伪随机数、拟随机数和任意分布随机变量抽样程序库使用说明

杨自强　魏公毅

中国科学院数学与系统科学研究院
计算数学与科学工程计算研究所

2005 年 1 月

前言

本程序库包括蒙特卡洛和拟蒙特卡洛方法抽样方面的三部分子程序，它们分别用于产生**伪随机数、拟随机数和任意分布随机变量抽样**. 程序既可用于串行计算，另外，还提供一些跳跃子程序，以方便并行计算.

本库所有子程序名字的前两个字母都是 RN, 第三个字母是 P, Q 或 G 分别与伪 (Pseudo) 随机数、拟 (Quasi) 随机数和一般 (General) 的分布随机变量抽样对应.

这三部分子程序结构上是相对独立，所以，有关的使用说明也分成相对独立的三部分. 它们依次是

I. 伪随机数发生器使用说明

1. 基本发生器

伪随机数简称随机数，本库提供如下 4 个基本的随机数发生器：

```
real function    RNPFR1(IX)
subroutine       RNPSRN(IX, RN, N)
double precision function    RNPFD1(IX)
subroutine       RNPSDN(IX, DRN, N)
```

它们都产生区间 $(0,1)$ 上的均匀分布随机数. 其中前两个返回单精度实型随机数，后两个返回双精度实型随机数. 函数形式的发生器每次引用得到 1 个随机数，过程形式的发生器每次调用得到 N 个随机数存放在实型数组 RN(1:N) 或双精度型数组 DRN(1:N) 内. 整型数组 IX(1:2) 存放递推值，其初值 (亦称发生器的种子) 由用户提供，应满足

```
0 < IX(1) < 2146058219,        0 < IX(2) < 2145434063
```

值得注意的是，上述随机数发生器是由两个乘同余递推式：

$$X_{i+1}^{(j)} \equiv a^{(j)} X_i^{(j)} \bmod M^{(j)}, \quad j=1,2, \qquad M^{(1)} > M^{(2)},$$

按如下方式组合得到的：

$$X_{i+1} \equiv (X_{i+1}^{(1)} - X_{i+1}^{(2)}) \bmod M^{(1)}, \quad r_{i+1} = X_{i+1}/M^{(1)}, \quad i=0,1,2,3,\cdots$$

$\{r_i\}$ 便是要产生的区间 (0,1) 上的随机数序列. 式中 $X_0^{(1)}$ 和 $X_0^{(2)}$ 为用户提供的初值 (即程序中的 IX(1) 和 IX(2)).

组合发生器可以比只由单个乘同余递推式构成的发生器产生周期更长且统计品质更好的序列. 上述 4 个发生器的核心递推式的具体参数都是相同的，它们是：

模： $M^{(1)} = 2146058219, \quad M^{(2)} = 2145434063,$

乘子： $a^{(1)} = 43465, \qquad a^{(2)} = 45271.$

各发生器的差别只在于调用方式和结果精度的不同. 所以，对于相同的种子 IX, 如果不考虑结果的单、双精度表达，都会得到相同的随机序列. 只要种子 IX 的两个值符合上述条件，则序列的周期与 IX 的具体值无关，其长度约为 2^{61}.

上述发生器的调用程序例子见第 3 节. 更详细的算法背景见其后的参考文献.

2. 并行计算中可能使用的辅助程序

前述基本发生器产生连贯的随机序列 $r_1, r_2, r_3, r_4, r_5 \cdots$ (亦称间隔为 1 的序列), 但并行计算中有时需要产生跳跃序列，例如跳跃间隔为 100 的序列 $r_1, r_{101}, r_{201}, r_{301}, \cdots$

事实上，并行计算中的各处理器常使用同一个随机数发生器，在具体实施中可有 **分段** 和 **跳跃** 两种不同的使用方案. 若计算中使用 p 个处理器 (编号为 $0,1,\cdots,p-1$), 每个处理器需用 n 个随机数，这时

分段方案 中各处理器使用的随机数是：

处理器 0:	$r_1,$	$r_2,$	$r_3,$	$\cdots,$	r_n
处理器 1:	$r_{n+1},$	$r_{n+2},$	$r_{n+3},$	$\cdots,$	r_{2n}
处理器 2:	$r_{2n+1},$	$r_{2n+2},$	$r_{2n+3},$	$\cdots,$	r_{3n}
$\cdots$	$\cdots$	$\cdots$	$\cdots$		$\cdots$
处理器 $p-1$:	$r_{(p-1)n+1},$	$r_{(p-1)n+2},$	$r_{(p-1)n+3},$	$\cdots,$	r_{pn}

跳跃方案 中各处理器使用的随机数是：

处理器 0:	$r_1,$	$r_{p+1},$	$r_{2p+1},$	$\cdots,$	$r_{(n-1)p+1}$
处理器 1:	$r_2,$	$r_{p+2},$	$r_{2p+2},$	$\cdots,$	$r_{(n-1)p+2}$
处理器 2:	$r_3,$	$r_{p+3},$	$r_{2p+3},$	$\cdots,$	$r_{(n-1)p+3}$
$\cdots$	$\cdots$	$\cdots$	$\cdots$		$\cdots$
处理器 $p-1$:	$r_p,$	$r_{2p},$	$r_{3p},$	$\cdots,$	r_{np}

