Stata for Programmers: Basics and Topics*

– *To Be Completed* –

Johannes Fleck

Spring 2019

This guide has two objectives. First, it aims to assist users of numerical computing languages such as Matlab and Julia to understand Stata code. Second, it seeks to enable them to implement procedures they are familiar with from their native languages. For both objectives, the guide assumes a basic prior exposure to Stata and focuses on pointing out its specific features.
The structure of the guide is as follows: The basics section provides an overview on selected Stata commands. Their selection was guided by the intent to identify those genuine to Stata. The topics section covers a number of procedures, mostly related to data import and export, saving figures as well as debugging code.
The information presented is derived from the official Stata manual, various user forum posts and other online sources. The links included in the text serve as starting points to a more thorough investigation of specific details. The commands included in this guide have been executed in Stata 15.1 (MP). Backward compatibility has not been assessed.

*First version: July 3, 2018. I am grateful to the European Central Banks's Household Finance and Consumption Group for providing me with a productive and supportive environment. I thank Miguel Ampudia, Giuseppe Floccari, Jirka Slacalek and Marco Felici for their answers to my questions and for sharing tips and tricks. Pascal Achard was especially helpful in answering numerous questions on specific expert level questions. All errors are mine. E-mail me for comments and suggestions: Johannes.Fleck@eui.eu

Contents

Tell me and I forget. Teach me and I may remember. Involve me and I learn.
Benjamin Franklin

1 Introduction

Stata is a collection of commands. There is no general principle behind these commands. Many functions take very specific input structures and, compared to other languages, there are fewer generalizable principles. Implies that becoming proficient means to have a lot of experience with certain functions.

Means that if we think difference between expert and beginner is how they deal with bugs pattern will look different for Stata: in other languages, can address bugs by smart guesses. In stata, need to reach out and tackle idiosyncratic features of the bug faced with. Learn to love bug reports makes learning in othe languages possible. Main problem in Stata is that bug reports can be very uninformative.

Also: most important escape characters in Julia are and .

What kind of data formats does Stata use? Compare matlab: strings, double, cells etc. In Stata, float, byte, int...

Note: Can save same variables in different formats. Example: ctrylist can be saved as str2 and byte. But will show up in different colors in workspace.

Stata is hard to learn. There are specific functions instead of generalizible principles. For example logical indexing.

some of it is presented in pseudo code

Is Stata a language? When was it developed? How many users does it have? R, SPSS, ... Is it fast? What are its strong and weak points?

Stata is serving the social science crowd which can make it confusing to programmers and users of general purpose languages. "Most of Stata's quirks are necessary in order to make it so user-friendly to social scientists." "In a lot of ways R is a more conventional language than Stata, but most social scientists find Stata much easier to learn. In part because Stata is willing to deviate from the conventions of general purpose programming languages, running and interpreting a regression in Stata looks like this "reg y x" instead of this "summary(lm(y x))" and loading a dataset looks like this "use mydata, clear" instead of this "data ¡- read.table(mydata.txt)". Stata has some pretty complicated syntax (e.g., the entire Mata language) but you can get a lot done with just a handful of simple commands like "use," "gen," and "reg"."

"Nonetheless all this means that when Stata native speakers like me learn a second programming language it can be a bit confusing"

Some peculiar featurs:

Do-files and Ado-files. In any other language a do-file would be called a script and an ado-file would be called a library. Also note that Stata very conveniently reads all your ado-files automatically, whereas most other languages require you to specifically load the relevant libraries into memory at the beginning of each script.

Commands, Programs, and Functions. In Stata a program is basically just a command that you wrote yourself. Stata is somewhat unusual in drawing a distinction between a command/program and a function. So in Stata a function usually means some kind of transformation that attaches its output to a variable or macro, as in "gen $\ln_income = log(income)$". In contrast a command/program is pretty much anything that doesn't directly attach to an operat
$-lm(y\ x)$". This is because \regress" in Stata is a command whereas \lm()" in R is a function. Also note that Stat

The Dataset. Stata is one of the only languages where it's appropriate to use the definite article in reference to data. (NetLogo is arguably another case of this). In other languages it's more appropriate to speak of "a data object" than "the dataset," even if there only happens to be one data object in memory. For the same reason, most languages don't "use" or "open" data, but "read" the data and assign it to an object. Another way to think about it is that only Stata has a "dataset" whereas other languages only have "matrices." Of course, Stata/Mata also has matrices but most Stata end users don't bother with them as they tend to be kind of a backend thing that's usually handled by ado-files. Furthermore, in other languages (e.g., Perl) it's common to not even load a file into memory but to process it line-by-line, which in Stata terms is kind of like a cross between the "file read/write" syntax and a "while" loop.

