NAG Library Function Document

nag_rand_init_nonrepeatable (g05kgc)

1 Purpose

nag_rand_init_nonrepeatable (g05kgc) initializes the selected base generator to generate a non-repeatable sequence of variates. The base generator can then be used by the group of pseudorandom number functions (see g05khc–g05kjc, g05ncc, g05ndc, g05pdc–g05pjc, g05pxc–g05pzc, g05rcc, g05rdc, g05ryc, g05rzc and g05sac–g05tlc) and the quasi-random scrambled sequence initialization function, nag_quasi_init_scrambled (g05ync).

2 Specification

```
#include <nag.h>
#include <nagg05.h>
void nag_rand_init_nonrepeatable (Nag_BaseRNG genid, Integer subid,
     Integer state[], Integer *lstate, NagError *fail)
```

3 Description

nag_rand_init_nonrepeatable (g05kgc) selects a base generator through the input value of the arguments **genid** and **subid**, and then initializes it based on the values taken from the real-time clock, resulting in the same base generator yielding different sequences of random numbers each time the calling program is run. It should be noted that there is no guarantee of statistical properties between sequences, only within sequences.

A definition of some of the terms used in this description, along with details of the various base generators can be found in the g05 Chapter Introduction.

4 References

L'Ecuyer P and Simard R (2002) *TestU01: a software library in ANSI C for empirical testing of random number generators* Departement d'Informatique et de Recherche Operationnelle, Universite de Montreal http://www.iro.umontreal.ca/~lecuyer

Maclaren N M (1989) The generation of multiple independent sequences of pseudorandom numbers *Appl. Statist.* **38** 351–359

Matsumoto M and Nishimura T (1998) Mersenne twister: a 623-dimensionally equidistributed uniform pseudorandom number generator *ACM Transactions on Modelling and Computer Simulations*

Wichmann B A and Hill I D (2006) Generating good pseudo-random numbers *Computational Statistics and Data Analysis* **51** 1614–1622

Wikramaratna R S (1989) ACORN - a new method for generating sequences of uniformly distributed pseudo-random numbers *Journal of Computational Physics* **83** 16–31

5 Arguments

1: **genid** – Nag_BaseRNG *Input*

On entry: must contain the type of base generator to use.

genid = Nag_Basic
NAG basic generator.

genid = Nag_WichmannHill_I
Wichmann Hill I generator.

genid = Nag_MersenneTwister
Mersenne Twister.

genid = Nag_WichmannHill_II
Wichmann Hill II generator.

genid = Nag_ACORN
ACORN generator.

genid = Nag_MRG32k3a
L'Ecuyer MRG32k3a generator.

See the g05 Chapter Introduction for details of each of the base generators.

Constraint: **genid** = Nag_Basic, Nag_WichmannHill_I, Nag_MersenneTwister, Nag_WichmannHill_II, Nag_ACORN or Nag_MRG32k3a.

2: **subid** – Integer *Input*

On entry: if **genid** = Nag_WichmannHill_I, **subid** indicates which of the 273 sub-generators to use. In this case, the $((|\mathbf{subid}| + 272) \ mod \ 273) + 1$ sub-generator is used.

If **genid** = Nag_ACORN, **subid** indicates the values of k and p to use, where k is the order of the generator, and p controls the size of the modulus, M, with $M = 2^{(p \times 30)}$. If **subid** < 1, the default values of $k = 10$ and $p = 2$ are used, otherwise values for k and p are calculated from the formula, **subid** $= k + 1000(p - 1)$.

If **genid** = Nag_MRG32k3a and **subid** mod $2 = 0$ the range of the generator is set to $(0, 1]$, otherwise the range is set to $(0, 1)$; in this case the sequence is identical to the implementation of MRG32k3a in TestU01 (see L'Ecuyer and Simard (2002)) for identical seeds.

For all other values of **genid**, **subid** is not referenced.

3: **state**[**lstate**] – Integer *Communication Array*

On exit: contains information on the selected base generator and its current state. If **lstate** < 1 then **state** may be **NULL**.

4: **lstate** – Integer * *Input/Output*

On entry: the dimension of the **state** array, or a value < 1. If the Mersenne Twister (**genid** = Nag_MersenneTwister) is being used and the skip ahead function nag_rand_skip_ahead (g05kjc) or nag_rand_skip_ahead_power2 (g05kkc) will be called subsequently, then you must ensure that **lstate** ≥ 1260.

