

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 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 Section 9). 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 | <i>Input/Output</i> |
| | <i>On entry:</i> ranks must contain a permutation of the Integers 0 to n – 1. | |
| | <i>On exit:</i> ranks contains the inverse permutation. | |
| 2: | n – size_t | <i>Input</i> |
| | <i>On entry:</i> the length of the array ranks . | |
| 3: | fail – NagError * | <i>Input/Output</i> |
| | 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 $n - 1$ 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.

NE_INT_ARG_GT

On entry, $n = \langle value \rangle$.

Constraint: $n \leq \langle value \rangle$.

n is limited to an implementation-dependent size which is printed in the error message.

NE_INT_ARG_LT

On entry, $n = \langle value \rangle$.

Constraint: $n \geq 0$.

7 Accuracy

Not applicable.

8 Further Comments

None.

9 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.

9.1 Program Text

```

/* nag_make_indices (m01zac) Example Program.
 *
 * Copyright 1990 Numerical Algorithms Group.
 *
 * Mark 2 revised, 1992.
 * Mark 5 revised, 1998.
 * Mark 7 revised, 2001.
 * Mark 8 revised, 2004.
 *
 */

#include <nag.h>
#include <nagx04.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)
{
    double x = *((const double *) a);
    double y = *((const double *) b);

```

```

    return(x < y?-1:(x == y?0:1));
}
#ifdef __cplusplus
}
#endif

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

    INIT_FAIL(fail);

    /* Check for command-line IO options */
    fpin = nag_example_file_io(argc, argv, "-data", NULL);
    fpout = nag_example_file_io(argc, argv, "-results", NULL);
    /* Skip heading in data file */
    fscanf(fpin, "%*[\n]");
    fprintf(fpout, "nag_make_indices (m01zac) Example Program Results\n");
    fscanf(fpin, "%" NAG_UFMT "%" NAG_UFMT "", &m, &n);
    if (m >= 1 && n >= 1 && n <= m)
    {
        if (!(vec = NAG_ALLOC(m*n, double)) ||
            !(rank = NAG_ALLOC(m, size_t)))
        {
            fprintf(fpout, "Allocation failure\n");
            exit_status = -1;
            goto END;
        }
        tdvec = n;
    }
    else
    {
        fprintf(fpout, "Invalid m or n.\n");
        exit_status = 1;
        return exit_status;
    }
    for (i = 0; i < m; ++i)
        for (j = 0; j < n; ++j)
            fscanf(fpin, "%lf", &VEC(i, j));
    /* 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)
    {
        fprintf(fpout, "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)
    {
        fprintf(fpout, "Error from nag_make_indices (m01zac).\n%s\n",
                fail.message);
        exit_status = 1;
        goto END;
    }
    fprintf(fpout, "Matrix with rows sorted according to column 1\n");
    for (i = 0; i < m; ++i)
    {
        for (j = 0; j < n; ++j)

```

```
        fprintf(fpout, " %7.1f ", VEC(rank[i], j));
        fprintf(fpout, "\n");
    }
END:
    if (fpin != stdin) fclose(fpin);
    if (fpout != stdout) fclose(fpout);
    if (vec) NAG_FREE(vec);
    if (rank) NAG_FREE(rank);
    return exit_status;
}
```

9.2 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
```

9.3 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
```
