Effective programming practices for economists

Facilitating reproducible economic research

Hans-Martin von Gaudecker [*]

March 8, 2017

This document provides detailed information on the course with the above title, which I am teaching in the winter term 2016/17. Please read the following pages carefully if you intend to take it, or if you are not sure whether and how you may benefit from it. Should you have any questions left afterwards, do not hesitate to drop by or send me a line at `hmgaudecker[at]uni-bonn[dot]de`

Contents

[*]Universität Bonn, Department of Economics, Lennéstraße 43, 53113 Bonn, Germany, `hmgaudecker[at]uni-bonn[dot]de`, tel. +49 228 73 9357 .

1 Overview

Course title: Effective Programming Practices for Economists

Method (hours per week): Lecture (3) + Tutorials (1)

Course level: PhD (advanced MSc students may also attend)

Course language: English

Prerequisites: Completion of first term in either program

Course description: Many economists spend much of their lives in front of a computer, analysing data or simulating economic models. Surprisingly few of them have ever been taught how to do this well. Class exposure to programming languages is most often limited to mastering {Stata, Matlab, EViews, ...} just well enough in order to perform simple tasks like running a basic regression. However, these skills do not scale up in a straightforward manner to handle complex projects such as a master's thesis, a research paper, or typical work in government or private business settings. As a result, economists spend their time wrestling with software, instead of doing work, but have no idea how reliable or efficient their programs are.

This course is designed to help fill in this gap. It is aimed at Master's and PhD students who expect to write their theses in a field that requires modest to heavy use of computations. Examples include applied microeconomics, econometrics, macroeconomics, computational economics — any field that either involves real-world data; or that does not generally lead to models with simple closed-form solutions.

We will introduce students to programming methods that will substantially reduce their time spent programming while at the same time making their programs more dependable and their results reproducible without extra effort. The course draws extensively on some simple techniques that are the backbone of modern software development, which most economists are simply not aware of. It shows the usefulness of these techniques for a wide variety of economic and econometric applications by means of hands-on examples. More information can be found on `http://www.wiwi.uni-bonn.de/gaudecker/` `/teaching.html`.

Contact person Hans-Martin von Gaudecker, Lennéstraße 43; 53113 Bonn, eMail `hmgaudecker[at]uni-bonn[dot]de`.

Lectures / Tutorials Mondays, 12:15 to 13:45 in Lecture Hall N, Fridays, 8:30 to 10:00 in Lecture Hall N.

2 Detailed course objectives

This course has two distinct but closely intertwined objectives:

1. Providing them with the tools to make their computations reproducible.
2. Enhancing students' programming efficiency.

There is not much reason to explain the first objective to economists beyond a simple calculation: Expect to spend $14 \times 1.5 + 7 \times 1.5 = 31.5$ hours in class and about three times as much working at home, so roughly 120 hours. Let's assume that this course will reduce your time spent programming by 25%. Leaving out capacity constraints during the term, taking this course will be efficient if you expect to spend more than twelve regular working weeks on research that is not paper-reading or pen-and-pencil-theorem-proving. This would include about any PhD that is not pure theory; or an empirical / computational Master's thesis.[1]

Personally, I find objective 1 much more important. The credibility of our profession — which is not remarkably high these days anyhow — is undermined if erroneous results appear in respected journals. To quote McCullough and Vinod (2003):

