The Practice of Reproducible Research

Case Studies and Lessons from the Data-Intensive Sciences

Justin Kitzes, Daniel Turek, Fatma Deniz (Eds.)

Cite as

Kitzes, J., Turek, D., & Deniz, F. (Eds.). (2017). *The Practice of Reproducible Research: Case Studies and Lessons from the Data-Intensive Sciences.* Oakland, CA: University of California Press.

Table of Contents

Part III: Low-level Case Studies

The Practice of Reproducible Research

Case Studies and Lessons from the Data-Intensive Sciences

Justin Kitzes, Daniel Turek, Fatma Deniz (Eds.)

This is the open, online version of the book *The Practice of Reproducible Research*, to be published in print form by the University of California Press in 2017.

This book contains a collection of 31 case studies of reproducible research workflows, written by academic researchers in the data-intensive sciences. Each case study describes how the author combined specific tools, ideas, and practices in order to complete a real-world research project. Emphasis is placed on the practical aspects of how the author organized his or her research to make it as reproducible as possible.

The Introduction and Part I of the book present general information about working reproducibly and synthesizes common themes from across the case studies. This summary section can be read as a stand alone introduction for beginners wishing to learn more about the general practices of reproducible research. Parts II and III of the book contain the 31 case study chapters themselves.

Please cite *The Practice of Reproducible Research* as:

> Kitzes, J., Turek, D., & Deniz, F. (Eds.). (2017). *The Practice of Reproducible Research: Case Studies and Lessons from the Data-Intensive Sciences*. Oakland, CA: University of California Press.

This book was funded in part by the Gordon and Betty Moore Foundation through Grant GBMF3834 and by the Alfred P. Sloan Foundation through Grant 2013-10-27 to the University of California, Berkeley, as well as by grants from these foundations to the eSciences Institute of the University of Washington and the data sciences initiative at New York University.

Version History

- v1.0.1 - *repro-case-studies* e7134cc, *repro-case-private* 5e63c6e
- v1.0 - *repro-case-studies* d5f5783, *repro-case-private* 5e63c6e

Preface: Nullius in Verba

P.B. Stark

The origins of the scientific method, epitomized by Sir Francis Bacon's work in the early 1600s, amount to insistence on direct evidence. This is reflected in the motto of The Royal Society, founded in 1660: *Nullius in verba*, which roughly means "take nobody's word for it" (The Royal Society, 2016). Fellows of the Royal Society did not consider a claim to be scientifically established unless it had been demonstrated experimentally in front of a group of observers (other fellows), who could see with their own eyes what happened (Shapin & Schaffer, 2011). Over time, Robert Boyle and others developed conventions for documenting experiments in sufficient detail, using prose and illustrations of the apparatus and experimental set up, that the reader could imagine being in the room, observing the experiment and its outcome.

Such observability---visibility into the process of generating results---provides the evidence that the scientific claim is true. It helps ensure we are not fooling ourselves or each other, accidentally or deliberately. It is a safeguard against error and fraud, and a springboard for progress, enabling others to replicate the experiment, to refine or improve the experiment, and to leverage the techniques to answer new questions. It generates and promulgates scientific knowledge *and* the means of generating scientific knowledge.

However, science has largely abandoned that transparency and observability, resulting in a devolution from *show me* to *trust me.* Scientific publications simply do not contain the information needed to know what was done, nor to try to replicate the experiment and data analysis. Peer reviewers and journal editors, the gatekeepers we rely upon to ensure the correctness of published results, cannot possibly vet submissions well, because they are not provided enough information to do the job. There are many reasons for this regression, among them, the rise of Big Science, the size of many modern data sets, the complexity of modern data analysis and the software tools used for data analysis, and draconian limits on the length of articles and even on electronic supplemental material. But as a consequence, most scientific publications provide little scientific evidence for the results they report.

It is impractial or impossible to repeat some experiments from scratch: who can afford to replicate CERN, the Hubble Space Telescope, or the National Health and Nutrition Examination Survey? Some data sets are too large to move efficiently, or contain information restricted by law or ethics. Lack of access to the underlying data obviously makes it impossible to replicate data analysis. But even when the data are available, reliance on

proprietary software or point-and-click tools and failure to publish code make it impossible to know exactly what was done to the data to generate the figures and tables in most scientific publications.

The (unfortunately rare) attempts to replicate experiments or data analyses often fail to support the original claims (Lehrer, 2010; Open Science Collaboration, 2015) Why?

One reason is the interaction between scientific publishing and statistics. Because journals are generally uninterested in publishing negative results or replications of positive results, the emphasis is on "discoveries." Selecting data, hypotheses, data analyses, and results to produce (apparently) positive results inflates the apparent signal-to-noise ratio and overstates statistical significance. The ability to automate many aspects of data analysis, such as feature selection and model selection, combined with the large number of variables measured in many modern studies and experiments, including "omics," high-energy physics, and sensor networks, make it essentially inevitable that many "discoveries" will be wrong (Ioannidis, 2005). A primary defense against being misled by this selection process, which includes *p*-hacking and the "file-drawer effect" (Nuzzo, 2015; Rosenthal, 1979), is to insist that researchers disclose what they tried before arriving at the analysis they chose to report or to emphasize.

I would argue that if a paper does not provide enough information to assess whether its results are correct, it is something other than science. Consequently, I think scientific journals and the peer review system must change radically: referees and editors should not "bless" work they cannot check because the authors did not provide enough information, including making available the software used to analyze the data. And scientific journals should not publish such work.

A crucial component of the chain of evidence is the software used to process and analyze the data. Modern data analysis typically involves dozens, if not hundreds of steps, each of which can be performed by numerous algorithms that are nominally identical but differ in detail, and each of which involves at least some ad hoc choices. If researchers do not make their code available, there is little hope of ever knowing what was done to the data, much less assessing whether it was the right thing to do.

And most software has bugs. For instance, a 2014 study by Coverity, based on code-scanning algorithms, found 0.61 errors per 1,000 lines of source code in open-source projects and 0.76 errors per 1,000 lines of source code in commercial software (Synopsys, 2015). Scientific software is not an exception, and few scientists use sound software engineering practices, such as rigorous testing---or even version control (Merali, 2010; Soergel, 2015). Using point-and-click tools, rather than scripted analyses, makes it easier to commit errors and harder to find them. One recent calamity attributable in part to poor computational practice is the work of Reinhart and Rogoff (2010), which was used to justify economic austerity measures in southern Europe. Errors in their Excel spreadsheet led to

the wrong conclusion (Herndon & Pollin, 2014). If they had scripted their analysis and tested the code instead of using spreadsheet software, their errors might have been avoided, discovered, or corrected before harm was done.

Working reproducibly makes it easier to get correct results and enables others to check whether results are correct. This volume focuses on how researchers in a broad spectrum of scientific applications document and reveal what they did to their data to arrive at their figures, tables, and scientific conclusions; that is, how they make the computational portion of their work more transparent and reproducible. This enables others to assess crucial aspects of the evidence that their scientific claims are correct, and to repeat, improve, and repurpose analyses and intellectual contributions embodied in software artifacts. Infrastructure to make code and data available in useful forms needs more development, but much is possible already, as these vignettes show. The contributors share how their workflows and tools enable them to work more transparently and reproducibly, and call out "pain points" where new tools and processes might make things easier. Whether you are an astrophysicist, an ecologist, a sociologist, a statistician, or a nuclear engineer, there is likely something between these covers that will interest you, and something you will find useful to make your own work more transparent and replicable.

References

Herndon, M. A., T., & Pollin, R. (2014). Does high public debt consistently stifle economic growth? A critique of reinhart and rogoff. *Cambridge Journal of Economics*, *38*, 257–279.

Ioannidis, J. (2005). Why most published research findings are false. *PLoS Medicine*, *2*(8), e124.

Lehrer, J. (2010). The truth wears off. *The New Yorker*. Retrieved from http://www.newyorker.com/magazine/2010/12/13/the-truth-wears-off

Merali, Z. (2010). Computational science: . . . Error . . . why scientific programming does not compute. *Nature*, *467*, 775–777.

Nuzzo, R. (2015). How scientists fool themselves – and how they can stop. *Nature*, *526*, 182–185.

Open Science Collaboration. (2015). Estimating the reproducibility of psychological science. *Science*, *349*, 943.

Reinhart, C., & Rogoff, K. (2010). Growth in a time of debt. *American Economic Review*, *100*, 573–578.

Rosenthal, R. (1979). The “file drawer problem” and tolerance for null results. *Psychological Bulletin*, *86*(3), 638–641.

Shapin, S., & Schaffer, S. (2011). *Leviathan and the air-pump: Hobbes, boyle, and the experimental life*. Princeton, NJ: Princeton University Press.

Soergel, D. (2015). Rampant software errors may undermine scientific results. *F1000Research*, *3*, 303.

Synopsys. (2015). Coverity scan open source report 2014. Retrieved from http://go.coverity.com/rs/157-LQW-289/images/2014-Coverity-Scan-Report.pdf

The Royal Society. (2016). The royal society | history. Retrieved from https://royalsociety.org/about-us/history/

Introduction

Justin Kitzes

Think back to the first laboratory science course that you ever took, perhaps a high school or an undergraduate chemistry or biology lab. Imagine sitting down on the first day, in a new room, surrounded by new classmates, in front of a new teacher, and encountering all of the strange sights and smells around you. Perhaps there were jars containing strange substances along the walls, oddly shaped glass and metal equipment, and safety gear to protect you from some mysterious danger.

As you entered this new physical and intellectual environment, preparing to learn the foundational knowledge and skills of a new field of science, what was the first thing that you were taught? Whatever it was, we suspect that it was *not* chemistry or biology. For most of us, the first instructions in a lab course were about how to perform basic tasks like cleaning the equipment, zeroing a balance, labeling a beaker, and recording every step that you performed in a lab notebook.

What did all of these seemingly menial tasks have to do with the science that you were supposed to be learning? Although it may not have been clear right away, these steps were all designed to ensure that, when you did conduct an experiment, you would be confident in the accuracy of your results and be able to clearly communicate what you did to someone. Together, these two factors would permit someone else to perform your same experiment and achieve the same result, verifying your findings. None of your actual experimental results would have been meaningful, or useful to others, had you not followed these basic procedures and principles.

Now jump forward again to the present, and consider the type of research work that you do today. Almost certainly, you are using methods, tools, and equipment that are significantly more complex than those that you encountered in your first lab experience. If you are like most scientists today, your research is also slowly, or not so slowly, shifting away from the traditional "lab bench" of your discipline and into the rapidly expanding world of scientific computing. There is scarcely a scientific discipline today that is not being rapidly transformed by an infusion of new hardware, software, programming languages, large messy data sets, and complex new methods for data analysis.

Unfortunately, however, many excellent and accomplished scientists never received even high school or undergraduate-level training in these types of computing skills. Many of us struggle along as best we can, trying to write scripts, work with uncomfortably large data

sets, make correctly formatted figures, write and edit papers with collaborators, and somehow not lose track of which data and which analysis led to what result along the way. These are difficult tasks for someone well-versed in scientific computing, much less for scientists who are trying to pick up these skills on the fly from colleagues, books, and workshops.

In one sentence, this book is about **how to take the basic principles of the scientific method that you learned at the lab bench and translate them to your laptop.** Its core goal is to provide concrete advice and examples that will demonstrate how you can make your computational and data-intensive research more clear, transparent, and organized. We believe that these techniques will enable you to do better science, faster, and with fewer mistakes.

Within the world of scientific computing practice, the techniques that we explore in this book are those that support the goal of *computational reproducibility*. For the purposes of this book, we define computational reproducibility as follows:

Computational reproducibility is the ability of a second investigator (including you in the future) to recreate the finished results of a study, including key quantitative findings, tables, and figures, given only a set of files related to the research project.

Thinking back to that first lab course, this would be equivalent to handing a textual writeup, a stack of equipment, and some raw materials to a classmate and asking them to arrive at the same findings that you did.

There are many reasons why we believe that that practicing computational reproducibility is perhaps the key foundational skill for scientific computing. Perhaps most importantly, working towards computational reproducibility will indirectly require you to follow many general scientific best practices for all of your digital analyses, including recording all steps in your research process, linking a final result back to the initial data and other inputs that generated it, and making all necessary data and inputs available to your colleagues.

Additionally, thinking explicitly about computational reproducibility helps to move the focus of research up a level from individual activities to the higher level of the entire scientific workflows. This change in perspective is becoming increasingly important as our work becomes so complex that the overarching grand perspective is not always obvious.

Finally, the computational reproducibility of an individual research project can often be substantially increased or decreased by an individual investigator, meaning that the skills that we will discuss in this book can immediately be put into practice in nearly all types of research projects. This level of control contrasts, for example, with more complex issues such as scienitific replicability (see Chapters 2 and 38), which are more heavily dependent on coordination among many scientists or on institutional actions.

This book is designed to demonstrate and teach how many of today's scientists are striving to make their research more computationally reproducible. The research described in this volume spans many traditional academic disciplines, but all of it falls into what may be called the data-intensive sciences. We define these fields as those in which researchers are routinely expected to collect, manipulate, and analyze large, heterogeneous, uncertain data sets, tasks that generally require some amount of programming and software development. While there are many challenges to achieving reproducibility in other fields that rely on fundamentally different research methods, including the social sciences and humanities, these approaches are not covered here.

This book is based on a collection of thirty-one contributed case study chapters, each authored by a leader in data-intensive research. Each case study presents the specific approach that the author used to attempt to achieve reproducibility in a real-world research project, including a discussion of the overall project workflow, key tools and techniques, and major challenges. The authors include both junior and senior scholars, ranging from graduate students to full professors. Many of the authors are affiliated with one of three Data Science Environments, housed at the University of California Berkeley, the University of Washington, and New York University. We are particularly grateful to the Gordon and Betty Moore Foundation and the Alfred P. Sloan Foundation for supporting these environments, which provided the intellectual space and financial support that made this book possible.

In addition to these contributed case studies, this book also includes synthesis chapters that introduce, summarize, and synthesize the best practices for data-intensive reproducible research. Part I of the book introduces several important concepts and practices in computational reproducibility and synthesizes lessons learned from the thirty-one case studies. In Chapter 2, *Assessing the Reproducibility of a Research Project*, Rokem, Marwick, and Staneva outline the factors that determine the extent to which a research project is computationally reproducible. In Chapter 3, *The Basic Reproducible Workflow Template*, Kitzes provides a step-by-step illustration of a core, cross-disciplinary reproducible workflow, suitable as a standalone first lesson for beginners or as a means of contextualizing the case study chapters for more advanced readers.

These preliminary discussions are followed by Chapter 4, Turek and Deniz's *Case Studies in Reproducible Research*, which describes the format of the contributed case studies and summarizes some of their key features. In Chapter 5, *Lessons Learned*, Huff synthesizes and discusses common themes across the case studies, focusing on identifying the tools and practices that brought scientists most reproducibility benefit per unit effort and the universal challenges in achieving reproducibility. Ram and Marwick's Chapter 6, *Building Towards a Future Where Reproducible, Open Science is the Norm*, includes a broad discussion of reproducible research in modern science, highlighting the gaps, challenges,

and opportunities going forward. Finally, an extended *Glossary* by Rokem and Chirigati in Chapter 7 defines, describes, and discusses key concepts, techniques, and tools used in reproducible research and mentioned throughout the case studies.

Part I of the book can be read as a standalone introduction to reproducible research practices in the data-intensive sciences. For readers wishing to learn more about the details of these practices, Part II and Part III of the book contain the thirty-one contributed case study chapters themselves, divided into high-level case studies that provide a description of an entire research workflow, from data acquisition through analysis (Part II), and low-level case studies that take a more focused view on the implementation of one particular aspect of a reproducible workflow (Part III).

This book unavoidably assumes some background on the part of readers. To make best use of this book, you should have some experience programming and writing short scripts, perhaps a few dozen lines of code, to analyze some data set. If you are not yet comfortable with this task, many good books and courses on basic programming skills in a scientific context are currently available. We would particularly recommend the online lessons and in-person trainings provided by the groups Software Carpentry and Data Carpentry. In addition to basic programming, we presume that you have at least some familiarity with the basic principles of scientific research, and that you are either a published author of peer-reviewed papers yourself or are aspiring to be one shortly.

For those who are relatively new to computational research and reproducibility, we suggest beginning by carefully reading the chapters in Part I of the book and attempting to work along with the basic workflow template described in Chapter 3, either exactly as presented or as adapted to a new research project of your own choosing. The case study chapters can then be skimmed, with particular attention paid to the high-level workflows in Part II. Chapter 7, the extended glossary, should be referred to regularly when encountering unfamiliar terms and concepts.

For those with more experience in computational research, particularly those who are interested in adapting and advancing their own existing research practices, we recommend focusing first on Chapter 4, *Case Studies in Reproducible Research* and then reviewing all of the case studies chapters themselves. We suggest reading the high-level case studies first, followed by the low-level case studies, with an eye towards identifying particular strategies that may be applicable to your own research problems. The *Lessons Learned* and *Supporting Reproducible Research* chapters will be useful in providing a synthesis of the current state of reproducible research and prospects and challenges for the future.

Regardless of your current background and skill set, we believe that you will find both inspiration and concrete, readily-applicable techniques in this book. It is always important to remember that reproducibility is a matter of degrees, and these examples will demonstrate

that while achieving full reproducibility may sometimes be difficult or impossible, much can be gained from efforts to move a research project incrementally in the direction of reproducibility.

Let's get started.

Assessing Reproducibility

Ariel Rokem, Ben Marwick, and Valentina Staneva

While understanding the full complement of factors that contribute to reproducibility is important, it can also be hard to break down these factors into steps that can immediately be adopted into an existing research program and immediately improve its reproducibility. One of the first steps to take is to assess the current state of affairs, and to track improvement as steps are taken to increase reproducibility even more. This chapter provides a few key points for this assessment.

What it means to make research reproducible

Although one of the objectives of this book is to discover how researchers are defining and implementing reproducibility for themselves, it is important at this point to briefly review some of the current scholarly discussion on what it means to strive for reproducible research. This is important because recent surveys and commentary have highlighted that there is confusion among scientists about the meaning of reproducibility (Baker, 2016b, 2016a). Furthermore, there is disagreement about how to define 'reproducible' and 'replicable' in different fields (Casadevall & Fang, 2010; Drummond, 2009; Easterbrook, 2014; Stodden, Borwein, & Bailey, 2013). For example, Goodman, Fanelli, & Ioannidis (2016) note that in epidemiology, computational biology, economics, and clinical trials, reproducibility is often defined as:

> the ability of a researcher to duplicate the results of a prior study using the same materials as were used by the original investigator. That is, a second researcher might use the same raw data to build the same analysis files and implement the same statistical analysis in an attempt to yield the same results.

This is distinct from replicability:

> which refers to the ability of a researcher to duplicate the results of a prior study if the same procedures are followed but new data are collected.

It is noteworthy that definitions above, which are broadly consistent with usage of these terms throughout this book, are totally opposite to the Association for Computing Machinery (ACM, the world's largest scientific computing society), which take their definitions from the International Vocabulary of Metrology. Here are the ACM definitions:

> Reproducibility (Different team, different experimental setup) The measurement can be obtained with stated precision by a different team, a different measuring system, in a different location on multiple trials. For computational experiments, this means that an independent group can obtain the same result using artifacts which they develop completely independently.
>
> Replicability (Different team, same experimental setup) The measurement can be obtained with stated precision by a different team using the same measurement procedure, the same measuring system, under the same operating conditions, in the same or a different location on multiple trials. For computational experiments, this means that an independent group can obtain the same result using the author's own artifacts.

We can see the heritage of the definitions of the ACM in literature on physics and the philosophy of science (Cartwright, 1991; Collins, 1984; Franklin & Howson, 1984). In her paper on the epistemology of scientific experimentation, Cartwright (1991), presents one of the first clear definitions of the key terms: 'replicability - doing the same experiment again' and 'reproducibility - doing a new experiment'.

The definition of Cartwright is at odd with our preferred definition, from Goodman et al. (2016). This is because we trace a different ancestry in the use of the term 'reproducible', one that recognises the central role of the computer in scientific practice, with less emphasis on empirical experimentation as the primary means for generating knowledge. Among the first to write about reproducibility in this way is geophysicist Jon Claerbout. He pioneered the use of the phrase 'reproducible research' to describe how his seismology research group used computer files to enable efficient regeneration of the figures and tables in theses and publications (Claerbout & Karrenfach, n.d.). We can see this usage more recently in Stodden, Leisch, & Peng (2014):

> Replication, the practice of independently implementing scientific experiments to validate specific findings, is the cornerstone of discovering scientific truth. Related to replication is reproducibility, which is the calculation of quantitative scientific results by independent scientists using the original datasets and methods. Reproducibility can be thought of as a different standard of validity because it forgoes independent data collection and uses the methods and data collected by the original investigator. Reproducibility has become an important issue for more recent research due to advances in technology and the rapid spread of computational methods across the research landscape.

It is this way of thinking about reproducibility that captures most of the variation in the way the contributors to this book use the term. One of the key ideas that the remainder of this chapter explores is that reproducibility is a matter of degree, rather than kind. This means that identifying the factors that can relatively easily and quickly be changed can

incrementally lead to an increase in the reproducibility of a research program. Identifying more challenging points, that would require more work, helps set long-term goals towards even more reproducible work, and helps identify practical changes that can be made over time.

Reproducibility can be assessed at several different levels: at the level of an individual project (e.g., a paper, an experiment, a method or a dataset), an individual researcher, a lab or research group, an institution, or even a research field. Slightly different kinds of criteria and points of assessment might apply to these different levels. For example, an institution upholds reproducibility practices if it institutes policies that reward researchers who conduct reproducible research. Meanwhile, a research field might be considered to have a higher level of reproducibility if it develops community-maintained resources that promote and enable reproducible research practices, such as data repositories, or common data-sharing standards.

This book focuses on the first of these levels, that of a specific research project. In this chapter we consider some of the ways that reproducibility can be assessed by researchers who might be curious about how they can improve their work. We have divided this assessment of reproducibility into three different broad aspects: automation and provenance tracking, availability of software and data, and open reporting of results. For each aspect we provide a set of questions to focus attention on key details where reproducibility can be enhanced. In some cases we provide specific suggestions about how the questions could be answered, where we think the suggestions might be useful across many fields.

The diversity of standards and tools relating to reproducible research is large and we cannot survey all the possible options in this chapter. We recommend that the researcher use the detailed case studies in following chapters for inspiration, tailoring choices to the norms and standards of your discipline.

Automation and provenance tracking

Automation of the research process means that the main steps in the project: transformations of the data -- various processing steps and calculations -- as well as the visualization steps that lead to the important inferences, are encoded in software and documented in such a way that they can reliably and mechanically be replicated. In other words, the conclusions and illustrations that appear in the article are the result of a set of computational routines, or scripts that can be examined by others, and re-run to reproduce these results.

To assess the sufficiency of automation in a project, one might ask:

- Can all figures/calculations that are important for the inference leading to the result be reproduced in a single button press? If not a single button press, can these be produced with a reasonably small effort? One way to achieve this goal is to write software scripts that embody every step in the analysis up to the creation of figures, and derivation of calculations. In assessment, you can ask: is it possible to point to the software script (or scripts) that generated every one of the calculations and data visualizations? Is it possible to run these scripts with reasonably minimal effort?
- Another set of questions refers to the starting point of the calculations in the previous question: what is the starting point of running these scripts? What is required as set-up steps to the calculations in these scripts? If the setup includes manual processing of data, or cumbersome setup of a computational environment, this detracts from the reproducibility of the research.

The main question underlying these criteria is how difficult it would be for another researcher to first reproduce the results of a research project, and then further build upon these results. Because research is hard, and error is ubiquitous (a point made in this context by Donoho and colleagues (2008)), the first person to benefit from automation is often the researcher performing the original research, when hunting down and eliminating error.

Provenance tracking is very closely related to automation (see glossary for definitions). It entails that the full chain of computational events that occurred from the raw data to a conclusion is tracked and documented. In cases in which automation is implemented, provenance trackingcan be instantiated and executed with a reasonably minimal effort.

When large data sets and complex analysis are involved, some processing steps may consume more time and computational resources than can be reasonably required to be repeatedly executed. In these cases, some other form of provenance tracking may serve to bolster reproducibility, even in the absence of a fully automatic processing pipeline. Items for assessment here are:

- If software was used in (pre)processing the data, is this software properly described? This includes documentation of the the version of the software that was used, and the settings of parameters that were used as inputs to this software.
- If databases were queried, are the queries fully documented? Are dates of access recorded?
- Are scripts for data cleaning included with the research materials, and do they include commentary to explain key decisions made about missing data and discarding data?

Another aspect of provenance tracking is the tracking of different versions of the software, and recording of the evolution of the software, including a clear delineation of the versions of the software that were used to support specific scientific findings. This can be assesessed

by asking:

- Is the evolution of the software available for inspection through a publicly accessible version control system? Are versions that contributed to particular findings clearly tagged in the version control history?

Availability of software and data

The public availability of the data and software are key components of computational reproducibility. To facilitate its evaluation, we suggest that researchers consider the following series of questions.

Availability of data

- Are the data available through an openly accessible database? Often data is shared through the internet. Here, we might ask about the long-term reliability of the web address: are the URLs mentioned in a manuscript permanently and reliably assigned to the dataset? One example of a persistent URL is a Digital Object Identifier (DOI). Several major repositories provide these for data sets (e.g., Figshare). Datasets accessible via persistent URLs increase the reproducibility of the research, relative to use of an individually maintained website, such as a lab group website or a researcher's personal website. This is because when an individually maintained websites changes its address or structure over time, the previously published URLs may no longer work. In many academic institutions, data repositories that provide persistent URLs are maintained by the libraries. These data repositories provide a secure environment for long-term citation, access, and reuse of research data.
- Are the data shared in a commonly used and well-documented file format? For tabular data, open file formats based on plain text, such as CSV (comma separated values) or TSV (tab separated values) are often used. The main benefit of text-based formats is their simplicity and transparency. On the other hand, they suffer from a loss of numerical precision, they are relatively large, and parsing them might still be difficult. Where available, strongly-typed binary formats should be preferred. For example multi-dimensional array data can be stored in formats such as HDF5. In addition, there are also open data formats that have been developed in specific research communities to properly store data and metadata relelvant to the analysis of data from this research domain. Examples include the FITS data format for astronomical data (Wells, Greisen, & Harten, 1981), and the NIFTI and DICOM file formats for medical imaging data (Larobina & Murino, 2014).

Proprietary file formats are problematic for reproducibility because they may not be usable on future computer systems due to intellectual property restrictions, obsolescence or incompatibility. However, one can still ask: if open formats are not suitable, is software provided to read the data into computer memory with reasonably minimal effort?

- If community standards exist, are files laid out in the shared database in a manner that conforms with these standards? For example, for neuroimaging data, does the file layout follow the Brain Imaging Data Structure (Gorgolewski et al., 2016) format?
- If data are updated, are different versions of the data clearly denoted? If data is processed in your analysis, is the raw data available?
- Is sufficient metadata provided? The type and amount of metadata varies widely by area of research, but a minimal set might include the research title, authors' names, description of collection methods and measurement variables, date, and license.
- If the data are not directly available, for example if the data are too large to share conveniently, or have restrictions related to privacy issues, do you provide sufficient instructions to obtain equivalent data? For example, are the experimental protocols used to acquire the original data sufficiently detailed?

Availability of software

- Is the software available to download and install? Software can also be deposited at repositories that issue persistent URLs, just like data sets. This can improve the longevity of its accessibility.
- Can the software easily be installed on different platforms? If a scripting language such as Python or R was used, it is better for reproducibility to share the source rather than compiled binaries that are platform-specific.
- Does the software have conditions on the use? For example, license fees, restrictions to academic or non-commercial use, etc.
- Is the source code available for inspection?
- Is the full history of the source code available for inspection through a publicly available version history?
- Are the dependencies of the software (hardware and software) described properly? Do these dependencies require only a reasonably minimal amount of effort to obtain and use? For example, if a research project requires the use of specialized hardware, it will be harder to reproduce. If it depends on expensive commercial software, likewise. Use of open-source software dependencies on commodity hardware is not always possible, but when possible electing to use these increases reproducibility.

Software Documentation

Documentation of the software is another factor in removing barriers to re-use. Several forms of documentation can be added to a research repository and each of them adds to reproducibility. Relevant questions include:

- Does the software include a README file? This provides information about the purpose of the software, its use and ways to contact the authors of the software (see more below).
- Is there any function/module documentation? This closely explains the different parts of the code, including the structure of the modules that make up the code; the inputs and outputs of functions; the methods and attributes of objects, etc.
- Is there any narrative documentation? This explains how the bits and pieces of the software work together; narrative documentation might also explain how the software should be installed and configured in different circumstances and can explain what order things should be executed.
- Are there usage examples? This is particularly important for scientific computing, usage examples demonstrate the kinds of transformations, analysis pipelines and visualizations that can be undertaken using the software, and provide a point of departure for new explorations using the software. Systems that allow examples to be routinely run as part of compiling the documentation are particularly useful, because they are automatically updated when the code is updated. One such system that was originally developed as part of the PyMVPA software library (Hanke et al., 2009) has been widely adopted and further developed by many other scientific Python libraries, including scikit-image (Van Der Walt et al., 2014) and scikit-learn (Pedregosa et al., 2011) and is now its own software project.

Software engineering

While not all scientific software needs to apply rigorous software engineering practices, using these practices increases the reproducibility and long-term sustainability of the software, and enables expansion of the software to handle extensions of the work. While a full implementation of these practices may be challenging for smaller projects, an awareness of the problems they are intended to solve can lead to better practices in other areas of the software development process. A few guidelines for assessing the software engineering of a computational research codebase follow.

Software testing is a practice that automates the checking of smaller units of the code, in addition and in support of the automation of the full pipeline, described above (see glossary for a detailed definition and typology of software testing). Questions that can be used to

assess the testing of the code include:

- Is a large proportion of the code covered by automatic testing that verifies that the software runs properly and is relatively error-free? Analysis software is often developed to deal with cases that are common in the data analyzed, and it often implicitly embodies assumptions about these common cases. However, some unusual cases (also called "corner cases" or "edge cases") may appear in the data, and it is important for the software to produce correct results in these cases as well. One might therefore ask: are corner cases covered, in addition to the common cases?
- Is a continuous integration system set up to validate the mechanisms for software installation and to run the full complement of tests? Does this system regularly update the software dependencies, such that the software properly runs on newer versions of these dependencies? Is the system set up to maintain backwards compatibility with older versions of these dependencies, in support of dependent developments?

Further open-source and software engineering practices can help support a community of users. These include:

- Semantic versioning is a way to communicate about the development of the software, and to allow others to depend on it. Is the software regularly released under a semantic versioning scheme? Are releases communicated widely to the user community? When standard installation channels exist, such as package managers (e.g., apt-get, pip) and repositories (e.g. CRAN, PyPi) exist, are new versions of the software made available through these mechanisms?
- Are there mechanisms in place to report and track bugs in the software? When bugs are fixed, are these fixes reported in release announcements?
- While private communication can be used to help individual users of the software, these modes of communication do not scale very well to a larger community of users. Requiring such private communication sets up barriers for users to reproduce the work. Setting up a public communication channel for users of the software to ask questions about use of the software increases the reproducibility. These can include public mailing lists, forums and/or chat rooms.

Copyright issues and other data encumbrances

Creative work, such as research, is protected by copyright laws. While these laws protect the rights of creators and researchers, they can also impede the distribution and verification of their work. Work that has no license or copyright information is still protected by copyright

law. This prevents others from having any rights to reproduce the work or build upon it. Therefore, the application of an appropriate license is important in increasing the reproducibility of the work.

Data and copyright. While copyright law doesn't generally protect facts or data, it does protect the creative acts that go into selection of the data that goes into a database or compilation. To remove doubt about the copyright status of data, a license needs to be chosen. To assess reproducibility, you can ask:

- If the data is openly accessible to others, is it released under a license that would allow them to use it?
- Is the license permissive enough to allow others to build upon the work and extend it?

One set of licenses that allows data providers to control what potential users of the data may do with the data are the Creative Commons licenses, and open licenses designed specifically for data sharing (Miller, Styles, & Heath, 2008). Stodden (2009) recommends the CC-0 (public domain) license for data to enable maximum flexibility for reuse.

Software and copyright The same questions apply to issues related software and copyright, with slight variations: When sharing the source code of the software for free, researchers are encouraged to provide a license which clarifies the conditions under which this code can be used, without infringing on the copyright of the author. A license which allows anybody to use the software, alter it, build upon it, include it in other software packages, and extend it facilitates reproducibility.

Permissive software licenses would allow all of the above with minimal restrictions (e.g., BSD license, MIT license). BSD licenses are unique in including a specific clause which prevents the use of the name of the software author in future derivatives, which protects the author from negative effects of unwarranted use of their software.

Copyleft licenses allow distribution and modification of the software, but require they are released under the same license. For example, if the original software is open source and free, all its copies and derivatives should be open source and free. Such license clearly restricts the use of the software within proprietary applications. For example, software developed in an academic context with a copyleft license could not be used as part of a commercial package. The GNU General Public Licence (GPL) is an example of a popular copyleft license.

- Does the software have an open-source license?
- Is this license sufficiently permissive to allow others to use the software, reproduce the results and extend them?

Proprietary information and software.

Often authors may not make the data or software available due to external restrictions. We might ask the following questions to assess the effect these restrictions might have on reproducibility:

- Is the availability and use of the data encumbered through proprietary, privacy or ethical restrictions? (For example, due to presence of sensitive personal information, or customer activity records.).
- Are there trade restrictions, or issues of national security that prohibit the open distribution of the data?
- Is the software closed-source or limited in its accessibility due to funding regulations (governmental restrictions, industrial sponsor requirements, etc.)?

Although these conditions obviously limit the degree of reproducibility that might be possible, there are options to improve the reproducibility of this kind of research. For example, a simulated dataset can sometimes be provided that mimics the key attributes of the real dataset. Where the software is restricted, authors are encouraged to provide sufficient information about key algorithms so that future studies might be executed on openly available data with more accessible software.

Open reporting of results

Crucial to reproducing a study is providing sufficient details about its execution through reports, papers, lab notebooks, etc. Researchers usually aim to publish their results in journals (or conference proceedings) with the aim to broadly distribute their discoveries. However, the choice of a journal may affect the availability and accessibility of their findings. Open access journals allow readers to access articles (usually online) without requiring any subscription or fees. While open access can take many forms, there are two common types of open access publication:

green access - the journal charges a subscription fee to readers for access to its contents, but allows the author to post a version of their article (preprint/postprint) on an electronic print website such as arXiv, EPrints Archive, on their own website, or on a institutional repository.

gold access - the journal does not charge any fees to readers, and makes a freely accessible online version of the article available at the time of publishing. Usually the author pays an article processing charge to enable free access by readers.

Clearly gold access journals provide the easiest and most reliable access to the article. However, since there are no subscription fees to cover publishing costs at gold open access journals and articles, the author is required to pay. Often the amount is over a thousand dollars per article. Authors should check with their institution whether it provides funds for

covering such fees. As a compromise, journals sometimes have an embargo on open access (delayed open access), i.e. there is a period of time during which the article cannot be freely accessed, after which either the journal automatically makes the article available or the authors are allowed to self-archive it.

Green open access is an attractive approach to making articles openly available because it is affordable to both readers and authors. According to a study of a random sample of articles in 2009 (Björk, Welling, Laakso, Majlender, & Guðnason, 2010), approximately 20% of the articles were freely accessible (9.8 % on publishers' websites and 11.9% elsewhere through search). A more recent larger study (Archambault et al., 2013) indicates that 43% of Scopus indexed papers between 2008 and 2011 were freely available by the end of 2012. It has been also shown that there is a substantial growth in the proportion of available articles. However, there are still many articles which have been given a green light for access, but they have not been self-archived. Thus it is important for authors to understand the journal's publishing policy and use the available resources (within their field, institution, and beyond) to make their work accessible to a wide audience. Many research-intensive universities, usually via the libraries, provide services to help researchers self-archive their publications.

There are many other methods for sharing research online at different stages of the work (before final results are even available). Preregistration of the hypotheses that are being tested in a study can prevent overly flexible analysis practices and HARKing (hypothesizing after results are known (Kerr, 1998)), which reduce the reproducibility of the results reported. Regular public updates can be achieved through electronic lab notebooks, wiki pages, presentation slides, blog posts, technical reports, preprints, etc. Sharing progress allows for quick dissemination of ideas, easy collaboration, and early detection and correction of flaws. Storing preliminary results and supplementary materials in centralized repositories (preregistration registries, public version control repositories, institutional reports) have potential to improve the discoverability and the availability lifespan of the works. Some important questions researchers can ask when evaluating publishing solutions include:

- Is this electronic publishing platform going to be available in 2 years? In 5 years? Longer?
- Can a simple web search on the topic recover a link to the publication and related materials?

Taking into account the sustainability and the ease of access of these solutions in the decision process is integral to improving the research reproducibility. There is also empirical evidence that publication in open access promotes the downstream use of the scientific findings, as evidenced by an approximately 10% increase in citations (Hajjem, Harnad, & Gingras, 2006) (and see also http://opcit.eprints.org/oacitation-biblio.html).

Conclusion

This chapter has attempted to outline the factors that determine the extent to which a research project is computationally reproducible. We have surveyed three different aspects where reproducibility can be assessed: automation and provenance tracking, availability of software and data, and open reporting of results. For each topic we provide a set of questions for researchers to consider about their own work and stimulate discussion on how computationally reproducibility can be improved. There are many more questions that could be asked, but we have tried to confine ourselves to questions that are relevant to key hurdles in improving reproducibility. We have observed these questions to be key points in making our own work more reproducible, and in assessing the work of our peers.

A key theme of this chapter is that there are many degrees of reproducibility. Computational reproducibility exists on a long spectrum from completely irreproducible research to complete computational reproducibility, with data, software and results all available for scrutiny, use and further exploration. Our hope is that by raising these questions and discussing some of the options, researchers can identify ways to move their work a little further along the spectrum towards improved reproducibility. We recommend a pragmatic approach to assessing and improving reproducibility, making incremental improvements from project to project, keeping an eye on the shifting norms of the field and the evolving standards and norms for data formats, metadata, repositories, etc. Over time, some of the specific suggestions we have offered here may fall out of fashion or be replaced by superior options. However, the general principles that we focus on with our questions for are likely to endure beyond the technical details, and serve as useful prompts for assessing reproducibility well into the future.

References

Archambault, E., Amyot, D., Deschamps, P., Nicol, A., Rebout, L., & Roberge, G. (2013). *Proportion of open access peer-reviewed papers at the european and world levels—2004-2011*. Science-Metrix. Retrieved from http://www.science-metrix.com/pdf/SM_EC_OA_Availability_2004-2011.pdf

Baker, M. (2016a). 1,500 scientists lift the lid on reproducibility. *Nature*, *533*(7604), 452–454.

Baker, M. (2016b). Muddled meanings hamper efforts to fix reproducibility crisis. *Nature News*. http://doi.org/doi:10.1038/nature.2016.20076

Björk, B.-C., Welling, P., Laakso, M., Majlender, H., P., & Guðnason, G. (2010). Open access to the scientific journal literature: Situation 2009. *PLoS ONE*, *5*(6). http://doi.org/10.1371/journal.pone.0011273

Cartwright, N. (1991). Replicability, reproducibility, and robustness: Comments on harry collins. *History of Political Economy*, *23*(1), 143–155. Journal Article.

Casadevall, A., & Fang, F. C. (2010). Reproducible science. *Infection and Immunity*, *78*(12), 4972–4975. http://doi.org/10.1128/IAI.00908-10

Claerbout, J. F., & Karrenfach, M. (n.d.). Electronic documents give reproducible research a new meaning. Conference Paper, Society of Exploration Geophysicists.

Collins, H. M. (1984). When do scientists prefer to vary their experiments? *Studies in History and Philosophy of Science Part A*, *15*(2), 169–174. Journal Article.

Donoho, D. L., Maleki, A., Rahman, I., Shahram, M., & Stodden, V. (2008). *15 years of reproducible research in computational harmonic analysis*. Department of Statistics, Stanford University.

Drummond, C. (2009). Replicability is not reproducibility: Nor is it good science. *Proc. Eval. Methods Mach. Learn. Workshop 26th ICML, Montreal, Quebec, Canada*. Retrieved from http://cogprints.org/7691/7/icmlws09.pdf

Easterbrook, S. M. (2014). Open code for open science? *Nature Geosci*, *7*(11), 779–781. Journal Article. http://doi.org/10.1038/ngeo2283

Franklin, A., & Howson, C. (1984). Why do scientists prefer to vary their experiments? *Studies in History and Philosophy of Science Part A*, *15*(1), 51–62. Journal Article.

Goodman, S. N., Fanelli, D., & Ioannidis, J. P. A. (2016). What does research reproducibility mean? *Science Translational Medicine*, *8*(341), 341ps12–341ps12. http://doi.org/10.1126/scitranslmed.aaf5027

Gorgolewski, K. J., Auer, T., Calhoun, V. D., Craddock, R. C., Das, S., Duff, E. P., … Poldrack, R. A. (2016). The brain imaging data structure: A standard for organizing and describing outputs of neuroimaging experiments. *BioRxiv*. http://doi.org/10.1101/034561

Hajjem, C., Harnad, S., & Gingras, Y. (2006). Ten-year cross-disciplinary comparison of the growth of open access and how it increases research citation impact. *ArXiv Preprint Cs/0606079*.

Hanke, M., Halchenko, Y. O., Sederberg, P. B., Hanson, S. J., Haxby, J. V., & Pollmann, S. (2009). PyMVPA: A python toolbox for multivariate pattern analysis of fMRI data. *Neuroinformatics*, *7*(1), 37–53.

Kerr, N. L. (1998). HARKing: Hypothesizing after the results are known. *Personality and Social Psychology Review*, *2*(3), 196–217.

Larobina, M., & Murino, L. (2014). Medical image file formats. *Journal of Digital Imaging*, *27*(2), 200–206.

Miller, P., Styles, R., & Heath, T. (2008). Open data commons, a license for open data. *LDOW*, *369*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … others. (2011). Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, *12*, 2825–2830.

Stodden, V. (2009). Enabling reproducible research: Open licensing for scientific innovation. *International Journal of Communications Law and Policy*, *13*, 1–25.

Stodden, V., Borwein, J., & Bailey, D. H. (2013). Setting the default to reproducible. *Computational Science Research. SIAM News*, *46*, 4–6.

Stodden, V., Leisch, F., & Peng, R. D. (2014). *Implementing reproducible research*. CRC Press.

Van Der Walt, S., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., … Yu, T. (2014). Scikit-image: Image processing in python. *PeerJ*, *2*, e453.

Wells, D. C., Greisen, E. W., & Harten, R. H. (1981). FITS - a Flexible Image Transport System, *44*, 363.

The Basic Reproducible Workflow Template

Justin Kitzes

The core of this book consists of a set of thirty-one contributed case studies, each showing an example of a scientific workflow that was designed, at least in part, to achieve the goal of reproducibility. These case studies are concerned mainly with the goal of computational reproducibility, the ability of a second researcher to receive a set of files, including data, code, and documentation, and to recreate or recover the outputs of a research project, including figures, tables, and other key quantitative and qualitative results.

The thirty-one case studies in this volume describe a wide variety of research projects, disciplines, methods, and tools. Behind this diversity, however, all of the case studies share many key principles and practices in common. In this chapter, we describe what we view as the basic, underlying reproducible research workflow that any scientist should master before continuing on to the complexities described in the case study chapters.

To demonstrate this basic workflow, this chapter walks through a complete, concrete example of perhaps the simplest realistic data-intensive research project: a regression analysis of a single tabular data set. This example is designed to provide useful background for understanding the case studies later in this book. It will also provide a self-contained introduction to the practice of reproducible research for beginning readers looking for a template to adapt to their own research needs. We particularly encourage beginning readers to work along interactively with the example in this chapter to get a feel for how a reproducible workflow can be implemented.

We begin this chapter with a general overview of three key practices that are needed to make any research project, no matter how simple, computationally reproducible. This is followed by a high-level overview of the basic reproducible research workflow. We then provide an extended example of how this workflow can be implemented in a simple research project. We conclude with some additional considerations that arise when transitioning from this simple workflow template to more complex workflows, such as those described in the contributed case study chapters.

Three Key Practices

Chapter 2 described a set of questions that can be used to assess, at a relatively fine grained level, the extent to which a research project is reproducible. At a higher level, we can summarize these recommendations in three general practices that arise repeatedly throughout all stages of a research project:

1. Clearly separate, label, and document all data, files, and operations that occur on data and files
2. Document all operations fully, automating them as much as possible, and avoiding manual intervention in the workflow when feasible
3. Design a workflow as a sequence of small steps that are glued together, with intermediate outputs from one step feeding into the next step as inputs

At a beginning level, the first of these practices largely involves placing files in a clear directory structure and creating metadata to describe them. The second is met by writing code, or scripts, to perform each step automatically, or where this is not possible, documenting all manual steps needed to complete a task at a level that would allow a second researcher to unambiguously repeat them. The third is met through the overall workflow design, especially a clear conceptualization of the different operations that need to occur sequentially and how they support each other.

Although not described in the example below, nearly all of the contributed case studies in this book use version control software as a tool for following the first two practices above. In short, version control is used to capture a snapshot of all of a project's files at any moment in time, allowing a researchers to easily review the history of the project and to manage future changes. Version control also provides a means of documenting and tracking changes to project files in a systematic and transparent manner.

In our experience, however, many beginners find version control more difficult to learn than the other steps described below, and thus we have chosen not to include it in this basic workflow template. However, once you feel comfortable with this basic workflow, we recommend that you progress to one of the many online tutorials that can help you learn to use version control systems. We particularly recommend the tutorials on `git` available from Software Carpentry.

The Stages of the Basic Reproducible Workflow

The basic reproducible research workflow can be divided into three main stages: data acquisition, data processing, and data analysis. These three stages are preceded by activities related to system setup, and are succeeded by steps that automate the entire

workflow as much as possible. While steps such as project brainstorming and publication may also be a key part of a research workflow, the tasks that relate to ensuring a project's reproducibility fall primarily within these stages.

Before beginning a data-intensive computational research project, a computer system with the tools necessary to complete the analysis must be located and set up. These activities can be more or less involved, depending primarily on the researchers level of access to the computer and the programming language that will be used for the analysis.

The first stage of the basic workflow is data acquisition, input, or creation. This stage commonly consists of collecting data from a primary source, such as field observation, experimental research, or surveys. However, it also includes acquiring data from an existing source, through web-scraping or communciation with other researchers, or generating data via simulation. Regardless of the method, the end result of this first stage is raw data.

The second stage involves processing or cleaning of the data produced in the first stage. Depending on the tools used and the author's strategies, this stage may include tasks such as manual data entry, visual data review, or systematic data manipulation or filtering using scripts or other software. At the completion of this second stage, the relevant data is digitized, cleaned, and fully prepared for analysis. Although this stage is often treated as minor, or less important, than the other two stages surrounding it, we have found that this stage often requires as much intellectual energy, and as many difficult decisions, as the other stages in this template.

The third stage is data analysis. The most common form of data analysis is formal statistics, but other activities in this stage include data visualization, assessing the performance of particular algorithms, and extending the data to address a hypothesis or draw a scientific conclusion. The defining attribute of this stage is that it analyzes, in some manner, the clean data produced in the second stage, and produces the desired scientific products of the research, generally quantitative results in the form of figures and tables that are then incorporated into manuscripts, talks, and other forms of communication.

Finally, following the three central stages, the reproducibility of a project can be greatly enhanced through the creation of a single controller script that can automatically execute all three stages to produce a finished result. When this type of "push button" workflow is unrealistic or impossible to achieve due to project constraints, detailed documentation of all non-automated steps should be created.

Setup

The setup activities that precede the three core stages of a reproducible workflow consist first of gaining access to a computer, or several computers, to use for a project. For this simple example, we will presume that the entire analysis will occur on a personal computer

for which the researcher has full administrator access.

There are generally three classes of tools that must be installed at this stage. The first of these is a shell or terminal program that allows access to the command line. The second is a plain text editor or a development environment that can be used to write code in a chosen language. The third is an installation of a programming language, including an interpreter or a compiler, that will execute the researcher's code. Alternatively, researchers may choose to use an integrated workflow program, such as VisTrails, Taverna, or Kepler, although this approach will not be discussed here.

For the basic workflow that follows, Mac or Linux users can use the pre-installed Terminal program on their systems, while Windows users can work at the Command Prompt. All users should install a plain text editor, of which many are available for each platform (many of the authors in this volume use Sublime Text). Finally, the examples below will make use of the R language, and users should download and install a recent version of R.

More detailed information on the above installation steps, as well as basic tutorials on how to use these tools, can be found in the Software Carpentry lessons.

