NAG Library Function Document

nag_rand_leap_frog (g05khc)

1 Purpose

nag_rand_leap_frog (g05khc) allows for the generation of multiple, independent, sequences of pseudorandom numbers using the leap-frog method.

2 Specification

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

3 Description

nag_rand_leap_frog (g05khc) adjusts a base generator to allow multiple, independent, sequences of pseudorandom numbers to be generated via the leap-frog method (see the g05 Chapter Introduction for details).

If, prior to calling nag_rand_leap_frog (g05khc) the base generator defined by **state** would produce random numbers $x_1, x_2, x_3, \ldots$, then after calling nag_rand_leap_frog (g05khc) the generator will produce random numbers $x_k, x_{k+n}, x_{k+2n}, x_{k+3n}, \ldots$.

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_leap_frog (g05khc).

The leap-frog algorithm can be used in conjunction with the NAG basic generator, both the Wichmann–Hill I and Wichmann–Hill II generators, the Mersenne Twister and L'Ecuyer.

4 References

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 total number of sequences required.

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

2: **k** – Integer *Input*

On entry: k, the number of the current sequence.

Constraint: $0 < \mathbf{k} \leq \mathbf{n}$.

3: **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.

4: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_INT

On entry, $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{n} \geq 1$.

NE_INT_2

On entry, $\mathbf{k} = \langle value \rangle$ and $\mathbf{n} = \langle value \rangle$.
Constraint: $0 < \mathbf{k} \leq \mathbf{n}$.

NE_INT_ARRAY

On entry, cannot use leap-frog 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.

NE_INVALID_STATE

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

7 Accuracy

Not applicable.

8 Parallelism and Performance

Not applicable.

9 Further Comments

The leap-frog method tends to be less efficient than other methods of producing multiple, independent sequences. See the g05 Chapter Introduction for alternative choices.

10 Example

This example creates three independent sequences using nag_rand_leap_frog (g05khc), after initialization by nag_rand_init_repeatable (g05kfc). Five variates from a uniform distribution are then generated from each sequence using nag_rand_basic (g05sac).

10.1 Program Text

```
/* nag_rand_leap_frog (g05khc) Example Program.
 *
 * Copyright 2008, 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>

#define STATE(I, J) state[(J - 1)*lstate + I - 1]

int main(void)
{
  /* Integer scalar and array declarations */
  Integer     exit_status = 0;
  Integer     i, j, 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 streams */
  Integer     n = 3;
  /* 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_leap_frog (g05khc) Example Program Results\n\n");

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

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

  /* Prepare n streams */
  for (i = 1; i <= n; i++)
    {
      /* Initialise the I'th stream to a repeatable sequence, using the same
         seed for each stream */
      nag_rand_init_repeatable(genid, subid, seed, lseed, &STATE(1, i),
                               &lstate, &fail);
      if (fail.code != NE_NOERROR)
        {
          printf("Error from nag_rand_init_repeatable (g05kfc).\n%s\n",
                  fail.message);
          exit_status = 1;
          goto END;
        }

      /* Prepare the I'th out of N streams*/
      nag_rand_leap_frog(n, i, &STATE(1, i), &fail);
      if (fail.code != NE_NOERROR)
```

```
        {
          printf("Error from nag_rand_leap_frog (g05khc).\n%s\n",
                  fail.message);
          exit_status = 1;
          goto END;
        }
    }

  /* Loop over each of the n streams */
  for (i = 1; i <= n; i++)
    {
      /* Generate nv variates from a univariate distribution */
      printf(" Stream %12ld\n", i);
      nag_rand_basic(nv, &STATE(1, i), 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 (j = 0; j < nv; j++)
        printf("%11.4f\n", x[j]);
      printf(" \n");
    }

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

  return exit_status;
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_rand_leap_frog (g05khc) Example Program Results

 Stream           1
     0.7460
     0.4925
     0.4982
     0.2580
     0.5938

 Stream           2
     0.7983
     0.3843
     0.6717
     0.6238
     0.2785

 Stream           3
     0.1046
     0.7871
     0.0505
     0.0535
     0.2375
```