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 <nag_stddef.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 5.9 2.9 0.5

to be sorted in ascending order, the ranks would be

2 3 1 0

and the indices would be

3 2 0 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 the example program).
nag_make_indices can be used to convert ranks to indices, or indices to ranks, as the two permutations are inverses of one another.

4. Parameters

ranks[**n**]
Input: **ranks** must contain a permutation of the Integers 0 to **n** − 1.
Output: **ranks** contains the inverse permutation.

n
Input: the length of the array **ranks**.

fail
The NAG error parameter, see the Essential Introduction to the NAG C Library.

5. Error Indications and Warnings

NE_INT_ARG_LT
On entry, **n** must not be less than 0: **n** = ⟨*value*⟩.

NE_INT_ARG_GT
On entry, **n** must not be greater than ⟨*value*⟩: **n** = ⟨*value*⟩.
n is limited to an implementation-dependent size which is printed in the error message.

NE_BAD_RANK
Invalid **ranks** vector.
Elements of **ranks** contain a value outside the range 0 to **n** − 1 or contain a repeated value. **ranks** does not contain a permutation of the Integers 0 to **n** − 1; on exit these elements are usually corrupted.

6. Further Comments

None.

7. See Also

None.

8. 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 to convert the rank vector to an index vector, which is used to refer to the rows in sorted order.

8.1. Program Text

```
/* nag_make_indices(m01zac) Example Program
 *
 * Copyright 1990 Numerical Algorithms Group.
 *
 * Mark 2 revised, 1992.
 */

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

#ifdef NAG_PROTO
static Integer compare(const Pointer a,const Pointer b)
#else
     static Integer compare(a,b)
     Pointer a, b;
#endif
{
  double x = *((double *)a);
  double y = *((double *)b);
  return (x<y ? -1 : (x==y ? 0 : 1));
}

#define MMAX 20
#define NMAX 20

main()
{
  double vec[MMAX][NMAX];
  size_t i, j, m, n, rank[MMAX];
  static NagError fail;

  fail.print = TRUE;
  /* Skip heading in data file */
  Vscanf("%*[^\n]");
  Vprintf("m01zac Example Program Results\n");
  Vscanf("%d%d", &m, &n);
  if (m>=0 && m<=MMAX && n>=0 && n<=NMAX)
    {
      for (i=0; i<m; ++i)
        for (j=0; j<n; ++j)
          Vscanf("%lf", &vec[i][j]);
      m01dsc((Pointer) vec, m, (ptrdiff_t)(NMAX*sizeof(double)), compare,
             Nag_Ascending, rank, &fail);
      if (fail.code != NE_NOERROR)
        exit(EXIT_FAILURE);
      m01zac(rank, m, &fail);
      if (fail.code != NE_NOERROR)
        exit(EXIT_FAILURE);
      Vprintf("Matrix with rows sorted according to column 1\n");
```

```
        for (i=0; i<m; ++i)
          {
            for (j=0; j<n; ++j)
              Vprintf(" %7.1f ", vec[rank[i]][j]);
            Vprintf("\n");
          }
        exit(EXIT_SUCCESS);
      }
    else
      {
        Vfprintf(stderr, "Data error: program terminated\n");
        exit(EXIT_FAILURE);
      }
}
```

8.2. Program Data

```
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
```

8.3. Program Results

```
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
```