计算中，跳跃方案要求各处理器每次产生间隔为 p 的随机数．至于分段方案，虽然在每个处理器内的序列没有跳跃，但在提供给各处理器的随机数初值时也有跳跃间隔 n 的问题．为此，本程序库提供如下 **5 个并行计算辅助子程序** ：

```
real function    RNPFRK(IX)
real function    RNPFRI(DK)
double precision function    RNPFDK(IX)
double precision function    RNPFDI(DK)
subroutine       RNPSKIP(J, DK, ISEED)
```

函数 RNPFRK (或 RNPFDK) 返回实型 (或双精度型) 随机数，其跳跃间隔为 K. 引用前需由函数 RNPFRI(或 RNPFDI) 初始化，其参数 DK 就是用户指定的跳跃间隔 K 值的双精度形式，若 K 值不变，就不必再次初始化．因为我们的随机数周期很长 (接近 2^{61}), K 使用双精度形式后，便可允许跳跃间隔超过整型数的上限 $2^{31}-1$.

子程序 RNPSKIP 把组合发生器内的第 J 个 (J=1,2) 乘同余递推式的种子 ISEED (相当于前述的 IX(1) 或 IX(2)) 从当前值快速地跳到间隔为 DK(双精度型) 的对应值.

重要提示：使用本节提供的子程序产生随机数时，由于某些中间结果可能超出双精度数能精确表示的范围，要作技术上的处理，故其效率可能要比第 1 节的相应程序低数倍，应尽量少用．用户在并行计算中宜使用 **分段方案** 产生随机数．这时，在每个处理器内，使用第 1 节的基本发生器产生一段连贯的随机数序列；而此处的跳跃算法子程序仅用于产生各个处理器的随机数种子 IX 值.

调用本节子程序的例子请参考下节.

3. 程序调用例子

本节给出三个调用程序的示范性例子.

调用基本发生器的程序例子

下述程序调用过程 RNGSR 产生 100 个实型随机数存放于数组 RN 内：

```
      INTEGER  N, IX(2)
      REAL     RN(100)
      N = 100              ! length of random sequence to be generated
      IX(1) = 20041215     ! seeds for generator
      IX(2) = 12345
      CALL  RNPSRN(IX, RN, N)
      END
```

上述程序中调用过程 RNPSRN 的那个语句可被循环引用函数 RNPFR1 的语句代替：

```
      DO 10 K=1,N
        RN(K) = RNPFR1(IX)
10    CONTINUE
```

调用跳跃发生器产生各处理器的随机数种子

并行计算中使用分段方案时可用如下的程序获得各处理器的随机数种子 (程序中的 NP 和 N 分别是处理器数目和每个处理器使用的随机序列长度，所得的种子放在整型数组 IXX 内):

```
      INTEGER  IX(2),IXX(2,0:NP-1)
      NP = 8                ! NP is the number of processors
      N = 100000000         ! N is length used for every processor
      IX(1) = 20041215      ! original seeds
      IX(2) = 12345
      WK = RNPFRI(DBLE(N))  ! initialization (skip N,  WK is useless)
      DO 10 I=0,NP-1
        IF (I .GT. 0) WK = RNPFRK(IX)
        IXX(1,I) = IX(1)    ! save seeds for processor I
        IXX(2,I) = IX(2)
10    CONTINUE
        ...  ...  ...  ...
```

更简便的分段方案

本方案利用组合发生器特点，为并行计算中的每个处理器预留长度接近 2^{31} 的一段随机数. 设组合发生器的两个乘同余递推式分别产生如下序列：

$$X_1^{(1)},\ X_2^{(1)},\ X_3^{(1)},\ \cdots \text{ 和 } X_1^{(2)},\ X_2^{(2)},\ X_3^{(2)},\ \cdots$$

最方便的分段方案是令计算机上 p 个处理器所使用的那段随机数的种子分别是

$$(X_1^{(1)}, X_1^{(2)}),\ (X_1^{(1)}, X_2^{(2)}),\ (X_1^{(1)}, X_3^{(2)}),\ \cdots,\ (X_1^{(1)}, X_p^{(2)}).$$

这时各处理器中不重叠的序列长不小于 $\{X_i^{(1)}\}$ 的周期长度 $(M^{(1)} - 1 \simeq 21.46\text{亿} \simeq 2^{31})$, 这样的长度能满足绝大多数用户的需要. 下面的程序可获得各处理器的随机数种子：

```
      INTEGER  IX(2),IXX(2,0:NP-1)
```

```
      NP = 8                ! NP is the number of processors
      IX(1) = 20041215      ! original seeds
      IX(2) = 12345
      DO 10 I=0,NP-1
        IF (I .GT. 0) CALL RNPSKIP(2, 1.0D0, IX(2))
        IXX(1,I) = IX(1)    ! save seeds for processor I
        IXX(2,I) = IX(2)
   10 CONTINUE
```

