NAG C Library Function Document

nag_quasi_random_uniform (g05yac)

1 Purpose

To generate multi-dimensional quasi-random sequences with a uniform probability distribution.

2 Specification

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

void nag_quasi_random_uniform (Nag_QuasiRandom_State state,
     Nag_QuasiRandom_Sequence seq, 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_uniform (g05yac) generates a set of points $x^1, x^2, \ldots, x^N$ with high uniformity in the S-dimensional unit cube $I^S = [0,1]^S$. One measure of the uniformity is the discrepancy which is defined as follows:

> Given a set of points $x^1, x^2, \ldots, x^N \in I^S$ and a subset $G \subset I^S$, define the counting function $S_N(G)$ as the number of points $x^i \in G$. For each $x = (x_1, x_2, \ldots, x_S) \in I^S$, let G_x be the rectangular s-dimensional region
>
> $$G_x = [0, x_1) \times [0, x_2) \times \ldots \times [0, x_S)$$
>
> with volume $x_1, x_2, \ldots, x_S$. Then the discrepancy of the points $x^1, x^2, \ldots, x^N$ is
>
> $$D_N^*\left(x^1, x^2, \ldots, x^N\right) = \sup_{x \in I^S} |S_N(G_x) - N x_1, x_2, \ldots, x_S|.$$
>
> The discrepancy of the first N terms of such a sequence has the form
>
> $$D_N^*\left(x^1, x^2, \ldots, x^N\right) \leq C_S (\log N)^S + \mathrm{O}\left((\log N)^{S-1}\right) \quad \text{for all} \quad N \geq 2.$$
>
> The principal aim in the construction of low-discrepancy sequences is to find sequences of points in I^S with a bound of this form where the constant C_S is as small as possible.

nag_quasi_random_uniform (g05yac) generates the low-discrepancy sequences proposed by Sobol, Faure and Neiderreiter. Here, both the Sobol and Neiderreiter sequences are implemented in binary arithmetic and make use of the bitwise exclusive-or operation, where possible (see the Users' Note).

4 References

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 been initialized by a prior call to nag_quasi_random_uniform (g05yac) 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: **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**.

When **iskip** = 0, all the terms of the sequence are generated. If **iskip** = k, the first k terms of the sequence are ignored and the first term of the sequence now corresponds to the kth term of the sequence when **iskip** = 0.

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

4: **idim** – Integer *Input*

On entry: the number of dimensions required.

Constraint: $1 \leq$ **idim** ≤ 40.

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

On exit: the random numbers. If **state** = **Nag_QuasiRandom_Cont**, **quasi**$[k-1]$ contains the random number for the kth dimension.

6: **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_uniform (g05yac). The contents of this structure should not be changed between calls.

7: **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. In the case of the Neiderreiter generator nag_quasi_random_uniform (g05yac) jumps to the appropriate starting point, while for the Sobol generator it simply steps through the sequence. In consequence the Sobol generator with large values of **iskip** will take a significant amount of time.

9 Example

This example program approximates the integral

$$\int_0^1 \ldots \int_0^1 \prod_{i=1}^{s} |4x_i - 2| dx_1, dx_2, \ldots, dx_s = 1,$$

where s is the number of dimensions.

9.1 Program Text

```
/* nag_quasi_random_uniform (g05yac) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 * Mark 7b revised, 2004.
 */

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

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

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

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

  /* Arrays */
  double *quasi;

#define QUASI(I) quasi[(I)-1]

  INIT_FAIL(fail);
  printf("nag_quasi_random_uniform (g05yac) Example Program Results\n\n");

  idim = 22;

  /* Allocate memory */
  if ( !(quasi = NAG_ALLOC(idim, double)))
    {
      Vprintf("Allocation error \n");
      exit_status = -1;
      goto END;
    }

  ntimes = 50000;
  seq = Nag_QuasiRandom_Faure;
  if (seq == Nag_QuasiRandom_Nied)
    skip = 1000;
  else
    skip = 0;
  /* Initialise quasi-random generator*/
  state = Nag_QuasiRandom_Init;
  /* nag_quasi_random_uniform (g05yac).
   * Multi-dimensional quasi-random number generator with a
   * uniform probability distribution
   */
  nag_quasi_random_uniform(state, seq, skip, idim, &QUASI(1), &GF, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Initialization Error from nag_quasi_random_uniform (g05yac)."
              "\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_uniform (g05yac), see above. */
      nag_quasi_random_uniform(state, seq, skip, idim, &QUASI(1), &GF, &fail);
      sum += fun(idim, &QUASI(1));
    }
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from nag_quasi_random_uniform (g05yac).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
```

```
    }
  vsbl = sum / (double) ntimes;
  Vprintf("Value of integral = %8.3f\n", vsbl);

  /* Finish quasi-random generator */
  state = Nag_QuasiRandom_Finish;
  /* nag_quasi_random_uniform (g05yac), see above. */
  nag_quasi_random_uniform(state, seq, skip, idim, &QUASI(1), &GF, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Finish Error from nag_quasi_random_uniform (g05yac).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

 END:
  if (quasi) NAG_FREE(quasi);
  return exit_status;
}

static double fun(Integer idim, double  x[])
{
  Integer j;
  double  tmp;

#define X(I) x[(I)-1]

  tmp = 1.0;
  for (j = 1; j <= idim; ++j)
    {
      tmp = tmp * ABS(X(j) * 4.0 - 2.0);
    }
  return tmp;
}
```

9.2 Program Data

None.

9.3 Program Results

```
nag_quasi_random_uniform (g05yac) Example Program Results

Value of integral =    0.987
```