AMD RANDOM NUMBER GENERATOR Library

Version 2.0

DISCLAIMER

The information contained herein is for informational purposes only, and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale.

Trademarks

AMD, the AMD Arrow logo, and combinations thereof, are trademarks of Advanced Micro Devices, Inc.

NAG, NAGWare, and the NAG logo are registered trademarks of The Numerical Algorithms Group Ltd.

Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

Contents

1 Introduction

The AMD Random Number Generator (RNG) Library is a set of random number generators and statistical distribution functions tuned specifically for AMD64 platform processors. The routines are available via both FORTRAN 77 and C interfaces.

The AMD RNG Library is a comprehensive set of statistical distribution functions which are founded on various underlying uniform distribution generators *(base generators)* including Wichmann- Hill and an implementation of the Mersenne Twister. In addition there are hooks which allow you to supply your own preferred base generator if it is not already included in the library. All RNG functionality and interfaces are described in the following sections.

2 General Information

2.1 Library Package

AMD Random Number Generator library is available as a tar file on the AMD Developer website. It works on Linux operating systems. For details on installation please refer latest AOCL User Guide on https://developer.amd.com/amd-aocl/.

2.2 FORTRAN and C interfaces

All routines come with both FORTRAN and C interfaces. Here we document how a C programmer should call AMD RNG routines.

In C code that uses AMD RNG routines, be sure to include the header file <rng.h>, which contains function prototypes for all AMD RNG C interfaces. The header file also contains C prototypes for FORTRAN interfaces, thus the C programmer could call the FORTRAN interfaces from C, though there is little reason to do so.

C interfaces to the library routines differ from FORTRAN interfaces in the following major respects:

- The FORTRAN interface names are appended by an underscore
- The C interfaces contain no workspace arguments; all workspace memory is allocated internally.
- Scalar input arguments are passed by value in C interfaces. FORTRAN interfaces pass all arguments (except for character string *length* arguments that are normally hidden from FORTRAN programmers) by reference.
- Most arguments that are passed as character string pointers to FORTRAN interfaces are passed by value as single characters to C interfaces. The character string *length* arguments of FORTRAN interfaces are not required in the C interfaces.
- Unlike FORTRAN, C has no native *complex* data type. AMD RNG C routines which operate on complex data use the types *complex* and *doublecomplex* defined in <rng.h> for single and double precision computations respectively. Some of the programs in the examples directory make use of these types.

2.3 Example programs calling AMD Random Number Generator Library

The /examples subdirectory of the top AMD RNG Library installation directory contains example programs showing how to call the RNG routines, along with a GNUmakefile to build and run them. Examples of calling both FORTRAN and C interfaces are included.

2.4 Example programs demonstrating performance

The /examples/performance subdirectory of the top AMD RNG library installation directory contains several timing programs designed to show the performance of the library when running on your machine. Again, a GNUmakefile may be used to build and run them.

In order to run a multi-threaded program, use the openmp compiled version of the library. In the examples/performance directory a command such as

```
% make OMP_NUM_THREADS=5
```

will run the timing programs on P processors, where P = 1, 2, 4, 5; i.e., P equals an integer power of 2 and also equals *OMP_NUM_THREADS* if this value is not a power of 2. The results for a particular routine are concatenated into one file.

Setting *OMP_NUM_THREADS* in this way is not useful if you are not on an SMP machine or are not using an OpenMP version of the library. Neither is it useful to set *OMP_NUM_THREADS* to a value higher than the number of processors (or processor cores) on your machine. A way to find the number of processors (or cores) under linux is to examine the special file /proc/cpuinfo which has an entry for every core.

Not all routines in the library are SMP parallelized, so in this context the *OMP_NUM_THREADS* setting only applies to those examples. The other timing programs run on one thread regardless of the setting of *OMP_NUM_THREADS*.

Note that all results generated by timing programs will vary depending on the load on your machine at run time.

3 Random Number Generators

Within the context of this document, a base random number generator (BRNG) is a mathematical algorithm that, given an initial state, produces a sequence (or stream) of variates (or values) uniformly distributed over the semi-open interval (0,1]. The period of the BRNG is defined as the maximum number of values that can be generated before the sequence starts to repeat. The initial state of a BRNG is often called the seed.

Note that this definition means that the value 1.0 may be returned, but the value 0.0 will not.

A pseudo-random number generator (PRNG) is a BRNG that produces a stream of variates that are independent and statistically indistinguishable from a random sequence. A PRNG has several advantages over a true random number generator in that the generated sequence is repeatable, has known mathematical properties and is usually much quicker to generate. A quasi-random number generator (QRNG) is similar to a PRNG, however the variates generated are not statistically independent, rather they are designed to give a more even distribution in multidimensional space. Many books on statistics and computer science have good introductions to PRNGs and QRNGs, see for example Knuth [1] or Banks [2]. All of the BRNGs supplied in the AMD Random Number Generator library are PRNGs.

In addition to standard PRNGs some applications require cryptographically secure generators. A PRNG is said to be cryptographically secure if there is no polynomial-time algorithm which, on input of the first l bits of the output sequence can predict the $(l+1)$st bit of the sequence with probability significantly greater than 0.5. This is equivalent to saying there exists no polynomial-time algorithm that can correctly distinguish between an output sequence from the PRNG and a truly random sequence of the same length with probability significantly greater than 0.5 [3].

A distribution generator is a routine that takes variates generated from a BRNG and transforms them into variates from a specified distribution, for example the Gaussian (Normal) distribution.

The AMD Random Number Generator library contains five base generators, and twenty-three distribution generators. In addition users can supply a custom built generator as the base generator for all of the distribution generators.

The base generators were tested using the Big Crush, Small Crush and Pseudo Diehard test suites from the TestU01 software library [8].

3.1 Base Generators

The five base generators (BRNGs) supplied with the AMD RNG library are; the NAG basic generator [4], a series of Wichmann-Hill generators [5], the Mersenne Twister [6], L'Ecuyer's combined recursive generator MRG32k3a [7] and the Blum-Blum-Shub generator [3].

Some of the generators have been slightly modified from their usual form to make them consistent between themselves. For instance, the Wichmann-Hill generators in standard form may return exactly 0.0 but not exactly 1.0. In this library, we return $1.0 - x$ to convert the value x into the semi-open interval (0, 1] without affecting any other randomness properties. The original Mersenne Twister algorithm returns an exact zero about one time in a few billion; the AMD RNG implementation returns a tiny non-zero number as surrogate for zero.

If a single stream of variates is required it is recommended that the Mersenne Twister base generator is used. This generator combines speed with good statistical properties and an extremely long period. The NAG basic generator is another quick generator suitable for generating a single stream. However it has a shorter period than the Mersenne Twister and being a linear congruential generator, its statistical properties are not as good.

If 273 or fewer multiple streams, with a period of up to 2^{80} are required then it is recommended that the Wichmann-Hill generators are used. For more streams or multiple streams with a longer period it is recommended that the L'Ecuyer combined recursive generator is used in combination with the skip ahead routine. Generating multiple streams of variates by skipping ahead is generally quicker than generating the streams using the leap frog method.

The Blum-Blum-Shub generator should only be used if a cryptographically secure generator is required. This generator is extremely slow and has poor statistical properties when used as a base generator for any of the distributional generators.

3.1.1 Initialization of the Base Generators

A random number generator must be initialized before use. Three routines are supplied within the library for this purpose: DRANDINITIALIZE, DRANDINITIALIZEBBS and DRANDINITIALIZEUSER. Of these, DRANDINITIALIZE is used to initialize all of the supplied base generators, DRANDINITIALIZEBBS supplies an alternative interface to DRANDINITIALIZE for the Blum-Blum-Shub generator, and DRANDINITIALIZEUSER allows the user to register and initialize their own base generator.

Both double and single precision versions of all RNG routines are supplied. Double precision names are prefixed by DRAND, and single precision by SRAND. Note that if a generator has been initialized using the relevant double precision routine, then the double precision versions of the distribution generators must also be used, and vice versa. This even applies to generators with no double or single precision parameters; for example, a call of DRANDDISCRETEUNIFORM must be preceded by a call to one of the double precision initializers (typically DRANDINITIALIZE).

No utilities for saving, retrieving or copying the current state of a generator have been provided. All of the information on the current state of a generator (or stream, if multiple streams are being used) is stored in the integer array *STATE* and as such this array can be treated as any other integer array, allowing for easy copying, restoring etc.

The statistical properties of a sequence of random numbers are only guaranteed within the sequence, and not between sequences provided by the same generator. Therefore it is likely that repeated initialization will render the numbers obtained less, rather than more, independent. In most cases there should only be a single call to one of the initialization routines, per application, and this call must be made before any variates are generated. One example of where multiple initialization may be required is briefly touched upon in Section 3.2 [Multiple Streams].

In order to initialize the Blum-Blum-Shub generator a number of additional parameters, as well as an initial state (seed), are required. Although this generator can be initialized through the DRANDINITIALIZE routine it is recommended that the DRANDINITIALIZEBBS routine is used instead.

DRANDINITIALIZE / SRANDINITIALIZE

Initialize one of the five supplied base generators; NAG basic generator, Wichmann-Hill generator, Mersenne Twister, L'Ecuyer's combined recursive generator (MRG32k3a) or the Blum-Blum-Shub generator.

(Note that SRANDINITIALIZE is the single precision version of DRANDINITIALIZE. The argument lists of both routines are identical except that any double precision arguments of DRANDINITIALIZE are replaced in SRANDINITIALIZE by single precision arguments - type REAL in FORTRAN or type float in C).

DRANDINITIALIZE (*GENID,SUBID,SEED,LSEED,STATE, LSTATE,INFO*) [SUBROUTINE]

INTEGER GENID [Input]

On input: a numerical code indicating which of the five base generators to initialize.

- 1 = NAG basic generator (Section 3.1.3 [Basic NAG Generator]).
- 2 = Wichmann-Hill generator (Section 3.1.4 [Wichmann-Hill Generator],).
- 3 = Mersenne Twister (Section 3.1.5 [Mersenne Twister]).
- 4 = L'Ecuyer's Combined Recursive generator (Section 3.1.6 [L'Ecuyer's Combined Recursive Generator]).
- 5 = Blum-Blum-Shub generator (Section 3.1.7 [Blum-Blum-Shub Generator]).

Constraint: $1 \leq GENID \leq 5$.

INTEGER SUBID [Input]

On input: if *GENID* = 2, then *SUBID* indicates which of the 273 Wichmann- Hill generators to use. If *GENID* = 5 then *SUBID* indicates the number of bits to use (υ) from each of iteration of the Blum-Blum-Shub generator. In all other cases *SUBID* is not referenced.

Constraint: If *GENID* = 2 then $1 \leq SUBID \leq 273$.

INTEGER SEED(*LSEED*) [Input]

On input: if *GENID*= 5 , then *SEED* is a vector of initial values for the base generator. These values must be positive integers. The number of values required depends on the base generator being used. The NAG basic generator requires one initial value, the Wichmann-Hill generator requires four initial values, the L'Ecuyer combined recursive generator requires six initial values and the Mersenne Twister requires 624 initial values. If the number of seeds required by the chosen generator is > *LSEED* then *SEED*(1) is used to initialize the NAG basic generator. This is then used to generate all of the remaining seed values required. In general it is best not to set all the elements of *SEED* to anything too obvious, such as a single repeated value or a simple sequence. Using such a seed array may lead to several similar values being created in a row when the generator is subsequently called. This is particularly true for the Mersenne Twister generator.

In order to initialize the Blum-Blum-Shub generator two large prime values, p and q are required as well as an initial value s. As p, q and s can be of an arbitrary size, these values are expressed as a polynomial in B, where $B = 2^{24}$. For example, p can be factored into a polynomial of order l_p, with $p = p_1 + p_2 B + p_3 B^2 + \cdots + p_{l_p} B^{l_p - 1}$. The elements of *SEED* should then be set to the following:

- $SEED(1) = l_p$
- $SEED(2)$ to $SEED(l_p + 1) = p_1$ to p_{l_p}
- $SEED(l_p + 2) = l_q$
- $SEED(l_p + 3)$ to $SEED(l_p + l_q + 2) = q_1$ to q_{l_q}
- $SEED(l_p + l_q + 3) = l_s$
- $SEED(l_p + l_q + 4)$ to $SEED(l_p + l_q + l_s + 3) = s_1$ to s_{l_s}

Constraint: If $GENID = 5$ then $SEED(i) > 0$, $i = 1, 2,$. If $GENID = 5$ then *SEED* must take the values described above.

INTEGER LSEED [Input/Output]

On input: either the length of the seed vector, *SEED*, or a value $<= 0$.

On output: if $LSEED < 0$ on input, then *LSEED* is set to the number of initial values required by the selected generator, and the routine returns. Otherwise *LSEED* is left unchanged.

INTEGER STATE(*LSTATE*) [Output]

On output: the state vector required by all of the supplied distributional and base generators.

INTEGER LSTATE [Input/Output]

On input: either the length of the state vector, *STATE*, or a value $<= 0$.

On output: if $LSTATE <= 0$ on input, then *LSTATE* is set to the minimum length of the state vector *STATE* for the base generator chosen, and the routine returns. Otherwise *LSTATE* is left unchanged.

Constraint: $LSTATE <= 0$ or the minimum length for the chosen base generator, given by:

- $GENID = 1$: $LSTATE \geq 16$,
- $GENID = 2$: $LSTATE \geq 20$,
- $GENID = 3$: $LSTATE \geq 633$,
- $GENID = 4$: $LSTATE \geq 61$,
- $GENID = 5$: $LSTATE \geq l_p + l_q + l_s + 6$, where l_p, l_q and l_s are the order of the polynomials used to express the parameters p, q and s respectively.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. If $INFO = -i$ on exit, the i-th argument had an illegal value. If $INFO = 1$ on exit, then either, or both of *LSEED* and / or *LSTATE* have been set to the required length for vectors *SEED* and *STATE* respectively. Of the two variables *LSEED* and *LSTATE*, only those which had an input value <= 0 will have been set. The *STATE* vector will not have been initialized. If $INFO = 0$ then the state vector, *STATE*, has been successfully initialized.

Example:

```
C     Generate 100 values from the Beta distribution
      INTEGER LSTATE,N
      PARAMETER (LSTATE=16,N=100)
      INTEGER I,INFO,SEED(1),STATE(LSTATE)
      DOUBLE PRECISION A,B
      DOUBLE PRECISION X(N)
C     Set the seed
      SEED(1) = 1234

C     Read in the distributional parameters
      READ(5,*) A,B

C     Initialize the STATE vector
      CALL DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)
C     Generate N variates from the Beta distribution
      CALL DRANDBETA(N,A,B,STATE,X,INFO)

C     Print the results
      WRITE(6,*) (X(I),I=1,N)
```

DRANDINITIALIZEBBS / SRANDINITIALIZEBBS

Alternative initialization routine for the Blum-Blum-Shub generator. Unlike the other base generators supplied with the library, the Blum-Blum-Shub generator requires two additional parameters, p and q as well as an initial state, s. The parameters p, q and s can be of an arbitrary size. In order to avoid overflow these values are expressed as a polynomial in B, where $B = 2^{24}$. For example, p can be factored into a polynomial of order l_p, with $p = p_1 + p_2 B + p_3 B^2 + \cdots + p_{l_p} B^{l_p - 1}$, similarly $q = q_1 + q_2 B + q_3 B^2 + \cdots + q_{l_q} B^{l_q - 1}$ and $s = s_1 + s_2 B + s_3 B^2 + \cdots + s_{l_s} B^{l_s - 1}$.

(Note that SRANDINITIALIZEBBS is the single precision version of DRANDINITIALIZEBBS. The argument lists of both routines are identical except that any double precision arguments of DRANDINITIALIZEBBS are replaced in SRANDINITIALIZEBBS by single precision arguments - type REAL in FORTRAN or type float in C).

DRANDINITIALIZEBBS (*NBITS,LP,P,LQ,Q,LS,S,STATE,LSTATE, INFO*) [SUBROUTINE]

INTEGER NBITS [Input]

On input: the number of bits, v, to use from each iteration of the Blum- Blum-Shub generator. If $NBITS < 1$ then $NBITS = 1$. If $NBITS > 15$ then $NBITS = 15$.

