NAG Library Function Document

nag_rand_hypergeometric (g05tec)

1 Purpose

nag_rand_hypergeometric (g05tec) generates a vector of pseudorandom integers from the discrete hypergeometric distribution of the number of specified items in a sample of size l, taken from a population of size k with m specified items in it.

2 Specification

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

3 Description

nag_rand_hypergeometric (g05tec) generates n integers x_i from a discrete hypergeometric distribution, where the probability of $x_i = I$ is

$$P(i = I) = \frac{l!m!(k-l)!(k-m)!}{I!(l-I)!(m-I)!(k-m-l+I)!k!} \qquad \text{if } I = \max(0, m+l-k), \ldots, \min(l, m),$$

$$P(i = I) = 0 \qquad \text{otherwise.}$$

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_hypergeometric (g05tec) with the same parameter values can then use this reference vector to generate further variates. The reference array is generated by a recurrence relation if $lm(k-l)(k-m) < 50k^3$, otherwise Stirling's approximation is used.

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_hypergeometric (g05tec).

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_hypergeometric (g05tec).

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: **ns** – Integer *Input*

On entry: l, the sample size of the hypergeometric distribution.

Constraint: $0 \leq \mathbf{ns} \leq \mathbf{np}$.

4: **np** – Integer *Input*

On entry: k, the population size of the hypergeometric distribution.

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

5: **m** – Integer *Input*

On entry: m, the number of specified items of the hypergeometric distribution.

Constraint: $0 \leq \mathbf{m} \leq \mathbf{np}$.

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

On entry: if **mode** = Nag_GenerateFromReference, the reference vector from the previous call to nag_rand_hypergeometric (g05tec).

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

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

7: **lr** – Integer *Input*

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

Suggested value:

if **mode** ≠ Nag_GenerateWithoutReference,
$\mathbf{lr} = 28 + 20 \times \sqrt{(\mathbf{ns} \times \mathbf{m} \times (\mathbf{np} - \mathbf{m}) \times (\mathbf{np} - \mathbf{ns}))/\mathbf{np}^3}$ approximately;
otherwise $\mathbf{lr} = 1$.

Constraints:

if **mode** = Nag_InitializeReference or Nag_InitializeAndGenerate, **lr** must not be too small, but the limit is too complicated to specify;
if **mode** = Nag_GenerateFromReference, **lr** must remain unchanged from the previous call to nag_rand_hypergeometric (g05tec).

8: **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.

9: **x**[**n**] – Integer *Output*

On exit: the pseudorandom numbers from the specified hypergeometric distribution.

10: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 3.2.1.2 in How to Use the NAG Library and its Documentation 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.

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

NE_INT_2

On entry, **m** = ⟨*value*⟩ and **np** = ⟨*value*⟩.
Constraint: $0 \leq$ **m** $\leq$ **np**.

On entry, **ns** = ⟨*value*⟩ and **np** = ⟨*value*⟩.
Constraint: $0 \leq$ **ns** $\leq$ **np**.

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 How to Use the NAG Library and its Documentation 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 How to Use the NAG Library and its Documentation for further information.

NE_PREV_CALL

The value of **ns**, **np** or **m** is not the same as when **r** was set up in a previous call with **mode** = Nag_InitializeReference or Nag_InitializeAndGenerate.

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_hypergeometric (g05tec) 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' Notefor your implementation for any additional implementation-specific information.

9 Further Comments

None.

10 Example

The example program prints 20 pseudorandom integers from a hypergeometric distribution with $l = 500$, $m = 900$ and $n = 1000$, generated by a single call to nag_rand_hypergeometric (g05tec), after initialization by nag_rand_init_repeatable (g05kfc).

10.1 Program Text

```
/* nag_rand_hypergeometric (g05tec) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */
/* 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 */
  Integer ns = 500;
  Integer m = 900;
  Integer np = 1000;

  /* 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;

  /* Initialize the error structure */
  INIT_FAIL(fail);

  printf("nag_rand_hypergeometric (g05tec) 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 = 28 + 20 *
         (int) sqrt(((double) m * ((double) ns / (double) np) *
                     ((double) (np - m) / (double) np) *
                     ((double) (np - ns) / (double) 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;
  }

  /* Initialize 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, initializing the reference
     vector at the same time */
  mode = Nag_InitializeAndGenerate;
  nag_rand_hypergeometric(mode, n, ns, np, m, r, lr, state, x, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_rand_hypergeometric (g05tec).\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_hypergeometric (g05tec) Example Program Results

         452
         444
         453
         454
         444
         450
         449
         454
         450
         452
         442
         447
         451
         442
         451
         447
         447
         462
         456
         450
```