NAG Library Function Document

nag_rand_orthog_matrix (g05pxc)

1 Purpose

nag_rand_orthog_matrix (g05pxc) generates a random orthogonal matrix.

2 Specification

```c
#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 $m \times n$ 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 $H_j = \text{diag}(I_{j-1}, H_j')$, where $I_{j-1}$ is the identity matrix and $H_j'$ is the Householder transformation that reduces $x_j$ to $r_j e_1$, $e_1$ being the vector with first element one and the remaining elements zero and $r_j$ being a scalar, and let $D = \text{diag}(\text{sign}(r_{11}), \text{sign}(r_{22}), \ldots, \text{sign}(r_{nn}))$. Then the product $U = DH_1 H_2 \ldots H_{n-1}$ 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


5 Arguments

1: \hspace{1em} \textit{side} – Nag_SideType \hspace{1em} \textit{Input}

\textit{On entry}: indicates whether the matrix $A$ is multiplied on the left or right by the random orthogonal matrix $U$.

- \textit{side} = Nag_LeftSide
  The matrix $A$ is multiplied on the left, i.e., premultiplied.

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

\textit{Constraint}: \textit{side} = Nag_LeftSide or Nag_RightSide.
2: \( \text{init} \) – \text{Nag} \_\text{InitializeA} \hspace{1cm} \text{Input} \\
\text{On entry:} \text{ indicates whether or not } \text{a} \text{ should be initialized to the identity matrix.} \\
\text{init} = \text{Nag} \_\text{InitializeI} \\
\text{a} \text{ is initialized to the identity matrix.} \\
\text{init} = \text{Nag} \_\text{InputA} \\
\text{a} \text{ is not initialized and the matrix } \text{A} \text{ must be supplied in } \text{a}. \\
\text{Constraint: } \text{init} = \text{Nag} \_\text{InitializeI} \text{ or } \text{Nag} \_\text{InputA}. \\

3: \( \text{m} \) – \text{Integer} \hspace{1cm} \text{Input} \\
\text{On entry: } \text{m}, \text{ the number of rows of the matrix } \text{A}. \\
\text{Constraints:} \\
\text{if } \text{side} = \text{Nag} \_\text{LeftSide}, \text{ m} > 1; \\
\text{otherwise } \text{m} \geq 1. \\

4: \( \text{n} \) – \text{Integer} \hspace{1cm} \text{Input} \\
\text{On entry: } \text{n}, \text{ the number of columns of the matrix } \text{A}. \\
\text{Constraints:} \\
\text{if } \text{side} = \text{Nag} \_\text{RightSide}, \text{ n} > 1; \\
\text{otherwise } \text{n} \geq 1. \\

5: \( \text{state}[\text{dim}] \) – \text{Integer} \hspace{1cm} \text{Communication Array} \\
\text{Note:} \text{ the dimension, } \text{dim}, \text{ of this array is dictated by the requirements of associated functions that} \\
\text{must have been previously called. This array MUST be the same array passed as argument } \text{state} \text{ in} \\
\text{the previous call to } \text{nag} \_\text{rand} \_\text{init} \_\text{repeatable} (g05\text{kfc}) \text{ or } \text{nag} \_\text{rand} \_\text{init} \_\text{nonrepeatable} (g05\text{kgc}). \\
\text{On entry:} \text{ contains information on the selected base generator and its current state.} \\
\text{On exit:} \text{ contains updated information on the state of the generator.} \\

6: \( \text{a}[^{\text{m} \times \text{pda}}] \) – \text{double} \hspace{1cm} \text{Input/Output} \\
\text{On entry:} \text{ if } \text{init} = \text{Nag} \_\text{InputA}, \text{ a} \text{ must contain the matrix } \text{A}, \text{ with the (i,j)th element of } \text{A} \text{ stored} \\
\text{in } \text{a}[^{(i-1) \times \text{pda} + j - 1}]. \\
\text{On exit:} \text{ the matrix } \text{UA} \text{ when } \text{side} = \text{Nag} \_\text{LeftSide} \text{ or the matrix } \text{AU} \text{ when } \text{side} = \text{Nag} \_\text{RightSide}. \\

7: \( \text{pda} \) – \text{Integer} \hspace{1cm} \text{Input} \\
\text{On entry:} \text{ the stride separating matrix column elements in the array } \text{a}. \\
\text{Constraint: } \text{pda} \geq \text{n}. \\

8: \( \text{fail} \) – \text{NagError} * \hspace{1cm} \text{Input/Output} \\
\text{The NAG error argument (see Section 3.6 in the Essential Introduction).} \\

6 \text{ Error Indicators and Warnings} \\

\text{NE} \_\text{ALLOC} \_\text{FAIL} \\
\text{Dynamic memory allocation failed.} \\
\text{See Section 3.2.1.2 in the Essential Introduction for further information.} \\

\text{NE} \_\text{BAD} \_\text{PARAM} \\
\text{On entry, argument } \langle \text{value} \rangle \text{ had an illegal value.} \\

\text{g05pxc.2} \hspace{1cm} \text{Mark 25}
NE_ENUM_INT
On entry, side = (value) and m = (value).
Constraint: if side = Nag_LeftSide, m > 1; otherwise m ≥ 1.
On entry, side = (value) and n = (value).
Constraint: if side = Nag_RightSide, n > 1; otherwise n ≥ 1.

NE_INT
On entry, pda = (value).
Constraint: pda > 0.

NE_INT_2
On entry, pda = (value) and n = (value).
Constraint: pda ≥ 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 $U^TU$ 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_Initialize 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", fail.message);
        exit_status = 1;
        goto END;
    }

    pda = n;
    a_size = pda * m;

    /* Allocate arrays */
    if (!(a = NAG_ALLOC(a_size, double)) ||
    END:
    exit_status = 1;
    goto END;
}

END:
exit_status = 1;
!(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", 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

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.176</td>
<td>0.740</td>
<td>-0.307</td>
<td>-0.572</td>
<td></td>
</tr>
<tr>
<td>0.659</td>
<td>-0.578</td>
<td>-0.219</td>
<td>-0.428</td>
<td></td>
</tr>
<tr>
<td>0.668</td>
<td>0.317</td>
<td>0.608</td>
<td>0.290</td>
<td></td>
</tr>
<tr>
<td>-0.297</td>
<td>-0.132</td>
<td>0.699</td>
<td>-0.637</td>
<td></td>
</tr>
</tbody>
</table>