Stage 1: Data Acquisition

The first stage in most data-intensive workflows involves the acquisition of raw data. For this example, we'll imagine a study in which we have collected field data on the tomatoes produced by plants in several different fields as part of an agricultural experiment.

Table 1 reports hypothetical measurements of the total yield of tomatoes, in kilograms per plant, produced by four plants in each of three fields with no management after planting (N), conventional management with fertilizers and pesticides (C), or organic management (O). The third column indicates whether substantial insect damage was noted on the plant leaves at the time of harvest. Of the fifteen plants marked for sampling, two of them, denoted with `NA` in the weight column, were killed before bearing fruit.

Table 1: Sample tomato data set

Field	Weight	Insect
N	5.8	Y
N	5.9	N
N	1.6	Y
N	4.0	Y
N	2.9	Y
C	12.4	N
C	11.5	N
C	9.3	N
C	NA	N
C	12.1	N
O	9.9	N
O	6.7	N
O	10.6	Y
O	3.7	Y
O	NA	N

This data should be entered into a spreadsheet program and saved as a CSV file. CSV files are a commonly used plain text format for storing tabular data, in which each row in a table is on a separate line and data for each column are separated by a comma. Plain text formats are often preferable to program-specific formats, such as XLSX, as they are more easily readable by a variety of software and by other researchers who may wish to work with this data.

Once this file is created, it should be given a name and saved in a useful location on a hard drive. Naming conventions vary widely between researchers, but in small projects such as this one, we recommend giving files names that usefully describe the contents, even if these are quite long. This table, for example, might be saved as `raw_yield_data.csv`. To avoid the possibility of errors later in the workflow, spaces, periods, and slashes should not be used in file names.

At the same time that data are saved, a metadata file should also be created and saved along side of it. The purpose of the metadata file is to document the source of the data file and any relevant information about its contents. While many disciplines have standards for metadata, a minimal metadata file consists of a simple text file that describes, in plain English, where the data file came from and what it describes. Such a file, which we can save as `README.txt` alongside the data file, might contain information like the following.

> Data collected by undergraduate assistants to Prof John Smith at the Berkeley Field Station. All plants were located in Field 3 and chosen for measurement when approximately 12" tall. Yields were recorded in August 2015.
>
> Field codes indicate no treatment (N), conventional (C), and organic (O). Yield is in kg, with NA indicating a plant that died prior to yield measurement. Insect damage assessed visually, Y indicates more than approximately 25% loss of leaf area.

The question then arises of where these two files, as well as all of the subsequent files that will be part of the project, should be saved. A common convention is to place all project files in a single directory, with a single layer of subdirectories for different types of files, such as data, source code, analysis results, etc. A structure such as the below, with all files and subfolders contained in a single folder called `tomato_project`, provides a useful starting point for simple projects.

```
|-- tomato_project
|   |-- data_raw
|   |   |-- raw_yield_data.csv
|   |   |-- README.txt
|   |-- data_clean
|   |-- results
|   |-- src
```

Stage 2: Data Processing

Once raw data has been collected and placed in a project directory, it nearly always requires some form of processing or cleaning before it can be used in an analysis. This step may involve removing invalid data, subsetting the original data, removing outliers, and other similar steps. The best approach for processing a raw data set, of course, is dependent on the questions that a researcher hopes to answer with this data and the particular type of analysis planned for Stage 3.

In this example, inspection of the raw data table revealed two plants without yield measurements, which we may wish to remove from the data before any further analysis. Given a goal of eventually conducting a two-sample t-test comparing the conventional to the organic yields, we also know that we can remove the no treatment plants from the table at this stage. For a small table such as this one, removing these rows is not strictly necessary, although such subsetting may dramatically improve the efficiency of subsequent analysis of larger data sets.

To make this stage fully reproducible, every step taken to process the data must be recorded with detail fine enough that only one processed data set could result from the combination of the raw data and the set of instructions. The simplest and the recommended way to

accomplish this is to encode the instructions for data processing as computer code, in a script, that will read in the raw data, execute various processing and cleaning operations, and save the resulting processed data as a new file.

Particularly for small tabular data, it can be tempting to skip this coding step, and instead open the file in a graphical editor, such as a spreadsheet program, delete the rows or columns that are not needed, and save the resulting file. In some instances, particularly where data files are stored in a proprietary format that can only be opened by certain programs, this manual approach may be the only option. Manual data processing, however, is prone to error and makes the "push button" automated workflow described later impossible.

As is the case with all research tasks, if this step must be done manually, ensure that the processed data file is accompanied by a very detailed human readable description, saved in a text file like the metadata file, that describes every operation performed on the raw data, to the level of what menu was selected and what button clicked in what order. Remember that if an unaffiliated researcher who you have never met cannot exactly, with 100% accuracy, reproduce the processed data file from the raw data and instructions, then this step is not fully reproducible. In many ways, this instruction file is itself similar to code, although it is intended to be executed by a human reader rather than by a computer.

For this tomato yield data, we can readily write a short script that will read the raw table, remove the rows with `NA` yields and those with a field code of `N`, and save the resulting processed data. The following R commands will perform these operations.

```
yield_data <- read.csv("yield_data.csv")
clean_yield_data <- na.omit(raw_yield_data[raw_yield_data$Field != "N", ])
write.csv(clean_yield_data, "clean_yield_data.csv")
```

While exploring the data, the commands above can be entered interactively into an interpreter window. Once a procedure for data processing has been identified, however, all of these commands should be placed in a separate file that when executed, will read the raw data, process it, and save the resulting processed data file. This ensures definitively that all necessary steps to reproduce this stage of the workflow were recorded properly and can be easily repeated at will.

In the simple directory structure described earlier, scripts and other code are saved in the `src` subfolder. To ensure that a script in the `src` directory will locate and save the appropriate files in the appropriate folders, we can modify the code above to the following, which modifies the locations where the files are read and written.

```
### Read in the raw data, assuming we are working in the src directory
raw_yield_data <- read.csv("../data_raw/raw_yield_data.csv")

### Clean the data by removing rows with NA and where 'Field' == N
clean_yield_data <- na.omit(raw_yield_data[raw_yield_data$Field != "N", ])

### Write the clean data to disk
write.csv(clean_yield_data, "../data_clean/clean_yield_data.csv")
```

The commands above, when saved as a script `clean_data.R` in the `src` subfolder, will read the table `raw_yield_data.csv` from the `data_raw` subfolder, clean it, and save the resulting cleaned table as `clean_yield_data.csv` in the `data_clean` subfolder. The cleaned data are placed in a different subfolder from the raw data to ensure that the original, raw data are never confused with any derived data products. Ideally the raw data files should never be manually altered but should be kept intact, with all changes and modifications saved to a separate file. This will ensure that you can always go back to the original data if you make a data processing decision that you regret.

To execute this script, navigate to the `src` subfolder in a terminal window and run the command `r clean_data.R`. For more information on working at the command line, see the Software Carpentry shell tutorial.

The project directory should now look like the following.

```
|-- tomato_project
|   |-- data_raw
|   |   |-- raw_yield_data.csv
|   |   |-- README.txt
|   |-- data_clean
|   |   |-- clean_yield_data.csv
|   |-- results
|   |-- src
|   |   |-- clean_data.R
```

Stage 3: Data Analysis

Once data are checked in and processed, the third stage of the basic reproducible workflow is data analysis. There are, of course, many different types of analyses that may be employed here and many different types of outputs that can result, including text-based results, tables, and figures. For this example, we'll perform an unpaired two sample t-test to determine whether the mean tomato yield per plant is significantly different in the conventional and organic fields.

As with data processing, data analysis may be done manually using graphical tools, such as a spreadsheet program. This is not recommended due to the difficulty of accurately capturing all of the minute details needed to allow a second researcher to exactly repeat the analysis without errors. Data analysis may also be performed interactively, with code entered into a "live" interpreter window until a final result is reached and saved. This step is often important as a means of exploration to determine what commands should be used for the analysis. Once interactive tools have been used to explore possible approaches, however, we recommend strongly that all commands needed to perform the data analysis be placed in separate file that will save the results when executed.

The code below should be saved in a script titled `analysis.R` in the `src` directory. When run, it will read the cleaned data table, perform the desired t-test, and save the summarized output of the text in the `results` subfolder as a plain text file `test_results.txt`. Although not applicable here, any other results including tables and figures should also be saved in the `results` subfolder.

```
### Load clean data, assuming we are in the src directory
clean_yield_data <- read.csv("../data_clean/clean_yield_data.csv")

### t-test of Weights by Field type: is there significant difference in
### tomato yield in the different fields?
t_test_Weight_Field <- with(clean_yield_data, t.test(Weight ~ Field)

### Write test result to plain text file
capture.output(t_test_Weight_Field, file = "../results/test_results.txt")
```

Note that several comments describing the analysis steps are included in the code above. Although the relatively simple commands here do not require extensive explanation, comments should be used liberally in all code files, as we have demonstrated in the examples here. While the code itself describes *what* operation is performed, comments should be used to describe *why*, and in a larger sense *how*, a desired analysis is being conducted. While the code itself is designed to reproduce the quantitative results of an analysis, code comments and other documentation are designed to help another researcher reproduce the thought process that went into structuring and writing code in a particular way.

At the conclusion of this stage, after the script `analysis.R` has been run in the same manner as the previous `clean_data.R` script, the project directory will appear as follows.

```
|-- tomato_project
|   |-- data_raw
|   |   |-- raw_yield_data.csv
|   |   |-- README.txt
|   |-- data_clean
|   |   |-- clean_yield_data.csv
|   |-- results
|   |   |-- test_results.txt
|   |-- src
|   |   |-- analysis.R
|   |   |-- clean_data.R
```

The `test_results.txt` file indicates that there is no detectable significant difference between the yields in the conventional and organic fields (p = 0.104).

Automation

At this stage, the reproducible workflow is essentially complete. We have written code that, when executed, will read and process our raw data table and save both a cleaned data table and the final results of our analysis. Most importantly, the final result of our analysis, the p-value for the comparison of the conventional and organic yields, can be reproduced by any researcher who has access to the original data and the code that we have written.

To make this workflow even easier to reproduce, a controller or driver script can be added to execute, in one step, all of the various subcomponents of the entire workflow. In this simple example, our workflow has only two steps that can be performed automatically: executing `clean_data.R` to generate the cleaned data table, and then executing `analysis.R` to perform the statistical test.

To create a single entry point that will perform our entire analysis, we can create a shell script, `runall.sh`, that we can save in the `src` directory. For this simple, example, the script need only contain two lines.

```
r clean_data.R
r analysis.R
```

To test out this controller script, delete the contents of the `data_clean` and the `results` directory to simulate giving a colleague only your raw data and code. From the command line, navigate to the `src` directory and run the command `sh runall.sh` to see the intermediate and final results of the workflow regenerate.

In addition to supporting reproducibility, the creation of a "push button" workflow like this has a second related side benefit, which is ensuring that any generated results are linked directly back to specific known data sets and analysis parameters. We and many of our colleagues

have been known to finish working on real projects, delete all results precisely as described above, and rerun the entire workflow using a controller script. This final step ensures that all results used in subsequent interpretation and presentation were, in fact, generated from the latest data and code in the project directory.

Conclusion

While some real world workflows are nearly as simple as the one shown here, many projects will be more complex, perhaps substantially so. The most immediate extension of the template shown here would be the need to accommodate a greater variety of file types, including many types of code files, several categories of results, binary executables, and documentation. From an organizational perspective, an additional level of subfolders can be be created within folders such as `src` and `results` to organize these additional files. Subfolders such as `doc` and `bin` within the main project directory can be used to house files related to documentation, including manuscripts, and compiled binaries.

Beyond the addition of more project files, more complex projects will require more complex workflows that allow, for example, files to be shared across multiple projects, the same analysis to be run on multiple data sets or parameter combinations, analysis to be run on remote computers including those in the cloud, etc. Many of these additional complexities are discussed in the contributed case studies that follow this chapter.

When moving beyond the tools and techniques described above, we first recommend that you learn to integrate a major version control software package into your workflow. Tutorials for software such as `git`, which is used by the majority of the case study authors, are readily available online.

A second direction that may also be of interest would be to use a literate programming approach. This approach involves creating a single source document in a language such as markdown or LaTeX, or using a "notebook" interface such as one provided by Sage or Jupyter, that contains the narrative text describing the project directly alongside code, figures, tables and other results of our report. In this framework, the single source document can be executed to run the code and obtain results directly alongside narrative description and documentation. This approach provides a self-contained file of text and code that is convenient for circulating to other readers by email or submitting for publication.

In closing, we note once again that the structure of this basic reproducible workflow, particularly the division of the workflow into the three core stages plus setup and automation, underlies all of the more complex case studies described in this volume. We encourage researchers, both beginning and advanced, to use the template in this chapter as a basic foundational framework for understanding, organizing, and creating reproducible workflows as part of real world research projects in the data-intensive sciences.

Case Studies in Reproducible Research

Daniel Turek and Fatma Deniz

Having discussed the context and the general practices of reproducible research, we will now shift focus to a collection of concrete examples of scientific research workflows, all of which strive to attain a high degree of reproducibility. These case studies of reproducible research are the foundation for our study of approaches and current best practices for achieving computational reproducibility. By studying these real-world examples, we are able to draw conclusions regarding the tools, software, and current trends of reproducible scientific research.

In this chapter, we begin by introducing the concept and format of the case studies, including the motivating factors behind the general framework of a case study. Next, we describe the methods and process of collecting the case studies from researchers spanning a range of scientific disciplines. The case studies themselves shed light upon a natural classification into two distinct categories. This classification is described, and an index of the case studies is provided. As a high level summary, we next present broad descriptions and summary statistics of the case studies. These provide insight into the currently most common tools and methodologies facilitating reproducible research. Finally, we provide some suggestions for reading the case study chapters to attain a deeper understanding of these examples. These suggestions are intended to help readers identify ideas and insights for crafting their own reproducible scientific research workflows.

What is a case study?

A case study is a comprehensive description of the computational workflow that a researcher used to complete a single, well-defined scientific research project. Each case study describes how particular tools, ideas, and practices have been combined to support reproducibility. Emphasis is placed on the *how*, rather than the why or what, of reproducible research. Each case study can be viewed as one approach among many possibilities for how a researcher approached the challenge of reproducibility.

Each case study follows a consistent, standardized format. Each begins with a short biography of the author, including their affiliation, discipline, and a brief abstract describing the subject of their case study. The body of each case study consists of the three core

sections: a workflow narrative accompanied by a flowchart diagram, a discussion of the most important tools and achievements of the workflow, and a discussion of the most significant problems encountered in achieving reproducibility.

The workflow narrative and diagram are the heart of each case study. The diagram outlines the project in a manner similar to a circuit diagram: boxes represent steps in the process, and arrows represent the flow of information into subsequent steps. Most diagrams are built around combinations of specialized tools, version controlled repositories, databases, scripts, and end products such as statistical conclusions, functional software, or scientific publication. The workflow narrative ties closely to the diagram, and explains various stages and flow of information shown in the diagram. The narrative provides an opportunity for authors to discuss topics such as the appropriate use of tools, how various steps were automated, the history of raw data, and whether the software that is used for analysis is publicly available with sufficient documentation and testing.

Following the workflow narrative and diagram, each case study highlights the main successes of reproducible research from the project. This *Key Benefits* section describes the ways in which following this reproducible workflow has improved the author's research, often by making it more efficient, transparent, and trustworthy in addition to more reproducible. This section may also discuss how the project benefited from the reproducible or open-source nature of other projects and how other researchers could reuse portions of the workflow.

Finally, in the *Pain Points* section, each case study reflects on the most troublesome obstacles encountered in the pursuit of reproducibility. These challenges may have been successfully navigated, or may still remain. Examples include data sets that could not be made publicly available, legacy code inherited from other scientists, or difficulties in collaborating with other scientists without experience or interest in reproducible research. These troublesome aspects should be equally as instructive as the successes and key tools, since they highlight the practical hurdles to producing fully reproducible research.

Case studies may also include a *Key Tools* section, which specifically points out any software or other tools that helped achieve a reproducible workflow. And finally, some case studies address several optional questions, which touch on the broader context of reproducible research and its challenges. Where provided, answers to these questions are included at the close of the case study. The optional questions posed to each author were:

- What does "reproducibility" mean to you?
- Why do you think that reproducibility in your domain is important?
- How or where did you learn about reproducibility?

- What do you see as the major challenges to doing reproducible research in your domain, and do you have any suggestions?
- What do you view as the major incentives for doing reproducible research?
- Are there any best practices that you would recommend for researchers in your field?
- Would you recommend any specific resources for learning more about reproducibility?

This format for case studies was designed largely before eliciting the case studies from contributing authors. This format was selected to serve several purposes. Foremost, the workflow narrative and diagram are intended to provide a clear visualisation of the end-to-end scientific workflow, as well as the author's commentary and description of this process. Either alone would not provide a comprehensive idea of their approach to achieving reproducibility. Second, the remaining sections are designed to clearly distinguish important aspects of the researchers' approach to reproducibility. While similar information may also appear in the workflow narrative, the *Key Benefits*, *Pain Points*, and *Key Tools* sections isolate these concepts, and force each author to reflect clearly on the strengths and weaknesses of their approach. Combined, these sections provide a comprehensive view of authors approach and experiences in their quest to achieve reproducibility.

Collecting the case studies

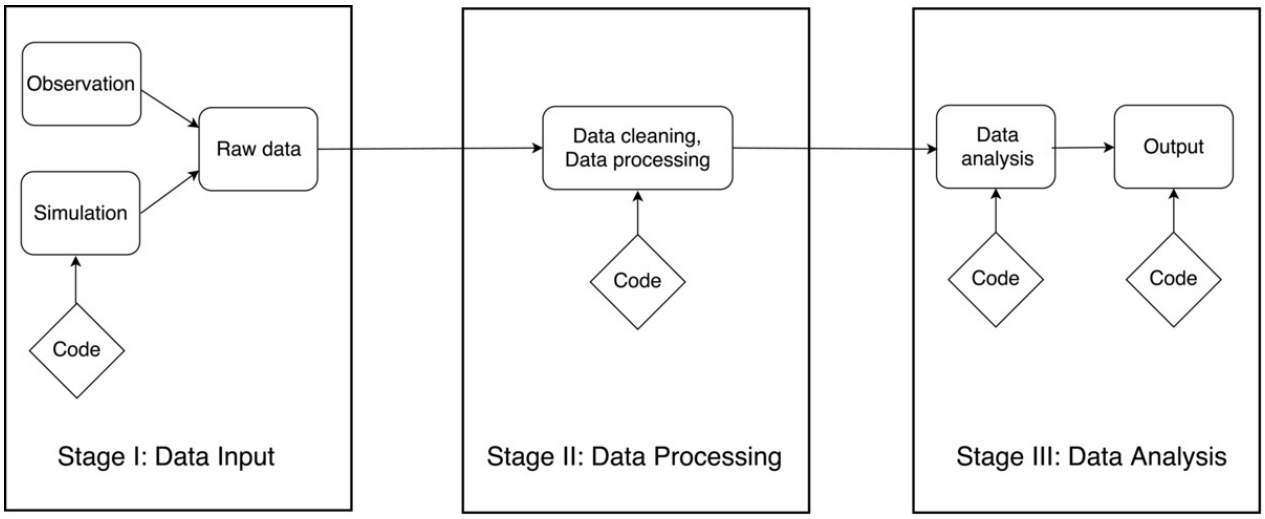
The process of collecting case studies was coordinated by a core group from the Berkeley Institute for Data Science, at the University of California, Berkeley. This process of collecting case studies spanned a period of approximately six months.

Initially, the core group drafted a general framework of a reproducibility case study. At its inception, this consisted only of the workflow diagram and accompanying narrative. Members of this group each wrote a case study describing one of their own research projects. After examining these initial submissions, a formal template for a case study was created. This consisted of the introductory biographical questions for each author, a description and guidelines for the narrative and diagram, and a set of questions regarding various aspects of reproducibility.

This template was later distributed to attendees of a Reproducibility Workshop hosted at the Berkeley Institute for Data Science. One session of this workshop gave attendees the opportunity to draft a case study describing their own research. Although attendees only had a few hours to work on their submissions during the workshop, the majority took additional time after the workshop to finalize their case study. A third and final round of case studies was later elicited through personal requests to leading scientific researchers.

Classification and Index

As described in the last chapter, a data-intensive research workflow can be divided into three main stages: data input/acquisition, data processing, and data analysis and outputs. The first stage represents data acquisition, input, or creation. Regardless of the source of the data (via collection, simulation, or otherwise), the final result of this stage is one or more raw data sets. The second stage includes both cleaning and processing of this raw data. This can include many different tasks such as consolidating, organizing, or digitizing, the output of which is a cleaned dataset fully prepared for the third stage. Finally, the third stage includes all statistical analyses, visualizations, and the creation of output products. This may frequently result in scientific publication, but many other forms of output are possible, such as software tools, public repositories, scientific conclusions, or actionable insights. An outline of a fully generic scientific workflow into these three distinct stages is shown in Figure 1.



Using this three-stage taxonomy, the case studies naturally fell into one of two broad categories. The first we called "high-level" case studies, which describe a complete scientific workflow involving all three stages. These generally provide a lighter treatment of each stage, and contain fewer technical details. The second category is called "low-level" case studies, which consists of those case studies describing only one or two of these three stages. These low-level examples generally provide a more detailed or technical treatment of the various stages. Low-level case studies are further classified by which stage(s) they describe.

Using this classification, we present in Table 1 an index of all case studies contained in this book. Each case study is classified as either high-level or low-level, and according to the scientific discipline from which it is drawn. This index is intended to help guide readers in their exploration of the case studies.

Table 1: Guide to case study chapters

Author	Discipline	Topic
HIGH LEVEL		

Anthony Arendt	Applied Physics	Impact of glacial melt on rising global sea levels
Pablo Barberá	Political Science	Studying political polarization on social media websites
Carl Boettiger	Theoretical Ecology	Forecasting and decision-making in ecological systems
Garret Christensen	Economics	Causal impacts of military history on soldier recruitment
Jan Gukelberger	Physics	Diagonalization simulations for quantum systems
Chris Hartgerink	Applied Statistics	Validating statistical methods to detect data fabrication
Chris Holdgraf	Neuroscience	Feature extraction for predictive models of the brain
David Holland	Applied Mathematics	Climate change and melting of the great ice sheets
Justin Kitzes	Ecology	Analyzing bat activity using autonomous acoustic detectors
Andy Krause	Civil Science	Analysis of US household locations in metropolitan areas
José Manuel Magallanes	Political Science	Using bill cosponsorship data to detect political trends
Benjamin Marwick	Anthropology	Understanding prehistoric hunter-gatherer behaviour
Olivier Mesnard	Aerospace Engineering	Full replication of computational fluid dynamics results
K. Jarrod Millman	Statistics / Psychology	Assessing reliability for human classification of autism
K.A.S. Mislan	Environmental Science	Comparison of blood-oxygen binding characteristics
Kellie Ottoboni	Statistics / Public Health	Analyzing association of salt consumption and mortality
Karthik Ram	Data Science	Developing tools to support stages of reproducible research
Ariel Rokem	Neuroscience	MRI studies of brain structure and function
Rachel Slaybaugh	Nuclear Engineering	Numerical methods to study neutral particle interactions
Daniela Ushizima	Image Processing	Devising machine vision and pattern recognition algorithms

Zhao Zhang	Computer Science	Image processing with cloud computing using Apache Spark
LOW LEVEL		
Kyle Barbary	Cosmology	Analyzing supernova data to measure universe expansion
Fatma Deniz	Image Processing	Generating two-tone Mooney images to study brain activity
Konrad Hinsen	Molecular Biophysics	Analysis of molecular dynamics trajectories for biomolecules
Kathryn Huff	Nuclear Engineering	Simulation framework for nuclear fuel cycle analysis
Randy LeVeque	Applied Mathematics	New approaches to probabilistic tsunami hazard assessment
Tara Madhyastha	Neuroscience	Neuroimaging workflow using automated build tool
Gilberto Pastorello	Computer Science	Data processing pipelines and data management solutions
Russell Poldrack	Neuroscience	Analysis of neuroimaging, behavioral, and metabolomic data
Valentina Staneva	Mathematics	Developing stochastic filtering methods for tracking objects
Daniel Turek	Statistics	Developing and testing efficient statistical algorithms

Trends among the case studies

Despite representing a wide range of scientific disciplines, many similarities exist between the various case studies. Here, we summarize several of the main trends and the emergent characteristics which can be observed. This includes a summary of the main languages used for computational research, trends in data sharing and version control, and other high level properties.

This book contains a total of 31 contributed case studies of reproducible workflows. Among them, 21 are high-level workflows describing the end-to-end process including data input or acquisition, data processing, and data analysis. The remaining 10 case studies are low-level workflows, which provide greater detail on one or two of these stages. Approximately one third of the low-level case studies discuss data input or acquisition (Stage 1), half describe

data processing (Stage 2), and half discuss data analysis (Stage 3). Note that some low-level case studies cover two of these stages, for example both data processing and data analysis.

Each of the 31 case studies represent a data-centric computational scientific workflow, and therefore describes various tools or languages for data management, data processing, or scientific computing. Although myriad computational tools are described, a few are extremely widely represented among the case studies. In particular, 17 of the 31 case studies (55%) make use of Python, an open-source, high-level and general-purpose programming language. This accurately reflects the current popularity of Python, thanks to to rapid development cycle, the high readability of Python code, and the extremely wide range of applications supported by Python. The next strongest representation is of R, an open source programming language for statistical computing, which is used in 13 of the 31 case studies (42%). This is an accurate representation of the wide-spread use of R among data analysts, and generally the statistics community at large, as R is now considered the primary ecosystem for statistical computing. Following Python and R, a vast range of other programs and computational tools have a comparatively modest representation among the case studies. To name just a few of the more mainstream tools, these include C/C++, MATLAB, Julia, Scala, Java/JavaScript, and oftentimes custom-developed software, although this listing is far from comprehensive.

Appropriate use of version control is a key aspect of modern reproducibility. This applies equally for software development and the computer code underlying computational workflows. Older (centralized) version control systems were more cumbersome for users, but the recent introduction of git and GitHub have made version control more accessible for smaller-scale projects. The vast majority (over 80%) of the case studies make use of git and GitHub for version controlling the development of software or analysis code, which represents one of the strongest trends among the case studies. A number of the remaining case studies explain that the nature of the workflow is not appropriate for version control, for example when describing a protocol for data management. Further, a few case studies make use of other version control software -- for example, Bitbucket or SVN -- but these represent a small minority.

In support of transparency and reproducibility, there is an on-going shift within academic communities in support of open data and data sharing. Indeed, 19 of the 31 case studies (over 60%) make use of publicly available data, or themselves describe the process of making their data publicly available. However, an open-data policy is not universally practiced, as in some disciplines the extreme overhead of data collection deters scientists from openly sharing it. That is an unfortunate reality in some fields, for example cosmology, astrophysics, or neuroscience, but the current trend among the scientific and academic communities is strongly moving towards the use of open data.

There is also a clear trend in the output medium of the case studies, although we believe this may be an artifact of the contributing group of authors rather than of reproducible workflows in general. The collection of case studies was drawn from the academic community, where primary emphasis is placed on scientific publication. All but a few of the case study workflows culminate in producing a scientific manuscript intended for peer-reviewed publication. Perhaps more important, slightly over one third of the case studies also describe a second output. This is typically manifested as a software product, or an analysis algorithm intended for wider use. Other, less common, secondary outputs include data management pipelines, or interactive websites.

Reading the case studies

As readers consider the design of their own reproducible scientific workflow, a wealth of knowledge and experience is available in the case studies presented at the end of this book. However, reading the case studies may be daunting, as many are technical and may assume familiarity with computational tools or specific application domains. For this reason, we now provide some suggestions for reading the case studies.

We encourage readers who are new to reproducible research to begin by skimming through the high-level case studies, which provide a general overview of research workflows from a variety of disciplines. This will provide a general idea of what is contained in the case studies, and may highlight disciplines that have faced, or have solved, similar challenges to those faced by the reader. Ecologists and cosmologists, for example, both often work with high-resolution spatial data, while neuroscientists and empirical economists may encounter similar issues surrounding data anonymization.

As readers become familiar with the format and presentation of the case studies, they might next consider a detailed reading of the case studies drawn from the most closely related disciplines to their own. In these examples, the nature of the scientific research is more likely to be familiar to the reader. In addition, they are likely to give an idea of what tools, challenges and approaches are being used in one's own discipline.

Finally, the motivated reader is encouraged to undertake detailed readings of both high-level and low-level case studies which address the tools or issues most closely related to your own research. Case studies will invariably discuss technical tools, topics, and methods that will not be familiar to you. Rather than including explanations of these technical concepts in each chapter, we have provided descriptions of the most common terms and tools in a technical glossary at the end of the book. Readers are encouraged to refer to this glossary frequently while reading through the case studies.

It is important to note that each case study is a problem-specific example of a reproducible workflow. Rather than attempting to recreate any particular workflow, ideas should be selected from a variety of case studies to create your own customized approach to reproducibility. However readers decide to navigate the collection of case studies, they should keep in mind that *every* case study has some useful insights to offer -- including those drawn from unrelated disciplines. We encourage readers to study a variety of the workflows presented, since this approach is most likely to give a flavor of the common techniques and best practices generally applicable to scientific research.

Lessons Learned

Kathryn Huff

Although the case study authors came from a variety of research backgrounds, a set of themes emerged out of this collection of their workflows. Similar struggles arose despite differing scientific fields (ecology, neuroscience, astronomy, nuclear engineering) and nearly irrespective of preferred programming language (i.e. R, Python, C++, Matlab). This chapter will summarize some of the common themes among the case studies, both painful and positive.

It should be noted that the sample of chapter contributors is not a representative sample of scientists in these research fields. Indeed, these scientists contributed to the book because they are particularly interested in open science and reproducibility. Accordingly, we can imagine that where these scientists have pain points, many of their colleagues may give up on reproducibility outright.

Some key findings include the optimistic observation that git and GitHub are nearly ubiquitous among the case study authors. Additionally, we saw that among respondents, scripting analysis wherever possible is widely accepted as essential. For both reasons, plain text file formats were preferred.\ Testing and continuous integration was seen as crucial to maintaining reproducibility by those who have integrated these steps into their process, but quite a few respondents didn't mention either practice as part of their workflow. Finally, some of the most successful approaches were those that fundamentally recognize and adapt to the "ubiquity of error" that the scientific method defends against (Donoho, Maleki, Rahman, Shahram, & Stodden, 2009).

Obstacles to reproducibility included issues with humans, computers, and the institutions that both inhabit. The case studies made clear, for example, that humans must be incentivized to spend time on tasks intended solely for reproducibility. This can be complicated by skill variation and disparate tool familiarity within research groups. But, even when the tools are used, a lack of access to restricted data or hardware can hobble reproducibility efforts of even the most determined scientists. Similarly, portability of one's workflow is still a challenge for those intent on openness, since packaging - especially installation of dependencies - remains a critical stumbling block to sharing and extending work.

In the following sections, this chapter will discuss the lessons we learned from the case studies. First, this chapter will briefly mention how various scientists perceive reproducibility, then it will focus on the pain points. Next we will make note of oft-mentioned workflow tools, and finally this chapter will note some novel ideas that the case study authors had up their sleeves.

The Meaning of Reproducibility

The case study authors were prompted to give many perspectives as they prepared their case studies. One was "Define what the term *reproducibility* means to you..."

Some authors teased out quite a bit of the subtlety embedded in this semantic question. K. Jarrod Millman, Kellie Ottoboni and Philip Stark, for example, broke down reproducibility into four distinct types.

1. *Computational reproducibility and transparency*, which emphasizes code documentation.
2. *Scientific reproducibility and transparency*, which emphasizes documentation of scientific decisions and accessibility of data.
3. *Computational correctness and evidence*, which emphasizes automated testing and validation.
4. *Statistical reproducibility*, which emphasizes transparency of data analysis the logical path to scientific conclusions.

Most authors, however, expressed some flavor of either computational reproduciblity or replicability.

Computational Reproducibility

There was general agreement among most authors about at least one aspect of what reproducibility means: that when provided with identical source code, input data, software, and computing environment configurations, that an independent party can exactly reproduce the results of the original work -- especially published results. This is described in our glossary as *computational reproducibility*.

This aspect of reproducibility was articulated particularly well by the following case study authors -- although each definition has its own interesting subtleties:

Jan Gukelberger:

> In general, given a publication (in a refereed journal), source codes and raw data (which might be available publicly or in the institute's repositories), an expert from my field should be able to understand, and in principle repeat, every step of the study from the running of the correct version of the simulation code to the final results presented in the published paper.

Justin Kitzes:

> I consider a study to be (computationally) reproducible when I can send a colleague a zip file containing my raw data and code and he or she can push a single button to create all of the results, tables, and figures in my analysis.

Andy Krause:

> "Reproducibility" means that a subsequent interested party can openly access the data, code, analytical workflow and data provenance to re-create the research (and ideally produce identical results) WITHOUT consulting the original researcher(s).

These echo a well-established perspective on reproducibility (Donoho et al., 2009; Stodden, 2010; G. Wilson et al., 2014) that is evolving as a community norm through checklists and pledges such as the "Reproducibility PI Manifesto" (Barba, 2012). A few of our case study authors have taken this pledge in which a PI vows to adopt practices that add a level of sustainability and extensibility to reproducible work:

1. Teaching group members about reproducibility
2. Maintaining all code and writing under version-control
3. Carrying out verification and validation and publishing the results
4. For main results in a publication, sharing data, plotting scripts, and figures under CC-BY
5. Uploading preprints to arXiv at the time of submission of a paper
6. Releasing code no later than the time of submission of a paper
7. Adding a "Reproducibility" statement to each publication
8. Keeping an up-to-date web presence

The importance of this sustainable, extensible kind of reproducibility was noted by Kyle Barbary:

> To me, reproducibility has two facets: the availability of usable software (preferably under an open-source license), and the availability of data (preferably in both raw and reduced forms). Together, these should give an outsider the ability to reproduce the results of a study from start to finish. I separate these two aspects because each can be beneficial without the other. For example, even without releasing data, it can still be quite beneficial to release software. If released under an open-source licence, this provides a different flavor of reproducibility - the ability to reproduce an algorithm described in a paper and use and improve that algorithm in subsequent work.

Replicability

When the final conclusions can be confirmed based on a different experiment, scientists consider this validation of the result. In this vein, Valentina Staneva distinguishes between exact and approximate reproducibility:

> "Exactly reproducible" - when a result can be regenerated exactly as suggested given the same set of inputs and parameters.
>
> "Approximately reproducible" - when a result or similar performance can be generated with similar or different methods than the one proposed on the same or possibly slightly different data.

Some have used the term "replicability" for this approximate reproducibility. Ariel Rokem put it this way:

> A higher standard, sometimes called 'replicability' would be to require that the same conclusions be reached if another group of researchers were to do the same experiments, and implement the same ideas in their analysis. Reproducibility does not guarantee replicability [Leek and Peng, 2015]. Some may even argue that reproducibility and replicability may sometimes be in conflict, because implementation errors can be propagated in reproduction, but not in replication [Peng2009, Baggerly2005].

Validation of a scientific result is achieved in this way when one can repeat the scientific work with a new method or implementation and draw the same conclusions.

Pain Points

We also asked the case study authors which features of their workflows presented challenges to reproducibility. Irrespective of the type of reproducibility being sought, we hoped that these pain points would reveal areas of particular need - workflow bottlenecks where innovation might improve the experience of reproducible science. The following

sections highlight some of these frustrating, time consuming, opaque, or fragile obstacles and mentions when they may represent high priority needs for better tools and improved strategies.

People and Skills

Research teams are diverse. Computational skills especially vary dramatically from one researcher to another even within the same lab. The blinding pace of innovation in software tools means that even well-prepared collaborators can't expect to always keep up with the newest tools. Manuscript preparation software, database formats, and version control systems used by one scientist may be equally modern but nonetheless incompatible with the software stack familiar to their collaborators.

When the case study authors reported that the bottleneck to adopting practices was related to a diversity of skills, the indication was universally that the process might have been more efficient or reproducible were there a greater and more homogeneous distribution of tool familiarity among their research group members. Time was wasted when simple tasks like communicating results or simultaneously editing a manuscript were crippled by one or more collaborators unfamiliar with tools used by their colleagues.

The concern also extended far beyond mere efficiency. One case study author noted that if a collaborator is unable to use the tools that are being employed, then they are at risk of being disenfranchised from the scientific process. This disenfranchisement is especially ethically problematic if a collaborator is unable to directly or simultaneously edit a co-authored manuscript due to their lack of familiarity with the processing tools (e.g. LaTeX.)

A scientist unwilling to disenfranchise their collaborators could certainly elect to use more widely used tools, accepting frustration with inefficiency as the price of collaboration. However, the price is often paid in reproducibility as well when those widely-used, lowest-common-denominator tools conflict with reproducibility goals. This is especially the case with tools such as Microsoft Word, Excel, or Matlab which were noted as particularly problematic fallbacks, as their closed-source GUI-based nature is fundamentally fragile to reproducibility issues.

So, in the interest of both reproducibility and efficiency, some case study authors were inclined to proceed with the use of preferred tools (e.g. LaTeX) nonetheless. Those scientists largely saw the pain point caused by a\ difficulty of communication with and understanding from their peers. The ethical quandary for those scientists competed with the commitment to more effectively communicate their results (reproducibly and transparently) with the larger scientific community -- even if it had the effect of hobbling communication internally.

Need: Better education of scientists in more reproducibility-robust tools.

Need: Widely used tools should be more reproducible so that the common denominator tool does not undermine reproducibility.

Dependencies, Build Systems, and Packaging

Just as scientists "stand on the shoulders of giants," our software perches upon forests of dependency trees. A single step in our workflow may rely on dozens of libraries and scientific software packages which may each, in turn, rely on many other libraries and packages.

Accordingly, the first obstacle for use, sharing, and adoption of any software stack or analysis workflow is often the battle to simply get the workflow running on a different machine than that on which it was created. This first obstacle easily becomes the last for busy scientists, and halts reproducibility in its tracks. If another scientist can't even install LAPACK with your special compiler flags, they have no hope of building, installing, or running your software pipeline. Reproducing or extending your work becomes an unreachable dream.

This packaging problem varies in magnitude and complexity from field to field. Where some software may require a cross platform build system capturing the compiler flags for weaving together a fleet of system libraries, other software simply requires a download and some documentation of the steps to execute. These case studies spanned the gamut therein.

Many case study authors noted that their data analysis pipeline relied on a specific computational platform or build environment. Their dependencies may be limited to certain platforms or features of the build environment may need to be customized for various options to function. These environment issues quickly become too complex to manage as a manual feat and must be packaged in a robust, cross-platform way if they have any hope of succeeding.

Case study authors packaged their work in myriad ways. Lightweight strategies were often fragile to cross-platform-configuration issues (e.g. bash scripts and makefiles). More robust solutions (e.g. virtual machines), however, are usually more clunky and often less transparent. While a one-click-download runnable virtual machine may be the most reliable option for replicating a simulation, it is also the most opaque to the user. Compromise solutions such as configuration and build systems (e.g. CMake) are often simultaneously clunky and transparent. Notably, there are subtle differences between these solutions. In particular, build and configuration systems are often more bespoke (and fragile) than broader (and often rigid) packaging systems like conda/bundler/packrat.

It's now thirty years after the invention of autotools, but cross platform configuration, build, and packaging systems are not yet a solved problem. Somehow, scientific software developers still await a robust and universal solution. Thankfully, there is hope. The tech

industry, facing similar issues, has developed portable container management systems (Bernstein, 2014) such as Docker and Kubernetes. These technologies are enabling a new generation of scientific packaging tools for reproducible and open science (e.g. tmpnb, binder, ReproZip (Chirigati, Shasha, & Freire, 2013), Code Data Environemnt (Guo, 2012), etc.).

Need: Improved configuration and build systems for portably packaging software, data, and analysis workflows.

Hardware Access

The build system situation is a special small-scale case of a larger problem of variable hardware access. At that end of the spectrum, there are challenges getting a workflow running on your collaborator's laptop in addition to your own. At the other extreme is access to high performance computing hardware or unique experimental devices.

The Large Hadron Collider is often brought up as an example of an experiment so large that it will never be repeated in a different experimental location on a comparable device, so in some ways it fails the verifiability test. Many fields suffer this on a small scale at the data collection stage, where, for example, an experiment can't reproduced by just anyone; it must be reproduced by someone with a similarly configured MRI machine.

This hardware access issue was noted at the data collection step in the case studies, but it was also noted at the analysis step. That is, when scientific simulations or large scale data analysis is conducted on very high performance, high capacity, or high throughput computer systems, it is similarly vulnerable to being irreproducible. Not only is access to such resources limited and the hardware often unique, but the sheer amount and variety of metadata likely needed to reproduce results without complication is often enormous. While high throughput computing is being democratized by cloud computing resources, the capability computing of high performance computing machines, necessary for some applications, is not yet replaced by those services. Even cloud computing has limitations, as noted by case study author Arendt, whose collaborators found connecting to cloud servers was limited by bandwidth constraints in Alaska.

Need: Reproducibility at scale for high performance computing. A detailed discussion of this need can be found at (Hunold & Träff, 2013)

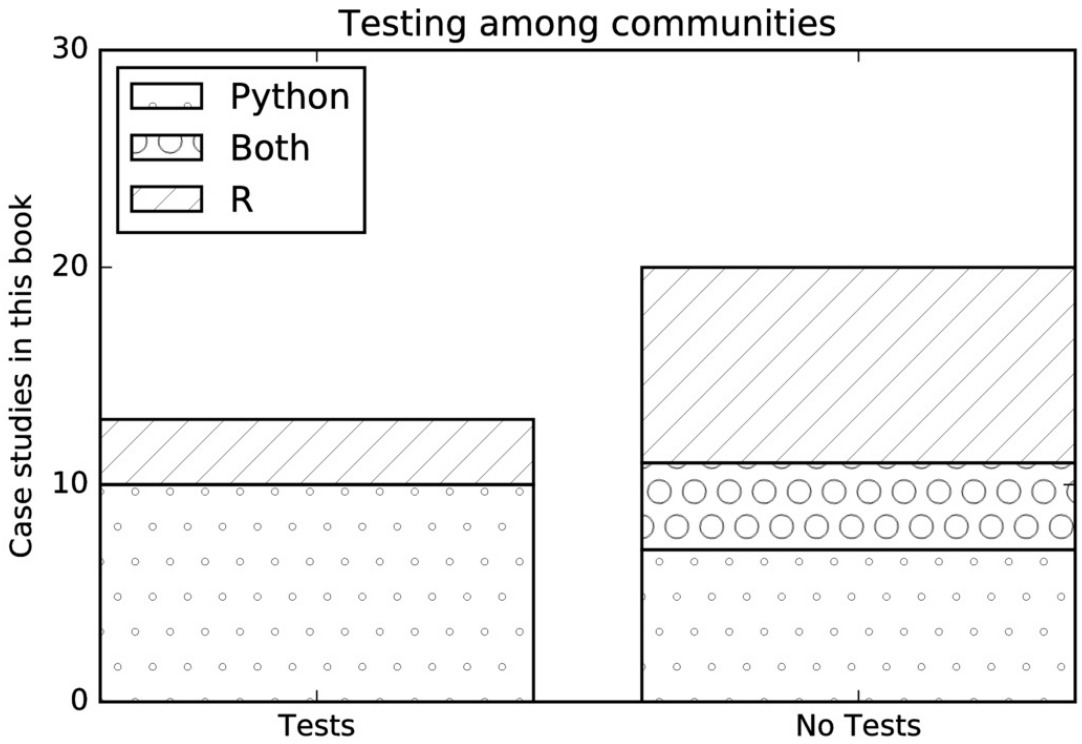
Need: Standardized hardware configurations and experimental procedures for limited-availability experimental apparatuses.

Testing

Many case study authors who developed code beyond simple scripts discussed testing that code systematically. This practice, in addition to being hygenic and improving robustness, is a type of self-check on reproducibility. Software tests and continuous integration in particular allow software authors to automate frequent checking that code consistently performs as expected even as new features are added.

Experiences varied. Some consider tests to be a core element in their workflows and emphasized unit testing -- comprehensive atomic tests at the function level -- for providing confidence that their work can be reproduced. Other case study authors perform integration tests before trusting a result, but not as an active part of development. In all cases, when tests were mentioned in the case studies, the authors were convinced of their utility at saving time and energy. However, not all case studies mentioned tests at all. One is left to wonder what is stopping those and other scientists from adopting these practices. Perhaps they are discouraged by the perceived effort of unit testing or the trade-offs of time spent now versus time saved later. In many cases, a lack of familiarity with unit testing may be the barrier.

It is also worth noting that the use of testing frameworks varied by language more than by scientific domain. In particular, the case study authors who used primarily Python reported testing at a much higher rate than case study authors who rely primarily on R. This is summarized in the figure below.



Of course this is not a statistically significant sample, so conclusions about these communities are somewhat premature, but resulting conversations have indicated to us that this difference in community adoption of testing practice may indeed be present.

Need: Better understanding of why researchers don't respond to the delayed incentives of unit testing as a practice.

Need: Norms encouraging greater adoption of unit testing irrespective of programming language.

Publishing

The most universally shared step in the research process is publication. In a literate programming sense, papers can be integrally automated and "runnable." For the majority of the case studies, however, the production of the research paper capturing the work was reported to be somewhat independent of the science. In other cases a more literate programming method was adopted through the use of Jupyter notebooks or judicious use of rmarkdown.

The workflows were very tool-driven in the sense that the tools used defined the way that the workflow progressed. The factions within the case studies included a LaTeX-based group, a knitr/rmarkdown/sweave contingent, and a frustrated Microsoft Word contingent.

Microsoft Word track changes deserves a special place in this discussion, so ubiquitous is its use worldwide and so consistent were the experiences of the four who mentioned it. Each scientist that mentioned Word had effectively two things to say:

(1) We used it because a collaborator couldn't figure out LaTeX and

(2) Track Changes certainly tracks changes, but is frustrating because the merging limitations mean that edits must be made in series rather than in parallel.

A reproducible paper is a large and varied task, perhaps demanding its own separate workflow.

Need: Broader community adoption around publication formats that allow parallel editing (i.e. any plain text markup language that can be version-controlled in a distributed manner.). Tools such as Overleaf and SageMathCloud are a beginning toward making LaTeX more approachable, but greater adoption is needed.

Data Versioning

Mere data storage is not always sufficient for the purpose of reproducibility. Occasionally, data may need to be versioned so that changes can be tracked, evaluation and cleaning steps can be rewound, and work can be extended.

Case study authors often either noted they were not versioning their data or noted that they were struggling to find a good way to do so. Since this challenge -- versioned storage of larger (or more varied, or high velocity) data -- is currently being encountered in the "big data" age of the software industry, active innovation in industry and new solutions are already being implemented.

Tools being developed to streamline the process of data versioning include GitHub Large File Service, Dat, git annex, datalad and others. While these operate in different ways, they typically involve compression, tracking of changes, and efficient retrieval.

Need: Greater scientific adoption of new industry-led tools and platforms for data storage, versioning, and management.

Time and Incentives

Perhaps the most vexing impediment to reproducibility the case study authors and their collaborators suffered from was a lack of time, incentives, or both.

Some case study authors, perhaps as a symptom of their recognition of the importance of reproducibility, were incentivized by confidence in the efficiency of these practices. Many noted the time they saved when repeating calculations, making modifications to analysis, and extending past work.

Conversely, the sentiment that ``time and efforts spent on creating reproducible research are not very well rewarded" (case study author Dr. Valentina Staneva) was eachoed by a few authors. This need for additional reward, support, and recognition for reproducible work is an institutional infrastructural issue especially in the academy where metrics for promotion and tenure are tied explicitly to papers and often fail to account for reusable software.

But, while the promotion and tenure process is in need of modernization, funds-granting organizations are moving faster. Private foundations like Moore, Sloan, and Helmsley are leading the charge by supporting large initiatives directed at scientific software reproducibility and transparency. Similarly, government institutions like NSF, NIH, and (less quickly) DOE, are incorporating requirements for openness and data planning as well. Similarly, some journals are implementing data and software submission requirements to incentivize reproducbility at the publication stage. The efficacy of these increased standards for publication and funding, however, depend fundamentally on the enforcement mechanism.

For the incentives to compel action, paper referrees must be willing to give due diligence by trying to run the submitted code and grant performance reviewers must be similarly be willing to review data management plan compliance.

Need: Increased community recognition of the benefits of reproducibility.

Need: Incentive systems where reproducibility is not only self-incentivizing.

Data restrictions

In the same way that transparent analysis is core to reproducing scientific work, access to raw data can also be essential for reproducibility. Indeed, it can be necessary for confirming conclusions during review and exploring alternative methods during extension by other scientists. In some fields, however, data access is legally restricted. Some such data restrictions concern human subjects research, such as survey and private medical data. Some restrictions concern national security, such as the restriction of export controlled nuclear data or risk map data. Researchers in these fields are therefore limited in their ability to do completely open science, but can often, behind the export control or IRB wall, share analysis methods with colleagues who do have access to the data.

