How to write a Stata program

Ben Jann

University of Bern, Institute of Sociology

Presentation at GESIS in Mannheim
October 30, 2014

Lars's questions I

- How/when do you decide whether a script should be turned into an ado-file?
- Which programming style best suits requirements of documentation?
- What are the things you have to keep in mind from the outset so that a script can become a module that is understandable for others? Or do you simply solve a problem first, and then worry about generalization etc. later on?
- Did you know in advance which ado-files will be successful or is this more like a lottery?
- How can you collaborate with others on a script? Does Github etc. help or is it better to program parts alone?
- What are the rough steps one has to take? Is there a review procedure / code review?

Lars's questions II

- How do you ensure the quality of your scripts?
- What are the obligations, e.g. in terms of maintenance, you commit yourself to by going public with a program?
- How long does it take from a existing script that solves a specific problem to a generalizes module?
- Is there a book that describes the approach or is this all self-taught based on Stata manuals?
- How do you deal with ownership/copyright e.g. if code is based on someone else's ideas, the code is the result of a mandate, or several people worked on a script? That is, how does everything work with acknowledgments and authorships?

How/when do you decide whether a script should be turned into an ado-file?

- It depends.
- `alphawgt`, `cprplot2`, ...: Modification of official ado file
- `fre`, `center`, `estout`, `coefplot`, ...: "This is an annoying task I have to do all the time! I really need a program for that to make my life easier. And hey, others might be interested too."
- `oaxaca`, `fairlie`, `kdens`, `mgof`, `rrlogit`, ...: Commands that came out of applied work.
- ...
- Often I work on a problem and then write a somewhat generalized program so that I do not have to copy similar code over and over again. Often this is just a number of `programs` at the header of a do-file. Depending on topic and if I think that others might be interested, I then turn some of that into generalized ados including documentation.

Which programming style best suits requirements of documentation?

- Don't know.
- Some of my guidelines:
  - Structure the code; e.g. use indentation for loops and such; use gaps between different parts etc.
  - Use more-or-less systematic and speaking names for locals and other objects in programs so that things are less confusing. At the same time, try to keep the names short.
  - Do not include too much documentation in the code. Try to use clear and clean code that speaks for itself. However:
    - ★ Use headers between sections of code that say what the following section will do.
    - ★ Use self-explaining names for subprograms and functions.
    - ★ Add small notes in places where a trick or so is used that is not obvious or the code is misleading.
  - Order code sections and subprograms/functions in a sensible way so that things are easier to understand.

What are the things you have to keep in mind from the outset so that a script can become a module that is understandable for others? Or do you simply solve a problem first, and then worry about generalization etc. later on?

- The latter, usually, see below

Did you know in advance which ado-files will be successful or is this more like a lottery?

- Well, I was pretty confident that programs such as `estout` or `center` will be popular. But I also wrote programs that – surprisingly for me – did not catch on (e.g. `adolist`).
- A good name and description do help. Promotion on statalist does help. Writing a Stata Journal article does help.

How can you collaborate with others on a script? Does Github etc. help or is it better to program parts alone?

- Cannot say much about that ...
- I don't use Github or so, too complicated for me.
- I'm a control freak. I don't like others messing around in my code.
- I use manual versioning, employing a small script that generates a dated zip-file. Each time I share code or make code public, I run the script to preserve the state of the program at that point in time.
- If I collaborate on a program with other people there is usually a clear division of labor. E.g. one is the boss who maintains the overall program, the other provides code bits and suggestions for improvement, but does mess around in the main code without flagging all changes.

What are the rough steps one has to take? Is there a review procedure / code review?

1. Write an adofile and a help file. Make sure to use file names that do not yet exist on SSC.
2. Send a zip file to Kit Baum for posting on SSC. Need to provide a title and a description.
3. Send a message to statalist announcing the program.
4. There is no code review for SSC programs. Kit does not check the code. This is your responsibility.
5. Possibly: Write a paper for Stata Journal (or, e.g., Journal of Statistical Software); SJ papers are peer reviewed. Depending on reviewer this includes code review.

How do you ensure the quality of your scripts?

- Well . . .
- One could write certification scripts (see `help cscript`). There's also a paper by Bill Gould on this stuff in the Stata Journal.
  - Gould, William (2001): Statistical software certification. The Stata Journal 1(1):29–50.
- Certification is extremely important at Stata Corp. For each line of code they have like 2 lines of certification (or maybe the other way round, can't remember).
- I rarely use certification scripts. Too clumsy, too time consuming, not enough incentive.
- I do check each addition I make to a program (i.e. confirm that it does what it should do; compare to results from other programs/software etc.), but I do not usually re-check all previous additions/changes.

How do you ensure the quality of your scripts?

