Introduction to Scientific Computing

Vahe Krrikyan/Johannes Pfeifer

University of Mannheim

pfeifer@uni-mannheim.de

May 20, 2016

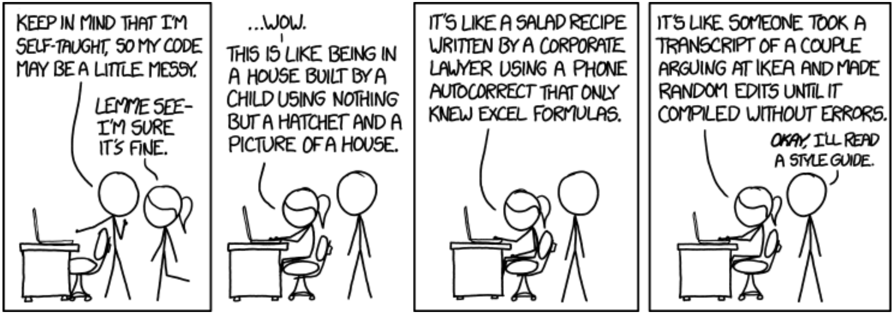
KEEP IN MIND THAT I'M SELF-TAUGHT, SO MY CODE MAY BE A LITTLE MESSY.
LEMME SEE– I'M SURE IT'S FINE.
...WOW. THIS IS LIKE BEING IN A HOUSE BUILT BY A CHILD USING NOTHING BUT A HATCHET AND A PICTURE OF A HOUSE.
IT'S LIKE A SALAD RECIPE WRITTEN BY A CORPORATE LAWYER USING A PHONE AUTOCORRECT THAT ONLY KNEW EXCEL FORMULAS.
IT'S LIKE SOMEONE TOOK A TRANSCRIPT OF A COUPLE ARGUING AT IKEA AND MADE RANDOM EDITS UNTIL IT COMPILED WITHOUT ERRORS.
OKAY, I'LL READ A STYLE GUIDE.

Figure: https://xkcd.com/1513/

Outline

1. Introduction
2. Motivation
3. Best Practices
4. Conclusion

Introduction

- This presentation is based on Greg Wilson et al. (2014). “Best Practices for Scientific Computing”. In: *PLoS Biology* 12.1, e1001745. DOI: 10.1371/journal.pbio.1001745

Introduction

- Nowadays scientific work almost impossible without scientific computing software
- Software as important as hardware in modern science and economics in particular
- Coding is an integral part of economic research, unless purely theoretical (even then software helps checking algebra)
- Development of Information Technologies, a driver for more sophisticated research techniques

Introduction

- Scientist spend 30% or more of their time on developing their own software (Hannay et al., 2009; Prabhu et al., 2011)
- Thus research quality and results highly dependent on developed software
- E.g. Managing and analyzing large amounts of data, running sophisticated regressions or quantitative experiments
- Even running simple OLS regressions requires coding
- Examples of programming languages used in economics: Matlab, Stata, Eviews, R, Fortran, Python, Julia, etc.

Motivation

- As already mentioned, significant time of research spent on developing software
- Thus knowing how to do it right is as important as learning programming.
  - Helps to get more reliable results
  - Decreases the amount of time needed to develop software and boosts the optimality of work
  - Allows for replicability (which increases the validity of the results)
- Mistakes in codes not only dangerous for the quality of the project, but also for those citing it (Domino Effect)

Motivation

- An example for the consequences of mistakes in coding:
  Carmen M. Reinhart and Kenneth S. Rogoff (2010). "Growth in a Time of Debt". In: *American Economic Review* 100.2, pp. 573–78. DOI: 10.1257/aer.100.2.573
- Herndon, Ash, and Pollin, (2014): "We replicate Reinhart and Rogoff (2010) and find that coding errors, selective exclusion of available data, and unconventional weighting of summary statistics lead to serious errors that inaccurately represent the relationship between public debt and GDP growth among 20 advanced economies in the post-war period".
- We wouldn't like to be in a situation like this, would we?

