matrix accum — Form cross-product matrices

Description

`matrix accum` accumulates cross-product matrices from the data to form $\mathbf{A} = \mathbf{X}'\mathbf{X}$.

`matrix glsaccum` accumulates cross-product matrices from the data by using a specified inner weight matrix to form $\mathbf{A} = \mathbf{X}'\mathbf{B}\mathbf{X}$, where $\mathbf{B}$ is a block diagonal matrix.

`matrix opaccum` accumulates cross-product matrices from the data by using an inner weight matrix formed from the outer product of a variable in the data to form

$$\mathbf{A} = \mathbf{X}_1'\mathbf{e}_1\mathbf{e}_1'\mathbf{X}_1 + \mathbf{X}_2'\mathbf{e}_2\mathbf{e}_2'\mathbf{X}_2 + \cdots + \mathbf{X}_K'\mathbf{e}_K\mathbf{e}_K'\mathbf{X}_K$$

where $\mathbf{X}_i$ is a matrix of observations from the ith group of the *varlist* variables and $\mathbf{e}_i$ is a vector formed from the observations in the ith group of the *opvar* variable.

`matrix vecaccum` accumulates the first variable against the remaining variables in *varlist* to form a row vector of accumulated inner products to form $\mathbf{a} = \mathbf{x}_1'\mathbf{X}$, where $\mathbf{X} = (\mathbf{x}_2, \mathbf{x}_3, \ldots)$.

Also see [M-5] **cross( )** for other routines for forming cross-product matrices.

Syntax

Accumulate cross-product matrices to form $\mathbf{X}'\mathbf{X}$

```
matrix accum A = varlist [if] [in] [weight] [, noconstant
    deviations means(m) absorb(varname) ]
```

Accumulate cross-product matrices to form $\mathbf{X}'\mathbf{B}\mathbf{X}$

```
matrix glsaccum A = varlist [if] [in] [weight] , group(groupvar)
    glsmat(W|stringvar) row(rowvar) [noconstant]
```

Accumulate cross-product matrices to form $\sum \mathbf{X}_i'\mathbf{e}_i\mathbf{e}_i'\mathbf{X}_i$

```
matrix opaccum A = varlist [if] [in] , group(groupvar)
    opvar(opvar) [noconstant]
```

Accumulate first variable against remaining variables

```
matrix vecaccum a = varlist [if] [in] [weight] [, noconstant]
```

varlist in `matrix accum` and in `matrix vecaccum` may contain factor variables (except for the first variable in `matrix vecaccum` *varlist*); see [U] **11.4.3 Factor variables**.

varlist may contain time-series operators; see [U] **11.4.4 Time-series varlists**.

`collect` is allowed with `matrix accum`, `matrix glsaccum`, `matrix opaccum`, and `matrix vecaccum`; see [U] **11.1.10 Prefix commands**.

`aweight`s, `fweight`s, `iweight`s, and `pweight`s are allowed; see [U] **11.1.6 weight**.

Options

`noconstant` suppresses the addition of a "constant" to the $\mathbf{X}$ matrix. If `noconstant` is not specified, it is as if a column of 1s is added to $\mathbf{X}$ before the accumulation begins. For instance, for `matrix accum` without `noconstant`, $\mathbf{X}'\mathbf{X}$ is really $(\mathbf{X},\mathbf{1})'(\mathbf{X},\mathbf{1})$, resulting in

$$\begin{pmatrix} \mathbf{X}'\mathbf{X} & \mathbf{X}'\mathbf{1} \\ \mathbf{1}'\mathbf{X} & \mathbf{1}'\mathbf{1} \end{pmatrix}$$

Thus the last row and column contain the sums of the columns of $\mathbf{X}$, and the element in the last row and column contains the number of observations. If p variables are specified in *varlist*, the resulting matrix is $(p+1)\times(p+1)$. Specifying `noconstant` suppresses the addition of this row and column (or just the column for `matrix vecaccum`).

`deviations`, allowed only with `matrix accum`, causes the accumulation to be performed in terms of deviations from the mean. If `noconstant` is not specified, the accumulation of $\mathbf{X}$ is done in terms of deviations, but the added row and column of sums are not in deviation format (in which case they would be zeros). With `noconstant` specified, the resulting matrix divided through by $N-1$, where N is the number of observations, is a covariance matrix.