INTEGER LP [Input]

On input: the order of the polynomial used to express p (l_p).
Constraint: $1 \le LP \le 25$.

INTEGER P(*LP*) [Input]

On input: the coefficients of the polynomial used to express p. $P(i) = p_i$, $i = 1$ to l_p.

Constraint: $0 \le P(i) < 2^{24}$

INTEGER LQ [Input]

On input: the order of the polynomial used to express q (l_q).
Constraint: $1 \le LQ \le 25$.

INTEGER Q(*LQ*) [Input]

On input: the coefficients of the polynomial used to express q. $Q(i) = q_i$, $i = 1$ to l_q.

Constraint: $0 \le Q(i) < 2^{24}$

INTEGER LS [Input]

On input: the order of the polynomial used to express s (l_s).
Constraint: $1 \le LS \le 25$.

INTEGER S(*LS*) [Input]

On input: the coefficients of the polynomial used to express s. $S(i) = s_i$, $i = 1$ to l_s.

Constraint: $0 \le S(i) < 2^{24}$

INTEGER STATE(*) [Output]

On output: the initial state for the Blum-Blum-Shub generator with parameters P,Q,S and $NBITS$.

INTEGER LSTATE [Input/Output]

On input: either the length of the state vector, *STATE*, or a value <= 0 .

On output: if *LSTATE* <= 0 on input, then *LSTATE* is set to the minimum length of the state vector *STATE* for the parameters chosen, and the routine returns. Otherwise *LSTATE* is left unchanged.

Constraint: $LSTATE \leq 0$ or $LSTATE \geq l_p + l_q + l_s + 6$

INTEGER INFO [Output]

On output: *INFO* is an error indicator. If *INFO* = $-i$ on exit, the i-th argument had an illegal value. If *INFO* = 1 on exit, then *LSTATE* has been set to the required length for the *STATE* vector. If *INFO* = 0 then the state vector, *STATE*, has been successfully initialized.

3.1.2 Calling the Base Generators

With the exception of the Blum-Blum-Shub generator, there are no interfaces for direct access to the base generators. All of the base generators return variates uniformly distributed over the semi-open interval (0, 1]. This functionality can be accessed using the uniform dis- tributional generator DRANDUNIFORM, with parameter $A = 0.0$ and parameter $B = 1.0$. The base generator used is, as usual, selected during the initialization process (see Section 3.1.1 [Initialization of the Base Generators]).

To directly access the Blum-Blum-Shub generator, use the routine DRANDBLUMBLUMSHUB.

DRANDBLUMBLUMSHUB / SRANDBLUMBLUMSHUB

Allows direct access to the bit stream generated by the Blum-Blum-Shub generator.

(Note that SRANDBLUMBLUMSHUB is the single precision version of DRANDBLUMBLUMSHUB. The argument lists of both routines are identical except that any double precision arguments of DRANDBLUMBLUMSHUB are replaced in SRANDBLUMBLUMSHUB by single precision arguments - type REAL in FORTRAN or type float in C).

DRANDBLUMBLUMSHUB (*N,STATE,X,INFO*) [SUBROUTINE]

INTEGER N [Input]

On input: number of variates required. The total number of bits generated is $24N$.

Constraint: $N \geq 0$.

INTEGER STATE(*) [Input/Output]

The *STATE* vector holds information on the state of the base generator being used and as such its minimum length varies. Prior to calling DRANDBLUMBLUMSHUB *STATE* must have been initialized. See Section 3.1.1 [Initialization of the Base Generators], for information on initialization of the *STATE* variable.

On input: the current state of the base generator.

On output: the updated state of the base generator.

INTEGER X(*N*) [Output]

On output: vector holding the bit stream. The least significant 24 bits of each of the $X(i)$ contain the bit stream as generated by the Blum-Blum-Shub generator. The least significant bit of $X(1)$ is the first bit generated, the second least significant bit of $X(1)$ is the second bit generated etc.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

3.1.3 Basic NAG Generator

The NAG basic generator is a linear congruential generator (LCG) and, like all LCGs, has the form:

$$x_i = a_1 x_{i-1} \bmod m_1,$$
$$u_i = \frac{x_i}{m_1},$$

where the u_i, $i = 1, 2, \cdots$ form the required sequence.

The NAG basic generator takes $a_1 = 13^{13}$ and $m_1 = 2^{59}$, which gives a period of approximately 2^{57}. This generator has been part of the NAG numerical library [4] since Mark 6 and as such has been widely used. It suffers from no known problems, other than those due to the lattice structure inherent in all LCGs, and, even though the period is relatively short compared to many of the newer generators, it is sufficiently large for many practical problems.

3.1.4 Wichmann-Hill Generator

The Wichmann-Hill [5] base generator uses a combination of four linear congruential generators (LCGs) and has the form:

$$
\begin{aligned}
w_i &= a_1 w_{i-1} \bmod m_1 \\
x_i &= a_2 x_{i-1} \bmod m_2 \\
y_i &= a_3 y_{i-1} \bmod m_3 \\
z_i &= a_4 z_{i-1} \bmod m_4 \\
u_i &= \left(\frac{w_i}{m_1} + \frac{x_i}{m_2} + \frac{y_i}{m_3} + \frac{z_i}{m_4}\right) \bmod 1,
\end{aligned}
$$

where the u_i, $i = 1, 2, \cdots$ form the required sequence. There are 273 sets of parameters, $\{a_i, m_i : i = 1, 2, 3, 4\}$, to choose from. These values have been selected so that the resulting generators are independent and have a period of approximately 2^{80} [5].

3.1.5 Mersenne Twister

The Mersenne Twister [6] is a twisted generalized feedback shift register generator. The algorithm is as follows:

- Set some arbitrary initial values $x_1, x_2, \cdots, x_r$, each consisting of w bits.
- Letting

$$
A = \begin{pmatrix} 0 & I_{w-1} \\ a_w & a_{w-1} \cdots a_1 \end{pmatrix},
$$

  where I_{w-1} is the $(w-1) \times (w-1)$ identity matrix and each of the a_i, $i = 1$ to w take a value of either 0 or 1 (i.e. they can be represented as bits). Define

$$
x_{i+r} = (x_{i+s} \oplus (x_i^{(w:(l+1))} | x_{i+1}^{(l:1)})A),
$$

  where $x_i^{(w:(l+1))} | x_{i+1}^{(l:1)}$ indicates the concatenation of the most significant (upper) $w - l$ bits of x_i and the least significant (lower) l bits of x_{i+1}.
- Perform the following operations sequentially:

$$
\begin{aligned}
z &= x_{i+r} \oplus (x_{i+r} \gg t_1) \\
z &= z \oplus ((z \ll t_2) \text{ AND } m_1) \\
z &= z \oplus ((z \ll t_3) \text{ AND } m_2) \\
z &= z \oplus (z \gg t_4) \\
u_{i+r} &= z/(2^w - 1),
\end{aligned}
$$

  where t_1, t_2, t_3 and t_4 are integers and m_1 and m_2 are bit-masks and "$>>t$" and "$<<t$" represent a t bit shift right and left respectively, $\oplus$ is bit-wise exclusively or (xor) operation and "AND" is a bit-wise and operation.

The u_{i+r} : $i = 1, 2, \ldots$ then form a pseudo-random sequence, with $u_i \in (0, 1)$, for all i. This implementation of the Mersenne Twister uses the following values for the algorithmic constants:

$$
\begin{aligned}
w &= 32\\
a &= 0x9908b0df\\
l &= 31\\
r &= 624\\
s &= 397\\
t_1 &= 11\\
t_2 &= 7\\
t_3 &= 15\\
t_4 &= 18\\
m_1 &= 0x9d2c5680\\
m_2 &= 0xefc60000
\end{aligned}
$$

where the notation $0xDD \cdots$ indicates the bit pattern of the integer whose hexadecimal representation is $DD \cdots$.

This algorithm has a period length of approximately $2^{19,937} - 1$ and has been shown to be uniformly distributed in 623 dimensions.

3.1.6 L'Ecuyer's Combined Recursive Generator

The base generator referred to as L'Ecuyer's combined recursive generator is referred to as MRG32k3a in [7] and combines two multiple recursive generators:

$$
\begin{aligned}
x_i &= a_{11}x_{i-1} + a_{12}x_{i-2} + a_{13}x_{i-3} \bmod m_1\\
y_i &= a_{21}y_{i-1} + a_{22}y_{i-2} + a_{23}y_{i-3} \bmod m_2\\
z_i &= x_i - y_i \bmod m_1\\
u_i &= \frac{z_i}{m_1},
\end{aligned}
$$

where the u_i, $i = 1, 2, \cdots$ form the required sequence and $a_{11} = 0$, $a_{12} = 1403580$, $a_{13} = -810728$, $m_1 = 2^{32} - 209$, $a_{21} = 527612$, $a_{22} = 0$, $a_{23} = -1370589$ and $m_2 = 2^{32} - 22853$.

Combining the two multiple recursive generators (MRG) results in sequences with better statistical properties in high dimensions and longer periods compared with those generated from a single MRG. The combined generator described above has a period length of approximately 2^{191}

3.1.7 Blum-Blum-Shub Generator

The Blum-Blum-Shub pseudo random number generator is cryptographically secure under the assumption that the quadratic residuosity problem is intractable [3]. The algorithm consists of the following:

- Generate two large and distinct primes, p and q, each congruent to 3 mod 4. Define $m = pq$.
- Select a seed s taking a value between 1 and $m - 1$, such that the greatest common divisor between s and m is 1.

- Let $x_0 = s^2 \ mod \ m$. For $I = 1, 2, \cdots$ generate:

$$x_i = x_{i-1}^2 \bmod m$$
$$z_i = v \text{ least significant bits of } x_i$$

  where $v \geq 1$.

- The bit-sequence $z_1, z_2, z_3, \cdots$ is then the output sequence used.

3.1.8 User Supplied Generators

All of the distributional generators described in Section 3.3 [Distribution Generators], require a base generator which returns a uniformly distributed value in the semi-open interval $(0, 1]$ and AMD RNG library includes several such generators (as detailed in Section 3.1). However, for greater flexibility, the library routines allow the user to register their own base generator function. This user-supplied generator then becomes the base generator for all of the distribution generators.

A user supplied generator comes in the form of two routines, one to initialize the generator and one to generate a set of uniformly distributed values in the semi-open interval $(0, 1]$. These two routines can be named anything, but are referred to as UINI for the initialization routine and UGEN for the generation routine in the following documentation.

In order to register a user supplied generator a call to DRANDINITIALIZEUSER must be made. Once registered the generator can be accessed and used in the same manner as the library supplied base generators. The specifications for DRANDINTIALIZEUSER, UINI and UGEN are given below. See the the example programs drandinitializeuser_example.f and drandinitializeuser_c_example.c to understand how to use these routines.

DRANDINITIALIZEUSER / SRANDINITIALIZEUSER

Registers a user supplied base generator so that it can be used with the AMD RNG distributional generators.

(Note that SRANDINITIALIZEUSER is the single precision version of DRANDINITIALIZEUSER. The argument lists of both routines are identical except that any double pre- cision arguments of DRANDINITIALIZEUSER are replaced in SRANDINITIALIZEUSER by single precision arguments – type REAL in FORTRAN or type float in C).

DRANDINITIALIZEUSER (*UINI,UGEN,GENID,SUBID,SEED,LSEED, STATE,LSTATE,INFO*) [SUBROUTINE]

SUBROUTINE UINI [Input]

On input: routine that will be used to initialize the user supplied generator, *UGEN*.

SUBROUTINE UGEN [Input]

On input: user supplied base generator.

INTEGER GENID [Input]

On input: parameter is passed directly to *UINI*. Its function therefore depends on that routine.

INTEGER SUBID [Input]

On input: parameter is passed directly to *UINI*. Its function therefore depends on that routine.

INTEGER SEED(*LSEED*) [Input]

On input: parameter is passed directly to *UINI*. Its function therefore depends on that routine.

INTEGER LSEED [Input/Output]

On input: length of the vector *SEED*. This parameter is passed directly to *UINI* and therefore its required value depends on that routine.
On output: whether *LSEED* changes will depend on *UINI*.

INTEGER STATE(*LSTATE*) [Output]

On output: the state vector required by all of the supplied distributional generators. The value of *STATE* returned by *UINI* has some housekeeping elements appended to the end before being returned by DRANDINITIALIZEUSER. See Section 3.1.8 [User Supplied Generators], for details about the form of *STATE*.

INTEGER LSTATE [Input/Output]

On input: length of the vector *STATE*. This parameter is passed directly to *UINI* and therefore its required value depends on that routine.
On output: whether *LSTATE* changes will depend on *UINI*. If *LSTATE* <= 0 then it is assumed that a request for the required length of *STATE* has been made. The value of *LSTATE* returned from *UINI* is therefore adjusted to allow for housekeeping elements to be added to the end of the *STATE* vector. This results in the value of *LSTATE* returned by DRANDINITIALIZEUSER being 3 larger than that returned by *UINI*.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. DRANDINITIALIZEUSER will return a value of 6 if the value of *LSTATE is* between 1 and 3. Otherwise *INFO is* passed directly back from *UINI*. It is recommended that the value of INFO returned by *UINI* is kept consistent with the rest of the AMD RNG library, that is if *INFO* = I on exit, the i-th argument had an illegal value. If *INFO* = 1 on exit, then either, or both of *LSEED* and / or *LSTATE* have been set to the required length for vectors *SEED* and *STATE* respectively and the *STATE* vector has not have been initialized. If *INFO* = 0 then the state vector, *STATE*, has been successfully initialized.

Example:

```
C     Generate 100 values from the Uniform distribution using
C     a user supplied base generator
      INTEGER LSTATE,N
      PARAMETER (LSTATE=16,N=100)
      INTEGER I,INFO,NSKIP,SEED(1),STATE(LSTATE)
      INTEGER X(N)
      DOUBLE PRECISION A,B

C     Set the seed
      SEED(1) = 1234

C     Set the distributional parameters
      A = 0.0D0
      B = 1.0D0

C     Initialize the base generator. Here RNGNB0GND is a user C
      supplied generator and RNGNB0INI its initializer
      CALL DRANDINITIALIZEUSER(RNGNB0INI,RNGNB0GND,1,0,SEED,
     *     LSEED,STATE,LSTATE,INFO)

C     Generate N variates from the Univariate distribution
      CALL DRANDUNIFORM(N,A,B,STATE,X,LDX,INFO)

C     Print the results
      WRITE(6,*) (X(I),I=1,N)
```

UINI

Specification for a user supplied initialization routine.

UINI (*GENID,SUBID,SEED,LSEED,STATE,LSTATE,INFO*) [SUBROUTINE]

INTEGER GENID [Input]

On input: the ID associated with the generator. It may be used for anything you like.

INTEGER SUBID [Input]

On input: the sub-ID associated with the generator. It may be used for anything you like.

INTEGER SEED(*LSEED*) [Input]

On input: an array containing the initial seed for your generator.

INTEGER LSEED [Input/Output]

On input: either the size of the *SEED* array, or a value < 1.

On output: if *LSEED* < 1 on entry, *LSEED* must be set to the required size of the *SEED* array. This allows a caller of UINI to query the required size.

INTEGER STATE(*LSTATE*) [Output]

On output: if *LSTATE* < 1 on entry, *STATE* should be unchanged.

Otherwise, *STATE* is a state vector holding internal details required by your generator. On exit from UINI, the array *STATE* must hold the following information:

STATE(1) = ESTATE, where ESTATE is your minimum allowed size of array *STATE*.

STATE(2) = MAGIC, where MAGIC is a magic number of your own choice. This can be used by your routine UGEN as a check that UINI has previously been called.

STATE(3) = GENID

STATE(4) = SUBID

STATE(5) ... STATE(ESTATE-1) = internal state values required by your generator routine UGEN; for example, the current value of your seed.

STATE(ESTATE) = MAGIC, i.e. the same value as STATE(2).

INTEGER LSTATE [Input/Output]

On input: either the size of the *STATE* array, or a value < 1.

On output: if *LSTATE* < 1 on entry, *LSTATE* should be set to the required size of the *STATE* array, i.e. the value ESTATE as described above. This allows the caller of UINI to query the required size.

Constraint: either *LSTATE* < 1 or $LSTATE \geq ESTATE$.

