NAG C Library Function Document

nag_rngs_orthog_matrix (g05qac)

1 Purpose

nag_rngs_orthog_matrix (g05qac) generates a random orthogonal matrix.

2 Specification

```
void nag_rngs_orthog_matrix (Nag_OrderType order, Nag_SideType side,
     Nag_InitializeA init, Integer m, Integer n, double a[], Integer pda,
     Integer igen, Integer iseed[], NagError *fail)
```

3 Description

nag_rngs_orthog_matrix (g05qac) pre- or post-multiplies an m by n matrix A by a random orthogonal matrix U, overwriting A. The matrix A may optionally be initialised to the identity matrix before multiplying by U, hence U is returned. U is generated using the method of Stewart (1980). The algorithm can be summarized as follows.

Let $x_1, x_2, \ldots, x_{n-1}$ follow independent multinormal distributions with zero mean and variance $I\sigma^2$ and dimensions $n, n-1, \ldots, 2$; let $H_j = \text{diag}(I_{j-1}, H_j^*)$, where I_{j-1} is the identity matrix and H_j^* is the Householder transformation that reduces x_j to $r_{jj}e_1$, e_1 being the vector with first element one and the remaining elements zero and r_{jj} being a scalar, and let $D = \text{diag}(\text{sign}(r_{11}), \text{sign}(r_{22}), \ldots, \text{sign}(r_{nn}))$. Then the product $U = DH_1H_2 \ldots H_{n-1}$ is a random orthogonal matrix distributed according to the Haar measure over the set of orthogonal matrices of n. See Stewart (1980), Theorem 3.3.

One of the initialisation functions nag_rngs_init_repeatable (g05kbc) (for a repeatable sequence if computed sequentially) or nag_rngs_init_nonrepeatable (g05kcc) (for a non-repeatable sequence) must be called prior to the first call to nag_rngs_orthog_matrix (g05qac).

4 References

Stewart G W (1980) The efficient generation of random orthogonal matrices with an application to condition estimates *SIAM J. Numer. Anal.* **17** 403–409

5 Parameters

1: **order** – Nag_OrderType *Input*

On entry: the **order** parameter 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 2.2.1.4 of the Essential Introduction for a more detailed explanation of the use of this parameter.

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

2: **side** – Nag_SideType *Input*

On entry: indicates whether the matrix A is multiplied on the left or right by the random orthogonal matrix U.

If **side** = **Nag_LeftSide**, the matrix A is multiplied on the left, i.e., pre-multiplied.

If **side** = **Nag_RightSide**, the matrix A is multiplied on the right, i.e., post-multiplied.

Constraint: **side** = **Nag_LeftSide** or **Nag_RightSide**.

3: **init** – Nag_InitializeA *Input*

On entry: indicates whether or not **a** should be initialised to the identity matrix.

If **init** = **Nag_InitializeI**, **a** is initialised to the identity matrix.

If **init** = **Nag_InputA**, **a** is not initialised and the matrix A must be supplied in **a**.

Constraint: **init** = **Nag_InitializeI** or **Nag_InputA**.

4: **m** – Integer *Input*

On entry: the number of rows of the matrix A, m.

Constraints:

if **side** = **Nag_LeftSide**, **m** > 1;
otherwise **m** ≥ 1.

5: **n** – Integer *Input*

On entry: the number of columns of the matrix A, n.

Constraints:

if **side** = **Nag_RightSide**, **n** > 1;
otherwise **n** ≥ 1.

6: **a**$[dim]$ – double *Input/Output*

Note: the dimension, dim, of the array **a** must be at least $\max(1, \textbf{pda} \times \textbf{n})$ when **order** = **Nag_ColMajor** and at least $\max(1, \textbf{pda} \times \textbf{m})$ when **order** = **Nag_RowMajor**.

If **order** = **Nag_ColMajor**, the (i, j)th element of the matrix A is stored in **a**$[(j-1) \times \textbf{pda} + i - 1]$ and if **order** = **Nag_RowMajor**, the (i, j)th element of the matrix A is stored in **a**$[(i-1) \times \textbf{pda} + j - 1]$.

On entry: if **init** = **Nag_InputA**, **a** must contain the matrix A.

On exit: the matrix UA when **side** = **Nag_LeftSide** or the matrix AU when **side** = **Nag_RightSide**.