Variables. Stata uses the term "variable" in the statistical or social scientific meaning of the term. In other languages this would usually be called a field or vector.

Macros. What most other languages call variables, Stata calls local and global "macros." Stata's usage of the local vs global distinction is standard. In other languages the concept of "declaring" a variable is usually a little more explicit than it is in Stata.

Stata is extremely good about expanding macros in situ and this can spoil us Stata users. In other languages you often have to do some kind of crude work around by first using some kind of concatenate function to create a string object containing the expansion and then you use that string object.

Reporting. Stata allows you to pass estimations on for further work (that's what return macros, ereturn matrices, and postestimation commands are all about), but it assumes you probably won't and so it is unusually generous in reporting most of the really interesting things after a command. In other languages you usually have to specifically ask to get this level of reporting. Another way to put it is that in Stata verbosity is assumed by default and can be suppressed with "quietly," whereas in R silence is assumed by default and verbosity can be invoked by wrapping the estimation (or an object saving the estimation) in the "summary()" function.

`https://www.stata.com/statalist/archive/2008-08/msg01258.html`

"It's not really a programming language. It's purpose-built as a statistical analysis package, similar to SPSS, SAS, Minitab, or gretl."

Whether you need to learn any particular package/language depends on your context. Knowing Icelandic is a huge advantage if you need to communicate with people who only speak Icelandic. Otherwise, not so much.

Likewise, there are specialized packages that are a huge advantage to people working in very specific research areas. If you're never going to work in that area, you don't need it.

You don't need to be fluent in a package/language in order for it to be helpful, but you do need a basic understanding. There is a turning point in your learning before which you're just not going to be able to use the language/package with anything but frustration. You just won't have enough vocabulary and basic ability to use the grammar without looking everything up.

But once you're past that point, you can do a lot, even before you're fluent. More importantly, you're poised to learn even more very quickly.

Be careful with if conditions:

If you post code like: if job $== 1$ job$_a ftershock == 0 replacepg0110_N EW = 0 replacepg0210_N EW = 0 repl$

IT WILL not work

This one will: replace pg$0110_N EW = 0 if job == 1 job_a ftershock == 0 replacepg0210_N EW =$
$0 if job == 1 job_a ftershock == 0 replacepg0510_N EW = pg0510_N EW + replrate/100 *$
$wagesif job == 1 job_a ftershock == 0 replacepg0510_N EW = pg0510_N EW + replrate/100 *$
$self employedincomeif job == 1 job_a ftershock == 0 // NOTE : THISASSUMESTHATRRFORWAGE$
$EMPLOYMENTARETHESAME$

2 Quotation signs

In Stata, you have to use an apostrophe (below the Esc key on US keyboards) as a left single quotation mark. As an illustration, for how single quotes should look like in Stata code, `example' is what you want. Using the correct ones is important because they are critical to access local macros (see below).

Some more details from the manual: "For the left single quote, we use the grave accent, which occupies a key by itself on most computer keyboards. On U.S. keyboards, the grave accent is located at the top left, next to the numeral 1. On some non-U.S. keyboards, the grave accent is produced by a dead key. For example, pressing the grave accent dead key followed by the letter a would produce à; to get the grave accent by itself, you would press the grave accent dead key followed by a space. This accent mark appears in our help files as `."

Note: The fact that an apostrophe is used as a quotation sign is a controversial issue among the Stata community (especially among new users or users who are used to other languages) and causes confusion as evidenced by the numerous online discussions on this topic.

