The Stata Journal (2010)
**10**, Number 1, pp. 125–142

# Mata Matters: Stata in Mata

William Gould
StataCorp
College Station, TX
wgould@stata.com

**Abstract.** Mata is Stata's matrix language. In the Mata Matters column, we show how Mata can be used interactively to solve problems and as a programming language to add new features to Stata. The subject of this column is using Mata to solve data analysis problems with the new Stata commands putmata and getmata, which were added to official Stata 11 in the update of 11 February 2010.

**Keywords:** pr0050, Mata, getmata, putmata

## 1 Introduction

Some problems are more easily solved in Mata than they are in Stata. The problem is that putting data from Stata to Mata and getting the result back again is difficult for casual users and tedious even for experienced users. The new Stata commands putmata and getmata solve that problem. These commands were added to official Stata 11 in the update of 11 February 2010.

With putmata, we can type

```
. putmata *
(12 vectors posted)
```

and thus create a column vector in Mata for each variable in our data. The vectors will have the same names as the variables. If we typed putmata * with the automobile data in memory, we would then have vectors named make, price, mpg, rep78, headroom, trunk, weight, length, turn, displacement, gear_ratio, and foreign available for use in Mata.

Note that you type putmata at the Stata dot prompt, not at the Mata colon prompt. Rather than typing putmata *, let's type

```
. putmata y=mpg X=(weight foreign 1)
(1 vector, 1 matrix posted)
```

Typing that creates a vector in Mata called y (which is just mpg, renamed) and a matrix called X (which contains the columns corresponding to weight, foreign, and a vector of 1s). We could then enter Mata and type

```
. mata
: b = invsym(X´X)*X´y
: yhat = X*b
: end
. _
```

Vector `yhat` now contains the predicted values from a regression of `y` on `X`. To post the Mata vector back into our Stata dataset, we could type

```
. getmata yhat
```

We would now have the new variable `yhat` in our Stata dataset.

The demonstration is intended to be motivational; I am not seriously suggesting you type the above instead of

```
. regress mpg weight foreign
  (output omitted)
. predict yhat
```

Nonetheless, the motivational example is the outline for what follows. We are going to discuss the details of `putmata` and `getmata`, we are going to use `putmata` as a jumping-off point to discuss writing Mata code to solve both statistical and data-management problems, and we are going to discuss how to package solutions in do-files.

Before we start, verify that you have the new commands `putmata` and `getmata`. In Stata, type

```
. help putmata
```

If you are told that help for `putmata` is not found, you need to update your Stata. You do that by typing

```
. update all
```

## 2 The putmata command

### 2.1 Syntax

The syntax of `putmata` is

`putmata` *putlist* [ *if* ] [ *in* ] [ `, replace` <u>`omit`</u>`missing view` ]

*putlist* can be any combination of the following:

*varname* or *varlist*
*vecname*=*varname*
*matname*=`(`*varlist*`)`
*matname*=`(`*varlist* # ...`)`

For example,

1. You can type `putmata mpg` to create in Mata the vector `mpg`.
2. You can type `putmata mpg weight` to create in Mata the vectors `mpg` and `weight`.

3. You can type `putmata *` to create in Mata vectors for every variable in the Stata dataset.

4. You can type `putmata y=mpg` to create Mata vector `y` containing the contents of Stata variable `mpg`.

5. You can type `putmata X=(weight foreign)` to create Mata matrix `X` containing `weight` in its first column and `foreign` in its second.

6. You can type `putmata X=(weight foreign 1)` to create Mata matrix `X` containing `weight` in its first column, `foreign` in its second, and constant `1` in its third.

You can even type `putmata y=mpg X=(weight foreign 1)` to perform the actions of examples 5 and 6 in a single line. If you specify the `omitmissing` option, however, it does matter whether you type separate or single commands, and you do *not* want to type separate commands:

```
. putmata y=mpg, omitmissing
. putmata X=(weight foreign 1), omitmissing
```

With the above commands, vector `y` will omit observations in which `mpg` contains missing. Matrix `X` will omit observations in which `weight` or `foreign` contain missing. What you want, however, is to omit observations from both `y` and `X` in which any of the variables contain missing. You want to type

```
. putmata y=mpg X=(weight foreign 1), omitmissing
```

## 2.2 Options

`replace` indicates that it is okay to replace an existing Mata vector or matrix. If you do not specify `replace` and the Mata vector or matrix already exists, `putmata` issues an error.

