sort( ) — Reorder rows of matrix

Description Syntax Remarks and examples Conformability Diagnostics Also see

Description

`sort(`*X*`,` *idx*`)` returns *X* with rows in ascending or descending order of the columns specified by *idx*. For instance, `sort(`*X*`, 1)` sorts *X* on its first column; `sort(`*X*`, (1,2))` sorts *X* on its first and second columns (meaning rows with equal values in their first column are ordered on their second column). In general, the *i*th sort key is column `abs(`*idx*`[`*i*`])`. Order is ascending if *idx*`[`*i*`]` > 0 and descending otherwise. Ascending and descending are defined in terms of [M-5] **abs( )** (length of elements) for complex.

`_sort(`*X*`,` *idx*`)` does the same as `sort(`*X*`,` *idx*`)`, except that *X* is sorted in place.

`jumble(`*X*`)` returns *X* with rows in random order. For instance, to shuffle a deck of cards numbered 1 to 52, one could code `jumble(1::52)`. See `rseed()` in [M-5] **runiform( )** for information on setting the random-number seed.

`_jumble(`*X*`)` does the same as `jumble(`*X*`)`, except that *X* is jumbled in place.

`order(`*X*`,` *idx*`)` returns the permutation vector—see [M-1] **Permutation**—that would put *X* in ascending (descending) order of the columns specified by *idx*. A row-permutation vector is a $1 \times c$ column vector containing the integers 1, 2, ..., *c* in some order. Vectors `(1\2\3)`, `(1\3\2)`, `(2\1\3)`, `(2\3\1)`, `(3\1\2)`, and `(3\2\1)` are examples. Row-permutation vectors are used to specify the order in which the rows of a matrix *X* are to appear. If *p* is a row-permutation vector, *X*`[`*p*`, .]` returns *X* with its rows in the order of *p*; *p* = `(3\2\1)` would reverse the rows of *X*. `order(`*X*`,` *idx*`)` returns the row-permutation vector that would sort *X* and, as a matter of fact, `sort(`*X*`,` *idx*`)` is implemented as *X*`[order(`*X*`,` *idx*`), .]`.

`unorder(`*n*`)` returns a $1 \times n$ permutation vector for placing the rows in random order. Random numbers are calculated by `runiform()`; see `rseed()` in [M-5] **runiform( )** for information on setting the random-number seed. `jumble()` is implemented in terms of `unorder()`: `jumble(`*X*`)` is equivalent to *X*`[unorder(rows(`*X*`)), .]`.

`_collate(`*X*`,` *p*`)` is equivalent to *X* = *X*`[`*p*`, .]`; it changes the order of the rows of *X*. `_collate()` is used by `_sort()` and `_jumble()` and has the advantage over subscripting in that no extra memory is required when the result is to be assigned back to itself. Consider

X = *X*`[`*p*`, .]`

There will be an instant after *X*`[`*p*`, .]` has been calculated but before the result has been assigned back to *X* when two copies of *X* exist. `_collate(`*X*`,` *p*`)` avoids that. `_collate()` is not a substitute for subscripting in all cases; `_collate()` requires *p* be a permutation vector.

Syntax

transmorphic matrix	`sort(`*transmorphic matrix X*`,` *real rowvector idx*`)`
void	`_sort(`*transmorphic matrix X*`,` *real rowvector idx*`)`
transmorphic matrix	`jumble(`*transmorphic matrix X*`)`
void	`_jumble(`*transmorphic matrix X*`)`
real colvector	`order(`*transmorphic matrix X*`,` *real rowvector idx*`)`
real colvector	`unorder(`*real scalar n*`)`
void	`_collate(`*transmorphic matrix X*`,` *real colvector p*`)`

where

1. X may not be a pointer matrix.
2. p must be a permutation column vector, a $c \times 1$ vector containing the integers 1, 2, …, c in some order.

Remarks and examples

If X is complex, the ordering is defined in terms of [M-5] **abs( )** of its elements.

Also see `invorder()` and `revorder()` in [M-5] **invorder( )**. Let p be the permutation vector returned by `order()`:

```
p = order(X, ...)
```

Then X`[`p`,.]` are the sorted rows of X. `revorder()` can be used to reverse sort order: X`[revorder(`p`),.]` are the rows of X in the reverse of the order of X`[`p`,.]`. `invorder()` provides the inverse transform: If $Y = X$`[`p`,.]`, then $X = Y$`[invorder(`p`),.]`.

Conformability

`sort(`X`,` *idx*`)`, `jumble(`X`)`:

X:	$r_1 \times c_1$
idx:	$1 \times c_2, c_2 \leq c_1$
result:	$r_1 \times c_1$

`_sort(`X`,` *idx*`)`, `_jumble(`X`)`:

X:	$r_1 \times c_1$
idx:	$1 \times c_2, c_2 \leq c_1$
result:	*void*; X row order modified

`order(`X`,` *idx*`)`:

X:	$r_1 \times c_1$
idx:	$1 \times c_2, c_2 \leq c_1$
result:	$r_1 \times 1$

`unorder(`*n*`)`:

n:	1×1	
result:	$n \times 1$	

`_collate(`*X*`, `*p*`)`:

X:	$r \times c$	
p:	$r \times 1$	
result:	*void*;	*X* row order modified

Diagnostics

`sort(`*X*`, `*idx*`)` aborts with error if any element of `abs(`*idx*`)` is less than 1 or greater than `rows(`*X*`)`.

`_sort(`*X*`, `*idx*`)` aborts with error if any element of `abs(`*idx*`)` is less than 1 or greater than `rows(`*X*`)`, or if *X* is a view.

`_jumble(`*X*`)` aborts with error if *X* is a view.

`order(`*X*`, `*idx*`)` aborts with error if any element of `abs(`*idx*`)` is less than 1 or greater than `rows(`*X*`)`.

`unorder(`*n*`)` aborts with error if $n < 1$.

`_collate(`*X*`, `*p*`)` aborts with error if *p* is not a permutation vector or if *X* is a view.

Also see

[M-5] **invorder( )** — Permutation vector manipulation

[M-5] **uniqrows( )** — Obtain sorted, unique values

[M-5] **ustrcompare( )** — Compare or sort Unicode strings

[M-4] **Manipulation** — Matrix manipulation