Title

stata.com

matrix — Introduction to matrix commands

Description Remarks and examples Also see

Description

An introduction to matrices in Stata is found in **[U] 14 Matrix expressions**. This entry provides an overview of the `matrix` commands and provides more background information on matrices in Stata.

Beyond the `matrix` commands, Stata has a complete matrix programming language, Mata, that provides more advanced matrix functions, support for complex matrices, fast execution speed, and the ability to directly access Stata's data, macros, matrices, and returned results. Mata can be used interactively as a matrix calculator, but it is even more useful for programming; see the *Mata Reference Manual*.

Remarks and examples

stata.com

Remarks are presented under the following headings:

Overview of matrix commands

Documentation on matrices in Stata is grouped below into three categories—Basics, Programming, and Specialized. We recommend that you begin with **[U] 14 Matrix expressions** and then read **[P] matrix define**. After that, feel free to skip around.

Basics

[U] 14 Matrix expressions	Introduction to matrices in Stata
[P] matrix define	Matrix definition, operators, and functions
[P] matrix utility	List, rename, and drop matrices
[P] matlist	Display a matrix and control its format

Programming

[P] matrix accum	Form cross-product matrices
[R] ml	Maximum likelihood estimation
[P] ereturn	Post the estimation results
[P] matrix rownames	Name rows and columns
[P] matrix score	Score data from coefficient vectors

Specialized

[P] makecns	Constrained estimation
[P] matrix mkmat	Convert variables to matrix and vice versa
[P] matrix svd	Singular value decomposition
[P] matrix symeigen	Eigenvalues and eigenvectors of symmetric matrices
[P] matrix eigenvalues	Eigenvalues of nonsymmetric matrices
[P] matrix get	Access system matrices
[P] matrix dissimilarity	Compute similarity or dissimilarity measures

Creating and replacing matrices

Matrices generally do not have to be preallocated or dimensioned before creation, except when you want to create an $r \times c$ matrix and then fill in each element one by one; see the description of the `J()` function in [P] **matrix define**. Matrices are typically created by `matrix define` or `matrix accum`; see [P] **matrix accum**.

Stata takes a high-handed approach to redefining matrices. You know that, when dealing with data, you must distinguish between creating a new variable or replacing the contents of an existing variable—Stata has two commands for this: `generate` and `replace`. For matrices, there is no such distinction. If you define a new matrix, it is created. If you give the same command and the matrix already exists, then the currently existing matrix is destroyed and the new one is defined. This treatment is the same as that given to macros and scalars.

Namespace

The term "namespace" refers to how names are interpreted. For instance, the variables in your dataset occupy one namespace—other things, such as value labels, macros, and scalars, can have the same name and not cause confusion.

Macros also have their own namespace; macros can have the same names as other things, and Stata can still tell by context when you are referring to a macro because of the punctuation. When you type `gen newvar=myname`, `myname` must refer to a variable. When you type `gen newvar=‘myname’`—note the single quotes around `myname`—`myname` must refer to a local macro. When you type `gen newvar=$myname`, `myname` must refer to a global macro.

Scalars and matrices share the same namespace; that is, scalars and matrices may have the same names as variables in the dataset, etc., but they cannot have the same names as each other. Thus when you define a matrix called, say, `myres`, if a scalar by that name already exists, it is destroyed, and the matrix replaces it. Correspondingly, when you define a scalar called `myres`, if a matrix by that name exists, it is destroyed, and the scalar replaces it.

Naming conventions in programs

If you are writing Stata programs or ado-files using matrices, you may have some matrices that you wish to leave behind for other programs to build upon, but you will certainly have other matrices that are nothing more than leftovers from calculations. Such matrices are called *temporary*. You should use Stata's `tempname` facility (see [P] **macro**) to name such matrices. These matrices will automatically be discarded when your program ends. For example, a piece of your program might read

```
tempname YXX XX
matrix accum ‘YXX’ = price weight mpg
matrix ‘XX’ = ‘YXX’[2...,2...]
```

Note the single quotes around the names after they are obtained from `tempname`; see [U] **18.3 Macros**.