`omitmissing` specifies to omit observations that contain missing values in any of the variables in *putlist* from the rows of the vectors and matrices created in Mata. In the motivational example in section 1, we coded `b = invsym(X'X)*X'y`, and we created `y` and `X` by typing `putmata y=mpg X=(weight foreign 1)`. We just assumed there were no missing values. Had there been missing values, we would have wanted to create `y` and `X` by typing

```
. putmata y=mpg X=(weight foreign 1), omitmissing
```

`view` specifies that the vector and matrices be created as views onto the Stata data rather than as copies of the contents of the data. Views can save considerable amounts of memory and they have other advantages as well, although sometimes those advantages can turn into disadvantages. All of which is to say, views should be used with caution. We will discuss views later.

## 3 Using putmata to produce mathematical and statistical results

`putmata` is all you need to solve some problems. For instance, consider solving the set of linear equations $\mathbf{y} = \mathbf{X}\mathbf{b}$ for $\mathbf{b}$. The solution can be obtained by premultiplying both sides by $\mathbf{X}^{-1}$, which results in $\mathbf{X}^{-1}\mathbf{y} = \mathbf{X}^{-1}\mathbf{X}\mathbf{b}$ or $\mathbf{b} = \mathbf{X}^{-1}\mathbf{y}$. If you were teaching a course on linear algebra, you could demonstrate this solution. You might start with `y` and `x` values entered into a Stata dataset:

```
. list

     ┌─────────────────────┐
     │   y   x1   x2   x3  │
     ├─────────────────────┤
  1. │  27    2   -5    6  │
  2. │ -20    3    7   -9  │
  3. │  -9   -8    2    1  │
     └─────────────────────┘
```

You might type the following:

```
. putmata y X=(x*)
(1 vector, 1 matrix posted)
. mata
───────────────────────────────────────────────── mata (type end to exit) ─────
: y
          1
    ┌───────┐
  1 │   27  │
  2 │  -20  │
  3 │   -9  │
    └───────┘

: X
        1    2    3
    ┌───────────────┐
  1 │   2   -5    6 │
  2 │   3    7   -9 │
  3 │  -8    2    1 │
    └───────────────┘

: b = luinv(X)*y
: b
       1
    ┌─────┐
  1 │ 4   │
  2 │ 7   │
  3 │ 9   │
    └─────┘

: X*b
          1
    ┌───────┐
  1 │   27  │
  2 │  -20  │
  3 │   -9  │
    └───────┘

: end
───────────────────────────────────────────────────────────────────────────────
```

You can read the online help or the manual about the Mata function `luinv()`. I chose it because I needed a matrix inverter that could handle nonsymmetric matrices.

More interestingly, let's consider the overdetermined linear set of equations $\mathbf{y} = \mathbf{Xb}$ when $\mathbf{X}$ is $n \times k$, $n > k$. We have more equations than the unknown coefficients. Linear regression $\mathbf{b} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$ provides one solution. It turns out that $\mathbf{b} = \mathbf{X}^{-1}\mathbf{y}$ provides the same solution if you define $\mathbf{X}^{-1}$ to be the Moore–Penrose generalized inverse for nonsquare matrices! In the Moore–Penrose inverse, $\mathbf{X}^{-1}\mathbf{X}$ equals the identity matrix, but $\mathbf{XX}^{-1}$ does not. In any case, we can demonstrate the equivalence:

```
. sysuse auto
(1978 Automobile Data)

. putmata y=mpg X=(weight foreign 1)
(1 vector, 1 matrix posted)

. mata
──────────────────────────────────────────────────── mata (type end to exit) ─────
: pinv(X)*y
                  1
    ┌─────────────────┐
  1 │   -.0065878864  │
  2 │   -1.650029106  │
  3 │    41.67970233  │
    └─────────────────┘

: end
──────────────────────────────────────────────────────────────────────────────────
```

You could compare the above result with the coefficients reported by typing

```
. regress mpg weight foreign
```

or you could compare it with the Mata calculation of `invsym(X'X)*X'y`.

Mata is a great way to teach. Just as importantly, if you have a matrix calculation you need to make based on your data, you can use `putmata` to post the appropriate vector and matrices from your data and then use Mata to calculate the result.

