Module 21.2: nag_rand_contin
Random Numbers from Continuous Distributions

`nag_rand_contin` provides procedures for generating sequences of independent pseudo-random numbers from continuous distributions.

Contents

Introduction

1 Terminology

This module is concerned with the generation of sequences of random numbers from continuous distributions.

Strictly speaking, the generated numbers are *pseudo-random* rather than true random numbers; however, their statistical properties — independence, randomness, etc. — are similar to those of true random numbers. In this module, the term 'random' will be used throughout, although strictly we mean 'pseudo-random'.

2 Continuous Distributions

This module provides procedures for the following continuous distributions: uniform, Normal (both univariate and multivariate), negative exponential, beta, and gamma (see Dagpunar [7], Kendall and Stuart [8], Knuth [9], Morgan [10] and Ripley [11] for further reading). The generated numbers are *real*.

The procedures are all *functions*, and may be *scalar valued* (returning one number per call), or *array valued* (returning several numbers per call).

An array-valued call with a result of dimension r will return the same sequence of random numbers as r consecutive scalar-valued calls, and will update the seed in the same way. For example, the following code fragments will give the same results in the array `v`.

```
v(1:r) = nag_rand_uniform( seed, r ) ! array-valued call
```

and

```
do i = 1, r
  v(i) = nag_rand_uniform( seed )    ! scalar-valued call
end do
```

3 Initialization of the Seed

All the procedures in this module make use of an argument `seed`, which is a structure of type `nag_seed_`*wp*. This must be initialized before use by calling the procedure `nag_rand_seed_set`. Both the type and its initialization procedure are defined by the module `nag_rand_util` (21.1), and described in its module document. However, they are also accessible via the `USE` statement for this module.

Procedure: nag_rand_uniform

1 Description

`nag_rand_uniform` returns random numbers from a uniform distribution. It is a generic function and the result may be either scalar or array valued depending on the presence of the argument `r`. The distribution of the random numbers may be over the interval $(0, 1)$ or (a, b).

2 Usage

`USE nag_rand_contin`

`[`*value* =`] nag_rand_uniform(seed  [,` *optional arguments*`])`

The function result is a scalar, of type real(kind=*wp*), over $(0, 1)$.

or

`[`*value* =`] nag_rand_uniform(seed, a, b  [,` *optional arguments*`])`

The function result is a scalar, of type real(kind=*wp*), over (a, b).

or

`[`*value* =`] nag_rand_uniform(seed, r  [,` *optional arguments*`])`

The function returns an array-valued result, of type real(kind=*wp*) and dimension (r), over $(0, 1)$.

or

`[`*value* =`] nag_rand_uniform(seed, a, b, r  [,` *optional arguments*`])`

The function returns an array-valued result, of type real(kind=*wp*) and dimension (r), over (a, b).

3 Arguments

3.1 Mandatory Arguments

seed — type(nag_seed_*wp*), intent(inout)

Input: the seed for generating random numbers (see the Module Introduction).

Output: an updated value of the seed.

a — real(kind=*wp*), intent(in)
b — real(kind=*wp*), intent(in)

Input: the end-points of the interval which defines the uniform distribution.

Default: `a` = 0.0 and `b` = 1.0.

Constraints: `a` ≠ `b`; `a` and `b` must both be included in the argument list, or both omitted.

r — integer, intent(in)

Input: the number of random numbers to be generated, if an array-valued result is required.

Note: this argument must be omitted if a scalar result is required. For `r` ≤ 0 an empty array will be returned.

3.2 Optional Argument

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 1 of this module document.

6 Further Comments

6.1 Mathematical Background

Let x be a uniformly distributed variate lying in the range a to b. The probability density function (PDF) of x is defined by

$$f(x) = \frac{1}{|b-a|} \qquad a \leq x \leq b.$$

This procedure returns $x_i = a + (b-a)u_i$, where u_i is a uniform (0,1) random number.

