14 Matrix expressions

Contents

14.1 Overview

Stata has two matrix programming languages, one that might be called Stata's older matrix language and another that is called Mata. Stata's Mata is the new one, and there is an uneasy relationship between the two.

Below we discuss Stata's older language and leave the newer one to another manual—the *Mata Reference Manual* ([M])—or you can learn about the newer one by typing `help mata`.

We admit that the newer language is better in almost every way than the older language, but the older one still has a use because it is the one that Stata truly and deeply understands. Even when Mata wants to talk to Stata, matrixwise, it is the older language that Mata must use, so you must learn to use the older language as well as the new.

This is not nearly as difficult, or messy, as you might imagine because Stata's older language is remarkably easy to use, and really, there is not much to learn. Just remember that for heavy-duty programming, it will be worth your time to learn Mata, too.

14.1.1 Definition of a matrix

Stata's definition of a matrix includes a few details that go beyond the mathematics. To Stata, a matrix is a named entity containing an $r \times c$ rectangular array of double-precision numbers (including missing values) that is bordered by a row and a column of names. For the dimensions of a matrix, see [R] **Limits**.

```
. matrix list A
A[3,2]
    c1  c2
r1   1   2
r2   3   4
r3   5   6
```

Here we have a 3×2 matrix named `A` containing elements 1, 2, 3, 4, 5, and 6. Row 1, column 2 (written $A_{1,2}$ in math and `A[1,2]` in Stata) contains 2. The columns are named `c1` and `c2` and the rows, `r1`, `r2`, and `r3`. These are the default names Stata comes up with when it cannot do better. The names do not play a role in the mathematics, but they are of great help when it comes to labeling the output.

The names are operated on just as the numbers are. For instance,

```
. matrix B=A'*A
. matrix list B
symmetric B[2,2]
    c1  c2
c1  35
c2  44  56
```

We defined $\mathbf{B} = \mathbf{A}'\mathbf{A}$. The row and column names of $\mathbf{B}$ are the same. Multiplication is defined for any $a \times b$ and $b \times c$ matrices, the result being $a \times c$. Thus the row and column names of the result are the row names of the first matrix and the column names of the second matrix. We formed $\mathbf{A}'\mathbf{A}$, using the transpose of $\mathbf{A}$ for the first matrix—which also interchanged the names—and so obtained the names shown.

14.2 Row and column names

Matrix rows and columns always have names. Stata is smart about setting these names when the matrix is created, and the matrix commands and operators manipulate these names throughout calculations, so the names typically are set correctly at the conclusion of matrix calculations.

For instance, consider the matrix calculation $\mathbf{b} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$ performed on real data:

```
. use https://www.stata-press.com/data/r17/auto
(1978 automobile data)
. matrix accum XprimeX = weight foreign
(obs=74)
. matrix vecaccum yprimeX = mpg weight foreign
. matrix b = syminv(XprimeX)*yprimeX'
. matrix list b
b[3,1]
                mpg
 weight  -.00658789
foreign  -1.6500291
  _cons   41.679702
```

These names were produced without our ever having given a special command to place the names on the result. When we formed matrix `XprimeX`, Stata produced the result

```
. matrix list XprimeX
symmetric XprimeX[3,3]
            weight    foreign      _cons
 weight  7.188e+08
foreign      50950         22
  _cons     223440         22         74
```

`matrix accum` forms $\mathbf{X}'\mathbf{X}$ matrices from data and sets the row and column names to the variable names used. The names are correct in the sense that, for instance, the (1,1) element is the sum across the observations of squares of `weight` and the (2,1) element is the sum of the product of `weight` and `foreign`.

Similarly, `matrix vecaccum` forms $\mathbf{y}'\mathbf{X}$ matrices, and it sets the row and column names to the variable names used, so `matrix vecaccum yprimeX = mpg weight foreign` resulted in

```
. matrix list yprimeX
yprimeX[1,3]
      weight  foreign    _cons
mpg  4493720      545     1576
```

The final step, `matrix b = invsym(XprimeX)*yprimeX'`, manipulated the names, and, if you think carefully, you can derive the rules for yourself. `invsym()` (inversion) is much like transposition, so row and column names must be swapped. Here, however, the matrix was symmetric, so that amounted to leaving the names as they were. Multiplication amounts to taking the column names of the first matrix and the row names of the second. The final result is

```
. matrix list b
b[3,1]
                mpg
 weight  -.00658789
foreign  -1.6500291
  _cons   41.679702
```

and the interpretation is $\texttt{mpg} = -0.00659\,\texttt{weight} - 1.65\,\texttt{foreign} + 41.68 + e$.