`means(`**m**`)`, allowed only with `matrix accum`, creates matrix **m**: $1\times(p+1)$ or $1\times p$ (depending on whether `noconstant` is also specified) containing the means of $\mathbf{X}$.

`absorb(`*varname*`)`, allowed only with `matrix accum`, specifies that `matrix accum` compute the accumulations in terms of deviations from the mean within the absorption groups identified by *varname*.

`group(`*groupvar*`)` is required with `matrix glsaccum` and `matrix opaccum` and is not allowed otherwise. In the two cases where it is required, it specifies the name of a variable that identifies groups of observations. The data must be sorted by *groupvar*.

In `matrix glsaccum`, *groupvar* identifies the observations to be individually weighted by `glsmat()`.

In `matrix opaccum`, *groupvar* identifies the observations to be weighted by the outer product of `opvar()`.

`glsmat(`**W** | *stringvar*`)`, required with `matrix glsaccum` and not allowed otherwise, specifies the name of the matrix or the name of a string variable in the dataset that contains the name of the matrix that is to be used to weight the observations in `group()`. *stringvar* must be `str8` or less.

`row(`*rowvar*`)`, required with `matrix glsaccum` and not allowed otherwise, specifies the name of a numeric variable containing the row numbers that specify the row and column of the `glsmat()` matrix to use in the inner-product calculation.

`opvar(`*opvar*`)`, required with `matrix opaccum`, specifies the variable used to form the vector whose outer product forms the weighting matrix.

Remarks and examples

Remarks are presented under the following headings:

matrix accum

matrix accum is a straightforward command that accumulates one matrix that holds $\mathbf{X}'\mathbf{X}$ and $\mathbf{X}'\mathbf{y}$, which is typically used in $\mathbf{b} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$. Say that we wish to run a regression of the variable price on mpg and weight. We can begin by accumulating the full cross-product matrix for all three variables:

```
. use https://www.stata-press.com/data/r19/auto
(1978 automobile data)

. matrix accum A = price weight mpg
(obs=74)

. matrix list A

symmetric A[4,4]
             price     weight        mpg      _cons
 price   3.448e+09
weight   1.468e+09  7.188e+08
   mpg     9132716    4493720      36008
 _cons      456229     223440       1576         74
```

In our accumulation, matrix accum automatically added a constant; we specified three variables and got back a 4 × 4 matrix. The constant term is always added last. In terms of our regression model, the matrix we just accumulated has $\mathbf{y} = \texttt{price}$ and $\mathbf{X} = (\texttt{weight}, \texttt{mpg}, \texttt{\_cons})$ and can be written as

$$\mathbf{A} = (\mathbf{y}, \mathbf{X})'(\mathbf{y}, \mathbf{X}) = \begin{pmatrix} \mathbf{y}'\mathbf{y} & \mathbf{y}'\mathbf{X} \\ \mathbf{X}'\mathbf{y} & \mathbf{X}'\mathbf{X} \end{pmatrix}$$

Thus we can extract $\mathbf{X}'\mathbf{X}$ from the submatrix of $\mathbf{A}$ beginning at the second row and column, and we can extract $\mathbf{X}'\mathbf{y}$ from the first column of $\mathbf{A}$, omitting the first row:

```
. matrix XX = A[2...,2...]

. matrix list XX

symmetric XX[3,3]
            weight        mpg      _cons
weight   7.188e+08
   mpg     4493720      36008
 _cons      223440       1576         74

. matrix Xy = A[2...,1]

. matrix list Xy

Xy[3,1]
             price
weight   1.468e+09
   mpg     9132716
 _cons      456229
```

We can now calculate $\mathbf{b} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$:

```
. matrix b = invsym(XX)*Xy

. matrix list b

b[3,1]
              price
weight    1.7465592
   mpg   -49.512221
 _cons    1946.0687
```

The same result could have been obtained directly from $\mathbf{A}$:

```
. matrix b = invsym(A[2...,2...])*A[2...,1]
```