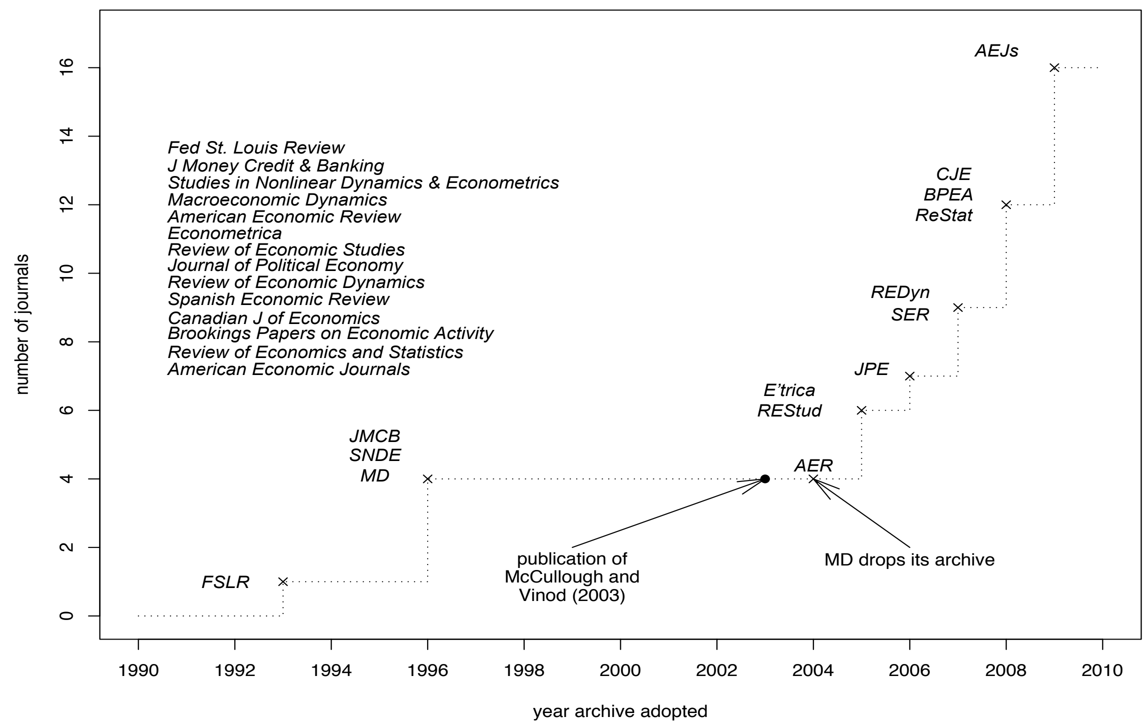
> Replication is the cornerstone of science. Research that cannot be replicated is not science, and cannot be trusted either as part of the profession's accumulated body of knowledge or as a basis for policy. Authors may think they have written perfect code for their bug-free software package and correctly transcribed each data point, but readers cannot safely assume that these error-prone activities have been executed flawlessly until the authors' efforts have been independently verified. A researcher who does not openly allow independent verification of his results puts those results in the same class as the results of a researcher who does share his data and code but whose results cannot be replicated: the class of results that cannot be verified, i.e., the class of results that cannot be trusted.

It is sad if not the substance, but controversies about the replicability of results make it to the first page of the Wall Street Journal (2005, covering the exchange Hoxby (2000) – Rothstein (2007a) – Hoxby (2007) – Rothstein (2007b)). There are some other well-known cases from top journals, see for example Levitt (1997) – McCrary (2002) – Levitt (2002) or the experiences reported in McCullough and Vinod (2003). Assuming that the incentives for replication are much smaller in lower-ranked journals, this is probably just the tip of the iceberg. As a consequence of such issues, many journals have implemented relatively strict replication policies, see Figure 1.

Exchanges such as those above are a huge waste of time and resources. Why waste? Because it is almost costless to ensure reproducibility from the beginning of a project — much is gained by just following a handful of simple rules. They just have to be known. The earlier, the better. From my own experience (Gaudecker, Soest, and Wengström, 2011), I can confirm that replication policies are enforced nowadays — and that it is rather painful to ensure *ex-post* that you can follow them. The number of journals implementing replication policies is likely to grow further — if you aim at publishing in any of them, the 25% time reduction noted above underestimates true savings by several orders of magnitude. And I did not get started on research ethics ...

[1]Similar calculation, more sophisticated: `http://software-carpentry.org/blog/2011/06/doing-the-math.html`

Figure 1: Economic Journals with Mandatory Data + Code Archives



Source: McCullough (2009).

3 Target audience

This course is intended for Master's and PhD students in their third or fifth term. More precisely, you should expect to get started on your first large-scale research project (i.e. your thesis, a computational project module, ...) during or right after the term. Next to your economics background, I will only assume that you have written smaller pieces of code before, like Stata .do-files or Matlab .m-files. Knowledge of a specific programming language is **not** required.