Researchers realized long ago that using matrix notation simplifies the description of complex calculations. What they may not have realized is that, corresponding to each mathematical definition of a matrix operator, there is a definition of the operator's effect on the names that can be used to carry the names forward through long and complex matrix calculations.

14.2.1 The purpose of row and column names

Mostly, matrices in Stata are used in programming estimators, and Stata uses row and column names to produce pretty output. Say that we wrote code—interactively or in a program—that produced the following coefficient vector `b` and covariance matrix `V`:

```
. matrix list b
b[1,3]
           weight  displacement         _cons
y1    -.00656711     .00528078     40.084522
. matrix list V
symmetric V[3,3]
                    weight  displacement         _cons
      weight     1.360e-06
displacement     -.0000103     .00009741
       _cons    -.00207455     .01188356     4.0808455
```

We could now produce standard estimation output by coding two more lines:

```
. ereturn post b V
. ereturn display
```

	Coefficient	Std. err.	z	P>|z|	[95% conf.	interval]
weight	-.0065671	.0011662	-5.63	0.000	-.0088529	-.0042813
displacement	.0052808	.0098696	0.54	0.593	-.0140632	.0246248
_cons	40.08452	2.02011	19.84	0.000	36.12518	44.04387

Stata's `ereturn` command knew to produce this output because of the row and column names on the coefficient vector and variance matrix. Moreover, we usually do nothing special in our code that produces `b` and `V` to set the row and column names because, given how matrix names work, they work themselves out.

Also, sometimes row and column names help us detect programming errors. Assume that we wrote code to produce matrices `b` and `V` but made a mistake. Sometimes our mistake will result in the wrong row and column names. Rather than the `b` vector we previously showed you, we might produce

```
. matrix list b
b[1,3]
        weight          c2       _cons
y1  -.00656711       42.23   40.084522
```

If we `post`ed our estimation results now, Stata would refuse because it can tell by the names that there is a problem:

```
. ereturn post b V
name conflict
r(507);
```

Understand, however, that Stata follows the standard rules of matrix algebra; the names are just along for the ride. Matrices are summed by position, meaning that a directive to form $\mathbf{C} = \mathbf{A} + \mathbf{B}$ results in $C_{11} = A_{11} + B_{11}$, regardless of the names, and it is not an error to sum matrices with different names:

```
. matrix list a
symmetric a[3,3]
               c1          c2          c3
   mpg      14419
weight    1221120  1.219e+08
 _cons        545       50950          22
. matrix list b
symmetric b[3,3]
                   c1        c2        c3
displacement  3211055
         mpg   227102     22249
       _cons    12153      1041        52
. matrix c = a + b
. matrix list c
symmetric c[3,3]
                    c1          c2          c3
displacement    3225474
         mpg    1448222  1.219e+08
       _cons      12698       51991          74
```

Matrix row and column names are used to label output; they do not affect how matrix algebra is performed.

14.2.2 Two-part names

Row and column names have two parts separated by a colon: *equation_name*:*opvarname*.

In the examples shown so far, the *equation_name* has been blank and the *opvarname*s have been simple variable names without factor-variable or time-series operators. A blank *equation_name* is typical. Run any single-equation model (such as `regress`, `probit`, or `logistic`), and if you fetch the resulting matrices, you will find that they have row and column names that use only *opvarname*s.

