NAG Library Function Document

nag_rand_copula_normal (g05rdc)

1 Purpose

nag_rand_copula_normal (g05rdc) sets up a reference vector and generates an array of pseudorandom numbers from a Normal (Gaussian) copula with covariance matrix C.

2 Specification

```
#include <nag.h>
#include <nagg05.h>
void nag_rand_copula_normal (Nag_OrderType order, Nag_ModeRNG mode,
     Integer n, Integer m, const double c[], Integer pdc, double r[],
     Integer lr, Integer state[], double x[], Integer pdx, NagError *fail)
```

3 Description

The Gaussian copula, G, is defined by

$$G(u_1, u_2, \ldots, u_m; C) = \Phi_C\left(\phi^{-1}_{C_{11}}(u_1), \phi^{-1}_{C_{22}}(u_2), \ldots, \phi^{-1}_{C_{mm}}(u_m)\right)$$

where m is the number of dimensions, Φ_C is the multivariate Normal density function with mean zero and covariance matrix C and $\phi^{-1}_{C_{ii}}$ is the inverse of the univariate Normal density function with mean zero and variance C_{ii}.

nag_rand_matrix_multi_normal (g05rzc) is used to generate a vector from a multivariate Normal distribution and nag_prob_normal (g01eac) is used to convert each element of that vector into a uniformly distributed value between zero and one.

One of the initialization functions nag_rand_init_repeatable (g05kfc) (for a repeatable sequence if computed sequentially) or nag_rand_init_nonrepeatable (g05kgc) (for a non-repeatable sequence) must be called prior to the first call to nag_rand_copula_normal (g05rdc).

4 References

Nelsen R B (1998) *An Introduction to Copulas. Lecture Notes in Statistics 139* Springer

Sklar A (1973) Random variables: joint distribution functions and copulas *Kybernetika* **9** 499–460

5 Arguments

1: **order** – Nag_OrderType *Input*

On entry: the **order** argument specifies the two-dimensional storage scheme being used, i.e., row-major ordering or column-major ordering. C language defined storage is specified by **order** = Nag_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

Constraint: **order** = Nag_RowMajor or Nag_ColMajor.

2: **mode** – Nag_ModeRNG *Input*

On entry: a code for selecting the operation to be performed by the function.

mode = Nag_InitializeReference
Set up reference vector only.

mode = Nag_GenerateFromReference
Generate variates using reference vector set up in a prior call to nag_rand_copula_normal (g05rdc).

mode = Nag_InitializeAndGenerate
Set up reference vector and generate variates.

Constraint: **mode** = Nag_InitializeReference, Nag_GenerateFromReference or Nag_InitializeAndGenerate.

3: **n** – Integer *Input*

On entry: n, the number of random variates required.

Constraint: $\mathbf{n} \geq 0$.

4: **m** – Integer *Input*

On entry: m, the number of dimensions of the distribution.

Constraint: $\mathbf{m} > 0$.

5: **c**[dim] – const double *Input*

Note: the dimension, *dim*, of the array **c** must be at least $\mathbf{pdc} \times \mathbf{m}$.

The (i,j)th element of the matrix C is stored in

$\mathbf{c}[(j-1) \times \mathbf{pdc} + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{c}[(i-1) \times \mathbf{pdc} + j - 1]$ when **order** = Nag_RowMajor.

On entry: the covariance matrix of the distribution. Only the upper triangle need be set.

Constraint: C must be positive semidefinite to ***machine precision***.

6: **pdc** – Integer *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) in the array **c**.

Constraint: $\mathbf{pdc} \geq \mathbf{m}$.

7: **r**[**lr**] – double *Communication Array*

On entry: if **mode** = Nag_GenerateFromReference, the reference vector as set up by nag_rand_copula_normal (g05rdc) in a previous call with **mode** = Nag_InitializeReference or Nag_InitializeAndGenerate.

On exit: if **mode** = Nag_InitializeReference or Nag_InitializeAndGenerate, the reference vector that can be used in subsequent calls to nag_rand_copula_normal (g05rdc) with **mode** = Nag_GenerateFromReference.

