

NAG Library Function Document

nag_rand_logistic (g05slc)

1 Purpose

nag_rand_logistic (g05slc) generates a vector of pseudorandom numbers from a logistic distribution with mean a and spread b .

2 Specification

```
#include <nag.h>
#include <nagg05.h>
void nag_rand_logistic (Integer n, double a, double b, Integer state[],
    double x[], NagError *fail)
```

3 Description

The distribution has PDF (probability density function)

$$f(x) = \frac{e^{(x-a)/b}}{b(1 + e^{(x-a)/b})^2}.$$

nag_rand_logistic (g05slc) returns the value

$$a + b \ln\left(\frac{y}{1-y}\right),$$

where y is a pseudorandom number uniformly distributed over $(0, 1)$.

One of the initialization functions nag_rand_init_repeatable (g05kfc) (for a repeatable sequence if computed sequentially) or nag_rand_init_nonrepeatable (g05kge) (for a non-repeatable sequence) must be called prior to the first call to nag_rand_logistic (g05slc).

4 References

Kendall M G and Stuart A (1969) *The Advanced Theory of Statistics (Volume 1)* (3rd Edition) Griffin
 Knuth D E (1981) *The Art of Computer Programming (Volume 2)* (2nd Edition) Addison–Wesley

5 Arguments

- | | | |
|----|--|--------------|
| 1: | n – Integer | <i>Input</i> |
| | <i>On entry:</i> n , the number of pseudorandom numbers to be generated. | |
| | <i>Constraint:</i> $n \geq 0$. | |
| 2: | a – double | <i>Input</i> |
| | <i>On entry:</i> a , the mean of the distribution. | |
| 3: | b – double | <i>Input</i> |
| | <i>On entry:</i> b , the spread of the distribution, where ‘spread’ is $\frac{\sqrt{3}}{\pi} \times$ standard deviation. | |
| | <i>Constraint:</i> $b \geq 0.0$. | |

- 4: **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.
- 5: **x**[**n**] – double *Output*
On exit: the *n* pseudorandom numbers from the specified logistic distribution.
- 6: **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, **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.

NE_INVALID_STATE

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

NE_REAL

On entry, **b** = *<value>*.
Constraint: **b** ≥ 0.0.

7 Accuracy

Not applicable.

8 Parallelism and Performance

`nag_rand_logistic` (g05slc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

Please consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

None.

10 Example

This example prints the first five pseudorandom real numbers from a logistic distribution with mean 1.0 and spread 2.0, generated by a single call to `nag_rand_logistic` (g05slc), after initialization by `nag_rand_init_repeatabl` (g05kfc).

10.1 Program Text

```

/* nag_rand_logistic (g05slc) 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>

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 distribution parameters */
    double     a = 1.0e0;
    double     b = 2.0e0;

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

    /* 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_logistic (g05slc) Example Program Results\n\n");

    /* Get the length of the state array */
    lstate = -1;
    nag_rand_init_repeatabl(genid, subid, seed, lseed, state, &lstate, &fail);
    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_rand_init_repeatabl (g05kfc).\n%s\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_repeatable(genid, subid, seed, lseed, state, &state, &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*/
nag_rand_logistic(n, a, b, state, x, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_rand_logistic (g05slc).\n%s\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_logistic (g05slc) Example Program Results

```

2.1193
-3.2544
3.1552
3.7510
-3.2944

```