3 Comments

There are four different options in Stata:

1. begin the line with *. This indicates that the line is to be ignored. Does not work within Mata.
2. begin the comment with //. This indicates that the rest of the line is to be ignored. The // comment indicator may be used at the beginning or at the end of a line. However, if the // indicator is at the end of a line, it must be preceded by one or more blanks. That is, you cannot type the following:

```
tabulate region// there are 4 regions in this dataset
```

3. place the comment between /* and */ delimiters. The /* and */ comment delimiter has the advantage that it may be used in the middle of a line, but it is more cumbersome to type than the other two comment indicators. What appears inside /* */ is ignored
4. comment indicator, ///, that instructs Stata to view from /// to the end of a line as a comment and to join the next line with the current line. /// is one way to make long lines more readable. Like the // comment indicator, the /// indicator must be preceded by one or more blanks. Is used rather infrequently.

4 Setting directories

Stata will interpret the forward slash / correctly as a path delimiter and it is recommended to use them. This is because

- it allows the code to run in a Mac (Unix) environment
- it prevents confusion with the backward slash's property as an escape character[1] in Stata. Consider the following example: Suppose you use a backslash in a command and want a macro expanded after it as in

```
use c:\data\`myfile'
```

[1] An escape character is a character which invokes an alternative interpretation on subsequent characters in a character sequence. You know this from Tex: To get $ you use \$ as \ is an escape character which suppresses the special meaning of $ as a math environment delimiter.

Because the backslash is an escape character and prevents macro substitution, Stata will literally try to read a file named `c:\data'myfile'`. Therefore, the escape character must also be able to escape itself - to tell Stata not to have it escape the opening macro character that might follow. Hence, you would have to code this as

```
use c:\data\\‘myfile’
```

or use a forward slash instead as suggested above.

4.1 Relative directories - navigating directories

"Stata seems to follow the same directory structure as most OS's in so much as '.' refers to the current directory, and '..' refers to the next level up."

5 log files

Can run more than one log file:

log using test1, replace log using test2, replace name(test2)

More details here: `https://www.statalist.org/forums/forum/general-stata-discussion/general/1396356-more-than-one-log-file`

There is a log file handle: `https://www.stata.com/statalist/archive/2008-06/msg00803.html`

6 Some commands

6.1 Set obs

`https://www.stata.com/manuals13/dobs.pdf`

Need to declare this if initializing variables like gen tets = 1.

VERY stata specific

6.2 cond

cond(x,a,b) returns a if x evaluates to true (not 0) and b if x evaluates to false (0).

More details here: `https://www.stata-journal.com/sjpdf.html?articlenum=pr0016`

6.3 group

Makes numerical variable out of categorical variable.

6.4 Compress

Also link to large datasets.

Note: This command affects the format of variables.

6.5 Sleep

VERY IMPORTANT. This can be an issue if you are running other applications which consume some resources and thereby slow down execution.

`https://www.statalist.org/forums/forum/general-stata-discussion/general/1405940-file-can`

6.6 Collapse

Very useful. Details here: `https://stats.idre.ucla.edu/stata/modules/collapsing-data-across-ob`

Note in particular the weight and median versus mean option

6.7 #delimit

The #delimit command is a Stata preprocessor command. It resets the character that marks the end of a command. It can be used only in do-files or ado-files.

In general, commands in a do-file can be delimited with a carriage return or a semicolon. When a do-file begins, the delimiter is a carriage return. The command "#delimit ;" changes the delimiter to a semicolon. To restore the carriage return delimiter inside a file, use "#delimit cr".

Note: When a do-file begins execution, the delimiter is automatically set to carriage return, even if it was called from another do-file that set the delimiter to semicolon. Also, the current do-file need not worry about restoring the delimiter to what it was because Stata will do that automatically.

WHAT DOES THIS SECOND SENTENCE MEAN?

Just because you have long lines does not mean that you must change the delimiter to semicolon. Stata does not care that the line is long. There are also other ways to indicate that more than one physical line is one logical line. One popular choice is ///

LINK THIS TO COMMENTS AND DIRECTORIES -¿ could be reason why had weird comment problem?

SUPER IMPORTANT: If you set ; as delimiter, whenever you use it within a comment which starts with a star for one line, it will cause statements in comment before to be read as a command. This can cause hard to debug problems... Punchline: if use ; DO NOT use it within comments

6.8 set seed

Affected commands: such as runiform() and rnormal().

See here for complete list: `https://www.stata.com/manuals13/dfunctions.pdf#dfunctionsDescriptio`

6.9 sca

scalar define scalar name = exp

"scalar define defines the contents of the scalar variable scalar name. The expression may be either a numeric or a string expression. String scalars can hold arbitrarily long strings, even longer than macros, and unlike macros, can also hold binary data."