7: **pda** – Integer *Input*

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

Constraints:

if **order** = **Nag_ColMajor**, **pda** $\geq$ **m**;
if **order** = **Nag_RowMajor**, **pda** $\geq$ **n**.

8: **igen** – Integer *Input*

On entry: must contain the identification number for the generator to be used to return a pseudo-random number and should remain unchanged following initialisation by a prior call to one of the functions nag_rngs_init_repeatable (g05kbc) or nag_rngs_init_nonrepeatable (g05kcc).

9: **iseed**$[4]$ – Integer *Input/Output*

On entry: contains values which define the current state of the selected generator.

On exit: contains updated values defining the new state of the selected generator.

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

The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT

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

On entry, **n** = ⟨*value*⟩.
Constraint: **n** ≥ 1.

On entry, **m** = ⟨*value*⟩.
Constraint: **m** ≥ 1.

NE_INT_2

On entry, **pda** = ⟨*value*⟩, **m** = ⟨*value*⟩.
Constraint: **pda** $\geq$ **m**.

On entry, **pda** = ⟨*value*⟩, **n** = ⟨*value*⟩.
Constraint: **pda** $\geq$ **n**.

NE_ENUM_INT

On entry, **side** = ⟨*value*⟩, **m** = ⟨*value*⟩.
Constraint: if **side** = **Nag_LeftSide**, **m** > 1;
otherwise **m** ≥ 1.

On entry, **side** = ⟨*value*⟩, **n** = ⟨*value*⟩.
Constraint: if **side** = **Nag_RightSide**, **n** > 1;
otherwise **n** ≥ 1.

NE_ORTHOGONAL_MATRIX

Orthogonal matrix of dimension 1 requested.

NE_ALLOC_FAIL

Memory allocation failed.

NE_BAD_PARAM

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

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 consult NAG for assistance.

7 Accuracy

The maximum error in $U^T U$ should be a modest multiple of ***machine precision*** (see Chapter x02).

8 Further Comments

nag_rngs_corr_matrix (g05qbc) computes a random correlation matrix from a random orthogonal matrix.

9 Example

Following initialisation of the pseudo-random number generator by a call to nag_rngs_init_repeatable (g05kbc), a 4 by 4 orthogonal matrix is generated using the **init** = **Nag_InitializeI** option and the result printed.

9.1 Program Text

```
/* nag_rngs_orthog_matrix(g05qac) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg05.h>

int main(void)
{
  /* Scalars */
  Integer  i, igen, j, m, n;
  Integer  exit_status=0;
  Integer  pda;
  NagError fail;
  Nag_OrderType order;

  /* Arrays */
  double   *a=0;
  Integer  iseed[4];

#ifdef NAG_COLUMN_MAJOR
#define A(I,J) a[(J-1)*pda + I - 1]
  order = Nag_ColMajor;
#else
#define A(I,J) a[(I-1)*pda + J - 1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);
  Vprintf("g05qac Example Program Results\n\n");
  m = 4;
  n = 4;
  /* Allocate memory */
  if ( !(a = NAG_ALLOC(m * n, double)) )
    {
      Vprintf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
#ifdef NAG_COLUMN_MAJOR
  pda = m;
#else
  pda = n;
#endif

  /* igen identifies the stream. */
  igen = 1;
  /* Initialise the seed to a repeatable sequence */
  iseed[0] = 1762543;
  iseed[1] = 9324783;
  iseed[2] = 423446;
  iseed[3] = 742355;

  g05kbc(&igen, iseed);

  g05qac(order, Nag_RightSide, Nag_InitializeI, m, n, a,
         pda, igen, iseed, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from g05qac.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  for (i = 1; i <= 4; ++i)
```

```
    {
      Vprintf("%1s", "");
      for (j = 1; j <= 4; ++j)
        {
          Vprintf("%9.3f", A(i,j));
          Vprintf("%s", j%4 == 0 || j == 4 ?"\n":" ");
        }
    }
 END:
  if (a) NAG_FREE(a);
  return exit_status;
}
```

9.2 Program Data

None.

9.3 Program Results

```
g05qac Example Program Results

    -0.219    -0.197    -0.413    -0.862
     0.438     0.006     0.762    -0.478
     0.699    -0.627    -0.322     0.120
    -0.521    -0.754     0.381     0.122
```