An Introduction to Stata Programming

Second Edition

CHRISTOPHER F. BAUM
Department of Economics and School of Social Work
Boston College

A Stata Press Publication
StataCorp LP
College Station, Texas

Published by Stata Press, 4905 Lakeway Drive, College Station, Texas 77845
Typeset in LaTeX 2ε
Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

ISBN-10: 1-59718-150-1
ISBN-13: 978-1-59718-150-1

Library of Congress Control Number: 2015955595

(Pages omitted)

Contents

(Pages omitted)

Preface

This book is a concise introduction to the art of Stata programming. It covers three types of programming that can be used in working with Stata: do-file programming, ado-file programming, and Mata functions that work in conjunction with do- and ado-files. Its emphasis is on the automation of your work with Stata and how programming on one or more of these levels can help you use Stata more effectively.

In the development of these concepts, I do not assume that you have prior experience with Stata programming, although familiarity with the command-line interface is helpful. While examples are drawn from several disciplines, my background as an applied econometrician is evident in the selection of some sample problems. The introductory first chapter motivates the why: why should you invest time and effort into learning Stata programming? In chapter 2, I discuss elementary concepts of the command-line interface and describe some commonly used tools for working with programs and datasets.

The format of the book may be unfamiliar to readers who have some familiarity with other books that help you learn how to use Stata. Beginning with chapter 4, each even-numbered chapter is a "cookbook" chapter containing several "recipes", 47 in total. Each recipe poses a problem: how can I perform a certain task with Stata programming? The recipe then provides a complete worked solution to the problem and describes how the features presented in the previous chapter can be put to good use. You may not want to follow a recipe exactly from the cookbook; just as in cuisine, a minor variation on the recipe may meet your needs, or the techniques presented in that recipe may help you see how Stata programming applies to your specific problem.

Most Stata users who delve into programming use do-files to automate and document their work. Consequently, the major focus of the book is do-file programming, covered in chapters 3, 5, 7, and 9. Some users will find that writing formal Stata programs, or ado-files, meets their needs. Chapter 11 is a concise summary of ado-file programming, with the cookbook chapter that follows presenting several recipes that contain developed ado-files. Stata's matrix programming language, Mata, can also be helpful in automating certain tasks. Chapter 13 presents a summary of Mata concepts and the key features that allow interchange of variables, scalars, macros, and matrices. The last chapter, cookbook chapter 14, presents several examples of Mata functions developed to work with ado-files. All the do-files, ado-files, Mata functions, and datasets used in the book's examples and recipes are available from the Stata Press website, as discussed in *Notation and typography*.

The second edition of this book contains several new recipes illustrating how do-files, ado-files, and Mata functions can be used to solve programming problems. Several recipes have also been updated to reflect new features in Stata added between versions 10 and 14. The discussion of maximum-likelihood function evaluators has been significantly expanded in this edition. The new topics covered in this edition include factor variables and operators; use of `margins`, `marginsplot`, and `suest`; Mata-based likelihood function evaluators; and associative arrays.

(Pages omitted)

1 Why should you become a Stata programmer?

This book provides an introduction to several contexts of Stata programming. I must first define what I mean by "programming". You can consider yourself a Stata programmer if you write do-files, which are text files of sequences of Stata commands that you can execute with the `do` ([R] **do**) command, by double-clicking on the file, or by running them in the Do-file Editor ([R] **doedit**). You might also write what Stata formally defines as a program, which is a set of Stata commands that includes the `program` ([P] **program**) command. A Stata program, stored in an ado-file, defines a new Stata command. You can also use Stata's matrix programming language, Mata, to write routines in that language that are called by ado-files. Any of these tasks involves Stata programming.[1]

With that set of definitions in mind, we must deal with the why: why should you become a Stata programmer? After answering that essential question, this text takes up the how: how you can become a more efficient user of Stata by using programming techniques, be they simple or complex.

Using any computer program or language is all about efficiency—getting the computer to do the work that can be routinely automated, reducing human errors, and allowing you to more efficiently use your time. Computers are excellent at performing repetitive tasks; humans are not. One of the strongest rationales for learning how to use programming techniques in Stata is the potential to shift more of the repetitive burden of data management, statistical analysis, and production of graphics to the computer. Let's consider several specific advantages of using Stata programming techniques in the three contexts listed above.

1. There are also specialized forms of Stata programming, such as dialog programming, scheme programming, and class programming. A user-written program can present a dialog, like any official Stata command, if its author writes a dialog file. The command can also be added to the User menu of Stata's graphical interface. For more information, see [P] **dialog programming** and [P] **window programming**. Graphics users can write their own schemes to set graphic defaults. See [G-4] **schemes intro** for details. Class programming allows you to write object-oriented programs in Stata. As [P] **class** indicates, this has primarily been used in Stata's graphics subsystem and graphical user interface. I do not consider these specialized forms of programming in this book.

Do-file programming

Using a do-file to automate a specific data-management or statistical task leads to reproducible research and the ability to document the empirical research process. This reduces the effort needed to perform a similar task at a later point or to document for your coworkers or supervisor the specific steps you followed. Ideally, your entire research project should be defined by a set of do-files that execute every step, from the input of the raw data to the production of the final tables and graphs. Because a do-file can call another do-file (and so on), a hierarchy of do-files can be used to handle a complex project.