4. 构造优良组合发生器的途径

L'Ecuyer-Tezuka (1991) 业已证明：前面介绍的一类组合发生器近似于一个关联发生器 (模很大的单个乘同余发生器), 从而简化了组合发生器的统计特性的分析, 因为乘同余发生器已被深入研究和广泛使用. 特别是谱检验方法, 只要知道 M, a 两参数值, 便可不必实际产生序列就能推知其主要的统计特性. 这为构造优良的组合发生器指明了方向：在计算机上产生两个乘同余发生器并算出其组合所得的关联发生器的 M, a 值, 进而作谱检验. 重复这一过程, 直至获得最佳或满意的谱检验结果. 这时, 对应的组合发生器就是最佳或较好者. 在实践中, 因为寻优过程对应着一个没有解析解的非线性整数规划问题, 而且在数值计算中, 大量的中间结果的精度要求已超过一般计算机的双精度, 甚至四倍精度范围. 我们在自建的任意精度运算平台上作了大量计算后, 得到了前面介绍的组合发生器. 其关联发生器谱检验值的 $2 \sim 6$ 维结果是：

$$0.9202, \quad 0.8329, \quad 0.8585, \quad 0.8261, \quad 0.8290$$

这里给出的是比值, 即本发生器的实际指标与理论极值之比, 它们明显优于 L'Ecuyer-Tezuka (1991) 提供的同类发生器的结果：

$$0.7824, \quad 0.8392, \quad 0.8417, \quad 0.8653, \quad 0.7770$$

对于我们的组合发生器所产生的长度为 10 亿个随机数的序列, 曾进行过较为充分的统计检验. 检验基于中国科学院计算中心 SASD 软件包的随机数检验模块 SUTEST (内含 12 类 27 种 61 个检验统计量), 并辅以 3 重 Kolmogorov-Smirnov 检验技术. 检验结果显示我们提供的组合发生器具有优良的统计品质.

参考文献

[1] 杨自强, 魏公毅 (2001), 常见随机数发生器的缺陷及组合随机数发生器的理论与实践, *数理统计与管理*, **20**(1), 45-51.

[2] 杨自强, 魏公毅 (2001), 综述：产生伪随机数的若干新方法, *数值计算与计算机应用*, **22**(3), 201-216.

[3] 魏公毅, 杨自强 (2001), 关于并行随机数发生器的若干算法, *数值计算与计算机应用*, **22(4)**, 311-320.

[4] L'Ecuyer, P. and S. Tezuka (1991) Structural Properties for Two Classes of Combined Random Number Generators, Mathematics of Computation, **57**(196), 735-746.

II. 拟随机数发生器使用说明

拟 (quasi) 随机数序列实质上是确定性的均匀序列，因此有些人更愿意称之为低偏差序列. 这种序列与经典蒙特卡罗方法中使用的伪 (pseudo) 随机数序列 (见第 I 部分) 不同. 使用拟随机数取代伪随机数的蒙特卡罗方法称为拟蒙特卡罗方法. 后者常用于更侧重均匀性而不太在乎其随机性的一类应用问题 (如高维积分问题和最优化问题等). 更详细的算法背景及其应用请见其后的参考文献 [1−6].

Sobol 序列 [2] 是常见的拟随机数序列. 本程序库提供的发生器使用 Gray 码技术 [3], 能快速地产生一个 M 维的 Sobol 序列，它由如下 **三个子程序** 组成：

```
subroutine       RNQSOBS(X, M)
subroutine       RNQSOBI(MAXM, INIT)
subroutine       RNQSOBK(NSKIP)
```

过程 RNQSOBS 将在实型数组 X(1:M) 内返回一个 M 维拟随机点，相继调用便可得到一个 M 维的拟随机数序列. 但在调用之前应依次调用另两个过程 RNQSOBI 和 RNQSOBK, 前者用于发生器的初始化，两参数由用户提供，其中 MAXM 表示其后使用中可能出现的维数 M 的最大值 (本程序规定 MAXM≤52†), INIT 与 Sobol 序列构造中的方向数初值选取有关，INIT 可以是区间 $[0, 2^{30}]$ 上的任一整数，常令 INIT=0, 这时本程序产生的 M 维 Sobol 序列的前 6 维结果与著名软件 Numerical Recipes[7] 相同. 此外，按低偏差序列的使用惯例，序列开头的 NSKIP 个点常被舍弃，这里的过程 RNQSOBK 可用于此目的 (这时常令 NSKIP=2^{12}).

在 **并行计算** 中，过程 RNQSOBK 还可为各处理器划分一段拟随机数序列. 例如处理器的编号为 MYID=0,1,2,3,$\cdots$, 每个处理器使用的序列长度为 N, 则各处理器内可安排如下几行程序：

```
      PARAMETER (M=9)   ! M-dimensional sequence to be generated
      REAL  X(M)        ! X stores an M-dimensional point of sequence
      N = 2**13         ! N is length used for every processor
      NSKIP = 2**12     ! discarding the first NSKIP points
      INIT = 0          ! initial values of direction number
      CALL RNQSOBI(M,INIT)
      CALL RNQSOBK(NSKIP + N*MYID)
      ...  ...  ...  ...  ...
      DO 10 I=1,N
        CALL RNQSOBS(X,M)  ! X returns an M-dimensional point
        ...  ...  ...  ...  ...
10    CONTINUE
      ...  ...  ...  ...
```

† 我们还开发了用于更高维数 (几百维，甚至过千维) 的程序，需要的用户请与我们联系.

