NAG Library Function Document

nag_quasi_rand_normal (g05yjc)

1 Purpose

nag_quasi_rand_normal (g05yjc) generates a quasi-random sequence from a Normal (Gaussian) distribution. It must be preceded by a call to one of the initialization functions nag_quasi_init (g05ylc) or nag_quasi_init_scrambled (g05ync).

2 Specification

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

```
void nag_quasi_rand_normal (Nag_OrderType order, const double xmean[],
     const double std[], Integer n, double quas[], Integer pdquas,
     Integer iref[], NagError *fail)
```

3 Description

nag_quasi_rand_normal (g05yjc) generates a quasi-random sequence from a Normal distribution by first generating a uniform quasi-random sequence which is then transformed into a Normal sequence using the inverse of the Normal CDF. The type of uniform sequence used depends on the initialization function called and can include the low-discrepancy sequences proposed by Sobol, Faure or Niederreiter. If the initialization function nag_quasi_init_scrambled (g05ync) was used then the underlying uniform sequence is first scrambled prior to being transformed (see Section 3 in nag_quasi_init_scrambled (g05ync) for details).

4 References

Bratley P and Fox B L (1988) Algorithm 659: implementing Sobol's quasirandom sequence generator *ACM Trans. Math. Software* **14(1)** 88–100

Fox B L (1986) Algorithm 647: implementation and relative efficiency of quasirandom sequence generators *ACM Trans. Math. Software* **12(4)** 362–376

Wichura (1988) Algorithm AS 241: the percentage points of the Normal distribution *Appl. Statist.* **37** 477–484

5 Arguments

Note: the following variables are used in the parameter descriptions:

$idim$ = **idim**, the number of dimensions required, see nag_quasi_init (g05ylc) or nag_quasi_init_scrambled (g05ync);

$liref$ = **liref**, the length of **iref** as supplied to the initialization functions nag_quasi_init (g05ylc) or nag_quasi_init_scrambled (g05ync).

$tdquas$ = **n** if **order** = Nag_RowMajor; otherwise $tdquas = idim$.

1: **order** – Nag_OrderType *Input*

On entry: the **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 **order** = Nag_RowMajor. See Section 2.3.1.3 in How to Use the NAG Library and its Documentation for a more detailed explanation of the use of this argument.

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

2: **xmean**[$idim$] – const double *Input*

On entry: specifies, for each dimension, the mean of the Normal distribution.

3: **std**[$idim$] – const double *Input*

On entry: specifies, for each dimension, the standard deviation of the Normal distribution.

Constraint: **std**$[i-1] \geq 0.0$, for $i = 1, 2, \ldots, idim$.

4: **n** – Integer *Input*

On entry: the number of quasi-random numbers required.

Constraint: **n** ≥ 0 and **n** + previous number of generated values $\leq 2^{31} - 1$.

5: **quas**[dim] – double *Output*

Note: the dimension, *dim*, of the array **quas** must be at least **pdquas** $\times$ $idim$.

The dimension, *dim*, of the array **quas** must be at least

max(1, **pdquas** $\times$ $idim$) when **order** = Nag_ColMajor;
max(1, **n** $\times$ **pdquas**) when **order** = Nag_RowMajor.

Where **QUAS**(i, j) appears in this document, it refers to the array element

quas$[(j-1) \times$ **pdquas** $+ i - 1]$ when **order** = Nag_ColMajor;
quas$[(i-1) \times$ **pdquas** $+ j - 1]$ when **order** = Nag_RowMajor.

On exit: contains the **n** quasi-random numbers of dimension $idim$, **QUAS**(i, j) holds the ith value for the jth dimension.

6: **pdquas** – Integer *Input*

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

Constraints:

if **order** = Nag_ColMajor, **pdquas** $\geq$ **n**;
if **order** = Nag_RowMajor, **pdquas** $\geq$ $idim$.

7: **iref**[$liref$] – Integer *Communication Array*

On entry: contains information on the current state of the sequence.

On exit: contains updated information on the state of the sequence.