Those who work with time-series data will find matrices with row and column names of the form *opvarname*. For time-series variables, *opvarname* is the variable name prefixed by a time-series operator such as `L.`, `D.`, or `L2D.`; see [U] 11.4.4 Time-series varlists. For example,

```
. matrix list example1
symmetric example1[3,3]
                            L.
              rate        rate       _cons
  rate   3.0952534
L.rate    .0096504   .00007742
 _cons  -2.8413483  -.01821928   4.8578916
```

We obtained this matrix by running a linear regression on `rate` and `L.rate` and then fetching the covariance matrix. Think of the row and column name `L.rate` no differently from how you think of `rate` or, in the previous examples, `r1`, `r2`, `c1`, `c2`, `weight`, and `foreign`.

Those who work with factor variables will also find row and column names of the *opvarname* form. For factor variables, *opvarname* is any factor-variable construct that references a single virtual indicator variable. For example, `3.group` refers to the virtual variable that is `1` when `group` = 3 and is `0` otherwise, `1.sex#3.group` refers to the virtual variable that is `1` when `sex` = 1 and `group` = 3 and is `0` otherwise, and `1.sex#c.age` refers to the virtual variable that takes on the values of `age` when `sex` = 1 and is `0` otherwise. For example,

```
. matrix list example2
symmetric example2[5,5]
                     0b.          1.     0b.sex#      1.sex#
                    sex         sex       c.age       c.age       _cons
     0b.sex           0
      1.sex           0   7.7785864
0b.sex#c.age          0   .08350827   .00231307
 1.sex#c.age          0  -.09705697   5.606e-17   .00223195
      _cons           0  -3.2868185  -.08350827  -2.131e-15   3.2868185
```

`1.sex#c.age` is a row name and column name just like `rate` or `L.rate` in the prior example. For details on factor variables and valid factor-variable constructs see [U] 11.4.3 Factor variables, [U] 26 Working with categorical data and factor variables, [U] 13.9 Indicator values for levels of factor variables, and [U] 20.12 Accessing estimated coefficients.

Factor-variable operators may be combined with the time-series operators `L.` and `F.`, leading to *opvarnames* such as `1L.sex` (the first lag of the level 1 indicator of `sex`) and `3L2.group` (the second lag of the level 3 indicator of `group`).

Equation names are used to label partitioned matrices and, in estimation, occur in the context of multiple equations. Here is a matrix with *equation_names* and simple (unoperated) *opvarnames*.

```
. matrix list example3
symmetric example2[5,5]
                      mpg:        mpg:        mpg:        mpg:        mpg:
                  foreign       displ       _cons     foreign       _cons
   mpg:foreign   1.6483972
     mpg:displ     .004747   .00003876
     mpg:_cons  -1.4266352  -.00905773   2.4341021
weight:foreign  -51.208454  -4.665e-19   15.224135   24997.727
  weight:_cons   15.224135   2.077e-17  -15.224135  -7431.7565   7431.7565
```

Here is an example with *equation_names* and operated variable names:

```
. matrix list example4
symmetric example3[5,5]
                      val:        val:        val:     weight:     weight:
                                    L.
                     rate        rate       _cons     foreign       _cons
      val:rate   2.2947268
    val:L.rate   .00385216    .0000309
     val:_cons  -1.4533912   -.0072726   2.2583357
weight:foreign  -163.86684   7.796e-17   49.384526   25351.696
  weight:_cons   49.384526  -1.566e-16  -49.384526   -7640.237    7640.237
```

`val:L.rate` is a column name, just as, in the previous section, `c2` and `foreign` were column names.

Say that this last matrix is the variance matrix produced by a program we wrote and that our program also produced a coefficient vector, `b`:

```
. matrix list b
b[1,5]
           val:        val:        val:     weight:     weight:
                         L.
          rate        rate       _cons     foreign       _cons
y1   4.5366753  -.00316923    20.68421  -1008.7968   3324.7059
```

Here is the result of posting and displaying the results:

```
. ereturn post b example4
. ereturn display
```

	Coefficient	Std. err.	z	P>|z|	[95% conf.	interval]
val						
rate						
—	4.536675	1.514836	2.995	0.003	1.567652	7.505698
L1	-.0031692	.0055591	-0.570	0.569	-.0140648	.0077264
_cons	20.68421	1.502776	13.764	0.000	17.73882	23.6296
weight						
foreign	-1008.797	159.2222	-6.336	0.000	-1320.866	-696.7271
_cons	3324.706	87.40845	38.036	0.000	3153.388	3496.023