Need: Standards around scrubbed and representational data so that analysis can be investigated separate from restricted data sets.

Actionable Recommendations

We are not the first to discuss reproducibility. Nor shall we be the last. Thus, many themes were repeated among the recommendations of the scientists:

- version control your code
- open your data
- automate everywhere possible
- document your processes
- test everything
- use free and open tools

Less common refrains that are already well established best practices within the current reproducibility climate include:

- avoid excessive dependencies
- when dependencies can't be avoided, package their installation

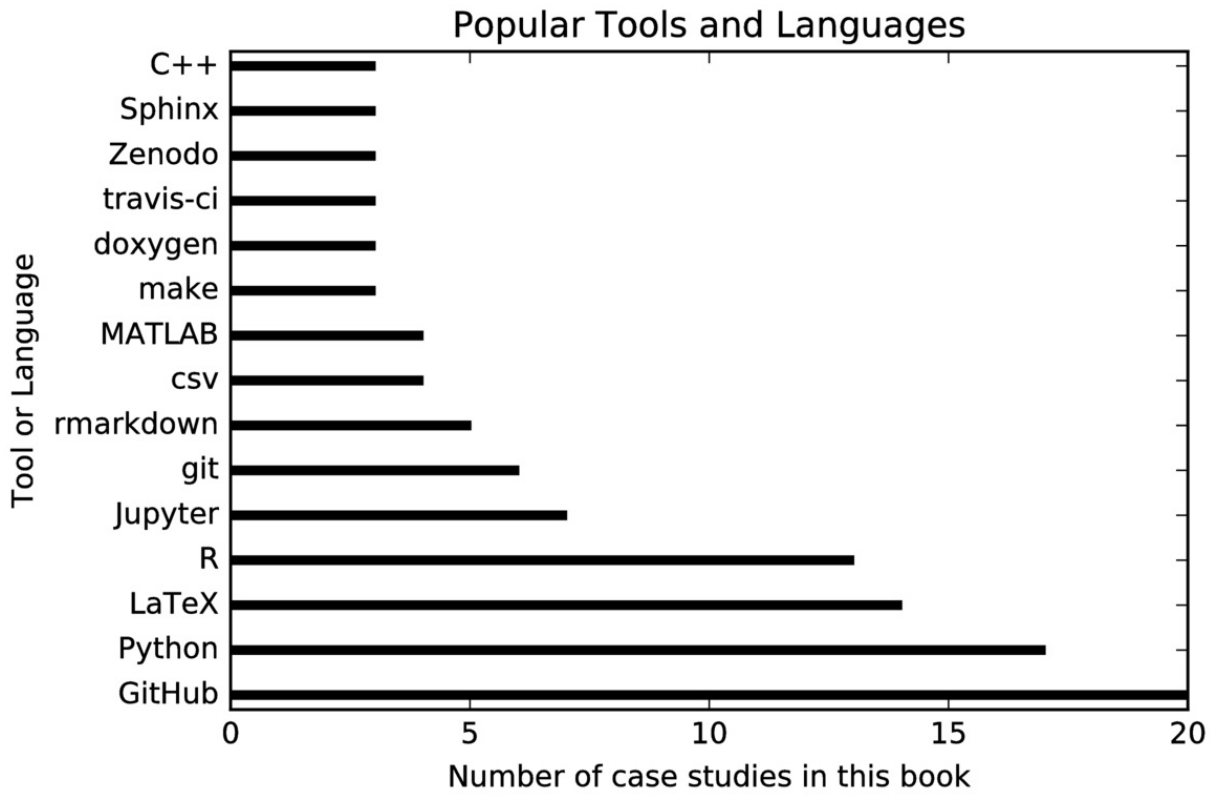
- host code on a collaborative platform (e.g. GitHub)
- get a Digital Object Identifier for your data and code
- avoid spreadsheets, plain text data is preferred ("timeless," even)
- explicitly set pseudorandom number generator seeds
- workflow and provenance frameworks may be too clunky for most scientists

At the core of many of these issues are human concerns around incentives and delayed return on investment. Education and community development will be needed to solve those issues where tool improvements fail to help.

Tools Used

The vast majority of scientists used a programming language such as R or Python to churn through analysis and automate processing.

Nearly all reported use of GitHub-based version controlled repositories at the core of their research work. Additionally, scientists often cited use of an ecosystem of tools appropriate for their work. The following sections categorize tools by their purpose, rather than by the ecosystem (e.g. R vs. Python) that they are found in most commonly.



Publishing

For publishing, the case study authors improved their reproducibility with What You See Is What You Mean (WYSIWYM) mark-up languages. Some preferred LaTeX/Overleaf, while others preferred the R/Knitr/RMarkdown/Sweave ecosystem. In combination with a text editor and distributed version control, both reproducibility and simultaneous collaboration are improved by these plain text mark-up languages.

Data Handling

The scientists used many different formats and systems for storing and cleaning their data. Some storage systems include both hierarchical (e.g. HDF5) and relational (e.g. SQL) database systems. Of particular note, the R community representation among these case studies boasted use of the RStudio IDE as part of the way they streamline access to the collection of R tools which enable some of these data repository solutions as well as data cleaning and exploration tasks.

Additionally, emerging ``data lakes'' for archived storage and retrieval such as Dataverse (King, 2007) were also mentioned. Case study authors even noted that specialized data lakes exist for certain scientific domains. Neurovault (Gorgolewski et al., 2016) was mentioned for neuroscience, but similar solutions exist other fields as well (e.g. Dryad (H. C. White, Carrier, Thompson, Greenberg, & Scherle, 2008) for ecology).

Testing Frameworks

In the Python ecosystem, tests can be run with frameworks such as nose or unittest. In C++, one can use Googletest. In both Python and C++ projects, testing frameworks were mentioned by multiple case study authors.

Although very few R users mentioned unit testing their code, the language does have options for unit test frameworks, with the testthat package (Wickham, 2011) being the most widely-used unit testing framework. Other available packages exist as well, such as the RUnit package.

Continuous Integration

Although few of the case study authors mentioned continuous integration, its use was lauded as essential. Reproducible practices are easiest to adopt when they require no time from the scientist. Even better, practices that save time are even easier to adopt. Continuous integration is just such a practice.

To get scientists to regularly run the tests for their software on a variety of platforms, don't require any effort of the scientist. That is, outsource the task of building and running the tests to the computer with a continuous integration system. Essential to production software,

continuous integration servers like CTest, Travis-CI, Jenkins, Batlab, and many others enable scientists to focus on implementation without worrying about checking constantly whether they're introducing bugs. If they introduce a bug, the continuous integration server will notice and send out and email or publish a status report.

DOI Management

A primary incentive for scientists is citation. Accordingly, the efforts they put toward reusable workflows will only seem worthwhile if those data, scripts, libraries, and analyses can be cited. Thus arises the DOI. Many methods exist for putting a citeable, persistent, digital stamp on one's code and data.\ Common services for archiving digital objects and providing DOIs were mentioned in the case studies including Zenodo, Figshare, and the Open Science Framework.

Other Recommendations

Perhaps most interesting among all of the recommendations are some insights that were only noted by one or two of the scientists. Some of these recommendations may be impactful if they see broader adoption.

Post Flurry Refactoring

Chapter authors Randall LeVeque and Rachel Slaybaugh each expressed their own version of the following idea:

> Make a habit of cleaning up code used to produce final results so that it's well documented and all the necessary steps are clearly laid out. Then run through them from scratch if possible to insure that it works. Even if you don't plan to share it with others, your future self will thank you.

This kind of workflow doubles down on the importance of documentation and clarity for users (including your future self). By emphasizing good practices during the day-to-day, this kind of workflow ensures that code is consistently useable by its author during development. By similarly employing best practices during the release or publication stage, this type of workflow effectively double-checks the reproducible nature of the work.

This workflow concept is especially enamoring because it recognizes the humanity of the scientist, who in a day-to-day work environment may let a few tasks necessary for full reproducibility to slip through the cracks. It then corrects for that element of human frailty by budgeting time at the wrapping-up stages to correct for that human error and solidify the process, like a time capsule for the future.

Standardized Data Formats

In the case studies, it was mentioned that a GIS standard is needed to help unify work with maps and geospatial data. This unification of work through collectively adopted standards applies to other fields as well. When a proliferation of standards complicates collaboration within a scientific domain, community agreement on a single standard can allow reproducibility across research groups.

A number of fields have long since successfully adopted data format standards, such as the Flexible Image Transfer System (FITS) files used in astronomy or the evaluated nuclear data files (ENDF) used in nuclear physics. Efforts are ongoing to standardize formats in other fields, where formats may be absent, insufficient, or (possibly worse) proliferant. For example, there is work in the emergent, data-driven field of neuroimaging to establish a standard for neurological images and see that standard adopted across the field (the Brain Imaging Data Structure).

It's worth noting, though, that even when a scientific domain has a community standard, it may not translate well in interdisciplinary work, when internal domain norms around data formats may hamper efforts to communicate. This can be ameliorated if domain standards are more universal standards, based perhaps on common data formats (e.g. SQL databases, HDF5, plain text).

Need: Community adoption for file format standards within some domains.

Need: Domain standards which translate well outside of their own scientific communities.

Conclusion

Many positive lessons came from this set of case studies. One was the reassurance that git and GitHub as well as automation in general are now ubiquitous among the case study authors - scientists seeking reproducibility. Accordingly, 'timeless' plain text file formats are preferred for code and small scale data. A return to transparent open formats based in plain text bodes well not just for reproducibility, but also for its siblings openness and transparency.

Some core needs that were identified include:

- Better education of scientists in more reproducibility-robust tools.
- Widely used tools should be more reproducible so that the common denominator tool does not undermine reproducibility.
- Improved configuration and build systems for portably packaging software, data, and analysis workflows.

- Reproducibility at scale for high performance computing.
- Standardized hardware configurations and experimental procedures for limited-availability experimental apparatuses.
- Better understanding of why researchers don't respond to the delayed incentives of unit testing as a practice.
- Greater adoption of unit testing irrespective of programming language.
- Broader community adoption around publication formats that allow parallel editing (i.e. any plain text markup language that can be version-controlled.
- Greater scientific adoption of new industry-led tools and platforms for data storage, versioning, and management.
- Increased community recognition of the benefits of reproducibility.
- Incentive systems where reproducibility need not be self-incentivizing.
- Standards around scrubbed and representational data so that analysis can be investigated separate from restricted data sets.
- Community adoption for file format standards within some domains.
- Domain standards that translate well outside of their own scientific communities.

While building and installation of dependencies remains a critical stumbling block, the most universal problems are human in nature, ranging from establishing basic toolkit familiarity within a team to motivating reproducible workflows according to their long term benefits. Similarly, though community norms both in and across domains are still in need of unification, many scientists interested in reproducibility are converging on a set of best practices for reproducible, open, robust science. Finally, some solutions - especially those related to human incentives - await institutional changes.

References

Barba, L. A. (2012). Reproducibility PI Manifesto. http://doi.org/https://dx.doi.org/10.6084/m9.figshare.104539.v1

Bernstein, D. (2014). Containers and Cloud: From LXC to Docker to Kubernetes. *IEEE Cloud Computing*, *1*(3), 81–84. http://doi.org/10.1109/MCC.2014.51

Chirigati, F., Shasha, D., & Freire, J. (2013). Reprozip: Using provenance to support computational reproducibility. In *Presented as part of the 5th USENIX Workshop on the Theory and Practice of Provenance*. Retrieved from https://www.usenix.org/conference/tapp13/technical-sessions/presentation/chirigati

Donoho, D. L., Maleki, A., Rahman, I. U., Shahram, M., & Stodden, V. (2009). Reproducible Research in Computational Harmonic Analysis. *Computing in Science & Engineering*, *11*(1), 8–18. http://doi.org/10.1109/MCSE.2009.15

Gorgolewski, K. J., Varoquaux, G., Rivera, G., Schwartz, Y., Sochat, V. V., Ghosh, S. S., … others. (2016). NeuroVault. org: A repository for sharing unthresholded statistical maps, parcellations, and atlases of the human brain. *NeuroImage*, *124*, 1242–1244. Retrieved from http://www.sciencedirect.com/science/article/pii/S1053811915003067

Guo, P. (2012). CDE: A tool for creating portable experimental software packages. *Computing in Science & Engineering*, *14*(4), 32–35. Retrieved from http://scitation.aip.org/content/aip/journal/cise/14/4/10.1109/MCSE.2012.36

Hunold, S., & Träff, J. L. (2013). On the State and Importance of Reproducible Experimental Research in Parallel Computing. *ArXiv:1308.3648 [Cs]*. Retrieved from http://arxiv.org/abs/1308.3648

King, G. (2007). An introduction to the Dataverse Network as an infrastructure for data sharing. *Sociological Methods & Research*, *36*(2), 173–199. Retrieved from http://smr.sagepub.com/content/36/2/173.short

Stodden, V. (2010). The Scientific Method in Practice: Reproducibility in the Computational Sciences. *SSRN Electronic Journal*. http://doi.org/10.2139/ssrn.1550193

White, H. C., Carrier, S., Thompson, A., Greenberg, J., & Scherle, R. (2008). The Dryad data repository: A Singapore framework metadata architecture in a DSpace environment. *Universitätsverlag Göttingen*, 157. Retrieved from http://www.oapen.org/download?type=document&docid=353956#page=173

Wickham, H. (2011). Testthat: Get started with testing. *The R Journal*, *3*(1), 5–10. Retrieved from http://immagic.com/eLibrary/ARCHIVES/GENERAL/JOURNALS/R110623I.pdf#page=5

Wilson, G., Aruliah, D. A., Brown, C. T., Chue Hong, N. P., Davis, M., Guy, R. T., … Wilson, P. (2014). Best Practices for Scientific Computing. *PLoS Biol*, *12*(1), e1001745. http://doi.org/10.1371/journal.pbio.1001745

Building Towards a Future Where Reproducible, Open Science is the Norm

Karthik Ram and Ben Marwick

The traditional boundaries between domain researcher and scientific programmer have been blurring rapidly over the past decade. Pressing societal issues such as global climate change, disease outbreaks, endangered species conservation, and drug discovery cut across traditional scientific silos. Successfully answering such interdisciplinary problems will require researchers to not only access and process ever-increasing quantities of data but also leveraging them in the context of their domain expertise. The cost of collecting this data is also dropping, and new technologies in every aspect of our lives now enable cheap and easy collection of high volumes of highly diverse data. As a result, scientific endeavors have come to rely on massive amounts of data being analyzed with a disparate set of tools and technologies.

Another consequence of the high volumes of data and increasing diversity of software tools is that scientists now produce a vast array of research products such as data, code, algorithms, in addition to traditional publications (Heather A Piwowar & Vision, 2013). Yet, until recently, funding agencies such as the US National Science Foundation did not consider any outputs beyond traditional peer-reviewed publications, as credit-worthy outcomes. While some fields, such as astronomy and high energy physics, have long recognized the importance of making the entire research pipeline publicly available, this is far from normal in most areas of science. In the last decade, many areas of science have had high-profile cases of non-reproducible research. Well-publicised retractions include Diederik Stapel in social psychology, Anil Potti in cancer research, Carmen Reinhart and Kenneth Rogoff in economics, and Marc Hauser in evolutionary biology. In addition, large-scale efforts to reproduce biomedical (Begley & Ellis, 2012) and psychological experiments (Open Science Collaboration, 2015) suggest that the prevalence of non-reproducible research has been underestimated, resulting in news headlines declaring a 'reproducibility crisis' in science. The issue of reproducibility is particularly timely given the recent rise in retractions from high profile journals (Van Noorden, 2011). While some aspects of this crisis are due to bad agents, there are also broader systemic problems that result in the production of non-reproducible research. In this chapter we briefly survey some of the gaps, challenges, and opportunities for improving the reproducibility of research.

Gaps: Reproducibility is hard

For many scientists, generating reproducible research is difficult because of the diversity of hardware and software in their workflow. For example, consider an analytical instrument that outputs data in a particular format, which then needs to be transformed and rearranged in several ways before being into input into a sequence of several different specialized computer programs for analysis. As the data is moved between each program - we can call this between space a 'gap' - additional manual inspection, readjustment and perhaps combination with other data is required. Gaps result from disconnected tools that have been combined to suit a specific research problem. The problems of handling the data in the gaps are typically solved by bespoke methods that are unique to each group or individual, using tools that were never intended for scientific research (e.g. `Make`), and are rarely produced with the intention of making them public. The custom and expedient nature of these gap-filling methods make it difficult to capture the entire workflow to enable other researchers to reproduce the result. Because of the high diversity of research problems and tools across different areas of science, attempts to integrate these into a single platform have had limited uptake outside of bioinformatics, where many of these pipeline frameworks were first developed (Leipzig, 2016).

Outside of bioinformatics, some researchers are filling these gaps by using literate programming style that allows programming code and narrative text to be interwoven within a single document. One example of this is the work of FitzJohn et al. (2014), who combined the R package knitr with `Make`, among other tools, to create a self-contained and self-documenting workflow for their ecological study. A similar example is the archaeological study by Clarkson et al. (2015), who also used knitr to combine narrative text and programming code to process data from diverse sources. Clarkson et al. also used Docker to provide a self-contained computational environment for their workflow, so that they key software dependencies could be bundled into their research repository with the data. This example is described in more detail in Marwick (2016).

We believe that the use of knitr in these two examples is part of a broader trend in the adoption of executable notebooks in science broadly. An executable notebook is a framework that allows narrative text (and its accessories, such as citations, figures, tables, etc.) and programming code that generates the figures and tables to be interwoven in a single source document. Among R users knitr (a descendant of Sweave) is currently the dominant tool for producing executable notebooks. For Python users there is Jupyter, which can also be used with other programming languages. Our hope is that executable notebooks will be the solution to the problem of gaps in research workflows.

Two other key elements of filling the gaps in the scientific workflow are training for scientists in efficient computer programming, and infrastructure for sharing and collaborating with code. Great progress has been made in these areas, with organizations such as Software Carpentry and Data Carpentry developing and delivering volunteer-led training workshops to researchers across the sciences. Their lesson materials are open source and online to

enable self-study for researchers unable to attend workshops in person. The infrastructure for sharing and collaborating has been made available by services such as GitHub and BitBucket. These services, based on the Git version control system, allow researchers to share their code, organize contributions to scientific software projects, and discover code produced by other researchers (Ram, 2013). In our view, the increase in demand by researchers for training in programming, and the rising popularity of GitHub as public repository for scientific code, reflect a trend toward increasing openness in the scientific process, and in the reproducibility of research.

Challenges: Changing the incentives

Traditional incentives in science prioritize highly cited publications of positive, novel, tidy results. The practice of enabling the reproducibility of those results to be assessed by making the data and code publicly available is not part of the traditional incentives of science. However, individual researchers can gain significant personal benefits for their open science efforts. While preparing and depositing data into an easily discoverable repository requires an upfront time investment, there are numerous benefits to doing so. The National Science Foundation (NSF), for example, requires a data management plan as part of the proposal (Donnelly & Jones, 2010) and also count these endeavors under their merit guidelines (NSF, 2012). Further, authors who share data alongside publications are also likely to be cited more (Heather A. Piwowar, Day, & Fridsma, 2007) and benefit from alternate metrics which are strongly correlated with citations (Heather A Piwowar & Vision, 2013).Citation benfits have been demonstrated for code sharing in research publications (Vandewalle, 2012).

The citation advantage from sharing research data has been demonstrated in numerous disciplines. Henneken and Accomazzi (2011) analysed 3814 articles in four astronomy journals and found that articles with links to open datasets on average acquired 20% more citations than articles without links to data. Restricting the sample to papers published in since 2009 in The Astrophysical Journal, Dorch (2012) found that papers with links to data receiving 50% more citations per paper per year, than papers without links to data. In 1,331 articles published in Paleoceanography between 1993 and 2010, Sears (2011) found that publicly available data in articles was associated with a 35% increase in citations. Similar positive effects of data sharing have been described in the social sciences. In 430 articles in the Journal of Peace Research, articles that offered data in any form, either through appendices, URLs, or contact addresses were on average cited twice as frequently as an article with no data but otherwise equivalent author credentials and article variables (Gleditsch & Strand, 2003).

It is clear that researchers in a number of different fields benefit from a citation advantage for their articles that include publicly available datasets. In addition to increased citations for data sharing, Pienta et al. (2010) found that data sharing is associated with higher publication productivity. They examined 7,040 NSF and NIH awards and concluded that a research grant award produces a median of five publications, but when data are archived a research grant award leads to a median of ten publications. These studies suggest the investment of effort in improving reproducibility by sharing data can have payoffs in the traditional incentive system. These efforts are also advantageous in the broader, but very slow, shift in incentives that favor reproducibility over novelty that we sense is occurring in some fields.

The incentivisation of novelty has led to widespread anxiety that sharing of data will result in getting one's own research scooped, and a lack of appropriate rewards for time spent documenting and sharing methods (Heather A. Piwowar et al., 2007). Even when there is an appreciation for open science, the technical challenges such as lack of appropriate skills and knowledge of best practices can hinder this process. By addressing both the cultural and technical challenges we can create a community of practice that would ensure that data sharing is the norm rather than the exception (Birnholtz & Bietz, 2003).

An important step forward in establishing norms for sharing data and using shared data is Daniel Kahneman's (2014) 'reproducibility etiquette'. He proposes that researchers intending to use an open dataset or code repository contact the original authors. When working with code written by others, he especially recommends having a discussion with the authors of the code. The purpose of this to give them a chance to fix bugs or respond to issues you have identified before you make any public statements (Eglen et al., 2016). He also recommends citing code and data in an appropriate fashion. In addition, researchers should also pay close attention to the license agreements attached to specific pieces of code, software, and data products as they unambiguously state the conditions under which such work can be used, adapted, and redistributed (Morin, 2012). Although this is a simple and non-technical detail, we expect that if these values become normalized than the common anxiety of sharing code and data will diminish, and more researcher will feel comfortable to make their work more reproducible.

Making one's research meaningfully reproducible is a significantly more involved effort than merely sharing a handful of scripts and datasets via open repositories (FitzJohn et al., 2014; Mesnard & Barba, 2016). Such activities represent the first of a series of rigorous steps necessary to make a research product truly reproducible. Many of the challenges lie in the analysis phase where the provenance of all inputs and dependencies need to be carefully tracked using automated workflows. It would be naive to suggest that researchers can make their work fully reproducible by following a few simple steps. Even when experienced

computational researchers such as FitzJohn et al and Mesnard et al began their study with full reproducibility in mind, challenges around inadequate tooling and workflow complexity made the task quite hard.

Despite such roadblocks, rapid improvements in tools and workflow technology will continue to lower barriers to reproducibility across various disciplines. In the meantime, any level of reproducibility brings us closer to overcoming the challenges.

Opportunities: The promise of open science

Science is in the midst of a dramatic transformation that is being driven by increasing access to large amounts of heterogeneous data. The long-established model where sole researchers collect and analyze their own data will no longer be the dominant approach and instead be replaced by one where disparate datasets from multiple sources are used. It is now widely accepted in many scientific disciplines that existing datasets can be used to solve novel problems not anticipated by the original investigator (Faniel & Zimmerman, 2011; Nielsen, 2012; Whitlock, McPeek, Rausher, Rieseberg, & Moore, 2010). Such open data can serve as a research accelerator, enabling scientists to rapidly collaborate on knowledge creation and synthesis efforts (Neylon, 2012). A similar pattern of collaboration and reuse is also emerging across the scientific software stack as is evident in the case studies described in this book. A rich suite of open source tools are rapidly lowering barriers to collaborations across disparate domains and institutions and helping accelerate the rate of scientific discovery in ways previously unimagined.

This new era of open science is enabling a community of practice that allows collaborations to scale more easily while various links in the chain of scientific reasoning to be used in different contexts. Part of the reason why scientific workflows are not properly curated or shared are an artifact of the way the credit system currently works in science. Due to insufficient incentives to share, original investigators spend very little time on activities other than publishing. As a result, valuable data, code and critical details on implementation are prone to disappearing or becoming less useful over time (Michener, Brunt, Helly, Kirchner, & Stafford, 1997). However the scholarly landscape is changing to provide both the incentives and means for increased data sharing.

Until recently, researchers who put time and effort into documenting and sharing data and details of their analysis were considered outliers. Now the scholarly landscape is in the midst of a revolution, and among the emerging changes are new incentive mechanisms for reporting research impact. For example, altmetrics (H. Piwowar, 2013) track influence of research outputs and data products outside of the traditional citation framework, providing more ways to measure success. Organizations and repositories including DataCite, figshare, Zenodo, Dryad, DataONE, and others provide the means for data to be cited independent of publications. Papers that share data are more likely to receive citations (Heather A. Piwowar

et al., 2007), and people who collect and deposit well-curated data can receive measurable recognition for their efforts. This is especially important as the scientific community is calling for data citation to be part of the tenure and promotion practice (Parsons, Duerr, & Minster, 2010).

Once a critical mass of scientists share their data and code, it would serve as a multiplier effect and allow disparate groups of researchers to rapidly solve problems such as climate change, (need a few other applications from other domains) (Peterson et al., 2002). We see these collaborations resulting from sharing data and code as one of the great opportunities to come from reproducible research.

Discussion

Our discussion so far has focused on the role of the researcher, and the gaps, challenges and opportunities they face. However, there are a few other key groups that are relevant to changing the norms to enhance the reproducibility of research.

Many funders such as the National Science Foundation (NSF) and National Institutes of Health (NIH) have long maintained data sharing requirements although they have been rarely enforced (Borgman, 2012). However, recent changes to funding policies have made these requirements more stringent and explicit. As of 2011, new NSF proposals require a data management plan (Donnelly & Jones, 2010). This plan requires details on how the data will be documented and where it would be deposited upon completion of the effort.

Many fields in science are in the midst of a data revolution and have adapted to the emerging challenges to varying degrees. At one extreme, disciplines such as astrophysics have fully embraced data driven science by developing and supporting the infrastructure, computational methods, and the culture to derive the most value from the data they generate (Venugopal, Buyya, & Ramamohanarao, 2006). At the other, many data-rich disciplines still lack the culture or the practice to leverage or benefit from past endeavors. Funding agencies can serve as sources of change for these disciplines where cultural change is slow.

A second group for whom reproducible research provides new opportunities are research libraries. Concerns about reproducibility now transcend individual disciplines, and there is a need for research institutes and university campuses to provide resources to support reproducible research. Researchers need information on what tools and services are available for reproducible research, and how they can get training for these. Libraries are becoming sensitive to this need, and some have started providing guides to data management planning, software tools for reproducible research, and training sessions. Two particularly good examples that we are aware of are the University of Utah Library Reproducibility of Research resource and the NYU Libraries' Guide to reproducibility.

Journal editors are a third group in the research community that have important opportunities to enact change in support of reproducibility. For example, journal editors could increase the importance of reproducibility by requiring (and enforcing) mandatory full data and code deposition, encouraging and even soliciting replication studies, and supporting reviewers who attempt to reproduce studies while reviewing the paper. Several journals have introduced new guidelines for authors and made specific proposals that attempt to address the problems of non-reproducible research (Begley & Ioannidis, 2015). We see this opportunity for editors to support reproducibility as part of a broader cultural change, one occurring at a generational scale, but that will substantially change the way we share our research outputs.

Conclusion

In this chapter we've surveyed some of the gaps, challenges and opportunities relating to reproducible research. We believe that for the majority of researchers there are now mature software solutions to the joining the gaps of a complex workflow. We are starting to see convergence in several disciplines on executable notebooks as one type of software for tackling the challenges of reproducible research. Reproducible research can provide benefits in the traditional incentive system, but our view is that some of the most compelling opportunities are in how incentives - and the practice of science more generally - can be changed by groups such as funding agencies, journal editors and libraries. Finally, we see opportunities for researchers in the form of new and more diverse research collaborations, equipped with uniquely large datasets to take problems of general interest and wide benefit to humanity. Our observations are that the pace of changes toward more reproducible research is accelerating, but that these are changes of a generational scale and so training, persistence, and optimism are vital to support the technical and policy efforts.

References

Begley, C. G., & Ellis, L. M. (2012). Drug development: Raise standards for preclinical cancer research. *Nature*, *483*(7391), 531–533. Journal Article. Retrieved from http://dx.doi.org/10.1038/483531a

Begley, C. G., & Ioannidis, J. P. (2015). Reproducibility in science: Improving the standard for basic and preclinical research. *Circulation Research*, *116*(1), 116–126. http://doi.org/10.1161/CIRCRESAHA.114.303819

Birnholtz, J. P., & Bietz, M. J. (2003). Data at work: Supporting sharing in science and engineering. In *Proceedings of the 2003 international aCM sIGGROUP conference on supporting group work* (pp. 339–348). ACM.

Borgman, C. L. (2012). The conundrum of sharing research data. *Journal of the American Society for Information Science and Technology*, *63*(6), 1059–1078.

Clarkson, C., Smith, M., Marwick, B., Fullagar, R., Wallis, L. A., Faulkner, P., … others. (2015). The archaeology, chronology and stratigraphy of madjedbebe (malakunanja ii): A site in northern australia with early occupation. *Journal of Human Evolution*, *83*, 46–64.

Donnelly, M., & Jones, S. (2010). Template for a data management plan. *Digital Curation Centre. Retrieved July*, *12*, 2010.

Dorch, S. (2012). On the citation advantage of linking to data: Astrophysics. Retrieved from https://halshs.archives-ouvertes.fr/hprints-00714715/

Eglen, S., Marwick, B., Halchenko, Y., Hanke, M., Sufi, S., Gleeson, P., … Poline, J.-B. (2016). Towards standard practices for sharing computer code and programs in neuroscience. *BioRxiv*. http://doi.org/10.1101/045104

Faniel, I. M., & Zimmerman, A. (2011). Beyond the data deluge: A research agenda for large-scale data sharing and reuse. *International Journal of Digital Curation*, *6*(1), 58–69.

FitzJohn, R. G., Pennell, M. W., Zanne, A. E., Stevens, P. F., Tank, D. C., & Cornwell, W. K. (2014). How much of the world is woody? *Journal of Ecology*, *102*(5), 1266–1272. http://doi.org/10.1111/1365-2745.12260

Gleditsch, N. P., & Strand, H. (2003). Posting your data: Will you be scooped or will you be famous? *International Studies Perspectives*, *4*(1), 72–107. http://doi.org/10.1111/1528-3577.04105

Henneken, E. A., & Accomazzi, A. (2011). Linking to data - effect on citation rates in astronomy. *CoRR*, *abs/1111.3618*. Retrieved from http://arxiv.org/abs/1111.3618

Kahneman, D. (2014). A new etiquette for replication. *Social Psychology*, *45*(4), 310.

Leipzig, J. (2016). A review of bioinformatic pipeline frameworks. *Briefings in Bioinformatics*. http://doi.org/10.1093/bib/bbw020

Marwick, B. (2016). Computational reproducibility in archaeological research: Basic principles and a case study of their implementation. *Journal of Archaeological Method and Theory*, 1–27.

Mesnard, O., & Barba, L. A. (2016). Reproducible and replicable cFD: It's harder than you think. *ArXiv*, 1605.04339.

Michener, W. K., Brunt, J. W., Helly, J. J., Kirchner, T. B., & Stafford, S. G. (1997). Nongeospatial metadata for the ecological sciences. *Ecological Applications*, *7*(1), 330–342.

Morin, J. A. S., Andrew AND Urban. (2012). A quick guide to software licensing for the scientist-programmer. *PLoS Comput Biol*, *8*(7), 1–7. http://doi.org/10.1371/journal.pcbi.1002598

Neylon, C. (2012). Science publishing: Open access must enable open use. *Nature*, *492*(7429), 348–349. Retrieved from http://dx.doi.org/10.1038/492348a

Nielsen, M. (2012). *Reinventing discovery: The new era of networked science*. Princeton University Press.

NSF. (2012). US NSF - Dear Colleague Letter - Issuance of a new NSF Proposal & Award Policies and Procedures Guide (NSF13004). Retrieved from http://www.nsf.gov/pubs/2013/nsf13004/nsf13004.jsp?WT.mc_id=USNSF_109

Open Science Collaboration. (2015). Estimating the reproducibility of psychological science. *Science*, *349*(6251). http://doi.org/10.1126/science.aac4716

Parsons, M. A., Duerr, R., & Minster, J.-B. (2010). Data citation and peer review. *Eos, Transactions American Geophysical Union*, *91*(34), 297–298.

Peterson, A. T., Ortega-Huerta, M. A., Bartley, J., Sánchez-Cordero, V., Soberón, J., Buddemeier, R. H., & Stockwell, D. R. (2002). Future projections for mexican faunas under global climate change scenarios. *Nature*, *416*(6881), 626–629.

Pienta, A. M., Alter, G. C., & Lyle, J. A. (2010). The enduring value of social science research: The use and reuse of primary research data. Retrieved from http://deepblue.lib.umich.edu/bitstream/handle/2027.42/78307/pienta_alter_lyle_100331.pdf

Piwowar, H. (2013). Altmetrics: Value all research products. *Nature*, *493*(7431), 159–159. http://doi.org/10.1038/493159a

Piwowar, H. A., & Vision, T. J. (2013). Data reuse and the open data citation advantage. *PeerJ*, *1*, e175.

Piwowar, H. A., Day, R. S., & Fridsma, D. B. (2007). Sharing detailed research data is associated with increased citation rate. *PLoS ONE*, *2*(3), e308. http://doi.org/10.1371/journal.pone.0000308

Ram, K. (2013). Git can facilitate greater reproducibility and increased transparency in science. *Source Code for Biology and Medicine*, *8*(1), 7.

Sears, J. (2011). Data sharing effect on article citation rate in paleoceanography. In *AGU fall meeting abstracts* (Vol. 1, p. 1628).

Van Noorden, R. (2011). The trouble with retractions. *Nature*, *478*(7367), 6–8. http://doi.org/10.1038/478026a

Vandewalle, P. (2012). Code sharing is associated with research impact in image processing. *Computing in Science and Engineering*, *14*(4), 42–47.

Venugopal, S., Buyya, R., & Ramamohanarao, K. (2006). A taxonomy of data grids for distributed data sharing, management, and processing. *ACM Computing Surveys (CSUR)*, *38*(1), 3.

Whitlock, M. C., McPeek, M. A., Rausher, M. D., Rieseberg, L., & Moore, A. J. (2010). Data archiving. *The American Naturalist*, *175*(2), 145–6. Retrieved from http://www.jstor.org/stable/10.1086/650340

Glossary

Ariel Rokem and Fernando Chirigati

Like other technical areas, the topic of computational reproducibility has its own terminology and jargon. The terms range from key concepts of the field, that are important when defining the parameters of reproducible research, to specific techniques and practices that are used in upholding computational reproducibility. Finally, there is a plethora of technical tools and practices that are mentioned througout this book. In this chapter, we provide some important definitions to help clarify the terms, techniques, and tools that are mentioned throughout the case studies and the other chapters.

Key Concepts

Scientific Experiment

A **scientific experiment**, or simply **experiment**, is a procedure carried out to validate or refute a hypothesis. In our modern times, many stages in a research project are done partially or entirely through the use of computer programs and processes, and that involve digital data that is consumed (**input data**) and produced (**output data**). This may include studies in which the experiments themselves are computational in nature. In this case, the experiment is often modelled as a **pipeline** (or **dataflow**): a sequence of *steps* that are connected by the flow of data, where the output data of a step is used as input data for the following step. A step can be represented by a computer program or a sequence of programs (a sub-pipeline), and it transforms the data it consumes as part of the procedure.

Reproducibility

Reproducibility is a cornerstone of science. Definitions vary greatly across scientific disciplines, but the meaning that we find most prevalent is the 'calculation of quantitative scientific results by independent scientists using the original datasets and methods' (Stodden, Leisch, & Peng, 2014). The goals of reproducibility go beyond duplicating someone else’s investigation: it also entails having reproducibility for yourself, defeating self-deception in scientific results (Ioannidis, 2005; Nuzzo, 2015), and extending another researcher’s methods to build your own work. Reproducibility is a matter of degree, not of

kind. We say that research is reproducible if reproducibility applies to the results to some extent. That is, some of the corresponding experiments and scientific methods are deemed to be reproducible.

Empirical and Computational Reproducibility

We can define different types of reproducible research as follows, adapted from Stodden (2014): empirical reproducibility, and computational reproducibility.

Empirical reproducibility entails communicating the procedure, protocols, equipment, and observations related to the experiment, but does not require making the computational assets (code and data) used during the research publicly available. This is often a minimal standard in science: published manuscripts contain descriptions and static figures and plots, and scientists need to follow these in hopes of building upon past research.

In **computational reproducibility**, in addition to the published manuscripts, the computational assets used to test all the hypothesis and derive the results are made available, which allows the computational processes to be reproduced verbatim and, in some cases, re-used. These assets may include, but are not limited to: the input data, either in extension (raw data) or in intension (a script that generates the data); the software (in binary or in source code); and the computational environment (computational dependencies and operating system information).

The notions of **verification** and **validation** are also commonly used when referring to reproducibility (Stodden et al., 2013): verification is concerned with the code solving the problem it claims to solve, while validation is concerned with the results being consistent with observations of the phenomenon being studied. In this sense, empirical reproducibility helps in the validation process, whereas computational reproducibility helps in the verification process, since the experiment execution can be investigated in more details.

With respect to the verification process, computational reproducibility helps identify if the code is not broken, and also pinpoint any statistical issues that may invalidate the results. For instance, *p-hacking* is a common bias in science where researchers select data or statistical analyses until non-significant results become significant. By having the data and all the artifacts, including the full chain of research events, one could tweak the different variables and vary the original analysis to detect how robust and significant the claims are.

Reproducibility Modes

Reproducibility can also be defined with respect to how results can be reproduced. Some of the terms often used in this regard are replicability, approximate reproducibility, and modular reproducibility.

Like reproducibility, there are conflicting definitions of **replicability** across different scientific domains. In some areas of science this is a synonym for **exact reproducibility**: the reproduced results are exact the same (meaning the exact same numbers) as the ones presented and discussed in the corresponding published manuscript. The computational assets, such as software, configuration parameters, and hardware, must be ideally the same to ensure replicability. Replicability guarantees reproducibility, but not the converse (Leek & Peng, 2015). In other areas, replicability can refer to a prior study being duplicated using the same procedures but with new data (Stodden et al., 2014).

Approximate reproducibility is related to having results that are similar to (and not the same as) the ones produced in the original experiment run. This often includes varying configuration parameters and input files to better verify how robust the experiment is, and perhaps simulating some steps that are harder to replicate. For example, an experiment that involves parallel and distributed computation may depend on the availability of massive serves for its replication; these steps can then be simulated or conducted at smaller scale to make its reproducibility feasible (Hunold & Träff, 2013). Also, some experiments are intrinsically difficult to replicate, such as the ones that require non-deterministic steps (random number generation) and access to third-party servers (code that is on servers cannot be controlled by researchers). Note that the reproduced results need to be consistent to the original ones to allow others to validate the experiment.

When an experiment supports **modular reproducibility**, its different steps and components can be reproduced individually, i.e., the experiment does not need to be reproduced in its entirety. For instance, if a single binary is shared for the entire experiment, it may be hard to reproduce only some of its steps; however, if the source code is made available, researchers will have more flexibility to use the parts of the experiment they want/need. Modular reproducibility allows **reusability**: the experiment can be more easily re-used for other purposes, thus making it possible for others to modify and build upon the original work.

Reproducibility Coverage

Another important aspect in reproducibility is its **coverage** (Freire, Bonnet, & Shasha, 2012): some experiments may not be reproduced in its entirety, including the ones that rely on data derived by third-party Web services or special hardware, or that require non-deterministic computational processes. But such experiments can, sometimes, be partially reproduced. For instance, if an experiment uses data that is derived by special or proprietary hardware, the data derivation process may not be reproducible, but the downstream analyses that use these data may be reproduced by others if the original data is made available.

Provenance

As the volume of digital data increases and the complexity of computational processes that manipulate these data grows, it is becoming increasingly important to manage their **provenance**. The Oxford English Dictionary defines provenance as *the source or origin of an object; its history and pedigree; a record of the ultimate derivation and passage of an item through its various owners*. Provenance helps determine the value, accuracy, and authorship of an object.

Computational provenance enables data products derived by computational processes to be interpreted and understood (Freire, Koop, Santos, & Silva, 2008). By examining a sequence of steps that led to a result, it is possible to obtain insights into the chain of reasoning used in the production of this result; to verify that the steps were performed according to acceptable procedures; and to identify what the inputs to the experiment were and where they came from.

Provenance is a critical ingredient for reproducibility (Freire & Silva, 2012). Providing detailed information about the provenance of results of an experiment tells about both the data and the sequence of steps that generated the findings. Through this information, it is possible to detect the required components of the experiment, and this facilitates the task of making it reproducible. The availability of such provenance information not only makes it possible to replicate the findings, but it also makes it easier to re-use and extend a result (by changing inputs and modifying the sequence of steps); in other words, detailed provenance allows modular reproducibility.

Provenance can be **describable** or **executable**. Describable provenance entails having a full description of the experiment (textual description, or a graph detailing all the steps) that serves to communicate in detail the computational aspects that one need to know to reproduce each step. Executable provenance, on the other hand, entails having an executable asset that can be directly used to reproduce the experiment (a binary or a scientific workflow (Davidson & Freire, 2008)).

There are different **provenance components** that must be captured to ensure the reproducibility of an experiment (Chirigati, Shasha, & Freire, 2013):

- ***Data*** entails the original input data used to execute the experiment, and the original output data to compare the raw results. Sharing the intermediate data (data produced by intermediate steps of the pipeline) may also be useful if some steps cannot be reproduced (see *coverage* in *Reproducibility*).
- ***Process*** entails all the computational programs and scripts used to execute the experiment. As mentioned before, this can be done by sharing either the source code or the binaries, which influences reusability.

- ***Environment*** entails all the assets belonging to the computational environment where the experiment was originally executed, which includes information about the operating system (e.g.: name and kernel version), hardware architecture (e.g.: 32 or 64 bits, number of computational nodes), and computational dependencies (e.g.: library and software packages on which the experiment depends to run). This component is important to allow an experiment to be portable to other computers, especially if they have different software and hardware systems.

Techniques

Version Control

Version control is a set of practices and tools originally used in software development to track the versions of software. These tools monitor, track and store changes to files within a circumscribed part of the file system, often referred to as a **repository**. The first generation of these systems are referred to as **centralized version control** systems (these include the **Concurrent Versions System (CVS)** and **Subversion (SVN)**). These systems rely on the existence and setup of a centralized server that stores the history of the changes to the code. In contrast, **distributed version control** systems (such as **git**, and **Mercurial**) do not depend on the presence of a centralized server. The history is instead stored together with the files in each user's computer. To facilitate collaboration and coordination of work on different users' computers, centralized servers are nevertheless often used as a common point for 'push' and 'pull' operations that synchronize the history between repositories stored on different computers and to merge work that is concurrently done on different files, or different parts of the same files by different users. Centralized servers can be set up on websites, and such websites offer other features. For instance, they display and allow browsing of the files in a virtual file-system-like website, and they provide web pages that can be used to browse the files in the repository, without downloading them. In addition, these websites provide for collaboration and communication among users (such as **bug-trackers**, pages in which errors in the code, or "bugs", can be reported and addressed). The use of version control tools in science has risen in recent years, with many large collaborative projects and institutions (e.g., CERN, LSST, and NCBI) using the services of websites such as **GitHub** to distribute and collaborate on software.

Literate Programming

Computer programs are read many more times than they are written (Wulf, 1977). Considering this fact, Knuth (Knuth, 1984) proposed that instead of focusing on computer programs as only a set of instructions to the computer, the focus of a computer program should be to explain what is (supposed to be) achieved through these instructions. This shift

in focus implies a more thoughtful approach to descriptive details of the software, such as function and variable names, and a substantial focus on documentation. In a research context, computer programs are embedded within documents, such as scientific papers. This practice is also described as 'literate computing', 'literate statistical programming', 'literate data analysis', and 'literate statistical practice', in recognition of the adoption of literate programming methods from a software development context into a data analysis context. Several systems, such as **knitr** and **Jupyter** allow the writing of documents, including papers, with the code embedded or interleaved with the text.

Data Publication

Full access to the computational assets that led to previously reported results are essential for **computational reproducibility**. **Data publication** (also known as **"data sharing"**) refers specifically to public availability of the data that was used (as distinct from the software, for example). If the data is stored digitally, this can be done by sending the data to specific collaborators, by creating digital copies, or making files available over the Internet. It can be done by uploading the data to publicly available websites that either can be accessed unencumbered, or require agreement to certain terms and conditions of use. In some cases, data size also limits the possibility of data publication and it is more practical to send physical copies of the data (for example, the so-called “connectome in a box” (Poldrack & Gorgolewski, 2014), distributed by the NIH-funded Human Connectome Project, which is a hard-drive version of large collections of human MRI data). Other limitations may include restrictions due to participant privacy (the **HIPAA**, or Health Insurance Portability and Accountability Act, enacted into law in the United States in 1996, restricts the information that can be made public about participants in research data; other similar laws apply elsewhere). While data could be considered factual information that cannot be copyrighted, research data often undergoes several steps of transformation before it can be useful: it is collected, aggregated, and manipulated, using significant investment of time or resources. Thus, it could represent an original and creative expression of the source (or "raw") data and may be considered copyrightable intellectual property. For data sharing to achieve its goals of reproducibility, it is therefore important to consider and define appropriate conditions of license to potential users when sharing data.

Munging

Research data is often quite “messy”. This means that it is not immediately tractable to the standard statistical analysis without additional steps (Milliken, 2006).

Data munging (also known as **data cleaning**) refers to the application of transformations to the data to bring from a “messy” state to a “tidy” state (Wickham, 2014). This may include filtering operations (exclusion of certain observations that contain missing values),

aggregation, and integration of data from different sources. According to some estimates, data munging is one of the principal activities of individuals conducting data analysis across different sectors, including research in both industry and academia (Dasu & Johnson, 2003; Lohr, 2014).

Figuratively, people speak collectively of these transformations and data “janitorial” work as data "munging". This word stems from either the English word "mung", which refers to a messy mixture of things (originally, a mixture of graines) (Oxford English Dictionary, 2016a), or the word "munge", referring to "eating greedily and noisily" (Oxford English Dictionary, 2016c) (possibly related to the word "munching"). More rarely, it also refers to wiping of a person’s nose (Oxford English Dictionary, 2016b), which could be a reference to the act of cleaning itself. Alternatively, this is derived from the acronym MUNG, meaning "mash until no good" (or recursively, "mung until no good"). To maintain reproducibility of these steps, **provenance tracking** must be used to maintain the transformations and intermediate states of the data.

Software Testing

There are several types of testing to be considered:

1. ***Unit testing***: This type of testing focuses on the operations of individual parts of the software ("units"). One rule of thumb is that unit testing should not require disk input/output, or access to the network. Unit testing works best when coupled with modular software design. In scientific software, unit testing takes the form of verification of known results from a specific function.
2. ***Integration testing***: This type of testing focuses on testing the combination of different parts of a system. For example, verifying that the outputs of one part of the system can be ingested as inputs by other parts of the system to produce reasonable results.
3. ***Regression testing***: This type of testing focuses on testing that previous results of a computation are maintained over time. This is useful to assess parts of the software for which it is hard to write unit tests. For example, parts of the software that contain random number generation can be tested to not deviate from a prior stored result by more than a certain factor.
4. ***End-to-end testing***: This type of testing verifies if the operations of an entire system, under realistic conditions, produce desired results. For example, an analysis pipeline that starts with raw experimental data (considered representative of the actual data that the system is designed to analyze) transforms and munges this data, and results in some statistical analysis. Testing an entire workflows is considered end-to-end testing (see also **continuous integration**, below).

Continuous Integration

In software development, **integration** refers to the steps taken at different stages of development to harmonize the operations of different parts of systems made up of small parts. The integration of new features into a software system can cause unexpected changes in its behavior. This is addressed by **software testing**: if the existing software has sufficient **test coverage** — that is, the tests exercise all the different parts of the software, and exercise a sufficiently broad range of scenarios: corner cases, handling of extreme and unusual values, etc. — then integration of a new piece of software would be evaluated against the expected behavior of the software. To make the process of integration easier, many advocate doing it *early and often* (Duvall, Matyas, & Glover, 2007). For integration testing to be **continuous**, automated systems can be configured to run the **test suite** of the software system (the full set of tests) each and every time a change to the software is introduced. Such publicly available systems include **Travis** and **CircleCI**. These services integrate well with websites that provide version control repositories, such as GitHub or Bitbucket, where new contributions to the software from collaborators can be set to trigger a run of the test suite on a publicly accessible server. Continuous integration on a remote server also help make sure that the dependencies of the software are well-defined, and protects against problems that arise from changing these software dependencies by triggering a test-failure whenever these dependencies change.