参考文献

[1] 杨自强 魏公毅 (2003), 拟蒙特卡罗方法与低偏差序列, 研究报告, 中国科学院 计算数学与科学工程计算研究所.

[2] Sobol', I.M. (1967), On the Distribution of Points in a Cube and Approximate Evaluation of Integrals, *USSR Comput. Math. Math. Phys.*, **7**, 86-112.

[3] Antonov, I.A. and Saleev,V.M. (1979), An Economic Method of Computing LP_t-sequences, *USSR Comput. Math. Math. Phys.*, **19**, 252-256.

[4] Niederreiter, H. (1992), *Random Number Generation and Quasi-Monte Carlo methods*, CBMS-NSF Regional Conference Series in Applied Mathematics, No. 63, SIAM.

[5] Bratley, P., Fox, B.L. and Niederreiter, H. (1992), Implementation and Tests of Low-Discrepancy Sequences, *ACM Transactions on Modeling and Computer Simulation*, **2**, No.2, 195-213.

[6] Ninomiya, S. and Tezuka, S. (1996), Toward Real-time Pricing of Complex Financial Derivatives, *Applied Mathematical Finance*, **3**, 1-20.

[7] Press, W.H. *et. al.* (1992), *Numerical Recipes in Fortran: The Art of Scientific Computing*, 2nd. ed., Cambridge University Press, Cambridge.

III. 任意分布随机变量抽样程序使用说明

这部分介绍本程序库提供的一组能产生 **任意离散分布** 和 **任意连续分布** 随机变量的计算机抽样程序. 虽然程序的核心使用了多种复合抽样技术以保证其**通用性**, 但另一方面, 程序内又在初始化阶段安排了大量人工智能性的自动分析判断方法, 以减少由用户提供的信息, 所以这是一组 **易用型程序**. 对于离散分布, 用户只需输入状态数目 m 和取这 m 个状态的概率 p_1,p_2,$\cdots$,p_m; 对于连续分布, 只需给出随机变量的密度函数 f(x) 及与此函数有关的其它两个信息 (自变量 x 的定义区间 (a,b) 及分布是否对称). 实测表明, 本程序 **抽样速度高**, 甚至比许多针对单个具体分布专门设计的常规算法还要快 (见附录 B 和参考文献 [1]), 是通用、易用、品质和效率俱佳的程序. 程序既可用于串行计算, 也可以用于并行计算, 后者请参考 2.3 节.

按惯例, 程序只考虑一元分布情形. 均匀分布的抽样 (即伪随机数) 比较简单, 已有更好的方法与程序 (例如见本程序库第 I 部分 *伪随机数发生器使用说明*), 不建议使用本程序.

1. 基本函数及其参数

任意分布随机变量抽样程序的用户界面, 主要包括如下 **四个函数**:

```
function     RNGDINI1(m, p)
function     RNGDSAM1(iseed)
function     RNGCINI1(f, a, b, isymm, ab)
function     RNGCSAM1(iseed)
```

其中第 4 个字母为 D 的函数用于任意离散分布抽样, 而同位置字母为 C 的函数用于任意连续分布情形. 名字中的其后字符串为 SAM1 的函数用于抽样, 反复引用可得到一个抽样序列, 但此前应作准备工作, 即引用名字内含字符串 INI1 的函数对其初始化, 且产生任一种分布都只需引用一次初始化程序. **函数中的各个参数的含义** 是:

- m — 整型数, 离散分布中的状态数目, 本程序目前限定 m≤512.
- p — 实型数组 p(1:m), 由用户提供的 m 个离散状态概率 p_1,p_2,$\cdots$,p_m, 与离散变量值 x_1,x_2,$\cdots$,x_m 对应, 程序把后者设定为 1,2,$\cdots$,m. 从而省去值 x_i 的输入. 详见下节注 1 、注 2.
- iseed — 整型数组 iseed(1:2), 存放伪随机数种子, 并随着函数的引用被更新.
  可选 $0 < \text{iseed}(1) < 2146058219$, $0 < \text{iseed}(2) < 2145434063$
  任意分布随机变量的抽样都是在伪随机数基础上得到的, 所以需要伪随机数种子. 本程序所得的任意分布抽样也依赖此种子.
- f — 实型外部函数, 用户提供的抽样密度函数 f(x). 例子见下节.
- a, b — 实型数, (a, b) 构成上述密度函数 f(x) 的定义区间. 程序允许 a=$-\infty$, 且 / 或 b=$+\infty$, 此时约定用数 999999.0 代表 ∞.
- isymm — 整型数 isymm=1 表示密度函数 f(x) 是对称的, 否则为非对称.
  利用对称性可缩短初始化时间, 但不会提高本程序的抽样速度.
  如果用户实在不清楚密度函数 f(x) 是否对称, 宁可设 isymm=0.
- ab — 实型数组 ab(1:2), 返回参数, ab(1)=a', ab(2)=b', 通常 a'=a, b'=b, 当 (a, b) 为无穷时, (a', b') 将是库内程序自动调整后的有穷区间.

2. 程序调用例子

本节给出离散分布和连续分布抽样的两个示范性程序例子.

