Title

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
<th>Remarks and examples</th>
<th>Conformability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**op_join — Row- and column-join operators**

### Syntax

\[ a , b \]
\[ a \ \backslash \ b \]

### Description

, and \ are Mata’s row-join and column-join operators.

### Remarks and examples

Remarks are presented under the following headings:

- Comma and backslash are operators
- Comma as a separator
- Warning about the misuse of comma and backslash operators

#### Comma and backslash are operators

That , and \ are operators cannot be emphasized enough. When one types

\[ : (1, 2 \ \backslash \ 3, 4) \]
\[ \begin{array}{c} 1 \ 2 \\ 2 \ 3 \ 4 \end{array} \]

one is tempted to think, “Ah, comma and backslash are how you separate elements when you enter a matrix.” If you think like that, you will not appreciate the power of , and \.

, and \ are operators in the same way that * and + are operators.

, is the operator that takes a \( r \times c_1 \) matrix and a \( r \times c_2 \) matrix, and returns a \( r \times (c_1 + c_2) \) matrix.

\ is the operator that takes a \( r_1 \times c \) matrix and a \( r_2 \times c \) matrix, and returns a \( (r_1 + r_2) \times c \) matrix.

, and \ may be used with scalars, vectors, or matrices:

\[ : a = (1 \ \backslash \ 2) \]
\[ : b = (3 \ \backslash \ 4) \]
: a, b
  
1  2
1  3  4

: c = (1, 2)
: d = (3, 4)
: c \ d
  
1  2
1  2  3  4

, binds more tightly than \, meaning that \( e, f \setminus g, h \) is interpreted as \( (e, f) \setminus (g, h) \). In this, , and \ are no different from \* and + operators: \* binds more tightly than + and \( e*f + g*h \) is interpreted as \( (e*f) + (g*h) \).

Just as it sometimes makes sense to type \( e*(f+g)*h \), it can make sense to type \( e,(f \setminus g),h \):

: e = 1 \ 2
: f = 5 \ 6
: g = 3
: h = 4
: e,(g\h),f
  
1  2  3
1  2  3  5
2  4  6

Comma as a separator

, has a second meaning in Mata: it is the argument separator for functions. When you type

: myfunc(a, b)

the comma that appears inside the parentheses is not the comma row-join operator; it is the comma argument separator. If you wanted to call \texttt{myfunc()} with second argument equal to row vector \((1,2)\), you must type

: myfunc(a, (1,2))

and not

: myfunc(a, 1, 2)

because otherwise Mata will think you are trying to pass three arguments to \texttt{myfunc()}. When you open another set of parentheses inside a function’s argument list, comma reverts to its usual row-join meaning.
Warning about the misuse of comma and backslash operators

Misuse or mere overuse of , and \ can substantially reduce the speed with which your code executes. Consider the actions Mata must take when you code, say,

\[ a \ \backslash \ b \]

First, Mata must allocate a matrix or vector containing \( \text{rows}(a)+\text{rows}(b) \) rows, then it must copy \( a \) into the new matrix or vector, and then it must copy \( b \). Nothing inefficient has happened yet, but now consider

\[ (a \ \backslash \ b) \ \backslash \ c \]

Picking up where we left off, Mata must allocate a matrix or vector containing \( \text{rows}(a)+\text{rows}(b)+\text{rows}(c) \) rows, then it must copy \( (a \ \backslash \ b) \) into the new matrix or vector, and then it must copy \( c \). Something inefficient just happened: \( a \) was copied twice!

Coding

\[
\text{res} = (a \ \backslash \ b) \ \backslash \ c
\]

is convenient, but execution would be quicker if we coded

\[
\text{res} = J(\text{rows}(a)+\text{rows}(b)+\text{rows}(c), \text{cols}(a), .)
\]
\[
\text{res}[1,\cdot] = a
\]
\[
\text{res}[2,\cdot] = b
\]
\[
\text{res}[3,\cdot] = c
\]

We do not want to cause you concern where none is due. In general, you would not be able to measure the difference between the more efficient code and coding \( \text{res} = (a \ \backslash \ b) \ \backslash \ c \). But as the number of row or column operators stack up, the combined result becomes more and more inefficient. Even that is not much of a concern. If the inefficient construction itself is buried in a loop, however, and that loop is executed thousands of times, the inefficiency can become important.

With a little thought, you can always substitute predeclaration using \( J() \) (see \([\text{M-5}] J()) and assignment via subscripting.

Conformability

\( a,b: \)

\[
a: \quad r \times c_1
\]
\[
b: \quad r \times c_2
\]
\[
\text{result:} \quad r \times (c_1+c_2)
\]

\( a \ \backslash \ b: \)

\[
a: \quad r_1 \times c
\]
\[
b: \quad r_2 \times c
\]
\[
\text{result:} \quad (r_1+r_2) \times c
\]

Diagnostics

, and \( \backslash \) abort with error if \( a \) and \( b \) are not of the same broad type.
Also see

[M-2] exp — Expressions

[M-2] intro — Language definition