Workflow Management

Many scientific projects rely on the execution of several steps of data processing, including data munging and different steps of data analysis. Workflow management systems help distribute and orchestrate the work that needs to be done on the computational resources that are available, but also helps in *tracking provenance* of the results, by storing details of the data, the process, and the executions that take place during the analysis (Davidson & Freire, 2008).

File Format Standards

Scientific data is saved in a myriad of file formats. A typical file format might include a **file header**, describing the layout of the data on disk, **metadata** associated with the data, and the data itself, often stored in binary format. In some cases (e.g., **CSV (or comma-separated value) files**), data will be stored as text. The danger of proliferation of file formats in scientific data lies in the need to build and maintain separate software tools to read, write and process all these data formats. This makes interoperability between different practitioners more difficult, and limits the value of data sharing, because access to the data in the files remains limited.

Licensing

In most countries in the world, creative work is protected by copyright laws. International conventions, and primarily the Berne Convention of 1886, protect the copyright of creators even across international borders for 50 years after the death of the creator. This means that copying and using the creative work is limited by conditions set by the creator, or another copyright holder. For example, in many cases musical recordings may not be copied and further distributed without the permission of the musician, or of the production company that has acquired the copyright from the musician. Facts about the universe that are discovered through research are not subject to copyright, but the collection, aggregation, analysis and interpretation of research data may be considered creative work, and could be protected by copyright laws. Thus, the consumption of research publications is governed by copyright law. Furthermore, even data sharing is often governed by copyright laws, because the compilation of data to be shared often requires a creative effort. Another case of resarch-relevant copyrighted products is software that is developed in the course of research. In all of these cases, if license terms are not explicitly specified, the work is considered to be protected as "all rights reserved". This means that no one but the creator of the work can use the work unencumbered. For software this means that copying and further distribution of the software is prohibited. Even running the software may be restricted. The exact selection of a license is beyond the scope of this section, but depends on your intentions and goals with regard to the software (Fogel, 2005; Hunter, 2004; Rosen & Einschlag, 2004).

Virtualization and Environment Isolation

Software often requires other software to run properly. The software and hardware elements that are required to properly run a program are known as the **software dependencies**. Because of differences in hardware and operating systems, and because of conflicting dependencies between different programs, the creation and maintenance of software environments that have all the dependencies for a software system is cumbersome, and may require substantial system administration expertise. Pre-configured software environments that include all of the dependencies, software, and sometimes also the data needed for an analysis can be provided through systems that present the user a virtual machine (or VM) that runs in an isolated manner. These systems for virtualization include **VirtualBox** and **Vagrant**.

These systems rely on the ability to store an entire virtual machine as a file that can be copied, and launched within other machine’s environment. In addition, some systems provide programmatic virtualization, and dependency management, through the creation of minimal virtual machines referred to as "containers". This includes the **Docker** system, which allows not only storing and publishing light-weight virtual machines, but also provenance tracking and version control of containers. Conflicting software dependencies

can also be managed through systems that isolate a computational environment by setting the parts of the file system that are visible, including the parts of the file system into which versions of dependency libraries are installed. In Python environment, isolation can be achieved through the use of virtual environments such as **virtualenv** and **conda**.

Tools

Programming Language and Related Tools

C/C++

C is one of the most widely used programming languages of all time. Designed to be a compiled language, C was used to re-implement the Unix operating system, and many high-level languages were implemented in C, including Python. C++ is an extension of C that provides support for object-oriented capabilities, and it has become one of the most widely used object-oriented languages, especially for large scale and high performance applications.

Go

Go is a compiled programming language developed at Google, mostly used in some of the Google's production systems.

IPython

IPython, or Interactive Python (Pérez & Granger, 2007), is a command shell that allows interactive computing for Python, including tab completion, history (provenance capture), parallel computing tools, and support for interactive data visualization.

Java

Java is a programming language that is compiled into Java bytecode and run on a Java Virtual Machine (JVM), which ensures that all implementations are interoperable in different environments.

JavaScript

JavaScript is an interpreted programming language extensively used for World Wide Web content production, alongside HTML and CSS.

Jupyter

Jupyter is a Web application that allows users to create and share *notebooks*, documents that contain live and dynamic code. The Jupyter project evolved from the original IPython, generalizing the interactive environment from being Python-specific to supporting over 40 programming languages.

Python

Python is a general-purpose interpreted programming language. While Python has a comprehensive standard library, PyP (the Python Package Index) allows users to search for and download a number of additional Python packages and libraries. Many of these packages are remarkably popular and widely used in different sciences, including:

- *NumPy*: this library provides support for large, multi-dimensional arrays and matrices, as well as implements a plethora of high-level mathematical functions that operate on these arrays and matrices. NumPy also allows the definition of arbitrary data types, which facilitates the integration with other libraries and tools.
- *SciPy*: this library builds on top of NumPy to provide many high-level and efficient numerical routines mainly for numerical integration and optimization.
- *matplotlib*: this library provides 2D plotting procedures for Python.
- *scikit-learn*: this library provides support for a variety of machine learning algorithms, including classification, regression, clustering, dimensionality reduction, and model selection techniques. scikit-learn is built on top of NumPy, SciPy, and matplotlib.
- *scikit-image*: this library provides support for a collection of image processing algorithms. Similar to scikit-learn, it is built on top of NumPy, SciPy, and matplotlib.
- *pandas*: this library brings to Python many data analysis functionalities, including high-level data manipulation tasks (selecting, filtering, slicing, sorting, grouping, plotting, etc.)
- *MNE*: this library includes a Python package for processing electroencephalography and magnetoencephalography data.
- *Nipype*: this library provides a uniform interface for creating workflows that integrate a collection of neuroimaging software and applications.

R

R is a widely used interpreted programming language for statistical computing, data analysis and visualization, with its popularity largely increasing in diverse scientific fields during the past few years (Tippmann, 2014). There is a large and vibrant community of scientists using and developing software in R, with over 8000 packages contributed to the Comprehensive R

Archive Network. These packages are free to download and extend the functionality of R by adding specialized statistical algorithms, visualization techniques and file handling methods. The following R packages are worth noticing:

- *knitr*: this library provides support for dynamic report generation: R code can be evaluated on the fly to generate documents (PDF, HTML or MS Word files) that automatically include the results of the R analysis.
- *knitcitations*: this library extends knitr by allowing users to add citations to the dynamic reports.
- *dplyr*: this library includes high-level functions for data manipulation tasks that resemble database-like queries (selecting, filtering, and summarizing the data).
- *stringr*: this library provides tools for manipulating text, using regular expressions and character strings.
- *caret*: this library provides an extensive suite of tools for training regression and classification models
- *ggplot2*: this library provides data visualization procedures for R.
- *Rcpp*: this library enables R functions to call C++ code for high performance computing.
- *devtools*: this library includes functions to simplify the development of a new R package.
- *testthat*: this library includes functions to set up unit testing for the code.

RStudio

RStudio is an integrated development environment (IDE) for R that includes both desktop and web server versions. Its code editor provides syntax highlighting, tab-completion, indenting, and definitions. It includes a debugging console, breakpoints, an environment panel, history, tracebacks, and integrated R help and documentation. It supports 2d and 3d visualizations, data display, and data manipulation. Knitr, markdown, and git are deeply integrated into RStudio, enabling version controlled programming via R markdown documents.

Ruby

Ruby is an interpreted programming language commonly used in Web development, and its syntax is broadly similar to that of Python.

Scala

Scala is a programming language intended to be compiled to Java bytecode and executed on a JVM. Java and Scala are interoperable in the sense that libraries from one language can be used inside the other language.

Documentation Generators

Doxygen

Doxygen is a tool that automatically generates documentation (in different formats) from annotated source code, supporting a number of different programming languages.

Read the Docs

Rea is a hosting service for software documentation. The service facilitates the process of generating documentation for the different versions of the code, Read the Docs can be set up to automatically build the documentation whenever a new version of the code is generated.

Roxygen

Roxygen is a Doxygen-like system for R.

Sphinx

Sphinx is a tool that can generate documentation in many different file formats.

Pandoc

Pandoc is a tool that can convert between many different file formats, including LaTeX, HTML, Microsoft Word documents, and Markdown files.

Version Control

Bitbucket

Bitbucket is a repository hosting service for two distributed version control systems: git and Mercurial. Similar to GitHub, is provides a Web-based interface to facilitate the collaboration in a project.

Git

Git is a distributed version control system that has become widely used in the past few years.

GitHub

GitHub is a git repository hosting service: developers maintain their git repositories on the Web. It provides a Web-based interface, as well as a desktop application, to facilitate the collaboration with other people in the same project. GitHub has numerous features, including, among others, forking, issue tracking, pull requests, and wikis.

SVN

Subversion, or SVN, is a centralized version control system.

Data Munging and Analysis

Apache Hadoop

Hadoop is a popular framework for distributed processing of large datasets across clusters of computers. Hadoop uses the map-reduce programming model for scaling up to multiple machines. Apache HDFS is the distributed file system used to store input, intermediate, and output data.

Apache Spark

Spark is a framework for distributed processing that, in contrast to Hadoop, provides in-memory primitives that can achieve better performance for a number of applications.

Connectome Workbench

The Connectome Workbench is a tool that provides multiple resources for mapping neuroimaging data.

MATLAB

MATLAB is a numerical computing environment and also a programming language widely popular for data and statistical analysis. It provides many useful features, especially for data management, matrix manipulation, and plotting.

Microsoft Excel

Excel is a spreadsheet system developed by Microsoft that has many different features, including graphing tools and support for a macro programming language.

MongoDB

MongoDB is a database system that has been widely used recently, in particular for distributed stores. Instead of storing data in multiple relational structures—such as in traditional relational systems — MongoDB is document-oriented, it stores data in a minimal number of documents.

pandas

panda is a Python library that has many data analysis functionalities, including high-level data manipulation tasks (selecting, filtering, slicing, sorting, grouping, plotting, etc).

SEPlib

SEPlib is a distributed software package for seismic data processing, including seismic processing routines, a graphics library, and a IO subroutine library.

Stata

Stata is a commercial data analysis and statistical analysis software.

Data Visualization

Adobe Photoshop

Adobe Photoshop is a popular commercial graphics editor, providing a plethora of features to compose and manipulate graphics.

D3

D3 is a JavaScript library used for manipulating data and creating 2D interactive information and data visualizations.

ggplot2

ggplot2 is a data visualization library for R.

matplotlib

matplotlib is a popular 2D plotting library for Python.

Workflow and Provenance Management

EUPS

EUPS is a version management tools that tracks the exact project computational dependencies.

Make

GNU Make and CMake are tools commonly used to build and derive executable programs from source file. These utilities obtain the dataflow of how to build a program from files called *makefiles*.

VisTrails

VisTrails is an open-source scientific workflow system that provides support for simulations, data exploration, and visualization, while having many capabilities for provenance capture, management, and analytics.

Software Testing and Continuous Integration

BuildBot

BuildBot is a Python-based continuous integration tool that automates the process of building and testing software projects.

CircleCI

CircleCI is a hosted continuous integration service for Web and mobile applications that, similar to Travis CI, can be used to automatically build and test projects hosted at GitHub.

Coveralls

Coveralls is a tool that automatically identifies the test coverage in a project, showing which parts of the code are not covered by the test suite.

devtools

devtools is a library that contains a series of functions to facilitate package development for R.

Google Test

Google Test is a unit testing library for C++ developed and used by Google.

Jenkins

Jenkins is a Java-based continuous integration tool that automates the process of building and testing software projects.

JIRA

JIRA is a commercial software for bug tracking, issue tracking, and project management.

Nose

nos is a Python library that implements functions to assist in writing and running software tests.

nose2

nose2 — a successor to nose — is a unit testing library for Python.

testthat

testtha is a unit testing library for R.

Travis CI

Travis CI is a hosted, distributed continuous integration service that can be used to automatically build and test projects hosted at GitHub. If the service is configured, every new commit to the GitHub repository triggers Travis CI, which tries to build the project and run tests. Travis CI is available for a number of different languages.

Virtualization and Environment Isolation

Amazon EC2

Amazon EC2 is a Web service that provides compute infrastructure in the cloud. Virtual environments can be created, launched, and terminated as needed, and users pay by the hour for active servers.

Docker

Docker is a tool that automates the deployment of applications inside software containers, which are much lighter than virtual machines: containers are isolated but share the operating system, and, when appropriate, binaries and libraries as well. boot2docker is a Linux distribution made specifically to run Docker containers.

Vagrant

Vagrant is a tool used to create and configure virtual environments, such as virtual machines and Docker containers.

Virtualenv

Virtualenv is a tool that creates isolated Python environments. This allows multiple Python projects that have different (and sometimes conflicting) dependencies to coexist in the same computer.

Data Sharing and Repositories

Amazon S3

Amazon S3 is a service for online file storage on the cloud. S3 has been widely used for Web hosting, image hosting, and storage for backup systems.

arXiv

arXiv is a repository of electronic preprints of scientific publications, and is widely used in the fields of mathematics and physics.

CrossRef

CrossRef is an official Digital Object Identifier (DOI) Registration Agency. A DOI is often assigned to a publication or research data so that it can be uniquely identified, and therefore, citable. Services like Dataverse and figshare automatically generate DOI's for data that is uploaded to their systems.

Dataverse

The Dataverse Project is a repository for sharing, citing, and archiving research data. It offers support for backups, recovery, data discovery and cataloging, metadata extraction, and preservation.

Docker Hub

Docker Hub is a service for building and shipping Docker containers. Docker Hub allows integration with GitHub and BitBucket, as well as collaboration between different users, among other features.

Dropbox

Dropbox is a service that hosts files on the Web as well as synchronizes files across different platforms. Dropbox also has file versioning features, where users may revisit old versions of their files without losing any work.

figshare

figshare is a repository for sharing and citing research data (results and manuscripts).

Flickr

Flickr is a service to host and share images and videos on the Web. It is widely popular among photo researchers and bloggers.

Mendeley and Zotero

Mendeley and Zoter are both Web services and desktop applications for managing and sharing research publications.

NeuroVault

NeuroVault is a repository for sharing statistical maps, parcellations, and atlases of the human brain.

Zenodo

Zenodo is a repository for sharing and citing research results, including data and publications.

Document Authoring

LaTeX

LaTeX is a word processor and a document markup language commonly used for writing research publications. In contrast with Microsoft Word, LaTeX is not a WYSIWYG editor: the document needs to be compiled to generate the finished product.

Microsoft Word

Word is a document and word processing software developed by Microsoft. Microsoft Word is a WYSIWYG editor, while editing, the content onscreen appears in a form that is similar to its appearance as a finished product (WYSIWYG stands for “What You See Is What You Get”).

Overleaf

Overleaf is an online platform for collaborative writing and publishing using LaTeX, with an integrated real-time preview that closely resembles a WYSIWYG editor.

File Formats

API

An API (or application programming interface) are elements of the design of a software system that that allows programmers to use the system to build applications out of it. For example, a software library API will be the design of functions and objects in the library that can be combined together to create new functions and objects.

CSV

The CSV (“Comma Separated Values”) file format stores data in a tabular fashion in plain text. This format is often used to transfer data between applications.

DO

A DO file is a Web-base Java program that is run by a Web server.

Dockerfile

A Dockerfile is a file that has a set of instructions and commands for building a Docker container.

FIF

A FIF (“Fractal Image File”) file stores images in fractals, which can be resized without losing image quality.

HDF5

The HDF5 file format is designed to store and organize large amounts of data. Different data models can be specified for storing data, including multidimensional arrays and tables.

ipynb

An ipynb file represents an IPython notebook document.

JSON

JSON is a data-interchange format that is both human- and machine-readable, storing and transmitting data as attribute-value pairs. It has been widely used recently, largely replacing XML.

Markdown

A Markdown file contains data in a simple markup language that facilitates the conversion from plain text to HTML and other formats. Common extensions for this file include *md* and *Rmd* (the latter represents an R Markdown file where R code is included among the text).

netCDF

The netCDF ("network common data format") file format is machine-independent format commonly used for sharing array-oriented scientific data.

PDF

The PDF (“Portable Document Format”) file format is commonly used to display documents in an interoperable manner.

RAID

RAID (redundant array of independent disks) is a system that confers robustness to data storage through redundancy across sub-partitions. Every bit of data is stored in at least two different partitions, such that if any given partition fails, it can be swapped out without incurring data loss.

SQL

A SQL (“Structured Query Language”) file contains a series of database queries to analyze and manage tables in a database. These queries are represented by statements written in SQL, a programming language designed for managing data in relational databases systems.

SVG

The SVG (“Scalable Vector Graphics”) file represents graphics using an XML-based format that offers support for interactivity and animation.

VT

A VT file stores a VisTrails workflow and its corresponding provenance.

XML

An XML (“Extensible Markup Language”) file stores data in XML, which is a markup language that encodes documents in a format that is both human- and machine-readable. XML is known to provide interoperability among different applications.

References

Chirigati, F., Shasha, D., & Freire, J. (2013). ReproZip: Using Provenance to Support Computational Reproducibility. In *Proceedings of the 5th uSENIX workshop on the theory and practice of provenance* (pp. 1:1–1:4).

Dasu, T., & Johnson, T. (2003). *Exploratory data mining and data cleaning*. John Wiley & Sons.

Davidson, S. B., & Freire, J. (2008). Provenance and Scientific Workflows: Challenges and Opportunities. In *Proceedings of the 2008 aCM sIGMOD international conference on management of data* (pp. 1345–1350).

Duvall, P. M., Matyas, S., & Glover, A. (2007). *Continuous integration: Improving software quality and reducing risk*. Pearson Education.

Fogel, K. (2005). *Producing open source software: How to run a successful free software project*. O’Reilly Media, Inc.

Freire, J., & Silva, C. T. (2012). Making Computations and Publications Reproducible with VisTrails. *Computing in Science Engineering*, *14*(4), 18–25.

Freire, J., Bonnet, P., & Shasha, D. (2012). Computational Reproducibility: State-of-the-art, Challenges, and Database Research Opportunities. In *Proceedings of the 2012 aCM sIGMOD international conference on management of data* (pp. 593–596).

Freire, J., Koop, D., Santos, E., & Silva, C. T. (2008). Provenance for Computational Tasks: A Survey. *Computing in Science and Engineering*, *10*(3), 11–21.

Hunold, S., & Träff, J. L. (2013). On the State and Importance of Reproducible Experimental Research in Parallel Computing. *CoRR*.

Hunter, J. (2004). Why we should be using BSD. Accessed: 2015-10-25. Retrieved from http://nipy.org/nipy/faq/johns_bsd_pitch.html

Ioannidis, J. P. A. (2005). Why Most Published Research Findings Are False. *PLoS Med*, *2*(8), e124. http://doi.org/10.1371/journal.pmed.0020124

Knuth, D. E. (1984). Literate programming. *The Computer Journal*, *27*(2), 97–111.

Leek, J. T., & Peng, R. D. (2015). Opinion: Reproducible research can still be wrong: Adopting a prevention approach. *Proceedings of the National Academy of Sciences*, *112*(6), 1645–1646. http://doi.org/10.1073/pnas.1421412111

Lohr, S. (2014, August 17). For big-data scientists, 'janitor work' is key hurdle to insights. *New York Times*. Retrieved from http://www.nytimes.com/2014/08/18/technology/for-big-data-scientists-hurdle-to-insights-is-janitor-work.html

Milliken, G. (2006). Messy data. In S. Kotz (Ed.), *Encyclopedia of statistical science*. Hoboken, NJ: Wiley.

Nuzzo, R. (2015). How scientists fool themselves – and how they can stop. *Nature*, *526*(7572), 182–185.

Oxford English Dictionary. (2016a). Mung, n.1 and adj. Oxford University Press. Retrieved from http://www.oed.com/view/Entry/113400

Oxford English Dictionary. (2016b). Munge, v.1. Oxford University Press. Retrieved from http://www.oed.com/view/Entry/123777?rskey=KZFDs3&result=1

Oxford English Dictionary. (2016c). Munge, v.2. Oxford University Press. Retrieved from http://www.oed.com/view/Entry/252110?rskey=KZFDs3&result=2

Pérez, F., & Granger, B. E. (2007). IPython: A system for interactive scientific computing. *Computing in Science and Engineering*, *9*(3), 21–29. http://doi.org/10.1109/MCSE.2007.53

Poldrack, R. A., & Gorgolewski, K. J. (2014). Making big data open: Data sharing in neuroimaging. *Nat. Neurosci.*, *17*(11), 1510–1517.

Rosen, L., & Einschlag, M. (2004). *Open source licensing*. Prentice Hill.

Stodden, V. (2014). What Scientific Idea is Ready for Retirement? Reproducibility. *Edge.org*. Retrieved from http://edge.org/response-detail/25340

Stodden, V., Bailey, D. H., Borwein, J., LeVeque, R. J., Rider, B., & Stein, W. (2013). Setting the Default to Reproducible: Reproducibility in Computational and Experimental Mathematics. *ICERM Workshop Report*. Retrieved from http://stodden.net/icerm_report.pdf

Stodden, V., Leisch, F., & Peng, R. D. (2014). *Implementing reproducible research*. CRC Press.

Tippmann, S. (2014). Programming tools: Adventures with R. *Nature*, *517*(7532), 109–110. http://doi.org/10.1038/517109a

Wickham, H. (2014). Tidy data. *J. Stat. Softw.*, *59*(10).

Wulf, W. A. (1977). Some thoughts on the next generation of programming languages. *Perspectives on Computer Science*, 217–234.

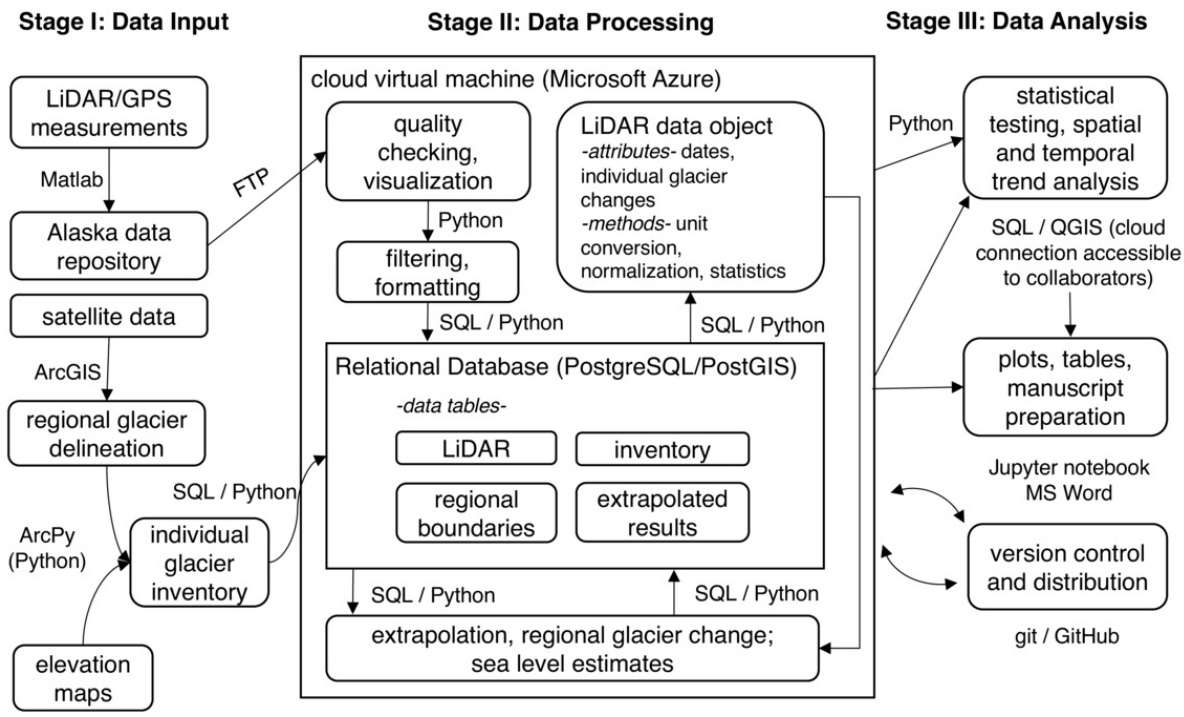
Processing of Airborne Laser Altimetry Data Using Cloud-based Python and Relational Database Tools

Anthony Arendt, Christian Kienholz, Christopher Larsen, Justin Rich and Evan Burgess

My name is Anthony Arendt and I hold a joint appointment as a Senior Research Scientist at the Applied Physics Laboratory, and a Research Fellow at the eScience Institute, University of Washington. I am part of a research team that studies the impact of glaciers on rising global sea levels, with a focus on the glaciers of Alaska and northwestern Canada. During the past 20 years my colleagues at the University of Alaska Fairbanks have been measuring the elevation changes of Alaska's glaciers using Light Detection and Ranging (LiDAR) data collected from a small aircraft. Our LiDAR system consists of a laser range finder and Global Positioning System (GPS) that measures the precise elevation along the centerline of the glacier surface. By repeating these observations through time, we estimate total changes in mass of each observed glacier, and then extrapolate these data to unmeasured glaciers based on information acquired from satellite imagery. From this we produce detailed maps of the spatial distribution of glacier mass change and the total contribution of these ice masses to global ocean change.

During the 20 year duration of the project the data analysis has evolved from manual manipulation of text files, to a semi-automated workflow that integrates Geographic Information System (GIS), relational database and Python tools within a cloud computing framework. Here we describe the workflow which culminated in a recent publication (Larsen et al., 2015). Core developers of the software include Evan Burgess, Christian Kienholz, Justin Rich, Anthony Arendt and Christopher Larsen.

Workflow



The workflow begins with annual field data collection that produces both GPS positional and LiDAR point cloud data, both in industry standard binary formats. Commercial proprietary software is used to process the data into four dimensional point observations (x, y, z and time), which are then further processed using Generic Mapping Tools (GMT) into gridded 10 m digital elevation models. These elevation maps are then subtracted from maps acquired at an earlier time to obtain a change in elevation along the flight line, using Matlab scripts. These results are stored in text files, with the file name describing the glacier name and start and end dates, and are located on a server at the University of Alaska Fairbanks. In a separate step, we assemble satellite imagery and regional digital elevation models for the Alaska region. We use ArcGIS, a commercial GIS software package, to manually digitize the glacier extent. ArcGIS provides a set of vector manipulation tools that enable our technicians to trace glacier outlines from satellite imagery. ArcGIS commands can also be scripted using the ArcPy library. We automate a series of GIS commands using ArcPy to calculate the distribution of glacier surface area with elevation for each of approximately 27,000 glaciers in Alaska.

The majority of our data processing and analysis occurs on a single Microsoft Azure Linux Virtual Machine (VM) that hosts a spatially enabled Relational Database (RDB). We find that hosting an RDB in the cloud is a core element of our reproducible workflow. Our RDB provides rapid query capabilities so that much of our spatial and temporal averaging can be

carried out using efficient database algorithms. Our cloud hosting enables colleagues to make direct connections to the database to access spatial data using their local GIS software. We use the open source PostgreSQL database with the PostGIS extension, to which we ingest point, line and polygon geospatial datasets. Relevant tables include:

- *inventory*: polygons of each glacier in Alaska with attributes of surface area, glacier type (whether terminating on land or ocean), name
- *regional boundaries*: polygons of outlines of mountain ranges or climatic zones over which we perform regional extrapolations
- *LiDAR*: measurements of elevation and volume changes on surveyed glaciers
- *extrapolated results*: final estimates of the volume change of every glacier in Alaska

Each time we acquire new altimetry observations we run a Python script to connect via secure FTP to the server in Fairbanks and search for new text files across the directory structure. We use the Python Pandas library as an interface between our text file and RDB data objects. Specifically, once we ingest the data into a Pandas DataFrame, we can employ a series of methods to generate simple plots and export the data directly to our PostgreSQL database. We use similar Python tools to create and update the *inventory* and *regional boundaries* tables, for example to accommodate changes in surface area as glaciers retreat.

The *LiDAR data object* is the foundation of all subsequent processing and analysis. The data object is created via a function call with parameters describing a single or a regional grouping of glaciers. Each instance of the data object has predefined attributes enabling users to rapidly acquire elements of the raw data in the *LiDAR* table. For example, a user can issue a request to the *LiDAR* table for a specific glacier, returning a data object whose attributes contain that glacier's elevation change, area, and other statistical information. The data object also has several methods that handle the majority of the standard data processing and filtering workflow. These methods include algorithms that carry out unit conversions, normalize the data, calculate statistics and perform mass change calculations for each glacier. To perform these calculations we issue Structured Query Language (SQL) commands to the database from within our Python scripts.

In a final processing step, we utilize the grouping functionality of the LiDAR object methods to generate average elevation change profiles across glaciers grouped by type or by spatial location. For example, we found similar elevation change distributions across glaciers with similar terminus types (i.e. whether terminating in land or in water). Therefore we generated LiDAR objects averaged over *type* groupings, as queried from our *inventory* table, thereby returning a single estimate of elevation change versus elevation. In a final step, we invoked a function that regionally extrapolates these profiles to the unmeasured ice masses stored in the *inventory* table, based on their distribution of area with elevation. This returns a dictionary of ice mass changes by group, as well as an optional new database table

containing mass change estimates for each of the 27,000 ice polygons in the region (table *extrapolated results*). All functions and methods run quickly, with the exception of steps involved in building *extrapolated results*, which takes about 15 minutes to run.

To analyze results and distribute our findings we host a permanent instance of a Jupyter notebook on our Linux VM and provide access to project team members. The notebooks, as well as the core Python scripts used to generate results, are also located in a GitHub repository. The notebook also contains markdown to provide metadata at each step in the analysis. Collaborators with experience writing SQL code can have direct access to the PostgreSQL database to perform their own queries. Other collaborators more familiar with GIS tools can connecting directly to the geospatially encoded tables to generate their own maps.

Pain points

Our team brought together researchers with different backgrounds and approaches to data handling and processing. The processing of raw LiDAR and GPS data is performed by a different group than the one handling the GIS and extrapolation portion of the project, and each uses different software tools. We dealt with this problem by creating standardized files at different stages of the processing chain. For example, the LiDAR/GPS team produced a stack of files processed to the point where they could be used for extrapolation, which were then ingested to the geospatial database. A challenge here is the data are replicated in multiple locations, requiring careful version control.

Another challenge is that some of our collaborators encountered problems when attempting to connect directly to our cloud computing resources. One issue is that Alaska has limited internet bandwidth so that transfer of data between Alaska and commercial cloud providers is slow. Another challenge is that many US government agencies have firewalls that restrict direct traffic with our cloud database services. Therefore our collaborators in government and/or those located in Alaska had to set up duplicate versions of our databases, creating challenges with version control and project management.

Key benefits

Our workflow provides a mechanism to continually update our analysis as new data arrive. Our project is funded for several more years, and we are now in a position to regenerate key figures and update sea level estimates every time we acquire new datasets. This will greatly diminish the time it takes for us to provide stakeholders with updated information on the status of Alaska glaciers and their contribution to sea level. Also, our data are uniquely dynamic, and must accommodate not only new data but changes to the base inventory as

glacier geometries (area and elevation) change over time. By having all our inventory data in relational tables we can update individual polygons and account for the feedback effects of glacier area on mass balance.

Our workflow also provides a stable foundation that can accommodate changes in team composition over time. As students and technicians join and leave the project we can have them use and contribute to a repository of scripts, rather than having to reinvent things from scratch.

We are well positioned to explore our data in ways not previously possible. New collaborators are joining our team and making direct connections to our database, generating complex queries that are exploring what climatic and geometric factors may be driving the glacier mass changes we are observing in the field. Other similar LiDAR observation programs do not provide access to relational databases, limiting researchers' ability to perform spatial and temporal queries.

Key tools

Hosting our resources in a cloud environment played a vital role in making our workflow reproducible. The cloud enabled us to co-locate our scripts with the observations, enabling rapid processing and minimizing the need to transfer files. Additionally, using a relational database to store our geospatial datasets provided efficient methods for us to explore a wide range of spatial relationships in our datasets.

Questions

What does "reproducibility" mean to you?

Reproducibility is a crucial component of our workflow due to the dynamic nature of our monitoring campaign, and the need to constantly update the position and elevation of glaciers as they change in response to climate. We achieve reproducibility through:

- Maintaining consistency in the input datasets
- Utilizing a series of scripts to automate data ingest and filtering
- Storing raw and filtered/processed data in a relational database
- Generating data objects that handle typical data analysis functions
- Scripting all manuscript figures in Jupyter notebooks

Why do you think that reproducibility in your domain is important?

Glaciology has become highly interdisciplinary in the past decade: oceanographers, climatologists, geodesists and glaciologists must integrate knowledge to close the sea level budget. Also, data from remote glacier regions is sparse, so any data we collect needs to be made available. By generating reproducible workflows we have a greater capacity to share information and to better understand exactly how each research team is processing their datasets.

How or where did you learn about reproducibility?

I learned these techniques through coursework, a visiting scientist appointment at Microsoft Research, and through self-directed learning.

What do you see as the major challenges to doing reproducible research in your domain, and do you have any suggestions?

The inability of non-specialists to make full use of our tools occasionally requires us to revert back to non-reproducible methods in order to get things done in a timely fashion. We are working to solve this problem by building lightweight Application Programming Interfaces enabling collaborators to access some of the core elements of our workflow through simple web protocols.

What do you view as the major incentives for doing reproducible research?

Within a research team, major incentives include: increased transparency in methods, increased accountability and ability to check for errors in processing, a reduction in spin-up time as new members join the team, and an ability to minimize duplication of effort. Between the team and other collaborators/stakeholders, we see major benefits in the ability to share and visualize results, and in our capacity to perform cross-disciplinary research.

Are there any best practices that you'd recommend for researchers in your field?

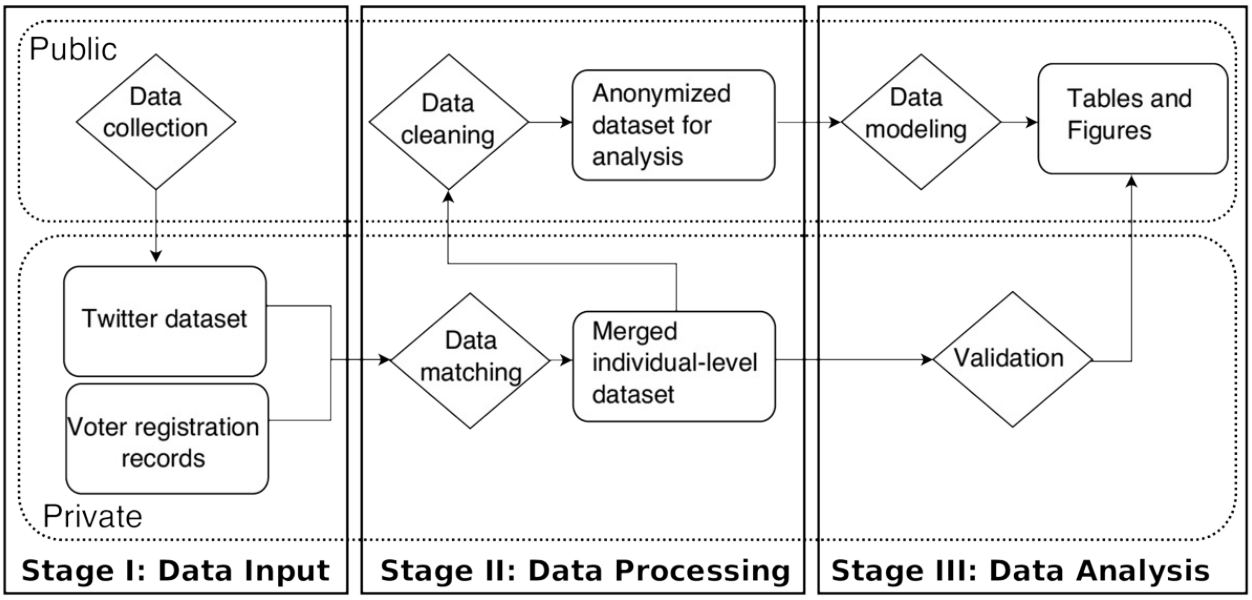
We recommend development and adherence to standards in geospatial data formats and distribution protocols.

The Trade-Off Between Reproducibility and Privacy in the Use of Social Media Data to Study Political Behavior

Pablo Barberá

My name is Pablo Barberá, and I am a political scientist who applies computational methods to the study of political and social behavior. I joined the School of International Relations at the University of Southern California as an Assistant Professor in 2016, after spending a year as a Moore-Sloan Fellow at the Center for Data Science in New York University. The workflow I describe here corresponds to part of my dissertation research, whose aim is to study political polarization on social media websites. In particular, here I focus on the research process that led to an article published in the journal Political Analysis in 2015, which presents a new method to estimate the political ideology of Twitter users based on the structure of their personal networks.

Workflow



An important concern in the design of projects involving social media data should be to guarantee that the private records of the individuals in the sample are protected, while ensuring that every step is reproducible. In the analysis described here, I only rely on publicly available information -- in particular, information about Twitter profiles and voting registration records in the state of Ohio, which is used for validation purposes. However, the goal of the project is to infer a sensitive latent trait about each Twitter users -- their

ideological position, on a scale from very liberal to very conservative. This is information that most users are not aware could be inferred based only on their public personal information, which raises the question of whether the concept of informed consent in sharing users' data -- as defined in the Terms of Service that users accept when they sign up for a Twitter account -- applies in this context as well.

To achieve the goal of reproducibility while adequately protecting users' data, all the analysis in the study takes place at two levels, private and public, as shown in the diagram in this chapter. The private level includes all the original data, which will be processed and merged, and then anonymized so that it can be included in the replication materials. These datasets will not be released, but they can be acquired by other researchers from their original sources. The second level contains all the R code and output (tables and figures), as well as the anonymized version of the dataset, which allows partial replication of the results in the paper even without access to the full dataset. After finalizing the project, the materials in this level were released in public repositories on GitHub and Dataverse.

As shown in the workflow diagram, the first step in the project was to collect a dataset that would allow me to reconstruct the networks of a sample of Twitter users. In particular, I compiled from Twitter's API the lists of followers of a set of around 500 political accounts in the U.S., which includes legislators, candidates, media outlets, etc. Then, I identified the list of users who follow at least three of these political accounts -- this will be the sample in the study. Finally, for each of these users, I also extracted their profile information, including their approximate location, which was parsed into geographic coordinates using the Data Science Toolkit. This data collection step was conducted using R tools developed by the Social Media and Political Participation Lab. All the R code used in this step was made public, but the complete Twitter dataset was stored privately in order to comply with Twitter's terms of service regarding data sharing.

The second step in the workflow involves two types of data processing tasks. First, the user-level information from Twitter was matched with publicly available voting registration records from the state of Ohio, which includes information about the party that each voter is registered with. A Twitter user was matched with a voter whenever there was a perfect and unique match of first name, last name, and county between these two datasets. This information will be used in the validation step in order to assess whether the ideology estimates that result from this method are correlated with offline measures of behavior, such as the number of times that a given voter has participated in a party primary election. The second part of the data processing task involves cleaning the Twitter dataset and building the networks that will be scaled in order to obtain estimates of their ideological positions. In particular, here I constructed an adjacency matrix that indicates whether each of the users in the sample follows each of the political accounts.

After these two steps are completed, I generated an anonymized version of both datasets. The anonymization was achieved by replacing Twitter and voter unique IDs by randomly generated numeric IDs. This will allow researchers to replicate every step of the analysis after this point, but without being able to identify the individuals in the sample.

In the data modeling step, the adjacency matrix that represents this network was scaled using the STAN software for Bayesian modeling using R. The model that was implemented was similar in nature to other latent space models applied to social networks. It builds upon the assumption that the existence of a following link between users and political accounts is inversely related to their distance on a latent ideological dimension. In other words, the intuition of the model is that users tend to follow political accounts that they perceive to be close to their own ideological position. This method returns estimates of the latent positions of both users and political accounts. The output of the model was carefully validated using a variety of offline measures of ideology for both types of actors, including roll-call votes in Congress, aggregate measures of ideology from surveys, and individual-level voting records. One of the strongest results of the paper is that individuals predicted to have the most extreme positions are those that most frequently vote in primary elections -- a clear indication that strength of ideological identities is correlated with strength of partisanship.

After conducting the analysis and validation, the last part of the project consisted of producing a series of tables and figures that summarize the dataset, describe the main results of the paper, and offer a graphical representation of the validation process. All figures and tables were generated using R. Throughout this process, I documented exactly what datasets were required to generate each figure, being careful to ensure that only the code and data available in the public level of the project were required in order to replicate them. These tables and figures were then integrated into the manuscript, written using LaTeX.

Pain points

The replication data and replication code were released in different platforms; the code in GitHub, the data in Dataverse. GitHub provides the ability to track changes in the code, and makes it easy to collaborate. Their online interface is easy to use, which reduces the entry costs for other researchers interested in forking the replication materials for their own projects. However, at the moment GitHub does not allow pushing files over 100 MB. Storing smaller files within this limit is not recommended either, since every change to this file is also stored in the repository. Dataverse, on the other hand, provides a free platform to store large files, with some built-in analysis tools, as well as some basic versioning system. However, it lacks the social layer of GitHub, the ability to collaborate, and a good interface to see differences between files using version control. As a result, at the moment there doesn't exist a single platform that combines the advantages of these two.

A problem that is more specific to the workflow described here is the difficulties in ensuring that the anonymization of private records is complete. As described above, replacing the original Twitter user and voter IDs with randomly generated numbers is an approach that in theory ensures anonymity. In practice, however, it might be possible to discover the identity of some of these individuals using only some of the other variables. For example, if there are unique patterns of following behavior in the dataset (e.g. only one individual follows all the political accounts in the dataset except for Barack Obama), another researcher could succeed at discovering her identity. These are edge cases, which may not occur in the dataset here, but it is a concern if the goal is to guarantee the privacy of all individuals in the sample. Recent developments in the field of cryptography, such as differential privacy, provide promising new methods to improve these research practices.

Key benefits

Most published studies that use social media datasets to study human behavior do not provide replication datasets. This is unfortunate because it represents an important obstacle towards ensuring reproducible scientific practices and limits the use of these materials for learning purposes, but it is also understandable, as the restrictive policies of social media companies imply that researchers need to devote a significant amount of time towards ensuring compliance with these policies. My hope is that the workflow described here can become a blueprint for future replication datasets in this field.

Questions

What does "reproducibility" mean to you?

A study is reproducible when a researcher external to that particular project, but familiar with the literature and methods, is able to obtain identical results by using the same datasets and following the same procedures as those described as in the research output, be it a published article or book. Researchers should also produce replication code and a lab book with more precise details about the analysis conducted as part of the study. However, this should be in addition to the description of the research process in the publication, since the output of running code may depend on software versions, for example. There is also the possibility that a set of results is not "correct," and simply the product of errors in the code or software bugs. In other words, being able to run a piece of code and obtain identical results as those described in a published output is not a necessary nor a sufficient condition for reproducibility.

When applied to studies that rely on social media data, the concept of reproducibility is slightly more nuanced. The Terms of Service of social networking platforms like Twitter or Facebook restrict the distribution of datasets obtained through their Application

Programming Interfaces (APIs) for privacy reasons. These companies have taken steps towards enforcing this requirement, including contacting researchers to request they take down replication datasets, even if they were used for research purposes only. The trade-off between ensuring individual privacy and allowing reproducibility is even more evident when social media datasets are combined with survey data or other individual records, as in the case I describe here. In these instances, reproducing a published study implies the additional steps of querying the API to reconstruct the original dataset and matching it with the individual records, which is inefficient and not always possible. The workflow I describe here represents my best attempt towards addressing these challenges and ensuring that other researchers can reproduce my results.

A Reproducible R Notebook Using Docker

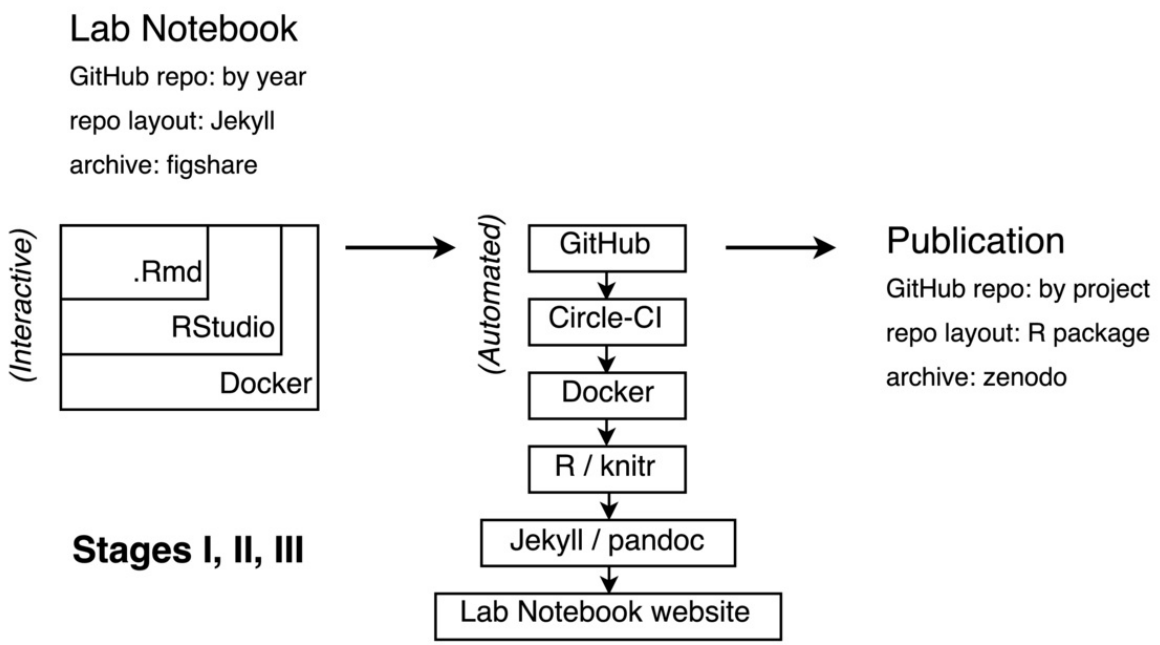
Carl Boettiger

My name is Carl Boettiger. I'm a theoretical ecologist in UC Berkeley ESPM working on problems of forecasting and decision-making in ecological systems. My work involves developing new computational and frequently data-intensive approaches to these problems.

My workflow seeks to provide a way to capture & reproduce the day-to-day to workings of a computational ecologist using freely available platforms (e.g. GitHub, Travis CI, Docker Hub) and open source software (`R` , RStudio, `git` , `docker` , `jekyll`) in the format of an online, open lab notebook. I have tweaked and adapted this workflow over the past 5 years, often experimenting with new technology. Other researchers have frequently told me how they have adopted parts of this approach, but rarely in an identical way.

My general approach to an open lab notebook has been described previously (Gewin, 2013, Mascarelli (2014)), while I focus on documenting more details of the workflow here. When possible, I have sought to leverage general-purpose tools rather than custom solutions: for instance, I organize project directories using the R package format, as described in Gentleman & Temple Lang (2007) and rrrpkg project, rather than introduce my own custom structure. Nevertheless, my current system no doubt remains too complex, specialized, esoteric and even fragile to be easily adopted by others. Rather, I encourage the reader to focus on specific elements or modules that look most practical, as others have done.

Workflow



A note to the reader: The following description is meant as a high level overview, which leans heavily on several powerful and well-developed tools and workflows including git/GitHub, docker/DockerHub, `.Rmd` /RStudio, and others. Table 1 provides a concise reference where a reader can learn more about these tools and their use.

Table 1: Tools used in workflow

Tool	Description / purpose	Website
git	Version control software	https://git-scm.org
GitHub	Online repository for sharing code managed with git	https://github.com
docker	Containerization software for portable computational environments	https://docker.com
DockerHub	Central hub for building and distributing docker containers	https://hub.docker.com
RStudio	IDE for editing R and Rmd files	https://rstudio.com
Rmd	Dynamic documentation format for R language	http://rmarkdown.rstudio.com
pandoc	convert between document formats	http://pandoc.org
servr	An R package for combining jekyll with Rmd	http://yihui.name/knitr-jekyll/
jekyll	Static website generator closely integrated with GitHub	https://jekyllrb.com
Circle CI	Flexible continuous integration software for executing scripts pushed to GitHub	https://circleci.com
figshare	Permanent data archiving platform	http://figshare.com
zenodo	Permanent archiving platform (handles code/software well)	http://zenodo.org

Interactive workflow

My daily workflow on an active project simply involves opening a new `.Rmd` document with the day's date in my lab notebook. In this file, I write the code, text, equations, and other elements of my work (see diagram, top left).

