NAG Library Function Document

nag_rand_orthog_matrix (g05pxc)

1 Purpose

nag_rand_orthog_matrix (g05pxc) generates a random orthogonal matrix.

2 Specification

```
#include <nag.h>
#include <nagg05.h>
```

```
void nag_rand_orthog_matrix (Nag_SideType side, Nag_InitializeA init,
     Integer m, Integer n, Integer state[], double a[], Integer pda,
     NagError *fail)
```

3 Description

nag_rand_orthog_matrix (g05pxc) pre- or post-multiplies an m by n matrix A by a random orthogonal matrix U, overwriting A. The matrix A may optionally be initialized 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 summarised 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 $H_j = \mathrm{diag}\left(I_{j-1}, H_j^*\right)$, 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 $D = \mathrm{diag}(\mathrm{sign}(r_{11}), \mathrm{sign}(r_{22}), \ldots, \mathrm{sign}(r_{nn}))$. 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 Theorem 3.3 in Stewart (1980).

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_orthog_matrix (g05pxc).

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 Arguments

1: **side** – Nag_SideType *Input*

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

side $=$ Nag_LeftSide
The matrix A is multiplied on the left, i.e., premultiplied.

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

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

2: **init** – Nag_InitializeA *Input*

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

init = Nag_InitializeI
  a is initialized to the identity matrix.

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

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

3: **m** – Integer *Input*

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

Constraints:

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

4: **n** – Integer *Input*

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

Constraints:

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

5: **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.

6: **a**[**m** $\times$ **pda**] – double *Input/Output*

On entry: if **init** = Nag_InputA, **a** must contain the matrix A, with the (i,j)th element of A stored in **a**$[(i-1) \times$ **pda** $+ j - 1]$.

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 column elements in the array **a**.

Constraint: **pda** $\geq$ **n**.

8: **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 $\langle value \rangle$ had an illegal value.

NE_ENUM_INT

On entry, **side** = ⟨*value*⟩ and **m** = ⟨*value*⟩.
Constraint: if **side** = Nag_LeftSide, $\mathbf{m} > 1$;
otherwise $\mathbf{m} \geq 1$.

On entry, **side** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: if **side** = Nag_RightSide, $\mathbf{n} > 1$;
otherwise $\mathbf{n} \geq 1$.

NE_INT

On entry, **pda** = ⟨*value*⟩.
Constraint: $\mathbf{pda} > 0$.

NE_INT_2

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

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.

7 Accuracy

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

8 Parallelism and Performance

nag_rand_orthog_matrix (g05pxc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_rand_orthog_matrix (g05pxc) 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

None.

10 Example

Following initialization of the pseudorandom number generator by a call to nag_rand_init_repeatable (g05kfc), a 4 by 4 orthogonal matrix is generated using the **init** = Nag_InitializeI option and the result printed.

10.1 Program Text

```
/* nag_rand_orthog_matrix (g05pxc) 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 A(I, J) a[I*pda + J]

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

  /* NAG structures */
  NagError        fail;

  /* Double scalar and array declarations */
  double          *a = 0;

  /* Set the size of the matrix to be generated */
  Integer         n = 4;
  Integer         m = 4;

  /* Set A to the identity matrix on input */
  Nag_InitializeA init = Nag_InitializeI;

  /* Multiple on the right hand side */
  Nag_SideType    side = Nag_RightSide;

  /* 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_orthog_matrix (g05pxc) 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;
    }

  pda = n;
  a_size = pda * m;

  /* Allocate arrays */
  if (!(a = NAG_ALLOC(a_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;
    }

  /* Generate the random orthogonal matrix*/
  nag_rand_orthog_matrix(side, init, m, n, state, a, pda, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_rand_orthog_matrix (g05pxc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

  /* Display the results */
  for (i = 0; i < m; i++)
    {
      printf("  ");
      for (j = 0; j < n; j++)
        printf("%8.3f%s", A(i, j), (j+1)%4?" ":"\n");
      if (n%4) printf("\n");
    }

 END:
  NAG_FREE(a);
  NAG_FREE(state);

  return exit_status;
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_rand_orthog_matrix (g05pxc) Example Program Results

     0.176    0.740   -0.307   -0.572
     0.659   -0.578   -0.219   -0.428
     0.668    0.317    0.608    0.290
    -0.297   -0.132    0.699   -0.637
```