NAG Library Function Document

nag_rand_gen_discrete (g05tdc)

1 Purpose

nag_rand_gen_discrete (g05tdc) generates a vector of pseudorandom integers from a discrete distribution with a given PDF (probability density function) or CDF (cumulative distribution function) p.

2 Specification

```
#include <nag.h>
#include <nagg05.h>
```

```
void nag_rand_gen_discrete (Nag_ModeRNG mode, Integer n, const double p[],
     Integer np, Integer ip1, Nag_DiscreteDistrib itype, double r[],
     Integer lr, Integer state[], Integer x[], NagError *fail)
```

3 Description

nag_rand_gen_discrete (g05tdc) generates a sequence of n integers x_i, from a discrete distribution defined by information supplied in **p**. This may either be the PDF or CDF of the distribution. A reference vector is first set up to contain the CDF of the distribution in its higher elements, followed by an index.

Setting up the reference vector and subsequent generation of variates can each be performed by separate calls to nag_rand_gen_discrete (g05tdc) or may be combined in a single call.

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_gen_discrete (g05tdc).

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: **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_gen_discrete (g05tdc).

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: **p**[**np**] – const double *Input*

On entry: the PDF or CDF of the distribution.

Constraints:

$0.0 \leq \mathbf{p}[i-1] \leq 1.0$, for $i = 1, 2, \ldots, \mathbf{np}$;
if **itype** = Nag_PDF, $\sum_{i=1}^{\mathbf{np}} \mathbf{p}[i-1] = 1.0$;
if **itype** = Nag_CDF, $\mathbf{p}[i-1] < \mathbf{p}[j-1]$, $i < j$ and $\mathbf{p}[\mathbf{np}-1] = 1.0$.

4: **np** – Integer *Input*

On entry: the number of values supplied in **p** defining the PDF or CDF of the discrete distribution.

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

5: **ip1** – Integer *Input*

On entry: the value of the variate, a whole number, to which the probability in **p**[0] corresponds.

6: **itype** – Nag_DiscreteDistrib *Input*

On entry: indicates the type of information contained in **p**.

itype = Nag_PDF
  p contains a probability distribution function (PDF).

itype = Nag_CDF
  p contains a cumulative distribution function (CDF).

Constraint: **itype** = Nag_PDF or Nag_CDF.

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

On entry: if **mode** = Nag_GenerateFromReference, the reference vector from the previous call to nag_rand_gen_discrete (g05tdc).

On exit: the reference vector.

8: **lr** – Integer *Input*

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

Suggested value:

if **mode** $\neq$ Nag_GenerateWithoutReference, $\mathbf{lr} = 10 + 1.4 \times \mathbf{np}$ approximately (for optimum efficiency in generating variates);
otherwise $\mathbf{lr} = 1$.

Constraints:

if **mode** = Nag_InitializeReference or Nag_InitializeAndGenerate, $\mathbf{lr} \geq \mathbf{np} + 8$;
if **mode** = Nag_GenerateFromReference, **lr** should remain unchanged from the previous call to nag_rand_gen_discrete (g05tdc).

9: **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.

10: **x**[**n**] – Integer *Output*

On exit: contains n pseudorandom numbers from the specified discrete distribution.

11: **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.

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.

On entry, **np** = ⟨*value*⟩.
Constraint: **np** > 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_PREV_CALL

The value of **np** or **ip1** is not the same as when **r** was set up in a previous call.
Previous value of **np** = ⟨*value*⟩ and **np** = ⟨*value*⟩.
Previous value of **ip1** = ⟨*value*⟩ and **ip1** = ⟨*value*⟩.

NE_REAL_ARRAY

On entry, at least one element of the vector **p** is less than 0.0 or greater than 1.0.

On entry, **itype** = Nag_CDF and the values of **p** are not all in stricly ascending order.

On entry, **itype** = Nag_PDF and the sum of the elements of **p** do not equal one.

On entry, **p**[**np** − 1] = ⟨*value*⟩.
Constraint: if **itype** = Nag_CDF, **p**[**np** − 1] = 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

Not applicable.

9 Further Comments

None.

10 Example

This example prints 20 pseudorandom variates from a discrete distribution whose PDF, p, is defined as follows:

n	p
-5	0.01
-4	0.02
-3	0.04
-2	0.08
-1	0.20
0	0.30
1	0.20
2	0.08
3	0.04
4	0.02
5	0.01

The reference vector is set up and and the variates are generated by a single call to nag_rand_gen_discrete (g05tdc), after initialization by nag_rand_init_repeatable (g05kfc).

10.1 Program Text

```
/* nag_rand_gen_discrete (g05tdc) 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, lr, 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 */
  Integer             np = 11;
```

```
Nag_DiscreteDistrib itype = Nag_PDF;
double              p[] = { 0.010e0, 0.020e0, 0.040e0, 0.080e0, 0.20e0,
                            0.30e0, 0.20e0, 0.080e0, 0.040e0, 0.020e0,
                            0.010e0 };
Integer             ip1 = -5;

/* Set the sample size */
Integer             n = 20;

/* 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_gen_discrete (g05tdc) 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 */
lr = 10+1.4*np;

/* 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, initialising the reference
   vector at the same time */
mode = Nag_InitializeAndGenerate;
nag_rand_gen_discrete(mode, n, p, np, ip1, itype, r, lr, state, x, &fail);
if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_rand_gen_discrete (g05tdc).\n%s\n",
            fail.message);
    exit_status = 1;
    goto END;
  }

/* Display the variates*/
for (i = 0; i < n; i++)
  printf("%12ld\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_gen_discrete (g05tdc) Example Program Results

           0
          -2
           1
           1
          -2
           0
           0
           1
           0
           1
          -3
          -1
           0
          -3
           0
          -1
          -1
           5
           2
           0
```