8: **lr** – Integer *Input*

On entry: the dimension of the array **r**. If **mode** = Nag_GenerateFromReference, it must be the same as the value of **lr** specified in the prior call to nag_rand_copula_normal (g05rdc) with **mode** = Nag_InitializeReference or Nag_InitializeAndGenerate.

Constraint: $\mathbf{lr} \geq \mathbf{m} \times (\mathbf{m}+1) + 1$.

9: **state**[dim] – Integer *Communication Array*

Note: the dimension, dim, of this array is dictated by the requirements of associated functions that must have been previously called. This array MUST be the same array passed as argument **state** in the previous call to nag_rand_init_repeatable (g05kfc) or nag_rand_init_nonrepeatable (g05kgc).

On entry: contains information on the selected base generator and its current state.

On exit: contains updated information on the state of the generator.

10: **x**[*dim*] – double *Output*

Note: the dimension, *dim*, of the array **x** must be at least

max(1, **pdx** × **m**) when **order** = Nag_ColMajor;
max(1, **n** × **pdx**) when **order** = Nag_RowMajor.

Where **X**(*i*, *j*) appears in this document, it refers to the array element

x[(*j* − 1) × **pdx** + *i* − 1] when **order** = Nag_ColMajor;
x[(*i* − 1) × **pdx** + *j* − 1] when **order** = Nag_RowMajor.

On exit: the array of values from a multivariate Gaussian copula, with **X**(*i*, *j*) holding the *j*th dimension for the *i*th variate.

11: **pdx** – Integer *Input*

On entry: the stride used in the array **x**.

Constraints:

if **order** = Nag_ColMajor, **pdx** ≥ **n**;
if **order** = Nag_RowMajor, **pdx** ≥ **m**.

12: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_INT

On entry, **lr** is not large enough, **lr** = ⟨*value*⟩: minimum length required = ⟨*value*⟩.

On entry, **m** = ⟨*value*⟩.
Constraint: **m** > 0.

On entry, **n** = ⟨*value*⟩.
Constraint: **n** ≥ 0.

NE_INT_2

On entry, **pdc** = ⟨*value*⟩ and **m** = ⟨*value*⟩.
Constraint: **pdc** ≥ **m**.

On entry, **pdx** = ⟨*value*⟩ and **m** = ⟨*value*⟩.
Constraint: **pdx** ≥ **m**.

On entry, **pdx** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: **pdx** ≥ **n**.

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

NE_INVALID_STATE

On entry, **state** vector has been corrupted or not initialized.

NE_POS_DEF

On entry, the covariance matrix C is not positive semidefinite to ***machine precision***.

NE_PREV_CALL

m is not the same as when **r** was set up in a previous call.
Previous value of **m** $= \langle value \rangle$ and **m** $= \langle value \rangle$.

7 Accuracy

See Section 7 in nag_rand_matrix_multi_normal (g05rzc) for an indication of the accuracy of the underlying multivariate Normal distribution.

8 Parallelism and Performance

nag_rand_copula_normal (g05rdc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_rand_copula_normal (g05rdc) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

The time taken by nag_rand_copula_normal (g05rdc) is of order nm^3.

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 $C + E = LL^{\mathrm{T}}$, where E is a diagonal matrix with small positive diagonal elements. This ensures that, even when C is singular, or nearly singular, the Cholesky factor L corresponds to a positive definite covariance matrix that agrees with C within ***machine precision***.

10 Example

This example prints ten pseudorandom observations from a Normal copula with covariance matrix

$$\begin{bmatrix} 1.69 & 0.39 & -1.86 & 0.07 \\ 0.39 & 98.01 & -7.07 & -0.71 \\ -1.86 & -7.07 & 11.56 & 0.03 \\ 0.07 & -0.71 & 0.03 & 0.01 \end{bmatrix},$$

generated by nag_rand_copula_normal (g05rdc). All ten observations are generated by a single call to nag_rand_copula_normal (g05rdc) with **mode** = Nag_InitializeAndGenerate. The random number generator is initialized by nag_rand_init_repeatable (g05kfc).