INTEGER INFO [Output]

On output: an error code, to be used in whatever way you wish; for example to flag an incorrect argument to UINI. If no error is encountered, UINI must set *INFO* to 0.

UGEN

Specification for a user supplied base generator.

UGEN (*N,STATE,X,INFO*) [SUBROUTINE]

INTEGER N [Input]

On input: the number of random numbers to be generated.

INTEGER STATE(*) [Input/Output]

On input: the internal state of your generator.

DOUBLE PRECISION X(*N*) [Output]

On output: the array of N uniform distributed random numbers, each in the semi-open interval (0.0, 1.0] – i.e. 1.0 is a legitimate return value, but 0.0 is not.

INTEGER INFO [Output]

On output: a flag which you can use to signal an error in the call of UGEN – for example, if UGEN is called without being initialized by UINI.

3.2 Multiple Streams

It is often advantageous to be able to generate variates from multiple, independent, streams. For example when running a simulation in parallel on several processors. There are four ways of generating multiple streams using the routines available in the AMD RNG library:

- (a) Using different seeds
- (b) Using different sequences
- I Block-splitting or skipping ahead
- (d) Leap frogging

The four methods are detailed in the following sections. Of the four, (a) should be avoided in most cases, (b) is only really of any practical use when using the Wichmann-Hill generator, and is then still limited to 273 streams. Both block-splitting and leap-frogging work using the sequence from a single generator, both guarantee that the different sequences will not overlap and both can be scaled to an arbitrary number of streams. Leap-frogging requires no *a-priori* knowledge about the number of variates being generated, whereas block-splitting requires the user to know (approximately) the maximum number of variates required from each stream. Block-splitting requires no *a-priori* information on the number of streams required. In contrast leap-frogging requires the user to know the maximum number of streams required, prior to generating the first value.

It is known that, dependent on the number of streams required, leap-frogging can lead to sequences with poor statistical properties, especially when applied to linear congruential generators (see Section 3.2.4 [Leap Frogging] for a brief explanation). In addition, for more complicated generators like a L'Ecuyer's multiple recursive generator leap-frogging can increase the time required to generate each variate compared to block-splitting. The additional time required by block-splitting occurs at the initialization stage, and not at the variate generation stage. Therefore in most instances block-splitting would be the preferred method for generating multiple sequences.

3.2.1 Using Different Seeds

A different sequence of variates can be generated from the same base generator by initializing the generator using a different set of seeds. Of the four methods for creating multiple streams described here, this is the least satisfactory. As mentioned in Section 3.1.1 [Initialization of the Base Generators], the statistical properties of the base generators are only guaranteed within sequences, not between sequences. For example, sequences generated from different starting points may overlap if the initial values are not far enough apart. The potential for overlapping sequences is reduced if the period of the generator being used is large. Although there is no guarantee of the independence of the sequences, due to its extremely large period, using the Mersenne Twister with random starting values is unlikely to lead to problems, especially if the number of sequences required is small. This is the only way in which multiple sequences can be generated with the AMD RNG library using the Mersenne Twister as the base generator.

If the statistical properties of different sequences must be provable then one of the other methods should be adopted.

3.2.2 Using Different Generators

Independent sequences of variates can be generated using different base generators for each sequence. For example, sequence 1 can be generated using the NAG basic generator, sequence 2 using the L'Ecuyer's Combined Recursive generator, sequence 3 using the Mersenne Twister. The Wichmann-Hill generator implemented in the library is in fact a series of 273 independent generators. The particular sub-generator being used can be selected using the *SUBID* variable (see [DRANDINITIALIZE], for details). Therefore, in total, 277 independent streams can be generated with each using an independent generator (273 Wichmann-Hill generators, and 4 additional base generators).

3.2.3 Skip Ahead

Independent sequences of variates can be generated from a single base generator through the use of block-splitting, or skipping-ahead. This method consists of splitting the sequence into k non-overlapping blocks, each of length n, where n is larger than the maximum number of variates required from any of the sequences. For example:

$$\underbrace{x_1, x_2, \cdots, x_n,}_{\text{block 1}} \quad \underbrace{x_{n+1}, x_{n+2}, \cdots, x_{2n},}_{\text{block 2}} \quad \underbrace{x_{2n+1}, x_{2n+2}, \cdots, x_{3n},}_{\text{block 3}} \text{ etc}$$

where $x_1, x_2, \ldots$ is the sequence produced by the generator of interest. Each of the k blocks provide an independent sequence.

The block splitting algorithm therefore requires the sequence to be advanced a large number of places. Due to their form this can be done efficiently for linear congruential generators and multiple congruential generators. The Amd RNG library provides block-splitting for the NAG Basic generator, the Wichmann-Hill generators and L'Ecuyer's Combined Recursive generator.

DRANDSKIPAHEAD / SRANDSKIPAHEAD

Advance a generator N places.

(Note that SRANDSKIPAHEAD is the single precision version of DRANDSKIPA- HEAD. The argument lists of both routines are identical except that any double precision arguments of DRANDSKIPAHEAD are replaced in SRANDSKIPAHEAD by single precision arguments – type REAL in FORTRAN or type float in C).

DRANDSKIPAHEAD (*N,STATE,INFO*) [SUBROUTINE]

INTEGER N [Input]

On input: number of places to skip ahead.
Constraint: $N \geq 0$.

INTEGER STATE(*) [Input/Output]

The *STATE* vector holds information on the state of the base generator being used and as such its minimum length varies. Prior to calling DRANDSKIPAHEAD *STATE* must have been initialized. See Section 3.1.1 [Initialization of the Base Generators], for information on initialization of the *STATE* variable. On input: the current state of the base generator.
On output: The *STATE* vector for a generator that has been advanced N places.
Constraint: The *STATE* vector must be for either the NAG basic, Wichmann-Hill or L'Ecuyer Combined Recursive base generators.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = $-$I on exit, the i-th argument had an illegal value.

Example:

```
C      Generate 3 * 100 values from the Uniform distribution
C      Multiple streams generated using the Skip Ahead method
       INTEGER LSTATE,N
       PARAMETER (LSTATE=16,N=100)
       INTEGER I,INFO,NSKIP
       INTEGER SEED(1),STATE1(LSTATE),STATE2(LSTATE),STATE3(LSTATE)
       DOUBLE PRECISION X1(N),X2(N),X3(N)
       DOUBLE PRECISION A,B

C      Set the seed
       SEED(1) = 1234

C      Set the distributional parameters
       A = 0.0D0
       B = 1.0D0

C      Initialize the STATE1 vector
       CALL DRANDINITIALIZE(1,1,SEED,1,STATE1,LSTATE,INFO)

C      Copy the STATE1 vector into other state vectors
       DO 20 I = 1,LSTATE
         STATE2(I) = STATE1(I)
         STATE3(I) = STATE1(I)
20     CONTINUE

C      Calculate how many places we want to skip, this
C      should be >> than the number of variates we
C      wish to generate from each stream
       NSKIP = N * N

C      Advance each stream, first does not need changing
       CALL DRANDSKIPAHEAD(NSKIP,STATE2,INFO)
       CALL DRANDSKIPAHEAD(2*NSKIP,STATE3,INFO)

C      Generate 3 sets of N variates from the Univariate distribution
       CALL DRANDUNIFORM(N,A,B,STATE1,X1,LDX,INFO)
       CALL DRANDUNIFORM(N,A,B,STATE2,X2,LDX,INFO)
       CALL DRANDUNIFORM(N,A,B,STATE3,X3,LDX,INFO)

C      Print the results
       DO 40 I = 1,N
         WRITE(6,*) X1(I),X2(I),X3(I)
40     CONTINUE
```

■

3.2.4 Leap Frogging

Independent sequences of variates can be generated from a single base generator through the use of leap-frogging. This method involves splitting the sequence from a single generator into k disjoint subsequences. For example:

$$
\begin{aligned}
&\text{Subsequence 1} : x_1, x_{k+1}, x_{2k+1}, \cdots \\
&\text{Subsequence 2} : x_2, x_{k+2}, x_{2k+2}, \cdots \\
&\qquad\qquad\ddots \\
&\text{Subsequence k} : x_k, x_{2k}, x_{3k}, \cdots
\end{aligned}
$$

each subsequence is then provides an independent stream.

The leap-frog algorithm therefore requires the generation of every kth variate of a sequence. Due to their form this can be done efficiently for linear congruential generators and multiple congruential generators. The library provides leap-frogging for the NAG Basic generator, the Wichmann-Hill generators and L'Ecuyer's Combined Recursive generator.

As an illustrative example, a brief description of the algebra behind the implementation of the leap-frog algorithm (and block-splitting algorithm) for a linear congruential generator (LCG) will be given. A linear congruential generator has the form $x_{i+1} = a_1 x_i \bmod m_1$. The recursive nature of a LCG means that

$$
\begin{aligned}
x_{i+v} &= a_1 x_{i+v-1} \bmod m_1 \\
&= a_1 (a_1 x_{i+v-2} \bmod m_1) \bmod m_1 \\
&= a_1^2 x_{i+v-2} \bmod m_1 \\
&= a_1^v x_i \bmod m_1
\end{aligned}
$$

The sequence can be quickly advanced v places by multiplying the current state (x_i) by $a_1^v \bmod m_1$, hence allowing block-splitting. Leap-frogging is implemented by using a_1^k, where k is the number of streams required, in place of a_1 in the standard LCG recursive formula.

In a linear congruential generator the multiplier a_1 is constructed so that the generator has good statistical properties in, for example, the spectral test. When using leap-frogging to construct multiple streams this multiplier is replaced with a_1^k, and there is no guarantee that this new multiplier will have suitable properties especially as the value of k depends on the number of streams required and so is likely to change depending on the application. This problem can be 26mphasized by the lattice structure of LCGs.

Note that, due to rounding, a sequence generated using leap-frogging and a sequence constructed by taking every kth value from a set of variates generated without leap-frogging may differ slightly. These differences should only affect the least significant digit.

DRANDLEAPFROG / SRANDLEAPFROG

Amend a generator so that it will generate every Kth value.

(Note that SRANDLEAPFROG is the single precision version of DRANDLEAPFROG. The argument lists of both routines are identical except that any double precision arguments of DRANDLEAPFROG are replaced in SRANDLEAPFROG by single precision arguments - type REAL in FORTRAN or type float in C).

DRANDLEAPFROG (*N,K,STATE,INFO*) [SUBROUTINE]

INTEGER N [Input]

On input: total number of streams being used.
Constraint: $N > 0$.

INTEGER K [Input]

On input: number of the current stream
Constraint: $0 < K \leq N$.

INTEGER STATE(*) [Input/Output]

The *STATE* vector holds information on the state of the base generator being used and as such its minimum length varies. Prior to calling DRANDLEAPFROG *STATE* must have been initialized. See Section 3.1.1 [Initialization of the Base Generators], for information on initialization of the *STATE* variable. On input: the current state of the base generator.
On output: The *STATE* vector for a generator that has been advanced K 1 places and will return every Nth value.
Constraint: The *STATE* array must be for either the NAG basic, Wichmann-Hill or L'Ecuyer Combined Recursive base generators.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0.
If $INFO = -I$ on exit, the i-th argument had an illegal value.

Example:

```
C       Generate 3 * 100 values from the  Uniform distribution
C       Multiple streams generated using the Leap Frog method
        INTEGER LSTATE,N
        PARAMETER (LSTATE=16,N=100)
        INTEGER  I,INFO
        INTEGER SEED(1),STATE1(LSTATE),STATE2(LSTATE),STATE3(LSTATE)
        DOUBLE  PRECISION  X1(N),X2(N),X3(N)
        DOUBLE  PRECISION A,B

C       Set   the   seed
        SEED(1)  = 1234

C       Set   the   distributional   parameters
        A  =  0.0D0
        B  =  1.0D0

C       Initialize  the  STATE1 vector
        CALL   DRANDINITIALIZE(1,1,SEED,1,STATE1,LSTATE,INFO)

C       Copy  the  STATE1  vector  into  other state  vectors
        DO  20  I = 1,LSTATE
          STATE2(I) = STATE1(I)
          STATE3(I)   = STATE1(I)
20      CONTINUE

C       Update   each   stream   so   they   generate   every   3rd   value
        CALL  DRANDLEAPFROG(3,1,STATE1,INFO)
        CALL DRANDLEAPFROG(3,2,STATE2,INFO)
        CALL  DRANDLEAPFROG(3,3,STATE3,INFO)

C       Generate   3   sets   of   N   variates   from   the Univariate   distribution
        CALL   DRANDUNIFORM(N,A,B,STATE1,X1,LDX,INFO)
        CALL DRANDUNIFORM(N,A,B,STATE2,X2,LDX,INFO)
        CALL  DRANDUNIFORM(N,A,B,STATE3,X3,LDX,INFO)

C       Print  the  results
        DO  40  I  = 1,N
          WRITE(6,*) X1(I),X2(I),X3(I)
40      CONTINUE
```

3.3 Distribution Generators

3.3.1 Continuous Univariate Distributions

DRANDBETA/SRANDBETA

Generates a vector of random variates from a beta distribution with probability density function, $f(X)$, where:

$$f(X) = \frac{\Gamma(A+B)}{\Gamma(A)\Gamma(B)} X^{A-1}(1-X)^{B-1}$$

if $0 \leq X \leq 1$ and $A, B > 0.0$, otherwise $f(X) = 0$.

(Note that SRANDBETA is the single precision version of DRANDBETA. The argument lists of both routines are identical except that any double precision arguments of DRAND- BETA are replaced in SRANDBETA by single precision arguments - type REAL in FOR- TRAN or type float in C).

DRANDBETA (*N,A,B,STATE,X,INFO*) [SUBROUTINE]

INTEGER N [Input]

On input: number of variates required.
Constraint: $N \geq 0$.

DOUBLE PRECISION A [Input]

On input: first parameter for the distribution.
Constraint: $A > 0$.

DOUBLE PRECISION B [Input]

On input: second parameter for the distribution.
Constraint: $B > 0$.

INTEGER STATE(*) [Input/Output]

The *STATE* vector holds information on the state of the base generator being used and as such its minimum length varies. Prior to calling DRANDBETA *STATE* must have been initialized. See Section 3.1.1 [Initialization of the Base Generators], for information on initialization of the *STATE* variable. On input: the current state of the base generator.
On output: the updated state of the base generator.

DOUBLE PRECISION X(*N*) [Output]

On output: vector of variates from the specified distribution.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If $INFO = -I$ on exit, the i-th argument had an illegal value.

Example:

```
C     Generate 100 values from the Beta distribution
      INTEGER LSTATE,N
      PARAMETER (LSTATE=16,N=100)
      INTEGER I,INFO,SEED(1),STATE(LSTATE)
      DOUBLE PRECISION A,B
      DOUBLE PRECISION X(N)
C     Set the seed
      SEED(1) = 1234

C     Read in the distributional parameters
      READ(5,*) A,B

C     Initialize the STATE vector
      CALL DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)
C     Generate N variates from the Beta distribution
      CALL DRANDBETA(N,A,B,STATE,X,INFO)

C     Print the results
      WRITE(6,*) (X(I),I=1,N)
```

DRANDCAUCHY / SRANDCAUCHY

Generates a vector of random variates from a Cauchy distribution with probability density function, $f(X)$, where:

$$f(X) = \frac{1}{\pi B(1 + (\frac{X-A}{B})^2)}$$

(Note that SRANDCAUCHY is the single precision version of DRANDCAUCHY. The argument lists of both routines are identical except that any double precision arguments of DRANDCAUCHY are replaced in SRANDCAUCHY by single precision arguments - type REAL in FORTRAN or type float in C).

DRANDCAUCHY (*N,A,B,STATE,X,INFO*) [SUBROUTINE]

INTEGER N [Input]

On input: number of variates required.
Constraint: $N \geq 0$.

DOUBLE PRECISION A [Input]

On input: median of the distribution.

DOUBLE PRECISION B [Input]

On input: semi-quartile range of the distribution.
Constraint: $B \geq 0$.

INTEGER STATE(*) [Input/Output]

The *STATE* vector holds information on the state of the base generator being used and as such its minimum length varies. Prior to calling DRANDCAUCHY *STATE* must have been initialized. See Section 3.1.1 [Initialization of the Base Generators], for information on initialization of the *STATE* variable. On input: the current state of the base generator.
On output: the updated state of the base generator.