❏ Technical note

matrix accum, with the deviations and noconstant options, can also be used to obtain covariance matrices. The covariance between variables x_i and x_j is defined as

$$C_{ij} = \frac{\sum_{k=1}^{n}(x_{ik} - \overline{x}_i)(x_{jk} - \overline{x}_j)}{n-1}$$

Without the deviations option, matrix accum calculates a matrix with elements

$$R_{ij} = \sum_{k=1}^{n} x_{ik}x_{jk}$$

and with the deviations option,

$$A_{ij} = \sum_{k=1}^{n}(x_{ik} - \overline{x}_i)(x_{jk} - \overline{x}_j)$$

Thus the covariance matrix $\mathbf{C} = \mathbf{A}/(n-1)$.

```
. matrix accum Cov = price weight mpg, deviations noconstant
(obs=74)

. matrix Cov = Cov/(r(N)-1)

. matrix list Cov

symmetric Cov[3,3]
             price      weight         mpg
 price     8699526
weight   1234674.8   604029.84
   mpg  -7996.2829  -3629.4261   33.472047
```

In addition to calculating the cross-product matrix, matrix accum records the number of observations in r(N), a feature we use in calculating the normalizing factor. With the corr() matrix function defined in [P] **matrix define**, we can convert the covariance matrix into a correlation matrix:

```
. matrix P = corr(Cov)

. matrix list P

symmetric P[3,3]
             price      weight         mpg
 price           1
weight   .53861146           1
   mpg  -.46859669  -.80717486           1
```

◁

matrix glsaccum

matrix glsaccum is a generalization of matrix accum useful in producing GLS-style weighted accumulations. Whereas matrix accum produces matrices of the form $\mathbf{X'X}$, matrix glsaccum produces matrices of the form $\mathbf{X'BX}$, where

$$B = \begin{pmatrix} \mathbf{W}_1 & 0 & \dots & 0 \\ 0 & \mathbf{W}_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \mathbf{W}_K \end{pmatrix}$$

The matrices $\mathbf{W}_k$, $k = 1, \ldots, K$ are called the weighting matrices for observation group k. In the matrices above, each of the $\mathbf{W}_k$ matrices is square, but there is no assumption that they all have the same dimension. By writing

$$\mathbf{X} = \begin{pmatrix} \mathbf{X}_1 \\ \mathbf{X}_2 \\ \vdots \\ \mathbf{X}_K \end{pmatrix}$$

the accumulation made by `matrix glsaccum` can be written as

$$\mathbf{X}'\mathbf{B}\mathbf{X} = \mathbf{X}_1'\mathbf{W}_1\mathbf{X}_1 + \mathbf{X}_2'\mathbf{W}_2\mathbf{X}_2 + \cdots + \mathbf{X}_K'\mathbf{W}_K\mathbf{X}_K$$

`matrix glsaccum` requires you to specify three options: `group(`*groupvar*`)`, `glsmat(`*stringvar*`)` or `glsmat(`*matvar*`)`, and `row(`*rowvar*`)`. Observations sharing the same value of *groupvar* are said to be in the same observation group—this specifies the group, k, in which they are to be accumulated. Before calling `matrix glsaccum`, you must `sort` the data by *groupvar*. How $\mathbf{W}_k$ is assembled is the subject of the other two options.

Think of there being a superweighting matrix for the group, which we will call $\mathbf{V}_k$. $\mathbf{V}_k$ is specified by `glsmat()`. The same supermatrix can be used for all observations by specifying a *matname* as the argument to `glsmat()`, or, if a variable name is specified, different supermatrices can be specified—the contents of the variable will be used to obtain the particular name of the supermatrix. (More correctly, the contents of the variable for the first observation in the group will be used: supermatrices can vary across groups but must be the same within group.)