- This is potentially dangerous as new changes might corrupt existing things.
- However, modularized programming helps a lot to prevent such problems (i.e. have isolated subroutines for different tasks).
- Furthermore, it is usually quite predictable which existing things might be affected, so they can be checked.
- Another good idea is to write robust code that complains if things are not as they should be (i.e., in a way, include certification directly in the code). This means that there will be `confirms` and `asserts` etc. in the code.
- In addition, I usually perform some robustness tests where I try to break the program (i.e. apply the program to very odd or corrupted data, use weird combinations of options, etc.).

What are the obligations, e.g. in terms of maintenance, you commit yourself to by going public with a program?

- Lots of emails.
- Bugs should be fixed.
- Sometimes: revise program so it takes account of new Stata features.
- In general: no obligation to revise and further develop programs once they are published.

How long does it take from a existing script that solves a specific problem to a generalized module?

- Depends on problem.
- `addplot`: Last August, I had some discussions with Vince Wiggins on some graph problem in the morning. In the afternoon I wrote the program, the helpfile and the description for SSC.
- `coefplot`: In a talk at the Stata Meeting in Potsdam on June 7, 2013, I presented a first crude version of the program (after investing about one day of programming). I then devoted probably about half of my total working time to the program until first release on SSC on August 25, 2013.

How long does it take from a existing script that solves a specific problem to a generalized module?

- Generalizing a script can take some time, but you get fast with experience. A lot is houskeeping that is similar for most programs.
- Once you have a decent version of a program, you probably need the same amount of time for documentation and cleaning up. But it is hard to say, because programming and documenting is usually an iterative process.

Is there a book that describes the approach or is this all self-taught based on Stata manuals?

- Me personally: self-taught, Stata manuals
- My advice to you:
  - Look at the manuals. They are great.
    - ★ In particular [U] **18 Programming Stata**, [P] **Programming**, and [M] **Mata Reference Manual**
  - For the basics:
    - ★ take Net Course 151
    - ★ have a look at: Baum, C. F. (2009). An Introduction to Stata Programming. College Station, TX: Stata Press.
  - A great reference for writing estimation programs is:
    - ★ Gould, W., J. Pitblado, W. Sribney (2010). Maximum Likelihood Estimation with Stata (4th ed). College Station, TX: Stata Press.
  - Search statalist www.stata.com/statalist/archive/ www.statalist.org/forums/
  - Look at official ados (`viewsource xyz.ado`). Use them as blueprints.
  - For useful programming tools, have a look at `help undocumented`.

How do you deal with ownership/copyright e.g. if code is based on someone else's ideas, the code is the result of a mandate, or several people worked on a script? That is, how does everything work with acknowledgments and authorships?

- User ados published at SSC are public domain. You can do with them whatever you want.
- Of course, it is good practice to acknowledge work/ideas by others. Include references in the documentation or in the code (or both).
- Collaborations: Include everyone as an author.
- Mandates: Whatever you agree on.
- If you revise/extend a program written by someone else:
  - Either agree on coauthorship with original author ...
  - or publish a spin-off under a new name and include a reference to the original program.

Some of my personal programming principles I

- Never change the data unless this is the purpose of the program and never leave anything behind that is not meant to be left behind.
  - Use `tempname`, `tempvar`, `tempfile`.
  - Use `sortpreserve` and `preserve` if necessary.
  - For programs that are supposed to change the data: Make sure that the data remains unchanged if the program fails (this includes the user pressing the break key).
- Avoid globals as much as possible.
  - If you cannot avoid them, use funny names (e.g. with a prefix) and make sure to clean up (see above).
- Do things as similar as possible as is done by Stata Corp.
  - Use their programs as blueprints.
  - Use standard solutions for standard tasks (e.g. for results display).
  - Stick to the standards with respect to syntax, housekeeping, results display, returns, and documentation as much as possible.

Some of my personal programming principles II

- Modularize complex programs and avoid redundancies to keep programs clean and robust, avoid confusion, and make programs memory efficient.
- Avoid dirty tricks and odd code as much as possible to improve robustness and readability. Try to be as general as possible.
- Always use compound double quotes (`‘"..."’`) for strings (unless it is guaranteed that a string cannot contain funny characters).
- Use declarations in Mata and set `matastrict` on.
- Keep a log file of changes and make regular backups (both dated).
- Use an environment that allows a smooth workflow. Templates can also be useful.
- Probably most important and most difficult: Think about the user.

Let's write a program

- I do inequality research and want produce graphs like that: http://www.youtube.com/watch?v=slTF_XXoKAQ.
- First step: write a script that does the job.
- Second step: I like it and want to produce a lot of such graphs ⇒ wrap it up as a basic program.
- Third step: I figure that others might be interested to, e.g. my collaborators or other people ⇒ write a general adofile and some documentation and post the program.
  - Possible additions
    - ★ weights
    - ★ return results in `r()` or `e()` and allow replay with different graph options
    - ★ support for bootstrap to compute confidence intervals
    - ★ results by subpopulations (implement a `by()` option)