Title

void — Void matrices

Syntax Description Remarks and examples Also see

Syntax

`J(0, 0, .)`	0×0 real matrix
`J(`r`, 0, .)`	$r \times 0$ real matrix
`J(0, `c`, .)`	$0 \times c$ real matrix
`J(0, 0, 1i)`	0×0 complex matrix
`J(`r`, 0, 1i)`	$r \times 0$ complex matrix
`J(0, `c`, 1i)`	$0 \times c$ complex matrix
`J(0, 0, "")`	0×0 string matrix
`J(`r`, 0, "")`	$r \times 0$ string matrix
`J(0, `c`, "")`	$0 \times c$ string matrix
`J(0, 0, NULL)`	0×0 pointer matrix
`J(`r`, 0, NULL)`	$r \times 0$ pointer matrix
`J(0, `c`, NULL)`	$0 \times c$ pointer matrix

Description

Mata allows 0×0, $r \times 0$, and $0 \times c$ matrices. These matrices are called *void matrices*.

Remarks and examples

Remarks are presented under the following headings:

Void matrices, vectors, row vectors, and column vectors

Void matrices contain nothing, but they have dimension information (they are 0×0, $r \times 0$, or $0 \times c$) and have an *eltype* (which is `real`, `complex`, `string`, or `pointer`):

1. A matrix is said to be void if it is 0×0, $r \times 0$, or $0 \times c$.

2. A vector is said to be void if it is 0×1 or 1×0.

3. A column vector is said to be void if it is 0×1.

4. A row vector is said to be void if it is 1×0.

5. A scalar cannot be void because it is, by definition, 1×1.

The function `J(`*r*`,` *c*`,` *val*`)` creates $r \times c$ matrices containing *val*; see [M-5] **J()**. `J()` can be used to manufacture void matrices by specifying *r* and/or *c* as 0. The value of the third argument does not matter, but its *eltype* does:

1. `J(0,0,.)` creates a real 0×0 matrix, as will `J(0,0,1)` and as will `J()` with any real third argument.
2. `J(0,0,1i)` creates a 0×0 complex matrix, as will `J()` with any complex third argument.
3. `J(0,0,"")` creates 0×0 string matrices, as will `J()` with any string third argument.
4. `J(0,0,NULL)` creates 0×0 pointer matrices, as will `J()` with any pointer third argument.

In fact, one rarely needs to manufacture such matrices because they arise naturally in extreme cases. Similarly, one rarely needs to include special code to handle void matrices because such matrices handle themselves. Loops vanish when the number of rows or columns are zero.

How to read conformability charts

In general, not only is no emphasis placed on how functions and operators deal with void matrices, no mention is even made of the fact. Instead, the information is buried in the *Conformability* section located near the end of the function's or operator's manual entry.

For instance, the conformability chart for some function might read

`somefunction(`*A*`,` *B*`,` *v*`)`:

A:	$r \times c$		
B:	$c \times k$		
v:	$1 \times k$	or	$k \times 1$
result:	$r \times k$		

Among other things, the chart above is stating how `somefunction()` handles void matrices. *A* must be $r \times c$. That chart does not say

A: $r \times c$, $r > 0$, $c > 0$

and that is what it would have said if `somefunction()` did not allow *A* to be void. Hence, *A* may be 0×0, $0 \times c$, or $r \times 0$.

Similarly, *B* may be void as long as `rows(`*B*`)==cols(`*A*`)`. *v* may be void if `cols(`*B*`)==0`. The returned result will be void if `rows(`*A*`)==0` or `cols(`*B*`)==0`.

Interestingly, `somefunction()` can produce a nonvoid result from void input. For instance, if *A* were 5×0 and *B*, 0×3, a 5×3 result would be produced. It is interesting to speculate what would be in that 5×3 result. Probably, if we knew what `somefunction()` did, it would be obvious to us, but if it were not, the following section, *Diagnostics*, would state what the surprising result would be.

As a real example, see [M-5] **trace()**. `trace()` will take the trace of a 0×0 matrix. The result is 0. Or see multiplication (`*`) in [M-2] **op_arith**. One can multiply a $k \times 0$ matrix by a $0 \times m$ matrix to produce a $k \times m$ result. The matrix will contain zeros.

Also see

[M-2] **intro** — Language definition