If you are going to use Mata to make real statistical calculations, I recommend you normalize your data so that the variables are roughly scaled similarly because, when you write matrix formulas, you are not going to concern yourself with using variants that are more numerically accurate. For instance, Stata does not calculate linear regression using $(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$, although the calculation it makes is algebraically equivalent, which is to say, would yield the same results on an infinite-precision computer. The calculation Stata makes is more precise on finite-precision computers. Stata removes the means (and later solves for the intercept separately), and it uses a solver to obtain the coefficients, and more. The details are long and involved and the point is this: you are not going to invest that kind of effort. You are going to code $(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$ or whatever is the equivalent for your problem. There is nothing numerically wrong with using such formulas as long as you do not tax them by having variables that differ too wildly in scale.

The automobile data is an example of a dataset that is sufficiently scaled. In the automobile data, `mpg` varies between 12 and 41 (mean 21.3), `weight` varies between 1,760 and 4,840 (mean 3,019.5), and `foreign` is either 0 or 1 (mean 0.2973). Scaling that varies by a few orders of magnitude is usually of no concern. Let me show you, however, that results would be more accurate if we divided `weight` by 1,000 and `mpg` by 10.

It is a theoretical property of linear regression—and easy to prove—that the sum of the residuals will be zero when the coefficient vector **b** is set to the least-squares result. When we calculate the sum of those residuals using **b** obtained from any finite-precision calculation, however, the sum will not be precisely zero. Using the example above, if we use the **b** obtained by Stata's `regress` command, the sum is –5.1e–15 (meaning $-5.1 \times 10^{-15}$). If we use **b** = `pinv(X)*y`, the sum is −2.3e–13. If we use **b** = `invsym(X'X)*X'y`, the sum is 7.1e–13. Actually, I have made an adjustment to all those numbers, which I will explain, but these are the right numbers for comparison. If we rescaled the data by dividing `weight` by 1,000 and `mpg` by 10, the errors would be 3.22e–14 for `pinv(X)*y` and −6.48e–13 for `invsym(X'X)*X'y`, and unchanged for `regress`. The two matrix calculations are more accurate when made on better scaled data—the errors were closer to zero—and the results from `regress` remain unchanged. `regress` is robust to scaling.

In any case, all the errors are small. The maximum average error per observation was a mere 7.1e–13/74 = 9.6e–15 miles per gallon. Nonetheless, errors were smaller when we used scaled data.

I mentioned that I adjusted the errors reported above. I did that because when one calculates error on a finite-precision computer, one obtains the desired error plus the error in making the error calculation itself! Were you to calculate these sums of the residuals in the obvious way, which you could do using Mata and by typing `sum(y-X*b)`, you would obtain results different from what I reported. You would obtain –2.5e–13 for `b` obtained from `regress`, –2.5e–12 for `b` = `pinv(X)*y`, and –6.7e–12 for `b` = `invsym(X'X)*X'y`. Those error calculations include not just the error caused by numerical error in the calculation of **b** but also the numerical error in the calculation of `sum(y-X*b)`. Such unadjusted results are usually adequate for ranking techniques, and in some sense they are actually better because they also include the error in how you would be likely to use the calculated results. The results are adequate for ranking because, whatever is the error in `sum(y-X*b)`, it is a function of `y` and `X`, and you are using the same `y` and `X` in all three calculations, so the error is roughly held constant. I say roughly because the error in the error calculation is also a function of `b` and `b` is not being held constant; but it is precisely the effect of the different `b`'s that we want to evaluate, so we will have to accept some contamination in our results. The various `b` vectors are nearly identical anyway, so the variation in the contamination cannot be much.

I, however, want to compare results from unscaled and rescaled data, which is to say, the `y` and `X` that will be used in `sum(y-X*b)` will differ, and thus the error in calculating the error could differ across comparisons. To prevent that, after obtaining `b` from each method, I made the error calculations on the scaled data in all cases, which is to say, on the same `y` and `X`. Thus, when calculating errors for **b** calculated on unscaled data, I multiplied the calculated `weight` coefficient by 100 and divided the other calculated coefficients by 10 to put them on the scale for data that had `mpg` divided by 10 and `weight` divided by 1,000. Multiplication and division introduce no error on modern digital computers (proof omitted). That rescaling, however, allowed `sum(y-X*b)` to be calculated using the same `y` and `X` in all cases, and thus I held roughly constant the error in the error calculation. My adjustment also resulted in more accurate results, but that is for other reasons I am not going to explain here because it is not necessary that my results be more accurate. It is sufficient that I have held the error in the error calculation roughly constant.

