Class 1: Computation & Economics – An Introduction

Damian Clarke

Friday January 24, 2020

Research Methods II
MRes. in Economics

Today's Plan

Basic Course Information

Discussion of Fundamental Programming Tools

- High-level vs. low-level programming languages
- Work-flow and Version Control
- Operating Systems
- Useful Practices
- How to Get Help

Discussion of a Matrix Based Language

Basic Course Information

Course Information

Welcome!! This is a "Research Methods" course, focused on computation for economics.

- In this course we will discuss over-arching principles in computation, applied to challenges encountered in economic research
- The goal is to make you comfortable enough with a broad range of tools so that you can attack any research question of interest
- The goal *is not* to explain one single language and its syntax. There is a wealth of such information available in just a few key strokes.
- But we will use a number of languages, and get into the types of issues frequently encountered 'in the wild' in economic research.

Course Information

The more 'bureaucratic' details of the course are included in the program.pdf document circulated with these slides.

- The course will consist of around 10 lectures via zoom, Fridays from 3-5pm
- I am also happy to arrange remote 'office hours'
- Any questions or suggestions are always welcome: damian.clarke@protonmail.com
- Additional important details are in the course program...

Discussion of Fundamental Programming Tools

An Overview

Overview

There are many many computational tools available which may make your work as an economist simpler/better/more impactful.

- There is not one 'right way' to do things
- There are, however, a number of general principles that are useful to adopt
- Here it is important to think about who we are working for
  - Ourselves in t_1
  - Others $t_2, \ldots, T_\infty$
  - Ourselves in $t_2, \ldots, T_N$
- It is certainly not the case that we should seek to minimize all (time) costs in t_1
- This explains the value of a course like this, and optimization of reproducible code
- Of course there is a trade-off at multiple margins in computing and writing code

Overview

Thinking about an economist deciding how to implement a research project (or more likely, a research agenda), I want to discuss a number of general principles today. As in everything, the idea is that you can find your own optimum response.

1. Programming languages
2. Work flow and version control
3. Operating systems
4. Useful Practices
5. How to get help

Discussion of Fundamental Programming Tools

High-level vs. low-level programming languages

Programming Languages

There are many programming languages which are used to conduct economic research in high level journals. This includes (though is not limited to):

- Stata, R, MATLAB/Octave, Julia
- Python
- Fortran, C/C++
- Perl, PHP
- SAS, SPSS

Above these are colour-coded according to whether they are interpreted, compiled, or JIT ("Just-in-Time") compiled.

Programming Languages

You may hear a distinction in programming languages between "high-level" and "low-level" languages.

- In a broad sense, this refers to the degree with which the language interacts with basic machine functions
- In **low level** languages, it is likely that you often:
  (a) need to request memory blocks
  (b) assign variable types
  (c) compile code to make an executable file
- In **high level** languages many of these functions are taken care of allowing you to focus on the application at hand. There is often a cost in terms of speed.
- I do not believe the "high" vs. "low" distinction is a value judgement!

What does this look like in practice?

An Example in C

```c
#include <stdio.h>
int main() {
  // printf() displays the
  // string inside quote
  printf("Hello, World!");
  return 0;
}
```

The above code must be compiled and then can be run.

An Example in Python

```python
print "Hello, World!"
```

The above code can be simply saved and run as is.

A Clarification

One thing to note is that this classification is relative. Originally, low-level referred to assembly language, and C as high level.

- Today, C is frequently referred to as low-level given lack of garbage collection, direct access to memory addresses, etc.
- Languages like Python are clearly high level: there is no need to explicitly define and fix types, nor consider memory locations and assignment
- Generally, there is an "abstraction penalty" with higher levels
- However, this is not always the case, eg Julia with "Just-in-Time" compilation seems to be very fast (see here).
- Jesús Fernández-Villaverde has more complete discussion on this here.

Discussion of Fundamental Programming Tools

Work-flow and Version Control

Workflow

The programming languages we use are often embedded in a more general 'workflow' which may include version control, typesetting/markdown, reproduction, etc.

- Building a general standardised work-flow will be useful across all your projects
- However this will tend to vary depending on the nature of the project, work 'locations', coauthor styles, etc.
- At a minimum, workflow should be designed to:
  (a) efficiently permit replication, and
  (b) avoid manual functions which potentially introduce errors, eg copy and paste.