❑ Technical note

Let's consider writing a regression program in Stata. (There is actually no need for such a program because Stata already has the `regress` command.) A well-written estimation command would allow the `level()` option for specifying the width of confidence intervals, and it would replay results when the command is typed without arguments. Here is a well-written version:

```
program myreg, eclass
        version 15.1
        if !replay() {
                syntax varlist(min=2 numeric) [if] [in] [, Level(cilevel)]
                marksample touse      // mark the sample
                tempname YXX XX Xy b hat V
                // compute cross products YXX = (Y'Y , Y'X \ X'Y , X'X)
                quietly matrix accum `YXX' = `varlist' if `touse'
                local nobs = r(N)
                local df = `nobs' - (rowsof(`YXX') - 1)
                matrix `XX' = `YXX'[2...,2...]
                matrix `Xy' = `YXX'[1,2...]
                // compute the beta vector
                matrix `b' = `Xy' * invsym(`XX')
                // compute the covariance matrix
                matrix `hat' = `b' * `Xy''
                matrix `V' = invsym(`XX') * (`YXX'[1,1] - `hat'[1,1])/`df'
                // post the beta vector and covariance matrix
                ereturn post `b' `V', dof(`df') obs(`nobs') depname(`1') /*
                                        */ esample(`touse')
                // save estimation information
                tokenize "`varlist'"  // put varlist into numbered arguments
                ereturn local depvar "`1'"
                ereturn local cmd "myreg"
        }
        else {  // replay
                syntax [, Level(cilevel)]
        }
        if "`e(cmd)'"!="myreg" error 301
        // print the regression table
        ereturn display, level(`level')
end
```

The syntax of our new command is

myreg *depvar indepvars* [*if*] [*in*] [, level(#)]

`myreg`, typed without arguments, redisplays the output of the last `myreg` command. After estimation with `myreg`, the user may use `correlate` to display the covariance matrix of the estimators, `predict` to obtain predicted values or standard errors of the prediction, and `test` to test linear hypotheses about the estimated coefficients. The command is indistinguishable from any other Stata estimation command.

Despite the excellence of our work, we do have some criticisms:

- `myreg` does not display the ANOVA table, R^2, etc.; it should and could be made to, although we would have to insert our own `display` statements before the `ereturn display` instruction.
- The program makes copious use of matrices with different names, resulting in extra memory use while the estimation is being made; the code could be made more economical, if less readable, by reusing matrices.
- `myreg` makes the least-squares calculation by using the absolute cross-product matrix, an invitation to numerical problems if the data are not consistently scaled. Stata's own `regress` command is more careful, and we could be, too: `matrix accum` does have an option for forming the cross-product matrix in deviation form, but its use would complicate this program. This does not overly concern us, although we should make a note of it when we document `myreg`. Nowadays, users

expect to be protected in linear regression but have no such expectations for more complicated estimation schemes because avoiding the problem can be difficult.

There is one nice feature of our program that did not occur to us when we wrote it. We use `invsym()` to form the inverse of the cross-product matrix, and `invsym()` can handle singular matrices. If there is a collinearity problem, `myreg` behaves just like `regress`: it omits the offending variables and notes that they are omitted when it displays the output (at the `ereturn display` step).

❑

❑ Technical note

Our linear regression program is longer than we might have written in an exclusively matrix programming language. After all, the coefficients can be obtained from $(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$, and in a dedicated matrix language, we would type nearly that, and obtaining the standard errors would require only a few more matrix calculations. In fact, we did type nearly that to make the calculation; the extra lines in our program have to do mostly with syntax issues and linking to the rest of Stata. In writing your own programs, you might be tempted not to bother linking to the rest of Stata. Fight this temptation.

Linking to the rest of Stata pays off: here we do not merely display the numerical results, but we display them in a readable form, complete with variable names. We made a command that is indistinguishable from Stata's other estimation commands. If the user wants to test `_b[denver]=_b[la]`, the user types literally that; there is no need to remember the matrix equation and to count variables (such as constrain the third minus the 15th variable to sum to zero).

❑

Also see

[P] **ereturn** — Post the estimation results

[P] **matrix define** — Matrix definition, operators, and functions

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

[U] 14 Matrix expressions

[U] 18 Programming Stata

Mata Reference Manual