NAG Library Function Document

nag_make_indices (m01zac)

1 Purpose

nag_make_indices (m01zac) inverts a permutation, and hence converts a rank vector to an index vector, or vice versa.

2 Specification

```
#include <nag.h>
#include <nagm01.h>
void nag_make_indices (size_t ranks[], size_t n, NagError *fail)
```

3 Description

There are two common ways of describing a permutation using an Integer vector **ranks**. The first uses ranks: **ranks**$[i]$ holds the index value to which the $(i+1)$th data element should be moved in order to sort the data; in other words its rank in the sorted order. The second uses indices: **ranks**$[i]$ holds the current index value of the data element which would occur in $(i+1)$th position in sorted order. For example, given the values

$$3.5 \quad 5.9 \quad 2.9 \quad 0.5$$

to be sorted in ascending order, the ranks would be

$$2 \quad 3 \quad 1 \quad 0$$

and the indices would be

$$3 \quad 2 \quad 0 \quad 1.$$

The m01d- functions generate ranks, and the m01e- functions require indices to be supplied to specify the re-ordering. However if it is desired simply to refer to the data in sorted order without actually re-ordering them, indices are more convenient than ranks (see Section 10). nag_make_indices (m01zac) can be used to convert ranks to indices, or indices to ranks, as the two permutations are inverses of one another.

4 References

None.

5 Arguments

1: **ranks**[**n**] – size_t *Input/Output*

On entry: **ranks** must contain a permutation of the Integers 0 to **n** − 1.

On exit: **ranks** contains the inverse permutation.

2: **n** – size_t *Input*

On entry: the length of the array **ranks**.

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

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_BAD_RANK

Invalid **ranks** vector.
Elements of **ranks** contain a value outside the range 0 to $\mathbf{n}-1$ or contain a repeated value. **ranks** does not contain a permutation of the Integers 0 to $\mathbf{n}-1$; on exit these elements are usually corrupted.

NE_INT_ARG_GT

On entry, $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{n} \leq \langle value \rangle$.
n is limited to an implementation-dependent size which is printed in the error message.

NE_INT_ARG_LT

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

7 Accuracy

Not applicable.

8 Parallelism and Performance

Not applicable.

9 Further Comments

None.

10 Example

The example program reads a matrix of real numbers and prints its rows with the elements of the 1st column in ascending order as ranked by nag_rank_sort (m01dsc). The program first calls nag_rank_sort (m01dsc) to rank the rows, and then calls nag_make_indices (m01zac) to convert the rank vector to an index vector, which is used to refer to the rows in sorted order.

10.1 Program Text

```
/* nag_make_indices (m01zac) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 2 revised, 1992.
 * Mark 5 revised, 1998.
 * Mark 7 revised, 2001.
 * Mark 8 revised, 2004.
 *
 */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nag_stddef.h>
#include <nagm01.h>


#ifdef __cplusplus
extern "C" {
#endif
static Integer NAG_CALL compare(const Nag_Pointer a, const Nag_Pointer b);
```

```
#ifdef __cplusplus
}
#endif

#define VEC(I, J) vec[(I) *tdvec + J]
int main(void)
{
  Integer  exit_status = 0, tdvec;
  NagError fail;
  double   *vec = 0;
  size_t   i, j, m, n, *rank = 0;

  INIT_FAIL(fail);

  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
  printf("nag_make_indices (m01zac) Example Program Results\n");
#ifdef _WIN32
  scanf_s("%" NAG_UFMT "%" NAG_UFMT "", &m, &n);
#else
  scanf("%" NAG_UFMT "%" NAG_UFMT "", &m, &n);
#endif
  if (m >= 1 && n >= 1)
    {
      if (!(vec = NAG_ALLOC(m*n, double)) ||
          !(rank = NAG_ALLOC(m, size_t)))
        {
          printf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }
      tdvec = n;
    }
  else
    {
      printf("Invalid m or n.\n");
      exit_status = 1;
      return exit_status;
    }
  for (i = 0; i < m; ++i)
    for (j = 0; j < n; ++j)
#ifdef _WIN32
      scanf_s("%lf", &VEC(i, j));
#else
      scanf("%lf", &VEC(i, j));
#endif
  /* nag_rank_sort (m01dsc).
   * Rank sort of set of values of arbitrary data type
   */
  nag_rank_sort((Pointer) vec, m, (ptrdiff_t)(n*sizeof(double)), compare,
                Nag_Ascending, rank, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_rank_sort (m01dsc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* nag_make_indices (m01zac).
   * Inverts a permutation converting a rank vector to an
   * index vector or vice versa
   */
  nag_make_indices(rank, m, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_make_indices (m01zac).\n%s\n",
              fail.message);
      exit_status = 1;
```

```
      goto END;
    }
  printf("Matrix with rows sorted according to column 1\n");
  for (i = 0; i < m; ++i)
    {
      for (j = 0; j < n; ++j)
        printf(" %7.1f ", VEC(rank[i], j));
      printf("\n");
    }
 END:
  NAG_FREE(vec);
  NAG_FREE(rank);
  return exit_status;
}

static Integer NAG_CALL compare(const Nag_Pointer a, const Nag_Pointer b)
{
  double x = *((const double *) a);
  double y = *((const double *) b);
  return(x < y?-1:(x == y?0:1));
}
```

10.2 Program Data

```
nag_make_indices (m01zac) Example Program Data
12 3
6.0 5.0 4.0
5.0 2.0 1.0
2.0 4.0 9.0
4.0 9.0 6.0
4.0 9.0 5.0
4.0 1.0 2.0
3.0 4.0 1.0
2.0 4.0 6.0
1.0 6.0 4.0
9.0 3.0 2.0
6.0 2.0 5.0
4.0 9.0 6.0
```

10.3 Program Results

```
nag_make_indices (m01zac) Example Program Results
Matrix with rows sorted according to column 1
     1.0      6.0      4.0
     2.0      4.0      9.0
     2.0      4.0      6.0
     3.0      4.0      1.0
     4.0      9.0      6.0
     4.0      9.0      5.0
     4.0      1.0      2.0
     4.0      9.0      6.0
     5.0      2.0      1.0
     6.0      5.0      4.0
     6.0      2.0      5.0
     9.0      3.0      2.0
```