In fact, this course will use Python as an instructional language. Why? Because it is (1) freely available for all operating systems, (2) has numerical abilities closely mirroring those of Matlab but is (3) much more versatile and (4) easily extended with languages such as Fortran or C, which dominate computationally intensive fields. It is **not** a course about Python[2] — but I will use it as an example to teach the core concepts you need. You will be able to apply them in other languages with little transfer.

A fair share of this course is really about tool choice — I will take care in pointing out which language is most appropriate for which problem; but it is more instructive to stick to one language for the course.

[2]Although it is `great for research` and it is becoming more and more widespread in economics. Check out `this great Online course and library for quantitative economics`, for example.

4 Course outline

The outline below is in the correct chronological order, but that the numbering of the topics does not coincide with lectures. Some topics will be shorter, some will be longer than 90 minutes. Also, I will often delay some advanced subtopics to points later in the course, where they fit in more naturally. But I'll try to keep the slides ordered by topic for easier reference.

1. **Introduction**

   (a) **Why reproducibility matters.**

   References: Dewald, Thursby, and Anderson (1986), King (1995), McCullough and Vinod (2003), McCullough (2007), Baiocchi (2007), McCullough (2009), Koenker and Zeileis (2009), Kleiber and Zeileis (2010), LeClere (2010), Merali (2010), Barnes (2010), Yale Law School Roundtable on Data and Code Sharing (2010) Ince, Hatton, and Graham-Cumming (2012)

   (b) **Core concepts.**

   References: Wilson et al. (2014), Wilson (2006), Kelly and Sanders (2008), Kelly, Hook, and Sanders (2009), Segal (2008)

   (c) **Required tools for getting started, installation instructions.**

2. **Math refresher**

   Reference: `http://www.jfsowa.com/logic/math.htm`

3. **Taking low-level control of your machine: The Shell**

   References: Shotts (2011),

4. **A very short introduction to LaTeX.**

   References: Kopka and Daly (2004), Loong (2007), Verfaille (2008)

5. **Organising your workflow and keeping a project's history: Version control, part I: Git on a single machine**

   References: Sink (2011),

6. **Basic programming: Simple data types, conditionals, control flow, functions, modules.**

   References: Sweigart (2015) Koepke (2011) Langtangen (2009), Campbell, Gries, Montojo, and Wilson (2009) Lutz (2007)

7. **Collaborating with others and keeping code on different machines: Version control, part II**

   References: Sink (2011),

8. **Strings, flexible data structures, file input / output**

   References: Langtangen (2009), Campbell, Gries, Montojo, and Wilson (2009) Lutz (2007)

9. **Getting your code to work: Systematic debugging.**

   References: Langtangen (2009), Campbell, Gries, Montojo, and Wilson (2009), Barr (2004)

10. **Scientific tools for Python.**

    References: Langtangen (2009), Campbell, Gries, Montojo, and Wilson (2009) Lutz (2007)

11. **Beyond eMails and Git: More on teamwork with Redmine**

12. **Automating project workflows, sensible project layouts**

    References: Gaudecker (2014) Fomel and Hennenfent (2007), Nagy (2013)

13. **Remembering how your code works: Automatic documentation.**

    References: Brandl (2010)

14. **Object-oriented programming.**

    References: Campbell, Gries, Montojo, and Wilson (2009) Freeman and Pryce (2010)

15. **Making sure your code works: Systematic testing.**

    References: Freeman and Pryce (2010), Dubois (2005) Arbuckle (2010)

16. **Handling economic data**

    References: McKinney (2012)

17. **Speeding Things Up**

18. **Processing strings, regular expressions.**

    References: Friedl (2006), Wilson (2005)

19. **Programming environments and programming languages.**

    References (environments): Doar (2005)
    References (specific languages): Panko (1998/2008), McCullough (2008), Langtangen (2008, 2009), Campbell, Gries, Montojo, and Wilson (2009)

References

Arbuckle, Daniel (2010). *Python Testing: Beginner's Guide*. Birmingham, UK: Packt Publishing.

