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, g05rde, g05rhc, g05rzc and g05sac–g05tlc) and the quasi-random scrambled sequence initialization function, nag_quasi_init_scrambled (g05ync).

2 Specification

```c
#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


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.

Mark 25
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  

On entry: if genid = Nag_WichmannHill_I, subid indicates which of the 273 sub-generators to use. In this case, the \((|\text{subid}| + 272) \mod 273 + 1\) 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 \geq 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 \geq 17;
if genid = Nag_WichmannHill_I, lstate \geq 21;
if genid = Nag_MersenneTwister, lstate \geq 633;
if genid = Nag_WichmannHill_II, lstate \geq 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 \geq 61;
otherwise lstate < 1.

5: fail – NagError *  

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 ⟨value⟩ had an illegal value.

**NE_INT**
On entry, lstate = ⟨value⟩.
Constraint: lstate ≤ 0 or lstate ≥ ⟨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 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.

7 Accuracy
Not applicable.

8 Parallelism and Performance
Not applicable.

9 Further Comments
None.

10 Example
This example prints the first five pseudorandom real numbers from a uniform distribution between 0 and 1, generated by nag_rand_basic (g05sac) after initialization by nag_rand_init_nonrepeatable (g05kgc).

10.1 Program Text
/* nag_rand_init_nonrepeatable (g05kgc) 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>
#include <nagg05.h>
int main(void)
{
/* Integer scalar and array declarations */
Integer exit_status = 0;
Integer i, lstate;
Integer *state = 0;

/* NAG structures */
NagError fail;

/* Double scalar and array declarations */
double *x = 0;

/* Set the sample size */
Integer n = 5;

/* Choose the base generator */
Nag_BaseRNG genid = Nag_Basic;
Integer subid = 0;

/* Initialise 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\n", fail.message);
    exit_status = 1;
    goto END;
}

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

/* Initialise the generator to a 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\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Generate the variates*/
nag_rand_basic(n, state, x, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_rand_basic (g05sac).\n\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Display the variates */
for (i = 0; i < n; i++)
{
    printf("%10.4f\n", x[i]);
}

END:
NAG_FREE(x);
NAG_FREE(state);
return exit_status;
}

10.2 Program Data
None.

10.3 Program Results
nag_rand_init_nonrepeatable (g05kgc) Example Program Results

0.9234
0.4207
0.8461
0.3662
0.8180