We have been using `matrix list` to see the row and column names on our matrices because `matrix list` works on all matrices. There is a better way to see the names when we are working with estimation results because estimation results have the same names on the rows and columns of the variance matrix, and those same names are also the column names for the coefficient vector. That better way is the `coeflegend` display option available on almost every estimation command. For example,

```
. use https://www.stata-press.com/data/r17/fvex
(Artificial factor variables' data)
. generate t = _n
. tsset t
  (output omitted)
. sureg (y = sex##group) (distance = d.age i1.2.sex)
  (output omitted)
. sureg, coeflegend
Seemingly unrelated regression
```

Equation	Obs	Params	RMSE	"R-squared"	chi2	P>chi2
y	2,998	5	20.03657	0.1343	464.08	0.0000
distance	2,998	2	181.3797	0.0005	0.92	0.6314

	Coefficient	Legend
y		
sex		
female	21.59726	_b[y:1.sex]
group		
2	11.42832	_b[y:2.group]
3	21.6461	_b[y:3.group]
sex#group		
female#2	-4.892653	_b[y:1.sex#2.group]
female#3	-6.220653	_b[y:1.sex#3.group]
_cons	50.5957	_b[y:_cons]
distance		
age		
D1.	.2230927	_b[distance:D.age]
L2.sex		
female	1.300898	_b[distance:1L2.sex]
_cons	57.96172	_b[distance:_cons]

We could have used `matrix list e(V)` or `matrix list e(b)` to see the names, but the limited space available to `matrix list` to write the names would have made the names more difficult to read. With `coeflegend`, the names are neatly arrayed in their own `Legend` column. One difference between `matrix list` and the `coeflegend` option is that `coeflegend` brackets the names with `_b[]`. That is because `coeflegend`'s primary use is to show us how to type coefficients in expressions and postestimation commands; see **[U] 13.5 Accessing coefficients and standard errors** and **[U] 20.12 Accessing estimated coefficients**. There the `_b[]` is required.

14.2.3 Setting row and column names

You reset row and column names by using the `matrix rownames` and `matrix colnames` commands.

Before resetting the names, use `matrix list` to verify that the names are not set correctly; often, they already are. When you enter a matrix by hand, however, the row names are unimaginatively set to `r1`, `r2`, ..., and the column names to `c1`, `c2`, ...

```
. matrix a = (1,2,3\4,5,6)
. matrix list a
a[2,3]
    c1  c2  c3
r1   1   2   3
r2   4   5   6
```

Regardless of the current row and column names, `matrix rownames` and `matrix colnames` reset them:

```
. matrix colnames a = foreign alpha _cons
. matrix rownames a = one two
. matrix list a
a[2,3]
     foreign    alpha    _cons
one        1        2        3
two        4        5        6
```

You may set the *operator* as part of the *opvarname*,

```
. matrix colnames a = foreign l.rate _cons
. matrix list a
a[2,3]
                  L.
     foreign     rate    _cons
one        1        2        3
two        4        5        6
```

The names you specify may be any virtual factor-variable indicators, and those names may include the base (`b.`) and omitted (`o.`) operators,

```
. matrix colnames b = 0b.sex 2o.arm 1.sex#c.age 1.sex#3.group#2.arm
. matrix list b
b[2,4]
                                        1.sex#
           0b.       2o.    1.sex#  3.group#
          sex       arm     c.age     2.arm
one         1         2         3         3
two         5         6         7         8
```

See [U] **11.4.3 Factor variables** for more about factor-variable operators.

You may set equation names:

```
. matrix colnames a = this:foreign this:l.rate that:_cons
. matrix list a
a[2,3]
        this:    this:    that:
                  L.
     foreign     rate    _cons
one        1        2        3
two        4        5        6
```

See [P] **matrix rownames** for more information.

14.2.4 Obtaining row and column names

`matrix list` displays the matrix with its row and column names. In a programming context, you can fetch the row and column names into a macro using

```
local ... : rowfullnames matname
local ... : colfullnames matname
local ... : rownames matname
local ... : colnames matname
local ... : roweq matname
local ... : coleq matname
```

`rowfullnames` and `colfullnames` return the full names (*equation_name*`:`*opvarnames*) listed one after the other.