Procedure: nag_rand_normal

1 Description

`nag_rand_normal` returns random numbers from a Normal distribution with mean a and standard deviation b. By default, it uses the method of Brent [5] but provides a facility for the method of Box–Muller (Box and Muller [4]), if required. It is a generic function: the result may be either scalar or array valued depending on the presence of the argument `r`. For greater efficiency, the Box–Muller method should be used on vector machines.

2 Usage

```
USE nag_rand_contin
[value =] nag_rand_normal(seed  [, optional arguments])
```

The function result is a scalar of type real(kind=*wp*).

or

```
[value =] nag_rand_normal(seed, r  [, optional arguments])
```

The function returns an array-valued result of type real(kind=*wp*) and dimension (r).

3 Arguments

3.1 Mandatory Arguments

seed — type(nag_seed_*wp*), intent(inout)

Input: the seed for generating random numbers (see the Module Introduction).

Output: an updated value of the seed.

r — integer, intent(in)

Input: the number of random numbers to be generated, if an array-valued result is required.

Note: this argument must be omitted if a scalar result is required. For `r` ≤ 0 an empty array will be returned.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

a — real(kind=*wp*), intent(in), optional

b — real(kind=*wp*), intent(in), optional

Input: the mean a and standard deviation b of the Normal distribution.

Default: `a` = 0.0 and `b` = 1.0.

Constraints: `b` > 0.0.

brent — logical, intent(in), optional

Input: the method to be used.

If `brent` = `.true.`, Brent's method is used;

if `brent` = `.false.`, Box–Muller's method is used.

Default: `brent` = `.true.`.

Note: the random numbers from the two methods are *not* the same.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	**Description**
301	An input argument has an invalid value.

5 Examples of Usage

Assume that all relevant arguments have been declared correctly as described in Section 3, and that input and input/output arguments have been appropriately initialized. This first example illustrates how a call to this procedure returns a scalar result.

```
x = nag_rand_normal(seed)
```

This second example illustrates how a call to this procedure returns an array-valued result from Box–Muller's method.

```
v(1:r) = nag_rand_normal(seed,r,brent=.false.)
```

6 Further Comments

6.1 Mathematical Background

Let x be a random variable from a Normal distribution with mean a and standard deviation b. The probability density function (PDF) of x is defined by

$$f(x) = \frac{1}{b\sqrt{2\pi}} \exp\left(-\frac{(x-a)^2}{2b^2}\right)$$

where $b > 0$.

6.2 Algorithmic Detail

The Box–Muller method consists of first generating a pair of uniform (0,1) random numbers (u_1 and u_2, say) and then using trigonometric functions to transform the u_i to a pair of independent standard Normal deviates:

$$x_1 = (-2\ln u_1)^{1/2}\cos(2\pi u_2) \quad \text{and} \quad x_2 = (-2\ln u_1)^{1/2}\sin(2\pi u_2).$$

One of the x_i is then used for x (i.e., $x = a + bx_i$) and the other is stored for subsequent generation of a new x.

Since the Box–Muller method only uses transformations, a vector of deviates can be computed in parallel on a vector processing machine, thus making it very efficient. On scalar machines Brent's method, which uses a rejection technique, will be more efficient. For details of Brent's method see Brent [5].

Procedure: nag_rand_mv_normal

1 Description

`nag_rand_mv_normal` returns a vector of n random numbers $x^T = (x_1, x_2, \ldots, x_n)$ from a multivariate Normal distribution with mean vector a and covariance matrix C. The covariance matrix C or its Cholesky decomposition must be supplied.

2 Usage

```
USE nag_rand_contin
[value =] nag_rand_mv_normal(seed, a  [, optional arguments])
```

The function returns an array of random numbers of type real(kind=*wp*) and of dimension(n).

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '**x**(n)' is used in the argument descriptions to specify that the array `x` must have exactly n elements.