DOUBLE PRECISION X(*N*) [Output]

On output: vector of variates from the specified distribution.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = $-$I on exit, the i-th argument had an illegal value.

Example:

```
C       Generate 100 values from the Cauchy distribution
        INTEGER LSTATE,N
        PARAMETER (LSTATE=16,N=100)
        INTEGER I,INFO,SEED(1),STATE(LSTATE)
        DOUBLE PRECISION A,B
        DOUBLE PRECISION X(N)
C       Set the seed
        SEED(1) = 1234

C       Read in the distributional parameters
        READ(5,*) A,B

C       Initialize the STATE vector
        CALL DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)
C       Generate N variates from the Cauchy distribution
        CALL DRANDCAUCHY(N,A,B,STATE,X,INFO)

C       Print the results
        WRITE(6,*) (X(I),I=1,N)
```

DRANDCHISQUARED / SRANDCHISQUARED

Generates a vector of random variates from a χ^2 distribution with probability density function, $f(X)$, where:

$$f(X) = \frac{X^{\frac{\nu}{2}-1}e^{-\frac{X}{2}}}{2^{\frac{\nu}{2}}(\frac{\nu}{2}-1)!},$$

if $X > 0$, otherwise $f(X) = 0$. Here ν is the degrees of freedom, DF.

(Note that SRANDCHISQUARED is the single precision version of DRANDCHISQUARED. The argument lists of both routines are identical except that any double precision arguments of DRANDCHISQUARED are replaced in SRANDCHISQUARED by single precision arguments – type REAL in FORTRAN or type float in C).

DRANDCHISQUARED (*N*,*DF*,*STATE*,*X*,*INFO*) [SUBROUTINE]

INTEGER N [Input]

On input: number of variates required.
Constraint: $N \geq 0$.

INTEGER DF [Input]

On input: degrees of freedom of the distribution.
Constraint: $DF > 0$.

INTEGER STATE(*) [Input/Output]

The *STATE* vector holds information on the state of the base generator being used and as such its minimum length varies. Prior to calling DRANDCHISQUARED *STATE* must have been initialized. See Section 3.1.1 [Initialization of the Base Generators], for information on initialization of the *STATE* variable. On input: the current state of the base generator.
On output: the updated state of the base generator.

DOUBLE PRECISION X(*N*) [Output]

On output: vector of variates from the specified distribution.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −I on exit, the i-th argument had an illegal value.

Example:

```
C       Generate 100 values from the Chi-squared distribution
        INTEGER LSTATE,N
        PARAMETER  (LSTATE=16,N=100)
        INTEGER I,INFO,SEED(1),STATE(LSTATE)
        INTEGER DF
        DOUBLE  PRECISION X(N)
C       Set the seed
        SEED(1)  = 1234

C       Read in the distributional parameters
        READ(5,*) DF

C       Initialize the STATE vector
        CALL  DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)
C       Generate N variates from the Chi-squared distribution
        CALL    DRANDCHISQUARED(N,DF,STATE,X,INFO)

C       Print the results
        WRITE(6,*) (X(I),I=1,N)
```

DRANDEXPONENTIAL / SRANDEXPONENTIAL

Generates a vector of random variates from an exponential distribution with probability density function, $f(X)$, where

$$f(X) = \frac{e^{-\frac{X}{A}}}{A}$$

if $X > 0$, otherwise $f(X) = 0$.

(Note that SRANDEXPONENTIAL is the single precision version of DRANDEXPONENTIAL. The argument lists of both routines are identical except that any double precision arguments of DRANDEXPONENTIAL are replaced in SRANDEXPONENTIAL by single precision arguments – type REAL in FORTRAN or type float in C).

DRANDEXPONENTIAL (*N*,*A*,*STATE*,*X*,*INFO*) [SUBROUTINE]

INTEGER N [Input]

On input: number of variates required.
Constraint: $N \geq 0$.

DOUBLE PRECISION A [Input]

On input: exponential parameter.
Constraint: $A \geq 0$.

INTEGER STATE(*) [Input/Output]

The *STATE* vector holds information on the state of the base generator being used and as such its minimum length varies. Prior to calling DRANDEXPONENTIAL *STATE* must have been initialized. See Section 3.1.1 [Initialization of the Base Generators], for information on initialization of the *STATE* variable. On input: the current state of the base generator.
On output: the updated state of the base generator.

DOUBLE PRECISION X(*N*) [Output]

On output: vector of variates from the specified distribution.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0.
If *INFO* = −I on exit, the i-th argument had an illegal value.

Example:

```
C       Generate 100 values from the Exponential distribution
        INTEGER LSTATE,N
        PARAMETER (LSTATE=16,N=100)
        INTEGER I,INFO,SEED(1),STATE(LSTATE)
        DOUBLE PRECISION A
        DOUBLE PRECISION X(N)
C       Set the seed
        SEED(1) = 1234

C       Read in the distributional parameters
        READ(5,*) A

C       Initialize the STATE vector
        CALL DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)
C       Generate N variates from the Exponential distribution
        CALL DRANDEXPONENTIAL(N,A,STATE,X,INFO)

C       Print the results
        WRITE(6,*) (X(I),I=1,N)
```

DRANDF / SRANDF

Generates a vector of random variates from an F distribution, also called the Fisher's variance ratio distribution, with probability density function, $f(X)$, where:

$$f(X) = \frac{(\frac{\mu+\nu-2}{2})!X^{\frac{\mu}{2}-1}\mu^{\frac{\mu}{2}}}{(\frac{\mu}{2}-1)!(\frac{\nu}{2}-1)!(1+\frac{\mu X}{\nu})^{\frac{\mu+\nu}{2}}\nu^{\frac{\mu}{2}}},$$

if $X > 0$, otherwise $f(X) = 0$. Here μ is the first degrees of freedom, ($DF1$) and ν is the second degrees of freedom, ($DF2$).

(Note that SRANDF is the single precision version of DRANDF. The argument lists of both routines are identical except that any double precision arguments of DRANDF are replaced in SRANDF by single precision arguments – type REAL in FORTRAN or type float in C).

DRANDF (*N,DF1,DF2,STATE,X,INFO*) [SUBROUTINE]

INTEGER N [Input]

On input: number of variates required.
Constraint: $N \geq 0$.

INTEGER DF1 [Input]

On input: first degrees of freedom.
Constraint: $DF1 \geq 0$.

INTEGER DF2 [Input]

On input: second degrees of freedom.
Constraint: $DF2 \geq 0$.

INTEGER STATE(*) [Input/Output]

The *STATE* vector holds information on the state of the base generator being used and as such its minimum length varies. Prior to calling DRANDF *STATE* must have been initialized. See Section 3.1.1 [Initialization of the Base Gener- ators], for information on initialization of the *STATE* variable.
On input: the current state of the base generator.
On output: the updated state of the base generator.

DOUBLE PRECISION X(*N*) [Output]

On output: vector of variates from the specified distribution.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0.
If $INFO = -I$ on exit, the i-th argument had an illegal value.

Example:

```
C     Generate 100 values from the F distribution
      INTEGER LSTATE,N
      PARAMETER (LSTATE=16,N=100)
      INTEGER I,INFO,SEED(1),STATE(LSTATE)
      INTEGER DF1,DF2
      DOUBLE PRECISION X(N)
C     Set the seed
      SEED(1) = 1234

C     Read in the distributional parameters
      READ(5,*) DF1,DF2

C     Initialize the STATE vector
      CALL DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)
C     Generate N variates from the F distribution
      CALL DRANDF(N,DF1,DF2,STATE,X,INFO)

C     Print the results
      WRITE(6,*) (X(I),I=1,N)
```

DRANDGAMMA / SRANDGAMMA

Generates a vector of random variates from a Gamma distribution with probability density function, $f(X)$, where:

$$f(X) = \frac{X^{A-1} e^{-\frac{X}{B}}}{B^A \Gamma(A)},$$

if $X \geq 0$ and A, $B > 0.0$, otherwise $f(X) = 0$.

(Note that SRANDGAMMA is the single precision version of DRANDGAMMA. The argument lists of both routines are identical except that any double precision arguments of DRANDGAMMA are replaced in SRANDGAMMA by single precision arguments - type REAL in FORTRAN or type float in C).

DRANDGAMMA (*N,A,B,STATE,X,INFO*) [SUBROUTINE]

INTEGER N [Input]

On input: number of variates required.
Constraint: $N \geq 0$.

DOUBLE PRECISION A [Input]

On input: first parameter of the distribution.
Constraint: $A > 0$.

DOUBLE PRECISION B [Input]

On input: second parameter of the distribution.
Constraint: $B > 0$.

INTEGER STATE(*) [Input/Output]

The *STATE* vector holds information on the state of the base generator being used and as such its minimum length varies. Prior to calling DRANDGAMMA *STATE* must have been initialized. See Section 3.1.1 [Initialization of the Base Generators], for information on initialization of the *STATE* variable. On input: the current state of the base generator.
On output: the updated state of the base generator.

DOUBLE PRECISION X(*N*) [Output]

On output: vector of variates from the specified distribution.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −I on exit, the i-th argument had an illegal value.

Example:

```
C     Generate 100 values from the Gamma distribution
      INTEGER LSTATE,N
      PARAMETER (LSTATE=16,N=100)
      INTEGER I,INFO,SEED(1),STATE(LSTATE)
      DOUBLE PRECISION A,B
      DOUBLE PRECISION X(N)
C     Set the seed
      SEED(1) = 1234

C     Read in the distributional parameters
      READ(5,*) A,B

C     Initialize the STATE vector
      CALL DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)
C     Generate N variates from the Gamma distribution
      CALL DRANDGAMMA(N,A,B,STATE,X,INFO)

C     Print the results
      WRITE(6,*) (X(I),I=1,N)
```

DRANDGAUSSIAN / DRANDGAUSSIAN

Generates a vector of random variates from a Gaussian distribution with probability density function, $f(X)$, where:

$$f(X) = \frac{e^{-\frac{(X-\mu)^2}{2\sigma^2}}}{\sigma\sqrt{2\pi}}.$$

Here μ is the mean, (XMU) and σ^2 the variance, (VAR) of the distribution.

(Note that SRANDGAUSSIAN is the single precision version of DRANDGAUSSIAN. The argument lists of both routines are identical except that any double precision arguments of DRANDGAUSSIAN are replaced in SRANDGAUSSIAN by single precision arguments - type REAL in FORTRAN or type float in C).

DRANDGAUSSIAN (*N,XMU,VAR,STATE,X,INFO*) [SUBROUTINE]

INTEGER N [Input]
: On input: number of variates required.
: Constraint: $N \geq 0$.

DOUBLE PRECISION XMU [Input]
: On input: mean of the distribution.

DOUBLE PRECISION VAR [Input]
: On input: variance of the distribution.
: Constraint: $VAR \geq 0$.

INTEGER STATE(*) [Input/Output]
: The *STATE* vector holds information on the state of the base generator being used and as such its minimum length varies. Prior to calling DRANDGAUSSIAN *STATE* must have been initialized. See Section 3.1.1 [Initialization of the Base Generators], for information on initialization of the *STATE* variable. On input: the current state of the base generator.
: On output: the updated state of the base generator.

DOUBLE PRECISION X(*N*) [Output]
: On output: vector of variates from the specified distribution.

INTEGER INFO [Output]
: On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −I on exit, the i-th argument had an illegal value.

Example:

```
C       Generate 100 values from the Gaussian distribution
        INTEGER LSTATE,N
        PARAMETER  (LSTATE=16,N=100)
        INTEGER I,INFO,SEED(1),STATE(LSTATE)
        DOUBLE  PRECISION XMU,VAR
        DOUBLE  PRECISION X(N)
C       Set the seed
        SEED(1)  = 1234

C       Read in the distributional parameters
        READ(5,*)  XMU,VAR

C       Initialize the STATE vector
        CALL  DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)
C       Generate N variates from the Gaussian distribution
        CALL    DRANDGAUSSIAN(N,XMU,VAR,STATE,X,INFO)

C       Print the results
        WRITE(6,*) (X(I),I=1,N)
```

DRANDLOGISTIC / SRANDLOGISTIC

Generates a vector of random variates from a logistic distribution with probability density function, $f(X)$, where:

$$f(X) = \frac{e^{\frac{(X-A)}{B}}}{B(1+e^{\frac{(X-A)}{B}})^2}.$$

(Note that SRANDLOGISTIC is the single precision version of DRANDLOGISTIC. The argument lists of both routines are identical except that any double precision arguments of DRANDLOGISTIC are replaced in SRANDLOGISTIC by single precision arguments - type REAL in FORTRAN or type float in C).

DRANDLOGISTIC (*N,A,B,STATE,X,INFO*) [SUBROUTINE]

INTEGER N [Input]

On input: number of variates required.
Constraint: $N \geq 0$.

DOUBLE PRECISION A [Input]

On input: mean of the distribution.

DOUBLE PRECISION B [Input]

On input: spread of the distribution. $B = \sqrt{3}\sigma/\pi$ where σ is the standard deviation of the distribution.
Constraint: $B > 0$.

INTEGER STATE(*) [Input/Output]

The *STATE* vector holds information on the state of the base generator being used and as such its minimum length varies. Prior to calling DRANDLOGISTIC *STATE* must have been initialized. See Section 3.1.1 [Initialization of the Base Generators], for information on initialization of the *STATE* variable. On input: the current state of the base generator.
On output: the updated state of the base generator.

DOUBLE PRECISION X(*N*) [Output]

On output: vector of variates from the specified distribution.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0.
If $INFO = -I$ on exit, the i-th argument had an illegal value.

Example:

```
C     Generate 100 values from the Logistic distribution
      INTEGER LSTATE,N
      PARAMETER  (LSTATE=16,N=100)
      INTEGER I,INFO,SEED(1),STATE(LSTATE)
      DOUBLE  PRECISION A,B
      DOUBLE  PRECISION X(N)
C     Set the seed
      SEED(1)  = 1234

C     Read in the distributional parameters
      READ(5,*) A,B

C     Initialize the STATE vector
      CALL  DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)
C     Generate N variates from the Logistic distribution
      CALL    DRANDLOGISTIC(N,A,B,STATE,X,INFO)

C     Print the results
      WRITE(6,*) (X(I),I=1,N)
```

DRANDLOGNORMAL / SRANDLOGNORMAL

Generates a vector of random variates from a lognormal distribution with probability density function, $f(X)$, where:

$$f(X) = \frac{e^{-\frac{(\log X-\mu)^2}{2\sigma^2}}}{X\sigma\sqrt{2\pi}},$$

if $X > 0$, otherwise $f(X) = 0$. Here μ is the mean, (XMU) and σ^2 the variance, (VAR) of the underlying Gaussian distribution.

(Note that SRANDLOGNORMAL is the single precision version of DRANDLOGNORMAL. The argument lists of both routines are identical except that any double precision arguments of DRANDLOGNORMAL are replaced in SRANDLOGNORMAL by single precision arguments – type REAL in FORTRAN or type float in C).

DRANDLOGNORMAL (*N,XMU,VAR,STATE,X,INFO*) [SUBROUTINE]

INTEGER N [Input]

On input: number of variates required.
Constraint: $N \geq 0$.

DOUBLE PRECISION XMU [Input]

On input: mean of the underlying Gaussian distribution.

DOUBLE PRECISION VAR [Input]

On input: variance of the underlying Gaussian distribution.
Constraint: $VAR \geq 0$.

INTEGER STATE(*) [Input/Output]

The *STATE* vector holds information on the state of the base generator being used and as such its minimum length varies. Prior to calling DRANDLOGNORMAL *STATE* must have been initialized. See Section 3.1.1 [Initialization of the Base Generators], for information on initialization of the *STATE* variable. On input: the current state of the base generator.
On output: the updated state of the base generator.

DOUBLE PRECISION X(*N*) [Output]

On output: vector of variates from the specified distribution.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0.
If $INFO = -I$ on exit, the i-th argument had an illegal value.

