Title

stata.com

cross( ) — Cross products

Description

`cross()` makes calculations of the form

$$X'X$$
$$X'Z$$
$$X'\,\mathrm{diag}(w)\,X$$
$$X'\,\mathrm{diag}(w)\,Z$$

`cross()` is designed for making calculations that often arise in statistical formulas. In one sense, `cross()` does nothing that you cannot easily write out in standard matrix notation. For instance, `cross(`X`, `Z`)` calculates $X'Z$. `cross()`, however, has the following differences and advantages over the standard matrix-notation approach:

1. `cross()` omits the rows in X and Z that contain missing values, which amounts to dropping observations with missing values.
2. `cross()` uses less memory and is especially efficient when used with views.
3. `cross()` watches for special cases and makes calculations in those special cases more efficiently. For instance, if you code `cross(`X`, `X`)`, `cross()` observes that the two matrices are the same and makes the calculation for a symmetric matrix result.

`cross(`X`, `Z`)` returns $X'Z$. Usually `rows(`X`)==rows(`Z`)`, but X is also allowed to be a scalar, which is then treated as if `J(rows(`Z`), 1, 1)` were specified. Thus `cross(1, `Z`)` is equivalent to `colsum(`Z`)`.

`cross(`X`, `w`, `Z`)` returns X'`diag(`w`)`Z. Usually, `rows(`w`)==rows(`Z`)` or `cols(`w`)==rows(`Z`)`, but w is also allowed to be a scalar, which is treated as if `J(rows(`Z`), 1, `w`)` were specified. Thus `cross(`X`,1,`Z`)` is the same as `cross(`X`,`Z`)`. Z may also be a scalar, just as in the two-argument case.

`cross(`X`, `xc`, `Z`, `zc`)` is similar to `cross(`X`, `Z`)` in that $X'Z$ is returned. In the four-argument case, however, X is augmented on the right with a column of 1s if xc`!=0` and Z is similarly augmented if zc`!=0`. `cross(`X`, 0, `Z`, 0)` is equivalent to `cross(`X`, `Z`)`. Z may be specified as a scalar.

`cross(`X`, `xc`, `w`, `Z`, `zc`)` is similar to `cross(`X`, `w`, `Z`)` in that X'`diag(`w`)`Z is returned. As with the four-argument `cross()`, X is augmented on the right with a column of 1s if xc`!=0` and Z is similarly augmented if zc`!=0`. Both Z and w may be specified as scalars. `cross(`X`, 0, 1, `Z`, 0)` is equivalent to `cross(`X`, `Z`)`.

Syntax

real matrix `cross(`*X*`,` *Z*`)`

real matrix `cross(`*X*`,` *w*`,` *Z*`)`

real matrix `cross(`*X*`,` *xc*`,` *Z*`,` *zc*`)`

real matrix `cross(`*X*`,` *xc*`,` *w*`,` *Z*`,` *zc*`)`

where

X:	*real matrix X*
xc:	*real scalar xc*
w:	*real vector w*
Z:	*real matrix Z*
zc:	*real scalar zc*

Remarks and examples

stata.com

In the following examples, we are going to calculate linear regression coefficients using $b = (X'X)^{-1}X'y$, means using $\sum x/n$, and variances using $n/(n-1) \times (\sum x^2/n - mean^2)$. See [M-5] **crossdev( )** for examples of the same calculations made in a more numerically stable way.

The examples use the automobile data. Because we are using the absolute form of the calculation equations, it would be better if all variables had values near 1 (in which case the absolute form of the calculation equations are perfectly adequate). Thus we suggest

```
. sysuse auto
. replace weight = weight/1000
```

Some of the examples use a weight `w`. For that, you might try

```
. generate w = int(4*runiform())+1
```

▷ Example 1: Linear regression, the traditional way

```
: y = X = .
: st_view(y, ., "mpg")
: st_view(X, ., "weight foreign")
:
: X = X, J(rows(X),1,1)
: b = invsym(X'X)*X'y
```

Comments: Does not handle missing values and uses much memory if `X` is large.

◁

▷ Example 2: Linear regression using cross( )

```
: y = X = .
: st_view(y, ., "mpg")
: st_view(X, ., "weight foreign")
:
: XX = cross(X,1 , X,1)
: Xy = cross(X,1 , y,0)
: b  = invsym(XX)*Xy
```

Comments: There is still an issue with missing values; `mpg` might not be missing everywhere `weight` and `foreign` are missing.

◁

▷ Example 3: Linear regression using cross( ) and one view

```
: // We will form
: //
: //   (y X)'(y X)  =  (y'y, y'X  \  X'y, X'X)
:
: M = .
: st_view(M, ., "mpg weight foreign", 0)
:
: CP = cross(M,1 , M,1)
: XX = CP[|2,2 \ .,.|]
: Xy = CP[|2,1 \ .,1|]
: b  = invsym(XX)*Xy
```

Comments: Using one view handles all missing-value issues (we specified fourth argument 0 to `st_view()`; see [M-5] **st_view( )**).

