The Stata Journal (2023)
23, Number 4, pp. 1045–1056 DOI: 10.1177/1536867X231212439

Improving flexibility and ease of matrix subsetting: The submatrix command

Daniele Spinelli
Department of Statistics and Quantitative Methods
University of Milano–Bicocca
Milan, Italy
daniele.spinelli@unimib.it

Abstract. Matrix manipulation in Stata can be a time-consuming and tedious task, especially when it is necessary to subset or rearrange elements from large matrices based on nonconsecutive elements. Compared with Mata, these tasks require more time, more code, and sometimes more complex output. The purpose of this article is to introduce `submatrix`, a command to manipulate matrix elements using row (and column) names, numbers, and equations.

Keywords: pr0077, submatrix, row names, column names, permutation vectors

1 Introduction

The use of matrices in statistics and econometrics is extensive. For instance, many spatial regression models require large contiguity matrices (Anselin 1988), while intergenerational mobility analysis and Markov models are based on transition matrices. In Stata, matrices are used for many purposes, spanning from the storage of regression coefficients and descriptive statistics to the construction of design matrices for regression. Matrix manipulation is also a useful tool for data entry, recoding of categorical variables, and creating categorical variables from continuous variables based on given thresholds (Cox 2012).

In the regression framework, models may have long coefficient vectors and large variance–covariance matrices, but research interest is often focused on a few coefficients. For example, in causal inference (Cunningham 2021), an event-study regression model including the interaction between the treatment variable and the time variable may also include a large set of control variables. However, scholars may be more interested in the former because it conveys information about the treatment effect and the parallel trends assumption. Another application in which regressions may produce large matrices is in discrete choice models (multinomial logistic regression, conditional logistic regression), for which coefficients are often estimated at the alternative level (Train 2009). Such situations involve the estimation of a large coefficient vector, but most attention is devoted to a subset of coefficients related to the treatment variable and its interactions (causal inference) or to some of the alternatives in the choice set.

However, when it is necessary to subset or rearrange elements of large matrices, the task of matrix manipulation can be time consuming and tedious. This is especially the case in Stata, in which these tasks require more time and code and sometimes produce more complex output compared with Mata. This is because, in the Stata environment, subscripting multiple elements of a matrix is allowed only when the elements are consecutive. For example, displaying the first three columns of a matrix `A` can be achieved with the command `matlist A[1 .. rowsof(A), 1 .. 3]`. However, displaying the first, fifth, and seventh columns of the same matrix requires using the command `matlist (A[1 .. rowsof(A), 1], A[1 .. rowsof(A), 5], A[1 .. rowsof(A), 7])`, which involves three instances of subscripting the columns of `A`; the inclusion of all the rows is also repeated three times. In addition, Stata allows subsetting matrices based on column and row names; however, this feature is limited to one name at a time.

The purpose of this article is to address these issues by introducing `submatrix`, a community-contributed command that supports nonadvanced users in the advanced extraction of submatrices from Stata matrices. The `submatrix` command subsets matrices from multiple nonconsecutive rows or columns using number or name subscripting. This command is particularly suitable for users who are not familiar with Mata because it introduces permutation vectors in the Stata environment exploiting *numlist*; this feature is already available in Mata and allows easier and faster matrix subscripting. Furthermore, `submatrix` allows replication, rearrangement, and deletion of elements from matrices.

The remainder of this article is structured as follows. Section 2 presents the syntax of `submatrix`; section 3 illustrates the use of `submatrix` in the contexts of panel-data regression, intergenerational mobility, and discrete choice models; section 4 concludes.

2 Syntax

The syntax of `submatrix` is

`submatrix` *matname*, *subsetting_options* [*other_options*]

where *matname* is an existing matrix. Users are required to specify at least one of the *subsetting_options* (see section 2.1).

2.1 Subsetting options

These options will be combined to define the criteria for subsetting *matname*. The options allow repeated arguments.

`rownames(`*string*`)` controls the names of the rows to be kept from matrix *matname*. It exits with an error if the row names of *matname* do not contain all the elements of *string* unless the `ignore` option is used (see `help matrix_subscripting`). Double quotes may be used to enclose strings that contain spaces.