The beauty of this approach is its flexibility. If you find an error in an earlier stage of the project, you need only to modify the code and then rerun that do-file and those following to bring the project up to date. For instance, a researcher may need to respond to a review of her paper—submitted months ago to an academic journal—by revising the specification of variables in a set of estimated models and estimating new statistical results. If all the steps that produce the final results are documented by a set of do-files, her task is straightforward. I argue that all serious users of Stata should gain some facility with do-files and the Stata commands that support repetitive use of commands.

That advice does not imply that Stata's interactive capabilities should be shunned. Stata is a powerful and effective tool for exploratory data analysis and ad hoc queries about your data. But data-management tasks and the statistical analyses leading to tabulated results should not be performed with "point-and-click" tools that leave you without an audit trail of the steps you have taken.

Ado-file programming

On a second level, you may find that despite the breadth of Stata's official and user-written commands, there are tasks you must repeatedly perform that involve variations on the same do-file. You would like Stata to have a command to perform those tasks. At that point, you should consider Stata's ado-file programming capabilities. Stata has great flexibility: a Stata command need be no more than a few lines of Stata code. Once defined, that command becomes a "first-class citizen". You can easily write a Stata program, stored in an ado-file, that handles all the features of official Stata commands such as `if` *exp*, `in` *range*, and command options. You can (and should) write a help file that documents the program's operation for your benefit and for those with whom you share the code. Although ado-file programming requires that you learn how to use some additional commands used in that context, it can help you become more efficient in performing the data-management, statistical, or graphical tasks that you face.

Mata programming for ado-files

On a third level, your ado-files can perform some complicated tasks that involve many invocations of the same commands. Stata's ado-file language is easy to read and write,

but it is interpreted. Stata must evaluate each statement and translate it into machine code. The Mata programming language (`help mata`) creates compiled code, which can run much faster than ado-file code. Your ado-file can call a Mata routine to carry out a computationally intensive task and return the results in the form of Stata variables, scalars, or matrices. Although you may think of Mata solely as a matrix language, it is actually a general-purpose programming language, suitable for many nonmatrix-oriented tasks, such as text processing and list management.

The level of Stata programming that you choose to attain and master depends on your needs and skills. As I have argued, the vast majority of interactive Stata users can and should take the next step of learning how to use do-files efficiently to take full advantage of Stata's capabilities and to save time. A few hours of investment in understanding the rudiments of do-file programming—as covered in the chapters to follow—will save you days or weeks over the course of a sizable research project.

A smaller fraction of users may choose to develop ado-files. Many users find that those features lacking in official Stata are adequately provided by the work of members of the Stata user community who have developed and documented ado-files, sharing them via the *Stata Journal*, the Statistical Software Components (SSC) archive,[2] or their own user site. However, developing a reading knowledge of ado-file code is highly useful for many Stata users. It permits you to scrutinize ado-file code—either that of official Stata or user-written code—and more fully understand how it performs its function. In many cases, minor modifications to existing code may meet your needs.

Mata has been embraced by programmers wishing to take advantage of its many features and its speed. Although this book does not discuss interactive use of Mata, I present two ways in which Mata can be used in ado-files: in "one-liners" to fulfill a single, specific task, and as functions to be called from ado-files.

1.1 Plan of the book

The chapters of this book present the details of the three types of Stata programming discussed above, placing the greatest emphasis on effective use of do-file programming. Each fairly brief chapter on the structure of programming techniques is followed by a "cookbook" chapter. These chapters contain several "recipes" for the solution of a particular, commonly encountered problem, illustrating the necessary programming techniques to compose a solution. Like in a literal cookbook, the recipes here are illustrative examples; you are free to modify the ingredients to produce a somewhat different dish. The recipes as presented may not address your precise problem, but they should prove helpful in devising a solution as a variation on the same theme.

2. For details on the SSC (Boston College) archive of user-contributed routines, type `help ssc`.

(Pages omitted)

4 Cookbook: Do-file programming I

This cookbook chapter presents for Stata do-file programmers several recipes using the programming features described in the previous chapter. Each recipe poses a problem and a worked solution. Although you may not encounter this precise problem, you should be able to recognize its similarities to a task that you would like to automate in a do-file.

4.1 Tabulating a logical condition across a set of variables

The problem.

When considering many related variables, you want to determine whether, for each observation, all variables satisfy a logical condition. Alternatively, you might want to know whether any satisfy that condition (for instance, taking on inappropriate values), or you might want to count how many of the variables satisfy the logical condition.[1]

The solution.

This would seem to be a natural application of `egen` ([D] **egen**), because that command already contains many rowwise functions to perform computations across variables. For instance, the `anycount()` function counts the number of variables in its varlist whose values for each observation match those of an integer numlist, whereas the `rowmiss()` and `rownonmiss()` functions tabulate the number of missing and nonmissing values for each observation, respectively. The three tasks above are all satisfied by `egen` functions from Nicholas Cox's `egenmore` package: `rall()`, `rany()`, and `rcount()`, respectively. Why not use those functions, then?

Two reasons come to mind: First, recall that `egen` functions are interpreted code. Unlike the built-in functions accessed by `generate`, the logic of an `egen` function must be interpreted each time it is called. For a large dataset, the time penalty can be significant. Second, to use an `egen` function, you must remember that there is such a function, and you must remember its name. In addition to Stata's official `egen` functions, documented in the online help files, there are many user-written `egen` functions available, but you must track them down.

For these reasons, current good programming practice suggests that you should avoid `egen` function calls in instances where the performance penalty might be an issue. This

1. This recipe relies heavily on Nicholas J. Cox's `egenmore` help file.