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
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 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_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

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 already been initialised by a prior call to nag_quasi_random_normal (g05ybc) with state = Nag_QuasiRandom_Init. Random numbers are output via array quasi;
   if 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.
   
   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: 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, . . . , idim − 1.

6: 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.
   
   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 ≤ 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_Con, 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 *
   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.

10: 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: \texttt{iskip} = \langle \text{value} \rangle.
On entry, \texttt{iskip} = \langle \text{value} \rangle.
Constraint: \texttt{iskip} \geq 0.
On entry, \texttt{idim} = \langle \text{value} \rangle.
Constraint: \texttt{idim} \leq 40.
On entry, \texttt{idim} = \langle \text{value} \rangle.
Constraint: \texttt{idim} \geq 2.
On entry \texttt{idim} is not even: \texttt{idim} = \langle \text{value} \rangle.

NE_INITIALIZATION
Incorrect initialisation.

NE_INT_ARRAY_ELEM_CONS
On entry, element \langle \text{value} \rangle of \texttt{std} \leq 0.0.

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 \langle \text{value} \rangle 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.
For more information see \texttt{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. */

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

g05ybc Example Program Results

Calculated value of the integral = 60.119
Exact value of the integral = 60.000