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

```c
#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:

\[
D^*_N(x^1, x^2, \ldots, x^N) = \sup_{x \in I_S} |S_N(G_x) - N x_1, x_2, \ldots, x_S|.
\]

which has the form

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

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


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 $order = \text{Nag\_RowMajor}$; otherwise $tdquas = idim$.

1: $order$ – Nag_OrderType $\quad$ 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 $order = \text{Nag\_RowMajor}$. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

Constraint: $order = \text{Nag\_RowMajor}$ or $\text{Nag\_ColMajor}$.

2: $n$ – Integer $\quad$ Input

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

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

3: $\text{quas}[pdquas \times tdquas]$ – double $\quad$ Output

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

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

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

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

4: $pdquas$ – Integer $\quad$ Input

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

Constraints:

if $order = \text{Nag\_RowMajor}$, $pdquas \geq idim$;
if $order = \text{Nag\_ColMajor}$, $pdquas \geq n$.

5: $\text{iref}[liref]$ – Integer $\quad$ 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 * $\quad$ 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 value \rangle had an illegal value.

NE_INITIALIZATION
On entry, \texttt{iref} has either not been initialized or has been corrupted.

NE_INT
On entry, \texttt{n} = \langle value \rangle.
Constraint: \texttt{n} \geq 0.

NE_INT_2
On entry, \texttt{pdquas} = \langle value \rangle, \texttt{idim} = \langle value \rangle.
Constraint: if \texttt{order} = \texttt{Nag\_RowMajor}, \texttt{pdquas} \geq \texttt{idim}.
On entry, \texttt{pdquas} = \langle value \rangle and \texttt{n} = \langle value \rangle.
Constraint: if \texttt{order} = \texttt{Nag\_ColMajor}, \texttt{pdquas} \geq \texttt{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_TOO_MANY_CALLS
On entry, value of \texttt{n} would result in too many calls to the generator: \texttt{n} = \langle value \rangle, generator has previously been called \langle value \rangle times.

7 Accuracy
Not applicable.

8 Parallelism and Performance
\texttt{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’ Note for your implementation for any additional implementation-specific information.
The Sobol, Sobol (A659) and Niederreiter quasi-random number generators in \texttt{g05ymc} have been parallelized, but require quite large problem sizes to see any significant performance gain. Parallelism is only enabled when \texttt{order} = \texttt{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. */
* Copyright 2014 Numerical Algorithms Group.
* Mark 9, 2009.
*/
/* Pre-processor includes */
#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.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;
    /* Initialise 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;
}

#ifdef NAG_FREE
NAG_FREE(liref);
#endif

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

<table>
<thead>
<tr>
<th>Variate</th>
<th>Value 1</th>
<th>Value 2</th>
<th>Value 3</th>
<th>Value 4</th>
<th>Value 5</th>
<th>Value 6</th>
<th>Value 7</th>
<th>Value 8</th>
<th>Value 9</th>
<th>Value 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.7197</td>
<td>0.5967</td>
<td>0.0186</td>
<td>0.1768</td>
<td>0.7803</td>
<td>0.4072</td>
<td>0.5459</td>
<td>0.3994</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.9697</td>
<td>0.3467</td>
<td>0.7686</td>
<td>0.9268</td>
<td>0.5303</td>
<td>0.1572</td>
<td>0.2959</td>
<td>0.1494</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.4697</td>
<td>0.8467</td>
<td>0.2686</td>
<td>0.4268</td>
<td>0.0303</td>
<td>0.6572</td>
<td>0.7959</td>
<td>0.6494</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.3447</td>
<td>0.4717</td>
<td>0.1436</td>
<td>0.3018</td>
<td>0.1553</td>
<td>0.7822</td>
<td>0.4209</td>
<td>0.0244</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.8447</td>
<td>0.9717</td>
<td>0.6436</td>
<td>0.8018</td>
<td>0.6553</td>
<td>0.2822</td>
<td>0.9209</td>
<td>0.5244</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.5947</td>
<td>0.2217</td>
<td>0.3936</td>
<td>0.0518</td>
<td>0.9053</td>
<td>0.0322</td>
<td>0.1709</td>
<td>0.7744</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.0947</td>
<td>0.7217</td>
<td>0.8936</td>
<td>0.5518</td>
<td>0.4053</td>
<td>0.5322</td>
<td>0.6709</td>
<td>0.2744</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.0635</td>
<td>0.1904</td>
<td>0.0498</td>
<td>0.4580</td>
<td>0.6240</td>
<td>0.2510</td>
<td>0.9521</td>
<td>0.8057</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0.5635</td>
<td>0.6904</td>
<td>0.5498</td>
<td>0.9580</td>
<td>0.1240</td>
<td>0.7510</td>
<td>0.4521</td>
<td>0.3057</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.8135</td>
<td>0.4404</td>
<td>0.2998</td>
<td>0.2080</td>
<td>0.3740</td>
<td>0.5010</td>
<td>0.7021</td>
<td>0.0557</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>