会有所差异。

例子编号	非并行向量化代码		并行向量化代码		总模拟次数**
	文件名	运行时间(s)*	文件名	运行时间(s)	N=
例子1	eg12.m	0.333~0.336	pareg1.m	0.236~0.239	300 万次
例子2	eg22.m	0.406~0.410	pareg2.m	0.366~0.371	800 万次
例子3	eg32.m	0.224~0.268	pareg3.m	0.182~0.189	200 万次
例子4	eg42.m	3.507~3.825	pareg4.m	2.187~2.324	30 万次
例子5	eg52.m	0.525~0.581	pareg5.m	0.325~0.331	200 万次

*：多次运行，取大致区间。

**：非并行代码的总模拟次数就是 N，并行代码的总模拟次数是每个 Parfor 循环中模拟次数*CoreNum。此处总模拟次数相同的非并行及并行程序，其结果误差的数量级相同，所以可以用运行时间长短来衡量程序速度；比较模拟次数不同程序的运行时间是没有意义的。

第六章 大规模蒙特卡洛实验的实现

Monte Carlo 方法得到的结果是随机变量。随机变量的特点是采样数量越多，方差越小，即意味着结果精确度越高。因此在对精度要求很高的时候，我们需要进行大规模蒙特卡洛模拟。

从理论来说，当 Monte Carlo 模拟次数达到无穷大时，所得的结果将变成没有误差的确定值。但是，由于计算机内存容量的限制，程序一次性能做的 Monte Carlo 模拟的次数是有限的。这个问题在内存消耗巨大的向量化代码中体现更为明显。

、一、原理与数学推导

以例四：计算亚式看涨期权价值为例子，其占用内存空间的大小可以用 N*T 来衡量。例如，在我机器上（3G 内存,32 位 WinXP 系统），当 N*T 不能大于 4000 万，这就意味着如果时间 T
=90，那么这段程序最多只能做 40 多万次 Monte Carlo 模拟。（大家可以修改该段程序，计算 40 万次模拟结果的标准差。）如果需要在内存空间限制的条件下，获得比这个标准差更小的结果，我们就需要新的办法。一般而言我们两条途径可以选择，无论哪条路径，原理都是避开耗用内存巨大的中间过程，每次计算一小部分，将得到的结果储存起来，等到所有模拟都做完后再汇总储存的结果。这两种途径的区别在于我们储存的结果是模拟后得到的裸数值还是中间汇总值。

为了方便后面比较前述两种方法的效率，以及在给定精度要求的前提下预计模拟次数，在讲述具体的做法前，我们先以前面章节的例四（即 eg42.m 文件）为例，用数学方法推导常规方法模拟计算结果的标准差如何来估计。

现有 N 条路径，每条路径对应的欧式看涨期权期初价值为 Vi,(i=1,2,3,...N)，则最后得到的蒙特卡洛方法计算的期权价值为$\frac{\sum_{i=1}^{N} V_i}{N}$。此 N 条路径的样本标准差为 S。由此我们推出模拟出的期权价值标准差为$\frac{S}{\sqrt{N}}$。（这也就是 Eg42.m 显示的标准差）。

继续以例四来说明这两种方法怎样做。现在假设我们的机器内存容量限制，一次性只能做 N 次(例如 N=40 万）模拟，而我们需要的精度要求更多的模拟次数。这两种方法如下：

（一）方法一：

1．1．模拟 N 次，得到每条模拟路径上的平均股价，对应程序 eg42.m 中的 Sa 变量。
2．2．将 Sa 变量整个储存到某个变量中，如 SaAll
3．3．重复如上过程 M 次，最后得到的 SaAll 中共有 NM 条路径上的平均股价值
4．4．计算这 MN 条路径在终点时刻的期权价值，再贴现回起点时刻
5．5．最后将这 MN 条路径起点时刻价值求平均

理论依据及数学推导：

依据这种求期权价值方法的原理，每条路径期权价值 Vij, i=1,2,....M, j=1,2,.....N. 服从正态分布。

$$V_{ij} \sim N(\mu, \sigma^2)$$

其中 μ 是期权真实价值。对 Vij 求平均，$\frac{\sum_{i=1}^{M}\sum_{j=1}^{N} V_{ij}}{MN} = \mu$，数学含义是这 MN 条路径期权的价值平均值是无偏的——即我们可以认为得到的平均值就是该期权的真实价值。误差分析也

