Title

stata.com

Interactive — Using Mata interactively

Description Remarks and examples Review Reference Also see

Description

With Mata, you simply type matrix formulas to obtain the desired results. Below we provide guidelines when doing this with statistical formulas.

Remarks and examples

stata.com

You have data and statistical formulas that you wish to calculate, such as $b = (X'X)^{-1}X'y$. Perform the following nine steps:

1. Start in Stata. Load the data.
2. If you are doing time-series analysis, generate new variables containing any *op*`.`*varname* variables you need, such as `l.gnp` or `d.r`.
3. Create a constant variable (`. generate cons = 1`). In most statistical formulas, you will find it useful.
4. Drop variables that you will not need. This saves memory and makes some things easier because you can just refer to all the variables.
5. Drop observations with missing values. Mata understands missing values, but Mata is a matrix language, not a statistical system, so Mata does not always ignore observations with missing values.
6. Put variables on roughly the same numeric scale. This is optional, but we recommend it. We explain what we mean and how to do this below.
7. Enter Mata. Do that by typing `mata` at the Stata command prompt. Do not type a colon after the `mata`. This way, when you make a mistake, you will stay in Mata.
8. Use Mata's `st_view()` function (see [M-5] **st_view()**) to create matrices based on your Stata dataset. Create all the matrices you want or find convenient. The matrices created by `st_view()` are in fact views onto one copy of the data.
9. Perform your matrix calculations.

If you are reading the entries in the order suggested in [M-0] **Intro**, see [M-1] **How** next.

1. Start in Stata; load the data

We will use the `auto` dataset and will fit the regression

$$\texttt{mpg}_j = b_0 + b_1\texttt{weight}_j + b_2\texttt{foreign}_j + e_j$$

by using the formulas

$$\mathbf{b} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$$
$$\mathbf{V} = s^2(\mathbf{X}'\mathbf{X})^{-1}$$

where

$$s^2 = \mathbf{e}'\mathbf{e}/(n-k)$$
$$\mathbf{e} = \mathbf{y} - \mathbf{X}\mathbf{b}$$
$$n = \text{rows}(\mathbf{X})$$
$$k = \text{cols}(\mathbf{X})$$

We begin by typing

```
. sysuse auto
(1978 automobile data)
```

2. Create any time-series variables

We do not have any time-series variables but, just for a minute, let's pretend we did. If our model contained lagged `gnp`, we would type

```
. generate lgnp = l.gnp
```

so that we would have a new variable `lgnp` that we would use in place of `l.gnp` in the subsequent steps.

3. Create a constant variable

```
. generate cons = 1
```

4. Drop unnecessary variables

We will need the variables `mpg`, `weight`, `foreign`, and `cons`, so it is easier for us to type `keep` instead of `drop`:

```
. keep mpg weight foreign cons
```

5. Drop observations with missing values

We do not have any missing values in our data, but let's pretend we did, or let's pretend we are uncertain. Here is an easy trick for getting rid of observations with missing values:

```
. regress mpg weight foreign cons
. keep if e(sample)
```

We estimated a regression using all the variables and then kept the observations `regress` chose to use. It does not matter which variable you choose as the dependent variable, nor the order of the independent variables, so we just as well could have typed

```
. regress weight mpg foreign cons
. keep if e(sample)
```

or even

```
. regress cons mpg weight foreign
. keep if e(sample)
```

The output produced by `regress` is irrelevant, even if some variables are dropped. We are merely borrowing `regress`'s ability to identify the subsample with no missing values.

Using `regress` causes Stata to make many unnecessary calculations and, if that offends you, here is a more sophisticated alternative:

```
. local 0 "mpg weight foreign cons"
. syntax varlist
. marksample touse
. keep if `touse'
. drop `touse'
```

Using `regress` is easier.

6. Put variables on roughly the same numeric scale

This step is optional, but we recommend it. You are about to use formulas that have been derived by people who assumed that the usual rules of arithmetic hold, such as $(a + b) - c = a + (b - c)$. Many of the standard rules, such as the one shown, are violated when arithmetic is performed in finite precision, and this leads to roundoff error in the final, calculated results.

You can obtain a lot of protection by making sure that your variables are on roughly the same scale, by which we mean their means and standard deviations are all roughly equal. By roughly equal, we mean equal up to a factor of 1,000 or so. So let's look at our data:

```
. summarize
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
         mpg |         74     21.2973    5.785503        12         41
      weight |         74    3019.459    777.1936      1760       4840
     foreign |         74    .2972973    .4601885         0          1
        cons |         74           1           0         1          1
```

Nothing we see here bothers us much. Variable `weight` is the largest, with a mean and standard deviation that are 1,000 times larger than those of the smallest variable, `foreign`. We would feel comfortable, but only barely, ignoring scale differences. If `weight` were 10 times larger, we would begin to be concerned, and our concern would grow as `weight` grew.

