NAG C Library Function Document

nag_quasi_random_normal (g05ybc)

1 Purpose

To generate multi-dimensional quasi-random sequences with a Gaussian or log-normal probability distribution.

2 Specification

```
#include <nag.h>
#include <nagg05.h>
void nag_quasi_random_normal (Nag_QuasiRandom_State state,
     Nag_QuasiRandom_Sequence seq, Nag_Distributions lnorm, const double mean[],
     const double std[], Integer iskip, Integer idim, double quasi[],
     Nag_QuasiRandom *gf, NagError *fail)
```

3 Description

Low discrepancy (quasi-random) sequences are used in numerical integration, simulation and optimization. Like pseudo-random numbers they are uniformly distributed but they are not statistically independent, rather they are designed to give more even distribution in multidimensional space (uniformity). Therefore they are often more efficient than pseudo-random numbers in multidimensional Monte Carlo methods.

nag_quasi_random_normal (g05ybc) generates multi-dimensional quasi-random sequences with a Gaussian or log-normal probability distribution. The sequences are generated in pairs using the Box–Muller method. This means that an even number of dimensions are required by this function. If an odd number of dimensions are required then the extra dimension must be computed, but can then be ignored.

This function uses the sequences as described in nag_quasi_random_uniform (g05yac).

4 References

Box G E P and Muller M E (1958) A note on the generation of random normal deviates *Ann. Math. Statist.* **29** 610–611

Brately 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) Implementation and Relative Efficiency of Quasirandom Sequence Generators *ACM Trans. Math. Software* **12 (4)** 362–376

5 Arguments

1: **state** – Nag_QuasiRandom_State *Input*

On entry: the type of operation to perform.

state = **Nag_QuasiRandom_Init**

The first call for initialization and there is no output via array **quasi**.

state = **Nag_QuasiRandom_Cont**

The sequence has already been initialized by a prior call to nag_quasi_random_normal (g05ybc) with **state** = **Nag_QuasiRandom_Init**. Random numbers are output via array **quasi**.

state = **Nag_QuasiRandom_Finish**

The final call to release memory and no further random numbers are required for output via array **quasi**.

Constraint: **state** = **Nag_QuasiRandom_Init**, **Nag_QuasiRandom_Cont** or **Nag_QuasiRandom_Finish**.

2: **seq** – Nag_QuasiRandom_Sequence *Input*

On entry: the type of sequence to generate.

seq = **Nag_QuasiRandom_Sobol**

A Sobol sequence.

seq = **Nag_QuasiRandom_Nied**

A Neiderreiter sequence.

seq = **Nag_QuasiRandom_Faure**

A Faure sequence.

Constraint: **seq** = **Nag_QuasiRandom_Sobol**, **Nag_QuasiRandom_Nied** or **Nag_QuasiRandom_Faure**.

3: **lnorm** – Nag_Distributions *Input*

On entry: indicates whether to create Gaussian or log-normal variates. If **lnorm** = **Nag_LogNormal** then the variates are log-normal, otherwise they are Gaussian.

Constraint: **lnorm** = **Nag_LogNormal** or **Nag_Normal**.

4: **mean**[**idim**] – const double *Input*

On entry: **mean**$[k-1]$ is the mean of distribution for the kth dimension.

5: **std**[**idim**] – const double *Input*

On entry: **std**$[k-1]$ is the standard deviation of the distribution for the kth dimension.

Constraint: **std**$[i] > 0.0$, for $i = 0, 1, \ldots,$ **idim** $- 1$.

6: **iskip** – Integer *Input*

On entry: the number of terms in the sequence to skip on initialization. **iskip** is not referenced when **seq** = **Nag_QuasiRandom_Faure**.

Constraint: if **seq** = **Nag_QuasiRandom_Nied** or **Nag_QuasiRandom_Sobol** and **state** = **Nag_QuasiRandom_Init**, **iskip** ≥ 0.

7: **idim** – Integer *Input*

On entry: the number of dimensions required.

Constraint: $2 \leq$ **idim** ≤ 40 and **idim** must be even.

8: **quasi**[**idim**] – double *Output*

On exit: the random numbers, generated in pairs. That is, on the first call with **state** = **Nag_QuasiRandom_Cont**, **quasi**$[k-1]$ contains the first quasi-random number for the kth dimension. On the next call **quasi**$[k-1]$ contains the second quasi-random number for the kth dimension, etc.

9: **gf** – Nag_QuasiRandom * *Communication Structure*

Note: **gf** is a NAG defined type (see Section 2.2.1.1 of the Essential Introduction).

Workspace used to communicate information between calls to nag_quasi_random_normal (g05ybc). The contents of this structure should not be changed between calls.

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

The NAG error argument (see Section 2.6 of the Essential Introduction).