Baiocchi, Giovanni (2007). "Reproducible Research in Computational Economics: Guidelines, Integrated Approaches, and Open Source Software". In: *Computational Economics* 30.1, pp. 19–40.

Barnes, Nick (2010). "Publish Your Computer Code: It is Good Enough". In: *Nature* 467, p. 753.

Barr, Adam (2004). *Find the Bug. A Book of Incorrect Programs.* Addison Wesley.

Brandl, Georg (2010). “Sphinx Documentation”. Available at `http://sphinx.pocoo.org/`.

Campbell, Jennifer, Paul Gries, Jason Montojo, and Gregory V. Wilson (2009). *An Introduction to Computer Science Using Python.* Ed. by Daniel H. Steinberg. Practical Programming. The Pragmatic Programmers.

Dewald, William G., Jerry G. Thursby, and Richard G. Anderson (1986). “Replication in Empirical Economics: The Journal of Money, Credit and Banking Project”. In: *American Economic Review* 76.4, pp. 587–603.

Doar, Matthew B. (2005). *Practical Development Environments.* Sebastopol, CA, USA: O’Reilly Media.

Dubois, Paul F. (2005). “Maintaining Correctness in Scientific Programs”. In: *Computing in Science & Engineering* 7.3, pp. 80–85.

Fomel, Sergey and Gilles Hennenfent (2007). “Reproducible Computational Experiments Using SCons”. In: *IEEE International Conference on Acoustics, Speech and Signal Processing, 2007. ICASSP 2007.* 4, pp. IV1257–IV1260.

Freeman, Steve and Nat Pryce (2010). *Growing Object-Oriented Software, Guided by Tests.* Boston, MA, USA: Addison Wesley.

Friedl, Jeffrey E. F. (2006). *Mastering Regular Expressions.* 3rd ed. O’Reilly Media.

Gaudecker, Hans-Martin von (2014). “Tutorial: Templates for Reproducible Research Projects”. `http://hmgaudecker.github.io/econ-project-templates/`.

Gaudecker, Hans-Martin von, Arthur van Soest, and Erik Wengström (2011). “Heterogeneity in Risky Choice Behaviour in a Broad Population”. In: *American Economic Review* 101.2, pp. 664–694.

Hoxby, Caroline M. (2000). “Does Competition among Public Schools Benefit Students and Taxpayers?” In: *American Economic Review* 90.5, pp. 1209–1238.

— (2007). “Does Competition among Public Schools Benefit Students and Taxpayers? Reply”. In: *American Economic Review* 97.5, pp. 2038–2055.

Ince, Darrel C., Leslie Hatton, and John Graham-Cumming (2012). “The Case for Open Computer Programs”. In: *Nature* 482.7386, pp. 485–488.

Kelly, Diane, D. Hook, and Rebecca Sanders (2009). “Five Recommended Practices for Computational Scientists who Write Software”. In: *Computing in Science & Engineering* 11.5, pp. 48–53.

Kelly, Diane and Rebecca Sanders (2008). “Assessing the Quality of Scientific Software”. Paper presented at the First International Workshop on Software Engineering for Computational Science and Engineering, Leipzig, Germany.

King, Gary (1995). “Replication, Replication”. In: *PS: Political Science and Politics* 28.3, pp. 444–452.

Kleiber, Christian and Achim Zeileis (2010). "The Grunfeld Data at 50". In: *German Economic Review* 11.4, pp. 404–417.

Koenker, Roger and Achim Zeileis (2009). "On Reproducible Econometric Research". In: *Journal of Applied Econometrics* 24.5, pp. 833–847.

Koepke, Hoyt (2011). "Why Python Rocks for Research". In: *Hacker Monthly* 8. Updated online version avaiable at `http://www.stat.washington.edu/~hoytak/blog/whypython.html`.

Kopka, Helmut and Patrick W. Daly (2004). *Guide to LaTeX*. 4th ed. Addison Wesley / Pearson Education.

Langtangen, Hans Petter (2008). *Python Scripting for Computational Science*. 3rd ed. Berlin and Heidelberg, Germany: Springer.