At the heart of my workflow is the dynamic documentation tool `knitr`. `knitr` is an R package that is tightly integrated into RStudio and R-markdown, or `.Rmd` format it supports for integrating code, documentation, equations, figures and other components of research into a single document. Its key feature is the ability to "knit" or "execute" the document, resulting in the code blocks being run and their output figures, tables, and so forth being displayed in the document. Text and code are written together in the popular, simple, and flexible markdown format, which is widely recognized by other tools (e.g. GitHub, a widely used code repository, and Jekyll, a ruby-based static website generator). Markdown is easily converted into other formats by `pandoc`, a conversion engine integrated into RStudio (and other popular platforms such as Jupyter) which can generate LaTeX, HTML, Microsoft Word and other document formats. This flexibility is useful later in turning my .Rmd files into either HTML pages for my laboratory notebook or into other formats suitable for traditional journal publication.

During active research, I often find it impractical to clearly separate out the stages of Data Input (Stage I), Data Processing (Stage II), and Data Analysis (Stage III). I merely strive to have all of these stages coded and explained in the .Rmd document.

I write / edit this `.Rmd` file inside an instance of RStudio which runs inside a Docker container, which in turn may be running on my laptop, an Amazon Web server, or even an NSF super-computing cluster depending my needs that day. RStudio is a popular integrated development environment for R users which can be run in server mode through a web browser. Docker is a popular containerization tool which allows one to create a portable image of one's entire software environment that can be easily moved around between different computers, regardless of architecture. I believe this has major implications for addressing common problems in reproducibility, as I have described more fully elsewhere (Boettiger, 2015). A Dockerfile in my notebook provides an executable recipe for building this computational environment on top of existing, general-purpose Docker images maintained by the Rocker project.

Automated workflow

At regular intervals I "commit" my notebook in `git` and "push" this progress to GitHub, a widely used version control system and public repository for code and other digital material. This triggers the automated build portion of my workflow, illustrated in the center of the diagram. A Continuous Integration platform (CircleCI in my case, as the more widely used platform, Travis, did not support Docker execution until much more recently) detects this commit, and begins to execute and assemble my code.

The CI platform begins by pulling down a public image of my computational environment, itself built automatically by Docker Hub from the Dockerfile in my repository. A separate Docker volume container can also be pulled from the Hub which contains results cached by

knitr for any code too intensive to run on the (free, public) CI platform.

As the notebook is already organized as a Jekyll repository, just with `.Rmd`-formatted posts instead of plain `markdown`, existing tools (see `servr`, Table 1) can easily execute the R code and format it as a new post in the notebook. Jekyll templates make it easy to add semantic metadata to the post automatically including bibliographic information, links to version history, commit hash, modification date and so forth. At this time a given exploration might not have a particular project connected to it -- it might build from several existing projects, a paper I'm reading, or represent an entirely new exploration. I use categories and tags in the notebook to associate the post with relevant projects or themes, which makes it easier to come back to. (Figuring out appropriate tags is harder than it sounds!)

Each year I archive the GitHub repository that contains that year's notebook on figshare, adding the DOI badge to the repository's README.

Project finalization / publication

Eventually multiple entries will relate to the same project. At this point, I frequently want to reuse code first developed in a previous entry. This is my signal that it is time to create a new project on GitHub. (Figuring this out is much harder than it sounds!) I create a new public GitHub repo using a name that matches a tag in the relevant notebook entries. In the `R/` directory I store functions that provide these reusable bits. For non-trivial functions, I try and develop unit tests (in the `/tests` directory) -- these usually come directly from the interactive tests I write in the notebook when first creating these functions. I also add minimal Roxygen documentation to the functions I create, usually just to remind me what the input and outputs are. Data goes in the `/data` directory; or more frequently, as R scripts that either simulate or download and clean the data from external sources.

Notebook pages do not load these functions as a single package -- as the package is constantly changing this is unlikely to continue to work anyway. Instead, they source in the script directly from the version-explicit links on GitHub. (I learned this the hard way). This avoids the burden of making sure the 'package' is always installable, it just serves as a convenient organizational skeleton.

I continue to develop, test, and explore results in the pages of the notebook, adding and modifying functions as necessary. This usually involves plenty of mistakes and dead ends that are captured in the history of an individual page (when I modify an existing workflow to correct the results) or are left as dead (or incompletely explored) ends in the various pages of the notebook under that category.

Once the work has coalesced around a particular set of ideas and results appropriate for a single manuscript, I begin drafting the manuscript as a `.Rmd` file in the GitHub repository, often based on `.Rmd` files from the notebook. The `rticles` package from RStudio provides

a template system which makes it easy to render `.Rmd` files into `pdf` articles for various journal formats.

When preparing for submission, I upload a copy of the manuscript (in `tex` format, generated from the `Rmd`) to the arXiv and configure the Zenodo permanent archive which connects automatically to GitHub, much like a Continuous Integration service. Zenodo then generates a permanent archive with a unique Digital Object Identifier (DOI) every time it detects a new 'release' on GitHub. GitHub releases are part of the `git` tag system and can be used to signal a new version of software or publication of a paper. A DOI badge from Zenodo is then displayed on the GitHub repository.

The reader is encouraged to view any of the real-world examples of this process in the repositories of my recent projects, such as https://github.com/cboettig/nonparametric-bayes, or in the pages of my online lab notebook at http://carlboettiger.info/lab-notebook.

Additional questions

- **Frequency:** How often does the step happen and how long does it take?

Frequency is highly variable -- from many commits a day to gaps of months. See my GitHub commit history for a more realistic answer.

- **Who:** Which members of your team participate (or not)?

Though most of my research projects involve others, I am the only researcher committing to my lab notebook, just as we see in paper notebooks. The final research product will see more direct involvement by others.

- **Tools:** Which software or online tools are used in the step? How are they used?

See Table 1.

Pain points

Knowing when to refactor and how to avoid fragile and opaque design. A good reproducible workflow should be like good software: built from simple, easy-to-understand modules that do one task well. Most reproducible workflows, mine included, can too readily resemble most scientific spaghetti code: pieces tacked together over the years because they got the job done. The best way to make a workflow or code understandable is to *refactor* it after it works, breaking it into well-defined, well-tested modules with clear input and output. Pretending research can be written like this from the start is fiction, but just capturing all the messiness provides none of the abstraction that makes something more re-usable and reliable. I don't have a good solution for how to do this though -- refactoring is demanding and offers few incentives.

Key benefits

A key benefit of this approach is making my work portable and scalable. By making it easy to reproduce my computational environment and analyses, it suddenly becomes much easier to re-run an analysis on a cloud machine or cluster if it proves too large for my local system.

A second benefit has been the ability to explore research ideas more easily. New ideas often build on old ones, and the dread of having to remember how some old stuff worked in the first place before tinkering with it to explore something new was often enough to make me turn to something easier.

Key tools

I believe any of the tools mentioned in Table 1 could be of use to a broader audience. I have tried to place the more general near the top -- GitHub and Docker address very general issues in computational reproducibility, justifying their wide adoption. These tools can inserted into many common workflow patterns without requiring significant re-tooling.

For R users, RStudio has made the Rmd format far more practical as an authoring environment, both for websites (e.g. with `servr` package) and journal articles (`rticles` package). However, these tools may require both a bigger shift from existing strategies and offer a smaller benefit.

The particular pattern I have used to chain this together with CI, etc, is probably less generally applicable, and has a higher learning curve than the afore-mentioned tools.

Questions

What does "reproducibility" mean to you?

Reproducibility in this context is 'computational reproducibility.' It means a good-faith effort to make sure that the analysis can produce qualitatively identical results while running on comparable hardware. This means certain things do not need to be reproduced: e.g. how long the code takes to run may vary by hardware and operating system, but this is okay. Nor am I not concerned with bitwise identical results, nor with necessarily reproducing stochastic random draws -- rather, I expect conclusions from reproducible results to be robust to the details of stochastic seed or choice of random number generator.

I am also concerned that reproducibility is modular -- that individual components of the analysis can be reproduced (and thus recombined or otherwise modified), and not merely provide a black box that can only replicate final outputs without variation or adjustment.

Lastly, I think it is important to identify *who* should be able to reproduce the analysis. Like the paper itself, the analysis requires a certain degree of expertise to understand, and I do not expect that individuals with no familiarity with programming, statistics, or scientific process can reproduce the analysis. However, I do expect that researchers with some scientific background in my area (e.g. the broadest readership of the journal in which it is published) and with minimal familiarity with the R language or similar computing langauage can reproduce the overall results after suitable investment of time and effort in reading the documentation.

Why do you think that reproducibility in your domain is important?

Reproducibility makes results more reliable, and more importantly, makes it easier to extend, test, and build upon existing results. Ultimately this makes it easier for an individual to build on their own work and the work of others, making for faster, better science.

How or where did you learn about reproducibility?

Independent study of examples, experimentation, and reading, and connecting with other researchers sharing similar interests through the internet and social media.

What do you see as the major challenges to doing reproducible research in your domain, and do you have any suggestions?

Not a standard practice. In the short-term it takes more time. It may also increase the probability of errors in your work being discovered.

What do you view as the major incentives for doing reproducible research?

Making research easier to do. Reproducible research facilitates collaboration, particularly with myself. It improves my confidence in my own results and helps me build more efficiently on work that I have already done.

Are there any best practices that you'd recommend for researchers in your field?

Adopting tools that are widely used within my field (and others) for reproducibility. These include: GitHub, Docker, rmarkdown.

Would you recommend any specific resources for learning more about reproducibility?

The documentation linked in Table 1 would be a great place to start on any of the individual tools. Additionally, see the reproducible research workshop developed by NESCent: https://github.com/Reproducible-Science-Curriculum

References

Boettiger, C. (2015). An introduction to Docker for reproducible research, with examples from the R environment. *ACM SIGOPS Operating Systems Review*, *49*(1), 71–79. http://doi.org/10.1145/2723872.2723882

Gentleman, R., & Temple Lang, D. (2007). Statistical Analyses and Reproducible Research. *Journal of Computational and Graphical Statistics*, *16*(1), 1–23. http://doi.org/10.1198/106186007X178663

Gewin, V. (2013). Turning point: Carl Boettiger. *Nature*, *493*(7434), 711–711. http://doi.org/10.1038/nj7434-711a

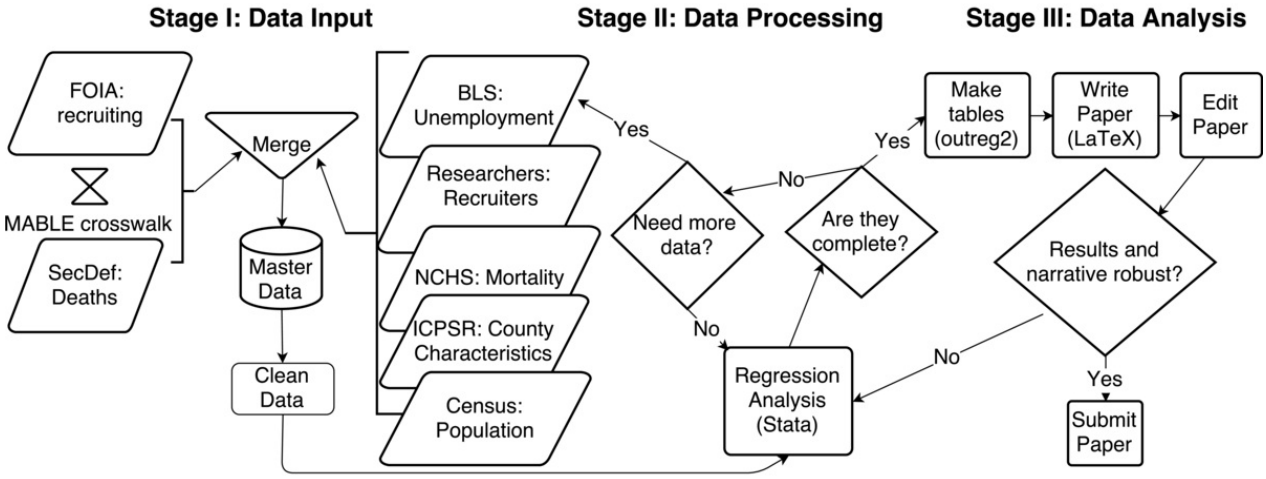
Mascarelli, A. (2014). Research tools: Jump off the page. *Nature*, *507*(7493), 523–525. http://doi.org/10.1038/nj7493-523a

Estimating the Effect of Soldier Deaths on the Military Labor Supply

Garret Christensen

My name is Garret Christensen. I am currently a project scientist at the Berkeley Initiative for Transparency in the Social Sciences and a fellow at the Berkeley Institute for Data Science. I work on questions of program impact evaluation in labor and development economics. I conducted the research described below as part of my dissertation work in economics at UC Berkeley, beginning in 2010. This research is on the effect of deaths of US soldiers in Iraq and Afghanistan on military recruiting. I use panel data methods with fixed effects to try and identify the causal impact of the death of a US soldier in Iraq or Afghanistan on recruiting in the soldier's home county.

Workflow



This project started with the idea, which I got from reading the newspaper, and hearing anecdotes about recruiting stations being overwhelmed after 9/11, and seeing popular reaction to the battle of Fallujah a few years later. The next step was obtaining the main data; a colleague just happened to have relevant data--the universe of US enlisted military recruits--through a Freedom of Information Act.

The other half of the main data is deaths of soldiers from Iraq and Afghanistan. I obtained this data from a public Defense Department website. Perhaps unsurprisingly, the original website is no longer operable. Luckily I still have, and have archived on Dataverse, the original dataset I downloaded and used. Updated versions of the data are also still available publicly. To merge the recruits and deaths data, I used Missouri Census Data Center's

MABLE/Geocorr to construct a geographic crosswalk. This uses census geographies, so I used the 2000 Census version, but sadly I don't think I recorded every exact option used when constructing this crosswalk.

I used Stata to merge this data together as well as to do all subsequent statistical analysis. No, Stata is not open source, but it's what most economists use. I did all my work in script (.do) files, and with the 'version' command, theoretically any other user should be able to produce the same results. The code was version controlled, but only after a fashion by updating script files with the date as part of the file name. Old versions of files just got dumped into an archive folder, where they were kept permanently.

The merged data was analyzed using the xtpoisson (poisson) and xtreg (linear) regression algorithms in Stata. Regression tables were output to tab delimited plain text files using the user-written 'outreg2' command, edited in Excel, saved as .pdfs, and then included in the LaTeX file that made up my paper. Clearly, Excel is the antithesis of reproducible, but I didn't change numbers in the tables, just formatting. Next on my to-do list with the paper is to cut out this clunky step and do directly from Stata to LaTeX. I think the only reason I started out this way is because I wasn't comfortable enough with LaTeX to figure it out.

Regression output was by no means complete the first time. I looped back numerous times to add more data from other sources, such as the Bureau of Labor Statistics, the Cenus, ICPSR, etc. This would require updating the merging and analysis code, reformatting tables, and changing the text in the paper that refers to specific output. Unfortunately that process is still ongoing because the paper has yet to be accepted at a journal.

Pain points

Given that this research is based on observational data, and I did not pre-specificy my statistical analysis plan, I highly doubt that any other researcher who looked at my original raw data (or even my cleaned final data) would agree on the exact set of regression specifications that I should include in my paper. For the sake of transparency, however, I try to include a nearly-exhaustive set of alternative specifications in a lengthy appendix to the paper. For example, all the results are available using both log-linear and Poisson regression specifications.

Regarding the data, even though I am very grateful to staff at the Defense Manpower Data Center who provided me with the data, I'm less than confident that multiple identical FOIA requests for the exact same data would result in identical datasets, since I don't have access to the original data, and have no way to verify if the dataset I was given was the true universe of observations I requested. Perhaps that's just an issue with all original data--you'll never be able to go back to all the homes in the census and check their answers, but you do have the ability of downloading the data from the census server. I don't have access to the

Defense Manpower Data Center's servers, but if I make available the data they gave me, does that mean we're reproducible? What if, as actually happened, you notice a completely implausibly low number of a certain type of observation in what is supposedly the universe of such observations?

Lastly, I'd say that my code is fairly well-documented, though it took a lot of work to get it there. Reading through it, I hope that other researchers could understand what the code is doing. There isn't yet a readme file but there is one master .do file that should (in theory) be able to reconstruct everything I've done from scratch. I have worked on the project for several years, with a few large breaks since economics journals can take 6 months to make decisions. I had to go back and extensively re-examine code that I no longer remembered. Having done that a few times, the code now seems fairly well-documented. This process would have been much easier had I kept a research log. I've had to open dozens of dated versions of the same file to find the last one written before a major change, which would have been much easier with version control or a research log. ### Key benefits

I would say that use of specific version control software is relatively new to the social sciences. When I began this work in 2010, I had never heard of git. I just used the method I learned from my adviser: include the date in the name of files, and every time you make significant changes to a script file (called a ".do file" in Stata), change the date. Using a distributed version control system (DVCS), as I do now, is a significant improvement.

Key tools

An excellent reproducibility tool to use is using the outreg2 (or estout) user-written commands in Stata to automatically turn regression output into journal-formatted tables. Although I use these commands, at present I have a clunky two-step process to first output the tables as .csv files before editing the formatting slightly then including the tables in my ultimate paper written in LaTeX. Ideal would be to use these commands to output the tables directly as .tex files, and include them in my paper file.

Questions

What does "reproducibility" mean to you?

Reproducibility to me is the ability of other researchers to get the same results as in my paper. In the weakest form, this would simply be for other researchers to be able to download my final datasets, run my final analysis code with nothing more than a file path name change or two, and get the exact results that are in my paper. A better version of reproducibility would be for other researchers to download my original raw datasets--the major two of which I have made publicly available using Harvard's Dataverse--redo my

extensive merging and cleaning of data, and then get the same results. Even better would be for others to go through the same Freedom of Information Act (FOIA) request process from the Office of the Secretary of Defense that a colleague and I did, redo the merging, redo my analysis, conduct the analysis they themselves see fit, and get the same results. I have some concerns about that, which are described below, but I've done the best I can, and rest on the assumption that any missing data is not correlated in a way that biases my estimates, and that I was thorough enough in my analysis that my results are robust to other forms of analysis.

Why do you think that reproducibility in your domain is important?

Significant errors have been discovered in high profile published economics research. In one sense, economics is doing well because many top journals require data sharing, so it's actually possible for these errors to be discovered since replicators have access to data. But without a systematic replication or code-checking of analysis, we still don't know what fraction of research suffers from these problems. Should we throw out the baby with the bathwater? I don't think so, but we don't know right now.

Also, even when economists share their data, they very rarely share their raw data, and all of their cleaning code, and instead only share their final data and analysis. We're humans, so we're probably making some coding mistakes that go unnoticed.

How or where did you learn about reproducibility?

My graduate adviser Edward Miguel taught me the simple method of version control with file names while I was working on a project of his as a graduate research assistant.

What do you see as the major challenges to doing reproducible research in your domain, and do you have any suggestions?

A lot of the advances in transparency and reproducibility in economics are coming from the medical sciences, where randomized control trials are nothing new. But RCTs are still a very small minority in economics. Like my work here, most work is observational. Economics only created the AEA RCT Registry in 2013, and there has been no serious discussion of registration of observational work. Should we register observational work? Should we pre-register our statistical analysis plans for observational work? This is all uncharted territory in economics.

What do you view as the major incentives for doing reproducible research?

Being reproducible requires extra up-front costs. In the long run, the benefits should outweigh the costs, because when someone comes along and wants to extend or replicate my research, they won't find any embarrassing errors in my work.

Are there any best practices that you'd recommend for researchers in your field?

Comment the hell out of your code so you know what you were doing when a journal makes a decision on your submission after 6 months. Save all your analysis files using version control. Use a one-click workflow to incorporate your tables directly into your paper so you don't lose track of output.

Would you recommend any specific resources for learning more about reproducibility?

J. Scott Long's *The Workflow of Data Analysis using Stata*

My *Manual of Best Practices in Transparent Social Science Research*

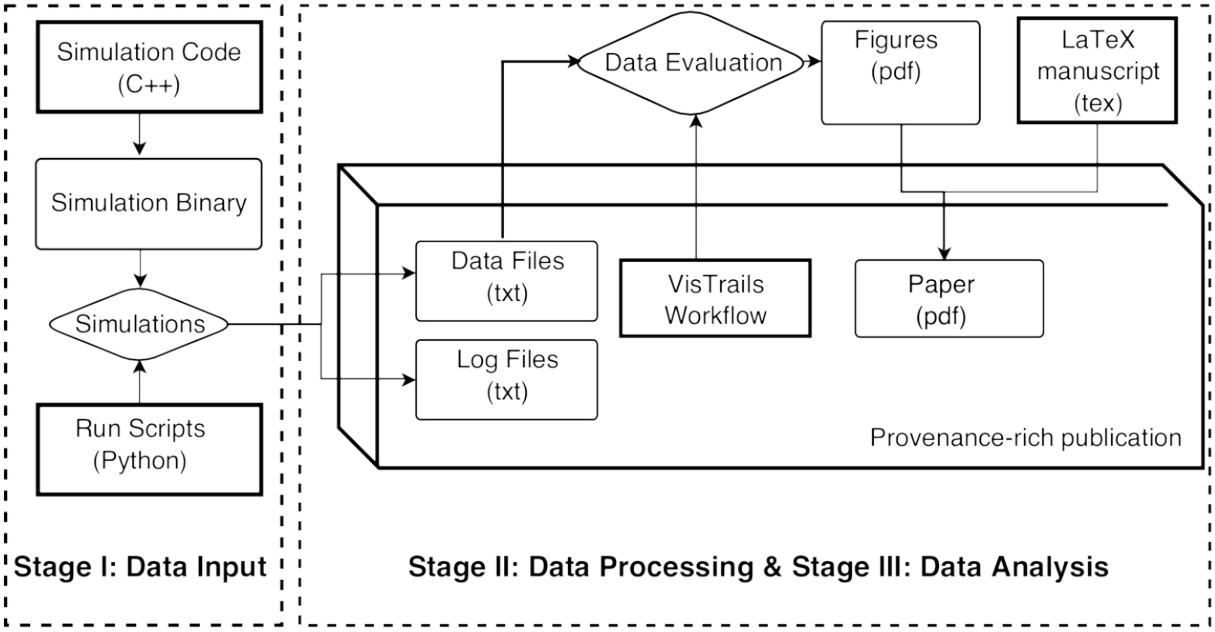
Turning Simulations of Quantum Many-Body Systems into a Provenance-Rich Publication

Jan Gukelberger and Matthias Troyer

My name is Jan Gukelberger, I am a computational condensed-matter physicist, who recently completed his PhD at the Institute for Theoretical Physics, ETH Zurich. This case study describes a project I worked on during my first year as a PhD student (2010-2011). This case study was conducted together with my advisor, Matthias Troyer.

Broadly speaking, the project's goal was to characterize a family of quantum many-body model systems. A specific model is described by a large matrix, the Hamiltonian, and its physical properties can be deduced from the lowest-lying eigenvalues (energies) and the corresponding eigenvectors (quantum states). Therefore I wrote a C++ program that would build the Hamiltonian matrix for a given set of model parameters, run an iterative diagonalization algorithm, and output the corresponding properties. Analysis of the results produced by this program for different parameters yielded a deeper understanding of the studied model family and corroborated analytical results obtained by colleagues. The analytical and numerical results were finally published together in Phys. Rev. B 85, 045414 (2012).

Workflow



Since the simulations may require a large amount of compute resources (on clusters or large workstations), it is usually not feasible or desirable to re-run the whole process in one go. We therefore typically adopt a two-step approach: The output of the simulation runs is treated as primary/raw data, which is archived along with log files containing detailed information about source code version, execution environment, and input parameters. The evaluation and transformation of this raw data to the final results (typically figures with plots) should then be as easily repeatable as possible, ideally with a single push of a button or script execution.

In this study, we opted to publish the raw data as supplementary information on the publisher's (APS) web server and provide workflow files for the VisTrails system, which would retrieve the raw data from the server and recreate the figures contained in the paper. VisTrails is an open-source scientific workflow and provenance management system which was used for the data evaluation and plotting tasks in the project. This way, any reader can inspect in detail and rerun all steps of our data analysis.

At the beginning of the project is the development of a simulation code in C++. Once the code is ready, it is used to explore the properties of the physical model under study. For this purpose, it is compiled and run with different input parameters on different systems (workstations and clusters). Because the simulation code is adapted and expanded continuously over the course of the study, it is essential to record what version of the code produced which results. To this end, we use a run script (Python), which records the code version (subversion revision), input parameters for the run, as well as details about the build configuration (compiler, libraries, etc.) and the system environment in which the code is run (host, date & time, dynamic libraries, etc.). All these details are written to a log file next to the data file that contains the results of the simulation. Both are semantically linked to each other by having the same file name, up to the extension (.dat and .log).

These output files constitute the raw data, which is collected on a desktop system for evaluation. The evaluation typically loads data files from several simulations (corresponding to different input parameters), computes some numerical transformations of the data, and finally produces one or more figures (pdf files) with data plots. We code the evaluation process in VisTrails workflows, making use of the ALPS package (delivered with VisTrails), which contains many utility routines for common processes like data transformations, fitting and plotting. We generally aim for a separate workflow (VT file) for each figure. This increases modularity and makes the development of the workflows easier, but implies that several VT files need to be opened and executed if all figures are to be recreated.

Finally, the manuscript of the paper is written in LaTeX, including the figure files created by the VT workflows. LaTeX compilation produces the paper as pdf, which constitutes the central part of the publication.

Publishing the paper together with the raw data and workflows, such that readers could easily inspect and reproduce our data evaluation process, turned out to be a challenge in its own right and required intense interaction with the publisher. Here, the main problem was the need for cross-references between the manuscript, the VT workflows, and the raw data, because the final location of each component only becomes available in the last step of the production process, when the files cannot be changed anymore without manual intervention from the production team. Some aspects of this issue are explained in detail in our report Publishing provenance-rich scientific papers, Procs. TAPP'11. In the end, the publisher was not able to insert links from the figures in the paper to the corresponding workflow files, but only a general reference to the supplementary material section on their server, where all the workflows could be downloaded.

Note: One collaborator actually recreated the figures with a different plotting tool before publication in order to improve their visual appearance. For this purpose, we amended the VT workflows to export the preprocessed data to an external file before plotting. Therefore, the figures presented in the paper are equivalent, but not identical, to the ones created by the VT workflows.

Pain points

Apart from the non-trivial publishing process, the main pain points during the study were connected to the fact that the data evaluation had to be done in the VisTrails GUI and to the opaque-ish VisTrails workflow file format:

- Data evaluation (execution of VT workflows) could not be scripted at that time.
- The evaluation could not be run on a cluster/via ssh.

- Version management was harder because viewing differences between versions was not as easy as looking at the diff file for a Python script.
- This also made the synchronization of workflows between different machines (e.g. laptop and workstation) less straightforward.

When now inspecting the "reproducible publication" on the APS server, three years after publication, some mid- to long-term issues become obvious, because both the used software and the publisher's infrastructure is evolving. Continuous testing and maintenance of the published instructions and workflows would be needed in order to keep up with the changes:

- The instructions we provided in the supplementary materials section accompanying the article do not work out of the box with the current VisTrails version: In the most recent stable VisTrails release at the time of writing (2.2), the ALPS package is broken and needs to be patched with the latest (not-yet-released) ALPS version. Otherwise initialization of the ALPS package fails and the workflow cannot be executed.
- The APS journals were not able to guarantee a long-term stable location for supplementary material. In fact, the URL has already changed, such that the workflows fail to fetch the raw data from the APS server, unless the URL is fixed manually in each workflow. For one specific example, the original location http://prb.aps.org/epaps/PRB/v85/i4/e045414/dyl_ladder_gap.zip has been changed to http://journals.aps.org/prb/supplemental/10.1103/PhysRevB.85.045414/dyl_ladder_gap.zip. Also, the cause of the error is not easy to fix for the uninitiated, because the DownloadFile module actually succeeds (it downloads the html file shown at the old URL), but the subsequent UnzipDirectory module fails with the message "BadZipFile: File is not a zip file". Hence we, the authors, need to prepare new workflows with adapted URLs and send them to the publisher for replacing the original ones whenever their infrastructure changes.

For these reasons I would now prefer to publish a self-contained archive containing the raw data and a script with minimal dependencies in a wide-spread language, such as Python, which reruns the analysis and reproduces the figures. This would be more robust with respect to changes in the publisher's infrastructure. Also, backwards compatibility issues might be expected to be solvable more easily in the long run for scripts in a wide-spread language, compared to special purpose solutions like VisTrails/ALPS (no matter how professional and helpful the developers of the software may be at the moment).

Key benefits

One of the most important points is recording exactly what version of the simulation code was run with what kind of input parameters. This excludes some of the worst cases of "non-reproducible results" and should definitely be a standard practice. (I cannot judge how established this practice is in our field because code and log files are rarely published.)

A second point is the actual publishing of raw data and evaluation workflows, allowing any reader to directly inspect all details of the evaluation process -- even those that the authors did not deem important enough (or forgot) to mention in the paper. This is clearly not widespread practice in our field and would be quite desirable in my opinion.

Questions

What does "reproducibility" mean to you?

In general, given a publication (in a refereed journal), source codes and raw data (which might be available publicly or in the institute's repositories), an expert from my field should be able to understand, and in principle repeat, every step of the study from the running of the correct version of the simulation code to the final results presented in the published paper.

How or where did you learn about reproducibility?

Some basic principles are quite evident, but integrating them in an efficient workflow may require some programming/version control experience. I came into contact with the VisTrails software due to a collaboration between our group and the VisTrails developers, aimed at integrating the evaluation tools of the ALPS package (developed within our group) with VisTrails.

What do you see as the major challenges to doing reproducible research in your domain, and do you have any suggestions?

The main challenge is probably making the recording of provenance data as simple as possible, so no-one has an excuse not to do it.

Another point is that simulation codes, raw data, and evaluation tools are rarely published. Most researchers are very reluctant to publish their codes, e.g. because they do not want competitors to publish results produced with their code before they can, or because they feel ashamed of the poor quality of their code. Raw data may be large and in non-standard format. And the evaluation may be performed by a chain of different tools, which makes publishing of the workflow hard.

What do you view as the major incentives for doing reproducible research?

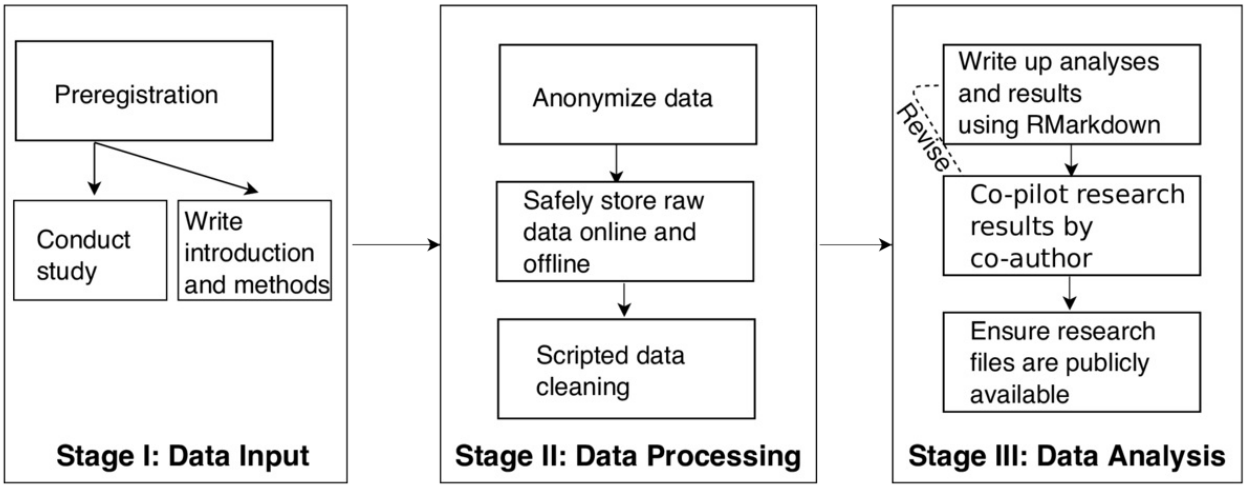
Apart from research ethics and institutional requirements demanding this, the recording of provenance information can make a researcher's life significantly easier when he/she discovers a discrepancy between different sets of results produced during a single study or in studies by different researchers. (Are the discrepancies caused by differences in the code, different input parameters, or data evaluation?)

Validating Statistical Methods to Detect Data Fabrication

Chris Hartgerink

My name is Chris Hartgerink, I am an applied statistician at Tilburg University specializing in detecting potential data fabrication with statistical methods. As a PhD candidate I pay attention to my workflow in order to increase efficiency and ensure it applies modern tools to improve my research. My case study will revolve around a project where I assess the performance of statistical methods to detect both genuine and fabricated data. In this project, I collect genuine datasets and invite researchers to fabricate datasets, to which I apply statistical methods to detect data fabrication.

Workflow



Statistical methods can be applied to detect potential data fabrication, but their validity is unknown. Simulating how researchers fabricate data is impossible because we simply do not know how researchers actually fabricate. Anecdotal summaries of uncovered misconduct cases do exist but are not generalizable. In order to test the performance of statistical methods in an ecologically valid manner, I invited researchers to fabricate data for a study for which we also have genuine data. With this, we can test the validity of a set of statistical methods to detect data fabrication. More specifically, genuine data contains sampling variance, whereas fabricated data frequently contains insufficient sampling variance. An example of such an analysis is testing whether nonsignificant p-values are uniform distributed, or whether higher p-values occur more frequently than lower p-values (which is theoretically implausible).

At the beginning of the project, we had an initial meeting to detail specific aspects of the project. This meeting is crucial to determine and assign responsibilities, start discussion of ethical considerations, and how the project will be conducted. For this project the main points we discussed were (i) the ethical obligation to guarantee participants that they would remain completely confidential, given that they were technically required to breach ethical standards, and (ii) how to convince the participants that this study had justified reasons to request this behavior. We required the participants to generate data, and decided to not save any identifying information by default and permanently delete the identifying they gave us as soon as practically possible. We decided to motivate the participants with the study goal itself and reward them depending on whether we could detect their data fabrication with statistical methods. Several default points I like to address in this initial meeting:

1. Agree that research files will be publicly shared by default and proper arguments are needed to *not* share research files publicly (e.g., ethics committee restraints).
2. Agree that the publicly shared research files are put in the public domain (licensed Creative Commons 0), to maximize breadth and clarity with respect to reuse rights of the research materials.
3. Agree that the research manuscript will be shared as a preprint upon completion for improved peer feedback, and that the manuscript will be published openly and with an appropriate and clear reuse license only (i.e., CC-BY or CC-0, definitely not CC-BY-Non-Commercial).
4. Role assignment (e.g., project lead, fund raiser, analyst, who will check the analyses).
5. Specifying research project

The project-lead subsequently followed up with a draft description of the project and an initial draft of the research materials. These contents included a technical description of the design, hypotheses and theoretical framework, but also a draft of how the data would be analyzed. This increased transparency in the analytic methods increases accountability amongst authors. Explicating these methods is important because things that seem obvious might not be, and this helps get everyone on the same page (e.g., do we include covariates? Do we agree how covariates are measured?). After sharing these drafts and iteratively revising them, the experiment was submitted to the ethical committee for review, given that the study involves human participants. Considering that each university has different procedures, I will not elaborate on these here.

The files that are drafted after the meeting are included in a GitHub repository, which allowed for version control of the files, providing a logbook. These files were synchronized to an Open Science Framework (OSF; osf.io) repository to increase discoverability and shorter URLs to include in the manuscript. Version control can be compared to track changes for computer files, allowing you to go back in time and view what changes were made and

when. With version control a logbook of changes is created, which improves the reproducibility of the research process if anyone is ever interested. My personal experience indicates that this logbook is rarely inspected (except in data audits), but it can serve as a handy reference when somebody asks you when a specific event occurred (e.g., when were the analyses programmed for the first time). Version control is most effective when started immediately after the initial meeting.

After the ethics approval was acquired, the version controlled research files were preregistered. This preregistration is done on the OSF. This preregistration makes an unalterable snapshot of the research files with a timestamp, which provided a confirmation that what we set out to do and expected was indeed a priori to actually conducting the study..

Following the preregistration, we actually conducted the study. Because our study ran for several weeks to reach the quotum of participants, I used that time to rewrite the preregistration into the introduction and method sections of the manuscript. Having planned this beforehand, I wrote my preregistration in such a way that it already resembled these sections. However, I can also recommend to be more detailed in the preregistration and subsequently prune it into the manuscript, given that manuscripts typically do not contain all the study details that matter.

When the study was conducted, the raw data were stored in a non-proprietary file format and as read-only files. It is important to ensure the file is read-only, so no accidental adjustments would be made to the file and that I could comply with the data policies (i.e., original file always needs to be retained). Saving these raw data as a non-proprietary format meant that I did not save it as an SPSS, Excel, or other commercial format, but as a clear-text file (e.g., a comma separated value (CSV) file). Clear-text files ensure that the data will remain readable in the future, whereas proprietary file formats might not be. Moreover, with clear-text files other people are not required to acquire commercial software packages to read the data.

After having safely stored the data, I cleaned the data in an automated, scripted manner. I conducted my data cleaning in `R`. I try not to clean data by hand, because I often forget what I have done; scripted data cleaning prevents this entirely. If I do need to manually clean data, the logbook provided by version control is a safety measure to allow the hard route to reproducibility; fully automated data cleaning is in the end the easy route to reproducibility. For example, in this project I had to split responses into separate datapoints, which required a few hours to automate, but doing it manually would have cost me more time and would have made it less reproducible (and more error-prone)

Subsequently I conducted the analyses in `R` with `RMarkdown`. `RMarkdown` allows data analysis and writing to be conjoined into one file. As such, all results were directly generated into the manuscript dynamically. This helped me to prevent errors and to increase the

reproducibility of results. For example, the statistical result F(1, 12) = 5.43, p = .037 would not be typed in by hand, but automatically generated with `R` code. With this we not only enhance reproducibility, but also prevent human errors; in the previous result p = .037 should be p = .038, a simple rounding error which is quickly made. For this project, I had written several specific functions to test for uniformity of p-values that were fabricated when there truly was no effect, to analyze the sampling fluctuation of the variances, and to combine these statistical tools to detect data fabrication. Based on these results, genuine and fabricated data collected were classified as genuine or detected with the statistical tools, which indicated the performance of these methods.

Upon completing the analyses, I requested a co-author to check all the analyses and results (what we call co-piloting). These comments lead to changes in the analysis script (e.g., data handling error), which were not a problem given the dynamic `RMarkdown` manuscript (another benefit: not having to redo all the numbers in the manuscript). After these errors were revised and checked once more, the manuscript and results were (mostly) reproducible. I typically do a final check to see if everything went as planned and whether all analyses can be run on an independent computer (i.e., whether there are unspecified dependencies).

The final step, prior to submission of the manuscript, is to ensure that the analyses corresponded to the preregistration and that all research files were made publicly available. Research has indicated that researchers who preregister analyses frequently report other analyses, indicating that is easy to forget what you actually planned to do at the start. Cross-checking this allows to pick up on these errors in time. Additionally, I have seen several articles where researchers said they made the research files publicly available, where their files were uploaded (e.g., Github) but not yet made public. These final checks ensure that results are according to the preregistration plan and can be accessed by others.

Pain points

The part of a reproducible workflow that I consider particularly painful is that of co-piloting analysis scripts. It shows when a researcher is reproducible but also shows it can be relatively complex to make reproducibility easy. It can sometimes take an entire day to check a colleague's analyses. However, as reproducibility increases co-piloting becomes less strenuous. Additionally, knowing the particularities of checking other people their work helps improve your own reproducibility. This is why I go through hoops to make sure *one* file is sufficient to get all the results in the manuscript and that dependencies or datafiles do not cause any trouble.

Another effortful aspect of a reproducible workflow is that the project lead often has to enforce reproducibility. I want my research to be reproducible, so I enforce this in my project. Co-authors need not have the same perspective on this and therefore do not feel

responsible for this. As such, you have to ensure that what they do is reproducible as well. If the project has a centralized project lead, this is not a huge problem. However, with more decentralized projects it can cause some difficulty. It requires you to structure the project thoroughly, but requires increasing effort with increasing project complexity (note that increasingly complex projects also have a higher need for reproducibility because of a higher potential for error-making.

Key benefits

My workflow has actively developed in recent years and this has culminated in analysis scripts that can run everything from the script itself. This requires nothing from the person trying to reproduce the results, except to download the script. It can be quite daunting when a researcher shares ten files and you have to find a way through them. It is not sufficient to be transparent. In order to become reproducible, it is highly important to structure your documents such that others, including your future-self, can understand what is going on.

Version control is a benefit within this reproducible workflow, considering that it goes beyond reproducibility of research results but also ensures reproducibility of the research process. My direct colleagues are starting to realize this as well; it is affirming to hear them stress that it helps them increase efficiency by allowing to retrace their steps. I hope that other colleagues will see the value in that sooner rather than later, (e.g., when their data gets audited).

Key tools

I use a set of tools which all have one thing in common: they are based on open formats that are timeless, inclusive, and can be used by anyone who has a computer. These open formats include the data in clear-text files, but also includes software packages that are open-source, whose code is checked by the open-source and academic community (e.g., `R`, `git`). It seems to me that the use of closed software has proliferated throughout the social sciences (where I operate most of the time) without the realization that it is actually hurting the future of science (e.g., irreproducibility of results), but also hurts current-day science. Not everybody can afford a license to SPSS or Microsoft Office, for example. Why exclude those who do not have those funds? Science is an enterprise that should be all-inclusive and not select on financial wealth of individuals or institutions. I try to reaffirm this principle by ensuring that all the tools I use are open-source and can be used by anyone who wants to.

Questions

What does "reproducibility" mean to you?

For me, reproducibility pertains to the reliability of research findings, which both includes direct reproducibility (i.e., can someone else reproduce the results by applying the described method to the same data?) and retest reliability (i.e., if we rerun the study, do we get similar results?). My case study focuses on direct reproducibility, that is, that anyone or a future-me can retrace the steps from the project in such a way that it is understandable and that the results are reliable.

Why do you think that reproducibility in your domain is important?

Scientists are humans and humans make mistakes. By using reproducible practices, we can discover these mistakes and not be led down a research path that is based on a mistake. It is important in my domain, because we preach that science has to be conducted in a reproducible manner.

How or where did you learn about reproducibility?

I got interested in reproducible practices during my master education when my supervisor introduced me to the idea of Open Science. I found myself wondering how to implement it in different stages of the research process, not knowing where to start documenting *during* the research. I learned much from colleagues across the world and across fields with who I discussed ways to be more reproducible (mostly on Twitter, which is a highly valuable resource for researchers).

What do you see as the major challenges to doing reproducible research in your domain, and do you have any suggestions?

Reproducible research is tiresome when you figure out a new way of doing things and then think that your previous work is incomplete. Additionally, not all colleagues are as enthousiastic about reproducibility and this can lead to discussions (also a good thing) that postpone implementing certain practices. It is very important to get everyone on the same page in the initial meeting on how the project will be managed, such that nobody is met with surprises and potential ambivalence at the end.

What do you view as the major incentives for doing reproducible research?

The main incentive for reproducible research is (future) efficiency. When you know that you can revisit projects from years ago and need at most 30 minutes to find what you are looking for is a major improvement over spending a day looking for that one specific value someone asked about in your email. It also helps revisit previous projects and see what I did, because I frequently unlearn things I require in a new project (e.g., I often forget how to make plots in the `ggplot2` package because I use it too infrequently, and I just reuse code from previous projects).

Are there any best practices that you'd recommend for researchers in your field?

The best practices I recommend any researcher to apply are the following:

1. License your work with an open license (CC-BY or CC-0), explicating free reuse of your materials and manuscript.
2. Script your data handling and analyses as much as possible, such that each step is reproducible.
3. Have a colleague check your analysis code, it is too easy to make mistakes. Not checking analysis code is comparable to not having co-authors proofread the manuscript.
4. Try and create an analysis script that can run automatically, downloading all required files and installing its dependencies. Otherwise, other people are likely to fail in reproducing your results, when they cannot get to the dependencies.

Would you recommend any specific resources for learning more about reproducibility?

I recommend the article by Karthik Ram on using version control in research. It opened my eyes on the use of version control as a project management tool that improves reproducibility at the lowest cost possible. Low threshold version control is available at the OSF, which provides online training tools (see osf.io).

Ram, K. (2013). Git can facilitate greater reproducibility and increased transparency in science. Source Code for Biology and Medicine, 8(1), 7.

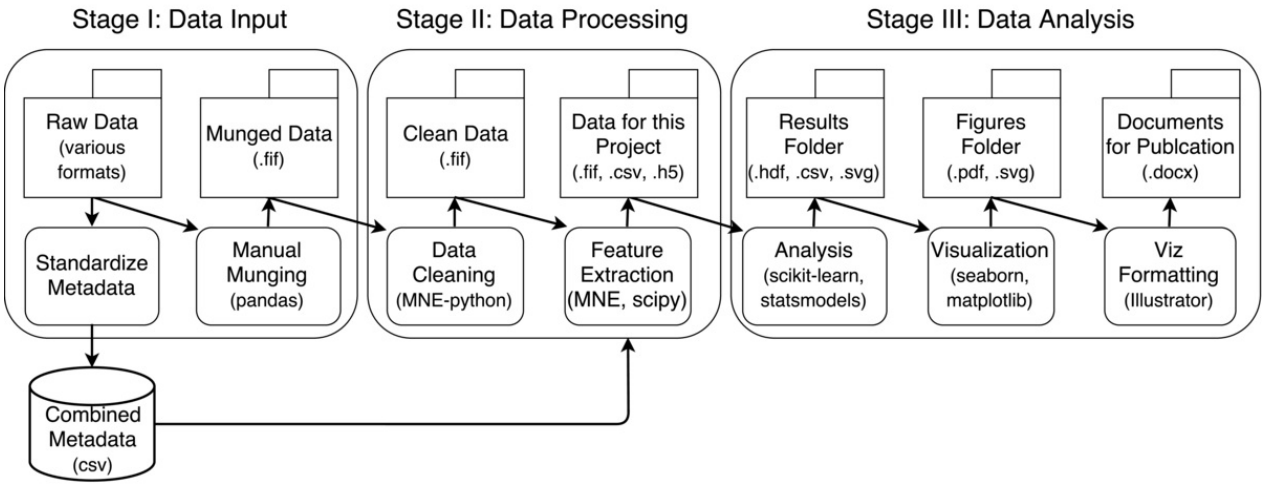
Feature Extraction and Data Wrangling for Predictive Models of the Brain in Python

Chris Holdgraf

My name is Chris Holdgraf, I am a senior graduate student with the Helen Wills Neuroscience Institute at UC Berkeley. My thesis work involves using predictive models to understand how auditory regions of the brain respond to acoustic features.

I am interested in how experience, learning, and assumptions about the world shape the way that we interact with low level features of sound. This involves a lot of computational work, signal processing, and data cleaning, utilizing a number of packages in the Python scientific ecosystem.

Workflow



My workflow involves taking raw data from a variety of sources, doing a few steps of munging on each one separately, combining them into a common structured format, and then doing further processing on this data. Here's a general breakdown:

Wrangling raw data

The raw data for my work involves electrophysiology signals collected from the brains of surgical patients in several sites across the country. This is challenging because the nature of the data is quite different from one location to another. I often get the neural data in many different formats, and a loose collection of metadata (e.g., sensor locations, names,

sampling rates) that can be anything from a PDF of hand-written notes to a structured text file. As such, the first step in my workflow is to wrangle all of this data into some kind of structured format.

First, I put all data into subject-specific folders. Each of these folders has sub-folders for different kinds of data (e.g., "raw", "munged", "meta"). The sub-folders will eventually be populated with data during processing, and the structure is consistent across all subjects so that I can easily parse them with scripts.

Next, I have a Jupyter notebook that is unique for each person, and is designed to take whatever raw format the data is in and turn it into a standardized version. This is called `munge_{subject_name}.ipynb`, and will output a file that I can use for the rest of my analyses. Jupyter notebooks are useful here because each subject is different, and will require a different set of steps to get the data ready. For this reason, I like to have lots of plots that go along with the analysis process, and a record of exactly what code was run to create the munged data for that subject.

Because the data often comes in different formats, I make the output of this step the same format for everyone. I use a Python package for neuroscience electrophysiology called `MNE-python`. This provides a common way of structuring data in order to streamline I/O, processing, and data analysis. I convert all of my raw data into the `.fif` format, which is a standard format for storing electrophysiology data. This means that I can read the data into other platforms (e.g., R or Matlab) fairly easily. The output files are stored in the folder `subject_name/munged`.

The final thing I do in this step is look at the metadata files for this subject, double check that all the values inside are correct (this is done with the munging notebook), and then insert them into a "combined" csv file of metadata. I have a Python script called `create_combined_meta.py` that will look through all the subject folders, find the metadata files that my munging notebook outputs, and turn them all into a single CSV file.

