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

```c
#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

\[
G(u_1, u_2, \ldots, u_m; C) = \Phi_C \left( \phi_{C_1}^{-1}(u_1), \phi_{C_2}^{-1}(u_2), \ldots, \phi_{C_m}^{-1}(u_m) \right)
\]

where \( m \) is the number of dimensions, \( \Phi_C \) is the multivariate Normal density function with mean zero and covariance matrix \( C \) and \( \phi_{C_i}^{-1} \) is the inverse of the univariate Normal density function with mean zero and variance \( C_{ii} \).

nag_rand_matrix_multi_normal (g05zpc) 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 (g05kjc) (for a non-repeatable sequence) must be called prior to the first call to nag_rand_copula_normal (g05rdc).

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

5 Arguments

1: \( \text{order} \) – Nag_OrderType
   \( \text{Input} \)
   On entry: the \( \text{order} \) 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 \( \text{order} = \text{Nag_RowMajor} \). See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.
   Constraint: \( \text{order} = \text{Nag_RowMajor} \) or \( \text{Nag_ColMajor} \).

2: \( \text{mode} \) – Nag_ModeRNG
   \( \text{Input} \)
   On entry: a code for selecting the operation to be performed by the function.
   \( \text{mode} = \text{Nag_InitializeReference} \)
   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: n \geq 0.

4: m – Integer
Input
On entry: m, the number of dimensions of the distribution.
Constraint: m > 0.

5: c[dim] – const double
Input
Note: the dimension, dim, of the array c must be at least pdc \times m.
The (i,j)th element of the matrix C is stored in
\begin{align*}
c((j-1) \times \text{pdc} + i - 1) & \quad \text{when order} = \text{Nag\_ColMajor}; \\
c((i-1) \times \text{pdc} + j - 1) & \quad \text{when order} = \text{Nag\_RowMajor}.
\end{align*}
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: pdc \geq m.

7: r[\text{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: lr \geq m \times (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 (g05ffc) or nag_rand_init_nonrepeatable (g05kfc).
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: \( x[\text{dim}] \) – double

Note: the dimension, \( \text{dim} \), of the array \( x \) must be at least
\[
\max(1, \text{pdx} \times m) \text{ when } \text{order} = \text{Nag\_ColMajor};
\]
\[
\max(1, n \times \text{pdx}) \text{ when } \text{order} = \text{Nag\_RowMajor}.
\]

Where \( X(i,j) \) appears in this document, it refers to the array element
\[
x[(j - 1) \times \text{pdx} + i - 1] \text{ when } \text{order} = \text{Nag\_ColMajor};
\]
\[
x[(i - 1) \times \text{pdx} + j - 1] \text{ when } \text{order} = \text{Nag\_RowMajor}.
\]

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: \( \text{pdx} \) – Integer

On entry: the stride used in the array \( x \).

Constraints:
\[
\text{if } \text{order} = \text{Nag\_ColMajor}, \text{pdx} \geq n;
\]
\[
\text{if } \text{order} = \text{Nag\_RowMajor}, \text{pdx} \geq m.
\]

12: \( \text{fail} \) – NagError*

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

6 Error Indicators and Warnings

**NE_ALLOC_FAIL**
Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

**NE_BAD_PARAM**
On entry, argument \( \langle \text{value} \rangle \) had an illegal value.

**NE_INT**
On entry, \( \text{lr} \) is not large enough, \( \text{lr} = \langle \text{value} \rangle \): minimum length required = \( \langle \text{value} \rangle \).

On entry, \( m = \langle \text{value} \rangle \).
Constraint: \( m > 0 \).

On entry, \( n = \langle \text{value} \rangle \).
Constraint: \( n \geq 0 \).

**NE_INT_2**
On entry, \( \text{pdc} = \langle \text{value} \rangle \) and \( m = \langle \text{value} \rangle \).
Constraint: \( \text{pdc} \geq m \).

On entry, \( \text{pdx} = \langle \text{value} \rangle \) and \( m = \langle \text{value} \rangle \).