By the way, I have still not told you what the true error is because I do not know it. To calculate the true error, I would have to calculate yet another rescaling that would minimize the error in the error calculation, and then I would report to you the error `y-sum(X*b)` calculated using that data, and I would add a plus-or-minus to the end of the reported result that represented the error in the error calculation itself.

All of which is a long way of saying that you should think about putting all your variables on roughly the same scale when you are not willing to think through the numerical issues.

## 4 Using putmata on subsets of observations

Assume we have the following code:

```
putmata y=mpg X=(weight length 1)
mata:
b = pinv(X)*y
b
end
```

where `b = pinv(X)*y` is standing in for some more complicated calculation you wish to make.

Say that we now wish to run this code on only the foreign cars in the data. We would modify the `putmata` command; the Mata code would remain unchanged:

```
putmata y=mpg X=(weight length 1) if foreign
mata:
b = pinv(X)*y
b
end
```

Whereas previously `y` would have been $74 \times 1$ and `X` would have been $74 \times 3$, now `y` will be $22 \times 1$ and `X` will be $22 \times 3$ because `foreign` is true in 22 observations in the data.

Say that we want to run on all our data, but this time, let's assume variables `mpg`, `weight`, and `length` have missing values. They do not in the automobile data, but we will imagine we are using some other dataset. The missing values in `y` and `X` will result in a $3 \times 1$ vector `b` containing missing values. If we want to run on only the complete observations, one solution would be

```
putmata  y=mpg  X=(weight length 1)  if mpg<. & weight<. & length<.
mata:
b = pinv(X)*y
b
end
```

An easier solution is

```
putmata  y=mpg  X=(weight length 1), omitmissing
mata:
b = pinv(X)*y
b
end
```

The `omitmissing` option omits observations with missing values in any of the variables to which we refer. If you specify `omitmissing`, it is important that you specify all the vectors and matrices you want to create with a single `putmata` command. If we typed `putmata y=mpg, omitmissing` and `putmata X=(weight length 1), omitmissing`, then vector `y` would omit observations for which `mpg` contains missing and `X` would omit observations for which `weight` or `length` contain missing, with the result that the `X` and `y` might not be conformable or, worse, be conformable but omit different observations.

## 5 The getmata command

### 5.1 Description

`getmata` is the reverse of `putmata`—it creates Stata variables from Mata vectors and matrices. In many cases, you will not need `getmata`. In the problems above, it was sufficient merely to report results. We used `putmata` to put our data into Mata, and we used Mata to calculate and display results. In other problems, you may create new vectors in Mata and need to put them back as variables in your data.

Here is a simplified version of a real problem that showed up on Statalist: You need to create new Stata variable $d$ from existing Stata variable $c$, to be defined as

$$d_i = \sum_{j|c_j>c_i} (c_j - c_i)$$

where $i$ and $j$ index observations. This problem can be solved in Stata, but it is easier to solve it in Mata because the Mata code we write is nearly identical to the mathematical statement of the problem. If `c` and `d` were Mata vectors, the code would be

```
d = J(rows(c), 1, 0)
for (i=1; i<=rows(c); i++) {
        for (j=1; j<=rows(c); j++) {
                if (c[j]>c[i]) d[i] = d[i] + (c[j] - c[i])
        }
}
```

The most difficult part of this solution to understand is the first line, `d = J(rows(c), 1, 0)`, and that is only because you may not be familiar with Mata's `J()` function. `d = J(rows(c), 1, 0)` creates a `rows(`*c*`)` × `1` column vector of `0`s. The arguments of `J()` are in just that order.

