matrix mkmat — Convert variables to matrix and vice versa

Syntax

Create matrix from variables

```
mkmat varlist [if] [in] [, matrix(matname) nomissing rownames(varname)
    roweq(varname) rowprefix(string) obs nchar(#)]
```

Create variables from matrix

```
svmat [type] A [, names(col|eqcol|matcol|string)]
```

Rename rows and columns of matrix

```
matname A namelist [, rows(range) columns(range) explicit]
```

where **A** is the name of an existing matrix, *type* is a storage type for the new variables, and *namelist* is one of 1) a varlist, that is, names of existing variables possibly abbreviated; 2) `_cons` and the names of existing variables possibly abbreviated; or 3) arbitrary names when the `explicit` option is specified.

Menu

mkmat

Data > Matrices, ado language > Convert variables to matrix

svmat

Data > Matrices, ado language > Convert matrix to variables

Description

`mkmat` stores the variables listed in *varlist* in column vectors of the same name, that is, $N \times 1$ matrices, where N = `_N`, the number of observations in the dataset. Optionally, they can be stored as an $N \times k$ matrix, where k is the number of variables in *varlist*. The variable names are used as column names. By default, the rows are named `r1`, `r2`, ...

`svmat` takes a matrix and stores its columns as new variables. It is the reverse of the `mkmat` command, which creates a matrix from existing variables.

`matname` renames the rows and columns of a matrix. `matname` differs from the `matrix rownames` and `matrix colnames` commands in that `matname` expands varlist abbreviations and allows a restricted range for the rows or columns. See [P] **matrix rownames**.

Options

`matrix(`*matname*`)` requests that the vectors be combined in a matrix instead of creating the column vectors.

`nomissing` specifies that observations with missing values in any of the variables be excluded ("listwise deletion").

`rownames(`*varname*`)` and `roweq(`*varname*`)` specify that the row names and row equations of the created matrix or vectors be taken from *varname*. *varname* should be a string variable or an integer positive-valued numeric variable. [Value labels are ignored; use `decode` (see [D] **encode**) if you want to use value labels.] Within the names, spaces and periods are replaced by an underscore (_).

`rowprefix(`*string*`)` specifies that the string *string* be prefixed to the row names of the created matrix or column vectors. In the prefix, spaces and periods are replaced by an underscore (_). If `rownames()` is not specified, `rowprefix()` defaults to `r`, and to nothing otherwise.

`obs` specifies that the observation numbers be used as row names. This option may not be combined with `rownames()`.

`nchar(`*#*`)` specifies that row names be truncated to # characters, $1 \leq \# \leq 32$. The default is `nchar(32)`.

`names(col|eqcol|matcol|`*string*`)` specifies how the new variables are to be named.
`names(col)` uses the column names of the matrix to name the variables.
`names(eqcol)` uses the equation names prefixed to the column names.
`names(matcol)` uses the matrix name prefixed to the column names.
`names(`*string*`)` names the variables *string*1, *string*2, ..., *stringn*, where *string* is a user-specified *string* and *n* is the number of columns of the matrix.
If `names()` is not specified, the variables are named **A**1, **A**2, ..., **A***n*, where **A** is the name of the matrix.

`rows(`*range*`)` and `columns(`*range*`)` specify the rows and columns of the matrix to rename. The number of rows or columns specified must be equal to the number of names in *namelist*. If both `rows()` and `columns()` are given, the specified rows are named *namelist*, and the specified columns are also named *namelist*. The range must be given in one of the following forms:

`rows(.)`	renames all the rows
`rows(2..8)`	renames rows 2–8
`rows(3)`	renames only row 3
`rows(4...)`	renames row 4 to the last row

If neither `rows()` nor `columns()` is given, `rows(.) columns(.)` is the default. That is, the matrix must be square, and both the rows and the columns are named *namelist*.

`explicit` suppresses the expansion of varlist abbreviations and omits the verification that the names are those of existing variables. That is, the names in *namelist* are used explicitly and can be any valid row or column names.