Summary of Best Practices

A list of suggestions relevant for scientific work in the field of economics

1. Write a program for people, not computers
2. Let the computer do the work
3. Make incremental changes
4. Don't repeat yourself
5. Plan for mistakes
6. Optimize software only after it works correctly
7. Document design and purpose, not mechanics
8. Collaborate

1.Write a Program for People

PROGRAMMING FOR NON-PROGRAMMERS
INCLUDE: LIMITED_PROGRAMMING SKILLS
when does it need to get done?
DEFINE: DEADLINE
NOW would be good.
LOAD CODE BY PREVIOUS PERSON ON PROJECT
huge mess
WHILE Project not(FINISHED) input = rand(typing)
SET self_esteem: LOW
I'm a hack!
FOR (time = ∞) DEBUG DEBUG DEBUG
IF code(works) ->EUPHORIA++
woooo!
hello world
SET self_esteem: ABOVE NORMAL
I am... THE ONE.
IF (NOT) organize + comment code for next person on project
yeah right.
JORGE CHAM © 2014
WWW.PHDCOMICS.COM

Figure: http://phdcomics.com/comics/archive.php?comicid=1690

1.Write a Program for People

- Software produced by researchers is intended not only to produce correct output, but also be understandable for others and for the researcher himself when he gets back to the code months later
- Others need to understand the codes so that they can replicate the project.
- Break the program into separate functions that conduct a separate task - easier for others to understand the program as they concentrate on a limited amount of information at a time.
- Be consistent with naming the variables: so they are distinctive and informative. Don't name variables randomly.
- Make code style and formatting consistent: e.g. k_t_ss and cSS_t (c_t_ss) inconsistent, harder to read and understand.

2.Let the Computer Do The Work

- Automate steps to minimize errors from manual work.
    - E.g. when working with data don't cleanse it manually, write a script that loads the raw data automatically and does the cleansing. Useful for understanding how the data has been manipulated.
- Save recent commands. E.g. Matlab has command history - track your previous steps and save them.
- If different software (e.g. Fortran, C, Matlab) is used and interfaced, automate workflows by using a build tool instead of running separate programs and codes manually.

3. Make Incremental Changes

- Work in small steps, understand each step and check its correctness. Planning huge amount of work in advance or making sizable changes in the codes not efficient:
  - This strategy prone to making mistakes (due to cognitive constraints)
  - Debugging - more complicated
  - Program requirements may change, so this method not flexible
- Use a version control system (VCS) especially when collaborating with others
  - Keeps track of all changes in codes, the authors of the changes, their comments.
  - Alerts in case of simultaneous updates of the codes by several co-authors (common Dropbox problem)
  - Allows restoring earlier states of the project easily
- Put all relevant files in the VCS. Makes the project easier to reproduce.

Version Control

- Git (https://git-scm.com/) is probably the main free open-source version control software
- The are good Guided User Interfaces available, e.g. https://tortoisegit.org/ for Windows
- Git interfaces easily with Github https://github.com/, one of the largest software depositories

4. Don't Repeat Yourself

- A single representation for every piece of data in the system. I.e. each parameter defined only once, each data used once and given a unique ID.
- Don't simply copy and paste codes, modularize them: combine them into small groups that implement a single function (e.g. functions in Matlab)
- Re-use codes instead of rewriting them. Relevant when using C or Fortran, where only simple functions are preprogrammed, others need to be programmed manually.

5.Plan for Mistakes

- Mistakes done even by professionals
- According to (McConnell, 2004) and NASA:
    - Industry average experience is about 1 to 25 errors per 1000 lines of code for delivered software
    - Applications Division at Microsoft experiences about 10 to 20 defects per 1000 lines of code during in-house testing, and 0.5 defect per 1000 lines of code in released product
    - Space-shuttle software has achieved a level of 1 defect in 500,000

5.Plan for Mistakes

- Difficult challenge to identify them!
- Assertions a good diagnostic tool, e.g. Matlab *assert* command that checks whether a particular statement holds true at given part of the code.
- Assertions useful as debugging is simplified. Program halts if something goes wrong and assertions highlight the problem.
- Use tests to evaluate your codes. Compare output with simplified cases or results of earlier trusted programs, check whether the output is consistent with your expectations, e.g. plot the output and look at its shape.
- Use a debugger to identify the mistakes. Matlab allows for creating breakpoints that help to implement only some part of the code.