2.1 离散分布抽样程序例子

下述程序引用函数 RNGDSAM1 产生具有 4 个状态的离散分布的 10000 次抽样, 结果依照数组 P 中给出的概率随机地出现 J=1,2,3,4. 抽样前, 程序引用函数 RNGDINI1 对 RNGDSAM1 作初始化. 若其返回值 WK> 0, 意味着初始化顺利, 若 WK= -1, 意味着用户给出的概率之和远离 1.0, 计算无法进行.

```
      INTEGER  M, N, ISEED(2)
      REAL     P(4)
      DATA     P/0.8607, 0.1291, 0.0097, 0.0005/
      M = 4                ! number of classes in discrete distribution
      WK = RNGDINI1(M, P)  ! initialization
      ISEED(1) = 314159265 ! seeds for generator
      ISEED(2) = 271828
      N = 10000              ! sample size
      DO 20 K=1,N
        J = RNGDSAM1(ISEED)  ! sampling
        ...  ...  ...  ...
20    CONTINUE
      ...  ...  ...  ...
```

注 1 : 由上节参数 P 的含义, 数组元素 P(i) 所含的概率 p_i 与离散变量值 x_i 对应. 抽样函数 RNGDSAM1 的返回值 J 对应着离散变量值 x_J, 所以 J 只是变量值的下标. 但是为了简单, 库内程序又把 x_i 约定为 i, 即 $\mathrm{x}_i = i$, 以避免值 x_1,x_2,$\cdots$,x_m 的输入, 所以 J 又同时被看作是变量值. 但有些分布 (如泊松分布) 的值 $\mathrm{x}_1 = 0$, 且 $\mathrm{x}_i = i-1$, $i = 1, 2, \cdots$, 由于此时的下标与值本身不同, 使用时应注意区分. 例如在上面的程序中, 如果数组 P 中的 4 个元素依次对应着泊松分布变量取值 0,1,2,3 时的概率, 则抽样值 J 减 1 之后才是泊松分布变量的取值.

注 2 : 有些离散分布 (如泊松分布) 理论上可有无穷多个状态, 用户给出有限的状态数目 M 代替时, 应考虑到被截去的概率足够小. 我们建议, 此允许误差也应与计划中的抽样数目 N 挂钩 (计算经验表明 EPS=0.49/N 有较好的效果). 例如可有如下的计算泊松分布概率的用户程序 (其中 V 为泊松分布的参数)

```
      EPS = 0.49/N
      P(1) = EXE(-V)
      SUM = P(1)
      DO 10 I=1,100
        P(I+1) = P(I)*V/i
        IF (P(I+1) .LE. EPS) GO TO 15
        SUM = SUM + P(I+1)
10    CONTINUE
15    M = I+1
      P(M) = 1.0 - SUM
```

2.2 连续分布抽样程序例子

下述程序引用函数 RNGCSAM1 产生具有正态分布的 10000 次抽样, 此分布密度函数是对称的, 定义域是 $(-\infty, \infty)$. 抽样结果 CS 是实值 (库内程序经自动分析后, 已使用 $(-4.5, 4.5)$ 代替用户按理论给出的无穷区间, 这时的概率损失仅为 0.000006). 抽样前, 程序引用函数 RNGCINI1 对 RNGCSAM1 作初始化, 其返回值 WK 对一般用户没有用处.

```
      INTEGER  ISYMM, ISEED(2)
      REAL     FN, A, B, AB(2)
      EXTERNAL FN
      A = -999999.0         ! interval (a,b), a = -infinity
      B = 999999.0          ! b = infinity
      ISYMM = 1             ! FN is symmetrical density
      WK = RNGCINI1(FN, A, B, ISYMM, AB)     ! initialization
      ISEED(1) = 314159265   ! seeds for generator
      ISEED(2) = 271828
      N = 10000              ! sample size
      DO 20 K=1,N
        CS = RNGCSAM1(ISEED) ! sampling
        ...  ...  ...  ...
  20  CONTINUE
      ...  ...  ...  ...
      END
C---------   normal density function
      FUNCTION   FN(X)
      REAL       X
      FN = 0.3989423*EXP(-0.5*X*X)
      RETURN
      END
```

注 3: 当密度函数有 **奇异点** 时. 应由用户自己处理. 例如威布尔分布密度为

$$f(x) = \tfrac{1}{2} x^{-\frac{1}{2}} e^{-x^{\frac{1}{2}}}, \quad x \in (0, \infty),$$

其中 x=0 是奇异点. 这时用户可选一个适当小的正数 ε, 并把定义区间改为 (ε, ∞); 或者修改 $(0, \varepsilon)$ 内函数 $f(x)$ 的定义, 以避免计算 $x^{-\frac{1}{2}}$ 时上溢.

2.3 并行计算中的调用例子

我们的任意分布抽样程序也可用于并行计算情形. 因为抽样阶段使用的函数所依赖的种子 ISEED 实质上是伪随机数的种子 (见第 1 节参数表), 所以可参照 **伪随机数发生器使用说明**(第 2,3 节) 中的并行计算场合处理. 毕竟任意分布抽样远比产生伪随机数复杂. 在并行计算中, 任意分布抽样只适合 **分段方案**, 且只能使用伪随机数发生器中的子程序 RNPSKIP(J,DK,ISEED), 该子程序使发生器内的第 J 个 (J=1,2) 种子 ISEED(J) 从当前值快速地跳到间隔为 DK(双精度型) 的对应值, 可为各个处理器提供种子. DK 使用双精度形式是为了允许跳跃间隔超过整型数的上限 $2^{31} - 1$. 值得注意的是, 产生任意分布的一个抽样时, 需要多个伪随机数 (离散场合是 1 个, 连续场合有时可达 3, 5 个, 平均是 1.15 个), 所以确定跳跃间隔时应考虑此因素.