This aggregated CSV has data for all subjects that I have, and makes it much easier to quickly look at information across datasets. To do so, I use `Pandas`. This is a package that lets you represent tabular-style data in memory, and also gives you "database-style" functionality with their objects (e.g., joining two `Pandas` objects with partially-overlapping fields). Doing this necessitates that all of my data is in "tidy" format. This simplifies things, because it means that while I have a separate file for each dataset, I have a single combined file across all subjects for their metadata.

I should note that this is the point where somebody usually suggests using a "proper" database like SQL instead of keeping my data in CSV / FIFF formats. I've found that the overhead added by reformatting my data for something like SQL isn't worth the benefit it

would give. If I were to start a new project - particularly with larger datasets - then I would likely consider a more robust data storage solution like SQL.

Cleaning the data

Once I've created my munged data, I can now use a single script for processing/analyzing all datasets. In the field of cognitive neuroscience, there are common preprocessing techniques that are carried out in order to improve the quality of the data (e.g., a few filtering steps and rejecting channels that are too noisy). I have a `clean_data.py` file that will look through the "munged" folder of each subject, load the data, run the cleaning, and then output the results to the folder `{subject_name}/clean`. This way, I know that any data in the clean folder is ready for further analysis.

At this point, I have cleaned data in each subject's folder, I also have that subject's metadata inserted into a CSV file with all subject information. From now on, I can just load a subject's data file, then load the metadata CSV file for everybody, and query only the rows that belong to the subject that I care about.

Defining a project

When I begin analyzing my data to answer a specific question, I create a new project-specific folder that exists alongside my "data" folder. Each project generally entails a number of analyses, and this is a way to keep them all in the same place. The project folder is structured similarly to the "data" folder. It has a sub-folder for "scripts", for "data", for "results", and for "documents" and any information necessary for publications that come out of this project.

For example, the first thing I might do is create some Python script to extract features of interest. I will put it in `project_name/script/feature_name/extract_feature.py`. The script assumes that there is data for each subject in the "clean" folder. It will pull the data from "clean," extract whatever feature I'm interested in, and then save the result to the project-specific folder, something like `project_name/data/my_feature/{subject_name}_feature.fif`. I parse all subject folders and save files in the same manner.

Storing the extracted features for all subjects in a single project-specific data folder makes it much easier to develop scripts/notebooks to further analyze the results, since I don't need to keep track of which features have been extracted for which subjects. I also develop feature extraction scripts using the Sun-Grid engine (a platform for dividing computation between a cluster of computers) for speeding up my analyses. I can do this relatively easily because the folder structure for each subject is the same.

Running analyses

Now that I have a set of features created for each subject, it is time to run analyses and answer questions. These scripts also exist in the project-specific folder, and assume that there is data in the "project_name/data/my_feature/" folder.

A difficulty that I've had is knowing when to keep your analyses in interactive notebooks vs. Python scripts. I generally pilot my analysis interactively - this lets me do sanity checks and on-the-fly calculations that help me develop the final analysis. Once I have code that does a specific analysis, I will put it in a `.py` script.

The output of this script then goes into a `project_name/results/my_analysis` folder. They may be in the form of PDFs and SVGs for further inspection, or data files (e.g., CSV) representing model results (such as regression coefficients). For anything consisting of lists, numpy arrays, or simple dataframes, I use the `h5io` package, which provides a fast way to read/write collections of data to `hdf5` files (another standardized data storage format).

Finally, I use the outputs of the analysis script to create visualizations and decide whether or not my analysis actually worked. I use another set of notebooks to read in the results, perform last-second wrangling, statistics, etc, and output visualizations. If I have a publication-ready figure in mind, I will create a notebook specifically for that figure in another folder called `project_name/fig_{analysis_name}`.

When creating actual figures for papers, I like to use software like Adobe Illustrator to make sure that my fonts are the proper size, well-spaced, etc. I use visualization notebooks to create high-res PNG versions of my data that have most formatting stripped away (except for the data). These plots are then linked to an Illustrator file, so that they are updated automatically when a new plot is created. This way I can easily arrange my plots and standardize fonts without doing a lot of manual tweaking.

Finally, I use Microsoft Word to write drafts which I put in the "doc" folder. These pull from the figures I've created in the "fig" folder. Ideally I would use text files here with LaTeX, but the team that I work with makes this prohibitive.

A general note

This process has been refined many times over the past year, and the original structure looked very different than this. My original file system had code and data living in totally different places. Moreover, it had project-specific scripts and more general utility scripts living in the same place. The goal of this file structure is to keep data and scripts together when they have a natural pairing, and to separate out my more production-ready functions/modules from "hackier" project-specific scripts. Below is a list of some things that I've learned along the way, and that have guided the development of this system:

1. For any data/code that are project-specific, keep these together in the same general file hierarchy.
2. Rather than creating metadata structures that live next to the data, come up with a “master” metadata template, then store entries from every subject in a CSV file that follows this template. Rather than storing data in separate subject files, include an entry with “subject id” in the metadata file so that you can pull out individual subjects in this way.
3. Utilize other packages whenever possible, particularly with the annoying task of data I/O. In my case, I store all my raw data as ‘.fif’ files, which can be opened easily in Python or Matlab with well-supported third party packages. I also use `pandas` and `h5io` to read / write metadata files, which makes it very easy to slice and dice these files for particular entries that I want.
4. More generally, take a “database” approach to how you store any data. Even though I’m not storing data in a MySQL database per-se, I can still draw inspiration for how this data is organized. Treat data entries as rows, and data features as columns, and then combine / split up the data using pandas database-style syntax (e.g., joins and merges). A great guideline for this is the "tidy" data specification described by Hadley Wickham.
5. Put ad-hoc code in a project-specific folder. Be much pickier about code that you expose in public Python modules for any project. If you think a function is worth generalizing, then move it out of the project folder and to its proper module, and document it extensively.
6. Do all coding with automatic PEP8 and PEP257 checkers. PEP8 is a set of standards for naming conventions, code syntax, using white space, etc to ensure that code is clean and readable. PEP257 is a set of similar guidelines for docstrings. Many "fully-featured" text editors have plugins that automatically check code using these guidelines and highlight errors, which is useful for quickly writing clean code.
7. Make a conscious effort to structure Python scripts differently from Jupyter notebooks. Structure code (and data) such that it lends itself well to scripting, rather than assuming interactive sessions for everything.
8. Use Jupyter notebooks to glance at the data and preliminary results, but move code into scripts as it becomes more refined or complex. This avoids creating a mega notebook with tons of different analyses in it.
9. Structure code so that some scripts live with the data that they operate on. E.g., if you’ve got a script that only operates on a specific collection of data, renames specific columns in that data, and always saves it to another location relative to the original data,

then create another folder “script” right next to that data folder. Put all data-specific scripts into this folder. This way, you know that scripts operate with relevant data nearby.

10. Finally, this is not specific to this project but has been very useful to me. Find opportunities to contribute to other open-source projects. Open pull requests and learn about how to use the code. The back-and-forths and input you get will make you a much better coder, and your codebase / research will greatly benefit from it.

Pain points

The hardest part of my workflow has been deciding how to balance flexibility and control. On the one hand, you don't want your scripts to be so specific to data that they break as soon as anything changes, but on the other hand creating code that generalizes well and isn't terribly confusing is really difficult to do. In my case, I initially made errors on both of these fronts, but the current structure of my data seems to be more intuitive, easy to maintain, and easy to grow.

Another big issue I've had lies in dealing with different formats of data and information. For example, I want to version control all of the code that I write, but:

- Does that mean that I should create one big repository for all of the code described above? What about a separate repository for each project?
- How should I split up the general modules and functions vs. the code that only lives with a particular project?
- Finally, how should I deal with the fact that there are lots of other "non-code" files living with this file structure (e.g., images)? Should they be version-controlled, or should the code just assume that the data lives in particular folders? What would happen if somebody else copied the code and didn't get the data?

I don't necessarily have good answers for these issues, and I'm still coming up with a solution that makes me happy, but these are some things that I'm thinking about.

Key benefits

The biggest benefit of this system is the fact that messy, subject-specific data is quickly turned into a standardized format that is consistent across all of my subjects. This is useful because it means you can write scripts that analyze the entire dataset without doing a lot of extra customization. Moreover, because the structure of the filesystem is the same for everybody (including things like naming conventions), it is easy to find what you want from each dataset.

Another benefit is the fact that I am storing code along with the data that it operates on. Some people feel strongly that this is a bad idea, but I've found it to be useful so long as the within-project folder structure still makes sense. In previous workflows, I had all of my code in one folder, and all of my data in another folder. This often led to confusions where I was unsure which code operated on what data. It also made it more difficult to connect the steps of preprocessing and feature extraction chains. Now, if I want to know all of the things that have been done to a collection of data files, I just need to look into its corresponding "script" folder.

Finally, by separating out operations that are true for all projects (e.g., data munging and cleaning) and those that are project-specific, the scope of individual projects becomes more clear and easy to follow. I think of the data pipeline as a single tree trunk, where projects branch out from this trunk and do extra things to the data, on top of the base workflow of preprocessing. Now, my file structure more naturally follows this concept.

Key tools

The two most useful tools that I have found are `Pandas` and `MNE-python`. Pandas made it much easier to embed metadata with the signals that I analyze. It allowed me to store information from lots of subjects in a single CSV file, and treat it as a "database" by using queries on it. `MNE-python` is a package for electrophysiology in neuroscience written in Python. When I discovered it, I found that it duplicated many of the functions I had already written, and in general did this much better than I had. Moreover, it has a lot of convenience functions for doing I/O, which up until then was a pain to maintain. By using these two packages, I was able to significantly cut down on the amount of custom-written functions that I used to wrangle my data.

Questions

What does "reproducibility" mean to you?

The discussion in this writeup covers the first 6 months of the project. To that extent, my definition of "reproducibility" means actually being able to reproduce my own results (aka, coding for my future self). I ran into a lot of issues to keep things streamlined and understandable in my own head, which made it difficult to interpret my findings. Obviously this would generalize to other scientists trying to reproduce my analyses and work as well.

Why do you think that reproducibility in your domain is important?

Because it's a guiding principle that will make my code more understandable, maintainable, and extendable for others and for myself.

How or where did you learn about reproducibility?

At first it came from teaching a few Software Carpentry classes and reading things online. Lately, I have gotten a lot of help by contributing to the `MNE-python` project, as I've found that going through the pull request process for a well-maintained project is a great way to learn a lot about coding well.

What do you see as the major challenges to doing reproducible research in your domain, and do you have any suggestions?

In my case, legality is a big problem because I'm dealing with medical data that cannot be shared publicly. Another big problem is simply a matter of training and incentive. Right now there is little opportunity to learn how to code well or how to make reproducible science. To make matters worse, I see very little incentive for anyone to actually do so (if they want to be a tenured faculty).

What do you view as the major incentives for doing reproducible research?

Other than the warm fuzzy feeling, I think the biggest advantage is that when you code and organize for other people, you also code and organize for yourself in the future. This makes your life much easier in the long run.

Are there any best practices that you'd recommend for researchers in your field?

Front-load a lot of thinking/planning before you just start creating scripts and functions. Spend a good chunk of time thinking "big picture" early on, then zoom in and build some stuff, then zoom back out and decide if it was a good idea or not. Don't get lost in the weeds.

Would you recommend any specific resources for learning more about reproducibility?

Software Carpentry is a great one, but most other stuff is just scattered on stack overflow unfortunately. I think that finding a good package that has a sweet-spot of contributors (aka, not so few that you don't get feedback, not so many that it's a huge pain to do anything). Try

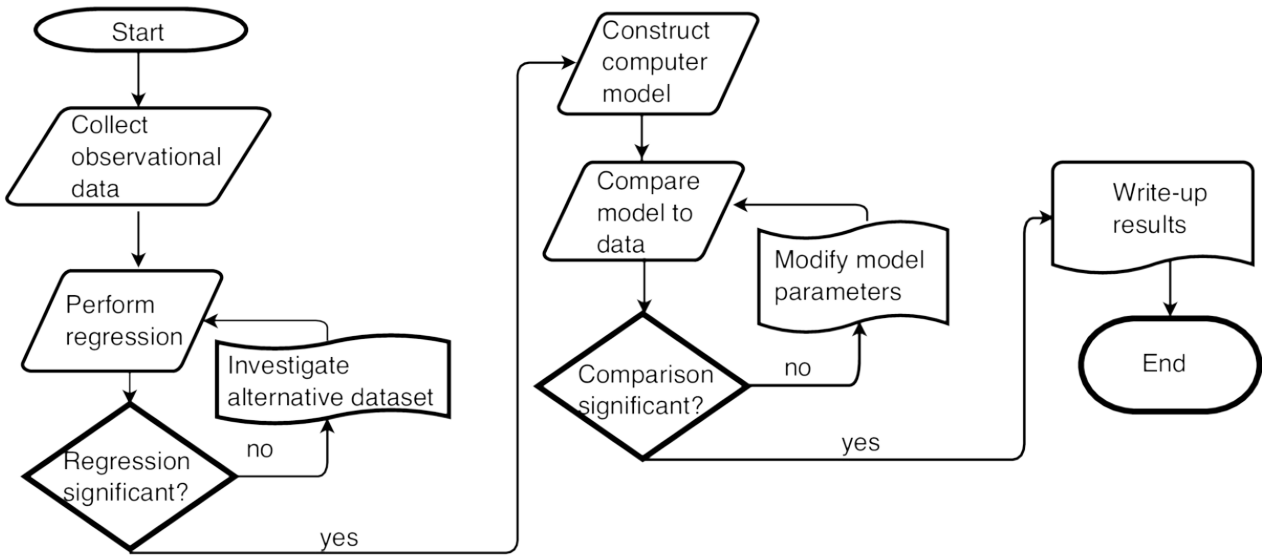
to contribute something via a pull request and learn from the other people in the community. It will be a great way to learn good coding principles. Finally, find a community around you (at the university, at local companies/hackathons, etc) that shares your interests. Spend time learning and teaching with these people.

Using Observational Data and Numerical Modeling to make Scientific Discoveries in Climate Science

David Holland and Denise Holland

My name is David Holland. I am a Professor of Mathematics and Atmosphere Ocean Science at New York University's Courant Institute. I study global sea level rise in a changing climate, specifically, how changes in global weather patterns affect the melting of the great ice sheets, with Denise Holland, who is the Field and Logistics Manager for our field research program.

Workflow



The goal of our workflow design is to use observations and computer models to make new discoveries about the natural environment particularly in the context of climate change. The approach we take in our algorithm design is to generally always look for an interesting phenomena in the observational record first. If we find it there, then we generally proceed to try to simulate it in a computer model as an independent check on the physical plausibility of

the phenomena in question. In this very specific workflow, we will illustrate our approach for the particular question: does the North Atlantic Ocean drive climate change in Western Antarctica?

The very first step in our workflow is to collect all available observational data of North Atlantic Ocean surface temperatures going back in time as far as is possible. The more spatial and temporal data we have, the more robust the results will be. On occasion, different datasets can be contradictory or inconsistent. In this case, we have to make subjective decisions if one dataset is better than the other. If we cannot make this decision, we cannot proceed and the algorithm has to terminate. On the other hand, if two different datasets agree, then we proceed with greater confidence of the quality of our input data. In our study, our analysis of the North Atlantic temperature data strongly showed us that there exists a 60 year period oscillation in the surface temperature in a broad pattern that covers the entire North Atlantic. This is known as the Atlantic Multi-decadal Oscillation (AMO). This result has been previously found by other researchers so our very first step has not only given us confidence in what we are doing, but also reproduces a result from other researchers. All of our science research tends to in fact work this way in the sense that our new findings tend to be built on reproducing previous work by others and then extending that into new, unchartered research areas.

The next dataset we investigate is the surface winds over the Western Antarctic region. In exact analogy with the processing of the North Atlantic surface temperatures above, we proceed here to look for long term trends in winds. We find such a trend and it matches with the North Atlantic Ocean surface temperatures, suggesting one is causing the other. We claim such a relation based on a formal regression calculation which shows our finding is statistically significant but still does not explain which is the cause and which is the effect.

The next step in the algorithm is to employ a physically based numerical model of the global climate system. Using such a model, we can impose the observed North Atlantic Ocean surface temperatures in the model, and the model can simulate the response of the global atmosphere to this imposed ocean forcing. We carry out the simulation and we find that North Atlantic Ocean temperature oscillations drive wind circulation anomalies in Western Antarctica. This is a very surprising, non-intuitive result. We also try to model in the opposite sense, and impose surface wind anomalies in Western Antarctica to see if they drive ocean temperature anomalies in the North Atlantic. The simulation showed that this does not happen. This gives us some confidence to conclude that the direction of flow of climate change is from the North Atlantic to West Antarctica.

At this final stage, having made a new discovery in two independent manners, one purely observational and one purely computer modeling, we are ready to report our findings to the scientific community. This involves a rigorous peer-review process that imposes a number of reproducibility requirements. A section of the manuscript must be devoted to explaining

where all datasets are located and how a reviewer or future reader could access the same datasets we use. Likewise, we are required to describe the computer model we used and how a future researcher can access the same model. While the main scientific article is relatively brief (about 4 printed pages), giving the reader the essential information on what we found and how we found it, we also write a supplementary materials section. This is an exhaustive description of each step we took with our observations and our modeling.

The original data sources that we used for ocean temperatures and atmospheric winds are on-line and available through national climate data repositories. The numerical modeling code is available on-line through national climate modeling centers. The regression calculations and the numerical simulations we preformed are very large and are not stored on-line but are archived at NYHU on hard disks off-line. The regression code is standard and available at many repositories such as part of Matlab. It is well documented, well tested, with many examples. The numerical climate code is used by a large number of people, it is well documented, well tested, with many examples. There are no restrictions on other researchers replicating or confirming our work. We warmly welcome such activity.

As mentioned in our published paper, there is a supplementary document that includes details about the data processing workflow. There is not the actual computer scripts used to perform the regressions nor those to run the numerical climate model. These could in fact be put on-line if there was a repository for such information. However, in our estimation, if someone was to try to reproduce our research it would probably be more natural for them to write their own scripts as this has the additional benefit that they might not fall into any error we may have accidentally introduced in our scripts.

Our published work is citeable as: Li, Xichen, David M. Holland, Edwin P. Gerber, and Changhyun Yoo. "Impacts of the north and tropical Atlantic Ocean on the Antarctic Peninsula and sea ice." Nature 505, no. 7484 (2014): 538-542.

Pain points

The most difficult part of reproducing these results is the sheer volume of the datasets involved and the amount of computational storage and time required to complete all these calculations. Transferring large volumes of data from super computers (where the main code is run) to personal computers (where the analysis is generally performed) is an onerous and time consuming task with many failure points. Often data transfer is incomplete, storage disks break or fail, and weeks or months of research time is lost. In such case, one has to simply go back to the start and begin again.

Key benefits

Our approach of independently using observational data and modeling is a stronger approach than that of just using one or the other. Our approach also is to abandon findings that are contradictory between independent observational datasets as this suggests that the data is of inadequate quality to proceed further with any analysis or conclusion. In other words, if you find something interesting in analysis of one dataset, but not in another similar one, despite the temptation to proceed with the interesting finding, one must acknowledge that the contradiction prevents moving forward.

Key tools

We have nothing special to report here but are aware of efforts in the computer science community to better track the workflow stages of a research project. In our project, such software would have been beneficial in that our workflow algorithm could be online and a user could click through it and find the scripts and datasets and models we used.

Questions

What does "reproducibility" mean to you?

"Reproducibility" for me means that someone, anyone, could read my published research, then take the datasets I have archived on the web and use them to reproduce the results I had in my published paper.

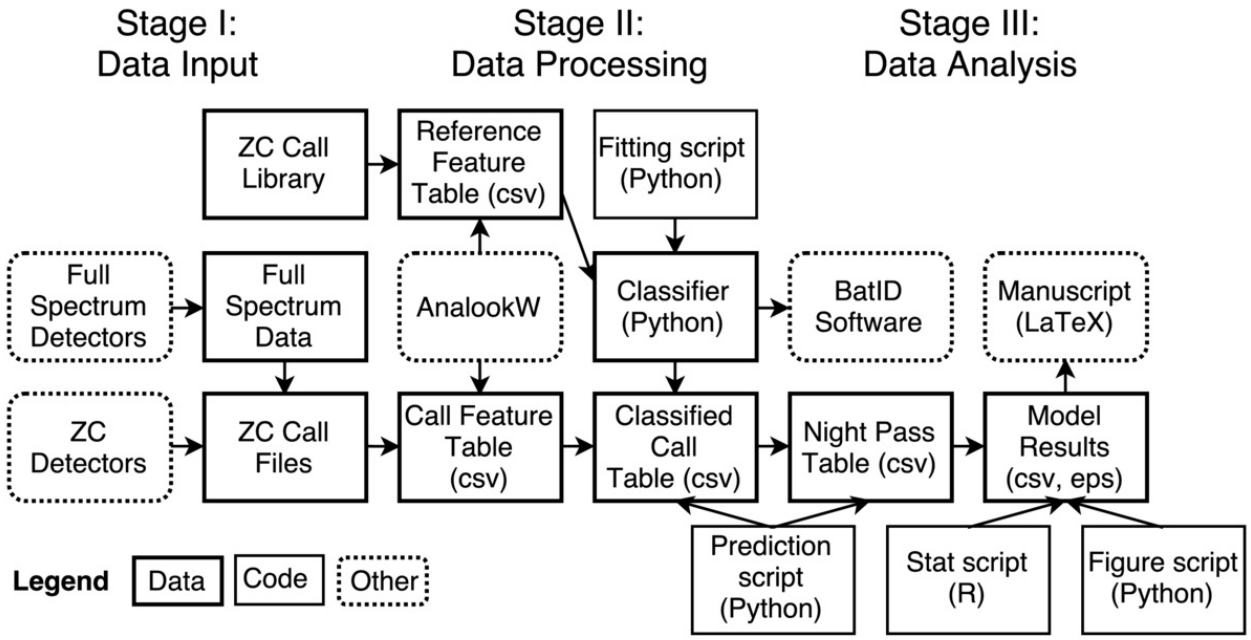
Reproducibility is at the heart of natural science. Without being able to perform an experiment and achieve a certain result, and then to have an independent scientist reproduce the same result, the research finding is murky. Our publications in high-profile journals, such as Science and Nature, demand that we include a methods section in our papers, as well as carefully document the location of all our datasets on the internet. Some of the research code developed requires years to master. Scientific funding and the number of scientists available to do the work is finite. Therefore not every scientific result can, or should be reproduced. The most important, paradigm shifting results should, however, be reproduced. In the case of climate science, important decisions by world leaders rely on scientific findings. These finding must be robust and reproducible in order to guide energy use policy. In our study, we reached the same conclusion using both purely observational data and then independently through a computer simulation of the same phenomenon. Finding the same result in two completely independent manners gives us confidence in our findings.

Analyzing Bat Distributions in a Human-Dominated Landscape with Autonomous Acoustic Detectors and Machine Learning Models

Justin Kitzes

My name is Justin Kitzes, and I am a quantitative ecologist who works to develop new theory and datasets for predicting the effects of land use and climate change on biodiversity. I am currently a postdoc in the Energy and Resources Group and a Data Science Fellow in the Institute for Data Science at the University of California, Berkeley. My field work focuses on the ecology and conservation of bats in complex, human-altered landscapes. This case study decribes the use of acoustic detectors, machine learning, and likelihood statistics to examine the effects of three large Northern California highways on bat activity.

Workflow



This study investigated whether several common species of bats showed decreased activity along three large highways near San Francisco Bay. Activity in this study was defined as the number of ultrasonic foraging calls recorded by autonomous acoustic detectors. The core tasks involved collecting raw call data using the detectors, extracting features of the recorded calls, classifying the calls to the species level, and performing statistical analysis

on the resulting nightly call counts as a function of predictor variables, including distance from the road. The complete analysis is described in a manuscript published in *PLoS ONE* in 2014. We later used a similar workflow to conduct a second study, published in *Agriculture, Ecosystems & Environment*, of the predictors of bat activity in vineyard landscapes.

Two different types of acoustic detectors were used, one of which recorded data in zero-crossing format and the other in full spectrum format. The full spectrum data were converted to zero-crossing format using a closed-source utility provided by the detector manufacturer, with conversion parameters selected to produce output similar to the recordings from the native zero-crossing detector. The free, closed-source software AnalookW, developed by an individual researcher who has been active for many years in bat call analysis, was used to filter out files containing only noise and, for the remaining files containing calls, extract twelve features describing from each call. These features were saved in a csv table.

The calls were then classified by species, which required the construction of a custom random forest classifier. A reference library containing zero-crossing calls made by individual bats identified in hand was obtained from a personal contact, and the same twelve features were extracted for these calls using AnalookW. A random forest classifier was trained on this data using the Python package scikit-learn v0.12. Classifier accuracy was evaluated using cross validation and a confusion matrix.

The classifier was then used to identify the recorded calls to the species level, creating a classified call table. This table was summarized into a nightly pass table, which aggregated the calls into passes consisting of multiple, closely spaced calls and summarized the number of passes of each species recorded in each night, at each distance from the road. Environmental and site variables were joined to this pass data to create the final table for statistical analysis.

As functions for fitting generalized linear mixed models (GLMMs) were not available in Python, statistical analysis was carried out in R. Exploratory analysis showed that a Poisson regression was not appropriate for the data, so a negative binomial GLMM was fit to the nightly counts of all species and separately for four common species. The model results were saved as a table that later appeared in the final manuscript. The model result table was then read by a Python script, which created and saved a figure that appeared in the final manuscript. The final manuscript was written in LaTeX and submitted to journals in that format.

In addition to the manuscript, a second output of this project was the open source software BatID, which bundled the classifier object with a browser-based interface to enable non-programmers to automatically classify California bat calls. This software is freely available for download and has been used by researchers in academia, government, and the private sector.

Pain points

At the beginning of the workflow, two closed-source graphical programs had to be used, one to convert a proprietary data format to the zero-crossing format and the second (AnalookW) to perform feature extraction on the zero-crossing call files. Both of these steps required parameters to be entered into these programs, which I was careful to document manually, as this information can otherwise easily be lost. AnalookW runs only on Windows, which required me (and any analysts wishing to use my later software BatID) to locate a Windows computer to complete this step. Although I code faster and more accurately in Python, I needed to switch to R for statistical analysis, as the necessary packages were not (and still are not) available for Python. A major headache at the manuscript stage arose because the R statistical functions reported output only as a non-machine-readable text file or as an object, which required me to create the final table, containing coefficients and standard errors for 14 variables across 5 models, by hand.

Once I created and released the BatID software, a problem immediately arose when the scikit-learn package was updated to v0.13, which could not read the classifier object created during my analysis. Additionally, the original BatID package required a user to install a full scientific Python stack, a task that proved difficult for precisely the audience of non-programmers that I was hoping to reach. I eventually used pyinstaller to create a standalone binary executable for Windows, reasoning that users of the software needed a Windows computer anyway in order to run AnalookW as a prior step in the analysis. Creating this distributable binary was not straightforward and took many days of trial-and-error testing and manual tuning.

Key benefits

Of all aspects of the analysis, I am particularly happy about the effort that I put in to creating the BatID standalone classifier software. As the many of my colleagues in ecology are non-programmers or novice programmers, I believe that these types of user-friendly tools are critical to advancing the state of science in my field, as well as to supporting the uptake of new methods by non-academic non-profit and agency scientists. I hope that more of my computationally-oriented colleagues will engage in similar activities in the future.

Ironically, of course, in creating a tool for non-programmers, I also created another graphical program that cannot easily be scripted into a workflow such as the one. I attempted to ameliorate this concern in the most recent BatID version by requiring users create a text file containing all parameters, which is read by the program along with the data file, and having the program save all results in the same directory as the parameter file, along with a log file. This at least ensures that there is, by default, some record of the program version, time, and parameters used to process the raw data into classified results tables.

Questions

What does "reproducibility" mean to you?

I consider a study to be (computationally) reproducible when I can send a colleague a zip file containing my raw data and code and he or she can push a single button to create all of the results, tables, and figures in my analysis. It can, of course, be quite challenging to achieve this goal with anything short of the simplest scientific workflows.

Why do you think that reproducibility in your domain is important?

I think that reproducibility is particularly important in fields like ecology in which researchers are constantly striving to make increasingly detailed inference and predictions using relatively scarce data. Although I do not have evidence to this effect, it seems logical to me that in these "high leverage" types of analyses, small analytical decisions (how data are cleaned, the options passed to optimizers, etc.) could play a disproportionate role in influencing the eventual study conclusions, and thus need to be fully documented and shared. An easy way to guarantee that all of these decisions are recorded is to make one's entire analysis reproducible by others. More broadly, I feel strongly that reproducibility is a basic component of good science. Now that "doing science" requires communicating more detail than can be easily expressed in narrative form in a manuscript, releasing code and data seems completely necessary, where feasible, across all domains.

How or where did you learn about reproducibility?

I started learning these tools through workshops, in particular Josh Bloom's Python bootcamp at UC Berkeley and by teaching Software Carpentry bootcamps with other experienced instructors. I also learned a great deal by working closely with Chloe Lewis, a former student in my Ph.D. lab who had previously worked at Microsoft, on the development of our software package *macroeco*, which I continue to maintain. I picked up more advanced techniques and ideas mostly through web searches while attempting to get unstuck from issues that arose in my own research and software development.

What do you see as the major challenges to doing reproducible research in your domain, and do you have any suggestions?

A major challenge in ecology continues to be data sharing and access. Many field ecologists, understandably, are reluctant to share their hard-won raw data with other researchers. I suspect that this caution arises both from a sense of professional necessity (i.e., I invested a ton of time collecting this data and I am going to be the one to publish all of the analyses using it) and from the feeling that the numbers alone cannot possibly capture all the subtle nuances they observed in the field that are important to truly understanding the data, its potential, and its limitations. In particular, information about sampling bias (as derived from choices of study site, sampling techniques, season, missing data, and many other factors) cannot easily be described in numeric form. I also suspect that many field ecologists recognize that this information isn't even in the manuscript in narrative form, leaving the person who collected the data is the only one qualified to analyze it. What data is published and available tends to be relatively small and of heterogeneous format, and thus tends to be locked up in printed pdf tables and other non-machine-readable formats.

What do you view as the major incentives for doing reproducible research?

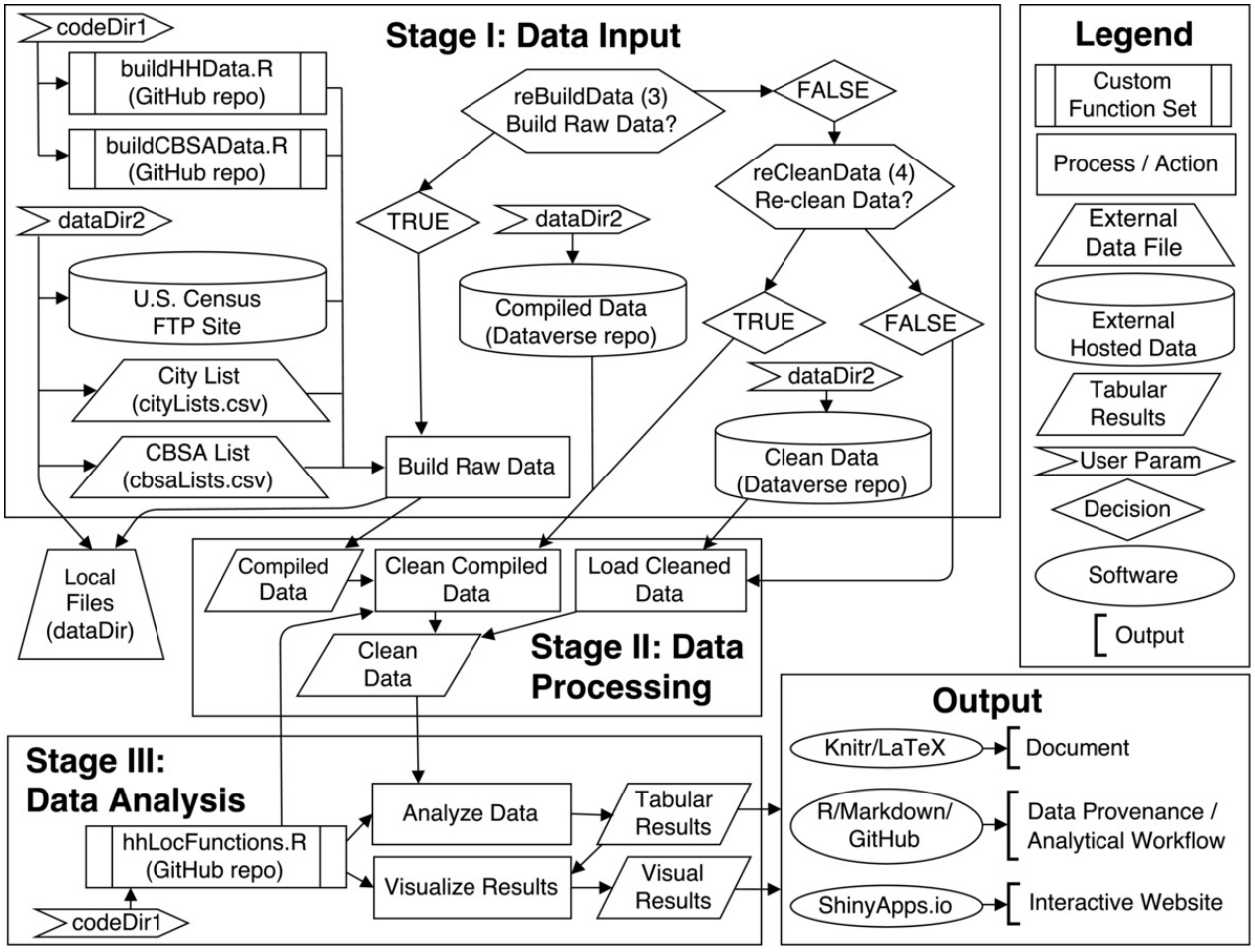
I'm not sure that there are any major external incentives in my field -- certainly, in principle, releasing reproducible research could increase the number of researchers who end up citing your manuscript, but this seems somewhat indirect. Some journals, like PLoS, are now mandating that all novel computer code be uploaded as manuscript supporting information, but it seems that this requirement is not thoroughly checked at this point.

An analysis of household location choice in major U.S. metropolitan areas using R

Andy Krause

I am Andy Krause, a Lecturer in Property (Real Estate) at the University of Melbourne. My research focuses on the spatial analysis of real estate markets, particularly in regards to valuation and location. This work was completed with my colleague, Hossein Estiri, Senior Fellow in the Institute of Translational Health Sciences at the University of Washington. Hossein uses data science approaches to study urban energy and health.

Workflow



This research analyzes the household location choices of American households in the largest 50 metropolitan areas in the United States. Households are broken down by five-year age cohorts (based on the age of the head of the householder) and mapped against the household's distance (census block group level) from the central business district (CBD) of

the metropolitan area in which they reside. In polycentric regions such as Seattle (Tacoma, Bellevue and Everett as alternative CBDs, analyses are conducted on distance to core center as well as secondary centers. An initial paper reporting the results is currently under review.

All data, code and analytical workflow are hosted on-line. Code and analytical workflow, including analytical script and custom function sets, are written in R and found on the project's GitHub Repository. The complete set of raw data is available through the U.S. Census. Users wishing to skip the data compiling and/or cleaning steps can download the compiled or cleaned data from the project's Dataverse Repository.

The *hhLocAnalysis.R* file is the main analysis script and the only file that needs to be executed. Two key path parameters and two key process parameters must be manually set at the beginning of the *hhLocAnalysis.R* script:

1. **codeDir**: Location of the cloned GitHub code repository
2. **dataDir**: Location of the compiled (and/or cleaned) data downloaded from Dataverse
3. **reBuildData**: Do you want to go through the entire data compilation process?
4. **reCleanData**: Do you want to re-clean data?

Additional parameters containing the file names of the downloaded intermediate data and the path to export the results may need to be set also be set prior to executing the script:

1. **rawDataFile**: (Optional). If **reBuildData** is equal to FALSE and **reCleanData** is equal to TRUE then you will need to provide the name of the compiled data file (within the **dataDir**) downloaded from Dataverse.
2. **cleanDataFile**: (Optional). If both **reBuildData** and **reCleanData** are FALSE then you will need to provide the name of the cleaned data file (within the **dataDir**) downloaded from Dataverse.
3. **figurePath**: (Optional) If you intended to output the plots enter directory to export to

This is the extent of manual operations. All other processes run automatically. If the data is fully built (**reBuildData** = TRUE and **reCleanData** = TRUE) this process may take multiple hours. Additionally, the user may change a number of the optional parameters that handle the distance scaling, overall number of metro-regions to analyze, maximum distance from central business district centroid to include in the data and whether or not computational progress is reported.

Stage 1: Data Collection

Based on the parameters selected above the data collection phase of the study either downloads the compiled (**reBuildData**=FALSE and **reCleanData**=TRUE) or cleaned data (**reBuildData**=FALSE and **reCleanData**=FALSE) from the Dataverse repository or compiles all of the raw data directly(**reBuildData**=TRUE). To compile the raw data, files for every county in the fifty largest metropolitan areas are downloaded, unzipped, cleaned and written out as a standardized .csv (comma-separated value) file. This raw data is hosted on the U.S. Census Bureau's FTP site. Custom functions to handle the data acquisition process were written in R and are found in the *buildHHData.R* and *buildCBSAData.R* files in the repository.

Stage 2: Data Processing

If the data is to be recleaned, then the data cleaning functions are employed at this step. In this process observations with missing data are removed and information on the core-based statistical areas (CBSAs) are added to the compiled data. If cleaned data is directly downloaded then this pre-cleaned data is passed forward to the analysis stage.

Stage 3: Data Analysis

The analytical process begins by calculating the location quotient distance profiles. Location quotient profiles measures the proportion of a given household type at a location versus the proportion of that household type in the entire metro region. Location quotients higher than 1 indicate that, relatively speaking, more of a given household exist at a given location then would be expected if households were randomly distributed. The *hhLocFunctions.R* file contains all of the custom functions necessary to calculate and visualize the location quotient results.

Data visualization of the results via a variety of different plotting functions follows. Final results, both tabular and visual, are then combined in an RStudio/Knitr file along with the narrative to create the final document (compiled in LaTeX). The full data provenance is described and hosted on the code repository via a Markdown file. Also note that the collaborative website Authorea (which offers git-based tracking and LaTeX support) was used by the authors to write the first draft of the narrative portion of the report.

Pain points

There are two major steps that we consider particularly painful. The first is convincing yourself (and co-authors) to take the time to properly document every action and to take the time to fully annotate the analytical workflow. This can be especially difficult when deadlines arise or when co-authors do not see the value in reproducibility. The second is the need to write custom functions that are generalizable. Writing very specific, single use functions can be easy, but are rarely useful in more than a single instance. Good reproducible research contains flexible functions than can accommodate changes or permutations thereby allowing subsequent users to expand or change your original analysis.

The current peer-review process also presents a considerable hurdle to reproducibility. In order to remain anonymous in the review process, we've had to build a set of anonymous code and data repositories and interactive websites for the review process and then switch over to our own repositories after the paper has been accepted. It means a lot of extra work as well as remembering which GitHub account we are signed into at all times. Along this line, judging by usage statistics, reviewers have been uninterested in actually examining the hosted code, data or results.

Key benefits

For us the biggest benefit is efficiency. The first time we do an analysis it usually takes longer than it would take other colleagues, but each time after the time savings multiply. One situation where this is particularly helpful is in responding to peer reviewer comments and requests. Changes to assumptions or sensitivity tests on parameters can be done in a matter of hours (or minutes), not days or weeks. This greatly shortens the re-submittal response time. Related, we constantly find ourselves borrowing old code and re-purposing it, making new analyses easier and faster.

(Andy) Better organization is another benefit. No more folders full of data files with version names and dates. No more mystery fields in a dataset. No more starting all over after forgetting what was previously done. My students and their Excel sheets with dozens of tabs and screen clips from SPSS (or other point and click-based statistical software) remind me of this benefit every semester. I am slowly incorporating more and more reproducibility into my classes, with the intent of breaking some of these bad habits that students have.

(Hossein) Another benefit would be built greater capacity for related research. Beyond the theoretical approach that can be used to study other metropolitan patterns, functions that we built in this research can be applied to facilitate other forms of research using census data. Researchers can adapt these functions to address other purposes. In an ideal scientific world where all research is reproducible, research will be more efficient because of the code that can be shared, re-used, or adapted for research or non-research purposes.

Key tools

The RStudio integrated development environment (IDE) and their related Shiny Apps (interactive web applications) have been a huge help in our reproducible research. If you are an R programmer and want to share your visualizations with non-programmers, we highly recommend these tools from RStudio. Using the IDE allows for easier navigation between multiple scripts, reviewing a history of plots and offering a view of all objects in the current computing environment.

Questions

What does "reproducibility" mean to you?

"Reproducibility" means that a subsequent interested party can openly access the data, code, analytical workflow and data provenance to re-create the research (and ideally produce identical results) WITHOUT consulting the original researcher(s). In this context, "reproducibility" can facilitate the verification of results from a given research project and also accelerate new research discoveries by providing reproducible modules that can be applied in other settings and/or for other purposes.

Why do you think that reproducibility in your domain is important?

(Andy) A majority of quantitative analyses in real estate (both academic and professional) is usually duplicated by numerous parties, widely disseminated and frequently updated; all characteristics that benefit from reproducible analyses. Despite these core facts of the discipline, there is very little, if any, discussion on or attempt to create reproducible research in the field.

(Hossein) In general, the importance of reproducibility in policy-/decision-oriented fields is not clear. It can certainly improve policy research, but one could debate whether or not reproducibility has direct benefits for decision-making.

How or where did you learn about reproducibility?

(Andy) My pre-academic background was as part of a team of expert witnesses in litigation support. In this industry, any analysis that was produced had to be reproducible by the opposition and therefore, my firm was constantly striving to produce more efficiency in their reproducible analyses.

What do you see as the major challenges to doing reproducible research in your domain, and do you have any suggestions?

(Andy) Two challenges exists in the real estate field. First, most data is proprietary and expensive and therefore it is hard to share data. Second, it is a small field that is composed of many senior individuals (both in academia and industry), many of whom are very resistant to change.

(Hossein) In health sciences the biggest concern is around data privacy. For example, research on individual-level patient information can hardly become fully reproducible, within conventional workflows.

What do you view as the major incentives for doing reproducible research?

Doing reproducible research is like installing solar panels in your home. It will cost you at the beginning, but down the road you will get benefits such as time savings, better quality output and the increased opportunity to collaborate/share ideas.

Are there any best practices that you'd recommend for researchers in your field?

No more manual data cleaning. Use code.

Would you recommend any specific resources for learning more about reproducibility?

For collaboration, if you want to get away from writing in LaTeX, you can try Authorea. If you are in Australia, the University of Melbourne's Research Platforms group offers a number of Research Bazaars, Software Carpentry and Reproducibility-related courses and event. It is open to researchers world-wide.

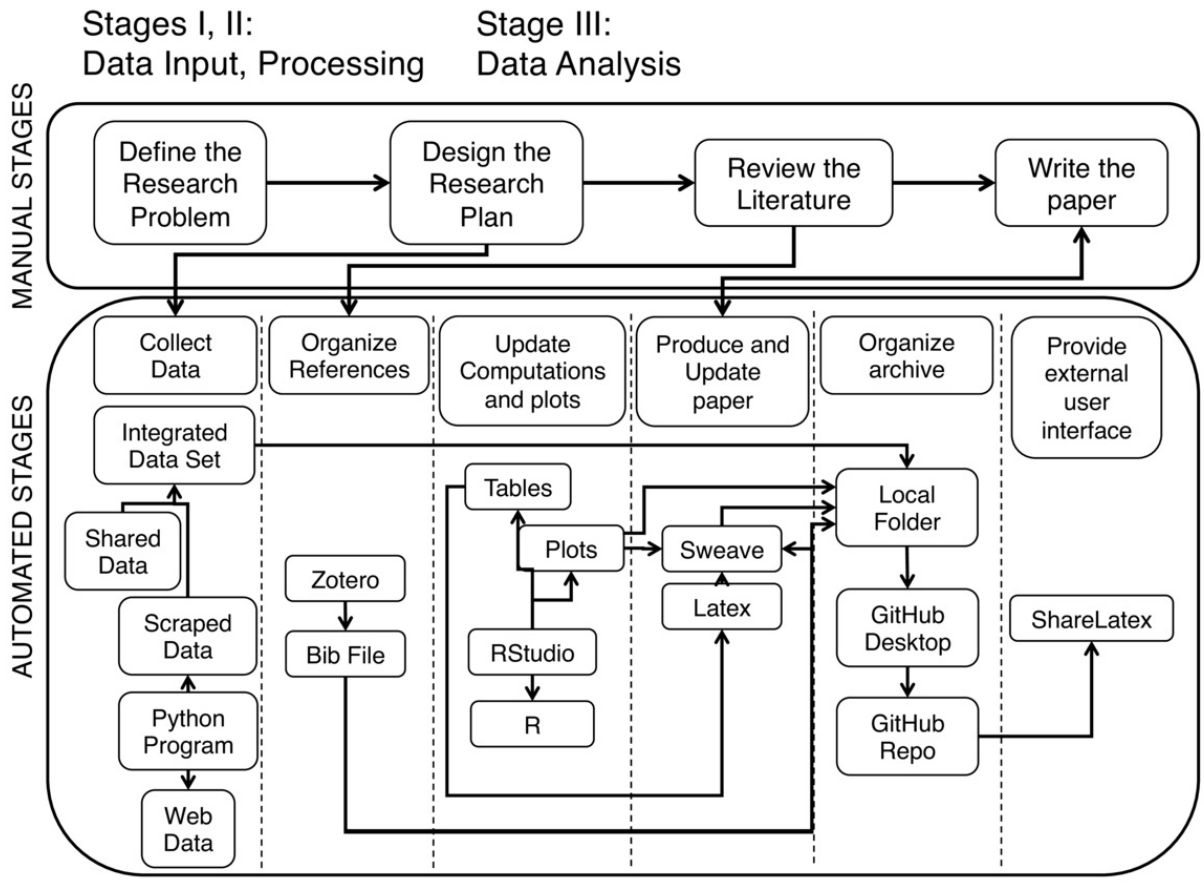
Analyzing Cosponsorship Data to Detect Networking Patterns in Peruvian Legislators

José Manuel Magallanes

My name is José Manuel Magallanes, I am a Senior Data Science Fellow at the eScience Institute of the University of Washington, where I am also a Visiting Professor at the Evans School of Public Policy and Governance (2015 - 2017). Since 2003, I have been Professor of Political Science and Public Policy Methodology at the Catholic University of Peru. My research is related to framing political and policy problems with a computational social science approach. I have dealt with different topics including electoral behavior, public management performance, climate change and social conflict, and Congress's legislators behavior. My contribution for this case will be a research carried out on bill cosponsorship data to detect key players, reveal association patterns, anticipate party splitting and detect tactics to get re-elected.

I have a BSc in Computer Science (UNMSM - Peru), a MA in Political Science and Public Management (PUCP - Peru), a Phd in Psychology (UNMSM - Peru) and a Phd in Computational Social Science (George Mason University-USA).

Workflow



The workflow above represents:

a. Manual Stages:

```
1.  Define the Research Problem.

2.  Design the Research Plan.

3.  Review the Literature.

4.  Paper writing.
```

b. Automated Stages:

```
1.  Collect Data.

2.  Organize References.

3.  Update computations and plots.

4.  Produce and update Paper.

5.  Organize archive.

6.  Provide external user interface.
```

A. Manual Stages

a1. **Defining the research problem.** There was an interest in the political science community in Peru to learn more on the dynamics of their National Congress. There were some particular facts that had been discussed in academia and in the media that called our attention:

- The low re-election rate of legislators the previous two elections (20%).
- The fact that some legislators migrated to other parties during their mandate (party switching).
- The fact that parties that have a good share of seats, end up splitting their seats (It is worth keeping in mind that Peru has a multiparty system).

The fact that some scholars in the USA were using bill cosponsorship data as a proxy to understand some of these issues motivated me to follow a similar approach.

This stage was done only once. I was the only one in charge. No particular tools were used in this stage.

a2. **Designing the Research plan**. This stage identified the main authors that have worked similar research problems before. The key ingredient in all cases was bill cosponsorship. However, most hypotheses were not the same I had, due to the different political regimes that researchers were focusing on. But, in all cases, bill cosponsorship was considered a good proxy to understand legislator's associative patterns. From this stage it was clear that:

- We will need to write code to extract the information from the Congress of Peru website, as the data is not available for download by any means. This process, also known as *web scraping*, is in charge getting poorly structured data and make it available for further computational or statistical analysis.
- To test the hypothesis, the information of a complete Congress will be needed (five years).