This procedure derives the value of the following problem parameter from the shape of the supplied arrays.

n — the dimension of the multivariate Normal distribution

3.1 Mandatory Arguments

seed — type(nag_seed_*wp*), intent(inout)

Input: the seed for generating random numbers (see the Module Introduction).

Output: an updated value of the seed.

a(n) — real(kind=*wp*), intent(in)

Input: the vector of means of the multivariate distribution.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

c(n, n) — real(kind=*wp*), intent(in), optional

Input: the covariance matrix of the multivariate distribution. Only the upper triangle needs to be supplied.

Constraints: C must be symmetric positive definite and `c` must be supplied if `chol` is not present.

chol($(n(n+1))/2$) — real(kind=*wp*), intent(inout), optional

Input: the Cholesky decomposition of the covariance matrix of the multivariate distribution, in packed storage. This decomposition may be obtained either from a previous call to this procedure with `c` present, or from a call to `nag_sym_lin_fac` (using packed storage with `uplo = 'l'`) from the module `nag_sym_lin_sys` (5.2).

Output: the Cholesky decomposition of `c`, in packed storage, if `c` is present.

Constraints: `chol` must be supplied, if `c` is *not* present.

rel_tol — real(kind=*wp*), intent(in), optional

Input: the maximum error in any element of C relative to the largest element of C. `rel_tol` is used in calculating the Cholesky decomposition of C.

Note: if `rel_tol` is less than `EPSILON(1.0_`*wp*`)`, then `EPSILON(1.0_`*wp*`)` is used instead.

Default: `rel_tol` = max(`EPSILON(1.0_`*wp*`)`,0.1/n).

Constraints: $0.0 \leq$ `rel_tol` $\leq 0.1/n$.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	**Description**
301	An input argument has an invalid value.
302	An array argument has an invalid shape.
303	Array arguments have inconsistent shapes.
305	Invalid absence of an optional argument.
320	The procedure was unable to allocate enough memory.

Warnings (error%level = 1):

error%code	**Description**
101	Optional argument is present but not needed.
	`rel_tol` is present when `c` is not present.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 2 of this module document.

6 Further Comments

6.1 Mathematical Background

Let X_i, $i = 1, \ldots, n$, be a vector of n variables with mean a and covariance matrix C. Suppose x_i, $i = 1, \ldots, n$, are the realization of the X_i. If C is non-singular or positive definite, the joint probability density function (PDF) of the elements of x is

$$f(x_1, x_2, \ldots, x_n) = (2\pi)^{-n/2} \mid C \mid^{-1/2} \exp\left(-\tfrac{1}{2}(x-a)^T C^{-1}(x-a)\right).$$

6.2 Algorithmic Detail

For any given positive definite matrix C, there exists a lower triangular matrix L such that $C = LL^T$. This procedure calculates L and returns $x = a + Lz$ given that $z^T = (z_1, z_2, \ldots, z_n)$ is a vector of independent standard normal random numbers. If both `c` and `chol` are specified, then this procedure returns the Cholesky matrix L in `chol`. However, if `c` is not given, then `chol` must be supplied and must contain the Cholesky matrix L.

6.3 Scaling

It is recommended that the diagonal elements of C should not differ too widely in order of magnitude. This may be achieved by scaling the variables, if necessary. The actual matrix decomposed is $C + E = LL^T$, where E is a diagonal matrix with small positive diagonal elements. This ensures that when C is singular or nearly singular, the Cholesky factor L corresponds to a positive definite covariance matrix that agrees with C within a tolerance determined by `rel_tol`.

6.4 Accuracy

The maximum absolute error in LL^T and hence in the covariance matrix of the resulting vectors is less than ($n\times$`rel_tol`$+(n+3)\times$`EPSILON(1.0_`wp`)`$/2$) times the maximum element of C. Under normal circumstances, the above will be small compared to sampling error.

6.5 Timing

The time taken by the procedure is proportional to n^3.

Procedure: nag_rand_beta

1 Description

`nag_rand_beta` returns random numbers from a beta distribution with parameters a and b. It is a generic function and the result may be either scalar or array valued depending on the presence of the argument `r`.