— (2009). *A Primer on Scientific Programming with Python*. Berlin and Heidelberg, Germany: Springer.

LeClere, Felicia (2010). "Too Many Researchers Are Reluctant to Share Their Data". In: *Chronicle of Higher Education*.

Levitt, Steven D. (1997). "Using Electoral Cycles in Police Hiring to Estimate the Effect of Police on Crime". In: *American Economic Review* 87.3, pp. 270–290.

— (2002). "Using Electoral Cycles in Police Hiring to Estimate the Effects of Police on Crime: Reply". In: *American Economic Review* 92.4, pp. 1244–1250.

Loong, Theresa Song (2007). "The Beginner's Forest of LaTeX". In: *PracTeX Journal* 3.

Lutz, Mark (2007). *Learning Python*. 3rd ed. O'Reilly Media.

McCrary, Justin (2002). "Using Electoral Cycles in Police Hiring to Estimate the Effect of Police on Crime: Comment". In: *American Economic Review* 92.4, pp. 1236–1243.

McCullough, B. D. (2007). "Got Replicability? The *Journal of Money, Credit and Banking Archive*". In: *Econ Journal Watch* 4.3, pp. 326–337.

— (2008). "Special Section on Microsoft Excel 2007". In: *Computational Statistics & Data Analysis* 52.10, pp. 4568–4569.

— (2009). "Open Access Economics Journals and the Market for Reproducible Economic Research". In: *Economic Analysis & Policy* 39.1, pp. 117–126.

McCullough, B. D. and Hrishikesh D. Vinod (2003). "Verifying the Solution from a Nonlinear Solver: A Case Study". In: *American Economic Review* 93.3, pp. 873–892.

McKinney, Wes (2012). *Python for Data Analysis*. O'Reilly Media.

Merali, Zeeya (2010). "Computational Science: ...error". In: *Nature* 467, pp. 775–777.

Nagy, Thomas (2013). *The Waf Book*. Available at `http://code.google.com/p/waf/`.

Panko, Raymond R. (1998/2008). "What We Know about Spreadsheet Errors". In: *Journal of End User Computing* 10.2. Revised version 2008 available at `http://panko.shidler.hawaii.edu/ssr/Mypapers/whatknow.htm`, pp. 15–21.

Rothstein, Jesse (2007a). "Does Competition among Public Schools Benefit Students and Taxpayers? Comment". In: *American Economic Review* 97.5, pp. 2026–2037.

— (2007b). "Rejoinder to Hoxby". Available at `http://gsppi.berkeley.edu/faculty/jrothstein/hoxby/rejoinder.pdf`.

Segal, Judith (2008). "Scientists and Software Engineers: A Tale of Two Cultures." Proceedings of the Psychology of Programming Interest Group, University of Lancaster, UK.

Shotts, William E. (2011). *The Linux Command Line: A Complete Introduction.* San Francisco, CA: No Starch Press.

Sink, Eric (2011). *Version Control by Example.* Champaign, Illinois: Pyrenean Gold Press.

Sweigart, Al (2015). *Automate the Boring Stuff with Python.* San Francisco, CA: No Starch Press.

Verfaille, Vincent (2008). "LaTeX Workshop for Advanced Users... Or at Least Not Beginners!" Available at `http://www.cirmmt.mcgill.ca/activities/workshops/training/latex/details/pdf-slides`.

Wall Street Journal (2005). "Novel Way to Assess School Competition Stirs Academic Row". Available at http://gsppi.berkeley.edu/faculty/jrothstein/hoxby/wsj.pdf.

Wilson, Gregory V. (2005). *Data Crunching.* The Pragmatic Programmers.

— (2006). "Where's the Real Bottleneck in Scientific Computing?" In: *American Scientist* 94.1, pp. 5–6.

Wilson, Gregory V. et al. (2014). "Best Practices for Scientific Computing". In: *PLoS Biology* 12.1, e1001745.

Yale Law School Roundtable on Data and Code Sharing (2010). "Reproducible Research: Addressing the Need for Data and Code Sharing in Computational Science". In: *Computing in Science and Engineering* 12.5, pp. 8–13.