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

```
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 optimisation. 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
>
> $$D_N^*(x^1, x^2, \ldots, x^N) = \sup_{x \in I^S} |S_N(G_x) - N x_1, x_2, \ldots, x_S|.$$
>
> The discrepancy of the first N terms of such a sequence has the form
>
> $$D_N^*(x^1, x^2, \ldots, x^N) \leq C_S (\log N)^S + O((\log N)^{S-1}) \quad \text{for all} \quad N \geq 2.$$
>
> 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 a small as possible.

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

4 References

Fox B L (1986) Implementation and Relative Efficiency of Quasirandom Sequence Generators *ACM Trans. Math. Software* **12 (4)** 362–376

Brately P and Fox B L (1988) Algorithm 659: Implementing Sobol's Quasirandom Sequence Generator *ACM Trans. Math. Software* **14 (1)** 88–100

5 Parameters

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

On entry: the type of operation to perform.

If **state** = **Nag_QuasiRandom_Init**, the first call for initialisation, and there is no output via array **quasi**.

If **state** = **Nag_QuasiRandom_Cont**, the sequence has been initialised by a prior call to nag_quasi_random_uniform (g05yac) with **state** = **Nag_QuasiRandom_Init**. Random numbers are output via array **quasi**.

If **state** = **Nag_QuasiRandom_Finish** then 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.

If **seq** = **Nag_QuasiRandom_Sobol**, a Sobol sequence.

If **seq** = **Nag_QuasiRandom_Nied**, a Niederreiter sequence.

If **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 initialisation. **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 * *Input/Output*

Note: gf is a NAG defined structure. See Section 2.2.1.1 of the Essential Introduction.

On entry/on exit: workspace used to contain information between calls to the function. The contents of this structure should not be changed.

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

The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT

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

On entry, **iskip** = ⟨*value*⟩.
Constraint: **iskip** ≥ 0.

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

On entry, **idim** = ⟨*value*⟩.
Constraint: **idim** ≥ 1.

NE_INITIALISATION

Incorrect initialisation.

NE_INTERNAL_ERROR

Unexpected error – Please contact NAG.

NE_TOO_MANY_CALLS

Too many calls to generator.

NE_ALLOC_FAIL

Memory allocation failed.

NE_BAD_PARAM

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

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 Niederreiter 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

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

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.
 */

#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("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;
  g05yac(state, seq, skip, idim, &QUASI(1), &GF, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Initialization Error from 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)
    {
      g05yac(state, seq, skip, idim, &QUASI(1), &GF, &fail);
      sum += fun(idim, &QUASI(1));
    }
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from 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;
  g05yac(state, seq, skip, idim, &QUASI(1), &GF, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Finish Error from 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

```
g05yac Example Program Results

Value of integral =    0.987
```