2 Usage

```
USE nag_rand_contin
[value =] nag_rand_beta(seed, a, b  [, optional arguments])
```

The function result is a scalar of type real(kind=*wp*).

or

```
[value =] nag_rand_beta(seed, a, b, r  [, optional arguments])
```

The function returns an array-valued result of type real(kind=*wp*) and dimension (r).

3 Arguments

3.1 Mandatory Arguments

seed — type(nag_seed_*wp*), intent(inout)

Input: the seed for generating random numbers (see the Module Introduction).

Output: an updated value of the seed.

a — real(kind=*wp*), intent(in)
b — real(kind=*wp*), intent(in)

Input: the parameters a and b of the beta distribution.

Constraints: `a` > 0.0 and `b` > 0.0.

r — integer, intent(in)

Input: the number of random numbers to be generated, if an array-valued result is required.

Note: this argument must be omitted if a scalar result is required. For `r` ≤ 0 an empty array will be returned.

3.2 Optional Argument

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.

5 Examples of Usage

Assume that all relevant arguments have been declared correctly as described in Section 3, and that input and input/output arguments have been appropriately initialized. This first example illustrates how a call to this procedure returns a scalar result.

```
x = nag_rand_beta(seed,a,b)
```

This second example illustrates how a call to this procedure returns an array-valued result.

```
v(1:r) = nag_rand_beta(seed,a,b,r)
```

6 Further Comments

6.1 Mathematical Background

Let x be a beta-distributed random variable with parameters a and b. The probability density function (PDF) of x is defined by

$$f(x) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} x^{a-1}(1-x)^{b-1} \qquad 0 \leq x < 1; \quad a, b > 0.$$

6.2 Algorithmic Detail

This procedure uses four different but alternative algorithms depending on the values of $\alpha = \max(a, b)$ and $\beta = \min(a, b)$.

The four algorithms used are as follows.

- For $\alpha < 0.5$ Jöhnk's algorithm is used (see Dagpunar [7]); this generates the beta variate as $u_1^{1/a}/(u_1^{1/a} + u_2^{1/b})$, where u_1 and u_2 are uniformly distributed random variates.
- For $\beta > 1$ the algorithm BB given in Cheng [6] is used. This involves the generation of an observation from a beta distribution of the second kind by the envelope rejection method using a log-logistic target distribution and then transforming it to a beta variate.
- For $\alpha > 1$ and $\beta < 1$ the switching algorithm given in Atkinson [2] is used. The two target distributions used are $f_1(x) = \beta x^\beta$ and $f_2(x) = \alpha(1-x)^{\beta-1}$, along with the approximation to the switching parameter of $t = (1-\beta)/(\alpha+1-\beta)$.
- In all other cases Cheng's BC algorithm (see Cheng [6]) is used with modifications suggested in Dagpunar [7]. This algorithm is similar to BB, which is used when $\beta > 1$, but it is tuned for small values of a and b.

Procedure: nag_rand_neg_exp

1 Description

`nag_rand_neg_exp` returns random numbers from a (negative) exponential distribution with mean a. It is a generic function: the result may be either scalar or array valued depending on the presence of the argument `r`.

2 Usage

```
USE nag_rand_contin
[value =] nag_rand_neg_exp(seed, a  [, optional arguments])
```

The function result is a scalar of type real(kind=*wp*).

or

```
[value =] nag_rand_neg_exp(seed, a, r  [, optional arguments])
```

The function returns an array-valued result of type real(kind=*wp*) and dimension (r).

3 Arguments

3.1 Mandatory Arguments

seed — type(nag_seed_*wp*), intent(inout)

Input: the seed for generating random numbers (see the Module Introduction).

Output: an updated value of the seed.

a — real(kind=*wp*), intent(in)

Input: the parameter a of the distribution.

Note: if `a` is negative, its absolute value is used.