`rownames` and `colnames` omit the equations and return *opvarnames*, listed one after the other.

`roweq` and `coleq` return the equation names, listed one after the other.

See [P] **macro** and [P] **matrix define** for more information.

14.3 Vectors and scalars

Stata does not have vectors as such—they are considered special cases of matrices and are handled by the `matrix` command.

Stata does have scalars, although they are not strictly necessary because they, too, could be handled as special cases. See [P] **scalar** for a description of scalars.

14.4 Inputting matrices by hand

You input matrices using

`matrix input` *matname* `= (...)`

or

`matrix` *matname* `= (...)`

In either case, you enter the matrices by row. You separate one element from the next by using commas (`,`) and one row from the next by using backslashes (\). If you omit the word `input`, you are using the expression parser to input the matrix:

```
. matrix a = (1,2\3,4)
. matrix list a
a[2,2]
    c1  c2
r1   1   2
r2   3   4
```

This has the advantage that you can use expressions for any of the elements:

```
. matrix b = (1, 2+3/2 \ cos(_pi), _pi)
. matrix list b
b[2,2]
            c1         c2
r1           1        3.5
r2          -1  3.1415927
```

The disadvantage is that the matrix must be small, say, no more than 50 elements.

`matrix input` has no such restriction, but you may not use subexpressions for the elements:

```
. matrix input c  = (1,2\3,4)
. matrix input d = (1, 2+3/2 \ cos(_pi), _pi)
invalid syntax
r(198);
```

Either way, after inputting the matrix, you will probably want to set the row and column names; see [U] **14.2.3 Setting row and column names** above.

For small matrices, you may prefer entering them in a dialog box. Launch the dialog box from the menu **Data > Matrices, ado language > Input matrix by hand**, or by typing `db matrix_input`. The dialog box is particularly convenient for small symmetric matrices.

14.5 Accessing matrices created by Stata commands

Some Stata commands—including all estimation commands—leave behind matrices that you can subsequently use. After executing an estimation command, type `ereturn list` to see what is available:

```
. use https://www.stata-press.com/data/r17/auto
(1978 automobile data)
. probit foreign mpg weight
  (output omitted)
. ereturn list
scalars:
                e(rank) =  3
                   e(N) =  74
                  e(ic) =  5
                   e(k) =  3
                e(k_eq) =  1
                e(k_dv) =  1
           e(converged) =  1
                  e(rc) =  0
                  e(ll) =  -26.84418900579869
         e(k_eq_model) =  1
                e(ll_0) =  -45.03320955699139
                e(df_m) =  2
                e(chi2) =  36.3780411023854
                   e(p) =  1.26069126402e-08
               e(N_cdf) =  0
               e(N_cds) =  0
                e(r2_p) =  .4039023807124771
macros:
            e(cmdline) : "probit foreign mpg weight"
                e(cmd) : "probit"
          e(estat_cmd) : "probit_estat"
            e(predict) : "probit_p"
          e(marginsok) : "default Pr"
       e(marginsnotok) : "stdp DEviance SCore"
              e(title) : "Probit regression"
           e(chi2type) : "LR"
                e(opt) : "moptimize"
                e(vce) : "oim"
               e(user) : "mopt__probit_d2()"
          e(ml_method) : "d2"
          e(technique) : "nr"
              e(which) : "max"
             e(depvar) : "foreign"
         e(properties) : "b V"
```

```
matrices:
                  e(b) :  1 x 3
                  e(V) :  3 x 3
                e(mns) :  1 x 3
              e(rules) :  1 x 4
               e(ilog) :  1 x 20
           e(gradient) :  1 x 3
functions:
             e(sample)
```

Most estimation commands leave behind `e(b)` (the coefficient vector) and `e(V)` (the variance–covariance matrix of the estimator):

```
. matrix list e(b)
e(b)[1,3]
       foreign:    foreign:    foreign:
           mpg      weight       _cons
y1  -.10395033  -.00233554    8.275464
```