`droprownames(`*string*`)` controls the names of the rows to be dropped from matrix *matname*. It exits with an error if the row names of *matname* do not contain all the elements of *string* unless the `ignore` option is used (see `help matrix_subscripting`). Double quotes may be used to enclose strings that contain spaces.

`colnames(`*string*`)` controls the names of the columns to be kept from matrix *matname*. It exits with an error if the column names of *matname* do not contain all the elements of *string* unless the `ignore` option is used (see `help matrix_subscripting`). Double quotes may be used to enclose strings that contain spaces.

`dropcolnames(`*string*`)` controls the names of the columns to be dropped from matrix *matname*. It exits with an error if the column names of *matname* do not contain all the elements of *string* unless `ignore` is used (see `help matrix_subscripting`). Double quotes may be used to enclose strings that contain spaces.

`rownum(`*numlist*`)` controls the numbers of the rows to be kept from matrix *matname*. It exits with an error if any of the numbers in *numlist* are larger than the row number of *matname* unless `ignore` is specified.

`droprownum(`*numlist*`)` controls the numbers of the rows to be dropped from matrix *matname*. It exits with an error if any of the numbers in *numlist* are larger than the row number of *matname* unless `ignore` is specified.

`colnum(`*numlist*`)` controls the numbers of the columns to be kept from matrix *matname*. It exits with an error if any of the numbers in *numlist* are larger than the column number of *matname* unless `ignore` is specified.

`dropcolnum(`*numlist*`)` controls the numbers of the columns to be dropped from matrix *matname*. It exits with an error if any of the numbers in *numlist* are larger than the column number of *matname* unless `ignore` is specified.

2.2 Other options

`rowvarlist` requests `submatrix` treat the names in `rownames()` and `droprownames()` as *varlist*. This option enables factor-variable expansion and the use of the `*` character for matching one or more characters.

`colvarlist` requests `submatrix` treat the names in `colnames()` and `dropcolnames()` as *varlist*. This option enables factor-variable expansion and the use of the `*` character for matching one or more characters.

`namesfirst` prioritizes subsetting based on `rownames()` and `colnames()` rather than using `rownum()` and `colnum()` first.

`ignore` requests that `submatrix` ignore any out-of-range element from *matname*. It affects `rownames()`, `droprownames()`, `colnames()`, `dropcolnames()`, `rownum()`, `droprownum()`, `colnum()`, and `dropcolnum()`. Using this option forces `submatrix` to return a result anyway.

2.3 Stored results

`submatrix` stores the following in `r()`:

Matrices
`r(mat)` subset of *matname* based on the subsetting options

3 Examples

This section overviews the use of `submatrix` in practice. The examples address the use of `submatrix` in a simulated context and in the frameworks of longitudinal data regression, intergenerational mobility, and discrete choice models.

3.1 Introductory example

This example illustrates the use of `submatrix` in the situation introduced in section 1. The aim is to extract nonconsecutive columns of a matrix (columns 1, 5, and 7). Using `submatrix`, there are three ways to isolate the target columns. The first one uses the option `colnum(1 5 7)`, the second drops irrelevant columns by using the option `dropcolnum(2(1)4 6)`, and the third targets the names of the columns of interest (option `colnames(c1 c5 c7)`).

```
. matrix A = (1,3,4,6,7,8,10 \ 1,3,4,6,7,8,10)
. submatrix A, colnum(1 5 7)
. matlist r(mat)
             |        c1         c5         c7
-------------+---------------------------------
          r1 |         1          7         10
          r2 |         1          7         10
. submatrix A, dropcolnum(2(1)4 6)
. matlist r(mat)
             |        c1         c5         c7
-------------+---------------------------------
          r1 |         1          7         10
          r2 |         1          7         10
. submatrix A, colnames(c1 c5 c7)
. matlist r(mat)
             |        c1         c5         c7
-------------+---------------------------------
          r1 |         1          7         10
          r2 |         1          7         10
```

3.2 Extracting submatrices from large matrices

Consider a situation in which a user is interested in a few elements of a large matrix with column (or row) names and, possibly, column (or row) equations. Furthermore, the matrix is large enough that the user does not precisely know the column numbers of certain elements. This may be the case in a regression with a large set of interactions and control variables (for example, an event study).