下面的程序用于并行计算中的任一处理器，MYID 是处理器编号，MYID=0,1, $\cdots$, N 是每个处理器拟使用的任意分布抽样序列长度，程序中以 1.25N 作为跳跃间隔 (即每个处理器可使用的伪随机数个数). 为突出重点和节省篇幅，程序中略去了某些变量的赋值 (如初始化函数的参数) 和用户应提供的外部函数等，但用户不难仿照 2.3 节的例子一一补上.

```
      WK = RNGCINI1(F, A, B, ISYMM, AB)     ! initialization
      ISEED(1) = 314159265   ! seeds for generator
      ISEED(2) = 271828
      N = 10000000           ! sample size
      NK = 1.25*N
      IF (MYID .GT. 0) THEN  ! MYID is processor number, MYID=0,1,2,...
        CALL RNPSKIP(1, DBLE(NK), ISEED(1))
        CALL RNPSKIP(2, DBLE(NK), ISEED(2))
      ENDIF
      DO 20 K=1,N
        CS = RNGCSAM1(ISEED) ! sampling
        ...  ...  ...  ...
20    CONTINUE
```

附录 A: 用于交叉产生多种随机分布抽样的几个别名子程序

在用户的程序中，使用本系统程序产生多种不同分布的随机变量抽样时，只要遵从 **无交叉** 原则，那么无论有多少种不同的分布，正文中提供的四个函数就已足够. 所谓无交叉是指程序对某种分布 (由参数确定) 初始化和产生需要的抽样序列都完成之后，才对另一种分布 (由另外的参数确定) 初始化和产生另一抽样序列. 只有如下场合允许直接的交叉使用：仅含两种分布，且一种是离散型，另一种是连续型. 除此以外，交叉使用时都要作技术处理，其中最为方便的是使用本系统另外提供的几个别名函数. 下面通过一个例子说明在一个程序内交叉使用多种分布随机变量抽样的方法.

例子: 计算三元函数 g(d,u,v) 的平均值：

$$Eg = \frac{1}{N}\sum_{k=1}^{N} g(d_k, u_k, v_k),$$

其中 $\{d_k\}$, $\{u_k\}$ 和 $\{v_k\}$ 是不同分布的三个随机变量 D,U,V 的抽样序列，其中 D 是离散分布， U, V 同属连续分布. 这时可有如下一段程序：

```
      WK = RNGDINI1(M, P)                    ! initialization for D
      WK = RNGCINI1(FU, AU, BU, ISYYMU, ABU)  ! initialization for U
      WK = RNGCINI2(FV, AV, BV, ISYMMV, ABV)  ! initialization for V
      SUM = 0.0
      DO 30 K=1,N
        D = RNGDSAM1(ISEED)          ! sampling from D
        U = RNGCSAM1(ISEED)          ! sampling from U
        V = RNGCSAM2(ISEED)          ! sampling from V
        SUM = SUM + G(D, U, V)
30    CONTINUE
      EG = SUM/N
```

上述程序中，不同的函数也可以采用不同的种子. 此外，为突出重点和节省篇幅，程序中略去了某些变量的赋值 (如样本大小，初始化函数的参数和伪随机数的种子等)，但用户不难仿照前节的例子一一补上. 要注意的是，这里的 (RNGCINI2, RNGCSAM2) 由系统提供，它们实际上是 (RNGCINI1, RNGCSAM1) 的拷贝，只是更改了名字最后的数字，故称 **别名函数**.

使用别名函数的原因可通过如下的分析得知：假设把 (RNGCINI2, RNGCSAM2) 改回 (RNGCINI1, RNGCSAM1), 这时第一次引用初始化函数 RNGCINI1 之后，其内部已保存了 U 的信息. 第二次引用同一个初始化函数 RNGCINI1 得到的是 V 的新信息，从而 U 的信息被 V 复盖. 于是其后两次引用的抽样函数 RNGCSAM1 实际上都只能返回 V 分布的抽样 (虽然因种子不同，抽样值会不同).

本库程序提供了用于离散型和连续型分布的各 3 组别名函数：

(RNGDINI2, RNGDSAM2),(RNGDINI3, RNGDSAM3),(RNGDINI4, RNGDSAM4)

(RNGCINI2, RNGCSAM2),(RNGCINI3, RNGCSAM3),(RNGCINI4, RNGCSAM4)

解决交叉引用问题也可以使用 **数组解决方案**. 详见如下一段程序：

```
      REAL  UU(10000), VV(10000)
      ...  ...  ...  ...
      N = 10000                              ! sample size
      WK = RNGDINI1(M, P)                    ! initialization for D
      WK = RNGCINI1(FU, AU, BU, ISYYMU, ABU)  ! initialization for U
      DO 10 K=1,N
10      UU(K) = RNGCSAM1(ISEED)              ! sampling from U
      WK = RNGCINI1(FV, AV, BV, ISYMMV, ABV)  ! initialization for V
      DO 20 K=1,N
20      VV(K) = RNGCSAM1(ISEED)              ! sampling from V
      SUM = 0.0
      DO 30 K=1,N
        D = RNGDSAM1(ISEED)                  ! sampling from D
        SUM = SUM + G(D, UU(K),VV(K))
30    CONTINUE
      EG = SUM/N
```

当样本量 N 很大时，可采用产生一段使用一段的办法缩减数组 UU 和 VV 的空间开销. 但这时，每段都要作一次初始化工作.