The easiest way to address our concern is to divide `weight` so that, rather than measuring weight in pounds, it measures weight in thousands of pounds:

```
. replace weight = weight/1000
variable weight was int now float
(74 real changes made)
. summarize
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
         mpg |         74     21.2973    5.785503        12         41
      weight |         74    3.019459    .7771936      1.76       4.84
     foreign |         74    .2972973    .4601885         0          1
        cons |         74           1           0         1          1
```

What you are supposed to do is make the means and standard deviations of the variables roughly equal. If `weight` had a large mean and reasonable standard deviation, we would have subtracted, so that we would have had a variable measuring weight in excess of some number of pounds. Or we could do both, subtracting, say, 2,000 and then dividing by 100, so we would have weight in excess of 2,000 pounds, measured in 100-pound units.

Remember, the definition of roughly equal allows lots of leeway, so you do not have to give up easy interpretation.

7. Enter Mata

We type

```
. mata
------------------------------------------------- mata (type end to exit) -----
: _
```

Mata uses a colon prompt, whereas Stata uses a period.

8. Use Mata's st_view() function to access your data

Our matrix formulas are

$$\mathbf{b} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$$
$$\mathbf{V} = s^2(\mathbf{X}'\mathbf{X})^{-1}$$

where

$$s^2 = \mathbf{e}'\mathbf{e}/(n-k)$$
$$\mathbf{e} = \mathbf{y} - \mathbf{X}\mathbf{b}$$
$$n = \text{rows}(\mathbf{X})$$
$$k = \text{cols}(\mathbf{X})$$

so we are going to need $\mathbf{y}$ and $\mathbf{X}$. $\mathbf{y}$ is an $n \times 1$ column vector of dependent-variable values, and $\mathbf{X}$ is an $n \times k$ matrix of the k independent variables, including the constant. Rows are observations, columns are variables.

We make the vector and matrix as follows:

```
: st_view(y=., ., "mpg")
: st_view(X=., ., ("weight", "foreign", "cons"))
```

Let us explain. We wish we could type

```
: y = st_view(., "mpg")
: X = st_view(., ("weight", "foreign", "cons"))
```

because that is what the functions are really doing. We cannot because `st_view()` (unlike all other Mata functions), returns a special kind of matrix called a view. A view acts like a regular matrix in nearly every respect, but views do not consume nearly as much memory, because they are in fact views onto the underlying Stata dataset!

We could instead create `y` and `X` with Mata's `st_data()` function (see [M-5] **st_data()**), and then we could type the creation of `y` and `X` the natural way,

```
: y = st_data(., "mpg")
: X = st_data(., ("weight", "foreign", "cons"))
```

st_data() returns a real matrix, which is a copy of the data Stata has stored in memory.

We could use st_data() and be done with the problem. For our automobile-data example, that would be a fine solution. But were the automobile data larger, you might run short of memory, and views can save lots of memory. You can create views willy-nilly—lots and lots of them—and never consume much memory! Views are wonderfully convenient and it is worth mastering the little bit of syntax to use them.

st_view() requires three arguments: the name of the view matrix to be created, the observations (rows) the matrix is to contain, and the variables (columns). If we wanted to create a view matrix Z containing all the observations and all the variables, we could type

```
: st_view(Z, ., .)
```

st_view() understands missing value in the second and third positions to mean all the observations and all the variables. Let's try it:

```
: st_view(Z, ., .)
                  <istmt>:  3499  Z not found
r(3499);
: _
```

That did not work because Mata requires Z to be predefined. The reasons are technical, but it should not surprise you that function arguments need to be defined before a function can be used. Mata just does not understand that st_view() really does not need Z defined. The way around Mata's confusion is to define Z and then let st_view() redefine it:

```
: Z = .
: st_view(Z, ., .)
```

You can, if you wish, combine all that into one statement

```
: st_view(Z=., ., .)
```

and that is what we did when we defined y and X:

```
: st_view(y=., ., "mpg")
: st_view(X=., ., ("weight", "foreign", "cons"))
```

The second argument (.) specified that we wanted all the observations, and the third argument specified the variables we wanted. Be careful not to omit the "extra" parentheses when typing the variables. Were you to type

```
: st_view(X=., ., "weight", "foreign", "cons")
```

you would be told you typed an invalid expression. st_view() expects three arguments, and the third argument is a row vector specifying the variables to be selected: ("weight", "foreign", "cons").

