

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



Copyright © 2009, 2016 by StataCorp LP
All rights reserved. First edition 2009
Second edition 2016

Published by Stata Press, 4905 Lakeway Drive, College Station, Texas 77845

Typeset in L^AT_EX 2_ε

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

No part of this book may be reproduced, stored in a retrieval system, or transcribed, in any form or by any means—electronic, mechanical, photocopy, recording, or otherwise—without the prior written permission of StataCorp LP.

Stata, **stata**, Stata Press, Mata, **mata**, and NetCourse are registered trademarks of StataCorp LP.

Stata and Stata Press are registered trademarks with the World Intellectual Property Organization of the United Nations.

L^AT_EX 2_ε is a trademark of the American Mathematical Society.

(Pages omitted)

Contents

	List of figures	xvii
	List of tables	xix
	Preface	xxi
	Acknowledgments	xxiii
	Notation and typography	xxv
1	Why should you become a Stata programmer?	1
	Do-file programming	2
	Ado-file programming	2
	Mata programming for ado-files	2
	1.1 Plan of the book	3
	1.2 Installing the necessary software	4
2	Some elementary concepts and tools	5
	2.1 Introduction	5
	2.1.1 What you should learn from this chapter	5
	2.2 Navigational and organizational issues	5
	2.2.1 The current working directory and profile.do	6
	2.2.2 Locating important directories: sysdir and adopath	6
	2.2.3 Organization of do-files, ado-files, and data files	7
	2.3 Editing Stata do- and ado-files	8
	2.4 Data types	9
	2.4.1 Storing data efficiently: The compress command	11
	2.4.2 Date and time handling	11
	2.4.3 Time-series operators	13
	2.4.4 Factor variables and operators	14

2.5	Handling errors: The capture command	16
2.6	Protecting the data in memory: The preserve and restore commands	17
2.7	Getting your data into Stata	18
2.7.1	Inputting and importing data	18
	Handling text files	19
	Free format versus fixed format	20
	The import delimited command	21
	Accessing data stored in spreadsheets	23
	Fixed-format data files	24
2.7.2	Importing data from other package formats	29
2.8	Guidelines for Stata do-file programming style	30
2.8.1	Basic guidelines for do-file writers	31
2.8.2	Enhancing speed and efficiency	33
2.9	How to seek help for Stata programming	33
3	Do-file programming: Functions, macros, scalars, and matrices	37
3.1	Introduction	37
3.1.1	What you should learn from this chapter	37
3.2	Some general programming details	38
3.2.1	The varlist	39
3.2.2	The numlist	39
3.2.3	The if exp and in range qualifiers	39
3.2.4	Missing-data handling	40
	Recoding missing values: The mvdecode and mvencode commands	41
3.2.5	String-to-numeric conversion and vice versa	42
	Numeric-to-string conversion	43
	Working with quoted strings	44
3.3	Functions for the generate command	44
3.3.1	Using if exp with indicator variables	47
3.3.2	The cond() function	49

<i>Contents</i>	ix
3.3.3	Recoding discrete and continuous variables 49
3.4	Functions for the egen command 51
	Official egen functions 52
	egen functions from the user community 53
3.5	Computation for by-groups 54
3.5.1	Observation numbering: <code>_n</code> and <code>_N</code> 55
3.6	Local macros 57
3.7	Global macros 60
3.8	Extended macro functions and macro list functions 60
3.8.1	System parameters, settings, and constants: <code>creturn</code> 62
3.9	Scalars 62
3.10	Matrices 64
4	Cookbook: Do-file programming I 67
4.1	Tabulating a logical condition across a set of variables 67
4.2	Computing summary statistics over groups 69
4.3	Computing the extreme values of a sequence 70
4.4	Computing the length of spells 71
4.5	Summarizing group characteristics over observations 76
4.6	Using global macros to set up your environment 78
4.7	List manipulation with extended macro functions 79
4.8	Using <code>creturn</code> values to document your work 81
5	Do-file programming: Validation, results, and data management 83
5.1	Introduction 83
5.1.1	What you should learn from this chapter 83
5.2	Data validation: The <code>assert</code> , <code>count</code> , and <code>duplicates</code> commands 83
5.3	Reusing computed results: The <code>return</code> and <code>ereturn</code> commands 90
5.3.1	The <code>ereturn list</code> command 94
5.4	Storing, saving, and using estimated results 97
5.4.1	Generating publication-quality tables from stored estimates 102
5.5	Reorganizing datasets with the <code>reshape</code> command 104

5.6	Combining datasets	109
5.7	Combining datasets with the append command	111
5.8	Combining datasets with the merge command	113
5.8.1	The one-to-one match-merge	115
5.8.2	The dangers of many-to-many merges	116
5.9	Other data management commands	117
5.9.1	The fillin command	117
5.9.2	The cross command	117
5.9.3	The stack command	118
5.9.4	The separate command	119
5.9.5	The joinby command	120
5.9.6	The xpose command	121
6	Cookbook: Do-file programming II	123
6.1	Efficiently defining group characteristics and subsets	123
6.1.1	Using a complicated criterion to select a subset of observations	124
6.2	Applying reshape repeatedly	125
6.3	Handling time-series data effectively	129
6.3.1	Working with a business-daily calendar	132
6.4	reshape to perform rowwise computation	133
6.5	Adding computed statistics to presentation-quality tables	136
6.6	Presenting marginal effects rather than coefficients	138
6.6.1	Graphing marginal effects with marginsplot	140
6.7	Generating time-series data at a lower frequency	141
6.8	Using suest and gsem to compare estimates from nonoverlapping samples	146
6.9	Using reshape to produce forecasts from a VAR or VECM	149
6.10	Working with IRF files	152
7	Do-file programming: Prefixes, loops, and lists	157
7.1	Introduction	157
7.1.1	What you should learn from this chapter	157