6.Optimize Software Only After It Works Correctly

- Wilson et al., (2014): "The most productive way to make code fast is to make it work correctly".
- First make the code work, then check for the possibilities to speed it up.
- So, write code in a high-level language (like Matlab), then switch to lower-level languages (e.g. Fortran) if computational time can be significantly decreased.
- Even if the optimality of coding with a low level language is known ex ante, the results from coding with a high-level language can serve as a test.

7.Document Design and Purpose, Not Mechanics

- Replicability of a project highly dependent on its documentation, it is also useful when new co-authors added to the project
- Document the reasons and interface (what it does) for a particular code, its inputs and outputs rather than explaining its mechanics
- If a particular part of a code needs substantial explanation, refactor the code and explain parts of it rather than writing a full paragraph explaining how it works

8.Collaborate

- Card and DellaVigna, (2013): "[...] the number of authors per paper [in one of the top-5 journals] has increased from 1.3 in 1970 to 2.3 in 2012"
- Collaboration very important for efficient coding as it allows cross checks – an optimal way for noticing and discarding mistakes.
- Pair programming an extreme way of cooperation when one is coding, and the other checking for mistakes and keeping in mind the larger picture.
- In economics high share of quantitative papers co-authored.

Useful Resources

- Martin Gaudecker's course materials at `http://wiwi.uni-bonn.de/gaudecker/teaching/prog_econ_slides.html`
- Software Carpentry at `https://software-carpentry.org/`

Conclusion

- As already mentioned, organization and optimization of programming procedure highly efficient as it:
  - decreases incidence of mistakes, and those made - easier to find,
  - increases possibility to replicate the project, making it more reliable,
  - makes time spent on writing codes much more efficient.
- Thus as everything else, programming doesn't just need to be done, it needs to be done correctly.
- Follow the rules, optimize your time, make it easier for you and for others.

Thank you for your attention!

Bibliography I

Card, David and Stefano DellaVigna (2013). “Nine Facts about Top Journals in Economics”. In: *Journal of Economic Literature* 51.1, pp. 144–61.

Hannay, Jo Erskine et al. (2009). “How Do Scientists Develop and Use Scientific Software?” In: *Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*. SECSE ’09. Washington, DC, USA: IEEE Computer Society, pp. 1–8. DOI: 10.1109/SECSE.2009.5069155.

Herndon, Thomas, Michael Ash, and Robert Pollin (2014). “Does high public debt consistently stifle economic growth? A critique of Reinhart and Rogoff”. In: *Cambridge Journal of Economics* 38.2, pp. 257–279. DOI: 10.1093/cje/bet075. eprint: http://cje.oxfordjournals.org/content/38/2/257.full.pdf+html.

McConnell, Steve (2004). *Code Complete*. 2nd ed. Microsoft Press.

Bibliography II

Prabhu, Prakash et al. (2011). “A Survey of the Practice of Computational Science”. In: *Proceedings 24th ACM/IEEE Conference on High Performance Computing, Networking, Storage and Analysis*. SC ’11. Seattle, Washington: ACM, 19:1–19:12. DOI: 10.1145/2063348.2063374.

Reinhart, Carmen M. and Kenneth S. Rogoff (2010). “Growth in a Time of Debt”. In: *American Economic Review* 100.2, pp. 573–78. DOI: 10.1257/aer.100.2.573.

Wilson, Greg et al. (2014). “Best Practices for Scientific Computing”. In: *PLoS Biology* 12.1, e1001745. DOI: 10.1371/journal.pbio.1001745.