Example:

```
C       Generate 100 values from the  Lognormal  distribution
        INTEGER LSTATE,N
        PARAMETER  (LSTATE=16,N=100)
        INTEGER I,INFO,SEED(1),STATE(LSTATE)
        DOUBLE  PRECISION XMU,VAR
        DOUBLE  PRECISION X(N)
C       Set  the  seed
        SEED(1)  = 1234

C       Read  in  the  distributional  parameters
        READ(5,*)  XMU,VAR

C       Initialize  the  STATE vector
        CALL  DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)
C       Generate  N  variates  from  the  Lognormal  distribution
        CALL  DRANDLOGNORMAL(N,XMU,VAR,STATE,X,INFO)

C       Print  the  results
        WRITE(6,*) (X(I),I=1,N)
```

DRANDSTUDENTST / SRANDSTUDENTST

Generates a vector of random variates from a Students T distribution with probability density function, $f(X)$, where:

$$f(X) = \frac{\frac{(\nu-1)}{2}!}{(\frac{\nu}{2})!\sqrt{\pi\nu}(1+\frac{X^2}{\nu})^{\frac{(\nu+1)}{2}}}.$$

Here ν is the degrees of freedom, *DF*.

(Note that SRANDSTUDENTST is the single precision version of DRANDSTUDENTST. The argument lists of both routines are identical except that any double precision arguments of DRANDSTUDENTST are replaced in SRANDSTUDENTST by single precision arguments – type REAL in FORTRAN or type float in C).

DRANDSTUDENTST (*N*,*DF*,*STATE*,*X*,*INFO*) [SUBROUTINE]

INTEGER N [Input]

On input: number of variates required.
Constraint: $N \geq 0$.

INTEGER DF [Input]

On input: degrees of freedom.
Constraint: $DF > 0$.

INTEGER STATE(*) [Input/Output]

The *STATE* vector holds information on the state of the base generator being used and as such its minimum length varies. Prior to calling DRANDSTUDENTST *STATE* must have been initialized. See Section 3.1.1 [Initialization of the Base Generators], for information on initialization of the *STATE* variable. On input: the current state of the base generator.
On output: the updated state of the base generator.

DOUBLE PRECISION X(*N*) [Output]

On output: vector of variates from the specified distribution.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0.
If *INFO* = −I on exit, the i-th argument had an illegal value.

Example:

```
C     Generate 100 values from the Students T distribution
      INTEGER LSTATE,N
      PARAMETER  (LSTATE=16,N=100)
      INTEGER I,INFO,SEED(1),STATE(LSTATE)
      INTEGER DF
      DOUBLE  PRECISION X(N)
C     Set the seed
      SEED(1)  = 1234

C     Read in the distributional parameters
      READ(5,*) DF

C     Initialize the STATE vector
      CALL  DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)
C     Generate N variates from the Students T distribution
      CALL DRANDSTUDENTST(N,DF,STATE,X,INFO)

C     Print the results
      WRITE(6,*) (X(I),I=1,N)
```

DRANDTRIANGULAR / SRANDTRIANGULAR

Generates a vector of random variates from a Triangular distribution with probability density function, $f(X)$, where:

$$f(X) = \frac{2(X - X_{MIN})}{(X_{MAX} - X_{MIN})(X_{MED} - X_{MIN})},$$

if $X_{MIN} < X \le X_{MED}$, else

$$f(X) = \frac{2(X_{MAX} - X)}{(X_{MAX} - X_{MIN})(X_{MAX} - X_{MED})},$$

if $X_{MED} < X \le X_{MAX}$, otherwise $f(X) = 0$.

(Note that SRANDTRIANGULAR is the single precision version of DRANDTRIANGULAR. The argument lists of both routines are identical except that any double precision arguments of DRANDTRIANGULAR are replaced in SRANDTRIANGULAR by single pre- cision arguments – type REAL in FORTRAN or type float in C).

DRANDTRIANGULAR (*N,XMIN,XMED,XMAX,STATE,X,INFO*) [SUBROUTINE]

INTEGER N [Input]

On input: number of variates required.
Constraint: $N \ge 0$.

DOUBLE PRECISION XMIN [Input]

On input: minimum value for the distribution.

DOUBLE PRECISION XMED [Input]

On input: median value for the distribution.
Constraint: $XMIN \le XMED \le XMAX$.

DOUBLE PRECISION XMAX [Input]

On input: maximum value for the distribution.
Constraint: $XMAX \ge XMIN$.

INTEGER STATE(*) [Input/Output]

The *STATE* vector holds information on the state of the base generator being used and as such its minimum length varies. Prior to calling DRANDTRIANGULAR *STATE* must have been initialized. See Section 3.1.1 [Initialization of the Base Generators], for information on initialization of the *STATE* variable. On input: the current state of the base generator.
On output: the updated state of the base generator.

DOUBLE PRECISION X(*N*) [Output]

On output: vector of variates from the specified distribution.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0.
If *INFO* = −I on exit, the i-th argument had an illegal value.

Example:

```
C       Generate 100 values from the Triangular distribution
        INTEGER LSTATE,N
        PARAMETER  (LSTATE=16,N=100)
        INTEGER I,INFO,SEED(1),STATE(LSTATE)
        DOUBLE PRECISION XMIN,XMED,XMAX
        DOUBLE  PRECISION X(N)
C       Set the seed
        SEED(1)  = 1234

C       Read in the distributional parameters
        READ(5,*)   XMIN,XMED,XMAX

C       Initialize the STATE vector
        CALL  DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)
C       Generate N variates from the Triangular distribution
        CALL   DRANDTRIANGULAR(N,XMIN,XMED,XMAX,STATE,X,INFO)

C       Print the results
        WRITE(6,*) (X(I),I=1,N)
```

DRANDUNIFORM / SRANDUNIFORM

Generates a vector of random variates from a Uniform distribution with probability density function, $f(X)$, where:

$$f(X) = \frac{1}{B - A}.$$

(Note that SRANDUNIFORM is the single precision version of DRANDUNIFORM. The argument lists of both routines are identical except that any double precision arguments of DRANDUNIFORM are replaced in SRANDUNIFORM by single precision arguments – type REAL in FORTRAN or type float in C).

DRANDUNIFORM (*N,A,B,STATE,X,INFO*) [SUBROUTINE]

INTEGER N [Input]

On input: number of variates required.
Constraint: $N \geq 0$.

DOUBLE PRECISION A [Input]

On input: minimum value for the distribution.

DOUBLE PRECISION B [Input]

On input: maximum value for the distribution.
Constraint: $B \geq A$.

INTEGER STATE(*) [Input/Output]

The *STATE* vector holds information on the state of the base generator being used and as such its minimum length varies. Prior to calling DRANDUNIFORM *STATE* must have been initialized. See Section 3.1.1 [Initialization of the Base Generators], for information on initialization of the *STATE* variable. On input: the current state of the base generator.
On output: the updated state of the base generator.

DOUBLE PRECISION X(*N*) [Output]

On output: vector of variates from the specified distribution.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0.
If *INFO* = −I on exit, the i-th argument had an illegal value.

Example:

```
C      Generate 100 values from the  Uniform distribution
       INTEGER LSTATE,N
       PARAMETER  (LSTATE=16,N=100)
       INTEGER I,INFO,SEED(1),STATE(LSTATE)
       DOUBLE  PRECISION A,B
       DOUBLE  PRECISION X(N)
C      Set the seed
       SEED(1)  = 1234

C      Read in the distributional parameters
       READ(5,*) A,B

C      Initialize the STATE vector
       CALL  DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)
C      Generate N variates from the Uniform distribution
       CALL  DRANDUNIFORM(N,A,B,STATE,X,INFO)

C      Print the results
       WRITE(6,*) (X(I),I=1,N)
```

DRANDVONMISES / SRANDVONMISES

Generates a vector of random variates from a Von Mises distribution with probability density function, $f(X)$, where:

$$f(X) = \frac{e^{\kappa \cos X}}{2\pi I_0(\kappa)}$$

where X is reduced modulo 2π so that it lies between $\pm\pi$, and κ is the concentration parameter VK.

(Note that SRANDVONMISES is the single precision version of DRANDVONMISES. The argument lists of both routines are identical except that any double precision arguments of DRANDVONMISES are replaced in SRANDVONMISES by single precision arguments - type REAL in FORTRAN or type float in C).

DRANDVONMISES (*N,VK,,STATE,X,INFO*) [SUBROUTINE]

INTEGER N [Input]

On input: number of variates required.
Constraint: $N \geq 0$.

DOUBLE PRECISION VK [Input]

On input: concentration parameter.
Constraint: $VK > 0$.

INTEGER STATE(*) [Input/Output]

The *STATE* vector holds information on the state of the base generator being used and as such its minimum length varies. Prior to calling DRANDVONMISES *STATE* must have been initialized. See Section 3.1.1 [Initialization of the Base Generators], for information on initialization of the *STATE* variable. On input: the current state of the base generator.
On output: the updated state of the base generator.

DOUBLE PRECISION X(*N*) [Output]

On output: vector of variates from the specified distribution.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If $INFO = -I$ on exit, the i-th argument had an illegal value.

Example:

```
C       Generate 100 values from the Von Mises distribution
        INTEGER LSTATE,N
        PARAMETER (LSTATE=16,N=100)
        INTEGER I,INFO,SEED(1),STATE(LSTATE)
        DOUBLE PRECISION VK
        DOUBLE PRECISION X(N)
C       Set the seed
        SEED(1) = 1234

C       Read in the distributional parameters
        READ(5,*) VK

C       Initialize the STATE vector
        CALL DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)
C       Generate N variates from the Von Mises distribution
        CALL DRANDVONMISES(N,VK,STATE,X,INFO)

C       Print the results
        WRITE(6,*) (X(I),I=1,N)
```

DRANDWEIBULL / SRANDWEIBULL

Generates a vector of random variates from a Weibull distribution with probability density function, $f(X)$, where:

$$f(X) = \frac{AX^{A-1}e^{-\frac{X^A}{B}}}{B},$$

if $X > 0$, otherwise $f(X) = 0$.

(Note that SRANDWEIBULL is the single precision version of DRANDWEIBULL. The argument lists of both routines are identical except that any double precision arguments of DRANDWEIBULL are replaced in SRANDWEIBULL by single precision arguments – type REAL in FORTRAN or type float in C).

DRANDWEIBULL (*N*,*A*,*B*,*STATE*,*X*,*INFO*) [SUBROUTINE]

INTEGER N [Input]

On input: number of variates required.
Constraint: $N \geq 0$.

DOUBLE PRECISION A [Input]

On input: shape parameter for the distribution.
Constraint: $A > 0$.

DOUBLE PRECISION B [Input]

On input: scale parameter for the distribution.
Constraint: $B > 0$.

INTEGER STATE(*) [Input/Output]

The *STATE* vector holds information on the state of the base generator being used and as such its minimum length varies. Prior to calling DRANDWEIBULL *STATE* must have been initialized. See Section 3.1.1 [Initialization of the Base Generators], for information on initialization of the *STATE* variable. On input: the current state of the base generator.
On output: the updated state of the base generator.

DOUBLE PRECISION X(*N*) [Output]

On output: vector of variates from the specified distribution.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0.
If *INFO* = −I on exit, the i-th argument had an illegal value.

Example:

```
C     Generate 100 values from the Weibull distribution
      INTEGER LSTATE,N
      PARAMETER (LSTATE=16,N=100)
      INTEGER I,INFO,SEED(1),STATE(LSTATE)
      DOUBLE PRECISION A,B
      DOUBLE PRECISION X(N)
C     Set the seed
      SEED(1) = 1234

C     Read in the distributional parameters
      READ(5,*) A,B

C     Initialize the STATE vector
      CALL DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)
C     Generate N variates from the Weibull distribution
      CALL DRANDWEIBULL(N,A,B,STATE,X,INFO)

C     Print the results
      WRITE(6,*) (X(I),I=1,N)
```

3.3.2 Discrete Univariate Distributions

DRANDBINOMIAL / SRANDBINOMIAL

Generates a vector of random variates from a Binomial distribution with probability, $f(X)$, defined by:

$$f(X) = \frac{M!P^X(1-P)^{(M-X)}}{X!(M-1)!}, X = 0, 1, \cdots, M$$

(Note that SRANDBINOMIAL is the single precision version of DRANDBINOMIAL. The argument lists of both routines are identical except that any double precision arguments of DRANDBINOMIAL are replaced in SRANDBINOMIAL by single precision arguments - type REAL in FORTRAN or type float in C).

DRANDBINOMIAL (*N,M,P,STATE,X,INFO*) [SUBROUTINE]

INTEGER N [Input]

On input: number of variates required.
Constraint: $N \geq 0$.

INTEGER M [Input]

On input: number of trials.
Constraint: $M \geq 0$.

DOUBLE PRECISION P [Input]

On input: probability of success.
Constraint: $0 \leq P < 1$.

INTEGER STATE(*) [Input/Output]

The *STATE* vector holds information on the state of the base generator being used and as such its minimum length varies. Prior to calling DRANDBINOMIAL *STATE* must have been initialized. See Section 3.1.1 [Initialization of the Base Generators], for information on initialization of the *STATE* variable. On input: the current state of the base generator.
On output: the updated state of the base generator.

INTEGER X(*N*) [Output]

On output: vector of variates from the specified distribution.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −I on exit, the i-th argument had an illegal value.

Example:

```
C      Generate 100 values from the Binomial distribution
       INTEGER LSTATE,N
       PARAMETER  (LSTATE=16,N=100)
       INTEGER I,INFO,SEED(1),STATE(LSTATE)
       INTEGER M
       DOUBLE PRECISION P
       INTEGER X(N)
C      Set the seed
       SEED(1)  = 1234

C      Read in the distributional parameters
       READ(5,*)  M,P

C      Initialize the STATE vector
       CALL  DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)
C      Generate N variates from the Binomial distribution
       CALL  DRANDBINOMIAL(N,M,P,STATE,X,INFO)

C      Print the results
       WRITE(6,*) (X(I),I=1,N)
```

DRANDGEOMETRIC / SRANDGEOMETRIC

Generates a vector of random variates from a Geometric distribution with probability, $f(X)$, defined by:

$$f(X) = P(1 - P)^{X}, X = 0, 1, \cdots$$

(Note that SRANDGEOMETRIC is the single precision version of DRANDGEOMETRIC. The argument lists of both routines are identical except that any double precision arguments of DRANDGEOMETRIC are replaced in SRANDGEOMETRIC by single precision arguments – type REAL in FORTRAN or type float in C).

DRANDGEOMETRIC (*N*,*P*,*STATE*,*X*,*INFO*) [SUBROUTINE]

INTEGER N [Input]

On input: number of variates required.
Constraint: $N \geq 0$.

DOUBLE PRECISION P [Input]

On input: distribution parameter.
Constraint: $0 \leq P < 1$.

INTEGER STATE(*) [Input/Output]

The *STATE* vector holds information on the state of the base generator being used and as such its minimum length varies. Prior to calling DRANDGEOMETRIC *STATE* must have been initialized. See Section 3.1.1 [Initialization of the Base Generators], for information on initialization of the *STATE* variable. On input: the current state of the base generator.
On output: the updated state of the base generator.

INTEGER X(*N*) [Output]

On output: vector of variates from the specified distribution.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0.
If *INFO* = −I on exit, the i-th argument had an illegal value.

Example:

```
C     Generate 100 values from the Geometric distribution
      INTEGER LSTATE,N
      PARAMETER  (LSTATE=16,N=100)
      INTEGER I,INFO,SEED(1),STATE(LSTATE)
      DOUBLE  PRECISION P
      INTEGER X(N)
C     Set the seed
      SEED(1)  = 1234

C     Read in the distributional parameters
      READ(5,*) P

C     Initialize the STATE vector
      CALL  DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)
C     Generate N variates from the Geometric distribution
      CALL DRANDGEOMETRIC(N,P,STATE,X,INFO)

C     Print the results
      WRITE(6,*) (X(I),I=1,N)
```

DRANDHYPERGEOMETRIC / SRANDHYPERGEOMETRIC

Generates a vector of random variates from a Hypergeometric distribution with probability, $f(X)$, defined by:

$$f(X) = \frac{s!m!(p-s)!(p-m)!}{X!(s-X)!(m-X)!(p-m-s+X)!p!},$$

if $X = \max(0, m+s\ p), , \min(l, m)$, otherwise $f(X) = 0$. Here p is the size of the population, (NP), s is the size of the sample taken from the population, (NS) and m is the number of labeled, or specified, items in the population, (M).