Workflow

A (stylised) economics research project may consist of the following steps:

1. Data collection
2. Data cleaning
3. Analysis and generation of results
4. Writing a paper and slides
5. Generation of replication materials

Workflow

It is very unlikely that one could (or would) want to use a single tool to complete all of these tasks. A somewhat complicated project may consist of the following type of tools for each step.

1. Data collection: **Webscraped with Python**
2. Data cleaning: **Some scripting language, eg Stata, R, MATLAB**
3. Analysis and generation of results: **Scripting language, eg Stata, R, MATLAB**
4. Writing a paper and slides: **LaTeX and pdflatex compiler**
5. Generation of replication materials **All housed in a Makefile**

Workflow

Likely what we would like to do here is:

1. Automate this process completely or as much as possible
   - What happens if next month we want to update at step 1, eg grabbing new data?
   - And what if later a referee wants us to use an alternative method of data cleaning at step 2?
2. 'Remember' what was being done in previous versions of code when changes are made to revert to old results if necessary

The first of these is more about designing an efficient replication process (for example using a Makefile or overview file), while the second is more about "version control". We will discuss these in turn.

Workflow

One other brief aside is related to the use of Dynamic documents or dynamic notebooks, where code, results, and text are all presented in a single dynamic "notebook" which can be viewed online, or printed as pdfs or similar.

- One such example is Jupyter which supports a number of languages
- Similarly, the `knitr` package in R is based on this idea.
- I have not used this, so can not comment too much
- Prima facie, it seems tricky to adopt this for an academic paper given journal styles, etc., but perhaps this is useful for problem sets or online visualisations of research results

Workflow – Make

`Make` is a Unix tool which specifies the full path a project follows, implying that rather than having to go through a workflow piece by piece, one can simply run the make file with a single command.

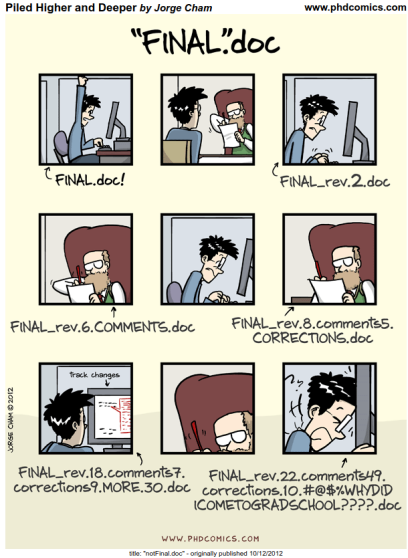
- The value added of make is that it only runs steps which are necessary
- If nothing prior to step x has been altered, make will see this and only run steps $x, x+1, \ldots$
- A very complete discussion of Make with examples is provided by Jesús Fernández-Villaverde
- Gentzkow and Shapiro have a very nice suggestion for Windows systems using a shell script to automate workflow (chapter 2 here)

Version Control

Version control is an extremely useful tool allowing teams (and individuals) to collectively work on projects, while tracking changes in an efficient and fully trackable way over time.

- The idea is that a single file can be tracked using the same file name (eg analysis.m), and version control will keep track of changes, allowing for the file to be compared side-by-side and reverted to previous versions
- There are a number of stand alone softwares that do this
- The concepts are also at times used within individual platforms (eg dropbox, word's "tracked-changes"
- Perhaps the most common/powerful option is to use git.

The Classic Reference Here



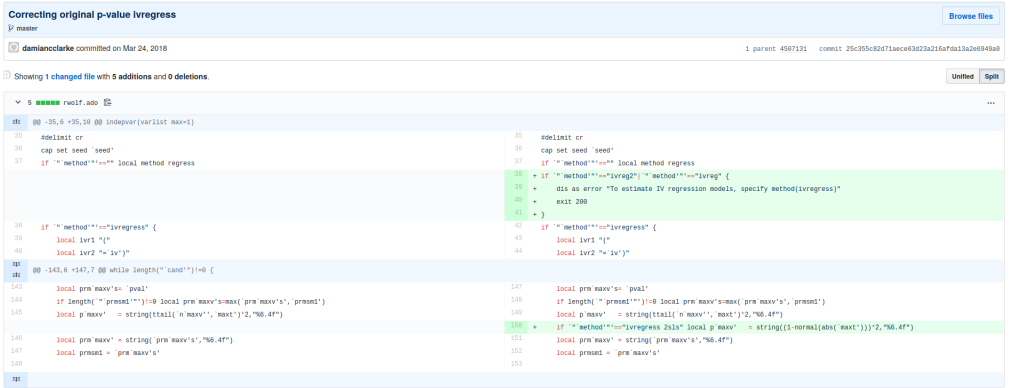
title: "notFinal.doc" - originally published 10/12/2012