Remarks and examples

stata.com

Remarks are presented under the following headings:

mkmat
svmat

mkmat

Although cross products of variables can be loaded into a matrix with the `matrix accum` command (see [P] **matrix accum**), programmers may sometimes find it more convenient to work with the variables in their datasets as vectors instead of as cross products. `mkmat` allows the user a simple way to load specific variables into matrices in Stata's memory.

▷ Example 1

`mkmat` uses the variable name to name the single column in the vector. This feature guarantees that the variable name will be carried along in any additional matrix calculations. This feature is also useful when vectors are combined in a general matrix.

```
. use http://www.stata-press.com/data/r13/test

. describe

Contains data from http://www.stata-press.com/data/r13/test.dta
  obs:            10
 vars:             3                          13 Apr 2013 12:50
 size:           120
-------------------------------------------------------------------------------
              storage   display    value
variable name   type    format     label      variable label
-------------------------------------------------------------------------------
x               float   %9.0g
y               float   %9.0g
z               float   %9.0g
-------------------------------------------------------------------------------
Sorted by:

. list

     +--------------+
     |  x    y    z |
     |--------------|
  1. |  1   10    2 |
  2. |  2    9    4 |
  3. |  3    8    3 |
  4. |  4    7    5 |
  5. |  5    6    7 |
     |--------------|
  6. |  6    5    6 |
  7. |  7    4    8 |
  8. |  8    3   10 |
  9. |  9    2    1 |
 10. | 10    1    9 |
     +--------------+

. mkmat x y z, matrix(xyzmat)

. matrix list xyzmat

xyzmat[10,3]
      x   y   z
 r1   1  10   2
 r2   2   9   4
 r3   3   8   3
 r4   4   7   5
 r5   5   6   7
 r6   6   5   6
 r7   7   4   8
 r8   8   3  10
 r9   9   2   1
r10  10   1   9
```

If the variables contain missing values, so will the corresponding matrix or matrices. Many matrix commands, such as the matrix inversion functions `inv()` and `invsym()`, do not allow missing values in matrices. If you specify the `nomissing` option, `mkmat` will exclude observations with missing values so that subsequent matrix computations will not be hampered by missing values. Listwise deletion parallels missing-value handling in most Stata commands.

◁

❑ Technical note

`mkmat` provides a useful addition to Stata's matrix commands, but it will work only with small datasets.

Stata limits matrices to no more than matsize × matsize, which means a maximum of 800 × 800 for Stata/IC and 11,000 × 11,000 for Stata/SE and Stata/MP. By limiting Stata's matrix capabilities to matsize × matsize, has not Stata's matrix language itself been limited to datasets no larger than matsize? It would certainly appear so; in the simple matrix calculation for regression coefficients $(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$, $\mathbf{X}$ is an $n \times k$ matrix (n being the number of observations and k being the number of variables), and given the matsize constraint, n must be less than 800 (or up to 11,000 in Stata/MP and Stata/SE).

Our answer is as follows: yes, $\mathbf{X}$ is limited in the way stated, but $\mathbf{X}'\mathbf{X}$ is a mere $k \times k$ matrix, and, similarly, $\mathbf{X}'\mathbf{y}$ is only $k \times 1$. Both of these matrices are well within Stata's matrix-handling capabilities, and the `matrix accum` command (see [P] **matrix accum**) can directly create both of them.

Moreover, even if Stata could hold the $n \times k$ matrix $\mathbf{X}$, it would still be more efficient to use `matrix accum` to form $\mathbf{X}'\mathbf{X}$. $\mathbf{X}'\mathbf{X}$, interpreted literally, says to load a copy of the dataset, transpose it, load a second copy of the dataset, and then form the matrix product. Thus two copies of the dataset occupy memory in addition to the original copy Stata already had available (and from which `matrix accum` could directly form the result with no additional memory use). For small n, the inefficiency is not important, but for large n, the inefficiency could make the calculation infeasible. For instance, with $n = 12{,}000$ and $k = 6$, the additional memory use is 1,125 kilobytes.

