NAG Library Function Document

nag_rand_permute (g05ncc)

1 Purpose

nag_rand_permute (g05ncc) performs a pseudorandom permutation of a vector of integers.

2 Specification

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

```
void nag_rand_permute (Integer indx[], Integer n, Integer state[],
     NagError *fail)
```

3 Description

nag_rand_permute (g05ncc) permutes the elements of an integer array without inspecting their values. Each of the $n!$ possible permutations of the n values may be regarded as being equally probable.

Even for modest values of n it is theoretically impossible that all $n!$ permutations may occur, as $n!$ is likely to exceed the cycle length of any of the base generators. For practical purposes this is irrelevant, as the time necessary to generate all possible permutations 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_permute (g05ncc).

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: **indx**[**n**] – Integer *Input/Output*

On entry: the n integer values to be permuted.

On exit: the n permuted integer values.

2: **n** – Integer *Input*

On entry: the number of values to be permuted.

Constraint: $\mathbf{n} \geq 1$.

3: **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.

4: **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** ≥ 1.

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_permute (g05ncc) 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

In the example program a vector containing the first eight positive integers in ascending order is permuted by a call to nag_rand_permute (g05ncc) and the permutation is printed. This is repeated a total of ten times, after initialization by nag_rand_init_repeatable (g05kfc).

10.1 Program Text

```
/* nag_rand_permute (g05ncc) 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, j, lstate;
  Integer     *index = 0, *state = 0;

  /* NAG structures */
  NagError    fail;

  /* Number of permutations*/
  Integer     m = 10;

  /* Sample 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_permute (g05ncc) 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 (!(index = 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(" %2"NAG_IFMT" Permutations of first %1"NAG_IFMT" integers\n",
          m, n);

  /* Permutate M times */
  for (j = 0; j < m; j++)
    {
      /* Set up the index vector */
      for (i = 0; i < n; i++)
        index[i] = i + 1;

      /* Call the permutation routine*/
      nag_rand_permute(index, n, state, &fail);
      if (fail.code != NE_NOERROR)
        {
          printf("Error from nag_rand_permute (g05ncc).\n%s\n",
                  fail.message);
          exit_status = 1;
          goto END;
        }

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

 END:
  NAG_FREE(index);
  NAG_FREE(state);

  return exit_status;
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_rand_permute (g05ncc) Example Program Results

 10 Permutations of first 8 integers
   6  2  4  8  1  3  5  7
   8  6  4  2  7  3  1  5
   4  2  8  7  5  6  3  1
   1  6  4  5  2  3  7  8
   1  7  3  8  4  2  5  6
   6  3  4  7  1  2  8  5
   6  4  1  8  2  5  3  7
   3  2  1  7  5  8  6  4
   4  2  1  5  3  6  8  7
   1  5  6  4  2  7  8  3
```