`c` is not a vector in Mata, however. We already know how to solve that:

```
. putmata c
```

It will hardly surprise you to learn that the way we get Mata vector `d` back into Stata afterward is

```
. getmata d
```

## 5.2 Syntax

Before we put all this together, let me describe the **getmata** command, the syntax of which is

`getmata` *getlist* [`, double` [<u>`up`</u>`date`|`replace`] `id(`*name*`) force`]

A *getlist* is much like a *putlist*, but reversed. A *getlist* can be any combination of the following:

*vecname*
*varname*=*vecname*
(*varname varname ... varname*)=*matname*
(*varname**)=*matname*

For example,

1. You can type `getmata x1` to create in Stata the new variable `x1` containing the contents of Mata vector `x1`.
2. You can type `getmata x1, update` to create or replace in Stata the variable `x1` containing the contents of Mata vector `x1`.
3. You can type `getmata x1 x2` to create in Stata the new variables `x1` and `x2` containing the contents of Mata vectors `x1` and `x2`.
4. You can type `getmata x1 x2, update` to create or replace in Stata the variables `x1` and `x2` containing the contents of Mata vectors `x1` and `x2`.
5. You can type `getmata (firstvar secondvar) = X` to create in Stata the new variables `firstvar` and `secondvar` containing the first and second columns of matrix `X`. `X` must be $N \times 2$. If `X` had three columns, then you would need to specify three variable names. Obviously, this construction can be used with the `update` option, as can all `getmata` constructions, so I will not mention it again.
6. You can type `getmata (myvar*) = X` to create in Stata the new variables `myvar1`, `myvar2`, ..., equal to the first, second, ..., columns of Mata matrix `X`.

## 5.3 Options

`double` creates new numeric variables as `double`s rather than the default `float`.

`update` or `replace` allows a vector to be placed in an existing variable. The two options have the same meaning unless the `id()` option is also specified.

`id(`*name*`)` is the topic of an entire section below.

`force` allows getting vectors that have fewer or more columns than observations in the data. You should never have to specify this option.

## 6 Using putmata and getmata

So now we can put together the solution of creating $d$ from $c$. To remind you, we wish to create new variable $d$ from existing variable $c$, where

$$d_i = \sum_{j|c_j>c_i} (c_j - c_i)$$

To show you that the solution works, I use a dataset containing the integers from 1 to 4. The solution is

```
. list

        ┌───┐
        │ c │
        ├───┤
    1.  │ 1 │
    2.  │ 2 │
    3.  │ 3 │
    4.  │ 4 │
        └───┘

. putmata c
(1 vector posted)

. mata
──────────────────────────────────────────────────── mata (type end to exit) ─────
: d = J(rows(c), 1, 0)

: for (i=1; i<=rows(c); i++) {
>         for (j=1; j<=rows(c); j++) {
>                 if (c[j]>c[i]) d[i] = d[i] + (c[j] - c[i])
>         }
> }

: end
──────────────────────────────────────────────────────────────────────────────────

. getmata d
. list

     ┌───────┐
     │ c   d │
     ├───────┤
  1. │ 1   6 │
  2. │ 2   3 │
  3. │ 3   1 │
  4. │ 4   0 │
     └───────┘
```

If I had to solve this problem, I would package my solution as a do-file.

```
——————————————————————————————————————————————————— begin myfile1.do ————————
version 11                                              // see note 1

clear mata                                              // see note 2
capture drop d                                          // see note 3

putmata c

mata:                                                   // see note 4
d = J(rows(c), 1, 0)
for (i=1; i<=rows(c); i++) {
        for (j=1; j<=rows(c); j++) {
                if (c[j]>c[i]) d[i] = d[i] + (c[j] - c[i])
        }
}
end

getmata d
————————————————————————————————————————————————————— end myfile1.do ————————
```

Notes:

1. Do-files should always begin with a `version` statement. That is what ensures that the do-file continues to work in years to come as new versions of Stata are released.

2. The do-file should not depend on Mata having certain vectors, matrices, or programs already loaded. To ensure this is true, we clear Mata.

3. It was easier for me to debug this do-file if I did not have to remember to `drop d` each time I reran it.

4. I coded `mata:` (`mata` with a colon), yet previously when I used Mata interactively, I omitted the colon. Coding `mata:` tells Mata to stop if any error occurs, which is exactly how I want my do-file to behave. Using `mata` without the colon tells Mata not to stop, but to instead give me an opportunity to fix what I mistyped, which is how I work interactively.