For instance, a slightly modified between-effects regression (see output below) from the examples in `help xtreg` would result in 250 estimated coefficients (and standard errors, p-values, etc.) stored in `r(table)`, a 9×250 matrix. Such an output may be overwhelming to read. In this section, `xtreg` is launched `quietly` to avoid excessive tables.

```
. webuse nlswork
(National Longitudinal Survey of Young Women, 14-24 years old in 1968)
. quietly xtreg ln_wage grade age c.age#c.age ttl_exp c.ttl_exp#c.ttl_exp tenure
> c.tenure#c.tenure 2.race not_smsa south i.year##(i.msp i.ind_code), be
. return list
matrices:
              r(table) :  9 x 250
```

Frequently, users are interested in a few coefficients. In keeping with the example above, a user aiming to visually inspect the subset of coefficients and p-values attached to `tenure`, `south`, and `i.msp#i.year` (interactions only) would extract the estimates after reading their names in `xtreg, coeflegend`. Because `tenure`, `south`, and `i.msp#i.year` are not consecutive in the specification of `xtreg` (and in `r(table)`), the user would run the `matlist` command three times. In the solution below, the multiple `matlist` calls contain the rows of `r(table)`, including the coefficients and p-values stacked columnwise. The p-values can be calculated from the `_b` and `_se` expressions, but this approach would take more time or would involve loops. The task can also be completed using Mata.

```
. quietly xtreg, coeflegend
. matlist ( r(table)["b", "tenure" .. "c.tenure#c.tenure"] \ r(table)["pvalue",
> "tenure" .. "c.tenure#c.tenure"] ) ', twidth(20)
                     |          b     pvalue
---------------------+----------------------
              tenure |   .0466337   2.40e-15
   c.tenure#c.tenure |  -.001749    7.95e-06
. matlist ( r(table)["b", "south"] \ r(table)["pvalue", "south"] )', twidth(20)
                     |          b     pvalue
---------------------+----------------------
               south |  -.0917312   5.16e-21
```

```
. matlist ( r(table)["b", "69.year#1.msp" .. "88.year#1.msp"] \
> r(table)["pvalue", "69.year#1.msp" .. "88.year#1.msp"] ) ', twidth(20)

                     |          b     pvalue
---------------------+----------------------
       69.year#1.msp |    .095698   .2939081
     70o.year#0b.msp |          0          .
       70.year#1.msp |  -.0551427   .5053928
     71o.year#0b.msp |          0          .
       71.year#1.msp |   .0236377    .753153
     72o.year#0b.msp |          0          .
       72.year#1.msp |  -.0217592   .8050339
     73o.year#0b.msp |          0          .
       73.year#1.msp |  -.0215855   .7933671
     75o.year#0b.msp |          0          .
       75.year#1.msp |  -.1747825   .0295381
     77o.year#0b.msp |          0          .
       77.year#1.msp |  -.1260273   .1385197
     78o.year#0b.msp |          0          .
       78.year#1.msp |   .0364713   .7034566
     80o.year#0b.msp |          0          .
       80.year#1.msp |  -.1614655   .1039684
     82o.year#0b.msp |          0          .
       82.year#1.msp |   .0579287    .558361
     83o.year#0b.msp |          0          .
       83.year#1.msp |  -.1123775   .2198865
     85o.year#0b.msp |          0          .
       85.year#1.msp |  -.1046524   .1894012
     87o.year#0b.msp |          0          .
       87.year#1.msp |  -.0228685   .7756839
     88o.year#0b.msp |          0          .
       88.year#1.msp |  -.0001004    .998787
```

In the last part of the output above, the base levels for the factor variables (that is, `70o.year#0b.msp`, `71o.year#0b.msp`, etc.) are shown in the matrix to avoid excessive coding. Removing the base levels prevents the column names of the selected variables of `r(table)` from being consecutive. To overcome this obstacle, users might use a loop or individually code each of the 14 calls to `matlist` related to the levels of year. The latter solution takes longer to code and is more difficult to troubleshoot. In the output below, the loop-based solution is displayed.

```
foreach t of numlist 69(1)73 75 77 78 80 82 83 85 87 88 {
    matrix target = nullmat(target)\ ( r(table)["b", "`t'.year#1.msp"]\ ///
        r(table)["pvalue", "`t'.year#1.msp"] )'
}
matlist target, twidth(20)
```