You can refer to `e(b)` and `e(V)` in any matrix expression:

```
. matrix myb = e(b)
. matrix list myb
myb[1,3]
       foreign:    foreign:    foreign:
           mpg      weight       _cons
y1  -.10395033  -.00233554    8.275464
. matrix c = e(b)*invsym(e(V))*e(b)'
. matrix list c
symmetric c[1,1]
           y1
y1  22.440542
```

14.6 Creating matrices by accumulating data

In programming estimators, matrices of the form $\mathbf{X}'\mathbf{X}$, $\mathbf{X}'\mathbf{Z}$, $\mathbf{X}'\mathbf{W}\mathbf{X}$, and $\mathbf{X}'\mathbf{W}\mathbf{Z}$ often occur, where $\mathbf{X}$ and $\mathbf{Z}$ are data matrices. `matrix accum`, `matrix glsaccum`, `matrix vecaccum`, and `matrix opaccum` produce such matrices; see [P] **matrix accum**.

We recommend that you not load the data into a matrix and use the expression parser directly to form such matrices, although see [P] **matrix mkmat** if that is your interest. If that is your interest, be sure to read the technical note at the end of [P] **matrix mkmat**. There is much to recommend learning how to use the `matrix accum` commands.

14.7 Matrix operators

You can create new matrices or replace existing matrices by typing

`matrix` *matname* `=` *matrix_expression*

For instance,

```
. matrix A = invsym(R*V*R')
. matrix IAR = I(rowsof(A)) - A*R
. matrix beta = b*IAR' + r*A'
. matrix C = -C'
. matrix D = (A, B \ B', A)
. matrix E = (A+B)*C'
. matrix S = (S+S')/2
```

The following operators are provided:

Operator	Symbol
Unary operators	
negation	-
transposition	'
Binary operators	
(lowest precedence)	
row join	\
column join	,
addition	+
subtraction	-
multiplication	*
division by scalar	/
Kronecker product	#
(highest precedence)	

Parentheses may be used to change the order of evaluation.

Note in particular that , and \ are operators; `(1,2)` creates a 1×2 matrix (vector), and `(A,B)` creates a `rowsof(A)` $\times$ `colsof(A)+colsof(B)` matrix, where `rowsof(A)` $=$ `rowsof(B)`. `(1\2)` creates a 2×1 matrix (vector), and `(A\B)` creates a `rowsof(A)+rowsof(B)` $\times$ `colsof(A)` matrix, where `colsof(A)` $=$ `colsof(B)`. Thus expressions of the form

```
matrix R = (A,B)*Vinv*(A,B)'
```

are allowed.

14.8 Matrix functions

In addition to the functions listed below, see [P] **matrix svd** for singular value decomposition, [P] **matrix symeigen** for eigenvalues and eigenvectors of symmetric matrices, and see [P] **matrix eigenvalues** for eigenvalues of nonsymmetric matrices. For a full description of the matrix functions, see [FN] **Matrix functions**.

Matrix functions returning matrices:

`cholesky(`M`)`	`I(`n`)`	`nullmat(`*matname*`)`
`corr(`M`)`	`inv(`M`)`	`sweep(`M`,`i`)`
`diag(`v`)`	`invsym(`M`)`	`vec(`M`)`
`get(`*systemname*`)`	`J(`r`,`c`,`z`)`	`vecdiag(`M`)`
`hadamard(`M`,`N`)`	`matuniform(`r`,`c`)`	

Matrix functions returning scalars:

`coleqnumb(`M`,`s`)`	`diag0cnt(`M`)`	`roweqnumb(`M`,`s`)`
`colnfreeparms(`M`)`	`el(`M`,`i`,`j`)`	`rownfreeparms(`M`)`
`colnumb(`M`,`s`)`	`issymmetric(`M`)`	`rownumb(`M`,`s`)`
`colsof(`M`)`	`matmissing(`M`)`	`rowsof(`M`)`
`det(`M`)`	`mreldif(`X`,`Y`)`	`trace(`M`)`

14.9 Subscripting

1. In matrix and scalar expressions, you may refer to *matname*[r,c], where r and c are scalar expressions, to obtain one element of *matname* as a scalar.