Constraints: `a` $\neq 0.0$.

r — integer, intent(in)

Input: the number of random numbers to be generated, if an array-valued result is required.

Note: this argument must be omitted if a scalar result is required. For `r` ≤ 0 an empty array will be returned.

3.2 Optional Argument

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.

5 Examples of Usage

Assume that all relevant arguments have been declared correctly as described in Section 3, and that input and input/output arguments have been appropriately initialized. This first example illustrates how a call to this procedure returns a scalar result.

```
x = nag_rand_neg_exp(seed,a)
```

This second example illustrates how a call to this procedure returns an array-valued result.

```
v(1:r) = nag_rand_neg_exp(seed,a,r)
```

6 Further Comments

6.1 Mathematical Background

Let x be a random variable from an exponential distribution with parameter a. The PDF of x is defined as

$$f(x) = \frac{1}{a}\exp\left(-x/a\right) \quad x > 0.$$

This procedure returns the sequence of values $-a\ln(y)$, where y is a random number from a uniform distribution over (0,1).

Procedure: nag_rand_gamma

1 Description

nag_rand_gamma returns random numbers from a gamma distribution with parameters a and b. It is a generic function: the result may be either scalar or array valued depending on the presence of the argument r.

2 Usage

```
USE nag_rand_contin
[value =] nag_rand_gamma(seed, a, b  [, optional arguments])
```

The function result is a scalar of type real(kind=*wp*).

or

```
[value =] nag_rand_gamma(seed, a, b, r  [, optional arguments])
```

The function returns an array-valued result of type real(kind=*wp*) and dimension (r).

3 Arguments

3.1 Mandatory Arguments

seed — type(nag_seed_*wp*), intent(inout)

Input: the seed for generating random numbers (see the Module Introduction).

Output: an updated value of the seed.

a — real(kind=*wp*), intent(in)
b — real(kind=*wp*), intent(in)

Input: the parameters a and b of the gamma distribution.

Constraints: a > 0.0 and b > 0.0.

r — integer, intent(in)

Input: the number of random numbers to be generated, if a vector-valued result is required.

Note: this argument must be omitted if a scalar result is required. For r ≤ 0 an empty array will be returned.

3.2 Optional Argument

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document nag_error_handling (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to nag_set_error before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.

5 Examples of Usage

Assume that all relevant arguments have been declared correctly as described in Section 3, and that input and input/output arguments have been appropriately initialized. This first example illustrates how a call to this procedure returns a scalar result.

```
x = nag_rand_gamma(seed,a,b)
```

This second example illustrates how a call to this procedure returns an array-valued result.

```
v(1:r)= nag_rand_gamma(seed,a,b,r)
```

6 Further Comments

6.1 Mathematical Background

Let x be a gamma-distributed variable. The probability density function (PDF) of x is defined by

$$f(x) = \frac{1}{b^a\Gamma(a)}x^{a-1}\exp(-x/b) \qquad 0 \le x \le 1; \quad a, b > 0.$$

6.2 Algorithmic Detail

This procedure uses one of the following three algorithms to generate random numbers depending upon the value of a.

- For $a < 1$ a switching algorithm described in Dagpunar [7] (called G6) is used. The target distributions are $f_1(x) = cax^{a-1}/t^a$ and $f_2(x) = (1-c)e^{-(x-t)}$, where $c = t(t + ae^{-t})$, and the switching parameter, t, is taken as $1 - a$. This is similar to Ahrens and Dieter's GS algorithm (Ahrens and Dieter [1]) in which $t = 1$.
- For $a = 1$ the gamma distribution reduces to the exponential distribution and so the method based on the logarithmic transformation of a uniform random variate is used.
- For $a > 1$ the algorithm given in Best [3] is used; this is based on using a Student's t-distribution with two degrees of freedom as the target distribution in an envelope rejection method.