3.2.1 Solution using submatrix

The rows related to coefficients and p-values are listed in the `rownames()` option, and the columns related to variables (`tenure`, `south`, and `i.msp#i.year`) are parsed by the `colnames()` option. Additionally, the `colvarlist` option forces `submatrix` to treat the arguments of `colnames()` and `dropcolnames()` as a varlist. This instruction expands the factor variables in the interaction `i.year#1.msp`. Although the amount of code is not drastically smaller than in the other method, the output is more readable and compact.

```
. webuse nlswork
(National Longitudinal Survey of Young Women, 14-24 years old in 1968)
. quietly xtreg ln_wage grade age c.age#c.age ttl_exp c.ttl_exp#c.ttl_exp tenure
> c.tenure#c.tenure 2.race not_smsa south i.year##(i.msp i.ind_code), be
. matrix results = r(table)
. submatrix results, rownames(b pvalue) colnames(tenure c.tenure#c.tenure south
> i.year#1.msp) colvarlist ignore
. matlist r(mat)', twidth(20)
```

	b	pvalue
tenure	.0466337	2.40e-15
c.tenure#c.tenure	-.001749	7.95e-06
south	-.0917312	5.16e-21
69.year#1.msp	.095698	.2939081
70.year#1.msp	-.0551427	.5053928
71.year#1.msp	.0236377	.753153
72.year#1.msp	-.0217592	.8050339
73.year#1.msp	-.0215855	.7933671
75.year#1.msp	-.1747825	.0295381
77.year#1.msp	-.1260273	.1385197
78.year#1.msp	.0364713	.7034566
80.year#1.msp	-.1614655	.1039684
82.year#1.msp	.0579287	.558361
83.year#1.msp	-.1123775	.2198865
85.year#1.msp	-.1046524	.1894012
87.year#1.msp	-.0228685	.7756839
88.year#1.msp	-.0001004	.998787

3.3 Removing rows and columns from a matrix

This example requires the use of `igmobil` (Savegnago 2016), a community-contributed command that can be installed by executing the command `net install st0437`. Consider a situation in which a user is interested in most of the elements of a matrix stored in memory. Such a situation may arise in the field of intergenerational mobility, in which transition matrices are often used. In keeping with Savegnago (2016), I generate a dataset with two income variables across two generations (`dad` and `son`). I then use `igmobil` to calculate intergenerational mobility indicators and the transition matrix based on vigintiles of the income distributions of parents and children. This matrix provides the probability that a child's income falls in vigintile k given that his or her parental income was in vigintile j. The transition matrix is stored in the matrix `transition` (20×20), and I assume that the user is particularly interested in the transitions among the top five and bottom five vigintiles (the extremes) of the distribution. In other words, the aim is to study the switching probabilities from low-income classes (parent) to high-income classes (child) and vice versa. To obtain the desired matrix in Stata, the user must subset `transition` four times.

```
. matrix drop _all
. matrix C = (.25, .5*.25 \ .5*.25, .25)
. set seed 12345
. drawnorm u0 u1, n(2000) cov(C)
(obs 2,000)
. generate son = exp(u1)
. generate dad = exp(u0)
. drop u*
. matrix drop C
. quietly igmobil son dad, matrix(transition) classes(20)
     Single-stage Indices
    Transition matrix Indices (based on 20 quantiles)
    Inequality related Indices
. matrix dir
    transition[20,20]
. matrix transition2 = ( transition[1..5, 1..5], transition[1..5, 16..20] ) \
> ( transition[16..20, 1..5], transition[16..20, 16..20] )
. matlist transition2
             |        c1         c2         c3         c4         c5
-------------+-------------------------------------------------------
          r1 |       .23        .13        .09        .08        .12
          r2 |       .13        .14         .1        .04        .07
          r3 |       .16         .1        .12        .05        .09
          r4 |       .16         .1        .15        .11        .05
          r5 |       .04         .1        .07        .07        .08
         r16 |         0          0        .02        .02        .05
         r17 |         0        .02        .01        .04        .02
         r18 |         0        .02        .02        .01        .02
         r19 |         0        .01        .01          0        .02
         r20 |       .01          0          0        .01        .02
             |       c16        c17        c18        c19        c20
-------------+-------------------------------------------------------
          r1 |       .01          0          0          0        .01
          r2 |         0        .03        .03        .02        .01
          r3 |       .01        .02          0          0          0
          r4 |       .04        .01        .01        .02        .01
          r5 |       .04        .04        .03          0          0
         r16 |       .12        .04        .12        .08        .04
         r17 |       .11        .06        .07        .05        .09
         r18 |       .11        .11        .11        .11        .11
         r19 |       .05        .08        .13        .11        .14
         r20 |       .06        .06        .09        .14        .29
```