(Note that SRANDHYPERGEOMETRIC is the single precision version of DRANDHYPERGEOMETRIC. The argument lists of both routines are identical except that any double precision arguments of DRANDHYPERGEOMETRIC are replaced in SRANDHYPERGEOMETRIC by single precision arguments - type REAL in FORTRAN or type float in C).

DRANDHYPERGEOMETRIC (*N,NP,NS,M,STATE,X,INFO*) [SUBROUTINE]

INTEGER N [Input]

On input: number of variates required.
Constraint: $N \geq 0$.

INTEGER NP [Input]

On input: size of population.
Constraint: $NP \geq 0$.

INTEGER NS [Input]

On input: size of sample being taken from population.
Constraint: $0 \leq NS \leq NP$.

INTEGER M [Input]

On input: number of specified items in the population.
Constraint: $0 \leq M \leq NP$.

INTEGER STATE(*) [Input/Output]

The *STATE* vector holds information on the state of the base generator being used and as such its minimum length varies. Prior to calling DRANDHYPERGEOMETRIC *STATE* must have been initialized. See Section 3.1.1 [Initialization of the Base Generators], for information on initialization of the *STATE* variable.
On input: the current state of the base generator.
On output: the updated state of the base generator.

INTEGER X(*N*) [Output]

On output: vector of variates from the specified distribution.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0.
If $INFO = -I$ on exit, the i-th argument had an illegal value.

Example:

```
C       Generate 100 values from the Hypergeometric distribution
        INTEGER LSTATE,N
        PARAMETER  (LSTATE=16,N=100)
        INTEGER I,INFO,SEED(1),STATE(LSTATE)
        INTEGER NP,NS,M
        INTEGER X(N)
C       Set the seed
        SEED(1)  = 1234

C       Read in the distributional parameters
        READ(5,*) NP,NS,M

C       Initialize the STATE vector
        CALL  DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)
C       Generate N variates from the Hypergeometric distribution
        CALL    DRANDHYPERGEOMETRIC(N,NP,NS,M,STATE,X,INFO)

C       Print the results
        WRITE(6,*) (X(I),I=1,N)
```

DRANDNEGATIVEBINOMIAL / SRANDNEGATIVEBINOMIAL

Generates a vector of random variates from a Negative Binomial distribution with probability $f(X)$ defined by:

$$f(X) = \frac{(M + X - 1)! P^X (1 - P)^M}{X!(M - 1)!}, \; X = 0, 1, \ldots$$

(Note that SRANDNEGATIVEBINOMIAL is the single precision version of DRANDNEGATIVEBINOMIAL. The argument lists of both routines are identical except that any double precision arguments of DRANDNEGATIVEBINOMIAL are replaced in SRANDNEGATIVEBINOMIAL by single precision arguments – type REAL in FORTRAN or type float in C).

DRANDNEGATIVEBINOMIAL (*N,M,P,STATE,X,INFO*) [SUBROUTINE]

INTEGER N [Input]

On input: number of variates required.
Constraint: $N \geq 0$.

INTEGER M [Input]

On input: number of failures.
Constraint: $M \geq 0$.

DOUBLE PRECISION P [Input]

On input: probability of success.
Constraint: $0 \leq P < 1$.

INTEGER STATE(*) [Input/Output]

The *STATE* vector holds information on the state of the base generator being used and as such its minimum length varies. Prior to calling DRANDNEGATIVEBINOMIAL *STATE* must have been initialized. See Section 3.1.1 [Initialization of the Base Generators], for information on initialization of the *STATE* variable.
On input: the current state of the base generator.
On output: the updated state of the base generator.

INTEGER X(*N*) [Output]

On output: vector of variates from the specified distribution.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0.
If *INFO* = −I on exit, the i-th argument had an illegal value.

Example:

```
C       Generate 100 values from the Negative Binomial distribution
        INTEGER LSTATE,N
        PARAMETER  (LSTATE=16,N=100)
        INTEGER I,INFO,SEED(1),STATE(LSTATE)
        INTEGER M
        DOUBLE PRECISION P
        INTEGER X(N)
C       Set the seed
        SEED(1)  = 1234

C       Read in the distributional parameters
        READ(5,*)  M,P

C       Initialize the STATE vector
        CALL  DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)
C       Generate N variates from the Negative Binomial distribution
        CALL      DRANDNEGATIVEBINOMIAL(N,M,P,STATE,X,INFO)

C       Print the results
        WRITE(6,*) (X(I),I=1,N)
```

DRANDPOISSON / SRANDPOISSON

Generates a vector of random variates from a Poisson distribution with probability $f(X)$ defined by:

$$f(X) = \frac{\lambda^X e^{-\lambda}}{X!}, X = 0, 1, \cdots,$$

where λ is the mean of the distribution, *LAMBDA*.

(Note that SRANDPOISSON is the single precision version of DRANDPOISSON. The argument lists of both routines are identical except that any double precision arguments of DRANDPOISSON are replaced in SRANDPOISSON by single precision arguments – type REAL in FORTRAN or type float in C).

DRANDPOISSON (*N*,*LAMBDA*,*STATE*,*X*,*INFO*) [SUBROUTINE]

INTEGER N [Input]

On input: number of variates required.
Constraint: $N \geq 0$.

INTEGER M [Input]

On input: number of failures.
Constraint: $M \geq 0$.

DOUBLE PRECISION LAMBDA [Input]

On input: mean of the distribution.
Constraint: $LAMBDA \geq 0$.

INTEGER STATE(*) [Input/Output]

The *STATE* vector holds information on the state of the base generator being used and as such its minimum length varies. Prior to calling DRANDPOISSON *STATE* must have been initialized. See Section 3.1.1 [Initialization of the Base Generators], for information on initialization of the *STATE* variable. On input: the current state of the base generator.
On output: the updated state of the base generator.

INTEGER X(*N*) [Output]

On output: vector of variates from the specified distribution.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0.
If *INFO* = −I on exit, the i-th argument had an illegal value.

Example:

```
C       Generate 100 values from the Poisson distribution
        INTEGER LSTATE,N
        PARAMETER  (LSTATE=16,N=100)
        INTEGER I,INFO,SEED(1),STATE(LSTATE)
        DOUBLE PRECISION LAMBDA
        INTEGER X(N)
C       Set the seed
        SEED(1)  = 1234

C       Read in the distributional parameters
        READ(5,*) LAMBDA

C       Initialize the STATE vector
        CALL  DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)
C       Generate N variates from the Poisson distribution
        CALL    DRANDPOISSON(N,LAMBDA,STATE,X,INFO)

C       Print the results
        WRITE(6,*) (X(I),I=1,N)
```

`DRANDDISCRETEUNIFORM / SRANDDISCRETEUNIFORM`

Generates a vector of random variates from a Uniform distribution with probability $f(X)$ defined by:

$$f(X) = \frac{1}{(B - A)}, X = A, A + 1, \cdots, B$$

(Note that SRANDDISCRETEUNIFORM is the single precision version of DRANDDISCRETEUNIFORM. The argument lists of both routines are identical except that any double precision arguments of DRANDDISCRETEUNIFORM are replaced in SRANDDISCRETEUNIFORM by single precision arguments - type REAL in FORTRAN or type float in C).

DRANDDISCRETEUNIFORM (*N,A,B,STATE,X,INFO*) [SUBROUTINE]

INTEGER N [Input]

On input: number of variates required.
Constraint: $N \geq 0$.

INTEGER A [Input]

On input: minimum for the distribution.

INTEGER B [Input]

On input: maximum for the distribution.
Constraint: $B \geq A$.

INTEGER STATE(*) [Input/Output]

The *STATE* vector holds information on the state of the base generator being used and as such its minimum length varies. Prior to calling DRANDDISCRETEUNIFORM *STATE* must have been initialized. See Section 3.1.1 [Initialization of the Base Generators], for information on initialization of the *STATE* variable.
On input: the current state of the base generator.
On output: the updated state of the base generator.

INTEGER X(*N*) [Output]

On output: vector of variates from the specified distribution.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0.
If *INFO* = −I on exit, the i-th argument had an illegal value.

Example:

```
C     Generate 100 values from the  Uniform distribution
      INTEGER LSTATE,N
      PARAMETER  (LSTATE=16,N=100)
      INTEGER I,INFO,SEED(1),STATE(LSTATE)
      INTEGER A,B
      INTEGER X(N)
C     Set the seed
      SEED(1)  = 1234

C     Read in the distributional parameters
      READ(5,*) A,B

C     Initialize the STATE vector
      CALL  DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)
C     Generate N variates from the Uniform distribution
      CALL DRANDDISCRETEUNIFORM(N,A,B,STATE,X,INFO)

C     Print the results
      WRITE(6,*) (X(I),I=1,N)
```

DRANDGENERALDISCRETE / SRANDGENERALDISCRETE

Takes a reference vector initialized via one of DRANDBINOMIALREFERENCE, DRANDGEOMETRIC REFERENCE, DRANDHYPERGEOMETRICREFERENCE, DRANDNEGATIVEBINOMIALREFERENCE, DRAND POISSONREFERENCE and generates a vector of random variates from it.

(Note that SRANDGENERALDISCRETE is the single precision version of DRANDGENERALDISCRETE. The argument lists of both routines are identical except that any double precision arguments of DRANDGENERALDISCRETE are replaced in SRANDGENERALDISCRETE by single precision arguments - type REAL in FORTRAN or type float in C).

DRANDGENERALDISCRETE (*N,REF,STATE,X,INFO*) [SUBROUTINE]

INTEGER N [Input]

On input: number of variates required.
Constraint: $N \geq 0$.

DOUBLE PRECISION REF(*) [Input]

On input: reference vector generated by one of the following: DRANDBINOMIALREFERENCE, DRANDGEOMETRICREFERENCE, DRANDHYPERGEOMETRICREFERENCE, DRANDNEGATIVEBINOMIALREFERENCE, DRANDPOISSONREFERENCE.

INTEGER STATE(*) [Input/Output]

The *STATE* vector holds information on the state of the base generator being used and as such its minimum length varies. Prior to calling DRANDGENERALDISCRETE *STATE* must have been initialized. See Section 3.1.1 [Initialization of the Base Generators], for information on initialization of the *STATE* variable.
On input: the current state of the base generator.
On output: the updated state of the base generator.

INTEGER X(*N*) [Output]

On output: vector of variates from the specified distribution.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0.
If *INFO* = $-$I on exit, the i-th argument had an illegal value.

Example:

```
C      Generate 100 values from the Binomial distribution
       INTEGER LSTATE,N
       PARAMETER  (LSTATE=16,N=100)
       INTEGER I,INFO,SEED(1),STATE(LSTATE)
       INTEGER M
       DOUBLE PRECISION P
       INTEGER X(N)
       INTEGER LREF
       DOUBLE PRECISION REF(1000)

C      Set the seed
       SEED(1)  = 1234

C      Read in the distributional parameters
       READ(5,*)  M,P

C      Initialize the STATE vector
       CALL  DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)

C      Initialize the reference vector
       LREF  = 1000
       CALL     DRANDBINOMIALREFERENCE(M,P,REF,LREF,INFO)

C      Generate N variates from the Binomial distribution
       CALL     DRANDGENERALDISCRETE(N,REF,STATE,X,INFO)
C      Print the results
       WRITE(6,*) (X(I),I=1,N)
```

DRANDBINOMIALREFERENCE / SRANDBINOMIALREFERENCE

Initializes a reference vector for use with DRANDGENERALDISCRETE. Reference vector is for a Binomial distribution with probability, $f(X)$, defined by:

$$f(X) = \frac{M!P^X(1-P)^{(M-X)}}{X!(M-1)!}, X = 0, 1, \cdots, M$$

(Note that SRANDBINOMIALREFERENCE is the single precision version of DRANDBINOMIALREFERENCE. The argument lists of both routines are identical except that any double precision arguments of DRANDBINOMIALREFERENCE are replaced in SRANDBINOMIALREFERENCE by single precision arguments - type REAL in FORTRAN or type float in C).

DRANDBINOMIALREFERENCE (*M,P,REF,LREF,INFO*) [SUBROUTINE]

INTEGER M [Input]

On input: number of trials.
Constraint: $M \geq 0$.

DOUBLE PRECISION P [Input]

On input: probability of success.
Constraint: $0 \leq P < 1$.

DOUBLE PRECISION REF(LREF) [Output]

On output: if *INFO* returns with a value of 0 then *REF* contains reference information required to generate values from a Binomial distribution using DRANDGENERALDISCRETE.

INTEGER LREF [Input/Output]

On input: either the length of the reference vector *REF*, or 1. –
On output: if $LREF = 1$ on input, then *LREF* is set to the recommended length of the reference vector and the routine returns. Otherwise *LREF* is left unchanged.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. If *INFO* = Lon exit, the i-th argument had an illegal value. If *INFO* = 1 on exit, then *LREF* has been set to the recommended length for the reference vector *REF*. If *INFO* = 0 then the reference vector, *REF*, has been successfully initialized.

Example:

```
C     Generate 100 values from the Binomial distribution
      INTEGER LSTATE,N
      PARAMETER (LSTATE=16,N=100)
      INTEGER I,INFO,SEED(1),STATE(LSTATE)
      INTEGER M
      DOUBLE PRECISION P
      INTEGER X(N)
      INTEGER LREF
      DOUBLE PRECISION REF(1000)

C     Set the seed
      SEED(1) = 1234

C     Read in the distributional parameters
      READ(5,*) M,P

C     Initialize the STATE vector
      CALL DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)

C     Initialize the reference vector
      LREF = 1000
      CALL DRANDBINOMIALREFERENCE(M,P,REF,LREF,INFO)

C     Generate N variates from the Binomial distribution
      CALL DRANDGENERALDISCRETE(N,REF,STATE,X,INFO)
C     Print the results
      WRITE(6,*) (X(I),I=1,N)
```

DRANDGEOMETRICREFERENCE / SRANDGEOMETRICREFERENCE

Initializes a reference vector for use with DRANDGENERALDISCRETE. Reference vector is for a Geometric distribution with probability, $f(X)$, defined by:

$$f(X) = P(1 - P)^X, X = 0, 1, \cdots$$

(Note that SRANDGEOMETRICREFERENCE is the single precision version of DRANDGEOMETRICREFERENCE. The argument lists of both routines are identical except that any double precision arguments of DRANDGEOMETRICREFERENCE are replaced in SRANDGEOMETRICREFERENCE by single precision arguments - type REAL in FORTRAN or type float in C).

DRANDGEOMETRICREFERENCE (*P,REF,LREF,INFO*) [SUBROUTINE]

DOUBLE PRECISION P [Input]

On input: distribution parameter.
Constraint: $0 \leq P < 1$.

DOUBLE PRECISION REF(LREF) [Output]

On output: if *INFO* returns with a value of 0 then *REF* contains reference information required to generate values from a Geometric distribution using DRANDGENERALDISCRETE.

INTEGER LREF [Input/Output]

On input: either the length of the reference vector *REF*, or -1.
On output: if *LREF* = -1 on input, then *LREF* is set to the recommended length of the reference vector and the routine returns. Otherwise *LREF* is left unchanged.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. If *INFO* = $-i$ on exit, the i-th argument had an illegal value. If *INFO* = 1 on exit, then *LREF* has been set to the recommended length for the reference vector *REF*. If *INFO* = 0 then the reference vector, *REF*, has been successfully initialized.

Example:

```
C     Generate 100 values from the Geometric distribution
      INTEGER LSTATE,N
      PARAMETER  (LSTATE=16,N=100)
      INTEGER I,INFO,SEED(1),STATE(LSTATE)
      DOUBLE  PRECISION P
      INTEGER X(N)
      INTEGER LREF
      DOUBLE PRECISION REF(1000)

C     Set the seed
      SEED(1)  = 1234

C     Read in the distributional parameters
      READ(5,*) P

C     Initialize the STATE vector
      CALL  DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)

C     Initialize the reference vector
      LREF  = 1000
      CALL    DRANDGEOMETRICREFERENCE(P,REF,LREF,INFO)

C     Generate N variates from the Geometric distribution
      CALL     DRANDGENERALDISCRETE(N,REF,STATE,X,INFO)
C     Print the results
      WRITE(6,*) (X(I),I=1,N)
```

DRANDHYPERGEOMETRICREFERENCE / SRANDHYPERGEOMETRICREFERENCE

Initializes a reference vector for use with DRANDGENERALDISCRETE. Reference vector is for a Hypergeometric distribution with probability, $f(X)$, defined by:

$$f(X) = \frac{s!m!(p-s)!(p-m)!}{X!(s-X)!(m-X)!(p-m-s+X)!p!},$$

if $X = \max(0, m+s-p), \ldots, \min(l, m)$, otherwise $f(X) = 0$. Here p is the size of the population, (NP), s is the size of the sample taken from the population, (NS) and m is the number of labeled, or specified, items in the population, (M).