Weighting matrix $\mathbf{W}_k$ is made from supermatrix $\mathbf{V}_k$ by selecting the rows and columns specified in `row(`*rowvar*`)`. In the simple case, $\mathbf{W}_k = \mathbf{V}_k$. This happens when there are m observations in the group and the first observation in the group has *rowvar* $= 1$, the second has *rowvar* $= 2$, and so on. To fix ideas, let $m = 3$ and write

$$\mathbf{V}_1 = \begin{pmatrix} v_{11} & v_{12} & v_{13} \\ v_{21} & v_{22} & v_{23} \\ v_{31} & v_{32} & v_{33} \end{pmatrix}$$

$\mathbf{V}$ need not be symmetric. Let's pretend that the first 4 observations in our dataset contain

obs. no.	*groupvar*	*rowvar*
1	1	1
2	1	2
3	1	3
4	2	...

In these data, the first 3 observations are in the first group because they share an equal *groupvar*. It is not important that *groupvar* happens to equal 1; it is important that the values are equal. The *rowvars* are, in order, 1, 2, and 3, so $\mathbf{W}_1$ is formed by selecting the first row and column of $\mathbf{V}_1$, then the second row and column of $\mathbf{V}_1$, and finally the third row and column of $\mathbf{V}_1$:

$$\mathbf{W}_1 = \begin{pmatrix} v_{11} & v_{12} & v_{13} \\ v_{21} & v_{22} & v_{23} \\ v_{31} & v_{32} & v_{33} \end{pmatrix}$$

or $\mathbf{W}_1 = \mathbf{V}_1$. Now consider the same data, but reordered:

obs. no.	*groupvar*	*rowvar*
1	1	2
2	1	1
3	1	3
4	2	...

$\mathbf{W}_1$ is now formed by selecting the second row and column, then the first row and column, and finally the third row and column of $\mathbf{V}_1$. These steps can be performed sequentially, reordering first the rows and then the columns; the result is

$$\mathbf{W}_1 = \begin{pmatrix} v_{22} & v_{21} & v_{23} \\ v_{12} & v_{11} & v_{13} \\ v_{32} & v_{31} & v_{33} \end{pmatrix}$$

This reorganization of the $\mathbf{W}_1$ matrix exactly undoes the reorganization of the $\mathbf{X}_1$ matrix, so $\mathbf{X}_1'\mathbf{W}_1\mathbf{X}_1$ remains unchanged. Given how $\mathbf{W}_k$ is assembled from $\mathbf{V}_k$, the order of the row numbers in the data does not matter.

`matrix glsaccum` is willing to carry this concept even further. Consider the following data:

obs. no.	*groupvar*	*rowvar*
1	1	1
2	1	3
3	1	3
4	2	...

Now *rowvar* equals 1 followed by 3 twice, so the first row and column of $\mathbf{V}_1$ are selected, followed by the third row and column twice; the second column is never selected. The resulting weighting matrix is

$$\mathbf{W}_1 = \begin{pmatrix} v_{11} & v_{13} & v_{13} \\ v_{31} & v_{33} & v_{33} \\ v_{31} & v_{33} & v_{33} \end{pmatrix}$$

Such odd weighting would not occur in, say, time-series analysis, where the matrix might be weighting lags and leads. It could well occur in an analysis of individuals in families, where 1 might indicate the head of household, 2 a spouse, and 3 a child. In fact, such a case could be handled with a 3×3 superweighting matrix *V*, even if the family became large: the appropriate weighting matrix $\mathbf{W}_k$ would be assembled, on a group-by-group (family-by-family) basis, from the underlying supermatrix.

matrix opaccum

`matrix opaccum` is a special case of `matrix glsaccum`. `matrix glsaccum` calculates results of the form

$$\mathbf{A} = \mathbf{X}_1'\mathbf{W}_1\mathbf{X}_1 + \mathbf{X}_2'\mathbf{W}_2\mathbf{X}_2 + \cdots + \mathbf{X}_K'\mathbf{W}_K\mathbf{X}_K$$

Often $\mathbf{W}_i$ is simply the outer product of another variable in the dataset; that is,

$$\mathbf{W}_i = \mathbf{e}_i\mathbf{e}_i'$$

where $\mathbf{e}_i$ is the $n_i \times 1$ vector formed from the n_i `groupvar()` observations of the variable specified in `opvar()`. The data must be sorted by *groupvar*.