10.1 Program Text

```
/* nag_rand_copula_normal (g05rdc) Example Program.
 *
 * Copyright 2008, Numerical Algorithms Group.
 *
 * Mark 9, 2009.
 */
/* Pre-processor includes */
#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg05.h>
```

```
#define X(I, J) x[(order == Nag_ColMajor)?(J*pdx + I):(I*pdx + J)]
#define C(I, J) c[(order == Nag_ColMajor)?(J*pdc + I):(I*pdc + J)]

int main(void)
{
  /* Integer scalar and array declarations */
  Integer       exit_status = 0;
  Integer       i, j, lstate, lr, x_size;
  Integer       *state = 0;
  Integer       pdx;

  /* NAG structures */
  NagError      fail;
  Nag_ModeRNG   mode;

  /* Double scalar and array declarations */
  double        *r = 0, *x = 0;

  /* Use column major order */
  Nag_OrderType order = Nag_RowMajor;

  /* Set the number of variables and variates */
  Integer       m = 4;
  Integer       n = 10;

  /* Input the covariance matrix */
  double        c[] = { 1.69e0, 0.39e0, -1.86e0, 0.07e0,
                        0.39e0, 98.01e0, -7.07e0, -0.71e0,
                        -1.86e0, -7.07e0, 11.56e0, 0.03e0,
                        0.07e0, -0.71e0, 0.03e0, 0.01e0 };
  Integer       pdc = 4;

  /* Choose the base generator */
  Nag_BaseRNG   genid = Nag_Basic;
  Integer       subid = 0;

  /* Set the seed */
  Integer       seed[] = { 1762543 };
  Integer       lseed = 1;

  /* Initialise the error structure */
  INIT_FAIL(fail);

  printf(
          "nag_rand_copula_normal (g05rdc) Example Program Results\n\n");

  /* Get the length of the state array */
  lstate = -1;
  nag_rand_init_repeatable(genid, subid, seed, lseed, state, &lstate, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_rand_init_repeatable (g05kfc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

  pdx = (order == Nag_ColMajor)?n:m;
  x_size = (order == Nag_ColMajor)?pdx * m:pdx * n;

  /* Calculate the size of the reference vector */
  lr = m*m+m+1;

  /* Allocate arrays */
  if (!(r = NAG_ALLOC(lr, double)) ||
      !(x = NAG_ALLOC(x_size, double)) ||
      !(state = NAG_ALLOC(lstate, Integer)))
    {
      printf("Allocation failure\n");
```

```
      exit_status = -1;
      goto END;
    }

  /* Initialise the generator to a repeatable sequence */
  nag_rand_init_repeatable(genid, subid, seed, lseed, state, &lstate, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_rand_init_repeatable (g05kfc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

  /* Set up reference vector and generate variates */
  mode = Nag_InitializeAndGenerate;
  nag_rand_copula_normal(order, mode, n, m, c, pdc, r, lr, state, x, pdx,
                         &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_rand_copula_normal (g05rdc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

  /* Display the results */
  for (i = 0; i < n; i++)
    {
      printf("  ");
      for (j = 0; j < m; j++)
        printf("%9.4f%s", X(i, j), (j+1)%10?" ":"\n");
      if (m%10) printf("\n");
    }

 END:
  NAG_FREE(r);
  NAG_FREE(x);
  NAG_FREE(state);

  return exit_status;
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_rand_copula_normal (g05rdc) Example Program Results

     0.6364    0.0517    0.4137    0.8817
     0.1065    0.2461    0.7993    0.3806
     0.7460    0.6313    0.2708    0.5421
     0.7983    0.0564    0.6868    0.9234
     0.1046    0.5790    0.8533    0.2208
     0.4925    0.2784    0.3513    0.5158
     0.3843    0.2349    0.9472    0.7801
     0.7871    0.9941    0.9403    0.2044
     0.4982    0.9015    0.7176    0.2914
     0.6717    0.5359    0.5961    0.4487
```