## 7 Using putmata and getmata on subsets of observations

In the example above where we created variable `d` from `c`, we assumed that there were no missing values in `c`, or at least we did not consider the issue. It turns out that our code produces several missing values in the presence of just one missing value. Below I have already dropped the data used in the previous example and have entered another dataset:

```
. list

     +---+
     | c |
     |---|
  1. | 1 |
  2. | 2 |
  3. | . |
  4. | 3 |
  5. | 4 |
     +---+

. do myfile1.do
  (output omitted)
. list

     +-------+
     | c   d |
     |-------|
  1. | 1   . |
  2. | 2   . |
  3. | .   0 |
  4. | 3   . |
  5. | 4   . |
     +-------+
```

We could modify the Mata code in `myfile1.do` to watch for missing values and to exclude them from the calculation, but we already know an easier way. Rather than creating Mata vector `c` to include all the observations from Stata variable `c`, we could create the vector to include only the nonmissing values by changing `putmata c` to read

```
putmata c if c<.
```

or

```
putmata c, omitmissing
```

The result of either of those commands will be to create vector `c` to be $4 \times 1$ rather than $5 \times 1$.

There is, however, an issue. At the end of our code where we post the Mata solution vector `d` to new Stata variable `d`—we coded `getmata d`—we will need to specify which five observations are to receive the four calculated results. `getmata` has a syntax for that, but before we can use it, we will need a variable that uniquely identifies the observations. In real data, you would be likely to already have such a variable, but in case you do not, it is easy to create such a variable. You type `generate` *newvar* `= _n`. Let's create such a variable in our data:

```
. generate fid = _n
. list

     ┌──────────┐
     │ c    fid │
     ├──────────┤
  1. │ 1      1 │
  2. │ 2      2 │
  3. │ .      3 │
  4. │ 3      4 │
  5. │ 4      5 │
     └──────────┘
```

`fid` is a perfectly good identification variable, but I am about to multiply `fid` by 10 just to emphasize to you that the identification variable does not have to correspond to observation numbers.

```
. replace fid = fid*10
(5 real changes made)
. list

     ┌──────────┐
     │ c    fid │
     ├──────────┤
  1. │ 1     10 │
  2. │ 2     20 │
  3. │ .     30 │
  4. │ 3     40 │
  5. │ 4     50 │
     └──────────┘
```

An identification variable is a variable that takes on different values for each observation in the data. The values could be 1, 2, ...; or they could be 1.25, –2, ...; or they could be Nick, Mary, and so on. The values can be numeric or string, and they can be in any order. All that is important is that the variable contain distinct values for each observation.

Now that we have an identification variable, we can modify the ending `getmata` command to read

```
getmata d, id(fid)
```

instead of just `getmata d`. The `id(fid)` option specifies that values in variable `fid` are to be matched with the values in vector `fid` to determine the observations of variable `d` that are to be filled in from vector `d`. For that to work, we must post to Mata the values of `fid`, so the entire solution reads

```
putmata fid c, omitmissing
mata:
Mata code goes here
end
getmata d, id(fid)
```

When we `putmata fid c, omitmissing` with our example data, two $4 \times 1$ vectors will be created in Mata, `fid` and `c`. The vectors will contain values from observations 1, 2, 4, and 5, omitting observation 3 because `c==.` in that observation. Thus vector `fid` will contain $(10, 20, 40, 50)'$. Later, at the end of our code, when we `getmata d, id(fid)`, Stata will compare the contents of vector `fid` $= (10, 20, 40, 50)'$ with the values of variable `fid`, and Stata will be able to work out that vector row 1 corresponds to observation 1, row 2 corresponds to observation 2, row 3 to observation 4, and row 4 to observation 5. In this example, `fid` increases with observation number, but that is not required.

Our updated do-file reads

begin `myfile2.do`

```
version 11

clear mata
capture drop d

putmata fid c, omitmissing                              // (changed)

mata:
d = J(rows(c), 1, 0)
for (i=1; i<=rows(c); i++) {
        for (j=1; j<=rows(c); j++) {
                if (c[j]>c[i]) d[i] = d[i] + (c[j] - c[i])
        }
}
end

getmata d, id(fid)                                      // (changed)
```

end `myfile2.do`

Here is the result of running the do-file:

```
. list

     ┌──────────┐
     │ c    fid │
     ├──────────┤
  1. │ 1     10 │
  2. │ 2     20 │
  3. │ .     30 │
  4. │ 3     40 │
  5. │ 4     50 │
     └──────────┘

. do myfile2
  (output omitted)

. list

     ┌──────────────┐
     │ c    fid   d │
     ├──────────────┤
  1. │ 1     10   6 │
  2. │ 2     20   3 │
  3. │ .     30   . │
  4. │ 3     40   1 │
  5. │ 4     50   0 │
     └──────────────┘
```

## 8 Using views

When you type or code `putmata x`, vector `x` is created as a copy of the Stata variable `x`. The variable and vector are separate things. An alternative is to make the Mata vector a view onto the Stata variable. You do that by typing `putmata x, view`. Now the variable and vector share the same recording of the values. Views use less memory than copies, although views are slightly less efficient in terms of execution time. Views have other advantages and disadvantages, too.