"Stata scalar variables are different from variables in the dataset. Variables in the dataset are columns of observations in your data. Stata scalars are named entities that store single numbers or strings, which may include missing values"

6.10 tokenize

http://thedatamonkey.blogspot.com/2011/01/stata-tokenizing-locals.html

https://www.stata.com/manuals13/ptokenize.pdf

6.11 clear

clear removes data and value labels from memory. It is the same as typing

```
drop _all
```

and

```
label drop _all.
```

clear matrix eliminates from memory all matrices created by Stata's matrix command; it does not eliminate Mata matrices from memory

clear programs eliminates all programs from memory. It is the same as

```
program drop _all
```

To check which programs are currently loaded, type

```
program dir
```

The ado in front indicates that the program was automatically loaded.

clear ado eliminates all automatically loaded ado-file programs from memory but not programs defined interactively or by do-files. It is the same as

```
program drop _allado
```

NOTE: All commands above keep local and global macros (this includes clear all). To delete them, use macro drop $_all$

6.12 discard

"In short, discard causes Stata to forget everything current without forgetting anything important, such as the data in memory."

Most important: all automatically loaded programs such as in ado files. eliminates information stored by the most recent estimation command and any other saved estimation results

More details here: `https://www.stata.com/manuals13/pdiscard.pdf`

Discard is useful to debug ado files: "Use discard to debug ado-files. Making a change to an ado-file will not cause Stata to update its internal copy of the changed program. discard clears all automatically loaded programs from memory, forcing Stata to refresh its internal copies with the versions residing on disk."

Program:

`https://www.stata.com/manuals13/pprogram.pdf`

See also here: https://www.stata.com/statalist/archive/2004-03/msg00704.html

6.13 which

The command which filename displays the directory of the file (if Stata knows where it is).

6.14 quietly

The manual documentation is here `https://www.stata.com/manuals13/pquietly.pdf`

```
qui command
```

Performs command but suppresses terminal output for the duration of command. This option is used frequently. It prevents the screen from being filled with non-relevant information. For example, the following sequence makes sure that only outliers (points below the 5th or the 95th percentile) are displayed. Without quietly, the screen will be clutered by the regression output, predict notes and summarize output.

```
program myprog
    quietly regress '1' '2'
    quietly predict resid, resid
    quietly sort resid
    quietly summarize resid, detail
    list '1' '2' resid if resid< r(p5) | resid> r(p95)
    drop resid
end
```

Note that you can use the qui command within an environment by using {} so that this sequence of commands is identical to above

```
program myprog
    quietly {
        regress '1' '2'
        predict resid, resid
        sort resid
        summarize resid, detail
    }
    list '1' '2' resid if resid< r(p5) | resid> r(p95)
    drop resid
end
```

Finish this!

and check **https://www.parisschoolofeconomics.eu/docs/yin-remi/do-file.pdf**

6.15 textsizestyle

Like Tex, Stata has a set of predefined text sizes. They can be seen here `https://www.stata.com/manuals13/g-4textsizestyle.pdf`

6.16 erase (rm)

```
erase|rm filename
```

This command erases files stored on disk. Note that filename can be a directory location such as `...\mydata\oldfile.dta`

Also, you can apply this command to a file directly after loading it with the use command; as this command loads the file into STATA's memory, deleting the locally saved version of the file is possible. It can be manipulated and saved without loss of data.

6.17 Changing variable names

```
rename oldname newname
```

This command changes the name of variable oldname to newname. Note that each variable has to be renamed in a separate command. The command also allows to rename groups of variables and to change the cases of variable names. You can do bulk partial changes via

```
renpfix male m
```

which changes the name of all variables starting with "male" such that the variable names will start with "m" only.

