Semicolons — Use of semicolons

Description Syntax Remarks and examples Also see

Description

Mata allows, but does not require, semicolons.

Use of semicolons is discussed below, along with advice on the possible interactions of Stata's `#delimit` instruction; see [P] **#delimit**.

Syntax

stmt

stmt ;

Remarks and examples

Remarks are presented under the following headings:

Optional use of semicolons

You can code your program to look like this

```
real scalar foo(real matrix A)
{
     real scalar   i, sum


     sum = 0
     for (i=1; i<=rows(A); i++) {
          sum = sum + A[i,i]
     }
     return(sum)
}
```

or you can code your program to look like this:

```
real scalar foo(real matrix A)
{
     real scalar   i, sum ;


     sum = 0 ;
     for (i=1; i<=rows(A); i++) {
          sum = sum + A[i,i] ;
     }
     return(sum) ;
}
```

That is, you may omit or include semicolons at the end of statements. It makes no difference. You can even mix the two styles:

```
real scalar foo(real matrix A)
{
     real scalar   i, sum ;


     sum = 0 ;
     for (i=1; i<=rows(A); i++) {
          sum = sum + A[i,i]
     }
     return(sum)
}
```

You cannot break a statement anywhere even if you use semicolons

Most languages that use semicolons follow the rule that a statement continues up to the semicolon.

Mata follows a different rule: a statement continues across lines until it looks to be complete, and semicolons force the end of statements.

For instance, consider the statement `x=b-c` appearing in some program. In the code, might appear

```
x = b -
c
```

or

```
x = b -
c ;
```

and, either way, Mata will understand the statement to be `x=b-c`, because the statement could not possibly end at the minus: `x=b-` makes no sense.

On the other hand,

```
x = b
- c
```

would be interpreted by Mata as two statements: `x=b` and `-c` because `x = b` looks like a completed statement to Mata. The first statement will assign `b` to *x*, and the second statement will display the negative value of `c`.

Adding a semicolon will not help:

```
x = b
- c ;
```

`x = b` is still, by itself, a complete statement. All that has changed is that the second statement ends in a semicolon, and that does not matter.

Thus remember always to break multiline statements at places where the statement could not possibly be interpreted as being complete, such as

```
x = b -
       c + (d
              + e)


myfunction(A,
               B, C,
                      )
```

Do this whether or not you use semicolons.

Use of semicolons to create multistatement lines

Semicolons allow you to put more than one statement on a line. Rather than coding

```
a = 2
b = 3
```

you can code

```
a = 2 ;  b = 3 ;
```

and you can even omit the trailing semicolon:

```
a = 2 ;  b = 3
```

Whether you code separate statements on separate lines or the same line is just a matter of style; it does not change the meaning. Coding

```
for (i=1; i<n; i++) a[i] = -a[i] ; sum = sum + a[i] ;
```

still means

```
for (i=1; i<n; i++) a[i] = -a[i] ;
sum = sum + a[i] ;
```

and, without doubt, the programmer intended to code

```
for (i=1; i<n; i++) {
     a[i] = -a[i] ;
     sum = sum + a[i] ;
}
```

which has a different meaning.

Significant semicolons

Semicolons are not all style. The syntax for the `for` statement is (see [M-2] **for**)

`for (`*exp*$_1$`;` *exp*$_2$`;` *exp*$_3$`)` *stmt*

Say that the complete `for` loop that we want to code is

```
for (x=init(); !converged(x); iterate(x))
```

and that there is no *stmt* following it. Then we must code

```
for (x=init(); !converged(x); iterate(x)) ;
```

Here we use the semicolon to force the end of the statement. Say we omitted it, and the code read

```
...
for (x=init(); !converged(x); iterate(x))
x = -x
...
```

The `for` statement would look incomplete to Mata, so it would interpret our code as if we had coded

```
for (x=init(); !converged(x); iterate(x)) {
     x = -x
}
```

Here the semicolon is significant.

Significant semicolons only happen following `for` and `while`.

Do not use #delimit ;

What follows has to do with Stata's `#delimit ;` mode. If you do not know what it is or if you never use it, you can skip what follows.

Stata has an optional ability to allow its lines to continue up to semicolons. In Stata, you code

```
. #delimit ;
```

and the delimiter is changed to semicolon until your do-file or ado-file ends, or until you code

```
. #delimit cr
```

We recommend that you do not use Mata when Stata is in `#delimit ;` mode. Mata will not mind, but you will confuse yourself.

When Mata gets control, if `#delimit ;` is on, Mata turns it off temporarily, and then Mata applies its own rules, which we have summarized above.

Also see

[M-2] **Intro** — Language definition

[P] **#delimit** — Change delimiter

For suggested citations, see the FAQ on citing Stata documentation.