   Examples:
   ```
   matrix A = A / A[1,1]
   generate newvar = oldvar / A[2,2]
   ```

2. In matrix expressions, you may refer to *matname*[s_r,s_c], where s_r and s_c are string expressions, to obtain a submatrix with one element. The element returned is based on searching the row and column names.

   Examples:
   ```
   matrix B = V["price","price"]
   generate sdif = dif / sqrt(V["price","price"])
   ```

3. In matrix expressions, you may mix these two syntaxes and refer to *matname*[r,s_c] or to *matname*[s_r,c].

   Example:
   ```
   matrix b = b * R[1,"price"]
   ```

4. In matrix expressions, you may use *matname*[r_1..r_2,c_1..c_2] to refer to submatrices; r_1, r_2, c_1, and c_2 may be scalar expressions. If r_2 evaluates to missing, it is taken as referring to the last row of *matname*; if c_2 evaluates to missing, it is taken as referring to the last column of *matname*. Thus *matname*[r_1...,c_1...] is allowed.

   Examples:
   ```
   matrix S = Z[1..4, 1..4]
   matrix R = Z[5..., 5...]
   ```

5. In matrix expressions, you may refer to *matname*[s_{r1}..s_{r2},s_{c1}..s_{c2}] to refer to submatrices where s_{r1}, s_{r2}, s_{c1}, and s_{c2}, are string expressions. The matrix returned is based on looking up the row and column names.

   If the string evaluates to an equation name only, all the rows or columns for the equation are returned.

   Examples:
   ```
   matrix S = Z["price".."weight", "price".."weight"]
   matrix L = D["mpg:price".."mpg:weight", "mpg:price".."mpg:weight"]
   matrix T1 = C["mpg:", "mpg:"]
   matrix T2 = C["mpg:", "price:"]
   ```

6. In matrix expressions, any of the above syntaxes may be combined.

   Examples:
   ```
   matrix T1 = C["mpg:", "price:weight".."price:displ"]
   matrix T2 = C["mpg:", "price:weight"...]
   matrix T3 = C["mpg:price", 2..5]
   matrix T4 = C["mpg:price", 2]
   ```

7. When defining an element of a matrix, use

   `matrix` *matname*`[`i`,`j`] =` *expression*

   where i and j are scalar expressions. The matrix *matname* must already exist.

   Example:
   ```
   matrix A = J(2,2,0)
   matrix A[1,2] = sqrt(2)
   ```

8. To replace a submatrix within a matrix, use the same syntax. If the expression on the right evaluates to a scalar or 1×1 matrix, the element is replaced. If it evaluates to a matrix, the submatrix with top-left element at (i, j) is replaced. The matrix *matname* must already exist.

   Example:
   ```
   matrix A = J(4,4,0)
   matrix A[2,2] = C'*C
   ```

14.10 Using matrices in scalar expressions

Scalar expressions are documented as *exp* in the Stata manuals:

```
generate newvar = exp if exp ...
replace  newvar = exp if exp ...
regress ... if exp ...
if exp {... }
while exp {... }
```

Most importantly, scalar expressions occur in `generate` and `replace`, in the `if` *exp* qualifier allowed on the end of many commands, and in the `if` and `while` commands for program control.

You will rarely need to refer to a matrix in any of these situations except when using the `if` qualifier and the `while` command.

In any case, you may refer to matrices in any of these situations, but the expression cannot require evaluation of matrix expressions returning matrices. Thus you could refer to `trace(A)` but not to `trace(A+B)`.

It can be difficult to predict when an evaluation of an expression requires evaluating a matrix; even experienced users can be surprised. If you get the error message "matrix operators that return matrices not allowed in this context", r(509), you have encountered such a situation.

The solution is to split the line in two. For instance, you would change

```
if trace(A+B)==0 {
        ...
}
```

to

```
matrix AplusB = A+B
if trace(AplusB)==0 {
        ...
}
```

or even to

```
matrix Trace = trace(A+B)
if Trace[1,1]==0 {
        ...
}
```

14.11 Reference

Miura, H. 2012. Stata graph library for network analysis. *Stata Journal* 12: 94–129.