◁

▷ Example 4: Linear regression using cross( ) and subviews

```
: M = X = y = .
: st_view(M, ., "mpg weight foreign", 0)
: st_subview(y, M, ., 1)
: st_subview(X, M, ., (2\.))
:
: XX = cross(X,1 , X,1)
: Xy = cross(X,1 , y,0)
: b  = invsym(XX)*Xy
```

Comments: Using subviews also handles all missing-value issues; see [M-5] **st_subview( )**. The subview approach is a little less efficient than the previous solution but is perhaps easier to understand. The efficiency issue concerns only the extra memory required by the subviews `y` and `X`, which is not much.

Also, this subview solution could be used to handle the missing-value problems of calculating linear regression coefficients the traditional way, shown in example 1:

```
: M = X = y = .
: st_view(M, ., "mpg weight foreign", 0)
: st_subview(y, M, ., 1)
: st_subview(X, M, ., (2\.))
:
: X = X, J(rows(X), 1, 1)
: b = invsym(X'X)*X'y
```

◁

▷ Example 5: Weighted linear regression, the traditional way

```
: M = w = y = X = .
: st_view(M, ., "w mpg weight foreign", 0)
: st_subview(w, M, ., 1)
: st_subview(y, M, ., 2)
: st_subview(X, M, ., (3\.))
:
: X = X, J(rows(X), 1, 1)
: b = invsym(X'diag(w)*X)*X'diag(w)'y
```

Comments: The memory requirements are now truly impressive because `diag(w)` is an $N \times N$ matrix! That is, the memory requirements are truly impressive when N is large. Part of the power of Mata is that you can write things like `invsym(X'diag(w)*X)*X'diag(w)'y` and obtain solutions. We do not mean to be dismissive of the traditional approach; we merely wish to emphasize its memory requirements and note that there are alternatives.

◁

▷ Example 6: Weighted linear regression using cross( )

```
: M = w = y = X = .
: st_view(M, ., "w mpg weight foreign", 0)
: st_subview(w, M, ., 1)
: st_subview(y, M, ., 2)
: st_subview(X, M, ., (3\.))
:
: XX = cross(X,1 ,w, X,1)
: Xy = cross(X,1 ,w, y,0)
: b  = invsym(XX)*Xy
```

Comments: The memory requirements here are no greater than they were in example 4, which this example closely mirrors. We could also have mirrored the logic of example 3:

```
: M = w = .
: st_view(M, ., "w mpg weight foreign", 0)
: st_subview(w, M, ., 1)
:
: CP = cross(M,1 ,w, M,1)
: XX = CP[|3,3 \ .,.|]
: Xy = CP[|3,2 \ .,2|]
: b  = invsym(XX)*Xy
```

Note how similar these solutions are to their unweighted counterparts. The only important difference is the appearance of `w` as the middle argument of `cross()`. Because specifying the middle argument as a scalar 1 is also allowed and produces unweighted estimates, the above code could be modified to produce unweighted or weighted estimates, depending on how `w` is defined.

◁

▷ Example 7: Mean of one variable

```
: x = .
: st_view(x, ., "mpg", 0)
:
: CP = cross(1,0 , x,1)
: mean = CP[1]/CP[2]
```

Comments: An easier and every bit as good a solution would be

```
: x = .
: st_view(x, ., "mpg", 0)
:
: mean = mean(x,1)
```

`mean()` (see [M-5] **mean( )**) is implemented in terms of `cross()`. Actually, `mean()` is implemented using the quad-precision version of `cross()`; see [M-5] **quadcross( )**. We could implement our solution in terms of `quadcross()`:

```
: x = .
: st_view(x, ., "mpg", 0)
:
: CP = quadcross(1,0 , x,1)
: mean = CP[1]/CP[2]
```

`quadcross()` returns a double-precision result just as does `cross()`. The difference is that `quadcross()` uses quad precision internally in calculating sums.

◁

▷ Example 8: Means of multiple variables

```
: X = .
: st_view(X, ., "mpg weight displ", 0)
:
: CP = cross(1,0 , X,1)
: n  = cols(CP)
: means = CP[|1\n-1|] :/ CP[n]
```

Comments: The above logic will work for one variable, too. With `mean()`, the solution would be

```
: X = .
: st_view(X, ., "mpg weight displ", 0)
:
: means = mean(X, 1)
```

◁

▷ Example 9: Weighted means of multiple variables

```
: M = w = X = .
: st_view(M, ., "w mpg weight displ", 0)
: st_subview(w, M, ., 1)
: st_subview(X, M, ., (2\.))
:
: CP = cross(1,0, w,  X,1)
: n  = cols(CP)
: means = CP[|1\n-1|] :/ CP[n]
```

Comments: Note how similar this solution is to the unweighted solution: `w` now appears as the middle argument of `cross()`. The line `CP = cross(1,0, w, X,1)` could also be coded `CP = cross(w,0, X,1)`; it would make no difference.

