NAG Library Function Document

nag_rand_copula_frank (g05rjc)

1 Purpose

nag_rand_copula_frank (g05rjc) generates pseudorandom uniform variates with joint distribution of a Frank Archimedean copula.

2 Specification

#include <nag.h>
#include <nagg05.h>
void nag_rand_copula_frank (Nag_OrderType order, Integer state[], double theta, Integer n, Integer m, double x[], Integer pdx, Integer sdx, NagError *fail)

3 Description

Generates \( n \) pseudorandom uniform \( m \)-variates whose joint distribution is the Frank Archimedean copula \( C_\theta \), given by

\[
C_\theta = \frac{1}{\theta} \ln \left[ 1 + \frac{(e^{-\theta u_1} - 1)(e^{-\theta u_2} - 1) \cdots (e^{-\theta u_m} - 1)}{(e^{-\theta} - 1)^{m-1}} \right],
\]

where \( \theta \in (0, \infty), \quad u_j \in (0, 1), \quad j = 1, \ldots, m; \)

with the special case:

\( C_\infty = \min(u_1, u_2, \ldots, u_m) \), the Fréchet–Hoeffding upper bound.

The generation method uses mixture of powers.

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_frank (g05rjc).

4 References


5 Arguments

1: \( \text{order} \) – Nag_OrderType

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: \( \text{state}[\text{dim}] \) – Integer

Communication Array

Note: the dimension, \( \text{dim} \), 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 \( \text{state} \) 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.

3: theta – double
On entry: \( \theta \), the copula parameter.
Constraint: \( \theta \geq 1.0 \times 10^{-6} \).

4: n – Integer
On entry: \( n \), the number of pseudorandom uniform variates to generate.
Constraint: \( n \geq 0 \).

5: m – Integer
On entry: \( m \), the number of dimensions.
Constraint: \( m \geq 2 \).

6: x[pdx × sdx] – double
Output
Note: where \( X(i, j) \) appears in this document, it refers to the array element
\[ x[(j - 1) \times \text{pdx} + i - 1] \] when \( \text{order} = \text{Nag ColMajor} \);
\[ x[(i - 1) \times \text{pdx} + j - 1] \] when \( \text{order} = \text{Nag RowMajor} \).
On exit: the pseudorandom uniform variates with joint distribution described by \( C_\theta \), with \( X(i, j) \) holding the \( i \)th value for the \( j \)th dimension if \( \text{order} = \text{Nag ColMajor} \) and the \( j \)th value for the \( i \)th dimension of \( \text{order} = \text{Nag RowMajor} \).

7: pdx – Integer
On entry: the stride separating row or column elements (depending on the value of \( \text{order} \)) in the array \( x \).
Constraints:
if \( \text{order} = \text{Nag ColMajor} \), \( \text{pdx} \geq \text{n} \);
if \( \text{order} = \text{Nag RowMajor} \), \( \text{pdx} \geq \text{m} \).

8: sdx – Integer
On entry: the secondary dimension of \( X \).
Constraints:
if \( \text{order} = \text{Nag ColMajor} \), \( \text{sdx} \geq \text{m} \);
if \( \text{order} = \text{Nag RowMajor} \), \( \text{sdx} \geq \text{n} \).

9: fail – NagError *
Input/Output
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.
On entry, \( m = \langle value \rangle \).
Constraint: \( m \geq 2 \).
On entry, \( n = \langle value \rangle \).
Constraint: \( n \geq 0 \).

On entry, \( pdx \) must be at least \( \langle value \rangle \): \( pdx = \langle value \rangle \).
On entry, \( sdx \) must be at least \( \langle value \rangle \): \( sdx = \langle value \rangle \).

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.

On entry, corrupt \( \text{state} \) argument.

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.

On entry, invalid \( \text{theta} \): \( \text{theta} = \langle value \rangle \).
Constraint: \( \text{theta} \geq 1.0 \times 10^{-6} \).

7 Accuracy
Not applicable.

8 Parallelism and Performance
nag_rand_copula_frank (g05rjc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

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
In practice, the need for numerical stability restricts the range of \( \theta \) such that:
the function requires \( \theta \geq 1.0 \times 10^{-6} \);
if \( \theta > -\ln \epsilon \), the function returns pseudorandom uniform variates with \( C_\infty \) joint distribution;
where \( \epsilon \) is the machine precision returned by nag_machine_precision (X02AJC).

10 Example
This example generates thirteen four-dimensional variates for copula \( C_{4,0} \).
10.1 Program Text

/* nag_rand_copula_frank (g05rjc) 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-1)*pdx + I-1):((I-1)*pdx + J-1)]

int main(void)
{
    /* Integer scalar and array declarations */
    Integer exit_status = 0;
    Integer i, j, lstate, pdx, sdx;
    Integer *state = 0;
    /* Double scalar and array declarations */
    double *x = 0;
    /* NAG structures */
    NagError fail;

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

    /* Set the number of variables and variates */
    Integer n = 13, m = 4;

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

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

    /* Set the theta parameter value */
    double theta = 4.0e0;

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

    printf(
        "nag_rand_copula_frank (g05rjc) 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;
    }

    /* Set matrix size and principal dimension according to storage order */
    pdx = (order == Nag_ColMajor)?n:m;
    sdx = (order == Nag_ColMajor)?m:n;

    /* Allocate arrays */
    if (!(x = NAG_ALLOC((pdx*sdx), double)) ||
        !NAG_ALLOCнт(x, double))
    {
        printf("Memory allocation failure\n");
        exit_status = 1;
        goto END;
    }

    printf("theta = %e\n", theta);
}

END:

return exit_status;

!(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;
}

/* Generate variates */
NAG_RAND_COPIULA_Frank(order, state, theta, n, m, x, pdx, sdx, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_rand_copula_frank (g05rjc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Display the results */
printf("Uniform variates with copula joint distrbution\n");
for (i = 1; i <= n; i++)
{
    printf(" ");
    for (j = 1; j <= m; j++)
        printf("%9.6f%s", X(i, j), j < m ? " " : "\n");
}

END:
NAG_FREE(x);
NAG_FREE(state);
return exit_status;
}

10.2 Program Data
None.

10.3 Program Results

NAG_RAND_COPIULA_Frank (g05rjc) Example Program Results

<table>
<thead>
<tr>
<th>Uniform variates with copula joint distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.567865 0.197720 0.868173 0.266408</td>
</tr>
<tr>
<td>0.096539 0.353158 0.977343 0.310208</td>
</tr>
<tr>
<td>0.552604 0.256186 0.634068 0.626682</td>
</tr>
<tr>
<td>0.803556 0.474669 0.731009 0.551512</td>
</tr>
<tr>
<td>0.204266 0.979743 0.362761 0.496830</td>
</tr>
<tr>
<td>0.477652 0.814617 0.392186 0.400528</td>
</tr>
<tr>
<td>0.416196 0.502048 0.507439 0.200816</td>
</tr>
<tr>
<td>0.370321 0.097088 0.052717 0.027783</td>
</tr>
<tr>
<td>0.435379 0.487979 0.409645 0.425912</td>
</tr>
<tr>
<td>0.269262 0.116882 0.063930 0.155473</td>
</tr>
<tr>
<td>0.012721 0.308042 0.235164 0.465872</td>
</tr>
<tr>
<td>0.073024 0.323863 0.201989 0.056817</td>
</tr>
<tr>
<td>0.236933 0.081710 0.311814 0.437014</td>
</tr>
</tbody>
</table>