- The information from the bill cosponsorship will need to be complemented with the Archives of the National Jury of Elections. There, personal information on every legislator is available. This information was shared, so no code was need to collect it.
- There will be a need for graph or social network techniques.
- The budget available will require the use of free tools.
- There will be a need to share the findings with other scholars.
- This research could be combined with other efforts in another similar countries. So there was a need to organize efficiently the process so that the data and the code could be reused.

This stage was done only once. I was the only one in charge. No particular tools were used in this stage.

a3. **Review of the literature**. This step allowed me to identify similar cases and organize my basic set of references. The references were continuously updated along the process. I was in charge of updating, but also got some recommendations from the users I shared my drafts with.

a4. **Write the paper**. As expected, this was a manual step. However, as I describe later, this was supported by different tools. As usual, this step was repeated many times. I was the only contributor.

B. Automated Stages

b1. **Collect Data**. Data collection had two main sources:

1. The Congress webpage. Here, the data had the key information to build the network.
2. The INFOGOB webpage. This webpage provided the information needed to organize some attributes of the legislators.

The INFOGOB webpage is organized in such a way that you can download information for different processes. It also helped me get the political history of every legislator.

The webpage of the Congress was much different. The information needed is visible as webpages, but they do not offer a download service or a mechanism to get the data (known as API - Application Program Interface) in a structured way. For this reason, a code for scraping the website was needed. The code was written in Python, relying mainly on the *beautiful soup* package. There were some extra code to clean the values collected.

So, with INFOGOB, I built the attributes of the legislators; and, with the scraped data, I built the network. Both data merged using Python to explore the network and the node attributes using Python's *Networkx*. The merged file was saved as a GraphMl file and also as a two separate file of edges and nodes, which will ease exporting into R.

This process was done entirely by me. It was the first part of the operational research and took around two weeks. The Python version I used was 2.7, and it was installed via Anaconda. I used the Spyder-app graphical user interface (GUI) to do the coding.

b2. **Organize references**. References are a key component of academic writing. In my case, besides papers and books, there was also the need to include webpages, white papers, code, data, and so on. As it common, there are set of references you know you would use when you start writing, but more come along the process as you exchange ideas with colleagues. In this particular aspect, the use of **Zotero** was very important. It allowed to create a BibTex file to be used later during the paper production process. This text will later be integrated into the LatTex document of this work. Automating this process not only helps you recover the right of citing a work, but also gives you the flexibility to later change the style (APA, Chicago, etc - see citation list) a particular publisher will require. This was extremely important as this research could be presented in social sciences or computer-science-related conferences.

This process was done entirely by me. This was a continuous process as the paper was written. The desktop version of Zotero was used. The BibTex was saved in the working folder.

b3. **Update computations and plots**. While data collection and structured datasets were produced in Python, the exploration of the data, the test of hypotheses, and the visualization of results was done in R. I decided to use R for a simple reason: RStudio can combine LaTeX and R in an easier way than Python via its *sweave* library. Sweave differentiates between text and R code; codes are organized in *chunks* that also can interact with the LaTeX code.

This eased the update of the tables and plots produced by the data, as sweave documents will rerun the R code and update whatever is needed. This was a crucial part to make this work more reproducible; and also for me, as I could try different *layouts* for the network plot and pay closer attention to final appearance of the paper.

This process was done entirely by me. This was a repetitive process as the paper was written. A Rnw file was produced using RStudio, which also produced a LaTeX file.

b4. **Produce and update Paper**. RStudio integrated my writing, the bibliography file, and the tables and plots generated using R into a LaTeX document, which finally will produce a pdf document (RStudio, via *knitr*, instead of *sweave*, can produce also an html document) .

Any change in whatever part of the main document or any of the files used was updated in the final product automatically.

It is very important to keep in mind that sweave allows LaTeX users to customize all the details in the document, which includes code highlighting or hiding, among other possibilities. I could even present the Python code inside the document as needed.

This process was done entirely by me. This was a continuous process as the paper was written. And in fact, there were many versions that I could share with colleagues. LaTeX complied the R chunks producing tables and plots, and compiled the bibliography into the main document from the BibTex file generated in Zotero.

b5. **Organize archive**. One of the first steps after the research questions were clear, and before any coding was made, was the creation of a GitHub private repository. This repository was cloned into my laptop, and all the files were organized in this folder, including code, data files, bibliography files and plots. In a way, using a repository that will be online forces you to organize your work and folders since the beginning. Before becoming a GitHub user, preparing the final version of my work took too much time; a good planing when using GitHub will force you to your system of folders ready when you are done with the paper. An additional advantage is the version control power you have when using GutHub, which I had to use just one time, to recover a version that had a code that produced a better plot than one I thought was going to work better. Without it, you need to be commenting and uncommenting code sections which increases, unnecessarily, the coding space.

This process was done entirely by me. This was a continuous process as the paper was written. The GitHub client was used for committing and synchronizing the local repository into GitHub.

b6. **Provide external user interface**. It was clear during the planing stages that I will need to share my drafts with other colleagues in order to get some feedback and/or discuss further collaboration on this matter. As the paper reflected an step-by-step approach, it would be easier for my colleagues to read the draft paper which included the code chunks, the plot and the tables. For that, I decided to use ShareLaTeX, which can collect the files in the GitHub repository and compile the LaTeX document. So, after I updated the GitHub with my last version, I could also ask ShareLaTeX to update its contents based on the latest document version I had recently pushed into GitHub.

This process was done entirely by me. However, the drafts were shared when most of the processing was finished. This was a continuous process as the paper was written. The selected users created ShareLaTeX accounts to see the LaTeX generated pdf version of my document. I allowed them to write comments in the LaTeX document using ShareLaTeX itself.

C. On the Data, Software and Processing

- **Data:** The raw data as well as the cleaned and aggregated data are online, in a private GitHub repo. The data can be share upon request and instructions are included on how to cite it. The data files have a table-like structure to be easily read into R, but other versions were produced in xml-like format as I thought I may need to use other network visualization programs like Gephi.
- **Software:** The Python code is also in the repository. The R code is embedded in the LaTeX code, and the paper itself describes the algorithms adopted in the paper. Most R chunks make constant use of the data scraped using Python. For GitHub, ShareLaTeX and Zotero, you only need to create and account and download the desktop version.
- **Processing:** The processing of the data is reflected in the Python code flow, and it is online. The Python and R codes are commented extensively. It would be fairly easy for an external researcher to follow the logic of the research and replicate the results, or simply change the data from other country and get all the tables and plots in the final PDF, as R, Python and LaTeX are connected.

Pain points

This work was not producing a blog or a notebook, but a paper. So the most challenging parts were:

1. Produce a quality layout where tables and plots are located in the right place is hard. LaTeX is not exactly what you see is what you get, so you need to learn how to override some default behavior in LaTeX for that.
2. You can become too excited as you learn to use LaTeX, so you start thinking all the time to make it better, and it takes too much extra time because you need to include more LaTeX functions and need to learn how to configure them. It is better to do that after the paper is done.
3. Scraping several webpages takes time, and you learn that your code may only be usable for those particular websites. I scraped many pages, but all came from the same institution, so a project that involves scraping from more than one institution will deal with much more complexity.
4. A particular pain point is the lack of a reproducibility culture in the field I work. Political scientists in my country are not used to reproducible research. In fact, for every key paper that dealt with the kind of data I used, no further instructions were found from the authors or in the authors' webpages. In most cases, it is only mentioned what data was used but no links or other related procedures were clear.

Key benefits

I consider the way I worked allowed me to obtain several benefits:

1. Planing your research in a reproducible way is a great advantage to the scientific community you belong to. But most of all, it forces you to plan your work better.
2. Including version control forces you to have well organized set of folders in your machine.
3. Following a reproducibility approach will allow you escalate your work if more data becomes available or if a colleague wants to make a comparative work. I am sure this is not impossible without this approach, but I am sure that researchers can become much more productive than in the past.
4. Another important benefit is that allowing colleagues to audit your work gives you enough input to make a newer and more robust version of your work.
5. You have the possibility to produce plots with different levels of quality. R allows you to produce simple quality plots and more complex formats. In this case, I was requested a higher resolution of a plot in vectorized format, and I simply recreate the one I had, changing a couple of parameters.

Key tools

LaTeX was a key component in all this research and its reproducibility level. It offers a way to organize the paper and interact with code and data files, including references and plots. This can not be done using Word, as far as I know. LaTeX is not a common software in social scientists in my country. RStudio is also a key ingredient. Its capacity to transform the R chunks and its output (including tables, values and plots) into LaTeX makes the flow and update of research even better. Both LaTeX and RStudio facilitate reproducibility. Without R, the barrier for producing papers is even higher, but it can be done. It gives you more confidence and save you lots of time to update/edit your manuscript, compare to copy, past or inserting procedures in MsWord. The flow is simply great.

Questions

What does "reproducibility" mean to you?

In general, I consider this term means the level of reconstruction of a research that can be achieved by a person foreign to the researcher / research team via the code and data available in some repository. For me, reproducibility is not only that the foreign person can

decompress and run and executable file to see the results, but be able to audit the whole process. The less feedback required by the auditor, the more reproducible a work is.

Why do you think that reproducibility in your domain is important?

Because computational social science is still young in many other countries. As in my case, data is just starting to become available, so following, and teaching, the reproducibility approach will benefit the research quality. In public policy, particularly, it will enable stakeholders participation in knowledge creation.

How or where did you learn about reproducibility?

I had no chance to have mentor or courses on this. I just felt the need to organize my work as many tools and data were available for my case. I was afraid that if I did not follow this approach I could easily get lost. Reproducibility demands good research planing, and it pays off.

What do you see as the major challenges to doing reproducible research in your domain, and do you have any suggestions?

If the data you are using is public, I see *less* problem. When the data is not, and you get access with a special permission, legal issues are always present. As for investment, my particular collection of tools are free, so it should not be a problem, unless your funding institution forces you to use particular tools. I also believe that this approach can be very challenging for older generations not used to this. I see less of a problem in younger generations of researchers.

What do you view as the major incentives for doing reproducible research?

That it allows you to improve your work.

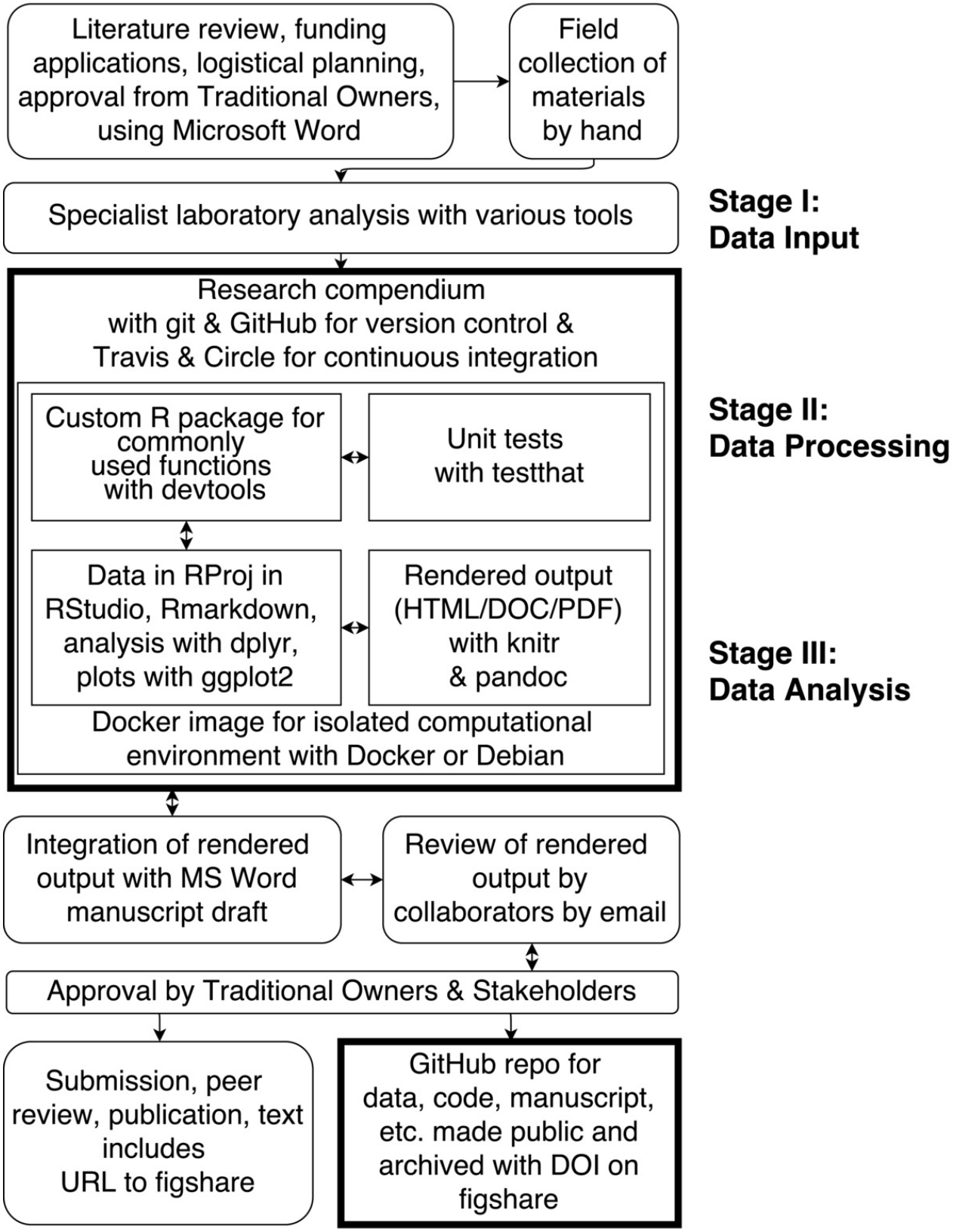
Using R and Related Tools for Reproducible Research in Archaeology

Ben Marwick

My name is Ben Marwick, and I am an Associate Professor of archaeology in the Department of Anthropology at the University of Washington, and a Senior Research Scientist at the University of Wollongong. My research interests include human-environment adaptations during the Pleistocene in Southeast Asia and Australia. My colleagues and I work with stone artefacts, organic and geological remains to understand past human behaviours and their environmental contexts. My narrative here describes one recent project from the initial concept to a specific publication (Clarkson et al. 2015), but the details below are typical of my experience with several projects focused on understanding prehistoric hunter-gatherer behaviour (cf. Marwick et al. 2016; 2017).

In the context of this case study, reproducibility refers to computational reproducibility, which means enabling other researchers and students to combine the code and data that we produce to obtain the same statistical results and data visualizations that we present in our publication. I also expect that the code could be used for empirical reproducibility, where our code is applied to a new dataset to generate substantively similar results to our published results. I have explored these definitions, and the principles that motivate my selection of tools, in more detail in Marwick (2016.)

Workflow



The boxes with a bold outline indicate key steps and tools that enabled computational reproducibility in our project.

One recent project I was involved in aimed to excavate Madjebebe rockshelter, a well-known archaeological site in northern Australia. The purpose of the excavations was to test the findings of previous excavations in the 1990s that uncovered controversial early evidence for

human occupation of Australia. The project was initiated with consultation with stakeholders, including Aboriginal traditional owners of the archaeological site, and a grant application written in Microsoft Word and circulated among the team by email.

The archaeological excavation was conducted with standard modern techniques. These include a combination of direct digital capture of artefact and feature provenance with a total station, digital photography and GIS, and hand-written paper notes using structured site recording forms. These data from these forms was later entered into an Excel spreadsheet.

At the conclusion of fieldwork, post-excavation analysis continued at the home institutions of each of the team members. The tools for data collections and analysis at this stage were according to the norms of each lab, but the final products from most of the team members at this stage were MS Word and Excel files. At this point, work began on a manuscript for publication, which was a MS Word document that was circulated among the authors by email.

As the specialist work concluded, the Excel and Word files were collected into an R Project using RStudio. The spreadsheets were converted to CSV files to ensure they could be accessed independent of any specific software. A research compendium was created, based on a custom R package, following the examples described in rrrpkg. This package was written to contain custom functions used often in the analysis. The devtools package was used to develop the custom R package in RStudio. The testthat package was used to write tests to ensure the package functions performed as expected while they were being developed. An R markdown file was created as part of the compendium, and edited in RStudio to recompute and extend the analysis and visualizations from the specialist labs, and combine the key pieces of narrative text from the lab reports that contain statistical results. The R markdown file is a kind of lab note book where code and text are interwoven in a single document. It summarizes and extends the work of the team specialists using R script. The code in the R markdown file used several R packages, including dplyr and reshape2 for data cleaning and analysis, rioja and analogue for specialist environmental methods, and ggplot2 for visualization. The runtimes of the analyses are rarely longer than 30 min, so writing code and narrative, and testing are the most time consuming tasks here.

The R package knitr and the pandoc program (included with RStudio) was used to execute the R markdown file to inspect the output as the code was being written. A Docker container was created to create an isolated computational and portable environment for writing the R markdown document and developing the package. The Docker image was backed up on the Docker Hub server (https://registry.hub.docker.com/u/benmarwick/mjb1989excavationpaper/), and tested using continuous integration from CircleCI. All of these components, data files, R markdown file, package files, etc. were all version controlled using git locally and backed-up on a repository at GitHub. The GitHub repository is here https://github.com/benmarwick/1989-excavation-

report-Madjebebe, and a snapshot of this repository at the time of acceptance of our 2015 *Journal of Human Evolution* paper is archived on figshare here: http://dx.doi.org/10.6084/m9.figshare.1297059. One of the downsides of using this compendium approach is that most of the work is done by just a few of the team members because not everyone is familiar with (or interested in) the tools.

While the analysis was being developed in the research compendium, a manuscript was being drafted in a MS Word document and circulated among the authors by email, and revised using track changes. The rendered output of R markdown document is also circulated among the authors by email. As the manuscript is updated, and new ideas are incorporated into the analysis, additional code is written, some code abandoned, new plots produced, and others deleted, etc. This is probably the messiest and least ideal part of the workflow as it involves manual updating of the MS Word document with new values and figures from the rendered R markdown document, and two unrelated version control systems (git and track-changes in MS Word). The non-linearity of the process was also challenging, as the authors negotiated how the manuscript and analysis should take shape.

As the review and updating cycle concluded, the manuscript was sent for review by the traditional owners of the land where the archaeological site is located. After this review, which involves some changes to the manuscript, the final draft was prepared for submission. At the same time, the GitHub repository that contains the research compendium was made public and continuous integration from Travis was added to monitor the effect of changes made during peer review. The compendium was also deposited at figshare and the persistent URL to the figshare repository was added to the text of the manuscript as a pointer to the data and code that generated the results and visualizations found in the paper. The MIT license was attached to the code (giving others permission to use and reuse the code), the CC0 license was attached to the data (meaning that the data are in the public domain), and a CC-BY license was attached to the text and figures (meaning that the text is free to use with proper attribution to the original authors). These licenses allow flexible reuse of our materials. The paper was then submitted for publication at the *Journal of Human Evolution*. At this point the data and software were openly available online for peer reviewers and others to inspect. The code includes the R package, which has documentation about installing the packages and using the functions, has unit tests, and has machine- and human- readable metadata about dependencies. We have also made available the Docker image that contains the compendium in an Linux environment so that all the dependencies external to R can be included in a single bundle.

Pain points

Some of the most notable pain points include:

- the inefficiencies of duplication of effort in translating the Excel-based analysis into R, and in moving between MS Word and R markdown for drafting the text. This happens because only a few members of the team are familiar with R and related command-line tools.
- the complexities of working on the draft manuscript and updating the analysis as the team explores different options and research directions. This challenges are typical of any large collaborative project, but I think multiplied here because of the 'two universes' of toolkits, with some of the team using Microsoft tools, and others using open source tools. Because of the greater flexibility and efficiency of R over spreadsheets for data analysis, we observed a disempowering of team members who are not familiar with R.
- overall, the research compendium is still quite a complex arrangement of tools and scripts, and productive engagement with it requires a high degree of enthusiasm and a high tolerance for trouble-shooting. This is a barrier for collaborators who don't share my interest in reproducibility. However, I'm optimistic that making and using research compendia will become simpler and more normal, and increasing awareness about reproducibility will motivate researchers to take a greater interest in incorporating these practices into their own work.

Key benefits

Some of the advantages that motivated us to pursue that approach include:

- A detailed human- and machine-readable record of all the steps in the analysis. This takes the methods out of Microsoft Excel, where they are often invisible due to ephemeral point-and-click behaviours, and reconstructs them in R scripts where every step is explicit. This makes it a lot easier to engage with questions like "can we do that again, but change X a little bit?" and "what happens if we add/exclude Y from the analysis?" This kind of exploratory work is most efficiently done using a scripting language because the equivalent work in a spreadsheet often requires redoing numerous manual steps of data manipulation simply to alter one small step in the analysis pipeline.
- An open and transparent record of the analysis for reviewers to inspect, this allows us to say 'we have nothing to hide', and the git repository allows us to show 'we already tried that, and this is what we got' because we have a complete history of our analytical efforts, even those that didn't lead to results included in the publication.
- We have a high degree of confidence that our results are correct. We can rerun the analyses repeatedly in an isolated and well-defined environment and get the same result each time.

- Our data and methods are available for reuse and application to new projects and contexts by us, and by other researchers and students. This saves time for us in the future, and has the potential to increase the impact of our work.
- The uniqueness of our workflow is a double-edged sword because it attracts attention to our project because of its exoticness, but because it's so unfamiliar few people can engage with it or use it in the ways we're hoping. As I developed this workflow I was worried it might be a once-off effort, and that it wouldn't be suitable or sustainable for future projects. Since that time, I've found the opposite to be true -- I've used a similar R-package-as-research-compendium approach as I've described here for subsequent scholarly publications (e.g. Marwick et al. 2016; Marwick et al. 2017). In evolving and simplifying this workflow I've enjoyed substantial gains in efficiency. I've also received a lot of interest in this approach from other groups outside of my discipline who are keen to adopt these practices to improve the reproducibility of their research.

Key tools

The key specialized tool that enhanced the reproducibility of our research is R, and the suite of user-contributed packages that extend its functionality. Many of these add several idioms that greatly improve the ease of use of R, such as dplyr, ggplot2, and knitr. The RStudio program was used to develop the code because it has many built-in conveniences that lower the cognitive burden of package development and coding. Although git and GitHub are not specific to R, use of git is deeply integrated into RStudio, so we consider it part of the R ecosystem. Similarly, Pandoc is a universal document format converter that is not unique to R, but since it is also built into RStudio we consider it part of the ecosystem also.

In addition to R and its ecosystem, we used several popular software engineering tools to help with quality control. These include Docker, a system for lightweight virtual environments (and boot2docker, which enables Docker on Windows and OSX), Travis, which builds and checks our R package each time a commit is made to the GitHub repository, and CircleCI, which is a similar service to build the Docker image and run some simple tests each time a commit is made that changes the dockerfile. We also used these services to render our R markdown documents each time a commit was make, to check that no errors had been accidentally introduced.

While R by itself ('base' R, without contributed packages) is familiar to many social scientists, the packages noted above that introduce powerful modern idioms are less well known. The broader R ecosystem and software engineering tools we used are almost totally unfamiliar to our peers, despite their ubiquity in the software development community. So we see a lot of potential for these tools to be of broader interest, ideally because of the reproducibility they efficiently enable, but likely also because of their novelty.

Questions

Why do you think that reproducibility in your domain is important?

Reproducibility is important because many important steps in our data analysis occur on the researchers' computers, but these steps are often not documented in a way that we can easily access, archive, and communicate with others. The use of software operated by a point-and-click interface is the key problem here. By changing the key analytical tool to a scripting language such as R, we change the nature of our computational work from closed and ephemeral, to open, reusable, and enduring. This makes it a lot easier to show what we've done, why we think the results we present are correct, and enable us and others to reuse and extend our work. These are fundamental for the advancement of science, and with improved reproducibility in our research, we can advance science faster and more reliably.

How or where did you learn about reproducibility?

Most of the reproducible practices in our project were self-taught by a few members in our team adopting practices they've observed in elsewhere, such as ecology and biology. Key resources in this self-teaching include Software Carpentry teaching materials, materials produced by rOpenSci, and instructive scholarly publications and blog posts with code examples, and GitHub repositories written by researchers in other fields who are highly progressive in enabling reproducible research. These include Carl Boettiger, Jenny Bryan, Rich FitzJohn, Karl Broman, and others in the rOpenSci community. Many of the idioms that greatly simplify using R for archaeological data analysis have been contributed by Hadley Wickham and his collaborators on the 'tidyverse' set of packages. The R community on StackOverflow is a great resource because of their strong emphasis on including reproducible examples in questions posted to the site. Many of the questions and problems I encounter have already been answered in several different ways on StackOverflow, often by highly skilled programmers.

What do you see as the major challenges to doing reproducible research in your domain, and do you have any suggestions?

In my field, where the datasets are usually not problematically large and compute times are not inconveniently long, I consider that most of the technology barriers are sufficiently low to be considered solved. The tools are stable, well-documented, and widely used in other domains, so I don't see any major technical challenges. The key challenge is human - not

everyone in the team has the skills to use the tools that enable reproducible research, and not everyone has the motivation and opportunity to learn. This contributes to the primary logistical challenge, which is the manual integration of project components using traditional low-reproducibility tools and the components that enable high-reproducibility. My suggestions are to ride the wave of generational change, and teach students and early career researchers about reproducible research as a normal part of doing research. This means teaching them to expect that analyses should be done with a scripting language (rather than point and click), and that code and data from other researchers should be openly available for inspection (rather then 'by request', which when requests are made, are often refused or ignored). This is the long game, waiting for generational change, but I think will be more effective than efforts full of sound and fury to change the entrenched behaviours of senior colleagues, who rarely have the time or inclination to learn new tools and workflows.

What do you view as the major incentives for doing reproducible research?

The major incentives are:

- increasing the certainly of the correctness of our results
- increasing the ease of tracking our analysis, and exploring new options
- increasing the impact of our work by increasing the ability of, and likelihood that, other researchers will use our methods, data and results.

Are there any best practices that you'd recommend for researchers in your field?

The generic practices I'd recommend for researchers in my field include:

- making raw data openly available in trustworthy repositories in open formats at the time of publication
- using scripts written in a widely used open source programming language to analyze the data
- making the scripts openly available in trustworthy repositories so that they can be used with the data to generate the figures and statistical results in the publication.

Would you recommend any specific resources for learning more about reproducibility?

- Stodden, V and Miguez, S (2014). Best Practices for Computational Science: Software Infrastructure and Environments for Reproducible and Extensible Research. Journal of Open Research Software 2(1):e21, DOI: http://dx.doi.org/10.5334/jors.ay
- Gandrud, C. (2013). Reproducible Research with R and R Studio. CRC Press. Chicago
- Reproducible Science Curriculum
- Software Carpentry
- Data Carpentry
- rOpenSci Reproducible Science Guide (and see the *further readings*)

References cited

Clarkson, C., Mike Smith, Ben Marwick, Richard Fullagar, Lynley A. Wallis, Patrick Faulkner, Tiina Manne, Elspeth Hayes, Richard G. Roberts, Zenobia Jacobs, Xavier Carah, Kelsey M. Lowe, Jacqueline Matthews, S. Anna Florin 2015 The archaeology, chronology and stratigraphy of Madjedbebe (Malakunanja II): A site in northern Australia with early occupation. *Journal of Human Evolution* Volume 83, June 2015, Pages 46–64 DOI: http://dx.doi.org/10.1016/j.jhevol.2015.03.014

Marwick, Ben, Hannah G. Van Vlack, Cyler Conrad, Rasmi Shoocongdej, Cholawit Thongcharoenchaikit and Seungki Kwak (2017) Adaptations to sea level change and transitions to agriculture at Khao Toh Chong rockshelter, Peninsular Thailand. *Journal of Archaeological Science* 77:94-108. DOI: http://dx.doi.org/10.1016/j.jas.2016.10.010

Marwick, Ben, Chris Clarkson, Sue O'Connor and Sophie Collins (2016) Early modern human lithic technology from Jerimalai, East Timor. *Journal of Human Evolution* 101:45-64. DOI: http://dx.doi.org/10.1016/j.jhevol.2016.09.004.

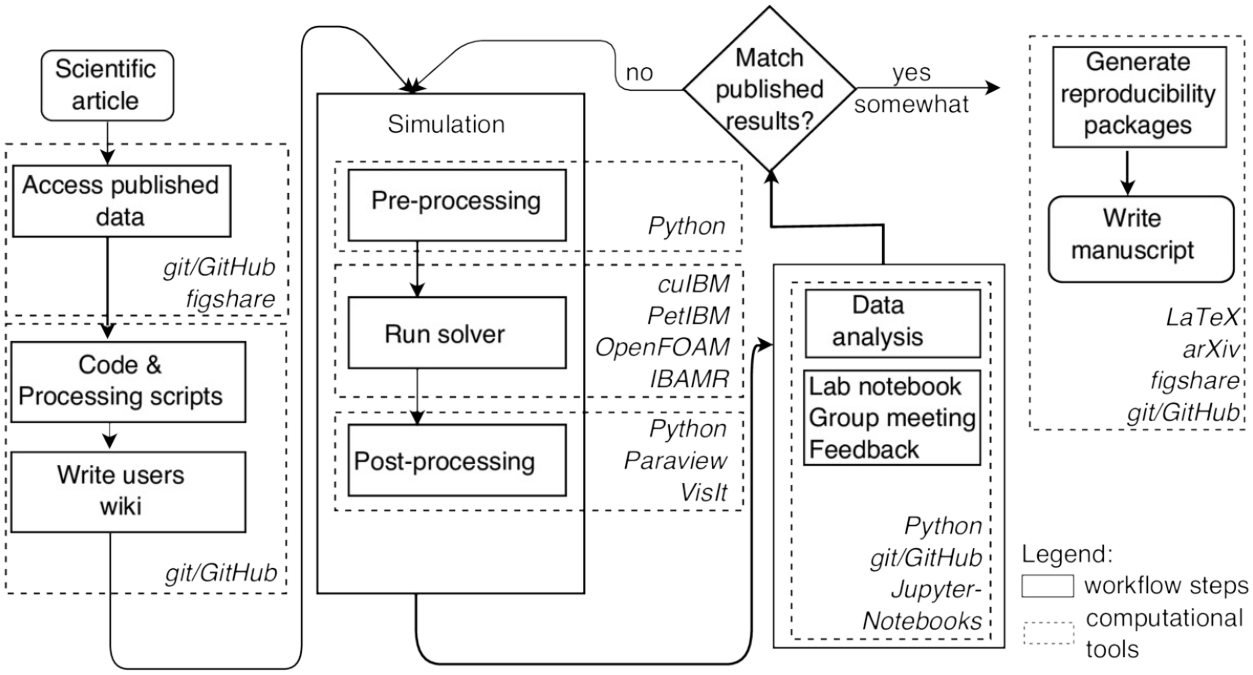
Marwick, Ben (2016). Computational reproducibility in archaeological research: Basic principles and a case study of their implementation. *Journal of Archaeological Method and Theory* 1-27. DOI: http://dx.doi.org/10.1007/s10816-015-9272-9

Achieving Full Replication of our Own Published CFD Results, with Four Different Codes

Olivier Mesnard and Lorena A. Barba

We are members of a computational research group led by Prof. Lorena Barba at the George Washington University in the department of Mechanical and Aerospace Engineering. We do our best to accomplish reproducible research and have for years worked to improve our practices to achieve this goal. According to the "Reproducibility PI Manifesto," pledged by Barba in 2012, all our research code is under version control and open source, our data is open, and we publish open pre-prints of all our publications. For the main results in a paper, we prepare file bundles with input and output data, plotting scripts and figure, and deposit them in the figshare repository. This case study describes what happened when we set out to complete a full replication of published results from our own group, using different Computational Fluid Dynamics (CFD) codes: a new code developed in our group, an open-source code developed by another group, and an open-source CFD library.

Workflow



Our research lab has developed over the years a consistent workflow that, we believe, leads to reproducible research. A previous study coming out of our lab, published in Krishnan et al. (2014), already satisfies the criteria of the "Reproducibility PI Manifesto" (Barba, 2012). That work studied the aerodynamics of flying snakes using our code cuIBM for solving the Navier-Stokes equations with an immersed-boundary method. The crux of the study was that, for a particular configuration, the snake's cross-section experiences a lift-enhancement. Here, we describe our effort to achieve full-replication of the main results, using four different Computational Fluid Dynamics (CFD) codes, including cuIBM. We encountered failures and difficulties, leading to improvements in our workflow and conclusions about the challenges for reproducibility in a scenario of highly unsteady flow dominated by vorticity (local spinning of the flow).

The first code we used to attempt replication is IcoFOAM: the unsteady laminar solver of the well-known CFD package OpenFOAM. We chose OpenFOAM because it is widely used, open-source, and documented: both code documentation and users' guide are available. With unstructured-mesh finite-volume solvers like IcoFOAM, the mesh generation step is most often what determines the quality of the solution, and we experienced that some meshes resulted in unphysical results. Our first tries led to inconsistent results and we had to replace the mesh-generation tool to get acceptable mesh quality. Setting the boundary condition at the domain outlet was particularly problematic, and made more difficult by lack of documentation for the type of boundary condition we needed. We invested several months of persistent efforts before finally replicating our previous findings (in terms of the lift characteristics) with IcoFOAM.

We then used IBAMR, an open-source library hosted on GitHub that provides several numerical methods for immersed bodies. One of them is specifically designed for non-deforming bodies, which is our situation. Bhalla et al. (2013) published a detailed validation of this method, and some examples are included in the code repository. After many failed attempts, we found that this method requires forcing the fluid to rest everywhere *inside* the immersed-body, not just at the boundary—this is not an intuitive option with immersed-boundary methods. In the end, we can say that the *scientific findings* of Krishnan et al. (2014) have been replicated, but we still see noticeable differences in the details of the flow characteristics.

The cuIBM and PetIBM codes are both being developed in our research lab and implement the same immersed-boundary method (Taira & Colonius, 2007). The GitHub code repositories include code documentation with Doxygen, users' documentation (on the GitHub wiki), as well as basic examples and tutorials. cuIBM uses CUSP, an open-source library for sparse linear algebra on a single CUDA-architecture Graphical Processing Unit (GPU). We used cuIBM again to confirm the reproducibility of the published findings in Krishnan et al. (2014). It is important to remark that we had to use the *same version* of the code, with the *same version* of the linear-algebra library to obtain the same numeric answers as before. In fact, our first attempts used a newer version of the CUSP library, and failed to replicate the findings! In PetIBM, we use the PETSc library to solve the linear systems on a distributed-memory machine. Even though the mathematical formulation in cuIBM and PetIBM is exactly the same, we observed that a different linear-algebra library could change the results. As of this writing, we have been unable to replicate with PetIBM the lift-enhancement feature of the flying snake.

The lessons learned from this case study are sobering. First, the vigilant practice of reproducible research must go beyond the open sharing of data and code. We now use Python scripts to automate our workflow—all scripts are version-controlled, code-documented and accept command-line arguments (to avoid code modification from users). Instead of using GUIs, we call the Python interpreter included in the visualization tools Paraview and VisIt to plot the numerical solution. Throughout, Jupyter Notebooks and Markdown files document partial project advances. Second, certain application scenarios pose special challenges. Here, we are working with the Navier-Stokes equations applied to highly unsteady flows dominated by vorticity, a particularly tough application for reproducibility. Third, extra care is needed when using external libraries for iterative solution of linear systems: they may introduce uncertainties.

As we now prepare a manuscript to publish the results of this project, it is being written using LaTeX and version-controlled in its own GitHub repository to facilitate collaboration between authors. To advocate open-science, the manuscript will be first available on arXiv. We will also provide, on the repository figshare, a reproducibility package for all simulations and

figures reported in the manuscript. These packages include the version of the software, the input parameters, information related to machine architecture, and the necessary scripts to run and post-process the simulation.

Pain points

A critical ingredient in a reproducible workflow is keeping a detailed, up-to-date, and version-controlled lab notebook. It is nearly unthinkable that a proper lab notebook for recording computational experiments could be kept without scripting all steps— pre-processing, running, post-processing—and automaticaly saving command-line inputs. In the project of this case study, we used four different CFD codes in batches of simulations spanning many parameter combinations, resulting in hundreds of runs. The run times varied between 1 and 3 days and the numerical solutions each generated between 3.5 and 16 gigabytes of data. Most of the simulations were run remotely on an HPC cluster at the George Washington University, and the solutions were then moved to several different local desktop machines for post-processing and storage. The lab notebook proved to be vital for tracking all simulations and data. Another aspect of this project that was very time consuming was becoming familiar with new software—it took even longer to familiarize ourselves with codes that offer poor users' documentation. Finally, we also spent considerable time developing automated scripts for analyzing the numerical solutions resulting from different codes (producing different output formats). These scripts, however, are essential to deliver reproducible computational experiments.

Key benefits

In the field of computational fluid dynamics, it can easily take six months or a year to develop software from scratch for solving a specific fluid-flow scenario. On publishing the results, if the authors do not release the code and data used for the study, it leaves any reader hoping to reproduce the results facing a steep time investment. Not surprisingly, studies attempting to reproduce previously published findings are rare. As we have illustrated with our campaign to achieve full replication of our own previous study, there are severe pitfalls and challenges in fluid-flow simulations under unsteady, highly vortical regimes. It is a distinct possibility that many published studies report wrong results. As noted by Leek and Peng (2015), increasing the level of reproducibility of published studies will help uncover flawed research findings. For this reason, the minimum level of reproducibility—making code and data available—is essential for increasing the confidence on any new scientific claims to knowledge generated computationally. Going beyond sharing code and data, full automation and digital recording of experimental campaigns offer the best guarantee of being able to extract scientific value from computational experiments.

Key tools

We use the version-control hosting platform GitHub to support our reproducible workflow. GitHub greatly facilitates collaboration when developing numerical codes and documentation. The platform also allows creating wiki pages for users' documentation. We use GitHub to write manuscripts, to record our group-meetings, and to store teaching materials. We also extensively use Python to automate analysis and post-processing. Progress reports and summaries for discussion in group meetings are best presented using Jupyter notebooks, where textual media is combined with code and visualizations. For a digital record of all steps taken in preparing a simulation and running it, bash scripting is essential. We also use Travis CI for running automated testing of the codes whenever a change is to be merged into the main repository.

Questions

What does "reproducibility" mean to you?

The starting point for our understanding of reproducibility is contained in the pledge "Reproducibility PI Manifesto" (Barba, 2012) which includes these steps:

1. teaching group members about reproducibility;
2. maintaining all code and writing under version-control;
3. carrying out verification and validation and publishing the results;
4. for main results in a publication, sharing data, plotting scripts, and figures under CC-BY;
5. uploading preprints to arXiv at the time of submission of a paper;
6. releasing code no later than the time of submission of a paper;
7. adding a "Reproducibility" statement to each publication;
8. keeping an up-to-date web presence.

Some of these items have to do with making our research materials and methods open access and discoverable. The core of this pledge is releasing the code, the data, and the analysis/visualization scripts. Already this can be time consuming and demanding. Yet, we have come to consider these steps the most basic level of reproducible research. On undertaking a full replication study of a previous publication by our research group, we came to realize how much more rigor is required to achieve this, in the context of computational fluid dynamics of unsteady flows. We use the term "full replication" in the sense presented by Peng (2011), that is, completing an independent study using new methods to collect new data, arriving in the end at the same scientific findings. In computational fluid dynamics, full

replication of the findings can involve using a different code that implements the same numerical method, or a code that implements a different numerical method altogether but solves the same mathematical model. Because we are solving the Navier-Stokes equations —an unsteady and nonlinear model—certain problem scenarios can present particular challenges to replication.

Why do you think that reproducibility in your domain is important?

In computational science, we use simulations and data analysis as tools for the creation and justification of scientific knowledge. This process of knowledge creation, as in all science, must also produce evidence to justify itself. Reproducibility is a way to provide grounds for trusting the scientific findings obtained computationally. Ensuring that a publication (along with the data used to generate the figures) is reproducible makes it easier for others to corroborate (or reject) a scientific hypothesis. Codes and data used to publish results should be version-controlled and open-source to facilitate reproducibility. Donoho and co-authors (2009) mentioned that we develop codes so that they can be used again by strangers and defined strangers as "anyone who doesn't possess our current short-term memory" (including ourselves in some years). We believe that reproducible research can also prevent scientists from "reinventing the wheel" by having to re-create complete software stacks to build from previously published work.

How or where did you learn about reproducibility?

The group's PI, Prof. Lorena Barba, plays an active role in raising awareness about reproducible research. Incoming students joining our research lab must start by learning the different tools mentioned in the "Reproducibility PI Manifesto". The Software Carpentry Foundation (through workshops and online ressources) also contributes to educate our group members and improve our workflow to achieve reproducible research.

What do you see as the major challenges to doing reproducible research in your domain, and do you have any suggestions?

Reproducible research can be time-consuming, requiring rigorous methods and organization. At various moments during the project, we had to pause and ask ourself if our research was currently reproducible. Often, this was prompted by a conversation or questioning during group meetings. In that sense, a strong collaborative culture in the

research group, and beyond in the wider community of the discipline, are vital to instill reproducibility practices in computational researchers. Lack of systematic and widespread educational programs that emphasize reproducible research is a serious obstacle.

What do you view as the major incentives for doing reproducible research?

Making your research more reproducible—e.g., providing reproducibility packages along with the manuscript—is a way of showcasing your skills, a medium for communicating research more transparently, and an invitation to give feedback on your work. If the research community is inclined to put more effort in doing reproducible research, it would prevent scientists from reinventing the wheel by rewriting software in order to build from your work. In the long run, it saves resources to achive scientific knowledge growth, both at the level of a community and within a research group.

Are there any best practices that you'd recommend for researchers in your field?

Again, we insist that automating all the computational workflow and diligently maintaining a lab notebook are fundamental to record your research. We try to avoid GUIs as much as possible and prefer to script everything so that analysis can be automated, reproducible, and recorded. This may be time-consuming but surely beneficial in the longer term of a research project.

Would you recommend any specific resources for learning more about reproducibility?

- Barba, L. A. (13 December 2012). "Reproducibility PI Manifesto", 10.6084/m9.figshare.104539. Presentation for a talk given at the ICERM workshop "Reproducibility in Computational and Experimental Mathematics". Published on figshare under CC-BY.
- Donoho, D. L., Maleki, A., Rahman, I. U., Shahram, M., & Stodden, V. (2009). Reproducible research in computational harmonic analysis. Computing in Science & Engineering, 11(1), 8-18.
- Leek, J. T., & Peng, R. D. (2015). Opinion: Reproducible research can still be wrong: Adopting a prevention approach. Proceedings of the National Academy of Sciences, 112(6), 1645-1646.
- Madeyski, L., & Kitchenham, B. A. (2015). Reproducible Research–What, Why and How. Wroclaw University of Technology, PRE W, 8.

- Peng, R. D. (2011). Reproducible research in computational science. Science (New York, Ny), 334(6060), 1226.
- Reproducible Research -- Coursera MOOC.
- Sandve, G. K., Nekrutenko, A., Taylor, J., & Hovig, E. (2013). Ten simple rules for reproducible computational research.
- Software Carpentry.
- Software Testing -- Udacity MOOC.
- Stark, P. B. (2015). Science is "show me", not "trust me". Blog post
- Vitek, J., & Kalibera, T. (2011, October). Repeatability, reproducibility, and rigor in systems research. In Proceedings of the ninth ACM international conference on Embedded software (pp. 33-38). ACM.

References

Bhalla, A. P. S., Bale, R., Griffith, B. E., & Patankar, N. A. (2013). A unified mathematical framework and an adaptive numerical method for fluid–structure interaction with rigid, deforming, and elastic bodies. *Journal of Computational Physics*, *250*, 446–476.

Krishnan, A., Socha, J. J., Vlachos, P. P., & Barba, L. A. (2014). Lift and wakes of flying snakes. *Physics of Fluids*, *26*(3), 031901.

Taira, K., & Colonius, T. (2007). The immersed boundary method: A projection approach. *Journal of Computational Physics*, *225*(2), 2118–2137.

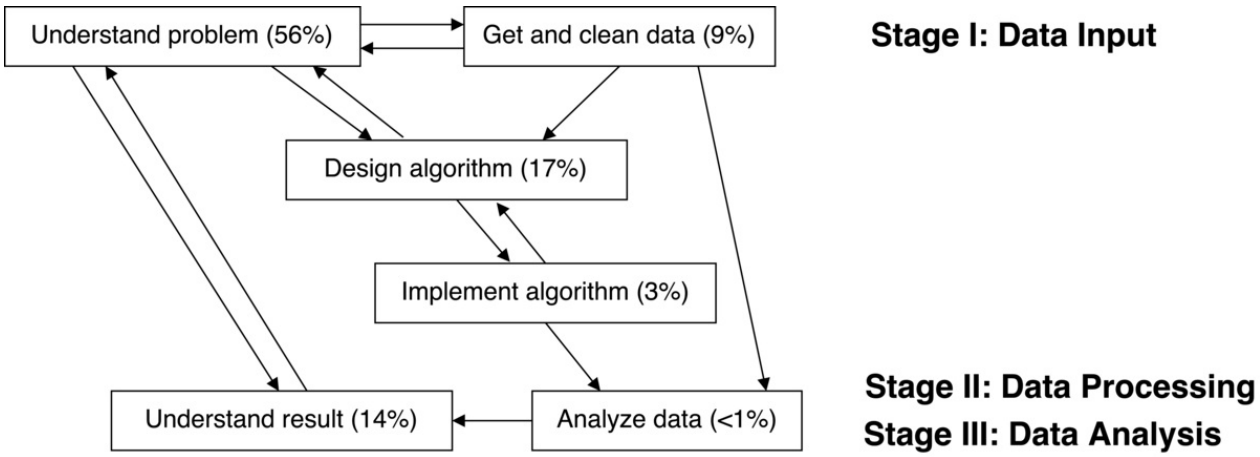
Reproducible Applied Statistics: Is Tagging of Therapist-Patient Interactions Reliable?

K. Jarrod Millman, Kellie Ottoboni, Naomi A. P. Stark and Philip B. Stark

We are three applied statisticians (JM, KO, PS) at UC Berkeley working with a domain specialist (NS) at the University of Pennsylvania. Our case study involves assessing inter-rater reliability (IRR) of the assignment of “tags” applied by human raters to classify interactions during therapy sessions with children on the autistic spectrum.

An extended version of this case study along with the analysis script and results can be found at https://github.com/statlab/nsgk.

Workflow



Our project arose from a pilot study NS was working on with Dr. Gilbert Kliman of the Children’s Psychological Health Center in San Francisco. To investigate therapeutic interventions with children on the autistic spectrum, Dr. Kliman and NS collected some observational data (described below). NS approached PS about the data and the problem NS was studying. After investigating the problem further, PS emailed JM and KO a one-page proposal for a stratified permutation test to assess inter-rater reliability using stratified samples. We (JM, KO, PS) had recently begun developing a general purpose Python package for permutation tests, called `permute`, based on our collaborations. PS suggested this would be an interesting example to include.

After coming to an initial understanding of NS's underlying research question and experiment, including how she collected the data, we (JM, KO, PS) cleaned the data, developed a nonparametric approach to assessing IRR appropriate to the experiment, implemented the approach in Python, incorporated the resulting code into our evolving Python package of permutation tests, applied the approach to the cleaned data, documented the code and the analysis, and wrote up the results in LaTeX.

We distinguish the following aspects of our project, which are typical in applied statistics:

- understand problem
- get and clean data
- design algorithm
- implement algorithm
- analyze data
- understand result

Figure 1 shows how each aspect of the project influenced the other aspects and gives estimates of the total person-hours we collectively spent on each aspect of the project. For example, if JM, KO, and PS spent an hour together discussing the problem in a meeting, then that meeting counts as 3 people hours.

We did not keep a detailed record of time spent, but our computational practices provide enough detail about who did what when that we believe our estimates provide an accurate qualitative account of the time demands for each aspect of the project. However, since these are only rough estimates, the reader should focus on the relative differences in the amount of time we spent on each aspect. We have found that researchers (ourselves included) often underestimate the time needed to understand the problem, acquire and clean the data, as well as understand the results, while overestimating the time needed for writing code. We have included our time estimates to give people an idea how "inexpensive" (or expensive) working more reproducibly is, to capture how our group understanding evolved, and in the hope that it might be instructive for students and collaborators.

Since we view computational reproducibility as a cross-cutting concern of all project aspects, we have adopted a set of computational practices, which we (JM, KO, PS) followed (almost) whenever we were working on the project. Exceptions include that we did not record all of our in-person discussions or whiteboard work. However, we endeavored to record summaries of these activities. These computational practices, described in Millman & Pérez (2014), are used widely in the open source scientific Python community. While developed for managing software contributions, these practices are ideal for ensuring computational reproducibility in scientific and statistical research. We will illustrate how we leverage the

software infrastructure and development practices of `permute` to conduct reproducible and collaborative applied statistics research with our colleagues. We discuss the software tools and practices briefly in Key tools and practices below.

