NAG Library Function Document

nag_quasi_rand_uniform (g05ymc)

1 Purpose

nag_quasi_rand_uniform (g05ymc) generates a uniformly distributed low-discrepancy sequence as proposed by Sobol, Faure or Niederreiter. 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_uniform (Nag_OrderType order, Integer n, double quas[],
     Integer pdquas, Integer iref[], NagError *fail)
```

3 Description

Low discrepancy (quasi-random) sequences are used in numerical integration, simulation and optimization. Like pseudorandom 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 pseudorandom numbers in multidimensional Monte–Carlo methods.

nag_quasi_rand_uniform (g05ymc) generates a set of points $x^1, x^2, \ldots, x^N$ with high uniformity in the S-dimensional unit cube $I^S = [0,1]^S$.

Let G be a subset of I^S and 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

$$G_x = [0, x_1) \times [0, x_2) \times \cdots \times [0, x_S)$$

with volume $x_1, x_2, \ldots, x_S$. Then one measure of the uniformity of the points $x^1, x^2, \ldots, x^N$ is the discrepancy:

$$D_N^*\left(x^1, x^2, \ldots, x^N\right) = \sup_{x \in I^S} |S_N(G_x) - N x_1, x_2, \ldots, x_S|.$$

which has the form

$$D_N^*\left(x^1, x^2, \ldots, x^N\right) \leq C_S (\log N)^S + O\Big((\log N)^{S-1}\Big) \quad \text{for all} \quad N \geq 2.$$

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.

The type of low-discrepancy sequence generated by nag_quasi_rand_uniform (g05ymc) depends on the initialization function called and can include those proposed by Sobol, Faure or Niederreiter. If the initialization function nag_quasi_init_scrambled (g05ync) was used then the sequence will be scrambled (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

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 function 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: **n** – Integer *Input*

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

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

3: **quas**[**pdquas** $\times$ $tdquas$] – double *Output*

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

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: **QUAS**(i, j) holds the ith value for the jth dimension.

4: **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_RowMajor, **pdquas** $\geq idim$;
if **order** = Nag_ColMajor, **pdquas** $\geq$ **n**.

5: **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.

6: **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 3.2.1.2 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** = ⟨*value*⟩.
Constraint: **n** ≥ 0.

NE_INT_2

On entry, **pdquas** = ⟨*value*⟩, $idim$ = ⟨*value*⟩.
Constraint: if **order** = Nag_RowMajor, **pdquas** ≥ $idim$.

On entry, **pdquas** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: if **order** = Nag_ColMajor, **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 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 3.6.5 in How to Use the NAG Library and its Documentation for further information.

NE_TOO_MANY_CALLS

On entry, value of **n** would result in too many calls to the generator: **n** = ⟨*value*⟩, generator has previously been called ⟨*value*⟩ times.

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_quasi_rand_uniform (g05ymc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

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' Notefor your implementation for any additional implementation-specific information.

The Sobol, Sobol (A659) and Niederreiter quasi-random number generators in nag_quasi_rand_uniform (g05ymc) 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) and nag_quasi_rand_uniform (g05ymc) to estimate the value of the integral

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

In this example the number of dimensions S is set to 8.

10.1 Program Text

```
/* nag_quasi_rand_uniform (g05ymc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */
/* Pre-processor includes */
#include <stdio.h>
#include <math.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, d, i, j, q_size;
  Integer *iref = 0;
  Integer pdquas;

  /* NAG structures */
  NagError fail;

  /* Double scalar and array declarations */
  double sum, tmp, vsbl;
  double *quas = 0;

  /* Number of dimensions */
  Integer idim = 8;

  /* Set the sample size */
  Integer n = 200;

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

  /* Use row major order */
  Nag_OrderType order = Nag_RowMajor;

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

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

  printf("nag_quasi_rand_uniform (g05ymc) Example Program Results\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 n quasi-random variates */
  nag_quasi_rand_uniform(order, n, quas, pdquas, iref, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_quasi_rand_uniform (g05ymc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Estimate integral by evaluating function at each variate and summing */
  sum = 0.0e0;
  for (i = 0; i < n; i++) {
    tmp = 1.0e0;
    for (d = 0; d < idim; d++)
      tmp *= fabs(4.0e0 * QUAS(i, d) - 2.0e0);
    sum += tmp;
  }

  /* Convert sum to mean value */
  vsbl = sum / (double) n;

  /* Print the estimated value of the integral */
  printf("Value of integral =  %8.4f\n\n", vsbl);

  /* Display the first 10 variates used */
  printf("First 10 variates\n");
  for (i = 0; i < 10; i++) {
    printf(" %3" NAG_IFMT " ", i + 1);
    for (j = 0; j < idim; j++)
      printf("%8.4f%s", QUAS(i, j), ((j + 1) % 20) ? " " : "\n");
    if (idim % 20)
      printf("\n");
  }

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

  return exit_status;
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_quasi_rand_uniform (g05ymc) Example Program Results
Value of integral =    1.0410

First 10 variates
   1   0.7197   0.5967   0.0186   0.1768   0.7803   0.4072   0.5459   0.3994
   2   0.9697   0.3467   0.7686   0.9268   0.5303   0.1572   0.2959   0.1494
   3   0.4697   0.8467   0.2686   0.4268   0.0303   0.6572   0.7959   0.6494
```

4	0.3447	0.4717	0.1436	0.3018	0.1553	0.7822	0.4209	0.0244
5	0.8447	0.9717	0.6436	0.8018	0.6553	0.2822	0.9209	0.5244
6	0.5947	0.2217	0.3936	0.0518	0.9053	0.0322	0.1709	0.7744
7	0.0947	0.7217	0.8936	0.5518	0.4053	0.5322	0.6709	0.2744
8	0.0635	0.1904	0.0498	0.4580	0.6240	0.2510	0.9521	0.8057
9	0.5635	0.6904	0.5498	0.9580	0.1240	0.7510	0.4521	0.3057
10	0.8135	0.4404	0.2998	0.2080	0.3740	0.5010	0.7021	0.0557