On exit: if **lstate** < 1 on entry, then the required length of the **state** array for the chosen base generator, otherwise **lstate** is unchanged. When **genid** = Nag_MersenneTwister (Mersenne Twister) a value of 1260 is returned, allowing for the skip ahead function to be subsequently called. In all other cases the minimum length, as documented in the constraints below, is returned.

Constraints:

if **genid** = Nag_Basic, **lstate** ≥ 17;
if **genid** = Nag_WichmannHill_I, **lstate** ≥ 21;
if **genid** = Nag_MersenneTwister, **lstate** ≥ 633;
if **genid** = Nag_WichmannHill_II, **lstate** ≥ 29;
if **genid** = Nag_ACORN, **lstate** $\geq \max((k + 1) \times p + 9, 14) + 3$, where k and p are defined by **subid**;
if **genid** = Nag_MRG32k3a, **lstate** ≥ 61;
otherwise **lstate** < 1.

5: **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_INT

On entry, **lstate** = ⟨*value*⟩.
Constraint: **lstate** ≤ 0 or **lstate** $\geq$ ⟨*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 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.

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_rand_init_nonrepeatable (g05kgc) is not threaded in any implementation.

9 Further Comments

None.

10 Example

In order to preserve the statistical properties of the base generators, nag_rand_init_nonrepeatable (g05kgc) should only be called once. If multiple streams of values are required then one of the methods described in Section 2.1.1 in the g05 Chapter Introduction should be used.

However, for illustrative purposes only, this example calls nag_rand_init_nonrepeatable (g05kgc) twice. At each call a sample of 500 values from a discrete uniform distribution are generated and then the two samples are compared.

10.1 Program Text

```
/* nag_rand_init_nonrepeatable (g05kgc) 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>

int main(void)
{
  /* Integer scalar and array declarations */
  Integer exit_status = 0;
  Integer i, lstate, same;
  Integer *state = 0, *x1 = 0, *x2 = 0;
  /* NAG structures */
  NagError fail;
  /* Set the sample size */
  Integer n = 500;
  /* Choose the base generator */
  Nag_BaseRNG genid = Nag_MersenneTwister;
  Integer subid = 0;

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

  printf("nag_rand_init_nonrepeatable (g05kgc) Example Program Results\n\n");

  /* Get the length of the state array */
  lstate = -1;
  nag_rand_init_nonrepeatable(genid, subid, state, &lstate, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_rand_init_nonrepeatable (g05kgc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

  /* Allocate arrays */
  if (!(x1 = NAG_ALLOC(n, Integer)) ||
      !(x2 = NAG_ALLOC(n, Integer)) ||
      !(state = NAG_ALLOC(lstate, Integer)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Initialize the generator to a non-repeatable sequence */
  nag_rand_init_nonrepeatable(genid, subid, state, &lstate, &fail);

  if (fail.code != NE_NOERROR) {
    printf("Error from nag_rand_init_nonrepeatable (g05kgc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

  /* Generate a sample of values from a discrete uniform distribution */
  nag_rand_discrete_uniform(n, -100, 100, state, x1, &fail);

  /* Re-initialize the generator to another non-repeatable sequence
     NB: In practice, in order to preserve its statistical properties,
     you should only initialize the RNG generators once */
  nag_rand_init_nonrepeatable(genid, subid, state, &lstate, &fail);

  /* Generate a second sample of values from the same distribution */
  nag_rand_discrete_uniform(n, -100, 100, state, x2, &fail);

  /* Check that the two samples are different */
  same = 1;
  for (i = 0; i < n; i++) {
    if (x1[i] != x2[i]) {
```

```
      same = 0;
      break;
    }
  }
  if (same) {
    printf("The two samples are the same\n");
    printf("whilst this is possible, it is unlikely\n");
  } else {
    printf("The two samples differ, as expected\n");
  }

END:
  NAG_FREE(x1);
  NAG_FREE(x2);
  NAG_FREE(state);

  return exit_status;
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_rand_init_nonrepeatable (g05kgc) Example Program Results

The two samples differ, as expected
```