▷ Example 1

Suppose that we have a panel dataset that contains five variables: `id`, `t`, `e` (a residual), and covariates `x1` and `x2`. Further suppose that we need to compute

$$\mathbf{A} = \mathbf{X}_1'\mathbf{e}_1\mathbf{e}_1'\mathbf{X}_1 + \mathbf{X}_2'\mathbf{e}_2\mathbf{e}_2'\mathbf{X}_2 + \cdots + \mathbf{X}_K'\mathbf{e}_K\mathbf{e}_K'\mathbf{X}_K$$

where $\mathbf{X}_i$ contains the observations on `x1` and `x2` when `id==i` and $\mathbf{e}_i$ contains the observations on `e` when `id==i`.

Below is the output from `xtdescribe` for our example data. There are 11 groups and the number of observations per group is not constant.

```
. use https://www.stata-press.com/data/r19/maccumxmpl

. xtdescribe, patterns(11)

      id:  1, 2, ..., 11                                         n =         11
       t:  1, 2, ..., 15                                         T =         15
           Delta(t) = 1 unit
           Span(t)  = 15 periods
           (id*t uniquely identifies each observation)

Distribution of T_i:   min      5%     25%       50%       75%     95%     max
                         5       5       7        10        13      15      15

     Freq.  Percent    Cum. |  Pattern
 ---------------------------+-----------------
        1      9.09    9.09 |  11111..........
        1      9.09   18.18 |  111111.........
        1      9.09   27.27 |  1111111........
        1      9.09   36.36 |  11111111.......
        1      9.09   45.45 |  111111111......
        1      9.09   54.55 |  1111111111.....
        1      9.09   63.64 |  11111111111....
        1      9.09   72.73 |  111111111111...
        1      9.09   81.82 |  1111111111111..
        1      9.09   90.91 |  11111111111111.
        1      9.09  100.00 |  111111111111111
 ---------------------------+-----------------
       11    100.00         |  XXXXXXXXXXXXXXX
```

If we were to calculate **A** with `matrix glsaccum`, we would need to form 11 matrices and store their names in a string variable before calling `matrix glsaccum`. This step slows down `matrix glsaccum` when there are many groups. Also all the information contained in the $\mathbf{W}_i$ matrices is contained in the variable `e`. It is this structure that `matrix opaccum` exploits to make a faster command for this type of problem:

```
. sort id t

. matrix opaccum  A2 = x1 x2, group(id) opvar(e)
```

◁

matrix vecaccum

The first variable in *varlist* is treated differently from the others by `matrix vecaccum`. Think of the first variable as specifying vector $\mathbf{y}$ and the remaining variables as specifying matrix $\mathbf{X}$. `matrix vecaccum` makes the accumulation $\mathbf{y}'\mathbf{X}$ to return a row vector with elements

$$a_i = \sum_{k=1}^{n} y_k x_{ki}$$

Like `matrix accum`, `matrix vecaccum` adds a constant, `_cons`, to $\mathbf{X}$ unless `noconstant` is specified.

`matrix vecaccum` serves two purposes. First, terms like $\mathbf{y}'\mathbf{X}$ often occur in calculating derivatives of likelihood functions; `matrix vecaccum` provides a fast way of calculating them. Second, it is useful in time-series accumulations of the form

$$\mathbf{C} = \sum_{t=1}^{T} \sum_{\delta=-k}^{k} \mathbf{x}'_{t-\delta}\mathbf{x}_t W_\delta r_{t-\delta} r_t$$

In this calculation, $\mathbf{X}$ is an observation matrix with elements x_{tj}, with t indexing time (observations) and j variables, $t = 1, \dots, T$ and $j = 1, \dots, p$. $\mathbf{x}_t$ $(1 \times p)$ refers to the tth row of this matrix. Thus $\mathbf{C}$ is a $p \times p$ matrix.

The Newey–West covariance matrix uses the definition $W_\delta = 1 - |\delta|/(k+1)$ for $\delta \le k$. To make the calculation, the user (programmer) cycles through each of the j variables, forming

$$z_{tj} = \sum_{\delta=-k}^{k} x_{(t-\delta)j} W_\delta r_{t-\delta} r_t$$

Writing $\mathbf{z}_j = (z_{1j}, z_{2j}, \dots, z_{Tj})'$, we can then say that $\mathbf{C}$ is

$$\mathbf{C} = \sum_{j=1}^{p} \mathbf{z}'_j \mathbf{X}$$

In this derivation, the user must decide in advance the maximum lag length, k, such that observations that are far apart in time must have increasingly small covariances to establish the convergence results.