6.18 Capture (cap)

"capture executes command, suppressing all its output (including error messages, if any) and issuing a return code of zero. The actual return code generated by command is stored in the built-in scalar rc"

"capture is useful in do-files and programs because their execution terminates when a command issues a nonzero return code. Preceding sensitive commands with the word capture allows the do-file or program to continue despite errors. Also do-files and programs can be made to respond appropriately to any situation by conditioning their remaining actions on the contents of the scalar rc."

capture can be combined with { } to produce capture blocks

SIF type	SIF examples	Units
weekly date	2,601	weeks since 1960w1
monthly date	600	months since 1960m1
quarterly date	200	quarters since 1960q1
half-yearly date	100	half-years since 1960h1
yearly date	2010	years since 0000

SIF type	Conversion function	Note
weekly date	tw = weekly(HRFstr, mask)	tw may be float or int
monthly date	tm = monthly(HRFstr, mask)	tm may be float or int
quarterly date	tq = quarterly(HRFstr, mask)	tq may be float or int
half-year date	th = halfyearly(HRFstr, mask)	th may be float or int
yearly date	ty = yearly(HRFstr, mask)	ty may be float or int

When to use it? 1) DEbugging SEE REFERENCES THERE 2) FOR LOG. SEE REFERENCES THERE 3) Sometimes you will write programs that do not care about the outcome of a Stata command. You may want to ensure, for instance, that some variable does not exist in the dataset. You could do so by including capture drop result.

6.19 datetime

Stata follows an idiosyncratic way of recording dates. This is called Stata internal form (SIF). SIF values are stored as regular Stata numeric variables. As the table below illustrates, 1960 is a common reference year

To convert non-Stata formats[2] use one of the ways shown in the folloing table.

Illustration of the mask element: You have a date stored in mystr, an example being "22/7/2010". In this case, you want to create an SIF date instead of a datetime. You type . gen eventdate = date(mystr, "DMY") In other words, mask is a placeholder for the input format of the non SIF date.

7 Importing

χ_{SJ}

[2]in the Stata manual, all non SIF formats are incorrectly labelled Human Readable Form (HRF)

7.1 import excel

When using this command, some numerical values get imported as strings. This can a) mess up future computation steps b) create cryptic error messages. See more here:

`https://www.stata.com/support/faqs/data-management/numeric-variables-input-as-string/`

8 Saving and exporting

8.1 putexcel

Need to declare file, name and sheet first: Putexcel set ...

Note: Long names of sheets will kill process but error message is completely uninformative: `https://www.statalist.org/forums/forum/general-stata-discussion/general/1307009-putexcel-and-export-excel-return-errors-if-the-sheet-name-is-longer-than-31-cha`

8.2 xml_tab

"Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable."

To save data from Stata as xml, use the xml$_tabcommand.Detailsarehere$:

Note that the command is a user provided one. Hence, it is not developed continuously and is known to have some bugs. Prefer the other option (see below).

8.3 matrix define plus putexcel

This is much more customizable. Both commands are official ones.

`https://www.stata.com/manuals13/pmatrixdefine.pdf` `https://www.stata.com/manuals13/pputexcel.pdf`

Illustration is here: `https://blog.stata.com/2017/01/10/creating-excel-tables-with-putexcel-pa`

8.4 graph export

The structure of the command is

```
graph export newfilename.suffix [, options]
```

The format can be specified in one of two ways:

1. Use a dedicated option

```
as(fileformat)
```

2. Set the output format by the suffix of newfilename.suffix

Available formats are: ps, eps, svg, wmf, emf, pdf, png, tif

The format can be set by a macro as illustrated by these commands:

```
global GRAPHFORMAT = "emf"
graph export "${GRAPHS}/somedirectory/filename.${GRAPHFORMAT}", replace
```

9 Macros

It is common practice to put braces around global macros when concotenating them. https://www.stata.com/statalist/archive/2008-04/msg00388.html

This is because of globals are not declared in quoation marks i.e. as strings, if there are white spaces in the directory names Stata will only concetante up to the spaces.

As variables of Stata code, macros are fundamental elements. A macro is a string of characters, called the macroname, that stands for another string of characters, called the macro contents. Macros can be global or local.

9.1 Local macros

The difference between local and global macros is that local macros are private and global macros are public.

The local macro alpha in myprog is private in that no other program can modify or even look at alpha's contents

It is possible to make locals created in a do file available interactively once the do file has finished executing. This is achieved using the include command. As far as I know, this only works for whole do files (rather than selections in the Do-file Editor window). There doesn't seem to be any way of making locals defined interactively available from a do file.