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

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_BAD_PARAM

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

NE_INITIALIZATION

On entry, **iref** has either not been initialized or has been corrupted.

NE_INT

On entry, **n** $= \langle value \rangle$.
Constraint: **n** ≥ 0.

NE_INT_2

On entry, **pdquas** $= \langle value \rangle$ and $idim = \langle value \rangle$.
Constraint: **pdquas** $\geq idim$.

On entry, **pdquas** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: **pdquas** $\geq$ **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 2.7.6 in How to Use the NAG Library and its Documentation for further information.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

NE_REAL_ARRAY

On entry, **std**$[\langle value \rangle] = \langle value \rangle$.
Constraint: **std**$[i] \geq 0.0$.

NE_TOO_MANY_CALLS

There have been too many calls to the generator.

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_quasi_rand_normal (g05yjc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_quasi_rand_normal (g05yjc) 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.

The Sobol, Sobol (A659) and Niederreiter quasi-random number generators in nag_quasi_rand_normal (g05yjc) have been parallelized, but require quite large problem sizes to see any significant performance gain. Parallelism is only enabled when **order** = Nag_ColMajor. The Faure generator is serial.

9 Further Comments

None.

10 Example

This example calls nag_quasi_init (g05ylc) to initialize the generator and then nag_quasi_rand_normal (g05yjc) to generate a sequence of five four-dimensional variates.

10.1 Program Text

```
/* nag_quasi_rand_normal (g05yjc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

/* Pre-processor includes */
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg05.h>
#define QUAS(I, J) quas[(order == Nag_ColMajor)?(J*pdquas + I):(I*pdquas + J)]

int main(void)
{
  /* Integer scalar and array declarations */
  Integer exit_status = 0;
  Integer liref, i, j, q_size;
  Integer *iref = 0;
  Integer pdquas;

  /* NAG structures */
  NagError fail;

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

  /* Number of dimensions */
  Integer idim = 4;

  /* Mean and standard deviation of the normal distribution */
  double xmean[] = { 1.0e0, 2.0e0, 3.0e0, 4.0e0 };
  double std[] = { 1.0e0, 1.0e0, 1.0e0, 1.0e0 };

  /* Set the sample size */
  Integer n = 5;

  /* Skip the first 1000 variates */
  Integer iskip = 1000;

  /* Use column major order */
  Nag_OrderType order = Nag_ColMajor;

  /* Choose the quasi generator */
  Nag_QuasiRandom_Sequence genid = Nag_QuasiRandom_Sobol;

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

  printf("nag_quasi_rand_normal (g05yjc) Example Program Results\n\n");

  pdquas = (order == Nag_RowMajor) ? idim : n;
  q_size = (order == Nag_RowMajor) ? pdquas * n : pdquas * idim;

  /* Calculate the size of the reference vector */
  liref = (genid == Nag_QuasiRandom_Faure) ? 407 : 32 * idim + 7;
```

```
  /* Allocate arrays */
  if (!(quas = NAG_ALLOC(q_size, double)) ||
      !(iref = NAG_ALLOC(liref, Integer)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Initialize the Sobol generator */
  nag_quasi_init(genid, idim, iref, liref, iskip, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_quasi_init (g05ylc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Generate a normal quasi-random number sequence */
  nag_quasi_rand_normal(order, xmean, std, n, quas, pdquas, iref, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_quasi_rand_normal (g05yjc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Print the estimated value of the integral */
  for (i = 0; i < n; i++) {
    printf("  ");
    for (j = 0; j < idim; j++)
      printf("%9.4f%s", QUAS(i, j), ((j + 1) % 4) ? " " : "\n");
    if (idim % 4)
      printf("\n");
  }

END:
  NAG_FREE(quas);
  NAG_FREE(iref);

  return exit_status;
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_quasi_rand_normal (g05yjc) Example Program Results

     1.5820    2.2448    0.9154    3.0722
     2.8768    1.6057    3.7341    5.4521
     0.9240    3.0223    2.3828    3.8154
     0.6004    1.9290    1.9355    3.4806
     2.0141    3.9061    3.3680    4.8479
```