The Newey–West estimator is in the class of generalized method of moments (GMM) estimators. The choice of a maximum lag length, k, is a reflection of the length in time beyond which the autocorrelation becomes negligible for estimating the variance matrix. The code fragment given below is merely for illustration of the matrix commands, because Stata includes estimation with the Newey–West covariance matrix in the `newey` command. See [TS] **newey** or Greene (2018, 999) for details on this estimator.

Calculations like $\mathbf{z}_j'\mathbf{X}$ are made by `matrix vecaccum`, and $\mathbf{z}_j$ can be treated as a temporary variable in the dataset.

```
assume ‘1’,‘2’, etc., contain the xs including constant
assume ‘r’ contains the r variable
assume ‘k’ contains the k range
tempname C factor t c
tempvar z

local p : word count ‘*’
matrix ‘C’ = J(‘p’,‘p’,0)
generate double ‘z’ = 0
forvalues d = 0/‘k’ {
                        /* Add each submatrix twice except for
                           the lag==0 case */
        scalar ‘factor’ = cond(‘d’>0, 1, .5)

        local w = (1 - ‘d’/(‘k’+1))
        capture mat drop ‘t’
        forvalues j = 1/‘p’ {
                replace ‘z’ = ‘‘j’’[_n-‘d’]*‘w’*‘r’[_n-‘d’]*‘r’
                mat vecaccum ‘c’ = ‘z’ ‘*’, nocons
                mat ‘t’ = ‘t’ \ ‘c’
        }
        mat ‘C’ = ‘C’ + (‘t’ + ‘t’’)*‘factor’
}
local ‘p’ = "_cons"                   // Rename last var to _cons
mat rownames ‘C’ = ‘*’
mat colnames ‘C’ = ‘*’
assume inverse and scaling for standard error reports
```

Treatment of user-specified weights

`matrix accum`, `matrix glsaccum`, and `matrix vecaccum` all allow weights. Here is how they are treated:

All three commands can be thought of as returning something of the form $\mathbf{X}_1'\mathbf{B}\mathbf{X}_2$. `matrix accum`, $\mathbf{X}_1 = \mathbf{X}_2$ and $\mathbf{B} = \mathbf{I}$; for `matrix glsaccum`, $\mathbf{X}_1 = \mathbf{X}_2$; and `matrix vecaccum`, $\mathbf{B} = \mathbf{I}$, $\mathbf{X}_1$ is a column vector and $\mathbf{X}_2$ is a matrix.

The commands really calculate $\mathbf{X}_1'\mathbf{W}^{1/2}\mathbf{B}\mathbf{W}^{1/2}\mathbf{X}_2$, where $\mathbf{W}$ is a diagonal matrix. If no weights are specified, $\mathbf{W} = \mathbf{I}$. Now assume that weights are specified, and let $\mathbf{v}$: $1 \times n$ be the specified weights. If `fweight`s or `pweight`s are specified, $\mathbf{W} = \text{diag}(\mathbf{v})$. If `aweight`s are specified, $\mathbf{W} = \text{diag}\{\mathbf{v}/(\mathbf{1}'\mathbf{v})(\mathbf{1}'\mathbf{1})\}$, meaning that the weights are normalized to sum to the number of observations. If `iweight`s are specified, they are treated like `fweight`s, except that the elements of $\mathbf{v}$ are not restricted to be positive integers.

Stored results

matrix accum, matrix glsaccum, matrix opaccum, and matrix vecaccum store the number of observations in r(N). matrix accum stores the number of absorption groups in r(k_absorb). matrix glsaccum (with aweights) and matrix vecaccum also store the sum of the weight in r(sum_w), but matrix accum does not.

Reference

Greene, W. H. 2018. *Econometric Analysis*. 8th ed. New York: Pearson.

Also see

[P] **matrix** — Introduction to matrix commands

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

[R] **ml** — Maximum likelihood estimation

[U] 14 Matrix expressions