One sets the contents of a local macro with the local command. The expression

```
local shortcut "myvar thisvar thatvar"
```

means that 'shortcut' is a synonym for "myvar thisvar thatvar" (note the single quotes). As a result, the following two commands are identical in Stata

```
list 'shortcut'
list myvar thisvar thatvar
```

Macros can be combined. Another macro defined as

```
local cmd "list"
```

can be used together with the one defined earlier so that the command

```
'cmd' 'shortcut'
```

produces *list myvar thisvar thatvar*. Omitting spaces when combining local macros results in merged string outcomes.

Example: First, define the local macro as

```
local age2 "33"
```

Next, embed it in another string:

```
display "My age is `age'"
```

so the result is: My age is 33

Each do file has its own local space, and the only locals it can see are those defined in that do file. Moreover, once a do file has finished executing, all of its locals are deleted, meaning that they need to be recreated if the do file is run again

9.2 Global Macros

Global macros are similar to local macros except that their contents are set by using global rather than local. Furthermore, their contents are addressed by prefixing them with a $ rather than enclosing them in ' '. For example, consider

```
global shortcut "alpha beta"
```

so now $shortcut is equivalent to "alpha beta". Note that local and global macros may have the same names, but even if they do, they are unrelated and are still distinguishable. Importantly, the local macro defined above is not overwritten by a global macro of the same name so that 'shortcut' still produces "myvar thisvar thatvar".

To see what is stored in a global macro, type

```
disp "$macroname"
```

This command shows the string inside of quotations. Note that typing

```
disp $macroname
```

will automatically execute the string as a command.

9.2.1 Dollar Signs in Paths

Dollar signs at the beginning of path specifications refer to global macros. For example, $global refers to a global macro with the name global.

One can define a global as

```
global test "c:\whatever\wherever"
```

and append it by additional specifications since

```
log using \$test\df.log
```

is interpreted as

```
log using c:\whatever\wherever\df.log
```

9.3 Difference Local and Global Macros

When you write a program using local macros, you need not worry that some other program has been written using local macros with the same names. Local macros are just that: local to your program. Global macros, on the other hand, are available to all programs. If both myprog and mysub use the global macro beta, they are using the same macro.

You can display all macros using the command

```
macro list
```

or macro dir. Local macros are predeced by an underscore.

Something SUPER IMPORTANT on local macros and debugging: "Perhaps the reason some or all of your locals are empty is that you wrote your code in the Do-file Editor window, but are submitting the lines of code by selecting a few at a time and running them, then selecting the next line(s) and running them, etc.. Unfortunately, to run a selection of lines Stata creates a temporary do-file containing the selected lines, so once that temporary do-file is finished, the local macros you defined within it vanish, because local macros are "local" to the do-file within which they are run." -¿ link this to debugging.

10 Debugging

Two main problems:

1. Can be very difficult to identify line of do file which cuases error, i.e. which specific command. Stata does not output where it happens.
2. Moreover, error messages can be very cryptic. Example: Type mismatch or invalid syntax.

10.1 general advice

1. Avoid nested do files
2. Avoid nested loops
3. Use brute force breaks (workspace stays as at break)
4. Write code in a linear way

10.2 macros

Each do file has its own local space, and the only locals it can see are those defined in that do file. Moreover, once a do file has finished executing, all of its locals are deleted, meaning that they need to be recreated if the do file is run again

However, often macros make mistakes very difficult to identify because what is actually hapenning in the commands is difficult to observe.

* One way to help in this is to add lots of display commands so that you can see what commands are actually being sent to Stata.

for example, in a loop di "i = ‘i’" di "gen var‘i’‘v’ = ‘i’ * ‘v’"

10.3 Line by line execution

Debugging is not a strong point of Stata. For users of Matlab etc. need to get used to certain tricks and workarounds. Does not make use of being non compiled language, no built in tool.

Note: When Stata stops on an error, it does NOT lose the workspace information. Means that the data loaded at time error occured can be accesses by you.

Can insert invalid command right before, so that see all the workspace right before buggy command gets executed

Alternatively: can set pause before error occurs and then copy paste lines of codes into terminal.

NOTE: IF you run line by line from do file edtor: lose local macros.

If copy paste from there into command line, keep them!

10.4 log files for debugging

10.4.1 debugging with local macros

main problem is that once Stata stops due to an error, local macros are lost. This means that cannot determine their content. Trick: Use a log file (see above) and print contents of local macros inside.