附录 B: 本库程序的抽样质量和效率

我们曾使用本库程序分别产生了二十余种离散或连续分布 (如泊松、二项、负二项、几何、超几何、对数、指数、正态、χ^2、t、F、威布尔、Γ、β、瑞利、对数正态、哥西、罗吉斯蒂、极值、拉普拉斯等分布) 的抽样各 10 万个，经计算样本均值、方差和理论值比较，并经样本频数和理论频数差异的 χ^2 检验，表明本库程序产生的各种离散或连续分布随机变量的抽样品质优异. 特别值得说明的是，我们所作的 χ^2 检验是十分细致的. 在所有连续分布中，用于检验的分组数目都多达 64 个，且 χ^2 值对应的

上侧概率大多数都在 50% 以上，而按惯例，此概率只需不小于 5% 便可称通过检验. 另一方面，对于理论上是无穷多状态的离散分布以及无穷区间上的连续分布，其尾部状态在我们的实际抽样中都得到了细致的反映.

表 B.1 中开列了几种分布在两个常见的不同类型机器上产生 1000 万个抽样的平均时间 (5 次 1000 万抽样的平均). 用于对比的 "常规方法" 来自常见的统计模拟著作，[3] 中有简明的介绍，对 "我们的方法" 有兴趣的读者可阅读附录 C. 计算中使用的编译器是 Compaq Visual Fortran (专业版 6.6B, 使用优化参数 /Ox /G5).

表 B.1 产生 1000 万个抽样的平均时间 (单位：秒)

分　布	AMD/K6-233		奔腾 P4/2.4GHz	
	我们的方法	常规方法	我们的方法	常规方法
伪 随 机 数	3.20	4.38	1.04	0.52
泊松分布 ($\lambda = 5$)	7.40	7.16	1.60	1.68
指数分布 ($\lambda = 1.0$)	8.12	8.18	1.87	1.73
指数分布 ($\lambda = 2.2$)	8.15	9.00	1.86	1.89
标准正态分布	8.25	24.82	1.90	5.56
$\chi^2(5)$ 分布	8.12	63.57	1.87	18.81
$t(5)$ 分布	8.26	84.85	1.90	22.40

上表中加列了产生伪随机数的时间，这是因为伪随机数是产生其他分布抽样的基础，而且计算简单，把产生其他分布的时间与之比较，容易给人留下深刻的印象. 产生伪随机数中的常规方法是指教科书上最常见也最简单的乘同余递推式的浮点双精度运算实现 (其序列周期是 $2**31-2$), 即

```
FUNCTION  RNG(IY)
INTEGER   IY
IY = DMOD(16807.0D0*IY, 2147483647.0D0)   ! 2**31-1=2147483647
RNG = IY*4.656613E-10                     ! namely IY/(2**31-1)
END
```

表中的 "我们的方法" 指前面 **I. 伪随机数发生器使用说明** 中的函数 RNPFR1, 它是两个乘同余递推式的组合 (其序列周期约为 $2**61$), 如果也用双精度实现，其时间比自然是上述简单发生器的两倍左右，但我们的程序实际上使用模分解技术，递推式只用整数运算，其比值在不同的计算机上，也象其他分布的不同抽样方法那样会有很大的差异. 但我们的方法在浮点双精度运算效率不高的计算机上优势明显：两个乘同余式递推的总时间比双精度运算实现单个乘同余式递推还快得多.

根据表 B.1 的结果，**用我们的方法产生任意分布随机变量的一次抽样时间平均是 1.6 至 2.6 个伪随机数的产生时间. 且离散分布所需时间约是连续情形的 90%**.

容易看出：在连续分布抽样中，**我们的方法所需的抽样时间与密度函数的复杂程度基本无关**, 而常规方法则可有 10 倍以上的差异. 后三个连续分布抽样时间的比值 (本程序 / 常规) 近似是：

正态分布： 1/3,　　$\chi^2(5)$ 分布： 1/8∼1/10,　　$t(5)$ 分布： 1/10∼1/12

这表明 **我们的方法在一般的连续分布抽样中优势极为明显**. 需要多说几句话的场合可能只有指数分布，因为这时公认的常规方法是 $-\frac{1}{\lambda}\ln R$, 即产生一个伪随机数 R 之后再取对数除以 λ 便妥，国内外未见有比此更简单的算法．但我们的上述测试结果令人欣喜不已，因为作为通用算法，我们的程序与公认的最佳专用算法所需的时间很接近，且当参数 $\lambda \neq 1.0$ 时，我们的方法还更快．即使当 $\lambda = 1.0$ 时 (这时公认算法还省去了一个浮点除法), 两种方法在不同机器上也互有胜负.