(Note that SRANDHYPERGEOMETRICREFERENCE is the single precision version of DRANDHYPERGEOMETRICREFERENCE. The argument lists of both routines are identical except that any double precision arguments of DRANDHYPERGEOMETRICREF- ERENCE are replaced in SRANDHYPERGEOMETRICREFERENCE by single precision arguments – type REAL in FORTRAN or type float in C).

DRANDHYPERGEOMETRICREFERENCE (*NP,NS,M,REF,LREF,INFO*) [SUBROUTINE]

INTEGER NP [Input]

On input: size of population.
Constraint: $NP \geq 0$.

INTEGER NS [Input]

On input: size of sample being taken from population.
Constraint: $0 \leq NS \leq NP$.

INTEGER M [Input]

On input: number of specified items in the population.
Constraint: $0 \leq M \leq NP$.

DOUBLE PRECISION REF(LREF) [Output]

On output: if *INFO* returns with a value of 0 then *REF* contains reference information required to generate values from a Hypergeometric distribution using DRANDGENERALDISCRETE.

INTEGER LREF [Input/Output]

On input: either the length of the reference vector *REF*, or -1.
On output: if $LREF = -1$ on input, then *LREF* is set to the recommended length of the reference vector and the routine returns. Otherwise *LREF* is left unchanged.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. If $INFO = -i$ on exit, the i-th argument had an illegal value. If $INFO = 1$ on exit, then *LREF* has been set to the recommended length for the reference vector *REF*. If $INFO = 0$ then the reference vector, *REF*, has been successfully initialized.

Example:

```
C      Generate 100 values from the Hypergeometric distribution
       INTEGER LSTATE,N
       PARAMETER  (LSTATE=16,N=100)
       INTEGER I,INFO,SEED(1),STATE(LSTATE)
       INTEGER NP, NS,M
       INTEGER X(N)
       INTEGER LREF
       DOUBLE PRECISION REF(1000)

C      Set the seed
       SEED(1)  = 1234

C      Read in the distributional parameters
       READ(5,*) NP, NS,M

C      Initialize the STATE vector
       CALL  DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)

C      Initialize the reference vector
       LREF  = 1000
       CALL   DRANDHYPERGEOMETRICREFERENCE(NP,  NS,M,REF,LREF,INFO)

C      Generate N variates from the Hypergeometric distribution
       CALL    DRANDGENERALDISCRETE(N,REF,STATE,X,INFO)
C      Print the results
       WRITE(6,*) (X(I),I=1,N)
```

DRANDNEGATIVEBINOMIALREFERENCE / SRANDNEGATIVEBINOMIALREFERENCE

Initializes a reference vector for use with DRANDGENERALDISCRETE. Reference vector is for a Negative Binomial distribution with probability $f(X)$ defined by:

$$f(X) = \frac{(M + X - 1)!P^X(1 - P)^M}{X!(M - 1)!}, X = 0, 1, \cdots$$

(Note that SRANDNEGATIVEBINOMIALREFERENCE is the single precision version of DRANDNEGATIVEBINOMIALREFERENCE. The argument lists of both routines are identical except that any double precision arguments of DRANDNEGATIVEBINOMIAL-REFERENCE are replaced in SRANDNEGATIVEBINOMIALREFERENCE by single precision arguments – type REAL in FORTRAN or type float in C).

DRANDNEGATIVEBINOMIALREFERENCE (*M,P,REF,LREF,INFO*) [SUBROUTINE]

INTEGER M [Input]

On input: number of failures.
Constraint: $M \geq 0$.

DOUBLE PRECISION P [Input]

On input: probability of success.
Constraint: $0 \leq P < 1$.

DOUBLE PRECISION REF(LREF) [Output]

On output: if *INFO* returns with a value of 0 then *REF* contains reference information required to generate values from a Negative Binomial distribution using DRANDGENERALDISCRETE.

INTEGER LREF [Input/Output]

On input: either the length of the reference vector *REF*, or −1.
On output: if $LREF = -1$ on input, then *LREF* is set to the recommended length of the reference vector and the routine returns. Otherwise *LREF* is left unchanged.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. If $INFO = -i$ on exit, the i-th argument had an illegal value. If *INFO* = 1 on exit, then *LREF* has been set to the recommended length for the reference vector *REF*. If $INFO = 0$ then the reference vector, *REF*, has been successfully initialized.

Example:

```
C       Generate 100 values from the Negative Binomial distribution
        INTEGER LSTATE,N
        PARAMETER  (LSTATE=16,N=100)
        INTEGER I,INFO,SEED(1),STATE(LSTATE)
        INTEGER M
        DOUBLE PRECISION P
        INTEGER X(N)
        INTEGER LREF
        DOUBLE PRECISION REF(1000)

C       Set the seed
        SEED(1)  = 1234

C       Read in the distributional parameters
        READ(5,*)  M,P

C       Initialize the STATE vector
        CALL  DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)

C       Initialize the reference vector
        LREF  = 1000
        CALL    DRANDNEGATIVEBINOMIALREFERENCE(M,P,REF,LREF,INFO)

C       Generate N variates from the Negative Binomial distribution
        CALL     DRANDGENERALDISCRETE(N,REF,STATE,X,INFO)
C       Print the results
        WRITE(6,*) (X(I),I=1,N)
```

DRANDPOISSONREFERENCE / SRANDPOISSONREFERENCE

Initializes a reference vector for use with DRANDGENERALDISCRETE. Reference vector is for a Poisson distribution with probability $f(X)$ defined by:

$$f(X) = \frac{\lambda^X e^{-\lambda}}{X!}, X = 0, 1, \cdots,$$

where λ is the mean of the distribution, ***LAMBDA***.

(Note that SRANDPOISSONREFERENCE is the single precision version of DRANDPOISSONREFERENCE. The argument lists of both routines are identical except that any double precision arguments of DRANDPOISSONREFERENCE are replaced in SRANDPOISSONREFERENCE by single precision arguments - type REAL in FORTRAN or type float in C).

DRANDPOISSONREFERENCE (*LAMBDA*,*REF*,*LREF*,*INFO*) [SUBROUTINE]

INTEGER M [Input]

On input: number of failures.
Constraint: $M \geq 0$.

DOUBLE PRECISION LAMBDA [Input]

On input: mean of the distribution.
Constraint: $LAMBDA \geq 0$.

DOUBLE PRECISION REF(LREF) [Output]

On output: if ***INFO*** returns with a value of 0 then ***REF*** contains reference information required to generate values from a Poisson distribution using DRANDGENERALDISCRETE.

INTEGER LREF [Input/Output]

On input: either the length of the reference vector ***REF***, or −1.
On output: if ***LREF*** = −1 on input, then ***LREF*** is set to the recommended length of the reference vector and the routine returns. Otherwise ***LREF*** is left unchanged.

INTEGER INFO [Output]

On output: ***INFO*** is an error indicator. If ***INFO*** = −i on exit, the i-th argument had an illegal value. If ***INFO*** = 1 on exit, then ***LREF*** has been set to the recommended length for the reference vector ***REF***. If ***INFO*** = 0 then the reference vector, ***REF***, has been successfully initialized.

Example:

```
C     Generate 100 values from the Poisson distribution
      INTEGER LSTATE,N
      PARAMETER  (LSTATE=16,N=100)
      INTEGER I,INFO,SEED(1),STATE(LSTATE)
      DOUBLE PRECISION LAMBDA
      INTEGER X(N)
      INTEGER LREF
      DOUBLE PRECISION REF(1000)

C     Set the seed
      SEED(1)  = 1234

C     Read in the distributional parameters
      READ(5,*) LAMBDA

C     Initialize the STATE vector
      CALL  DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)

C     Initialize the reference vector
      LREF  = 1000
      CALL    DRANDPOISSONREFERENCE(LAMBDA,REF,LREF,INFO)

C     Generate N variates from the Poisson distribution
      CALL      DRANDGENERALDISCRETE(N,REF,STATE,X,INFO)
C     Print the results
      WRITE(6,*) (X(I),I=1,N)
```

3.3.3 Continuous Multivariate Distributions

DRANDMULTINORMAL / SRANDMULTINORMAL

Generates an array of random variates from a Multivariate Normal distribution with probability density function, $f(X)$, where:

$$f(X) = \sqrt{\frac{|C^{-1}|}{(2\pi)^M}} e^{-(X-\mu)^T C^{-1}(X-\mu)},$$

where μ is the vector of means, XMU.

(Note that SRANDMULTINORMAL is the single precision version of DRANDMULTINORMAL. The argument lists of both routines are identical except that any double precision arguments of DRANDMULTINORMAL are replaced in SRANDMULTINORMAL by single precision arguments - type REAL in FORTRAN or type float in C).

DRANDMULTINORMAL (*N,M,XMU,C,LDC,STATE,X,LDX,INFO*) [SUBROUTINE]

INTEGER N [Input]

On input: number of variates required.
Constraint: $N \geq 0$.

INTEGER M [Input]

On input: number of dimensions for the distribution.
Constraint: $M \geq 1$.

DOUBLE PRECISION XMU(M) [Input]

On input: vector of means for the distribution.

DOUBLE PRECISION C(LDC,M) [Input]

On input: variance / covariance matrix for the distribution.

INTEGER LDC [Input]

On input: leading dimension of C in the calling routine.
Constraint: $LDC \geq M$.

INTEGER STATE(*) [Input/Output]

The $STATE$ vector holds information on the state of the base generator being used and as such its minimum length varies. Prior to calling DRANDMULTINORMAL $STATE$ must have been initialized. See Section 3.1.1 [Initialization of the Base Generators], for information on initialization of the $STATE$ variable. On input: the current state of the base generator.
On output: the updated state of the base generator.

DOUBLE PRECISION X(LDX,M) [Output]

On output: matrix of variates from the specified distribution.

INTEGER LDX [Input]

On input: leading dimension of X in the calling routine.
Constraint: $LDX \geq N$.

INTEGER INFO [Output]

On output: $INFO$ is an error indicator. On successful exit, $INFO$ contains 0.
If $INFO = -i$ on exit, the i-th argument had an illegal value.

Example:

```
C       Generate 100 values from the
C       Multivariate Normal distribution
        INTEGER LSTATE,N, MM
        PARAMETER (LSTATE=16,N=100,MM=10)
        INTEGER I,J,INFO,SEED(1),STATE(LSTATE)
        INTEGER LDC,LDX,M
        DOUBLE PRECISION X(N,MM),XMU(MM),C(MM,MM)
C       Set array sizes
        LDC = MM
        LDX = N

C       Set the seed
        SEED(1) = 1234

C       Read in the distributional parameters
        READ(5,*) M
        READ(5,*) (XMU(I),I=1,M)
        DO 20 I = 1,M
          READ(5,*) (C(I,J),J=1,M)
20      CONTINUE

C       Initialize the STATE vector
        CALL DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)
C       Generate N variates from the
C       Multivariate Normal distribution
        CALL DRANDMULTINORMAL(N,M,XMU,C,LDC,STATE,X,LDX,INFO)

C       Print the results
        DO 40 I = 1,N
          WRITE(6,*) (X(I,J),J=1,M)
40      CONTINUE
```

DRANDMULTISTUDENTST / SRANDMULTISTUDENTST

Generates an array of random variates from a Multivariate Students T distribution with probability density function, $f(X)$, where:

$$f(X) = \frac{\Gamma\left(\frac{(\nu+M)}{2}\right)}{(\pi\nu)^{\frac{m}{2}}\Gamma(\frac{\nu}{2})\,|C|^{\frac{1}{2}}}\left(1+\frac{(X-\mu)^T C^{-1}(X-\mu)}{\nu}\right)^{-\frac{(\nu+M)}{2}},$$

where μ is the vector of means, *XMU* and ν is the degrees of freedom, *DF*.

(Note that SRANDMULTISTUDENTST is the single precision version of DRANDMULTISTUDENTST. The argument lists of both routines are identical except that any double precision arguments of DRANDMULTISTUDENTST are replaced in SRANDMULTISTUDENTST by single precision arguments - type REAL in FORTRAN or type float in C).

DRANDMULTISTUDENTST (*N,M,DF,XMU,C,LDC,STATE,X,LDX,INFO*) [SUBROUTINE]

INTEGER N [Input]

On input: number of variates required.
Constraint: $N \geq 0$.

INTEGER M [Input]

On input: number of dimensions for the distribution.
Constraint: $M \geq 1$.

INTEGER DF [Input]

On input: degrees of freedom.
Constraint: $DF > 2$.

DOUBLE PRECISION XMU(M) [Input]

On input: vector of means for the distribution.

DOUBLE PRECISION C(LDC,M) [Input]

On input: matrix defining the variance / covariance for the distribution. The variance / covariance matrix is given by $\frac{\nu}{\nu-2}C$, where ν are the degrees of freedom, *DF*.

INTEGER LDC [Input]

On input: leading dimension of *C* in the calling routine.
Constraint: $LDC \geq M$.

INTEGER STATE(*) [Input/Output]

The *STATE* vector holds information on the state of the base generator being used and as such its minimum length varies. Prior to calling DRANDMULTISTUDENTST *STATE* must have been initialized. See Section 3.1.1 [Initialization of the Base Generators], for information on initialization of the *STATE* variable.
On input: the current state of the base generator.
On output: the updated state of the base generator.

DOUBLE PRECISION X(LDX,M) [Output]

On output: matrix of variates from the specified distribution.

INTEGER LDX [Input]

On input: leading dimension of X in the calling routine.
Constraint: $LDX \geq N$.

INTEGER INFO [Output]

On output: $INFO$ is an error indicator. On successful exit, $INFO$ contains 0. If $INFO = -i$ on exit, the i-th argument had an illegal value.

Example:

```
C      Generate 100 values from the
C      Multivariate Students T distribution
       INTEGER LSTATE,N, MM
       PARAMETER (LSTATE=16,N=100,MM=10)
       INTEGER I,J,INFO,SEED(1),STATE(LSTATE)
       INTEGER LDC,LDX,M,DF
       DOUBLE PRECISION X(N,MM),XMU(MM),C(MM,MM)
C      Set array sizes
       LDC = MM
       LDX = N

C      Set the seed
       SEED(1) = 1234

C      Read in the distributional parameters
       READ(5,*) M,DF
       READ(5,*) (XMU(I),I=1,M)
       DO 20 I = 1,M
         READ(5,*) (C(I,J),J=1,M)
20     CONTINUE

C      Initialize the STATE vector
       CALL DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)
C      Generate N variates from the
C      Multivariate Students T distribution
       CALL DRANDMULTISTUDENTST(N,M,DF,XMU,C,LDC,STATE,X,LDX,INFO)

C      Print the results
       DO 40 I = 1,N
         WRITE(6,*) (X(I,J),J=1,M)
40     CONTINUE
```

DRANDMULTINORMALR / SRANDMULTINORMALR

Generates an array of random variates from a Multivariate Normal distribution using a reference vector initialized by DRANDMULTINORMALREFERENCE.

(Note that SRANDMULTINORMALR is the single precision version of DRANDMULTINORMALR. The argument lists of both routines are identical except that any double precision arguments of DRANDMULTINORMALR are replaced in SRANDMULTINORMALR by single precision arguments - type REAL in FORTRAN or type float in C).

DRANDMULTINORMALR (*N,REF,STATE,X,LDX,INFO*) [SUBROUTINE]

INTEGER N [Input]

On input: number of variates required.
Constraint: $N \geq 0$.

DOUBLE PRECISION REF(*) [Input]

On input: a reference vector generated by DRANDMULTINORMALREFERENCE.