Example 1: Generation of Repeatable and Non-repeatable Sequences of Random Numbers from a Uniform Distribution

This example program shows how `nag_rand_seed_set` and `nag_rand_uniform` may be used to give repeatable and non-repeatable sequences of random numbers from a uniform distribution. `nag_rand_uniform` is called both as an array-valued function and as a scalar-valued function.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_rand_contin_ex01

  ! Example Program Text for nag_rand_contin
  ! NAG fl90, Release 4. NAG Copyright 2000.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_out
  USE nag_rand_contin, ONLY : nag_rand_uniform, nag_rand_seed_set, &
   nag_seed_wp => nag_seed_dp
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: r = 5
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  CHARACTER (*), PARAMETER :: fmt = '(1X,8F8.4)'
  ! .. Local Scalars ..
  INTEGER :: i, k
  TYPE (nag_seed_wp) :: seed
  ! .. Local Arrays ..
  REAL (wp) :: x(r), y(r), z(r)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) 'Example Program Results for nag_rand_contin_ex01'
  WRITE (nag_std_out,*)

  ! repeatable sequence of random numbers

  WRITE (nag_std_out,*) &
   'repeatable sequence of random numbers from a uniform distribution'
  WRITE (nag_std_out,*) 'using array-valued function'

  k = 0

  CALL nag_rand_seed_set(seed,k)

  x = nag_rand_uniform(seed,r)
  WRITE (nag_std_out,fmt) x
  WRITE (nag_std_out,*)

  WRITE (nag_std_out,*) 'the same sequence using scalar-valued function'

  k = 0

  CALL nag_rand_seed_set(seed,k)

  DO i = 1, r
    y(i) = nag_rand_uniform(seed)
  END DO
  WRITE (nag_std_out,fmt) y
```

```
    WRITE (nag_std_out,*)

    ! non-repeatable sequence of random numbers

    WRITE (nag_std_out,*) 'non-repeatable sequence of random &
     &numbers from a uniform distribution'

    CALL nag_rand_seed_set(seed)

    z = nag_rand_uniform(seed,r)
    WRITE (nag_std_out,fmt) z

  END PROGRAM nag_rand_contin_ex01
```

2 Program Data

None.

3 Program Results

```
 Example Program Results for nag_rand_contin_ex01

 repeatable sequence of random numbers from a uniform distribution
 using array-valued function
   0.7951  0.2257  0.3713  0.2250  0.8787

 the same sequence using scalar-valued function
   0.7951  0.2257  0.3713  0.2250  0.8787

 non-repeatable sequence of random numbers from a uniform distribution
   0.4829  0.3082  0.1763  0.6258  0.9510
```

Example 2: Generation of a Vector of Random Numbers from a Multivariate Normal Distribution

This example illustrates the use of `nag_rand_mv_normal` for generating a vector of random numbers from a bivariate Normal distribution with means $\mu_1 = 1.0$, $\mu_2 = 2.0$; variances $\sigma_{11} = 2.0$, $\sigma_{22} = 3.0$; and covariances $\sigma_{12} = \sigma_{21} = 1.0$. The Cholesky decomposition of the given covariance matrix is calculated in the first call and it is used in other subsequent calls.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_rand_contin_ex02

  ! Example Program Text for nag_rand_contin
  ! NAG fl90, Release 4. NAG Copyright 2000.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_write_mat, ONLY : nag_write_tri_mat
  USE nag_rand_contin, ONLY : nag_rand_mv_normal, nag_rand_seed_set, &
   nag_seed_wp => nag_seed_dp
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  CHARACTER (*), PARAMETER :: fmt = '(/1x,a/(1X, 2F5.1))'
  ! .. Local Scalars ..
  INTEGER :: i, k, n, nw, r
  TYPE (nag_seed_wp) :: seed
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: a(:), c(:,:), chol(:), x(:,:)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) 'Example Program Results for nag_rand_contin_ex02'

  READ (nag_std_in,*)          ! skip heading in data file
  READ (nag_std_in,*) n        ! number of variables
  READ (nag_std_in,*) r        ! number of random number sequences

  nw = (n*(n+1))/2

  ALLOCATE (a(n),c(n,n),chol(nw),x(r,n)) ! Allocate storage

  READ (nag_std_in,*) a        ! mean vector
  READ (nag_std_in,*) (c(i,i:),i=1,n) ! covariance matrix

  k = 0

  CALL nag_rand_seed_set(seed,k)

  x(1,:) = nag_rand_mv_normal(seed,a,c=c,rel_tol=0.01_wp,chol=chol)

  DO i = 2, r
    x(i,:) = nag_rand_mv_normal(seed,a,chol=chol)
  END DO

  WRITE (nag_std_out,fmt) 'Mean Vector', a
```

```
      CALL nag_write_tri_mat('u',c,title='Covariance Matrix')

      WRITE (nag_std_out,'(/1x,a/(1X, 2F8.4))') 'Random Number Sequence', &
       (x(i,:),i=1,r)

      DEALLOCATE (a,c,chol,x)      ! Deallocate storage

    END PROGRAM nag_rand_contin_ex02
```

