NAG Library Function Document

nag_rand_gamma (g05sjc)

1 Purpose

nag_rand_gamma (g05sjc) generates a vector of pseudorandom numbers taken from a gamma distribution with parameters a and b.

2 Specification

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

3 Description

The gamma distribution has PDF (probability density function)

$$f(x) = \frac{1}{b^a \Gamma(a)} x^{a-1} e^{-x/b} \quad \text{if } x \geq 0; \quad a, b > 0$$

$$f(x) = 0 \qquad \text{otherwise.}$$

One of three algorithms is used to generate the variates depending upon the value of a:

(i) if $a < 1$, a switching algorithm described by Dagpunar (1988) (called G6) is used. The target distributions are $f_1(x) = cax^{a-1}/t^a$ and $f_2(x) = (1-c)e^{-(x-t)}$, where $c = t/(t + ae^{-t})$, and the switching argument, t, is taken as $1 - a$. This is similar to Ahrens and Dieter's GS algorithm (see Ahrens and Dieter (1974)) in which $t = 1$;

(ii) if $a = 1$, the gamma distribution reduces to the exponential distribution and the method based on the logarithmic transformation of a uniform random variate is used;

(iii) if $a > 1$, the algorithm given by Best (1978) is used. This is based on using a Student's t-distribution with two degrees of freedom as the target distribution in an envelope rejection method.

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_gamma (g05sjc).

4 References

Ahrens J H and Dieter U (1974) Computer methods for sampling from gamma, beta, Poisson and binomial distributions *Computing* **12** 223–46

Best D J (1978) Letter to the Editor *Appl. Statist.* **27** 181

Dagpunar J (1988) *Principles of Random Variate Generation* Oxford University Press

Hastings N A J and Peacock J B (1975) *Statistical Distributions* Butterworth

5 Arguments

1: **n** – Integer *Input*

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

Constraint: **n** ≥ 0.

2: **a** – double *Input*

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

Constraint: **a** > 0.0.

3: **b** – double *Input*

On entry: b, the parameter of the gamma distribution.

Constraint: **b** > 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 gamma distribution.

6: **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, **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_REAL

On entry, $\mathbf{a} = \langle value \rangle$.
Constraint: $\mathbf{a} > 0.0$.

On entry, $\mathbf{b} = \langle value \rangle$.
Constraint: $\mathbf{b} > 0.0$.

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_rand_gamma (g05sjc) 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 a set of five pseudorandom numbers from a gamma distribution with parameters $a = 5.0$ and $b = 1.0$, generated by a single call to nag_rand_gamma (g05sjc), after initialization by nag_rand_init_repeatable (g05kfc).

10.1 Program Text

```
/* nag_rand_gamma (g05sjc) 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 distribution parameters */
  double      a = 5.0e0;
  double      b = 1.0e0;

  /* 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_gamma (g05sjc) 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(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, &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*/
  nag_rand_gamma(n, a, b, state, x, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_rand_gamma (g05sjc).\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_gamma (g05sjc) Example Program Results

    5.0702
    6.1337
    3.1018
    3.9863
    4.9648
```