10.4.2 result screen limitation

Face the problem that cannot scroll back indefinitely. So if you want to see steps early on, cannot retrieve by scrolling back.

10.5 set trace

It echos the lines Stata executes internally.

Problem of Stata (especially for new users), error messages or not always informative. Example: Can get get "no observations" error if trying to execute command on type of data for which not comformable. For example run regression on string variable.

Set trace prints detailed output on the computation steps to the result window. Helps to figure out at which step error occurs. Is useful especially for built in functions to get better picture on what step causes problems and how to fix them.

Enter set trace on and run program. Use set trace off to disable it again. Note: the program execution takes much longer when trace is set on. So if you know the line cuasing the error, you can just inlcude set more on right before it into the do file.

Add: local macros in the line by line execution... see above.

10.6 User provided tools

There is something which seems relevant: `https://ideas.repec.org/c/boc/bocode/s457008.html` but I havent tried it out myself yet.

11 adopath

Allows to set a directory where ado files are stored.

```
adopath + path_or_codeword
```

adds a new directory or moves an existing directory to the end of the search path stored in the global macro `S_ADO` (++ does the same for the beginning of the search path). For example, to add `C:\myprogs` to the end of the ado-path type

```
adopath + C:\myprogs
```

12 SSC

The Statistical Software Components (SSC) is the premier Stata download site for community-contributed software on the web. ssc provides a convenient interface to the resources available there.

The command

```
ssc install programname
```

installs the program with name programname. The programs are downloaded as ado files into the Stata directory. On a PC, they can be found at `"~\ado\plus"` while on a Mac they are acessible through the library. Remember that any ado file can be found by using the which command (see commands above).

```
ssc describe programname
```

Describes, but does not install, the specified package.

```
search programname
```

Searches the keyword database and the SSC database for a package with the name programname.

On the ECB desktops, any SSC inquiry results in an error: host not found r(631) According to the documentation, this is due to the ECB network firewall settings and can only be resolved by the network administrator.

IF NO ADMIN PERMISSION TO USE SSC INSTALL, CAN DOWNLOAD and ADD MANUALLY? YES - but then get now updates to them of course.

To find specific packages, either use findit "wildcard" or use help "wildcard". If Stata can contact the SSC server, it will search the list of existing packages there as well.

search searches the SSC and other places, too. search provides a GUI interface from which programs can be installed, including the programs at the SSC archive

I think search and findit are the same...

If you dont know the name of the packages but some related command or concept. Use findit followed by a key term such as Heckman, sdmx, iv, etc.

The findit command first searches Stata's official help files and notes that there is an official heckman command and several other related commands (this makes findit a powerful tool for figuring out how to do things in Stata in general, not just for finding user-written programs). It then searches Stata's web site and locates several FAQ entries, plus an example on UCLA's large statistics web site. It then begins to list relevant user-written programs, organized into "packages." Programs that were described in the Stata Journal or the older Stata Technical Bulletin are listed first.

While few user-written programs are updated as frequently as ice, it's still important to get the latest versions of any user-written programs you install. Sometimes updates will include important bug fixes, though the SSC archive has quality control measures in place to try to catch bugs before the program is distributed.

The easiest way to check that your user-written programs are up-to-date is to type:

```
adoupdate
```

The adoupdate command notes where each package was downloaded from and goes back to that location to see if a more recent version is available. If there is, you can install the latest version by typing:

```
adoupdate, update
```

You can get a list of the packages you've installed by typing:

```
ado dir
```

DOES THIS INCLUDE PACKAGES YOU INSTALLED MANUALLY? YES!!! I found out because before SSC install did not work and I manyally installed ainequal which is now in the list.

You can remove a specific package using

```
ado uninstall package
```

13 SDMX

SDMX (Statistical Data and Metadata eXchange) is an international initiative that aims at standardising and modernising ("industrialising") the mechanisms and processes for

the exchange of statistical data and metadata among international organisations and their member countries. It is supported by the BIS, the ECB, Eurostat, the IMF, the OECD, the UN and the World Bank. This includes data provided by the ECB's statiscal data warehouse (SDW). An excellent tutorial on SDMX can be found here: `http://ec.europa.eu/eurostat/web/sdmx-infospace/welcome`

There are at least three exisisting codes which use SDMX to pull data into Stata: getdata, sdmxuse.[3] and the SDMX Connector for STATA developed by Attilio Mattiocco from the Bank of Italy. Available here: `https://github.com/amattioc/SDMX/releases`. They cannot be installed via SSC and instructions are here: `https://github.com/amattioc/SDMX/wiki/SDMX-Connector-for-STATA`

Note that getdata is based on Attilio's connector which is developed by using the Stata SDMX connector licensed to the Bank of Italy.