很简单，求这 MN 个值的样本标准差 S——对应的 Matlab 函数是 std()——用这个样本标准差来替代σ。接着，由$std(\frac{\sum_{i=1}^{M}\sum_{j=1}^{N}V_{ij}}{MN})=\frac{\sigma}{\sqrt{MN}}=\frac{S}{\sqrt{MN}}$。这也就是 eg42.m 程序汇报的“模拟结果标准差”。

由此我们发现，每轮做 N 次模拟，共做 M 轮得到的标准差等于$\frac{\sigma}{\sqrt{MN}}$,是只做一轮模拟得到的标准差（等于$\frac{\sigma}{\sqrt{N}}$）的$\frac{1}{\sqrt{M}}$，简单地说，假定我们做 N 次模拟得到的标准差是 0.005,如果我们希望得到 0.001 的标准差，那么由上述关系：

$$\frac{\sigma}{\sqrt{N}}=0.005$$
$$\frac{\sigma}{\sqrt{NM}}=0.001$$

求解此方程，得到 M=25。由此可知，我们总共需要做大约 25 轮，每轮 N 次蒙特卡洛模拟。

这种方法计算方便，但是额外占用空间比较大，例如这里我们需要一个额外的大小为 MN 的一维向量存储这 MN 条路径对应的平均股价。由于 N 已经很大了，如果总模拟次数很多，即 M 也很大，那么内存的耗费将很可观。

（二）方法二：

1．1．模拟 N 次，得到每条模拟路径上的平均股价，对应程序 eg42.m 中的 Sa 变量。

2．2．计算这 N 条路径在终点时刻的期权价值，再贴现回起点时刻

3．3. 求这 N 条路径期权价值的均值 Vi，储存到 Vall 数组；并计算这些路径的方差 Si2，储存到 S2all 数组中。

4．4. 重复上述过程 M 次，于是 Vall 中共储存 M 个 Vi 值；S2all 中共储存 M 个 Si2 值。

5．5．最后将这 M 个 Vi 值取平均，此平均值就是模拟所得到的期权价值。

理论依据及数学推导：

依据这种求期权价值方法的原理，每条路径期权价值 Vij, i=1,2,....M, j=1,2,.....N. 服从正态分布。

$$V_{ij}\sim N(\mu,\sigma^2)$$

其中 μ 是期权真实价值。和方法一不同，这里加总 Vij 分两步走。第一步先对 j 加总，得到每轮 N 次模拟的均值:

$$V_i=\frac{\sum_{j=1}^{N}V_{ij}}{N},i=1,2,\ldots M$$

由数学定理：对独立的正态分布加总得到的还是正态分布。得到 Vi 服从正态分布

$$V_i\sim N(\mu,\frac{\sigma^2}{N})$$

又 Vi 的方差：

$$S_i^2 = \frac{\sum_{j=1}^{N}(V_{ij} - V_i)^2}{N-1} = \sigma^2, i = 1,2,\ldots M$$

第二步再对 Vi 加总:

$$V = \frac{\sum_{i=1}^{M} V_i}{M} = \frac{\sum_{i=1}^{M}\sum_{j=1}^{N} V_{ij}}{M} = \mu$$

即我们直接对 Vi 求平均值就得到期权真实价值的无偏估计。

再由独立的 M 个 Vi 正态分布相加再取平均值依旧是正态分布这一原理，我们可以得到上式 V 服从正态分布为：

$$V = \frac{\sum_{i=1}^{M} V_i}{M} \sim N(\mu, \frac{\sigma^2}{MN}) \qquad (*)$$

接下来做方差分析。我们可以从 Vi 的方差出发。数学定理告诉我们，Vi 的样本方差经过变换服从卡方分布。

$$\frac{(N-1)S_i^2}{\sigma^2} \sim \chi(N-1)$$

而 M 轮模拟得到的 M 个样本方差之间是独立的，因此这些样本方差累加起来也服从卡方分布：

$$\frac{(N-1)\sum_{i=1}^{M} S_i^2}{\sigma^2} = \sum_{i=1}^{M} \frac{(N-1)S_i^2}{\sigma^2} \sim \chi(M(N-1))$$