7.2	Prefix commands	157
7.2.1	The by prefix	158
7.2.2	The statsby prefix	160
7.2.3	The xi prefix and factor-variable notation	161
7.2.4	The rolling prefix	162
7.2.5	The simulate and permute prefixes	164
7.2.6	The bootstrap and jackknife prefixes	167
7.2.7	Other prefix commands	169
7.3	The forvalues and foreach commands	169
8	Cookbook: Do-file programming III	177
8.1	Handling parallel lists	177
8.2	Calculating moving-window summary statistics	178
8.2.1	Producing summary statistics with rolling and merge	180
8.2.2	Calculating moving-window correlations	181
8.3	Computing monthly statistics from daily data	182
8.4	Requiring at least n observations per panel unit	184
8.5	Counting the number of distinct values per individual	185
8.6	Importing multiple spreadsheet pages	186
9	Do-file programming: Other topics	189
9.1	Introduction	189
9.1.1	What you should learn from this chapter	189
9.2	Storing results in Stata matrices	189
9.3	The post and postfile commands	193
9.4	Output: The export delimited, outfile, and file commands	196
9.5	Automating estimation output	199
9.6	Automating graphics	203
9.7	Characteristics	207
10	Cookbook: Do-file programming IV	211
10.1	Computing firm-level correlations with multiple indices	211
10.2	Computing marginal effects for graphical presentation	214

10.3	Automating the production of L ^A T _E X tables	216
10.4	Extracting data from graph files' sersets	220
10.5	Constructing continuous price and returns series	225
11	Ado-file programming	231
11.1	Introduction	231
11.1.1	What you should learn from this chapter	232
11.2	The structure of a Stata program	232
11.3	The program statement	233
11.4	The syntax and return statements	234
11.5	Implementing program options	237
11.6	Including a subset of observations	238
11.7	Generalizing the command to handle multiple variables	240
11.8	Making commands byable	242
	Program properties	243
11.9	Documenting your program	244
11.10	egen function programs	246
11.11	Writing an e-class program	248
11.11.1	Defining subprograms	250
11.12	Certifying your program	250
11.13	Programs for ml, nl, and nlsur	252
	Maximum likelihood estimation of distributions' parameters	255
11.13.1	Writing an ml-based command	260
11.13.2	Programs for the nl and nlsur commands	263
11.14	Programs for gmm	265
11.15	Programs for the simulate, bootstrap, and jackknife prefixes	270
11.16	Guidelines for Stata ado-file programming style	272
11.16.1	Presentation	273
11.16.2	Helpful Stata features	274
11.16.3	Respect for datasets	274
11.16.4	Speed and efficiency	275

<i>Contents</i>	xiii
11.16.5 Reminders	275
11.16.6 Style in the large	276
11.16.7 Use the best tools	276
12 Cookbook: Ado-file programming	277
12.1 Retrieving results from rolling	277
12.2 Generalization of egen function pct9010() to support all pairs of quantiles	280
12.3 Constructing a certification script	282
12.4 Using the ml command to estimate means and variances	287
12.4.1 Applying equality constraints in ml estimation	289
12.5 Applying inequality constraints in ml estimation	291
12.6 Generating a dataset containing the longest spell	294
12.7 Using suest on a fixed-effects model	297
13 Mata functions for do-file and ado-file programming	301
13.1 Mata: First principles	301
13.1.1 What you should learn from this chapter	302
13.2 Mata fundamentals	302
13.2.1 Operators	303
13.2.2 Relational and logical operators	304
13.2.3 Subscripts	305
13.2.4 Populating matrix elements	305
13.2.5 Mata loop commands	307
13.2.6 Conditional statements	308
13.3 Mata's st_ interface functions	309
13.3.1 Data access	309
13.3.2 Access to locals, globals, scalars, and matrices	311
13.3.3 Access to Stata variables' attributes	312
13.4 Calling Mata with a single command line	312
13.5 Components of a Mata function	316
13.5.1 Arguments	316

13.5.2	Variables	317
13.5.3	Stored results	317
13.6	Calling Mata functions	318
13.7	Example: <code>st_</code> interface function usage	320
13.8	Example: Matrix operations	322
13.8.1	Extending the command	327
13.9	Mata-based likelihood function evaluators	329
13.10	Creating arrays of temporary objects with pointers	331
13.11	Structures	334
13.12	Additional Mata features	337
13.12.1	Macros in Mata functions	337
13.12.2	Associative arrays in Mata functions	338
13.12.3	Compiling Mata functions	340
13.12.4	Building and maintaining an object library	341
13.12.5	A useful collection of Mata routines	342
14	Cookbook: Mata function programming	343
14.1	Reversing the rows or columns of a Stata matrix	343
14.2	Shuffling the elements of a string variable	346
14.3	Firm-level correlations with multiple indices with Mata	348
14.4	Passing a function to a Mata function	353
14.5	Using subviews in Mata	356
14.6	Storing and retrieving country-level data with Mata structures	358
14.7	Locating nearest neighbors with Mata	363
14.8	Using a permutation vector to reorder results	368
14.9	Producing \LaTeX tables from <code>svy</code> results	370
14.10	Computing marginal effects for quantile regression	375
14.11	Computing the seemingly unrelated regression estimator	379
14.12	A GMM-CUE estimator using Mata's <code>optimize()</code> functions	384

Contents

xv

References

397

Author index

403

Subject index

405

(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.