NAG Toolbox

nag_rand_init_skipahead (g05kj)

1 Purpose

nag_rand_init_skipahead (g05kj) allows for the generation of multiple, independent, sequences of pseudorandom numbers using the skip-ahead method.

The base pseudorandom number sequence defined by **state** is advanced n places.

2 Syntax

```
[state, ifail] = nag_rand_init_skipahead(n, state)
[state, ifail] = g05kj(n, state)
```

3 Description

nag_rand_init_skipahead (g05kj) adjusts a base generator to allow multiple, independent, sequences of pseudorandom numbers to be generated via the skip-ahead method (see the G05 Chapter Introduction for details).

If, prior to calling nag_rand_init_skipahead (g05kj) the base generator defined by **state** would produce random numbers $x_1, x_2, x_3, \ldots$, then after calling nag_rand_init_skipahead (g05kj) the generator will produce random numbers $x_{n+1}, x_{n+2}, x_{n+3}, \ldots$.

One of the initialization functions nag_rand_init_repeat (g05kf) (for a repeatable sequence if computed sequentially) or nag_rand_init_nonrepeat (g05kg) (for a non-repeatable sequence) must be called prior to the first call to nag_rand_init_skipahead (g05kj).

The skip-ahead algorithm can be used in conjunction with any of the six base generators discussed in Chapter G05.

4 References

Haramoto H, Matsumoto M, Nishimura T, Panneton F and L'Ecuyer P (2008) Efficient jump ahead for F2-linear random number generators *INFORMS J. on Computing* **20(3)** 385–390

Knuth D E (1981) *The Art of Computer Programming (Volume 2)* (2nd Edition) Addison–Wesley

5 Parameters

5.1 Compulsory Input Parameters

1: **n** – INTEGER

n, the number of places to skip ahead.

Constraint: $\mathbf{n} \geq 0$.

2: **state**(:) – INTEGER array

Note: the actual argument supplied **must** be the array **state** supplied to the initialization routines nag_rand_init_repeat (g05kf) or nag_rand_init_nonrepeat (g05kg).

Contains information on the selected base generator and its current state.

5.2 Optional Input Parameters

None.

5.3 Output Parameters

1: **state**(:) – INTEGER array

Contains updated information on the state of the generator.

2: **ifail** – INTEGER

ifail $= 0$ unless the function detects an error (see Section 5).

6 Error Indicators and Warnings

Errors or warnings detected by the function:

ifail $= 1$

Constraint: **n** ≥ 0.

ifail $= 2$

On entry, **state** vector has been corrupted or not initialized.

ifail $= 3$

On entry, cannot use skip-ahead with the base generator defined by **state**.

ifail $= 4$

On entry, the base generator is Mersenne Twister, but the **state** vector defined on initialization is not large enough to perform a skip ahead. See the initialization function nag_rand_init_repeat (g05kf) or nag_rand_init_nonrepeat (g05kg).

ifail $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

ifail $= -399$

Your licence key may have expired or may not have been installed correctly.

ifail $= -999$

Dynamic memory allocation failed.

7 Accuracy

Not applicable.

8 Further Comments

Calling nag_rand_init_skipahead (g05kj) and then generating a series of uniform values using nag_rand_dist_uniform01 (g05sa) is more efficient than, but equivalent to, calling nag_rand_dist_uniform01 (g05sa) and discarding the first n values. This may not be the case for distributions other than the uniform, as some distributional generators require more than one uniform variate to generate a single draw from the required distribution.

To skip ahead $k \times m$ places you can either

(a) call nag_rand_init_skipahead (g05kj) once with **n** $= k \times m$, or

(b) call nag_rand_init_skipahead (g05kj) k times with **n** $= m$, using the **state** vector output by the previous call as input to the next call

both approaches would result in the same sequence of values. When working in a multithreaded environment, where you want to generate (at most) m values on each of K threads, this would translate into either

(a) spawning the K threads and calling nag_rand_init_skipahead (g05kj) once on each thread with $\mathbf{n} = (k-1) \times m$, where k is a thread ID, taking a value between 1 and K, or

(b) calling nag_rand_init_skipahead (g05kj) on a single thread with $\mathbf{n} = m$, spawning the K threads and then calling nag_rand_init_skipahead (g05kj) a further $k-1$ times on each of the thread.

Due to the way skip ahead is implemented for the Mersenne Twister, approach (a) will tend to be more efficient if more than 30 threads are being used (i.e., $K > 30$), otherwise approach (b) should probably be used. For all other base generators, approach (a) should be used. See the G05 Chapter Introduction for more details.

9 Example

This example initializes a base generator using nag_rand_init_repeat (g05kf) and then uses nag_rand_init_skipahead (g05kj) to advance the sequence 50 places before generating five variates from a uniform distribution using nag_rand_dist_uniform01 (g05sa).

9.1 Program Text

```
     function g05kj_example

fprintf('g05kj example results\n\n');

% Initialize the seed
seed = [nag_int(1762543)];

% genid and subid identify the base generator
genid = nag_int(1);
subid =  nag_int(1);
lseed =  nag_int(1);

% Initialize the generator to a repeatable sequence
[state, ifail] = g05kf( ...
                        genid, subid, seed);

% Advance the sequence n places
n = nag_int(50);
[state, ifail] = g05kj( ...
                        n, state);

% Generate nv variates from a uniform distribution
nv = nag_int(5);
[state, x, ifail] = g05sa(...
                          nv, state);

% Display variates
disp(x);
```

9.2 Program Results

```
     g05kj example results

    0.2071
    0.8413
    0.8817
    0.5494
    0.5248
```