自由度为 M(N-1)的卡方分布的均值为 M(N-1)，由此性质我们有

$$E\{\frac{(N-1)\sum_{i=1}^{M} S_i^2}{\sigma^2}\} = M(N-1)$$

$$\Rightarrow \sum_{i=1}^{M} S_i^2 = M\sigma^2$$

$$\Rightarrow \sigma^2 = \frac{\sum_{i=1}^{M} S_i^2}{M}$$

上述式子的作用在于，我们将 M 轮模拟得到的 Si2 值取平均，此均值即可作为对 σ^2 的估计。再由(*)式表示的 V 的分布形式，可以计算：M 轮、每轮 N 条模拟路径得到的期权价值估计值的方差为：

$$Var(V) = \frac{\sigma^2}{MN} = \frac{\sum_{i=1}^{M} S_i^2}{M^2 N}$$

标准差就是：

$$Std(V)=\sqrt{Var(V)}=\frac{\sqrt{\sum_{i=1}^{M}S_i^2}}{M\sqrt{N}}$$

附注：计算这个标准差还有一种等价的方法，即不储存 S2 而是直接用 Vi 来计算。直接求 Vi 的标准差最后除以根号 M，得到的也是 MN 此模拟计算出的期权估计值的标准差。这种方法在 M 数量很大的情况下可用，否则其误差可能较大。依据个人经验，当 M>40 时结果比较合适。

总结：由此我们看到这两种方法得到的最终结果从统计学意义上来说完全相同。在看方法二的内存占用。由于我们只需要记录 M 个 Vi 和 Si 值，所以占用内存比方法一要小很多。

而且，第二种方法还有一个好处，就是 1~3 这些步骤在 eg42.m 中已经存在，因此可以很方便地在 eg42.m 的基础上写出第二种方法对应的代码。

综上，我们这里只介绍第二种方法。

、二、算法的实现

下面我们介绍如何将 eg42.m 改写为可以模拟任意多条路径的代码。首先我们分析 eg42.m。（具体见视频教程中的讲解）

1．1．参数设定的语句块。这些语句只需要运行一次，因此放入程序最首段。

2．2．路径模拟语句块。先生成了 N 条路径，再计算每条路径的股票价格，再得到每条路径对应的期末期权价值，然后贴现到期初，从而得到这 N 条路径各自对应的期初的期权价值 Vij。

3．3. 平均这 N 条路径，储存到变量 Vall 中，同时求这 N 个 Vij 值的方差，储存到 S2all 变量中

4．4．重复如上过程 M 次

5．5.最后由前面的数学推导，Vall 平均值就是期权价值估计值，加总 S2all 再除以(M^2N)最后开根号就得到了期权价值估计值的标准差。

、三、并行算法的实现

并行算法上实现这个过程很简单。注意到 2~3 这两需要循环 M 次的步骤恰好可以作为一个计算单元，故我们直接将这个计算单元放入 parfor 语句块中即可。其他部分就和前面章节所述的并行计算方法一致。

程序的具体写法在视频教程中详述。这里需要补充的是 Matlab 并行计算的内部操作过程。

这里 parfor 语句块中共要进行 M 次循环，Matlab 是这样对这 M 个循环块作任务调度分配的：

1．1．将这 M 个任务组成一个任务池

2．2．启动时给每个 CPU 核心分配一个任务

3．3．CPU 核心计算完成后，Matlab 再给它分配任务，直到任务池中的所有任务都完成

故，并不是说每个核心计算的任务数量相同，而是运算速度快的核心计算的任务可能更多一些。

最后总结一下：

通过这样编程，计算机 Monte Carlo 模拟的计算能力几乎可以达到无限。这是由于每轮可以模拟 N 次，这个 N 是受到计算机内存容量限制；但是我们可以模拟 M 轮，由于模拟 M 轮所占用空间只要 2*M 大小的向量空间，故 M 可以很大很大。当然，此时更需要考虑的是时间限制。例如在我机器上，eg42.m 每轮模拟 40 万次所需要时间大约是 5 秒钟，倘若我们要做 M=1 万轮，则总共需要时间是 5 万秒钟，几乎要运行大半天时间了——当然，用并行计算在多核心的机器上可以很有效地减少计算时间。