More generally, matrices in statistical applications tend to have dimensions $k \times k$, $n \times k$, and $n \times n$, with k small and n large. Terms dealing with the data are of the generic form $\mathbf{X}'_{k_1 \times n}\mathbf{W}_{n \times n}\mathbf{Z}_{n \times k_2}$. ($\mathbf{X}'\mathbf{X}$ fits the generic form with $\mathbf{X} = \mathbf{X}$, $\mathbf{W} = \mathbf{I}$, and $\mathbf{Z} = \mathbf{X}$.) Matrix programming languages cannot deal with the deceptively simple calculation $\mathbf{X}'\mathbf{W}\mathbf{Z}$ because of the staggering size of the $\mathbf{W}$ matrix. For $n = 12{,}000$, storing $\mathbf{W}$ requires a little more than a gigabyte of memory. In statistical formulas, however, $\mathbf{W}$ is given by formula and, in fact, never needs to be stored in its entirety. Exploitation of this fact is all that is needed to resurrect the use of a matrix programming language in statistical applications. Matrix programming languages may be inefficient because of copious memory use, but in statistical applications, the inefficiency is minor for matrices of size $k \times k$ or smaller. Our design of the various `matrix accum` commands allows calculating terms of the form $\mathbf{X}'\mathbf{W}\mathbf{Z}$, and this one feature is all that is necessary to allow efficient and robust use of matrix languages.

Programs for creating data matrices, such as that offered by `mkmat`, are useful for pedagogical purposes and for a specific application where Stata's matsize constraint is not binding, it seems so natural. On the other hand, it is important that general tools not be implemented by forming data matrices because such tools will be drastically limited in dataset size. Coding the problem in terms of the various `matrix accum` commands (see [P] **matrix accum**) is admittedly more tedious, but by abolishing data matrices from your programs, you will produce tools suitable for use on large datasets.

❑

svmat

▷ Example 2

Let's get the vector of coefficients from a regression and use `svmat` to save the vector as a new variable, save the dataset, load the dataset back into memory, use `mkmat` to create a vector from the variable, and finally, use `matname` to rename the columns of the row vector.

```
. use http://www.stata-press.com/data/r13/auto
(1978 Automobile Data)

. quietly regress mpg weight gear_ratio foreign

. matrix b = e(b)

. matrix list b

b[1,4]
        weight  gear_ratio     foreign       _cons
y1  -.00613903   1.4571134  -2.2216815   36.101353

. matrix c = b'

. svmat double c, name(bvector)

. list bvector1 in 1/5

     +------------+
     |   bvector1 |
     |------------|
  1. | -.00613903 |
  2. |  1.4571134 |
  3. | -2.2216815 |
  4. |  36.101353 |
  5. |          . |
     +------------+

. save example
file example.dta saved

. use example

. mkmat bvector1 if bvector1< .

. matrix list bvector1

bvector1[4,1]
      bvector1
r1  -.00613903
r2   1.4571134
r3  -2.2216815
r4   36.101353

. matrix d = bvector1'

. matname d wei gear for _cons, c(.)

. matrix list d

d[1,4]
              weight  gear_ratio     foreign       _cons
bvector1  -.00613903   1.4571134  -2.2216815   36.101353
```

◁

Acknowledgment

`mkmat` was written by Ken Heinecke of the Federal Reserve Bank of Minneapolis.

References

Gould, W. W. 1994. ip6.1: Data and matrices. *Stata Technical Bulletin* 20: 10. Reprinted in *Stata Technical Bulletin Reprints*, vol. 4, pp. 70–71. College Station, TX: Stata Press.

Heinecke, K. 1994. ip6: Storing variables in vectors and matrices. *Stata Technical Bulletin* 20: 8–9. Reprinted in *Stata Technical Bulletin Reprints*, vol. 4, pp. 68–70. College Station, TX: Stata Press.

Sribney, W. M. 1995. ip6.2: Storing matrices as variables. *Stata Technical Bulletin* 24: 9–10. Reprinted in *Stata Technical Bulletin Reprints*, vol. 4, pp. 71–73. College Station, TX: Stata Press.

Also see

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

[P] **matrix accum** — Form cross-product matrices

[M-4] **stata** — Stata interface functions

[U] **14 Matrix expressions**