Programming with Stata

Automate your work, share your routines with colleagues, generate reproducible research, document your research project, or provide the Stata user community with new commands. Stata has everything you need to store and execute a sequence of commands, program your own command with all the features of an official Stata command, or write routines and commands using Stata's matrix programming language, Mata. And with PyStata, you get comprehensive Python integration—harness the power of Python from your Stata code, and harness the power of Stata from your Python code. Stata also lets you incorporate C, C++, and Java plugins. You can even embed Java code directly in your Stata code!

Automating your work using a do-file

You can automate your work using a do-file, which is a text file that contains a set of Stata commands.

For reproducible analyses, store all your commands in a do-file. For instance, you might fit a logit model of employment status on variables representing education level and participation in a training program:

```
logit employed education i.training
```

You then compute the population-averaged treatment effect of the training program:

```
margins r.training
```

You then create a classification table comparing observed with predicted employment status:

```
estat classification
```

Save these commands in your do file, add **version 19.5** to the top of the file, and rerun them at any time to reproduce your results.

Sometimes, your analyses require real programming features. You can incorporate those into your do-file as well. You might loop over a variable list:

```
foreach variable of varlist myvars {
        your code
}
```

You might do computations by groups:

```
by group: mycommand
```

Or you might create and use local macros, the variables of Stata programs:

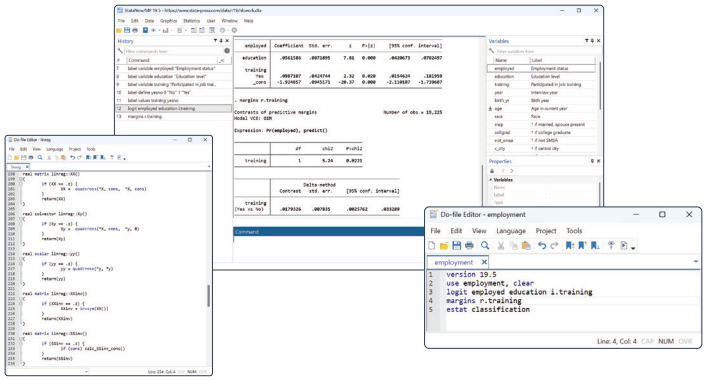
```
local myregressors x1 x2 x3 x4 x5
probit y `myregressors'
logit y `myregressors'
```

You can also access the results produced by an estimation command and use them in subsequent commands. Say you fit a model and want to rescale your coefficient vector. The coefficients are stored in **e(b)**, and you can create a matrix of rescaled coefficients by typing

```
matrix A = 3*e(b)
```

Once you have everything in your do-file, easily rerun all of these commands by typing

```
do mydofile
```



Write your own command using an ado-file

Anyone can write a Stata command using an ado-file. This extensibility allows everyone to share their programs with a worldwide community.

To write a command, you first need to determine the mandatory and optional arguments a user needs to type and parse them. This is easily done with Stata's **syntax** command. **syntax** examines what is typed and matches it to the accepted syntax of the command. The individual components are stored in local macros, where you can access them.

Say we are writing a maximum likelihood estimator that is not currently available in Stata and that allows time-series operators, factor variables, **if** and **in** conditions to restrict our estimation sample, frequency weights, and all of Stata's variance–covariance estimators. We type

```
syntax varlist(ts fv) [if] [in] [fweights],
       vce(passthru)]*
```

After your **syntax** command, how do you write your maximum likelihood estimator? Easy:

```
mlexp (mylikelihood) ..., 'vce' ...
```

You just need to write the expression for mylikelihood, and **mlexp** does the rest for you. It will compute robust, cluster–robust, jackknife, and bootstrap standard errors and allow weights and much more.

Want your output to look like that of a Stata command? Easy. Post your results using **ereturn post**, and use **ereturn display**.

```
ereturn post b V ...
ereturn display, ...
```

You have a Stata table that takes Stata's display and format options. Now anyone can type

```
mycommand y x1 x2 x3, options
```

Adding a matrix programming component

You can incorporate the results of matrix manipulations using Mata, Stata's matrix language, into your commands. Mata has the advanced matrix operations you need: access to the power of LAPACK and built-in solvers and optimizers to make implementing your own maximum likelihood, GMM, or other estimators easier. Mata makes the possibilities of Stata programming boundless.

If you want to enter Mata, type

```
mata
```

Get your data into a matrix with name **D**:

```
D = st_data(.,.)
```

Or get only your dependent variable **y** and your regressors **x1** and **x2**:

```
y = st_data(., "y")
X = st_data(., "x1 x2")
```

Compute the product **X'X** and then its Cholesky decomposition:

```
XX = quadcross(X,X)
A = cholesky(XX)
```

Compute $\mathbf{(X'X)^{-1}}$ and regression coefficients:

```
INVXX = invsym(quadcross(X,X))
betamata = quadcross(INVXX, quadcross(X,y))
```

Pass your regression coefficients stored in **betamata** from Mata to Stata with the name **betastata**:

```
st_matrix("betastata", betamata)
```

Close your Mata session:

```
end
```

This is just a glimpse of all you can do when programming with Stata and Mata. To learn more about programming, please visit **stata.com/programming-features**.