NAG Library Function Document

nag_rand_beta (g05sbc)

1 Purpose

nag_rand_beta (g05sbc) generates a vector of pseudorandom numbers taken from a beta distribution with parameters a and b.

2 Specification

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

3 Description

The beta distribution has PDF (probability density function)

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

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

One of four algorithms is used to generate the variates depending on the values of a and b. Let α be the maximum and β be the minimum of a and b. Then the algorithms are as follows:

(i) if $\alpha < 0.5$, Johnk's algorithm is used, see for example Dagpunar (1988). This generates the beta variate as $u_1^{1/a} / \left(u_1^{1/a} + u_2^{1/b} \right)$, where u_1 and u_2 are uniformly distributed random variates;

(ii) if $\beta > 1$, the algorithm BB given by Cheng (1978) is used. This involves the generation of an observation from a beta distribution of the second kind by the envelope rejection method using a log-logistic target distribution and then transforming it to a beta variate;

(iii) if $\alpha > 1$ and $\beta < 1$, the switching algorithm given by Atkinson (1979) is used. The two target distributions used are $f_1(x) = \beta x^{\beta}$ and $f_2(x) = \alpha(1-x)^{\beta-1}$, along with the approximation to the switching argument of $t = (1-\beta)/(\alpha+1-\beta)$;

(iv) in all other cases, Cheng's BC algorithm (see Cheng (1978)) is used with modifications suggested by Dagpunar (1988). This algorithm is similar to BB, used when $\beta > 1$, but is tuned for small values of a and b.

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_beta (g05sbc).

4 References

Atkinson A C (1979) A family of switching algorithms for the computer generation of beta random variates *Biometrika* **66** 141–5

Cheng R C H (1978) Generating beta variates with nonintegral shape parameters *Comm. ACM* **21** 317–322

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

2: **a** – double *Input*

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

Constraint: $\mathbf{a} > 0.0$.

3: **b** – double *Input*

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

Constraint: $\mathbf{b} > 0.0$.

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 beta 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, $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{n} \geq 0$.

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, $\mathbf{a} = \langle value \rangle$.
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_beta (g05sbc) 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

To generate an observation, y, from the beta distribution of the second kind from an observation, x, generated by nag_rand_beta (g05sbc) the transformation, $y = x/(1-x)$, may be used.

10 Example

This example prints a set of five pseudorandom numbers from a beta distribution with parameters $a = 2.0$ and $b = 2.0$, generated by a single call to nag_rand_beta (g05sbc), after initialization by nag_rand_init_repeatable (g05kfc).

10.1 Program Text

```
/* nag_rand_beta (g05sbc) 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 = 2.0e0;
  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_beta (g05sbc) 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_beta(n, a, b, state, x, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_rand_beta (g05sbc).\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_beta (g05sbc) Example Program Results

    0.5977
    0.6818
    0.1797
    0.4174
    0.4987
```