由泊松分布可以看出， **两种方法在离散分布抽样效率上接近**, 且在不同机器上也互有胜负.

总之，我们提供了一个用于任意分布随机变量抽样的个品质和效率俱佳的程序.

附录 C: 本库程序的算法简介

算法背景涉及较多的计算机抽样知识，一般用户不必细读．本库提供的程序中，离散和连续两类分布抽样由两个不同的程序块处理，每个程序块由一个多入口函数构成．离散型包括两个入口，对应着前面介绍过的两个函数 (RNGDINI1, RNGDSAM1), 依次完成初始化和抽样两阶段的任务．连续型也类似，对应着另两个函数 (RNGCINI1, RNGCSAM1). 初始化是一次性的工作，而抽样阶段则是千万次地被引用．为提高抽样效率，按惯例，只要能在初始化阶段做的事，即使代价高一些，也决不放在抽样阶段内.

离散型问题的初始化：

函数 RNGDINI1 需自动检查 m 个离散状态概率 p(1:m) 的和与 1.0 是否有太大的差异，并分析 m 个离散状态概率的分布特点，以选取一种最合适的算法 (本系统有三个算法备选，依次是罐子法 (urn), 别名法 (alias) 和直接法). 接着便是为选定的那个算法作进一步的准备工作 (如别名法中造出别名表，直接法中造出累积概率表 $(q(j) = \sum_{i=1}^{j} p(i),\ j = 1, 2, \cdots, m)$ 等．最后由库内程序中的 SAVE 语句保存有关的中间结果为抽样阶段的函数 RNGDSAM1 使用．此外 RNGDINI1 也返回选中的算法代号：1, 2 或 3 依次与前述三个算法对应．但当发现用户给出的状态概率之和远离 1.0 时，将返回值 -1 并停止计算.

连续型问题的初始化：

连续型问题比离散型问题的初始化工作更为复杂．它包括

(1) 判断密度函数的定义区间 (a,b). 如果 a=$-\infty$ 或 / 和 b=$+\infty$, 则程序通过密度函数的分析计算，自动找出一个有限区间来代替 (a,b). 例如正态分布，本程序自动找出 $(-4.5, 4.5)$ 代替 $(-\infty, \infty)$, 其概率损失仅为 0.000006.

(2) 细分区间 (a,b) 成 m 个子区间， m+1 个分点满足：

$$\mathrm{a} = x_1 < x_2 < x_3 < \cdots < x_m < x_{m+1} = \mathrm{b}.$$

这时程序通过计算密度函数 (包括利用由用户提供的是否对称信息) 自动确定合适的细分数目 m 及具体的分点 x_i 放入数组 x(1:m+1). 分点通常是非等距地配置，密度大的地方，分点也相对靠得近些，但配置方案也考虑到分布在小概率区 (尾部) 的形态能得到充分的反映.

(3) 找出每个子区间内密度函数曲线的最大值和最小值 (分别存入数组 SU(1:m) 和

SL(1:m)), 并计算密度函数曲线下的最大矩形面积, 其 m 个之和记为 p_a (为控制其后抽样阶段的计算误差积累, 上述的矩形面积实际上还要经过某种误差截断处理). 计算经验表明, 在大多数场合下, 常可令 m=256, 这时的 p_a 可达 0.95 以上.

最后, 如同离散型问题那样, 通过库内程序中的 SAVE 语句保存上述所有中间结果供抽样阶段的函数 RNGCSAM1 使用. 此外 RNGCINI1 也返回程序选定的 m 值.

连续型分布抽样阶段的算法简述:

在连续型分布抽样中, 库内程序使用组合抽样技术. 首先, 它以很大的概率 $p_a \simeq 0.95$ 作阶梯分布抽样. 这是因为前述 (3) 中密度函数曲线下的最大矩形可作为原来分布的初步近似. 另外, m 个矩形又可看作是含有 m 级的阶梯分布密度函数, 其抽样十分简单, 可基于罐子法处理, 每次抽样所需的时间约为产生两个伪随机数的时间. 这就保证了抽样的高效性. 另一方面, 为了提高抽样精度, 本程序还以小概率 $1 - p_a \simeq 0.05$ 从补偿分布中抽样. 补偿分布是指每个子区间内上述最大矩形之上, 密度函数曲线之下的小面积构成的分布. 在补偿分布中使用舍选法抽样 (上述 (3) 中的 SU(1:m) 和 SL(1:m) 值用于此抽样), 由于补偿分布的特点, 舍选法抽样效率通常可达 1/2, 如果密度函数被每个子区间内的两段折线代替, 抽样阶段就不再需要计算原来的密度函数值. 因此总的抽样效率很高.

参考文献

[1] 杨自强, 魏公毅 (2005), 任意分布随机变量的实用抽样方法和程序实现, *研究报告*, 中国科学院 计算数学与科学工程计算研究所.

[2] 杨自强, 魏公毅 (2003), 任意分布随机变量的实用抽样方法, *研究报告*, 中国科学院计算数学与科学工程计算研究所.

[3] 裴鹿成 (2000), 任意分布的自动抽样方法, *安徽大学学报, 自然科学版*, 2000, No. 3A, 1-6.