3.3.1 Solution using submatrix

The required submatrix can be obtained by deleting from `transition` the rows and columns from 6 to 15. This can be achieved by combining the `dropcolnum()` and `droprownum()` options. Compared with the native Stata solution, this strategy is more flexible because the extremes of the columns and rows to drop can be changed more easily. For example, a user wanting to study the four extreme vigintiles can simply change the *numlist* in `dropcolnum()` from `6(1)15` to `5(1)16`.

```
. submatrix transition, dropcolnum(6(1)15) droprownum(6(1)15)
. matlist r(mat)
```

	c1	c2	c3	c4	c5
r1	.23	.13	.09	.08	.12
r2	.13	.14	.1	.04	.07
r3	.16	.1	.12	.05	.09
r4	.16	.1	.15	.11	.05
r5	.04	.1	.07	.07	.08
r16	0	0	.02	.02	.05
r17	0	.02	.01	.04	.02
r18	0	.02	.02	.01	.02
r19	0	.01	.01	0	.02
r20	.01	0	0	.01	.02

	c16	c17	c18	c19	c20
r1	.01	0	0	0	.01
r2	0	.03	.03	.02	.01
r3	.01	.02	0	0	0
r4	.04	.01	.01	.02	.01
r5	.04	.04	.03	0	0
r16	.12	.04	.12	.08	.04
r17	.11	.06	.07	.05	.09
r18	.11	.11	.11	.11	.11
r19	.05	.08	.13	.11	.14
r20	.06	.06	.09	.14	.29

3.4 Selecting rows and removing columns at the same time

This example shows more complex subsetting of `transition` (introduced in section 3.3). The option `dropcolnum()` is used to remove the columns from 1 to 7 and from 14 to 20, while rows 5, 10, 15, and 20 are selected based on their names using the `rownames()` option. This situation assumes that the user is focused on the probability that a child belongs to the middle class (from the 8th to the 13th vigintile) conditional on a parent belonging to specific income quantiles.

```
. submatrix transition, dropcolnum(1(1)7 14(1)20) rownames(r5 r10 r15 r20)
. matlist r(mat)
```

	c8	c9	c10	c11	c12	c13
r5	.06	.08	.07	.04	.06	.05
r10	.09	.04	.03	.03	.03	.05
r15	.04	.06	.07	.06	.03	.08
r20	.01	.01	.03	.04	.04	.04

