NAG Library Function Document

nag_rand_logarithmic (g05tfc)

1 Purpose

nag_rand_logarithmic (g05tfc) generates a vector of pseudorandom integers from the discrete logarithmic distribution with parameter a.

2 Specification

```
#include <nag.h>
#include <nagg05.h>
void nag_rand_logarithmic (Nag_ModeRNG mode, Integer n, double a, double r[],
     Integer lr, Integer state[], Integer x[], NagError *fail)
```

3 Description

nag_rand_logarithmic (g05tfc) generates n integers x_i from a discrete logarithmic distribution, where the probability of $x_i = I$ is

$$P(x_i = I) = -\frac{a^I}{I \times \log(1-a)}, \quad I = 1, 2, \ldots,$$

where $0 < a < 1$.

The variates can be generated with or without using a search table and index. If a search table is used then it is stored with the index in a reference vector and subsequent calls to nag_rand_logarithmic (g05tfc) with the same parameter value can then use this reference vector to generate further variates.

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_logarithmic (g05tfc).

4 References

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

5 Arguments

1: **mode** – Nag_ModeRNG *Input*

On entry: a code for selecting the operation to be performed by the function.

mode = Nag_InitializeReference
Set up reference vector only.

mode = Nag_GenerateFromReference
Generate variates using reference vector set up in a prior call to nag_rand_logarithmic (g05tfc).

mode = Nag_InitializeAndGenerate
Set up reference vector and generate variates.

mode = Nag_GenerateWithoutReference
Generate variates without using the reference vector.

Constraint: **mode** = Nag_InitializeReference, Nag_GenerateFromReference, Nag_InitializeAndGenerate or Nag_GenerateWithoutReference.

2: **n** – Integer *Input*

On entry: n, the number of pseudorandom numbers to be generated.

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

3: **a** – double *Input*

On entry: a, the parameter of the logarithmic distribution.

Constraint: $0.0 < \mathbf{a} < 1.0$.

4: **r**[**lr**] – double *Communication Array*

On entry: if **mode** = Nag_GenerateFromReference, the reference vector from the previous call to nag_rand_logarithmic (g05tfc).

If **mode** = Nag_GenerateWithoutReference, **r** is not referenced and may be **NULL**.

On exit: **mode** ≠ Nag_GenerateWithoutReference, the reference vector.

5: **lr** – Integer *Input*

On entry: the dimension of the array **r**.

Suggested value:

if **mode** ≠ Nag_GenerateWithoutReference, $\mathbf{lr} = 18 + \frac{40}{1-\mathbf{a}}$;
otherwise $\mathbf{lr} = 1$.

Constraints:

if **mode** = Nag_InitializeReference or Nag_InitializeAndGenerate, **lr** must not be too small, but the lower limit is too complicated to specify;
if **mode** = Nag_GenerateFromReference, **lr** must remain unchanged from the previous call to nag_rand_logarithmic (g05tfc).

6: **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.

7: **x**[**n**] – Integer *Output*

On exit: the n pseudorandom numbers from the specified logarithmic distribution.

8: **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, **lr** is too small when **mode** = Nag_InitializeReference or Nag_InitializeAndGenerate: **lr** = ⟨*value*⟩, minimum length required = ⟨*value*⟩.

On entry, **n** = ⟨*value*⟩.
Constraint: **n** ≥ 0.

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.

NE_PREV_CALL

The value of **a** is not the same as when **r** was set up in a previous call.
Previous value of **a** = ⟨*value*⟩ and **a** = ⟨*value*⟩.

NE_REAL

On entry, **a** = ⟨*value*⟩.
Constraint: 0.0 < **a** < 1.0.

NE_REF_VEC

On entry, some of the elements of the array **r** have been corrupted or have not been initialized.

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_rand_logarithmic (g05tfc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

None.

10 Example

This example prints 10 pseudorandom integers from a logarithmic distribution with parameter $a = 0.9999$, generated by a single call to nag_rand_logarithmic (g05tfc), after initialization by nag_rand_init_repeatable (g05kfc).

10.1 Program Text

```
/* nag_rand_logarithmic (g05tfc) 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     lr, i, lstate;
  Integer     *state = 0, *x = 0;

  /* NAG structures */
  NagError    fail;
  Nag_ModeRNG mode;

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

  /* Set the distribution parameters */
  double      a = 0.99990e0;

  /* Set the sample size */
  Integer     n = 10;

  /* 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_logarithmic (g05tfc) 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;
    }

  /* Calculate the size of the reference vector,
     we are not using r, so lr can be set to 0 */
  lr = 0;

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

```
    }

  /* Initialise the generator to a repeatable sequence */
  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;
    }

  /* Generate the variates, dont use a reference vector
     as argument a is close to 1 */
  mode = Nag_GenerateWithoutReference;
  nag_rand_logarithmic(mode, n, a, r, lr, state, x, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_rand_logarithmic (g05tfc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

  /* Display the variates*/
  for (i = 0; i < n; i++)
    printf("%12"NAG_IFMT"\n", x[i]);

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

  return exit_status;
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_rand_logarithmic (g05tfc) Example Program Results

           6
          23
        2765
          30
           3
           1
         299
         968
         166
           4
```