Version Control – git

git is a particular open source software that fully deals with version control

- It can be used on all operating systems
- It also has a number of options for graphical user interfaces
- Online platforms can simply interact with git making it very easy to engage in collaborations (eg github, gitLab).
- Here we will not review the full details of using git, but there are many good resources:
  - Frank Pinter: Git for economists
  - Jesús Fernández-Villaverde: Chapter on git
- There is an online repository for code in this course which you are welcome to "fork" into, though this is not necessary

A Simple Example (from github)



Correcting original p-value ivregress

master

Browse files

damianclarke committed on Mar 24, 2018 — 1 parent 4507131 — commit 25c355c82d71aece63d23a216afda13a2e6949a0

Showing 1 changed file with 5 additions and 0 deletions.

Unified | Split

5 ■■■■■ rwolf.ado

```
@@ -35,6 +35,10 @@ indepvar(varlist max=1)
 #delimit cr
 cap set seed `seed'
 if `"`method'"'=="" local method regress
+if `"`method'"'=="ivreg2"|`"`method'"'=="ivreg" {
+    dis as error "To estimate IV regression models, specify method(ivregress)"
+    exit 200
+}
 if `"`method'"'=="ivregress" {
     local ivr1 "("
     local ivr2 "=`iv')"
@@ -143,6 +147,7 @@ while length("`cand'")!=0 {
     local prm`maxv's= `pval'
     if length(`"`prmsm1'"')!=0 local prm`maxv's=max(`prm`maxv's', prmsm1')
     local p`maxv'   = string(ttail(`n`maxv'', maxt')^2,"%6.4f")
+    if `"`method'"'=="ivregress 2sls" local p`maxv'   = string((1-normal(abs(`maxt')))^2,"%6.4f")
     local prm`maxv' = string(`prm`maxv's',"%6.4f")
     local prmsm1 = `prm`maxv's'
```

Version Control

There is another benefit to this, and that is, if used correctly, it is a much more transparent way to share the whole history of an empirical project.

- This is very much related to the ongoing "replication crisis" and "p-hacking" discussions in empirical social sciences.
- Across versions of code we can document any changes in specifications or statistical tests
- Of course, this is not a robust solution

Discussion of Fundamental Programming Tools

Operating Systems

Operating Systems

The operating system (OS) is the general interface and management system we interact with when we use our computers. A good understanding of the OS is key to efficient use of your computer.

- There are two broad classes of operating systems
  - Unix and Unix-like operating systems (includes things like Linux, MacOS)
  - Windows family
- There are a number of tools to try to make it possible to have the best of both worlds (eg emulators, WINE, Cygwin)
- Most servers and high-performance computing clusters are based on Unix OSs.

Operating Systems – Unix

There are a number of tools which will be much easier to use on Unix-like systems

- Things like make, gcc and bash are written for Unix
- In my opinion it is much easier to get up and running with a new programming environment in Unix-like systems
- Systems like Mint, Debian, Ubuntu, etc., are free, and not affected by the vast majority of computer viruses
- You have control over your OS
- If you have not used Unix before, the drawback is that there is a fixed cost that must be paid in learning
- In these classes I will not assume anything about the OS you use, and all code examples work regardless of OS

Discussion of Fundamental Programming Tools

Useful Practices

Useful Practices

Before moving on, let's discuss a few generally useful (or 'best') practices. We will briefly discuss broad ideas here, and illustrate these throughout the term. Ljubica Ristovska has a very good set of slides going into more depth. Here we will discuss very briefly:

1. Documentation
2. Consistent Style
3. Automation and Abstraction
4. Testing, Debugging and Profiling

We will also plan to make this a case of 'learning by doing' during this course and our careers.

Documentation

Research projects should be documented extensively both "globally" (eg using a README file) and "locally" (eg commenting within scripts).