3.5 Extracting and re-sorting submatrices using equation names

The options `colnames()` and `rownames()` are used to subset matrix `b`, which is obtained from a multinomial logit model. In the framework of discrete choice models, users are often interested in coefficients (marginal utilities) related to single variables or to single choices. In this example, the categorical dependent variable is the insurance status (indemnity, prepaid, uninsured), and the covariates are age, sex, ethnicity, and site (see `help mlogit`). The base outcome is no insurance. The semicolon character is used in the `dropcolnames()` option to indicate the equation and retrieve all the elements whose status is not uninsured (the base outcome, for which all coefficients are zero by construction). The `colnames()` option is used to retrieve the coefficients of `age` and `male`. The resulting matrix is saved in `b1` and then re-sorted using `submatrix`.

```
. matrix drop _all
. webuse sysdsn1, clear
(Health insurance data)
. mlogit insure age male nonwhite i.site, base(3)
Iteration 0:  Log likelihood = -555.85446
Iteration 1:  Log likelihood = -534.67443
Iteration 2:  Log likelihood = -534.36284
Iteration 3:  Log likelihood = -534.36165
Iteration 4:  Log likelihood = -534.36165
Multinomial logistic regression                         Number of obs =    615
                                                        LR chi2(10)   =  42.99
                                                        Prob > chi2   = 0.0000
Log likelihood = -534.36165                             Pseudo R2     = 0.0387

------------------------------------------------------------------------------
      insure | Coefficient  Std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
Indemnity    |
         age |   .0077961   .0114418     0.68   0.496    -.0146294    .0302217
        male |  -.4518496   .3674867    -1.23   0.219     -1.17211     .268411
    nonwhite |  -.2170589   .4256361    -0.51   0.610     -1.05129    .6171725
             |
        site |
          2  |   1.211563   .4705127     2.57   0.010     .2893747    2.133751
          3  |   .2078123   .3662926     0.57   0.570     -.510108    .9257327
             |
       _cons |   1.286943   .5923219     2.17   0.030     .1260134    2.447872
-------------+----------------------------------------------------------------
Prepaid      |
         age |  -.0039489   .0115994    -0.34   0.734    -.0266832    .0187855
        male |   .1098438   .3651883     0.30   0.764    -.6059122    .8255998
    nonwhite |   .7577178   .4195759     1.81   0.071    -.0646357    1.580071
             |
        site |
          2  |   1.324599   .4697954     2.82   0.005     .4038165    2.245381
          3  |  -.3801756   .3728188    -1.02   0.308    -1.110887    .3505358
             |
       _cons |   1.556656   .5963286     2.61   0.009      .387873    2.725438
-------------+----------------------------------------------------------------
Uninsure     |  (base outcome)
------------------------------------------------------------------------------
. matrix b = e(b)
```

```
. submatrix b, dropcolnames("Uninsure:") colnames(age male)
. matlist r(mat)
             | Indemnity | Prepaid   | Indemnity | Prepaid
             |       age |       age |      male |      male
-------------+-----------+-----------+-----------+----------
          y1 |  .0077961 | -.0039489 | -.4518496 |  .1098438
. matrix b1 = r(mat)
. submatrix b1, colnames( Indemnity: Prepaid:)
. matlist r(mat)
             | Indemnity            | Prepaid
             |       age       male |       age       male
-------------+----------------------+---------------------
          y1 |  .0077961  -.4518496 | -.0039489   .1098438
```

4 Conclusion

The `submatrix` command can be used to subset existing matrices based on nonconsecutive elements. A useful development from StataCorp would be to incorporate permutation vectors and matrix subscripting based on multiple row (or column) names in the syntax of the `matrix` command. This could allow users to exclude the first column from a matrix `AA` by typing `matrix BB = AA[1 .. rows(AA), -1]` in the command window or by selecting nonconsecutive rows using row names with the command `matrix BB = AA[r1 r9, 1 .. rowsof(AA)]`.

5 Acknowledgments

I thank the editor, an anonymous referee, and Flavio Porta for helpful comments. Any remaining errors are mine.

6 Programs and supplemental material

To install the software files as they existed at the time of the publication of this article, type

```
. net sj 23-4
. net install st0077      (to install program files, if available)
. net get st0077          (to install ancillary files, if available)
```

7 References

Anselin, L. 1988. *Spatial Econometrics: Methods and Models*. Dordrecht: Kluwer. https://doi.org/10.1007/978-94-015-7799-1.

Cox, N. J. 2012. Speaking Stata: Matrices as look-up tables. *Stata Journal* 12: 748–758. https://doi.org/10.1177/1536867X1201200413.

Cunningham, S. 2021. *Causal Inference: The Mixtape*. New Haven, CT: Yale University Press. https://doi.org/10.12987/9780300255881.

Savegnago, M. 2016. igmobil: A command for intergenerational mobility analysis in Stata. *Stata Journal* 16: 386–402. https://doi.org/10.1177/1536867X1601600207.

Train, K. E. 2009. *Discrete Choice Methods with Simulation*. 2nd ed. Cambridge: Cambridge University Press. https://doi.org/10.1017/CBO9780511805271.

About the author

Daniele Spinelli is a postdoc research fellow at the Department of Statistics and Quantitative Methods at the University of Milano–Bicocca.