6 Error Indicators and Warnings

NE_BAD_PARAM

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

NE_INITIALIZATION

Incorrect initialization.

NE_INT

On entry, **idim** = ⟨*value*⟩.
Constraint: **idim** ≤ 40.

On entry, value of skip too large: **iskip** = ⟨*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.

NE_TOO_MANY_CALLS

Too many calls to generator.

7 Accuracy

Not applicable.

8 Further Comments

The maximum length of the generated sequences is $2^{29} - 1$, this should be adequate for practical purposes. For more information see nag_quasi_random_uniform (g05yac).

9 Example

This example program calculates the sum of the expected values of the kurtosis of 20 independent Gaussian samples. A quasi-random Faure sequence generator is used.

9.1 Program Text

```
/* nag_quasi_random_normal (g05ybc) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 * Mark 7b revised, 2004.
 */

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

```
static double fun(Integer idim, double  mean[], double std[],
                  double x[]);

int main(void)
{
  /* Scalars */
  Integer  i, idim, ntimes, skip;
  Integer  exit_status=0;
  double   sum, val1, val2;
  NagError fail;
  static Nag_QuasiRandom GF;
  Nag_QuasiRandom_Sequence seq;
  Nag_QuasiRandom_State    state;
  Nag_Distributions        dist;

  /* Arrays */
  double *mean, *quasi, *std;

#define STD(I) std[(I)-1]
#define QUASI(I) quasi[(I)-1]
#define MEAN(I) mean[(I)-1]

  INIT_FAIL(fail);
  Vprintf("nag_quasi_random_normal (g05ybc) Example Program Results\n\n");

  idim = 20;
  /* Allocate memory */
  if ( !(mean = NAG_ALLOC(idim, double)) ||
       !(quasi = NAG_ALLOC(idim, double)) ||
       !(std = NAG_ALLOC(idim, double)) )
    {
      Vprintf("Allocation error \n");
      exit_status = -1;
      goto END;
    }
  ntimes = 10000;
  dist = Nag_Normal;
  seq = Nag_QuasiRandom_Faure;
  if (seq == Nag_QuasiRandom_Nied)
    skip = 1000;
  else
    skip = 0;

  for (i = 1; i <= idim; ++i)
    {
      MEAN(i) = 2.0;
      STD(i) = 1.0;
    }

  /* Initialise quasi-random generator*/
  state = Nag_QuasiRandom_Init;
  /* nag_quasi_random_normal (g05ybc).
   * Multi-dimensional quasi-random number generator with a
   * Gaussian or log-normal probability distribution
   */
  nag_quasi_random_normal(state, seq, dist, &MEAN(1), &STD(1), skip, idim,
                          &QUASI(1), &GF, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Initialization Error from nag_quasi_random_normal (g05ybc)."
              "\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Evaluate integrand at quasi-random locations and sum */
  sum = 0.0;
  state = Nag_QuasiRandom_Cont;
  for (i = 1; i <= ntimes; ++i)
    {
      /* nag_quasi_random_normal (g05ybc), see above. */
```

```
      nag_quasi_random_normal(state, seq, dist, &MEAN(1), &STD(1), skip,
                              idim, &QUASI(1), &GF, &fail);
      sum += fun(idim, &MEAN(1), &STD(1), &QUASI(1));
    }
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from nag_quasi_random_normal (g05ybc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }
  val1 = sum / (double) ntimes;
  Vprintf("Calculated value of the integral = %8.3f\n\n",val1);
  val2 = (double) idim * 3.0;
  Vprintf("Exact value of the integral      = %8.3f\n",val2);

  /* Finish quasi-random generator */
  state = Nag_QuasiRandom_Finish;
  /* nag_quasi_random_normal (g05ybc), see above. */
  nag_quasi_random_normal(state, seq, dist, &MEAN(1), &STD(1), skip, idim,
                          &QUASI(1), &GF, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Finish Error from nag_quasi_random_normal (g05ybc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }
 END:
  if (mean) NAG_FREE(mean);
  if (quasi) NAG_FREE(quasi);
  if (std) NAG_FREE(std);
  return exit_status;
}

static double fun(Integer idim, double  mean[], double std[], double x[])
{
  Integer j;
  double tmp1, tmp2;
#define X(I) x[(I)-1]
#define STD(I) std[(I)-1]
#define MEAN(I) mean[(I)-1]

  tmp1 = 0.0;
  for (j = 1; j <= idim; ++j)
    {
      tmp2 = (X(j) - MEAN(j)) / STD(j);
      tmp1 += tmp2 * tmp2 * tmp2 * tmp2;
    }
  return tmp1;
}
```

9.2 Program Data

None.

9.3 Program Results

```
nag_quasi_random_normal (g05ybc) Example Program Results

Calculated value of the integral =   60.119

Exact value of the integral      =   60.000
```