NAG Library Function Document

nag_rand_sample (g05ndc)

1 Purpose

nag_rand_sample (g05ndc) selects a pseudorandom sample without replacement from an integer vector.

2 Specification

```
#include <nag.h>
#include <nagg05.h>
void nag_rand_sample (const Integer ipop[], Integer n, Integer isampl[],
     Integer m, Integer state[], NagError *fail)
```

3 Description

nag_rand_sample (g05ndc) selects m elements from a population vector **ipop** of length n and places them in a sample vector **isampl**. Their order in **ipop** will be preserved in **isampl**. Each of the $\binom{n}{m}$ possible combinations of elements of **isampl** may be regarded as being equally probable.

For moderate or large values of n it is theoretically impossible that all combinations of size m may occur, unless m is near 1 or near n. This is because $\binom{n}{m}$ exceeds the cycle length of any of the base generators. For practical purposes this is irrelevant, as the time taken to generate all possible combinations is many millenia.

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_sample (g05ndc).

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: **ipop**[**n**] – const Integer *Input*

On entry: the population to be sampled.

2: **n** – Integer *Input*

On entry: the number of elements in the population vector to be sampled.

Constraint: **n** ≥ 1.

3: **isampl**[**m**] – Integer *Output*

On exit: the selected sample.

4: **m** – Integer *Input*

On entry: the sample size.

Constraint: $1 \leq$ **m** $\leq$ **n**.

5: **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.

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, $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{n} \geq 1$.

NE_INT_2

On entry, $\mathbf{m} = \langle value \rangle$ and $\mathbf{n} = \langle value \rangle$.
Constraint: $1 \leq \mathbf{m} \leq \mathbf{n}$.

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.

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_rand_sample (g05ndc) 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

The time taken by nag_rand_sample (g05ndc) is of order n.

In order to sample other kinds of vectors, or matrices of higher dimension, the following technique may be used:

(a) set **ipop**$[i-1] = i$, for $i = 1, 2, \ldots, n$;

(b) use nag_rand_sample (g05ndc) to take a sample from **ipop** and put it into **isampl**;

(c) use the contents of **isampl** as a set of indices to access the relevant vector or matrix.

In order to divide a population into several groups, nag_rand_permute (g05ncc) is more efficient.

10 Example

In the example program random samples of size $1, 2, \ldots, 8$ are selected from a vector containing the first eight positive integers in ascending order. The samples are generated and printed for each sample size by a call to nag_rand_sample (g05ndc) after initialization by nag_rand_init_repeatable (g05kfc).

10.1 Program Text

```
/* nag_rand_sample (g05ndc) 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, m;
  Integer     *ipop = 0, *isampl = 0, *state = 0;

  /* NAG structures */
  NagError    fail;

  /* Population size */
  Integer     n = 8;

  /* 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_sample (g05ndc) 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;
    }

  if (!(ipop = NAG_ALLOC(n, Integer)) ||
      !(isampl = NAG_ALLOC(n, Integer)) ||
      !(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;
    }

  printf("  Samples from the first %1"NAG_IFMT" integers\n", n);
  printf("  Sample size     Values\n");

  /* Initialise the population*/
  for (i = 0; i < n; i++)
    ipop[i] = i + 1;

  /* Generate samples of different sizes*/
  for (m = 1; m <= n; m++)
    {
      nag_rand_sample(ipop, n, isampl, m, state, &fail);
      if (fail.code != NE_NOERROR)
        {
          printf("Error from nag_rand_sample (g05ndc).\n%s\n",
                  fail.message);
          exit_status = 1;
          goto END;
        }

      /* Display the results*/
      printf(" %6"NAG_IFMT"           ", m);
      for (i = 0; i < m; i++)
        printf("%2"NAG_IFMT"%s", isampl[i], (i + 1)%8?" ":"\n");
      if (m%8) printf("\n");
    }

 END:
  NAG_FREE(ipop);
  NAG_FREE(isampl);
  NAG_FREE(state);

  return exit_status;
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_rand_sample (g05ndc) Example Program Results

  Samples from the first 8 integers
  Sample size     Values
      1            2
      2            3  6
```

```
 3          1  5  7
 4          2  6  7  8
 5          1  2  3  4  8
 6          1  3  4  5  6  7
 7          1  3  4  5  6  7  8
 8          1  2  3  4  5  6  7  8
```