Constraint: \( \text{pdx} \geq m \).

On entry, \( \text{pdx} = \langle \text{value} \rangle \) and \( n = \langle \text{value} \rangle \).
Constraint: \( \text{pdx} \geq n \).

**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.
An unexpected error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in the Essential Introduction for further information.

**NE_INVALID_STATE**

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

**NE_NO_LICENCE**

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in the Essential Introduction for further information.

**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 $m = \langle\text{value}\rangle$ and $m = \langle\text{value}\rangle$.

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

nag_randCopulaNormal (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 X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users’ Note for your implementation for any additional implementation-specific information.

9 Further Comments

The time taken by nag_randCopulaNormal (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 $C + E = LL^T$, 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_randCopulaNormal (g05rdc). All ten observations are generated by a single call to nag_randCopulaNormal (g05rdc) with mode = Nag_InitializeAndGenerate. The random number generator is initialized by nag_randInitRepeatable (g05kfc).
10.1 Program Text

/* nag_rand_copula_normal (g05rdc) Example Program.

* Copyright 2014 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\n", fail.message);
        exit_status = 1;
        goto END;
    }
    /* Call the routine */
    lstate = -1;
    nag_rand_copula_normal(order, m, n, c, pdc, &r, &fail);
    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_rand_copula_normal (g05rdc).\n\n", fail.message);
        exit_status = 1;
        goto END;
    }
    /* Print the results */
    printf("nag_rand_copula_normal (g05rdc) Example Program Results\n\n");
    for (i = 0; i < m; i++)
    {
        for (j = 0; j < n; j++)
        {
            printf("%f %f\n", x[i*n + j], x[(i+1)*n + j]);
        }
    }
    printf("\n\n");
    return 1;
}

END:
return exit_status;

Mark 25

10.1 Program Text

/* nag_rand_copula_normal (g05rdc) Example Program.

* Copyright 2014 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\n", fail.message);
        exit_status = 1;
        goto END;
    }
    /* Call the routine */
    lstate = -1;
    nag_rand_copula_normal(order, m, n, c, pdc, &r, &fail);
    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_rand_copula_normal (g05rdc).\n\n", fail.message);
        exit_status = 1;
        goto END;
    }
    /* Print the results */
    printf("nag_rand_copula_normal (g05rdc) Example Program Results\n\n");
    for (i = 0; i < m; i++)
    {
        for (j = 0; j < n; j++)
        {
            printf("%f %f\n", x[i*n + j], x[(i+1)*n + j]);
        }
    }
    printf("\n\n");
    return 1;
}

END:
return exit_status;
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("\n");
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

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.6364</td>
<td>0.0517</td>
<td>0.4137</td>
<td>0.8817</td>
</tr>
<tr>
<td>0.1065</td>
<td>0.2461</td>
<td>0.7993</td>
<td>0.3806</td>
</tr>
<tr>
<td>0.7460</td>
<td>0.6313</td>
<td>0.2708</td>
<td>0.5421</td>
</tr>
<tr>
<td>0.7983</td>
<td>0.0564</td>
<td>0.6868</td>
<td>0.9234</td>
</tr>
<tr>
<td>0.1046</td>
<td>0.5790</td>
<td>0.8533</td>
<td>0.2208</td>
</tr>
<tr>
<td>0.4925</td>
<td>0.2784</td>
<td>0.3513</td>
<td>0.5158</td>
</tr>
<tr>
<td>0.3843</td>
<td>0.2349</td>
<td>0.9472</td>
<td>0.7801</td>
</tr>
<tr>
<td>0.7871</td>
<td>0.9941</td>
<td>0.9403</td>
<td>0.2044</td>
</tr>
<tr>
<td>0.4982</td>
<td>0.9015</td>
<td>0.7176</td>
<td>0.2914</td>
</tr>
<tr>
<td>0.6717</td>
<td>0.5359</td>
<td>0.5961</td>
<td>0.4487</td>
</tr>
</tbody>
</table>