2 Program Data

```
Example Program Data for nag_rand_contin_ex02
  2               : number of variables
  5               : number of random number sequences
  1.0  2.0        : End of mean vector (A)
  2.0  1.0
       3.0        : End of covariance matrix (C)
```

3 Program Results

```
 Example Program Results for nag_rand_contin_ex02

 Mean Vector
   1.0  2.0
 Covariance Matrix
       2.0000     1.0000
                  3.0000

 Random Number Sequence
   1.7697  4.4481
   3.2678  3.0583
   3.1769  2.3651
  -0.1055  1.8395
   1.2933 -0.1850
```

Additional Examples

Not all example programs supplied with NAG *fl*90 appear in full in this module document. The following additional examples, associated with this module, are available.

`nag_rand_contin_ex03`

Generation of real random numbers from a uniform distribution with end-points (a, b) or $(0, 1)$, using both scalar-valued and array-valued function calls.

`nag_rand_contin_ex04`

Generation of real random numbers from a negative exponential distribution with known parameter, using both scalar-valued and array-valued function calls.

`nag_rand_contin_ex05`

Generation of real random numbers from a beta distribution with known parameters, using both scalar-valued and array-valued function calls.

`nag_rand_contin_ex06`

Generation of real random numbers from a gamma distribution with known parameters, using both scalar-valued and array-valued function calls.

`nag_rand_contin_ex07`

Generation of real random numbers from a Normal distribution with known parameters using both Brent and Box–Muller methods, and scalar-valued function calls.

`nag_rand_contin_ex08`

Generation of real random numbers from a Normal distribution with known parameters, using both Brent and Box–Muller methods, and array-valued function calls.

References

[1] Ahrens J H and Dieter U (1989) A convenient sampling method with bounded computation times for Poisson distributions *Amer. J. Math. Management Sci.* 1–13

[2] Atkinson A C (1979) A family of switching algorithms for the computer generation of beta random variates *Biometrika* **66** 141–5

[3] Best D J (1978) Letter to the Editor *Appl. Statist.* **29** 181

[4] Box G E P and Muller M E (1958) A note on the generation of random normal deviates *Ann. Math. Statist.* **29** 610–611

[5] Brent R P (1974) Algorithm 488 *Comm. ACM* 704

[6] Cheng R C H (1978) Generating beta variates with nonintegral shape parameters *Comm. ACM* **21** 317–322

[7] Dagpunar J (1988) *Principles of Random Variate Generation* Oxford University Press

[8] Kendall M G and Stuart A (1976) *The Advanced Theory of Statistics (Volume 3)* Griffin (3rd Edition)

[9] Knuth D E (1981) *The Art of Computer Programming (Volume 2)* Addison-Wesley (2nd Edition)

[10] Morgan B J T (1984) *Elements of Simulation* Chapman and Hall

[11] Ripley B D (1987) *Stochastic Simulation* Wiley