13.1 getdata

This package is available via SSC install.

```
getdatacodes
```

opens the same SDMX Helper Tool as does sdmxHelp.

Its use is as follows:

Main problem: There is a weird error on some machines when using ECB data. See the issue on the package I posted. Has not been able to be resolved. Using the SDMX connector of Attilio is not an option since this kills the easy reproducibility of the code (it is not available via SSC install. I suggested it but I doubt it will happen any time soon).

The jar file which is bundled in getdata is two years older than the one avaibale on Attilios github section. I suspect this could be the problem...

Note: The getdata package installs a large number of ado files as well as the jar file. The ado files include the getTimeSeries.ado file which is originally from Attilio

13.2 sdmxuse

WHAT ABOUT sdmxuse? Do they all have search window as Attilio's?

[3] On the ECB desktops, these cannot be installed via ssc install due to IT restrictions. Contact IT services to try to remove this constraint.

14 ODBC

"odbc allows you to load, write, and view data from Open DataBase Connectivity (ODBC) sources into Stata. ODBC is a standardized set of function calls for accessing data stored in both relational and nonrelational database-management systems."

"Stata's odbc command allows you to load, write, and view data from Open DataBase Connectivity (ODBC) sources. Before you start using the odbc command in Stata, you must first set up a data source name (DSN) in the ODBC Data Source Administrator. Setting up a DSN differs slightly depending on your version of Windows: Windows 8 or Vista or Windows 7." From here: `https://www.stata.com/support/faqs/data-management/configuring-odbc-win/` Following the instructions on this link, new sources can be set easily.

Once this is done, the official documentation of the odbc command is here `https://www.stata.com/manuals/dodbc.pdf` and contains useful explanations and examples.

```
odbc list
```

produces a list of ODBC data source names to which Stata can connect

15 Timer

STATA has a timer which reports results in seconds. Use is straightforward but note that STATA allows to run up to 100 timers simultaneously. Details are here: `https://www.stata.com/help.cgi?timer`

Important: If you include clear all anywhere in your code, this also clears the timer.

16 Large datasets (Speed)

Despite being far from low level, Stata is surprisingly fast. SHOW info. Reason is probably that procedures have been optimized over time (see history of Stata, ie number of versions). Despite this, age of big data is a challenge to Stata.

Regarding speed: Use built in functions. As a novice: Invest some time to try out different ways to do the same. Just as in other languages, there are usually several options to get things done. Example...

16.1 Some best practices and practical tipps

Some useful practical suggestions to work with large datasets in a time efficient manner are here: `http://www.nber.org/stata/efficient/`

Even more useful best practices are here `http://www.econometricsbysimulation.com/2012/07/7-ways-to-speed-up-your-do-files.html` and listed here:

- Drop unused data
- Drop unnecessary variables
- Remove visual feedback (use qui command)
- Remove redundant data writing and reading commands
- Use the compress command

- Convert long string variables to factor variables (use encode longstring, gen(factor$_{l}ongstring$))

16.2 Tools

Seems that Stata community has noted in recent years that big data places a challenge to Stata language. A number of routines which improve the speed of the existing ones have been proposed and discussed:

gtools: Speed

`https://gtools.readthedocs.io/en/latest/index.html` `https://github.com/mcaceresb/stata-gtools`

ftools: For large datasets `https://www.stata.com/meeting/baltimore17/slides/Baltimore17_Correia.pdf`

16.2.1 Parallel

`https://github.com/gvegayon/parallel`

`https://github.com/sergiocorreia/ftools`

16.3 What about mixed language programming

Matlab common practice to profile, identify bottlenekcs and then outsource specific tasks to low level laanguages via Mex commands.

How to do this in Stata?

16.3.1 Stata and R

https://github.com/haghish/rcall

http://www.haghish.com/packages/Rcall.php