Commenting in Scripts

- All programs we will likely work with have a way to write comments.
- These are points implied for human readers, that the machine will ignore (hence syntax of the comment itself can be human language).
- JFV: "You will probably spend more time reading code than writing code"
- Should be used throughout code wherever something is not absolutely clear
- At least 'section explanations' should be used
- In the same way code is maintained, comments should be maintained.
- Err on the side of verbosity.

Documentation

Research projects should be documented extensively both "globally" (eg using a README file) and "locally" (eg commenting within scripts).

README files

- These are files in the main directory giving global instructions relating to a project
- They may explain the procedures followed from start to finish, the external libraries needed, directory structure, etc.
- This should be the first thing you look for when you download a file.
- It may be called README, README.TXT, READ.ME, ..., or, in the case of markdown README.md
- 1 example, a README file from replication materials of a paper of Climent, Sonia and me: http://qed.econ.queensu.ca/jae/datasets/clarke001/readme.coq.txt

Consistent Style

There are a number of frequently used stylistic choices in programming, though often these aren't based on explicit (syntactic) rules.

- For example, people have an ideal directory structure they like to use for projects
- Certain default choices are used for data management (eg delimiters)
- Spacing in code is often user-specific (though there are languages where rules are in place)
- Intelligent variable naming can simplify work considerably

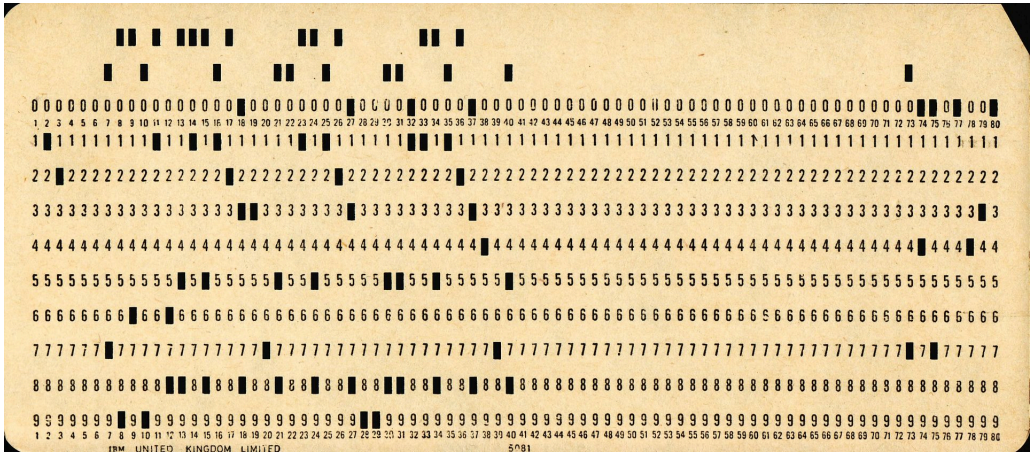
In many cases, even if there are no rules, internal consistency is key as this allows for re-use of standardized code.

Consistent Style

In some languages, spacing is a big thing.

- For example, Python uses indentations in code to find the end of loops and conditional statements
- Fortran reserves initial spaces in lines for particular keys or statements
- Often, you see codes which do not exceed 80 characters per line (and this is my preference). Why?

“Institutions” are Ingrained and Slow to Change



Consistent Style

Variable naming is also a form of documentation...

- Good variable naming can make things a lot easier, and bad variable naming can be a disaster.
- For example, consider a binary variable for male/female, and the potential names {`var1`,`gender`,`female`}
- There are particular standards one can elect to follow, eg `snake_case` or `camelCase`, and consistency is good
- Some languages also permit the bundling of additional information with variables (like variable labels), and then even more information can be contained

Automation and Abstraction

Where possible automation (avoiding human interaction) and abstraction (writing code that is flexible for many circumstances) should be embraced.

- For example, imagine arriving at an expectation using the law of iterated expectations: $E(Y) = \sum_{x=x_{min}}^{x_{max}} E(Y|X = x) \cdot P(X = x)$.
- One could tabulate X and then calculate the expected value of Y at each level of X by hand.
- But it would be better to get a list of all values of X and iterate through these to take the expected value of Y at each level automatically (AUTOMATION).
- And best of all, if we had a function to do this for any two variables of interest (ABSTRACTION).