INTEGER STATE(*) [Input/Output]

The *STATE* vector holds information on the state of the base generator being used and as such its minimum length varies. Prior to calling DRANDMULTINORMALR *STATE* must have been initialized. See Section 3.1.1 [Initialization of the Base Generators], for information on initialization of the *STATE* variable.
On input: the current state of the base generator.
On output: the updated state of the base generator.

DOUBLE PRECISION X(LDX,M) [Output]

On output: matrix of variates from the specified distribution.

INTEGER LDX [Input]

On input: leading dimension of X in the calling routine.
Constraint: $LDX \geq N$.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0.
If $INFO = -i$ on exit, the i-th argument had an illegal value.

Example:

```
C         Generate 100 values from the
C         Multivariate Normal distribution
          INTEGER LSTATE,N, MM
          PARAMETER (LSTATE=16,N=100,MM=10)
          INTEGER I,J,INFO,SEED(1),STATE(LSTATE)
          INTEGER LDC,LDX,M
          DOUBLE PRECISION X(N,MM),XMU(MM),C(MM,MM)
          INTEGER LREF
          DOUBLE PRECISION REF(1000)

C         Set array sizes
          LDC = MM
          LDX = N

C         Set the seed
          SEED(1) = 1234

C         Read in the distributional parameters
          READ(5,*) M
          READ(5,*) (XMU(I),I=1,M)
          DO 20 I = 1,M
            READ(5,*) (C(I,J),J=1,M)
20        CONTINUE

C         Initialize the STATE vector
          CALL DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)

C         Initialize the reference vector
          LREF = 1000
          CALL DRANDMULTINORMALREFERENCE(M,XMU,C,LDC,REF,LREF,INFO)

C         Generate N variates from the
C         Multivariate Normal distribution
          CALL DRANDMULTINORMALR(N,REF,STATE,X,LDX,INFO)
C         Print the results
          DO 40 I = 1,N
            WRITE(6,*) (X(I,J),J=1,M)
40        CONTINUE
```

DRANDMULTISTUDENTSTR / SRANDMULTISTUDENTSTR

Generates an array of random variates from a Multivariate Students T distribution using a reference vector initialized by DRANDMULTISTUDENTSTREFERENCE.

(Note that SRANDMULTISTUDENTSTR is the single precision version of DRANDMULTISTUDENTSTR. The argument lists of both routines are identical except that any double precision arguments of DRANDMULTISTUDENTSTR are replaced in SRANDMULTISTUDENTSTR by single precision arguments - type REAL in FORTRAN or type float in C).

DRANDMULTISTUDENTSTR (*N,REF,STATE,X,LDX,INFO*) [SUBROUTINE]

INTEGER N [Input]

On input: number of variates required.
Constraint: $N \geq 0$.

DOUBLE PRECISION REF(*) [Input]

On input: a reference vector generated by DRANDMULTISTUDENTSTREFERENCE.

INTEGER STATE(*) [Input/Output]

The *STATE* vector holds information on the state of the base generator being used and as such its minimum length varies. Prior to calling DRANDMULTISTUDENTSTR *STATE* must have been initialized. See Section 3.1.1 [Initialization of the Base Generators], for information on initialization of the *STATE* variable.
On input: the current state of the base generator.
On output: the updated state of the base generator.

DOUBLE PRECISION X(LDX,M) [Output]

On output: matrix of variates from the specified distribution.

INTEGER LDX [Input]

On input: leading dimension of X in the calling routine.
Constraint: $LDX \geq N$.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0.
If $INFO = -i$ on exit, the i-th argument had an illegal value.

Example:

```
C     Generate 100 values from the
C     Multivariate Students T distribution
      INTEGER LSTATE,N, MM
      PARAMETER (LSTATE=16,N=100,MM=10)
      INTEGER I,J,INFO,SEED(1),STATE(LSTATE)
      INTEGER LDC,LDX,M,DF
      DOUBLE PRECISION X(N,MM),XMU(MM),C(MM,MM)
      INTEGER LREF
      DOUBLE PRECISION REF(1000)

C     Set array sizes
      LDC = MM
      LDX = N

C     Set the seed
      SEED(1) = 1234

C     Read in the distributional parameters
      READ(5,*) M,DF
      READ(5,*) (XMU(I),I=1,M)
      DO 20 I = 1,M
        READ(5,*) (C(I,J),J=1,M)
20    CONTINUE

C     Initialize the STATE vector
      CALL DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)

C     Initialize the reference vector
      LREF = 1000
      CALL DRANDMULTISTUDENTSTREFERENCE(M,DF,XMU,C,LDC,REF,LREF,INFO)

C     Generate N variates from the
C     Multivariate Students T distribution
      CALL DRANDMULTISTUDENTSTR(N,REF,STATE,X,LDX,INFO)
C     Print the results
      DO 40 I = 1,N
        WRITE(6,*) (X(I,J),J=1,M)
40    CONTINUE
```

DRANDMULTINORMALREFERENCE / SRANDMULTINORMALREFERENCE

Initializes a reference vector for use with DRANDMULTINORMALR. Reference vector is for a Multivariate Normal distribution with probability density function, $f(X)$, where:

$$f(X) = \sqrt{\frac{|C^{-1}|}{(2\pi)^M}} e^{-(X-\mu)^T C^{-1} (X-\mu)},$$

where μ is the vector of means, XMU.

(Note that SRANDMULTINORMALREFERENCE is the single precision version of DRANDMULTINORMALREFERENCE. The argument lists of both routines are identi- cal except that any double precision arguments of DRANDMULTINORMALREFERENCE are replaced in SRANDMULTINORMALREFERENCE by single precision arguments - type REAL in FORTRAN or type float in C).

DRANDMULTINORMALREFERENCE (*M,XMU,C,LDC,REF,LREF,INFO*) [SUBROUTINE]

INTEGER M [Input]

On input: number of dimensions for the distribution.
Constraint: $M \geq 1$.

DOUBLE PRECISION XMU(M) [Input]

On input: vector of means for the distribution.

DOUBLE PRECISION C(LDC,M) [Input]

On input: variance / covariance matrix for the distribution.

INTEGER LDC [Input]

On input: leading dimension of C in the calling routine.
Constraint: $LDC \geq M$.

DOUBLE PRECISION REF(LREF) [Output]

On output: if *INFO* returns with a value of 0 then *REF* contains reference information required to generate values from a Multivariate Normal distribution using DRANDMULTINORMALR.

INTEGER LREF [Input/Output]

On input: either the length of the reference vector *REF*, or -1.
On output: if $LREF = -1$ on input, then *LREF* is set to the recommended length of the reference vector and the routine returns. Otherwise *LREF* is left unchanged.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. If $INFO = -i$ on exit, the i-th argument had an illegal value. If $INFO = 1$ on exit, then *LREF* has been set to the recommended length for the reference vector *REF*. If $INFO = 0$ then the reference vector, *REF*, has been successfully initialized.

Example:

```
C     Generate 100 values from the
C     Multivariate Normal distribution
      INTEGER LSTATE,N, MM
      PARAMETER (LSTATE=16,N=100,MM=10)
      INTEGER I,J,INFO,SEED(1),STATE(LSTATE)
      INTEGER LDC,LDX,M
      DOUBLE PRECISION X(N,MM),XMU(MM),C(MM,MM)
      INTEGER LREF
      DOUBLE PRECISION REF(1000)

C     Set array sizes
      LDC = MM
      LDX = N

C     Set the seed
      SEED(1) = 1234

C     Read in the distributional parameters
      READ(5,*) M
      READ(5,*) (XMU(I),I=1,M)
      DO 20 I = 1,M
        READ(5,*) (C(I,J),J=1,M)
20    CONTINUE

C     Initialize the STATE vector
      CALL DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)

C     Initialize the reference vector
      LREF = 1000
      CALL DRANDMULTINORMALREFERENCE(M,XMU,C,LDC,REF,LREF,INFO)

C     Generate N variates from the
C     Multivariate Normal distribution
      CALL DRANDMULTINORMALR(N,REF,STATE,X,LDX,INFO)
C     Print the results
      DO 40 I = 1,N
        WRITE(6,*) (X(I,J),J=1,M)
40    CONTINUE
```

DRANDMULTISTUDENTSTREFERENCE / SRANDMULTISTUDENTSTREFERENCE

Initializes a reference vector for use with DRANDMULTISTUDENTSTR. Reference vector is for a Multivariate Students T distribution with probability density function, $f(X)$, where:

$$f(X) = \frac{\Gamma\left(\frac{(\nu+M)}{2}\right)}{(\pi\nu)^{\frac{m}{2}}\Gamma(\frac{\nu}{2})\,|C|^{\frac{1}{2}}}\left(1+\frac{(X-\mu)^T C^{-1}(X-\mu)}{\nu}\right)^{-\frac{(\nu+M)}{2}},$$

where μ is the vector of means, *XMU* and ν is the degrees of freedom, *DF*.

(Note that SRANDMULTISTUDENTSTREFERENCE is the single precision version of DRANDMULTISTUDENTSTREFERENCE. The argument lists of both routines are identical except that any double precision arguments of DRANDMULTISTUDENTSTRE- FERENCE are replaced in SRANDMULTISTUDENTSTREFERENCE by single precision arguments - type REAL in FORTRAN or type float in C).

DRANDMULTISTUDENTSREFERENCE [SUBROUTINE]

(*M,DF,XMU,C,LDC,REF,LREF,INFO*)

INTEGER M [Input]

On input: number of dimensions for the distribution.
Constraint: $M \geq 1$.

INTEGER DF [Input]

On input: degrees of freedom.
Constraint: $DF > 2$.

DOUBLE PRECISION XMU(M) [Input]

On input: vector of means for the distribution.

DOUBLE PRECISION C(LDC,M) [Input]

On input: matrix defining the variance / covariance for the distribution. The variance / covariance matrix is given by $\frac{\nu}{\nu-2}C$, where ν are the degrees of freedom, *DF*.

INTEGER LDC [Input]

On input: leading dimension of *C* in the calling routine.
Constraint: $LDC \geq M$.

DOUBLE PRECISION REF(LREF) [Output]

On output: if *INFO* returns with a value of 0 then *REF* contains reference information required to generate values from a Multivariate Students T distribution using DRANDMULTISTUDENTSTR.

INTEGER LREF [Input/Output]

On input: either the length of the reference vector *REF*, or -1.
On output: if $LREF = -1$ on input, then *LREF* is set to the recommended length of the reference vector and the routine returns. Otherwise *LREF* is left unchanged.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. If *INFO* = $-i$ on exit, the i-th argument had an illegal value. If *INFO* = 1 on exit, then *LREF* has been set to the recommended length for the reference vector *REF*. If *INFO* = 0 then the reference vector, *REF*, has been successfully initialized.

Example:

```
C       Generate 100 values from the
C       Multivariate Students T distribution
        INTEGER LSTATE,N, MM
        PARAMETER (LSTATE=16,N=100,MM=10)
        INTEGER I,J,INFO,SEED(1),STATE(LSTATE)
        INTEGER LDC,LDX,M,DF
        DOUBLE PRECISION X(N,MM),XMU(MM),C(MM,MM)
        INTEGER LREF
        DOUBLE PRECISION REF(1000)

C       Set array sizes
        LDC = MM
        LDX = N

C       Set the seed
        SEED(1) = 1234

C       Read in the distributional parameters
        READ(5,*) M,DF
        READ(5,*) (XMU(I),I=1,M)
        DO 20 I = 1,M
          READ(5,*) (C(I,J),J=1,M)
20      CONTINUE

C       Initialize the STATE vector
        CALL DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)

C       Initialize the reference vector
        LREF = 1000
        CALL DRANDMULTISTUDENTSTREFERENCE(M,DF,XMU,C,LDC,REF,LREF,INFO)

C       Generate N variates from the
C       Multivariate Students T distribution
        CALL DRANDMULTISTUDENTSTR(N,REF,STATE,X,LDX,INFO)
C       Print the results
        DO 40 I = 1,N
          WRITE(6,*) (X(I,J),J=1,M)
40      CONTINUE
```

3.3.4 Discrete Multivariate Distributions

DRANDMULTINOMIAL / SRANDMULTINOMIAL

Generates a matrix of random variates from a Multinomial distribution with probability, $f(X)$, defined by:

$$f(X) = \frac{M!}{\prod_{i=1}^{K} X_i!} \prod_{i=1}^{K} p_i^{X_i},$$

where $X = \{X_1, X_2, \cdots, X_K\}$, $P = \{P_1, P_2, \cdots, P_K\}$, $\sum_{i=1}^{k} X_i = 1$ and $\sum_{i=1}^{k} P_i = 1$.

(Note that SRANDMULTINOMIAL is the single precision version of DRANDMULTINOMIAL. The argument lists of both routines are identical except that any double precision arguments of DRANDMULTINOMIAL are replaced in SRANDMULTINOMIAL by single precision arguments - type REAL in FORTRAN or type float in C).

DRANDMULTINOMIAL (*N,M,P,K,STATE,X,LDX,INFO*) [SUBROUTINE]

INTEGER N [Input]

On input: number of variates required.
Constraint: $N \geq 0$.

INTEGER M [Input]

On input: number of trials.
Constraint: $M \geq 0$.

DOUBLE PRECISION P(K) [Input]

On input: vector of probabilities for each of the K possible outcomes.
Constraint: $0 \leq P_i \leq 1, i = 1, 2, \cdots, K, \sum_{i=1}^{k} P_i = \mathbf{1}$.

INTEGER K [Input]

On input: number of possible outcomes.
Constraint: $K \geq 2$.

INTEGER STATE(*) [Input/Output]

The *STATE* vector holds information on the state of the base generator being used and as such its minimum length varies. Prior to calling DRANDBINOMIAL *STATE* must have been initialized. See Section 3.1.1 [Initialization of the Base Generators], for information on initialization of the *STATE* variable. On input: the current state of the base generator.
On output: the updated state of the base generator.

INTEGER X(LDX,K) [Output]

On output: matrix of variates from the specified distribution.

INTEGER LDX [Input]

On input: leading dimension of X in the calling routine.
Constraint: $LDX \geq N$.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0.
If $INFO = -i$ on exit, the i-th argument had an illegal value.

Example:

```
C Generate 100 values from the Multinomial  distribution
        INTEGER LSTATE,N,M
        PARAMETER (LSTATE=16,N=100,M=10)
        INTEGER I,J,INFO,SEED(1),STATE(LSTATE)
        INTEGER LDX,K
        INTEGER    X(N,M)
        DOUBLE  PRECISION  P(M)

C   Set  array sizes
        LDX  = N

C  Set  the  seed
        SEED(1)  =  1234

C Read in the distributional parameters
        READ(5,*) K
        READ(5,*)  (P(I),I=1,K)

C  Initialize  the  STATE vector
        CALL  DRANDINITIALIZE(1,1,SEED,1,STATE,LSTATE,INFO)

C   Generate   N   variates   from   the   Multinomial   distribution
        CALL DRANDMULTINOMIAL(N,M,P,K,STATE,X,LDX,INFO)

C  Print  the  results
        DO  20  I = 1,N
          WRITE(6,*)  (X(I,J),J=1,K)
        20 CONTINUE
```

4 References

- [1] D. E. Knuth, *The Art of Computer Programming* Addison-Wesley, 1997.
- [2] J. Banks, *Handbook on Simulation*, Wiley, 1998.
- [3] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, Chapter 5, CRC Press, 1996.
- [4] Chapter Introduction G05 - Random Number Generators *The NAG Fortran Library Manual, Mark 21* Numerical Algorithms Group, 2005.
- [5] N. M. Maclaren, The generation of multiple independent sequences of pseudorandom numbers, *Appl. Statist.*, 1989, 38, 351-359.
- [6] M. Matsumoto and T. Nishimura, Mersenne twister: A 623-dimensionally equidis- tributed uniform pseudorandom number generator, *ACM Transactions on Modelling and Computer Simulations*, 1998.
- [7] P. L'Ecuyer, Good parameter sets for combined multiple recursive random number generators, *Operations Research*, 1999, 47, 159-164.
- [8] P. L'Ecuyer and R. Simard, *TestU01: A Software Library in ANSI C for Empirical Testing of Random Number Generators*, Departement d'Informatique et de Recherche Operationnelle, Universite de Montreal, 2002. Software and user's guide available at http://www.iro.umontreal.ca/~lecuyer