At this point, we suggest you type

```
: y
  (output omitted)
: X
  (output omitted)
```

to see that y and X really do contain our data. In case you have lost track of what we have typed, here is our complete session so far:

```
. sysuse auto
. generate cons = 1
. keep mpg weight foreign cons
. regress mpg weight foreign cons
. keep if e(sample)
. replace weight = weight/1000
. mata
: st_view(y=., ., "mpg")
: st_view(X=., ., ("weight", "foreign", "cons"))
```

9. Perform your matrix calculations

To remind you: our matrix calculations are

$$\mathbf{b} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$$
$$\mathbf{V} = s^2(\mathbf{X}'\mathbf{X})^{-1}$$

where

$$s^2 = \mathbf{e}'\mathbf{e}/(n-k)$$
$$\mathbf{e} = \mathbf{y} - \mathbf{X}\mathbf{b}$$
$$n = \text{rows}(\mathbf{X})$$
$$k = \text{cols}(\mathbf{X})$$

Let's get our regression coefficients,

```
: b = invsym(X'X)*X'y
: b
                  1
  +---------------+
1 |  -6.587886358 |
2 |  -1.650029004 |
3 |   41.67970227 |
  +---------------+
```

and let's form the residuals, define n and k, and obtain s^2,

```
: e  = y - X*b
: n  = rows(X)
: k  = cols(X)
: s2 = (e'e)/(n-k)
```

so we are able to calculate the variance matrix:

```
: V = s2*invsym(X'X)
: V
[symmetric]
                  1              2              3
  +-----------------------------------------------+
1 |   .4059128628                                 |
2 |   .4064025078   1.157763273                   |
3 |  -1.346459802   -1.57131579   4.689594304     |
  +-----------------------------------------------+
```

We are done.

We can present the results in more readable fashion by pulling the diagonal of V and calculating the square root of each element:

```
: se = sqrt(diagonal(V))
: (b, se)
                 1              2
  +-------------------------------+
1 |  -6.587886358    .6371129122  |
2 |  -1.650029004    1.075994086  |
3 |   41.67970227    2.165547114  |
  +-------------------------------+
```

You know that if we were to type

```
: 2+3
  5
```

Mata evaluates the expression and shows us the result, and that is exactly what happened when we typed

```
: (b, se)
```

`(b, se)` is an expression, and Mata evaluated it and displayed the result. The expression means to form the matrix whose first column is `b` and second column is `se`. We could obtain a listing of the coefficient, standard error, and its t statistic by asking Mata to display `(b, se, b:/se)`,

```
: (b, se, b:/se)
                 1              2              3
  +-----------------------------------------------+
1 |  -6.587886358    .6371129122  -10.34021793  |
2 |  -1.650029004    1.075994086  -1.533492633  |
3 |   41.67970227    2.165547114   19.24673077  |
  +-----------------------------------------------+
```

In the expression above, `b:/se` means to divide the elements of `b` by the elements of `se`. `:/` is called a colon operator and you can learn more about it by seeing [M-2] **op_colon**.

We could add the significance level by typing

```
: (b, se, b:/se, 2*ttail(n-k, abs(b:/se)))
                 1              2              3              4
  +-------------------------------------------------------------+
1 |  -6.587886358    .6371129122  -10.34021793   8.28286e-16  |
2 |  -1.650029004    1.075994086  -1.533492633   .1295987129  |
3 |   41.67970227    2.165547114   19.24673077   6.89556e-30  |
  +-------------------------------------------------------------+
```

Those are the same results reported by `regress`; type

```
. sysuse auto
. replace weight = weight/1000
. regress mpg weight foreign
```

and compare results.

Review

Our complete session was

```
. sysuse auto
. generate cons = 1
. keep mpg weight foreign cons
. regress mpg weight foreign cons
. keep if e(sample)
. replace weight = weight/1000
. mata
: st_view(y=., ., "mpg")
: st_view(X=., ., ("weight", "foreign", "cons"))
: b = invsym(X'X)*X'y
: b
: e = y - X*b
: n = rows(X)
: k = cols(X)
: s2= (e'e)/(n-k)
: V = s2*invsym(X'X)
: V
: se = sqrt(diagonal(V))
: (b, se)
: (b, se, b:/se)
: (b, se, b:/se, 2*ttail(n-k, abs(b:/se)))
: end
```

Reference

Gould, W. W. 2006. Mata Matters: Interactive use. *Stata Journal* 6: 387–396.

Also see

[M-1] **Intro** — Introduction and advice