

NAG Library Function Document

nag_rand_skip Ahead (g05kjc)

1 Purpose

nag_rand_skip Ahead (g05kjc) 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 Specification

```
#include <nag.h>
#include <nagg05.h>
void nag_rand_skip Ahead (Integer n, Integer state[], NagError *fail)
```

3 Description

nag_rand_skip Ahead (g05kjc) 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_skip Ahead (g05kjc) the base generator defined by **state** would produce random numbers x_1, x_2, x_3, \dots , then after calling nag_rand_skip Ahead (g05kjc) the generator will produce random numbers $x_{n+1}, x_{n+2}, x_{n+3}, \dots$.

One of the initialization functions nag_rand_init_repeatable (g05kfc) (for a repeatable sequence if computed sequentially) or nag_rand_init_nonrepeatable (g05kgc) (for a non-repeatable sequence) must be called prior to the first call to nag_rand_skip Ahead (g05kjc).

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 Arguments

1: **n** – Integer *Input*

On entry: n , the number of places to skip ahead.

Constraint: $n \geq 0$.

2: **state**[*dim*] – Integer *Communication Array*

Note: the dimension, *dim*, of this array is dictated by the requirements of associated functions that must have been previously called. This array **MUST** be the same array passed as argument **state** in the previous call to nag_rand_init_repeatable (g05kfc) or nag_rand_init_nonrepeatable (g05kgc).

On entry: contains information on the selected base generator and its current state.

On exit: contains updated information on the state of the generator.

3: **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_ARRAY_SIZE

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_repeatable` (g05kfc) or `nag_rand_init_nonrepeatable` (g05kgc).

NE_BAD_PARAM

On entry, argument $\langle value \rangle$ had an illegal value.

NE_INT

On entry, $n = \langle value \rangle$.

Constraint: $n \geq 0$.

NE_INT_ARRAY

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

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_INVALID_STATE

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

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

Calling `nag_rand_skip Ahead` (g05kjc) and then generating a series of uniform values using `nag_rand_basic` (g05sac) is more efficient than, but equivalent to, calling `nag_rand_basic` (g05sac) 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_skip_ahead (g05kjc)` once with $\mathbf{n} = k \times m$, or
- (b) call `nag_rand_skip_ahead (g05kjc)` k times with $\mathbf{n} = m$, 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_skip_ahead (g05kjc)` 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_skip_ahead (g05kjc)` on a single thread with $\mathbf{n} = m$, spawning the K threads and then calling `nag_rand_skip_ahead (g05kjc)` 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.

10 Example

This example initializes a base generator using `nag_rand_init_repeatable (g05kfc)` and then uses `nag_rand_skip_ahead (g05kjc)` to advance the sequence 50 places before generating five variates from a uniform distribution using `nag_rand_basic (g05sac)`.

10.1 Program Text

```

/* nag_rand_skip_ahead (g05kjc) 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    nv = 5;

    /* Set the number of elements to advance the sequence */
    Integer    n = 50;

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

    /* Set the seed */
    Integer    seed[] = { 1762543 };

```

```

Integer      lseed = 1;

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

printf("nag_rand_skip_ahead (g05kjc) Example Program Results\n\n");

/* Get the length of the state array */
lstate = -1;
nag_rand_init_repeatabe(genid, subid, seed, lseed, state, &lstate, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_rand_init_repeatabe (g05kfc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
}

/* Allocate arrays */
if (!(x = NAG_ALLOC(nv, 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_repeatabe(genid, subid, seed, lseed, state, &lstate, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_rand_init_repeatabe (g05kfc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
}

/* Advance the sequence N places*/
nag_rand_skip_ahead(n, state, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_rand_skip_ahead (g05kjc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
}

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

/* Display the variates*/
for (i = 0; i < nv; i++)
    printf("%11.4f\n", x[i]);

END:
NAG_FREE(x);
NAG_FREE(state);

return exit_status;
}

```

10.2 Program Data

None.

10.3 Program Results

nag_rand_skip_ahead (g05kjc) Example Program Results

0.2071
0.8413
0.8817
0.5494
0.5248