The mean() solution to the problem is

```
: M = w = X = .
: st_view(M, ., "w mpg weight displ", 0)
: st_subview(w, M, ., 1)
: st_subview(X, M, ., (2\.))
:
: means = mean(X, w)
```

◁

▷ Example 10: Variance matrix, traditional approach 1

```
: X = .
: st_view(X, ., "mpg weight displ", 0)
:
: n     = rows(X)
: means = mean(X, 1)
: cov   = (X'X/n - means'means)*(n/(n-1))
```

Comments: This above is not 100% traditional since we used mean() to obtain the means, but that does make the solution more understandable. The solution requires calculating X′, requiring that the data matrix be duplicated. Also, we have used a numerically poor calculation formula.

◁

▷ Example 11: Variance matrix, traditional approach 2

```
: X = .
: st_view(X, ., "mpg weight displ", 0)
:
: n     = rows(X)
: means = mean(X, 1)
: cov   = (X:-means)'(X:-means) :/ (n-1)
```

Comments: We use a better calculation formula and, in the process, increase our memory usage substantially.

◁

▷ Example 12: Variance matrix using cross( )

```
: X = .
: st_view(X, ., "mpg weight displ", 0)
:
: n     = rows(X)
: means = mean(X, 1)
: XX    = cross(X, X)
: cov   = ((XX:/n)-means'means)*(n/(n-1))
```

Comments: The above solution conserves memory but uses the numerically poor calculation formula. A related function, crossdev(), will calculate deviation crossproducts:

```
: X = .
: st_view(X, ., "mpg weight displ", 0)
:
: n     = rows(X)
: means = mean(X, 1)
: xx    = crossdev(X,means, X,means)
: cov   = xx:/(n-1)
```

See [M-5] **crossdev( )**. The easiest solution, however, is

```
: X = .
: st_view(X, ., "mpg weight displ", 0)
:
: cov = variance(X, 1)
```

See [M-5] **mean( )** for a description of the `variance()` function. `variance()` is implemented in terms of `crossdev()`.

◁

▷ Example 13: Weighted variance matrix, traditional approaches

```
: M = w = X = .
: st_view(M, ., "w mpg weight displ", 0)
: st_subview(w, M, ., 1)
: st_subview(X, M, ., (2\.))
:
: n     = colsum(w)
: means = mean(X, w)
: cov   = (X'diag(w)*X:/n - means'means)*(n/(n-1))
```

Comments: Above we use the numerically poor formula. Using the better deviation formula, we would have

```
: M = w = X = .
: st_view(M, ., "w mpg weight displ", 0)
: st_subview(w, M, ., 1)
: st_subview(X, M, ., (2\.))
:
: n     = colsum(w)
: means = mean(X, w)
: cov   = (X:-means)'diag(w)*(X:-means) :/ (n-1)
```

The memory requirements include making a copy of the data with the means removed and making an $N \times N$ diagonal matrix.

◁

▷ Example 14: Weighted variance matrix using cross( )

```
: M = w = X = .
: st_view(M, ., "w mpg weight displ", 0)
: st_subview(w, M, ., 1)
: st_subview(X, M, ., (2\.))
:
: n     = colsum(w)
: means = mean(X, w)
: cov   = (cross(X,w,X):/n - means'means)*(n/(n-1))
```

Comments: As in example 12, the above solution conserves memory but uses a numerically poor calculation formula. Better is to use `crossdev()`:

```
: M = w = X = .
: st_view(M, ., "w mpg weight displ", 0)
: st_subview(w, M, ., 1)
: st_subview(X, M, ., (2\.))
:
: n     = colsum(w)
: means = mean(X, w)
: cov   = crossdev(X,means, w, X,means) :/ (n-1)
```

and easiest is to use `variance()`:

```
: M = w = X = .
: st_view(M, ., "w mpg weight displ", 0)
: st_subview(w, M, ., 1)
: st_subview(X, M, ., (2\.))
:
: cov = variance(X, w)
```

See [M-5] **crossdev( )** and [M-5] **mean( )**.

◁

Comment concerning cross( ) and missing values

`cross()` automatically omits rows containing missing values in making its calculation. Depending on this feature, however, is considered bad style because so many other Mata functions do not provide that feature and it is easy to make a mistake.

The right way to handle missing values is to exclude them when constructing views and subviews, as we have done above. When we constructed a view, we invariably specified fourth argument 0 to `st_view()`. In formal programming situations, you will probably specify the name of the `touse` variable you have previously constructed in your ado-file that calls your Mata function.

Conformability

`cross(`X`, `xc`, `w`, `Z`, `zc`)`:

X:	$n \times v_1$	or	1×1,	1×1	treated as if $n \times 1$
xc:	1×1				(optional)
w:	$n \times 1$	or	$1 \times n$	or 1×1	(optional)
Z:	$n \times v_2$				
zc:	1×1				(optional)
result:	$(v_1 + (xc \neq 0)) \times (v_2 + (zc \neq 0))$				

Diagnostics

`cross(`X`, `xc`, `w`, `Z`, `zc`)` omits rows in X and Z that contain missing values.

Also see

[M-5] **crossdev( )** — Deviation cross products

[M-5] **quadcross( )** — Quad-precision cross products

[M-5] **mean( )** — Means, variances, and correlations

[M-4] **statistical** — Statistical functions

[M-4] **utility** — Matrix utility functions