Say that you type `putmata x` and then, in Mata, code `x[1]=20`. Changing vector `x` leaves the variable `x` unchanged. If you had typed `putmata x, view`, however, changing vector `x` would simultaneously change variable `x`, because the variable and the vector are the same thing. Sometimes, that is an advantage. At other times, it is a disadvantage.

There is more to know. If you are working with views and, in the middle of the Mata session, take a break and return to Stata, it is important that you do not modify the Stata data in certain ways. When you create a view, Stata records notes about the mapping. Those notes might read that variable vector `x` is a view onto variable 3, observations 2 though 20 and observation 39. If you change the sort order of the data, the view will still be working with observations 2 through 20 and 39 even though those observations now contain different data! If you were to drop the first or second variable, the view would still be working with variable 3 even though that will now be a different variable! Alternatively, if you update variable 3 with improved values, those improvements will appear in the Mata vector, too.

The memory savings offered by views is considerable when working with large datasets. Say that you have a dataset containing 1,000,000 observations on 200 variables. That dataset might be 800,000,000 bytes in size, or 763 megabytes. (To obtain megabytes, you divide by $1{,}024^2$.) Typing `putmata *` would create copies of each variable, meaning creation of two hundred 1,000,000-element double-precision vectors. You would just have consumed another $200 \times 1{,}000{,}000 \times 8/1{,}024^2 = 1{,}526$ megabytes of memory, or $1{,}526/1{,}024 = 1.5$ gigabytes. Typing `putmata *, view`, however, would consume only 24 or so kilobytes of memory, a practically insignificant amount.

All the examples shown so far work equally well with copies or views. We simply would need to add the `view` option to the `putmata` commands.

If we are going to work with views, we could make `d` a view, too. If we make `d` a view, we can eliminate the `getmata` commands at the end of our code, because views are the variable and thus they put themselves back. This even means we could eliminate the `fid` variable because views will handle their own alignment of vectors and variables.

Remember that the do-file creates new variable `d` from existing variable `c`. We modify the do-file to create new variable `d` at the outset, in Stata, and then create views onto both `c` and `d`.

In the creation of those views, we can omit the observations that have `c>=.` by simply including the `omitmissing` option with `putmata`. Finally, we delete the now irrelevant `getmata` command at the end. Our code reads begin `myfile3.do`

```
version 11

clear mata
capture drop d
generate d = 0                                           // see note 1

putmata c d, omitmissing view                            // see note 2

mata:
d[.] = J(rows(c), 1, 0)                                  // see note 3
for (i=1; i<=rows(c); i++) {
        for (j=1; j<=rows(c); j++) {
                if (c[j]>c[i]) d[i] = d[i] + (c[j] - c[i])
        }
}

end

replace d=. if c==.                                      // see note 4
                                                         // see note 5
```

end `myfile3.do`

Notes:

1. We now create new variable `d` at the outset. We create it containing 0, not missing values. That is important because we are about to issue a `putmata` command with the `omitmissing` option, and we do not want the missing values in `d` to cause all the observations to be omitted.

2. We include the `view` option on the `putmata` command, and we include variable `d`.

3. We could have deleted this line, but instead I modified it to remind you not to make a terrible error. The line `d[.] = J(rows(c), 1, 0)` fills in `d` with zeros. I could have omitted the line because `d` is already filled with zeros. I did not delete it because I wanted an excuse to call your attention to the left-hand side of the assignment. I changed what was previously

   ```
   d = J(rows(c), 1, 0)
   ```

   to

   ```
   d[.] = J(rows(c), 1, 0)
   ```

   I changed `d` to `d[.]`. That change is of great importance. What we previously coded created vector `d`. What I now code changes the values stored in existing vector `d`. If I left what we coded previously, Mata would discard the view stored in `d` and create a new `d` as a regular Mata vector unconnected to Stata. Our Mata code would have worked, but none of the values stored in regular vector `d` would have made it back to Stata variable `d`.

