NAG Toolbox for MATLAB

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.

If you have large files of data or complicated data structures to be sorted you 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 Functions

Four categories of functions are provided:

- functions which rearrange the data into sorted order (M01C-);
- functions which determine the ranks of the data, leaving the data unchanged (M01D);
- functions which rearrange the data according to pre-determined ranks (M01E);
- service functions (M01Z).

In the first two categories, functions are provided for double and integer numeric data, and for character data. In the third category there are functions for rearranging double, ***complex*16***, 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- function should be used, since this requires no extra workspace and is faster than any other method. There are no M01C- functions 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 functions, 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 function. 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 m01dz, which is, however, much less efficient than the other M01D functions.

To create an index vector so that data can be referred to in sorted order, first call an M01D function to determine the ranks, and then call m01za to convert the vector of ranks into an index vector.

To rearrange data according to pre-determined ranks: use an M01E function 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 m01zc and then using the simple code-framework given in the document for m01zc (assuming that the elements of data all occupy equal storage).

Examples of these operations can be found in the function documents of the relevant functions.

4 Index

5 References

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