NAG Fortran Library Chapter Introduction

M01 – Sorting

Contents

1 Scope of the Chapter

This chapter is concerned with sorting numeric or character data. It handles only the simplest types of data structure and it is concerned only with **internal** sorting – that is, sorting a set of data which can all be stored within the program.

Users with large files of data or complicated data structures to be sorted should use a comprehensive sorting program or package.

2 Background to the Problems

The usefulness of sorting is obvious (perhaps a little too obvious, since sorting can be expensive and is sometimes done when not strictly necessary). Sorting may traditionally be associated with data processing and non-numerical programming, but it has many uses within the realm of numerical analysis. For example, within the NAG Fortran Library, sorting is used to arrange eigenvalues in ascending order of absolute value, in the manipulation of sparse matrices, and in the ranking of observations for nonparametric statistics.

The general problem may be defined as follows. We are given N items of data

$$R_1, R_2, \ldots, R_N.$$

Each item R_i contains a key K_i which can be ordered relative to any other key according to some specified criterion (for example, ascending numeric value). The problem is to determine a permutation

$$p(1), p(2), \ldots, p(N)$$

which puts the keys in order:

$$K_{p(1)} \leq K_{p(2)} \leq \ldots \leq K_{p(N)}.$$

Sometimes we may wish actually to **rearrange** the items so that their keys are in order; for other purposes we may simply require a table of **indices** so that the items can be referred to in sorted order; or yet again we may require a table of **ranks**, that is, the positions of each item in the sorted order.

For example, given the single-character items, to be sorted into alphabetic order

E B A D C

the indices of the items in sorted order are

3 2 5 4 1

and the ranks of the items are

5 2 1 4 3.

Indices may be converted to ranks, and vice versa, by simply computing the inverse permutation.

The items may consist solely of the key (each item may simply be a number). On the other hand, the items may contain additional information (for example, each item may be an eigenvalue of a matrix and its associated eigenvector, the eigenvalue being the key). In the latter case there may be many distinct items with equal keys, and it may be important to preserve the original order among them (if this is achieved, the sorting is called '**stable**').

There are a number of ingenious algorithms for sorting. For a fascinating discussion of them, and of the whole subject, see Knuth (1973).

3 Recommendations on Choice and Use of Available Routines

Note: refer to the Users' Note for your implementation to check that a routine is available.

Four categories of routines are provided:

– routines which rearrange the data into sorted order (M01C-);

– routines which determine the ranks of the data, leaving the data unchanged (M01D);

– routines which rearrange the data according to pre-determined ranks (M01E);

– service routines (M01Z).

In the first two categories, routines are provided for ***real*** and integer numeric data, and for character data. In the third category there are routines for rearranging ***real***, ***complex***, integer and character data. Utilities for the manipulation of sparse matrices can be found in Chapter F11.

If the task is simply to rearrange a one-dimensional array of data into sorted order, then an M01C- routine should be used, since this requires no extra workspace and is faster than any other method. There are no M01C- routines for more complicated data structures, because the cost of rearranging the data is likely to outstrip the cost of comparisons. Instead, a combination of M01D and M01E routines, or some other approach, must be used as described below.

For many applications it is in fact preferable to separate the task of determining the sorted order (ranking) from the task of rearranging data into a pre-determined order; the latter task may not need to be performed at all. Frequently it may be sufficient to refer to the data in sorted order via an index vector, without rearranging it. Frequently also one set of data (e.g., a column of a matrix) is used for determining a set of ranks, which are then applied to other data (e.g., the remaining columns of the matrix).

To determine the ranks of a set of data, use an M01D routine. Routines are provided for ranking one-dimensional arrays, and for ranking rows or columns of two-dimensional arrays. For ranking an arbitrary data structure, use M01DZF, which is, however, much less efficient than the other M01D routines.

To create an index vector so that data can be referred to in sorted order, first call an M01D routine to determine the ranks, and then call M01ZAF to convert the vector of ranks into an index vector.

To rearrange data according to pre-determined ranks: use an M01E routine if the data is stored in a one-dimensional array; or if the data is stored in a more complicated structure

either use an index vector to generate a new copy of the data in the desired order

or rearrange the data without using extra storage by first calling M01ZCF and then using the simple code-framework given in the document for M01ZCF (assuming that the elements of data all occupy equal storage).

Examples of these operations can be found in the routine documents of the relevant routines.

4 Index

5 Routines Withdrawn or Scheduled for Withdrawal

The following routines have been withdrawn. Advice on replacing calls to those withdrawn since Mark 13 is given in the document 'Advice on Replacement Calls for Withdrawn/Superseded Routines'.

Withdrawn Routine	Mark of Withdrawal	Replacement Routine(s)
M01AAF	13	M01DAF
M01ABF	13	M01DAF
M01ACF	13	M01DBF
M01ADF	13	M01DBF
M01AEF	13	M01DEF and M01EAF
M01AFF	13	M01DEF and M01EAF
M01AGF	13	M01DFF and M01EBF
M01AHF	13	M01DFF and M01EBF
M01AJF	16	M01DAF, M01ZAF and M01CAF
M01AKF	16	M01DAF, M01ZAF and M01CAF
M01ALF	13	M01DBF, M01ZAF and M01CBF
M01AMF	13	M01DBF, M01ZAF and M01CBF
M01ANF	13	M01CAF
M01APF	16	M01CAF
M01AQF	13	M01CBF
M01ARF	13	M01CBF
M01BAF	13	M01CCF
M01BBF	13	M01CCF
M01BCF	13	M01CCF
M01BDF	13	M01CCF

6 References

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