NAG Toolbox

nag_sort_intmat_rank_columns (m01dk)

1 Purpose

nag_sort_intmat_rank_columns (m01dk) ranks the columns of a matrix of integer numbers in ascending or descending order.

2 Syntax

```
[irank, ifail] = nag_sort_intmat_rank_columns(im, m1, n1, order, 'm2', m2, 'n2', n2)
[irank, ifail] = m01dk(im, m1, n1, order, 'm2', m2, 'n2', n2)
```

Note: the interface to this routine has changed since earlier releases of the toolbox:

At Mark 22: **m2** was made optional.

3 Description

nag_sort_intmat_rank_columns (m01dk) ranks columns **n1** to **n2** of a matrix, using the data in rows **m1** to **m2** of those columns. The ordering is determined by first ranking the data in row **m1**, then ranking any tied columns according to the data in row **m1** + 1, and so on up to row **m2**.

nag_sort_intmat_rank_columns (m01dk) uses a variant of list-merging, as described on pages 165–166 in Knuth (1973). The function takes advantage of natural ordering in the data, and uses a simple list insertion in a preparatory pass to generate ordered lists of length at least 10. The ranking is stable: equal columns preserve their ordering in the input data.

4 References

Knuth D E (1973) *The Art of Computer Programming (Volume 3)* (2nd Edition) Addison–Wesley

5 Parameters

5.1 Compulsory Input Parameters

1: **im**(ldm, **n2**) – INTEGER array

ldm, the first dimension of the array, must satisfy the constraint $ldm \geq$ **m2**.

Rows **m1** to **m2** of columns **n1** to **n2** of **im** must contain integer data to be ranked.

2: **m1** – INTEGER

The index of the first row of **im** to be used.

Constraint: **m1** > 0.

3: **n1** – INTEGER

The index of the first column of **im** to be ranked.

Constraint: **n1** > 0.

4: **order** – CHARACTER(1)

If **order** = 'A', the columns will be ranked in ascending (i.e., nondecreasing) order.

If **order** = 'D', into descending order.

Constraint: **order** = 'A' or 'D'.

5.2 Optional Input Parameters

1: **m2** – INTEGER

Default: the first dimension of the array **im**.

The index of the last row of **im** to be used.

Constraint: **m2** $\geq$ **m1**.

2: **n2** – INTEGER

Default: the second dimension of the array **im**.

The index of the last column of **im** to be ranked.

Constraint: **n2** $\geq$ **n1**.

5.3 Output Parameters

1: **irank**(**n2**) – INTEGER array

Elements **n1** to **n2** of **irank** contain the ranks of the corresponding columns of **im**. Note that the ranks are in the range **n1** to **n2**: thus, if the ith column of **im** is the first in the rank order, **irank**(i) is set to **n1**.

2: **ifail** – INTEGER

ifail = 0 unless the function detects an error (see Section 5).

6 Error Indicators and Warnings

Errors or warnings detected by the function:

ifail = 1

On entry, **m2** $<$ 1,
or **n2** $<$ 1,
or **m1** $<$ 1,
or **m1** $>$ **m2**,
or **n1** $<$ 1,
or **n1** $>$ **n2**,
or $ldm <$ **m2**.

ifail = 2

On entry, **order** is not `A' or `D'.

ifail = −99

An unexpected error has been triggered by this routine. Please contact NAG.

ifail = −399

Your licence key may have expired or may not have been installed correctly.

ifail = −999

Dynamic memory allocation failed.

7 Accuracy

Not applicable.

8 Further Comments

The average time taken by the function is approximately proportional to $n \times \log(n)$, where $n = \mathbf{n2} - \mathbf{n1} + 1$.

9 Example

This example reads a matrix of integers and ranks the columns in descending order.

9.1 Program Text

```
     function m01dk_example

fprintf('m01dk example results\n\n');

im = [nag_int(5),4,3,2,2,1,9,4,4,2,2,1;
               3,8,2,5,5,6,9,8,9,5,4,1;
               9,1,6,1,2,4,8,1,2,2,6,2];

m1 = nag_int(1);
n1 = nag_int(1);
order = 'Descending';

[irank, ifail] = m01dk( ...
                        im, m1, n1, order);

fprintf('Data\n');
for i = 1:size(im,1)
  fprintf('%5d',im(i,:));
  fprintf('\n');
end
fprintf('\nRanks\n');
fprintf('%5d',irank);
fprintf('\n');
```

9.2 Program Results

```
     m01dk example results

Data
    5    4    3    2    2    1    9    4    4    2    2    1
    3    8    2    5    5    6    9    8    9    5    4    1
    9    1    6    1    2    4    8    1    2    2    6    2

Ranks
    2    4    6    9    7   11    1    5    3    8   10   12
```