Automation and Abstraction

This also ties in with the idea we discussed earlier of having a Make file to automate a program. Throughout this course we will return to discuss points of Automation and Abstraction further

- A clear case of this is in writing *functions* which allow us to separate out and re-use key codes
- This also allows for us to focus our attention on perfecting this once, and then use optimized code in the future
- Everything you do in a research project should be replicable using a machine
- In general, if you find yourself typing code into a language's command line by hand thinking you will remember what you did in the future, you are in trouble!

Testing, Debugging and Profiling

These three points come up as soon as you have some code written:

- Testing: My code runs, but does it do what it should?
- Debugging: My code does not run/compile. Why?
- Profiling: My code runs very slowly. Why?

Testing, Debugging and Profiling

In terms of testing, this is a constant process which requires vigilance.

- In empirical applications, it is often a good idea to see if a code returns 'correct' values in simulated data
- Testing should be done modularly. Each script should be tested in a stand-alone way
- Similarly, each argument in a function should be tested
- Consider whether you can proactively build in tests into code. For example, if an argument is the rate of unemployment, check that it is bounded between 0 and 100, otherwise generate an exception.

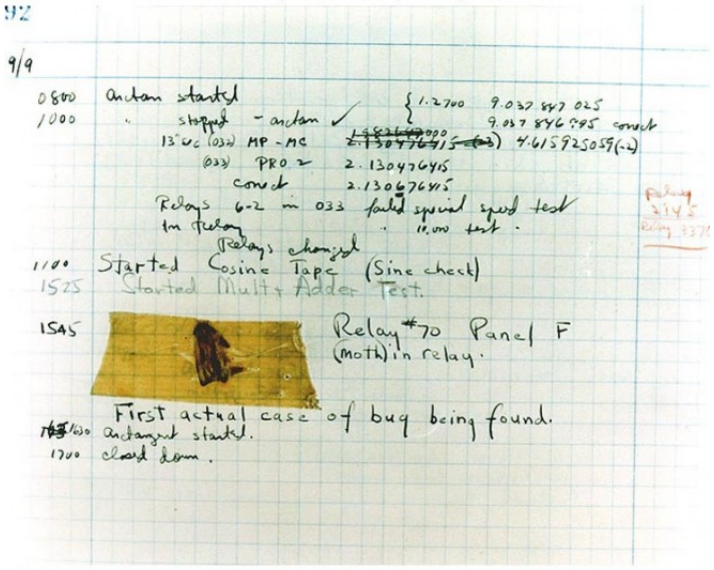


Figure: The Famous 'Computer Bug' from Grace Hopper's Notebook

Testing, Debugging and Profiling

Debugging refers to finding errors that are causing a script not to run.

- Most languages and compilers (especially those that we will use) have very good inbuilt debugging tools.
- You should read error messages and consult user's manuals as a first option when bugs come up.
- In the worst case scenario, pasting the exact error message into a search engine may provide relief.
- Note that sometimes programs issue warnings but still allow a script to run. These warnings should be at least looked into to, as often the best case is that they will show inefficiencies.
- Do not think that just because a script runs it is correct. Always go back to testing.

Figure: Wisdom of the Ancients

Testing, Debugging and Profiling

Finally, the issue of profiling is about finding 'hot spots' in a code.

- If a code is slow to run, it may be that there is a bottleneck that can be optimized
- But note, efficiency only comes after accuracy. Only begin optimizing when you are sure your code is correct
- Many languages have profilers which allow you to profile very easily (eg MATLAB)
- In languages where these aren't present, you can run time tests portion by portion
- Time permitting, we will discuss some points about parallelization and speeding things up.
- Do not over-optimize. There is a trade-off between machine time and programmer's time!

Discussion of Fundamental Programming Tools

How to Get Help

How to get help

Programmers can be somewhat brusque, but there is generally a willingness to help, often following the open source style movement.

- Stackoverflow is probably the 'front page' here, and offers support in all languages
- However, there are also a range of language-specific options. The statalist is a wonderful example of this.
- For more particular syntax type queries, all languages have some type of manual
- The challenge is often in knowing the right question to ask.
  - Having a basic vocabulary (which we will discuss in this course) is very useful
  - Eg, a search like "find occurrence of substring in string in python" gives more focused results than "matching letters in text in python" (although both will eventually lead to your answer!)
- Search engines are your friend (and not just google)

Discussion of a Matrix Based Language

Discussion of a Matrix Based Language

Moving to the MATLAB command line...