4. We add the line `replace d=. if c<=.`. I admit that was something that I discovered I needed to add the first time I tested this do-file and looked at the output.

What I saw was that `d` = 0 in the observation in which `c==.`. That happened because we created `d` containing zeros at the outset. It would have been better if we had created `d` by coding

```
generate d = 0 if c<.
```

rather than `generate d = 0`. I left the mistake in, however, to show that the author is not infallible.

5. We omit the line `putmata d` or `putmata d, id(fid)`. Vector `d` is variable `d`. We need not worry about alignment because when the view `d` was created, it was created as a view onto only the relevant observations.

My personal opinion concerning views is that I avoid them for variables that appear on the left-hand side of the assignment operator. That is, I would have left `d` as a regular vector and left in the `getmata d, id(fid)`. If you review the above notes, all the complication was caused by `d` being a view. I had to remember to code `d[.] =` ... rather than `d =`, which I invariably forget. I cannot fill `d` with missing at the outset because `putmata, omitmissing` will then omit all the observations. Concerning the latter, there are more clever ways I could have handled that. I could have filled in `d` with 0 and performed the `putmata`, as I did, and then immediately changed the contents of `d` to be missing. Even so, I try to avoid using views for variables to which I will be making assignments. I do use views for right-hand-side variables because, in that case, views have no implications for subsequent code.

Anyway, this do-file works:

```
. list

     +---+
     | c |
     |---|
  1. | 1 |
  2. | 2 |
  3. | . |
  4. | 3 |
  5. | 4 |
     +---+

. do myfile3
  (output omitted)

. list

     +-------+
     | c   d |
     |-------|
  1. | 1   6 |
  2. | 2   3 |
  3. | .   . |
  4. | 3   1 |
  5. | 4   0 |
     +-------+
```

## 9 Conclusion

Some problems are more easily solved in Mata than in Stata. In fact, Mata and Stata complement each other well because problems that are easy in one are often difficult in the other. With `putmata`, it is easy to move your data into Mata. With `getmata`, you can move data back from Mata to Stata if necessary. I showed two classes of examples:

1. *Analysis.* In analysis situations, you use `putmata`, but you do not need `getmata`. I showed how to obtain $\mathbf{b} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$, and I showed that the same results could be obtained by $\mathbf{b} = \mathbf{X}^{-1}\mathbf{y}$ for a suitable definition of matrix inversion. Both of these examples would be useful in teaching, but you are to imagine that these simple formulas stand in for more lengthy calculations implementing the latest result found in the professional journals. I once gave a talk where I dropped into Mata to calculate a generalized method of moments estimator for a Poisson model with an endogenous variable, and I did so in a dozen or so lines of Mata code using the formulas right from the original paper. Stata now does generalized method of moments, so there is no reason to rehash an old talk here.

2. *Data management.* I showed how to create a difficult-to-calculate variable using Mata. Here you use `putmata` to get the data into Mata, and you use `getmata` to get the result back into Stata. Stata is wonderful at data management and most complicated tasks are made easy. Every so often, however, one comes upon a problem where the Stata solution is elusive. There is one, you know, and usually it requires only a few lines, but you cannot imagine what they might be. In such cases, it is usually quicker to drop into Mata and go directly at the solution.

`putmata` and `getmata` are useful commands, but bear in mind that they were designed to help solve custom data analysis problems: the types of problems that arise in a particular analysis and that one solves in do-files. They were not designed for use by programmers coding general solutions implemented as ado-files. `putmata` and `getmata` create and work with global vectors and matrices, and that is why their results are so easy to use. That same feature makes them inappropriate for ado-files. Programmers writing ado-files need results stored in local vectors and matrices. Stata already has tools for creating such local vectors and matrices, namely, `st_data()`, `st_view()`, and `st_store()`; see [M-5] **st_data( )** and [M-5] **st_view( )**. Programmers may wish to think of `putmata` as `st_data()` and `st_view()`, and `getmata` as `st_store()`, for interactive and do-file use.

### About the author

William Gould is president of StataCorp, head of development, and principal architect of Mata.