1 Purpose

nag_quasi_rand_lognormal (g05ykc) generates a quasi-random sequence from a log-normal 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

```c
#include <nag.h>
#include <nagg05.h>
void nag_quasi_rand_lognormal (Nag_OrderType order, const double xmean[],
const double std[], Integer n, double quas[], Integer pdquas,
Integer iref[], NagError *fail)
```

3 Description

nag_quasi_rand_lognormal (g05ykc) generates a quasi-random sequence from a log-normal distribution by first generating a uniform quasi-random sequence which is then transformed into a log-normal sequence using the exponential of 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


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 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

*Constraint:* `order = Nag_RowMajor` or `Nag_ColMajor`.
2: \texttt{xmean[idim]} – const double \hfill Input

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

*Constraint:* \[ xmean[i-1] \leq \left| -\log (\text{nag\_real\_safe\_small\_number}) - 10.0 \times \text{std}[i-1] \right|, \] for \( i = 1, 2, \ldots, \text{idim} \).

3: \texttt{std[idim]} – const double \hfill Input

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

*Constraint:* \( \text{std}[i-1] \geq 0.0, \) for \( i = 1, 2, \ldots, \text{idim} \).

4: \texttt{n} – Integer \hfill Input

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

*Constraint:* \( n \geq 0 \) and \( n + \text{previous number of generated values} \leq 2^{31} - 1 \).

5: \texttt{quas[dim]} – double \hfill Output

*Note:* the dimension, \( \text{dim} \), of the array \texttt{quas} must be at least \( \text{pdquas} \times \text{idim} \).

The dimension, \( \text{dim} \), of the array \texttt{quas} must be at least

\[ \max(1, \text{pdquas} \times \text{idim}) \quad \text{when order = Nag\_ColMajor}; \]
\[ \max(1, n \times \text{pdquas}) \quad \text{when order = Nag\_RowMajor}. \]

Where \( \text{QUAS}(i, j) \) appears in this document, it refers to the array element

\[ \text{quas}[(j-1) \times \text{pdquas} + i - 1] \quad \text{when order = Nag\_ColMajor}; \]
\[ \text{quas}[(i-1) \times \text{pdquas} + j - 1] \quad \text{when order = Nag\_RowMajor}. \]

*On exit:* \( \text{QUAS}(i, j) \) holds the \( i \)th value for the \( j \)th dimension.

6: \texttt{pdquas} – Integer \hfill Input

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

*Constraints:*

\[
\begin{align*}
\text{if order = Nag\_ColMajor, pdquas} & \geq n; \\
\text{if order = Nag\_RowMajor, pdquas} & \geq \text{idim}.
\end{align*}
\]

7: \texttt{iref[liref]} – Integer \hfill 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: \texttt{fail} – NagError * \hfill 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.
NE_INITIALIZATION
On entry, iref has either not been initialized or has been corrupted.

NE_INT
On entry, n = ⟨value⟩.
Constraint: n ≥ 0.

NE_INT_2
On entry, pdquas = ⟨value⟩ and idim = ⟨value⟩.
Constraint: pdquas ≥ idim.
On entry, pdquas = ⟨value⟩ and n = ⟨value⟩.
Constraint: pdquas ≥ 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_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.

NE_REAL_ARRAY
On entry, at least one element of xmean is too large, xmean[⟨value⟩] = ⟨value⟩.
Constraint: xmean[i] ≤ ⟨value⟩.
On entry, std[⟨value⟩] = ⟨value⟩.
Constraint: std[i] ≥ 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_lognormal (g05ykc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_quasi_rand_lognormal (g05ykc) 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 g05ykc 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_lognormal (g05ykc) to produce a sequence of five four-dimensional quasi-random numbers variates.

10.1 Program Text

/* nag_quasi_rand_lognormal (g05ykc) Example Program. *
 * Copyright 2014 Numerical Algorithms Group. *
 * Mark 9, 2009. */

/* 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 underlying 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;
    /* Initialise the error structure */
    INIT_FAIL(fail);
    printf(  
        "nag_quasi_rand_lognormal (g05ykc) 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;
}

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

/* Generate a log-normal quasi-random number sequence */
nag_quasi_rand_lognormal(order, xmean, std, n, quas, pdquas, iref, &fail);
if (fail.code != NE_NOERROR){
    printf("Error from nag_quasi_rand_lognormal (g05ykc).
%s
", 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)?":":"
    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_lognormal (g05ykc) Example Program Results

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.8648</td>
<td>9.4382</td>
<td>2.4979</td>
<td>21.5895</td>
</tr>
<tr>
<td>17.7572</td>
<td>4.9813</td>
<td>41.8501</td>
<td>233.2386</td>
</tr>
<tr>
<td>2.5195</td>
<td>20.5384</td>
<td>10.8353</td>
<td>45.3933</td>
</tr>
<tr>
<td>1.8229</td>
<td>6.8823</td>
<td>6.9276</td>
<td>32.4808</td>
</tr>
<tr>
<td>7.4938</td>
<td>49.7034</td>
<td>29.0198</td>
<td>127.4745</td>
</tr>
</tbody>
</table>