Understand problem (80 hours)

The Kliman-Stark research project sought to identify characteristics of effective clinical interactions with children on the autistic spectrum. The project first required developing a set of characteristics that observers could use to "tag" what was happening in each 30-second interval of a therapy session. After they developed a taxonomy of clinical interactions, Kliman and NS had a number of trained raters watch videos of therapy sessions and label each 30-second interval using the collection of tags. For the classification system to be meaningful and useful, different raters must generally agree on whether a given tag applies to a given video segment: there must be inter-rater reliability. Of course, if a tag is never used or is always used, inter-rater reliability will be perfect, but the tag is useless for distinguishing clinical interactions.

That led to a statistical question: how to assess the evidence in the tagged videos that different raters tag interactions the same way? After numerous conversations, it made sense to consider the null hypothesis to be that, conditional on the number of times a given rater applied a given label to a given video, all assignments of that label to time stamps in the video by that rater are equally likely, and the ratings given by different raters are exchangeable (essentially, that raters are independent).

Once PS had an initial understanding of NS's problem, we (JM, KO, PS) met regularly (approximately weekly, sometimes more) as a team to discuss the project. Initially these discussions involved a lot of work on whiteboards and asking a lot of probing questions. This helped us develop a shared understanding of the problem, understanding that improved by explaining things to one another and by asking hard questions about our planned approach and whether it could address the question of interest. As our understanding of the problem progressed, our work transitioned from working on whiteboards to testing our ideas out on a computer. We often used pair programming at this stage and sometimes we all sat in front of one computer, while one of us typed code in an interactive IPython session. This helped ensure that we all understood the problem well and it also helped us catch errors (typos as well as conceptual misunderstandings).

Get and clean data (13 hours)

The tag data were collected by NS and raters working at her direction. The data comprise ratings of segments of 8 videos by 10 trained raters. Each video is divided into approximately 40 time segments. In each time segment, none, any, or all of 183 types of

activity might be taking place. The raters indicated which of those activities was taking place during each segment of each video.

PS received the data from NS as an Excel spreadsheet that had been entered by hand by NS and an assistant. Understanding the “data dictionary” and vetting for obvious errors entailed several rounds of email between PS and NS before PS had a version of the data that did not have obvious errors. PS then exported the Excel data to comma-separated value (CSV) format. The original data contained personally identifying information. Using regular expressions in an interactive text editor, PS substituted unique numerical identifiers for raters’ names in the CSV file. While this step was not performed reproducibly (i.e., not scripted), it can be checked readily. After PS generated the original anonymized data, JM committed it to our repository and added a data loader with tests to ensure that if the data changed we would know. At this point, we (JM, PS) screened the anonymized data for transcription errors (e.g., duplicate rows). This involved writing a number of quality assurance tools (e.g., to find duplicate consecutive rows), which are now included in `permute`. Once we identified entries incompatible with our understanding of what should be in the data, JM wrote a `sed` script to “correct” the inferred typos. The exact commands used to clean the data are included in the commit corresponding to that cleaning step. After carefully examining the data for potential errors and documenting every change we made and why, we sent the cleaned data and an explanation of what we did to NS to verify that the corrections were appropriate. As a result, we provide the cleaned data in our project repository as well as a careful account of its provenance.

Design algorithm (25 hours)

Although the test we eventually implemented was very similar to the original test proposed by PS at the start of the project, we (JM, KO, PS) spent significant time focused on “problem appreciation,” some of which resulted in considerable simplification of the algorithm used to implement the test. We also developed a more general terminology (see Table 1).

Mapping between terms from our motivating problem (NSGK) and the terms used in our general algorithm (IRR).

NSGK	IRR
183 types of activity	T tags
8 videos	S strata
40 segments/videos	N_S items/strata
10 raters	R raters

We decided to assess rater reliability in identifying (i.e., tagging) each of the 183 types of activity separately, because they are of separate interest. This introduces questions about whether inferences are to be made about each tag separately (per-comparison error rate, PCER) or simultaneously (familywise error rate, FWER), or whether we are concerned with the fraction of tags we conclude are reliable that in fact are not reliable (false discovery rate, FDR). Ultimately, we decided that the PCER was the most relevant error criterion, since the tags are individually interesting. As a “first cut” through the rating scheme, eliminating tags that are clearly not reliable across raters simplifies the scheme and reduces the cognitive burden on raters, because they do not have to keep so many categories of activity in mind. We imagined that if we could eliminate a substantial number of the tags as unreliable, there would be a repeat of the tagging using a different set of raters to validate or refine the results, reducing the rate of “false positives.” On the other hand, incorrectly rejecting tags as unreliable could eliminate a potentially useful predictor of successful therapeutic outcomes, so the FWER seemed far too stringent a criterion. See the Understand result section below for more discussion.

Since each of the videos contained different sessions of therapist-patient interactions, in general rated by different people, we stratified the test by video. A literature search for approaches to assessing IRR led us to conclude that there was no existing suitable method for several reasons: the experiment was stratified; there were multiple raters but not the same set for all videos; and standard methods required indefensible parametric assumptions or population models, which we hoped to avoid. After deciding to use permutation tests, we (JM, KO, PS) then determined that permuting each rater’s ratings within a video, independently across raters and across videos, made sense as the appropriate invariant under the null hypothesis. We chose to use concordance of ratings as our partial test statistic within each stratum. We (JM, PS) derived a simple expression for efficiently computing the concordance. To combine tests across strata, we (JM, KO, PS) used the nonparametric combination (NPC) of tests (Pesarin & Salmaso, 2010) with Fisher’s combining function. Finally, we developed a computationally efficient approach to finding the overall *p*-value for the NPC test.

Implement algorithm (5 hours)

Once we had a blueprint of the algorithm, KO led the implementation effort. She did most of the coding; JM and PS reviewed the code and discussed the implementation. Following our software development practices, KO also wrote tests for every function she implemented. After a few iterations of coding, testing, and review, KO finalized our implementation and we merged it into `permute`.

KO wrote three functions to implement our general IRR algorithm:

1. a function to compute the IRR partial test statistic from a binary matrix with one row per rater and one column per item;
2. a function to simulate the permutation distribution of the IRR partial test statistic for a matrix of ratings of a single stratum;
3. a function to simulate the permutation distribution of the NPC test statistic by combining the *S* distributions of the IRR partial test statistic for each of the *S* strata.

Analyze data (1 hour)

Once we merged KO's implementation of the general algorithm (including tests) into `permute`, KO wrote a short script (about 50 lines of Python) to analyze the cleaned data from NS.

Since we included the main workhorse functions in `permute`, the analysis script contained only high-level commands:

1. Load the cleaned data
2. For each of the 183 categories of activity:
    i. For each of the 8 videos:
        i. Compute the mean and standard deviation of the number of times the tag was applied
        ii. Compute the IRR partial test statistic
    ii. Simulate the permutation distribution of the NPC test statistic for each tag combined over the *8* videos, and report a single *p*-value
3. Save the results to a CSV file

Understand result (20 hours)

At a high level, even the summary statistics we computed were useful: some tags were never applied by any rater to any video. Presumably, the tag taxonomy could be simplified by eliminating those tags from the universe of labels, reducing the cognitive burden on the human raters. There were also tags that were used so frequently that high concordance was virtually guaranteed—and therefore high inter-rater concordance was not evidence of inter-rater reliability. This may also imply that any differences in efficacy of therapy are not attributable to whether the corresponding activity is taking place, since it is often taking place, at least in these sessions. Whether it makes sense to keep such tags in the taxonomy

depends in part on subject matter knowledge: are those interactions typical only in the videos in these evaluation data, or are they typical of all therapeutic interventions with children on the autistic spectrum?

At the other extreme, there were tags for which the concordance of use was quite low, but still highly significant. This raises the scientific question of what threshold level of agreement among raters makes a tag interesting or useful, separate from whether the agreement is statistically significant. That is a matter we need to discuss at greater length with the domain specialists. It also points to a frequent situation in statistics: practical significance and statistical significance are not the same thing, and one must consider “fitness for use” when devising summary statistics.

We hope that the concrete findings will lead to a refinement of the taxonomy and additional tests of reliability. We hope that those tests will involve greater automation of data collection and transcription, to eliminate some of the sources of error in the data. Regardless, this work has led to a new nonparametric test for inter-rater reliability, now available publicly in the `permute` package.

Pain points

Given our different backgrounds and experiences we (JM, KO, PS) each found different points in the process challenging. However, for all of us the most challenging aspect -- and the most time-consuming -- was the necessary struggle to understand the scientific question and the experiment well enough to devise an approach to answering the question.

For KO and PS there was a learning curve to master the tools and practices. This involved understanding the data model used by git, acquiring habits such as writing tests for all functions and following a common style guide, and learning to contribute to the project repository indirectly through GitHub’s pull request mechanism. JM was already familiar with the tools and practices, and devoted significant time to teaching KO and PS the workflow. Once mastered, the benefits of these tools and habits outweigh the time and effort spent learning them.

For JM the most painful part of the project was vetting hand-entered data to look for errors and inconsistencies. Not only was this laborious, but it involved inferring what the data should have been without any direct way to ensure that these inferences were correct: the original raters and videos were not available to us. The solution to this pain point is to automate data collection as much as possible. However, when data have already been entered by hand, there is not much that can be done other than being cautious when “fixing” data entry errors and recording every aspect of the data cleaning process.

Key benefits

Since Buckheit & Donoho (1995) popularized the idea of computational reproducibility, applied statisticians have increasingly embraced version control and process automation. Many of our colleagues have made the idea of computational reproducibility central in both the classroom and the lab. Some ask anyone working with them to follow a set of computational practices including version control.

However, the computational practices described in this study (see Key tools and practices) go beyond the standard work habits of our colleagues. Our computational practices provide the following benefits:

1. it reduces the number of errors introduced by new code and changes to existing code
2. it makes it easy to modify the analysis when errors are found, to apply the analysis to new datasets, and so on
3. the process is self-documenting, making it easier to draft a paper about the results or to pick up where we left off after working on something else
4. the methods are abstracted from the analysis and incorporated into a package so that others can discover, check, use, and extend our methods.

Key tools and practices

As part of the development of our software package `permute`, we invested significant effort in setting up a development infrastructure to ensure our work is tracked, thoroughly and continually tested, and incrementally improved and documented. To this end, we have adopted best practices for software development used by successful open source projects (Millman & Pérez, 2014).

Version control and code review

We (JM, KO, PS) use git as our version control system (VCS) and GitHub as the public hosting service for our official `upstream` repository statlab/permute. Each of us has our own copy, or fork, of the `upstream` repository. We each work on our own repositories and use the `upstream` repository as our coordination or integration repository.

This allows us to track and manage how our code changes over time and to review new functionality before merging it into the `upstream` repository. To get new code or text integrated in the `upstream` repository, we use GitHub's *pull request* mechanism. This enables us to review code and text before integrating it. Below, we describe how we automate testing our code to generate reports for all pull requests. This way we can reduce the risk that changes to our code break existing functionality. Once a pull request is reviewed and accepted, it is merged into the `upstream` repository.

Requiring all new code to undergo review provides several benefits. Code review increases the quality and consistency of our code. It helps maintain a high level of test coverage (see below). Moreover, it also helps keep the development team aware of the work other team members are doing. While we are currently a small team and we meet regularly, having the code review system in place will make it easier for new people to contribute as well as capturing our design discussions and decisions for future reference.

Testing and continuous integration

We used the `nose` testing framework for automating our testing procedures. This is the standard testing framework used by the core packages in the scientific Python ecosystem. Automating testing allows us to monitor a proxy for code correctness when making changes as well as simplifying the code review process for new code. Without automated testing, we would have to manually test all the code every time a change is proposed. The `nose` testing framework simplifies test creation, discovery, and execution. It has an extensive set of plugins to add functionality for coverage reporting, test annotation, profiling, as well as inspecting and testing documentation.

Our goal is to test every line of code. For example, not only do we want to test every function in our package, but if a specific function has a conditional branching structure we test each possible execution path through that function. Having tested each line of code increases our confidence in our code and provides some assurance that changes we make do not break existing code. It also increases our confidence that new code works, which reduces the friction of accepting contributions. Currently over 98% of the lines of code in `permute` get executed at least once by our test system.

We often work on several pull requests simultaneously. These pull requests may take several weeks or months before they are reviewed, improved, and accepted in our `upstream` repository. While we are working on one pull request, we may merge several others. Since the underlying code base is changing, each pull request may potentially introduce integration conflicts when we attempt to merge it back into the main line. To mitigate the difficulty in managing these conflicts we employ continuous integration and track our test coverage.

Continuous integration works as follows: Each pull request (as well as a new commit to an existing pull request) triggers an automated system to run the full test suite on the updated code. Specifically, we have configured Travis CI and `coveralls` to be automatically triggered whenever a commit is made to a pull request or the `upstream` master. These systems run the full test suite using different versions of our dependencies (e.g., Python 2.7 and 3.4) every time a new commit is made to a repository or a pull is requested. Travis CI checks that all the tests pass, while `coveralls` generates a test coverage report so that we can monitor what parts of our code are checked by a test and which are not. This system

checks whether any of our automated tests fail as well as tracks the percentage of our code that is covered by our automated tests. This means that when you review a pull request, you can immediately see whether the proposed changes break any tests and whether the new code decreases the overall test coverage.

Documentation

We use Sphinx as our documentation system and have extensive developer documentation and the foundation for high-quality user documentation. Sphinx is the standard documentation system for Python and is used by the core scientific Python packages. We use Python docstrings and follow the NumPy docstring standard to document all the modules and functions in `permute`. Using Sphinx and some NumPy extensions, we have a system for autogenerating the project documentation (as HTML or PDF) using the docstrings as well as stand-alone text written in a light-weight markdown-like language, called reStructuredText. This system enables us to easily embed references, figures, code that is auto-run during documentation generation, as well as mathematics using LaTeX.

Release management

Our development workflow ensures that the official `upstream` repository is always stable and ready for use. This means anyone can install our official upstream master at any time and start using it. We also make official releases available as source tarballs and as Python built-packages uploaded to the Python Package Index, or PyPI, with release announcements posted to our mailing list.

By making official releases whenever we reach an important stage of an applied project, we are able to easily recover the exact version of our analysis at a later date. To install the exact version of `permute` used in this case study, type the following command from a shell prompt (assuming you have Python and a recent version of `pip`):

```
$ pip install permute==0.1a2
```

Questions

What does "reproducibility" mean to you?

In this case study, *reproducibility* means:

- *Computational reproducibility and transparency.* We have documented (and scripted) nearly every step of the analysis—from cleaning to coding to code execution—and made the code and documentation publicly available.

- *Scientific reproducibility and transparency.* We documented much of the discussion leading to our decisions to take each step in the analysis. We made the data publicly available in an open format, with an adequate data dictionary.
- *Computational correctness and evidence.* We tested our code thoroughly and in an automated fashion, to have justifiable confidence that the code does what it was intended to do. We made those tests publicly available, so that others can see how we validated our computations.
- *Statistical reproducibility.* We invested time to understand the fundamental problem and the results of our analysis so that we do *not* draw conclusions that are not justified by the data, the manner in which it was acquired, and our domain understanding. We avoided “p-hacking” and other potentially misleading selective reporting, and made all our analyses publicly available.

By keeping all code, text, and data in a public version-controlled repository, we have made our well-documented analysis available for anyone to examine, check, modify, or reuse. We published the data used in our study -- both the original anonymized version as well as our cleaned version including the commands necessary to produce the cleaned version from the anonymized one. In addition to making what we did transparent to anyone who is interested, working in this way means that when errors are found we can identify how and when those errors were introduced. We have written tests for almost all our code, which means we have a high level of confidence that as we change our code we will catch any errors we might have introduced, and can correct them quickly and easily. And since we have automated the process of running our analysis, if errors are identified and corrected, it is easy to rerun the entire analysis from start to finish.

If you have standard tools on your computer and network access, you can run our complete analysis of the cleaned data by typing the following three commands from a Unix shell prompt:

```
$ git clone git@github.com:statlab/nsgk.git
$ cd nsgk/nsgk
$ make
```

The first command creates a directory `nsgk` in your current working directory with a copy of the project repository (i.e., a directory with our code, data, and text along with the provenance of these documents). This directory contains this document as well as everything needed to run our analysis. Inside `nsgk/nsgk` there is a `Makefile`, our analysis script `analysis.py`, and the output `results.csv` of that script.

When you enter the command `make`, the following commands will be run:

```
virtualenv -p /usr/bin/python2.7 venv
venv/bin/pip install --upgrade pip
venv/bin/pip install -r requirements.txt
venv/bin/python analysis.py
```

The first command creates a new virtual environment (`venv`) for Python 2.7. Using this new virtual environment (`venv`) the subsequent commands respectively upgrade the Python package manager (`pip`) to the most recent version, install the necessary Python package dependencies (`numpy 1.11.0` , `scipy 0.17.0` , and `permute 0.1a2`), and run the analysis script `analysis.py` .

References

Buckheit, J. B., & Donoho, D. L. (1995). Wavelab and reproducible research. In A. Antoniadis & G. Oppenheim (Eds.), *Wavelets and statistics*. Springer.

Millman, K. J., & Pérez, F. (2014). Developing open-source scientific practice. In V. Stodden, F. Leisch, & R. D. Peng (Eds.), *Implementing reproducible research* (pp. 149–183). Chapman; Hall/CRC.

Pesarin, F., & Salmaso, L. (2010). *Permutation tests for complex data: Theory, applications and software.* John Wiley & Sons.

A Dissection of Computational Methods Used in a Biogeographic Study

K. A. S. Mislan

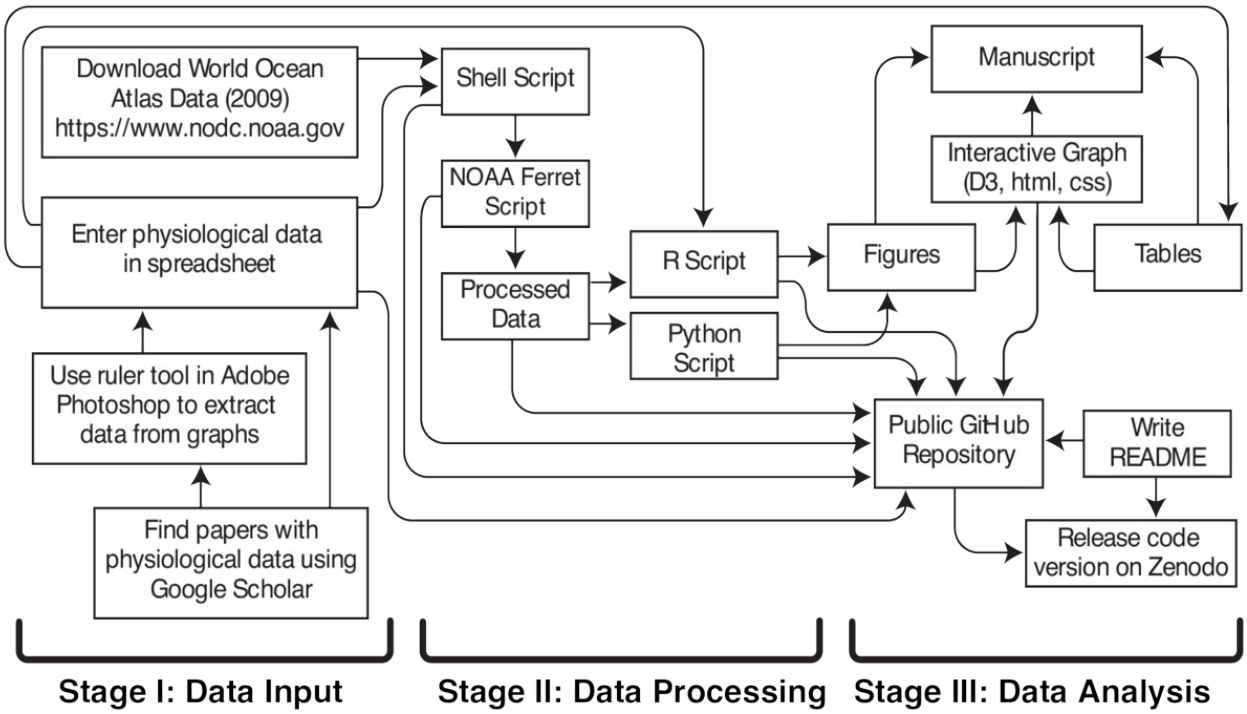
My name is Allison Smith, I am an ecophysiologist and my research focuses on organism-environment interactions in the ocean. In particular, I am interested in forecasting the effects of climate change on marine ecosystems.

I recently published research on the fundamental niche of pelagic animals in the global ocean. The study included a comparison of blood-oxygen binding characteristics of different species obtained from published papers. Thresholds for blood-oxygen binding characteristics were mapped in the ocean using gridded oceanographic data. My workflow details my process for obtaining and analyzing data for the project. In order to increase the reproducibility of the study, code used for the project was put in a long-term archive.

Research paper:\ Mislan, K. A. S., Dunne, J. P. and Sarmiento, J. L. (2016), The fundamental niche of blood-oxygen binding in the pelagic ocean. Oikos. https://doi.org/10.1111/oik.02650

Code archive:\ Mislan, K. A. S., Dunne, J. P. and Sarmiento, J. L. (2015). P50 Depth Analysis v1.0. Zenodo. http://dx.doi.org/10.5281/zenodo.31951

Workflow



Obtaining data from existing resources was the first step. World Ocean Atlas 2009 (WOA09) data is publicly available through the National Oceanic and Atmospheric Administration (NOAA) National Centers for Environmental Information. The WOA09 data is on a geographic grid and available in several different file formats. I downloaded the network common data form (netCDF) file format. The NetCDF file format facilitates access and sharing of scientific data in arrays. There are many tools available to read and manipulate netCDF files. NOAA Ferret is publicly available software for visualization and analysis and has built-in functions designed for efficient processing of geographic data in netCDF file format. One of the objectives of the analysis was to vary two input parameters and determine the effect on geographic characteristics. I automated the process of creating NOAA Ferret scripts with different input parameters by writing a shell script that generated and processed NOAA Ferret scripts (.jnl files). The processed files were saved as netCDF files.

The physiological data were gleaned from published studies found through Google Scholar searches for key words. The data were extracted from the papers and put into a spreadsheet. The data of interest were often available in tables and the text of the papers. In some cases, the data were only available in scatter plots. The ruler tool in Adobe Photoshop was used to manually extract the data from plots. Once all the information from the studies was entered into the spreadsheet, the data from the spreadsheet were saved as a tab-delimited text file. The data were then read into R to determine parameters for the analysis of the oceanographic data. Additionally, the data were plotted in a scatter plot in R and each point in the plot had a number assigned to it. Information about the individual points was put into two tables. However, the numbers on the plot had to be matched to numbers in the two tables to get relevant information from the plot, which was inefficient. Therefore, I created a web-based interactive graph that embeds information from the tables into the scatter plot

using a javascript library called Data-Driven Documents (D3). In the interactive graph, the information for each point is visible when the cursor is placed on the point. The interactive graph also includes options to select different legends for the graph that highlight additional groupings for the points. A link to the interactive graph was included in the publication.

R and Python were used to create the figures although it would have been possible to create all the figures in one or the other of these software packages. I learned R before Python so I tend to do most of my analysis in R because I am most familiar with it. However, I like the tools for making geographic plots in Python so I used Python to make the geographic plots. Then I wrote the paper.

Code is not usually included in the methods section of a paper due to space and formatting constraints. However, the code tells a much more complete story of my analyses. In order to prepare my code for archiving, I made some modifications. I created a folder structure that would make it easy for a user to find files. The folder structure included a folder for code, folders for input files to run the code, output files produced by the code, and graphs produced by the code. I annotated my code while I was writing it, but the annotations were usually short and meaningful only to me. Descriptive annotations were added to the code being archived. I also changed the file paths to be referenced solely within the folder structure (../../) so that the code would be independent of my home directories (/Users/kasmislan/code/project).

My code worked on my computer, but it might not work the same way on a different computer. For example, different operating systems have different methods for rounding numbers. I generated output files for my code on my computer and moved the files to the test files folder. Then I wrote a script that automatically compares output files produced by another user using my code on another operating system to the test files I produced using my code on my operating system. If there are differences, then a user will be able to identify and address them.

The most critical step for archiving my code was writing the documentation (README file). The documentation includes a description of the purpose of the code, references to research articles, a list of software dependencies including versions, clear step-by-step instructions on how to use the code, and clear step-by-step instructions on how to use the test files. I also included a section to acknowledge my funding sources and others who helped with the project.

The final step was to submit the code with the Massachusetts Institute of Technology (MIT) License to Zenodo, a long-term archive. The MIT License has few restrictions which maximizes that ways in which my code can used and adapted by others. I sent the link to the code archive to the journal so that it could be included in the publication.

Pain points

My primary pain point is that archiving my code takes additional time after I am ready to submit a scientific paper. The most time-consuming steps are associated with making the code usable by someone else on their own computer. As I wrap-up a project, my files are cluttered because I have an exploratory phase as I am analyzing data where I write code that is not ultimately used for the results presented in the paper. I have to identify and organize the relevant code files. Then I have to modify filepaths, annotate the code, create test files, and write a README file with instructions for using the code which also takes additional time. Another pain point is trying to find someone to test the code to make sure that the instructions are comprehensible, and the code runs on other computers and operating systems without errors. This step requires another person to spend time to help me archive my code. In the current scientific research system, archived code is not valued as highly as scientific papers so the extra time spent by scientists and code testers to archive code does not directly translate into greater scientific success. Some academics have hypothesized that papers with archived code are cited more often, but this has not been universally verified. In my experience, my archived code is limited to a specific analysis and, while the availability of code may increase the confidence of the scientific community in my results, I do not think that my archived code is generating more citations of my scientific papers.

Key benefits

Reproducibility has always been an important component of research in my field. In the past, instructions for reproducing research were put in a methods section in journal articles. The increasing importance of code in my field is changing the way reproducibility is accomplished because it is not possible to include code in a methods section. However, it is necessary to have access to the code to reproduce the research.

Key tools

I have always believed that making my code available is important, but, until recently, I was not sure how to do it. GitHub is a "game-changer" for sharing code with others in a reliable, consistent, and discoverable way. After I was introduced to GitHub, I was able to start archiving my code. My code is posted to GitHub and a permanent copy and digital object identfier (doi) are generated by Zenodo.

There are specific instructions on GitHub for releasing code to Zenodo:

https://guides.github.com/activities/citable-code/

Questions

What does "reproducibility" mean to you?

Reproducibility means that sufficient descriptive information and resources are provided for someone to be able to repeat the same study. As part of my research as a scientist, I write code to manipulate and visualize large quantities of data to obtain results. I believe that my code should be adapted and made available so that it is usable by others in my field.

Why do you think that reproducibility in your domain is important?

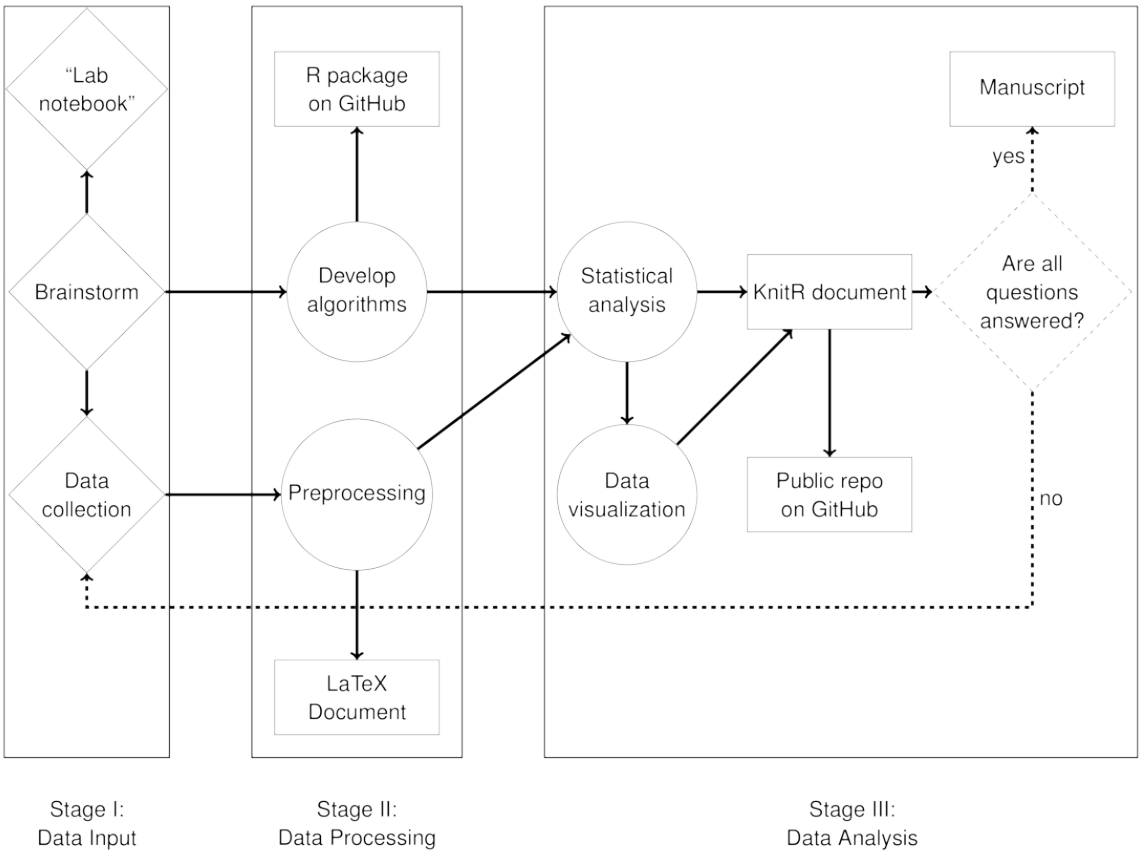
Reproducibility has long been a central tenet of the scientific research process in my field because data from new studies are compared to data from earlier studies. Not so long ago, all the data collected during a study were included in published research articles, and the analyses could be easily described in a materials and methods section of the articles. Recent increases in the quantity of data and the concurrent increase in the complexity of the analyses has made it impossible to include the data itself or the same level of descriptive detail in the materials and methods sections of research articles. I think that my domain values reproducibility but the current format for publishing research articles is not able to fully accommodate the modern scientific research process.

A Statistical Analysis of Salt and Mortality at the Level of Nations

Kellie Ottoboni

My name is Kellie Ottoboni. I'm currently a PhD student in the Department of Statistics at UC Berkeley and a Data Science Fellow at the Berkeley Institute for Data Science. My research focuses on nonparametric statistics, causal inference, and applications in health and social sciences. The project I describe in this case study is an investigation of the association between salt consumption and mortality at the level of nations.

Workflow



This project started as a collaboration between my advisor and a professor of public health at UC Irvine, along with one of his students. I became involved after our collaborators had begun putting together the dataset. The data consists of demographic and socioeconomic

variables, as well as gender-specific estimated sodium, alcohol, and tobacco consumption for 38 countries. The data were pulled together from several sources and the variables are described in a text file.

The first step was to decide what data we needed to answer the question: does salt consumption have an effect on a nation's life expectancy at age 30? We decided that the best way to address this would be to consider males and females separately, and to use a country as its own "control" by looking at the changes in life expectancy, alcohol, tobacco, and sodium consumption, and economic variables over time. We had the variables we needed, but gender-specific alcohol data were missing for one year. These couldn't be obtained so we imputed them based on gender ratios in each country and overall consumption that year. We also had to remove countries with missing data on life expectancy, the outcome of interest. In addition to the R code implementing these steps, I described them in a LaTeX report as I went along.

The method of analysis was a novel hypothesis test that I have been developing with my advisor. The premise of the method is to predict the outcome of interest, change in life expectancy, using all covariates *except* the treatment of interest, sodium. If sodium is still related to life expectancy after controlling for known health predictors, then sodium consumption will be associated with the residuals of the model. This is simply a mathematical fact: some of the variation that the model cannot explain will be associated with sodium. But how large of an association is statistically significant? We answered this question using nonparametric permutation tests. After discussing how our approach might answer the problem at hand, I wrote R code for the two main steps in the algorithm: the model selection and the nonparametric test of association. I had already written code for our proposed statistical method in R package format in a public GitHub repository, so I added the new code from this project into the package. I developed the code iteratively by running it on our dataset and checking that the output looked sensible. This isn't the best way to write code: ideally, I would have invented simple test cases and checked my functions against those.

After writing each component of R code, I combined all of the preprocessing, analysis, and plot scripts into an R Markdown file. knitr allows you to compile R Markdown, R code chunks interleaved with markdown text, into a PDF or HTML document. This way, I could send my collaborators the results quickly and in a user-friendly format. I posted all the scripts, data, and the compiled HTML document in a public GitHub repository dedicated to this project.

At this point, we were unsatisfied with the scope of the analysis and wanted to ask more questions. In particular, we wanted to run the same test of association but use tobacco and alcohol in place of sodium. These analyses would serve as a "sanity check" that the method performs as expected. Given that we know that both alcohol and smoking have negative effects on health, we would expect to find a negative association between the model's

residuals and alcohol and tobacco use. The original dataset included alcohol consumption per capita, but no measures of tobacco use. This required our collaborators at UC Irvine to gather this data from an existing journal article. After receiving the tobacco usage data from them and performing the model-based matching analysis with it, we realized that the measure of smoking that we used was not a good measure of a nation's overall tobacco consumption. We again gathered different tobacco data and ran the analysis one last time. Since the R scripts had already been written, redoing the analysis with each version of the data amounted to changing a few lines of code. I kept each version of the data used and named them according to the date when it was sent to me.

We believe that we have done all the statistical analysis that we need to answer the scientific questions that we posed. The analysis portion of the project took about six months to complete. Our collaborators are preparing a manuscript. We are using LaTeX and communicating by email.

The workflow diagram distinguishes three types of activities: thinking and planning steps are marked with diamonds, action and implementation are marked with circles, and documentation and outputs are marked in rectangles. While arrows point to the right and boxes separate the three stages of the workflow, this is a bit artificial. In my experience, there is a lot of iteration involved in applied statistics projects. In addition, the number of nodes in each stage is not reflective of the amount of time spent. The majority of my time was spent planning and documenting. It was quick to make minor changes once the preprocessing and analyses were scripted.

Pain points

A seemingly trivial problem I struggle with is keeping track of files. Using git certainly simplifies the version control aspect of file organization, but that doesn't help when I've forgotten where I put a chunk of code I wrote two weeks ago. Around the beginning of this project, I started keeping a "lab notebook" to organize my thoughts on all the various projects I'm doing. I keep a folder of text files just for myself where I jot down the date, ideas and concerns, notes on what work I've done, and the names of files or folders where I saved that work. It has helped tremendously when I need to remind myself things about a project, and it's also a nice way to archive meeting notes and save ideas that I might want to share with collaborators later on.

A pain point particular to this project was trying to encourage my collaborators to use the GitHub repository. Ultimately, we ended up sending data and results back and forth by email. Pushing updated data and results to the repository would have been more efficient.

The data collection part of the project was opaque. Our repository does not include any scripts used to collect the data from various databases and journal articles or scripts to merge these data sources. At one point, under pressure of a deadline, I manually entered tobacco consumption figures into an Excel spreadsheet. All the preprocessing and analyses of the data are reproducible, but the process of collecting the data is not.

Key benefits

The biggest advantage of a reproducible workflow is efficiency. There were many iterations of the analysis for this project. By having preprocessing, analyses, and results written in the same document, it was easy to make small changes and ensure that they appeared throughout the report.

Incorporating the main functions for this analysis into an R package with larger scope will be beneficial in the future. The functions I wrote for hypothesis testing here are well-documented and uniform in their style, inputs, and outputs. Having them all in a package on GitHub makes it easy for anybody who reads our paper to install the package and replicate the results.

Key tools

RStudio and knitr were key for this workflow. This was my first time using knitr and I am pleased with the quality of the reports I created to share the results with my colleagues. All steps of the data analysis are in the documents, alongside my commentary and explanation of the steps. I hope that this makes the statistical methods transparent to my collaborators and future readers. Additionally, having all the tables and figures in one place will make it easy to put results into the manuscript.

Questions

What does "reproducibility" mean to you?

I think that a data analysis project is reproducible if there are enough breadcrumbs (in the form of code and instructions) for anybody to recreate the analysis from start to finish. In another sense, a project is reproducible if someone can carry out a different analysis on the data and arrive at qualitatively similar conclusions.

Why do you think that reproducibility in your domain is important?

Researchers tend to blame the "reproducibility crisis" on statistics, and in particular p-values. It's our job as statisticians to fight this claim by making statistical analyses as correct and as transparent as possible, so people know exactly where their p-values are coming from.

How or where did you learn about reproducibility?

I learned some of these practices from other students in my department and the rest were self-taught using resources on the internet.

Are there any best practices that you'd recommend for researchers in your field?

Explain every step in the data preprocessing and analysis carefully and thoroughly. Document and comment code liberally. Make code and data publicly available.

Would you recommend any specific resources for learning more about reproducibility?

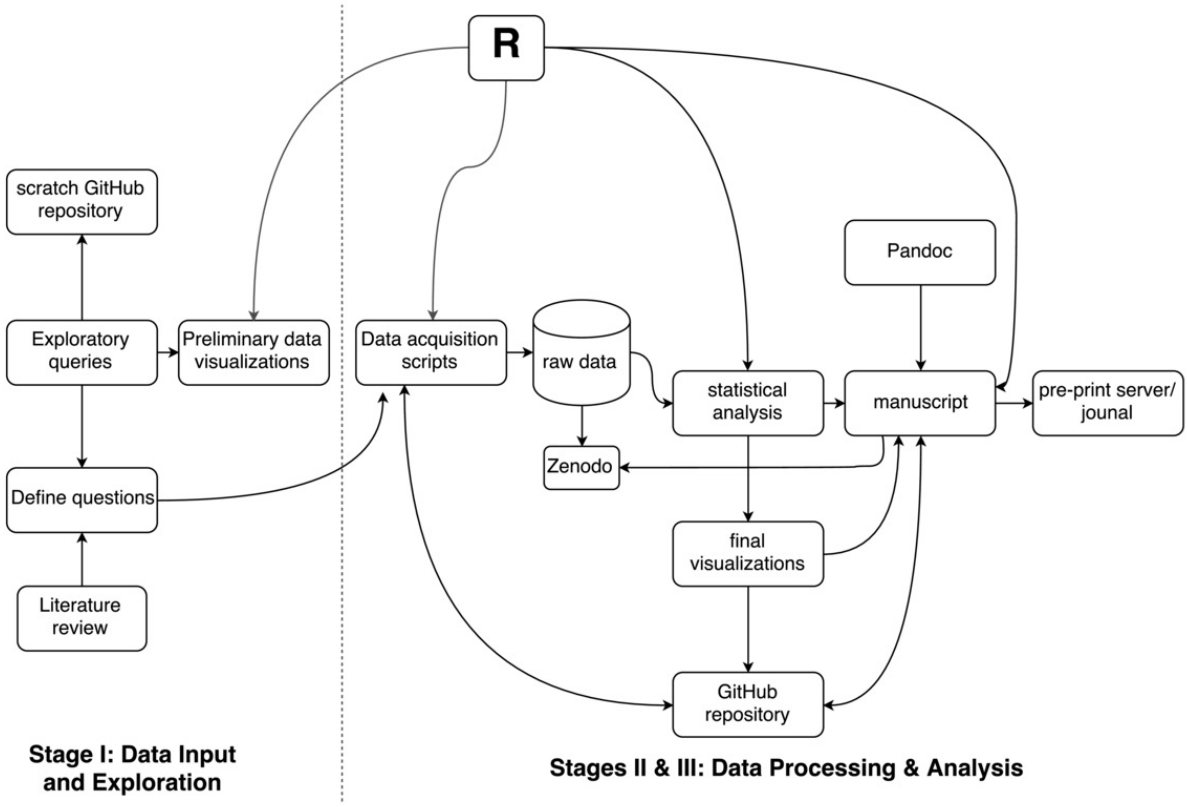
Hadley Wickham's *R Packages* book is an invaluable resource. He demystifies the R package and shows how to use RStudio to make the workflow smooth and efficient.

Reproducible Workflows For Understanding Large Scale Ecological Effects Of Climate Change

Karthik Ram

My name is Karthik Ram and I am a quantitative food web ecologist by training. I received my PhD in 2009 from the University of California, Davis. Since then I've studied the impacts of climate change on the rate at which the landscape greens up in spring time, and the consequent effects on a large mammal food web, among various other projects. The Yellowstone project was my entry into data science, where I was thrust into the challenges of validating (reproducing) someone else's work before continuing on with my own research. During this time I also encountered various pain points, which led me to create my own version control system before I learned of the existence of git. In my current job I've transitioned from a full-time research scientist to a hybrid role, where I spent part of my time on research activities but the rest on developing tools and workflows to support various stages of a reproducible research workflow which I describe below.

Workflow



Even though this narrative does not describe any specific project that I work on, it captures the general workflow I employ for all my projects. With all of my messy, raw data ready, I spend considerable time on exploratory data analysis during which I generate several visualizations. This process allows me get a sense for any early quality issues with the data and the kinds of steps I will need to employ to make the data usable for analysis. All of the code I use for this process (usually `R`), along with the outputs (rendered markdown and figures using packages such as `ggplot2` and `knitr/rmarkdown`) are committed to a scratch GitHub repository. This allows me to share early insights into my data with my collaborators. This process is highly iterative, and I generate various visualizations (across multiple branches) to gain a better insight into my data. This process takes me a few weeks as I multi task other projects. During this time, project collaborators and various others, including my Twitter followers, provide constructive feedback.

During this process I also document any data cleaning steps I'll need to undertake before I begin any data analysis steps. At this time I also deposit my raw, unprocessed data into a persistent repository such as figshare, or a Zenodo collection, and obtain a permanent identifier. figshare is a private company that provides free data archiving to individual users. Zenodo is an EU funded research archive that allows scientists to deposit various types of digital objects, including software and code.

I simultaneously start writing code to clean my data using a scripted workflow which also involves mostly R. I sometimes use a bash script or two to pre-process the data using old unix tools like `sed` and `awk` but with recent developments in the R toolkit (`data.table` ,

`dplyr`, `tidyr`, and `rvest`) I rely less and less on my bash scripts. These scripts are called inside a Make file, which allows me to generate my cleaned datasets at any time with a simple command line call e.g. `make clean_data`. At this time I also start the process of creating a separate library for the project to capture the right versions of my tool dependencies, such that further updates to my computer don't affect the reproducibility of my work. In the case of `R`, I use a library called `packrat` to accomplish this. Once my data are cleaned, I deposit them back at the same identifier on the persistent data repository and include the DOI in the text of my paper.

The data analysis and modeling steps vary based on a project to project basis as some involve simulations on a cluster. For smaller projects, a handful of scripts accomplish this process. If any of my code is reusable in more than one step, I capture these into common functions, and sometimes convert this into a separate package. This allows me to further modularize my code. For projects that involve simulations on clusters, I create scripts that work on smaller examples for local testing, with full version that run on high performance clusters (HPC) and write out my results to disk.

Somewhere along the way, I also begin a Rmarkdown file, i.e. my manuscript, which is merely a markdown file with embedded R code captured inside code fences with some metadata. In addition to including small snippets of code, I am also able to source in larger chunks of code from my scripts without cluttering my document. The manuscript is also rendered from my Makefile frequently. I also include additional code to turn the `Rmarkdown` → `markdown` → `PDF` (using `Pandoc`), which gives me a sense for the how the final manuscript might look line. All of the code, figures, and raw/rendered markdown files are committed to my manuscript's GitHub repository. I have configured git to ignore large intermediate files that could easily be generated again in the future. As a researcher who practices open science, I leave the manuscript publicly accessible in my (or collaborators') GitHub repositories. GitHub now renders both the PDF, and also both the unparsed markdown (RMarkdown) and the markdown files on the browser, allowing anyone to review my work in progress.

For citations, I use the `knitcitations` R package to embed DOIs into my text, which are automatically rendered into full parsed citations and a bibliography during the Make process. For projects that cite content such as blog posts, I rely on Mendeley's bibliography rather than retrieving citations from Crossref using `knitcitations`.

Pain points

The two biggest pain points in my research are related to black box data and unscripted data processing steps. I frequently collaborate with researchers who process one or more chunks of data using proprietary, closed sourced software. Quite often, these steps are also not

scripted, requiring human intervention to update outputs as input data change. This combination of factors results in out of data versions of one or more pieces of input data simply because there was no automated way to determine what steps needed to be rerun.

In an ideal world, all my data would be a few simple queries away, allowing me to write concise scripts to retrieve the raw data before analysis. In reality, my data are a hodgepodge of manually entered data, sensor derived data (often bulk downloads after a mandatory sign in step), and other data retrieved via application programming interfaces (API). I try to alleviate the burden for anyone trying to reproduce my results by depositing all of my raw data and associated scripts into either institutional or other repositories so that others can replicate my research.

Key benefits

The key benefits to the approach I have outlined above are that anyone with the right technical skills can clone all of my code from a repository and re-run all the steps necessary to acquire the raw unprocessed data, munge the data, then run all associated statistical analysis and generate the full manuscript as published in a pre-print server or journal. Depending on the complexity of a paper, this could either be a single step, or a series of steps linked together in a Makefile. My work almost always include instructions in a `README` file.

Key tools

I've outlined my major tools inline but briefly: Programming tools: R, Make, git, Pandoc Services: GitHub, Zenodo, figshare, Mendeley

Questions

What does "reproducibility" mean to you?

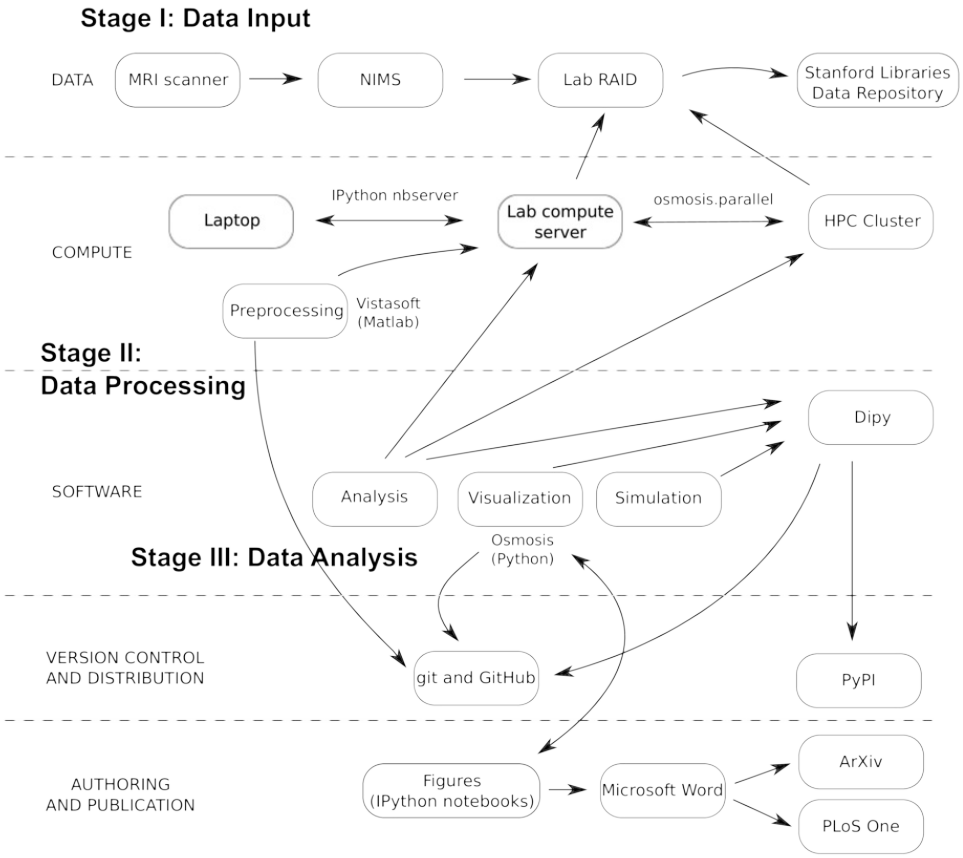
In the broadest sense, reproducibility means that I would be able to read a paper in my area of expertise and be able to run another version of the study (and experiments described within) by following the methods and protocols described within that paper. In my specific context, clear instructions would allow me to implement an experimental design/setup using identical organisms and chemical reagents. After the experiment is completed and the data are entered, I would then be able to analyze my data using models and parameters, possibly even reusing code where such methods where implemented. This would allow me to obtain results and compare them to the original.

Reproducibility in human neuroimaging research: a practical example from the analysis of diffusion MRI

Ariel Rokem

My name is Ariel Rokem. I am a Data Scientist at the University of Washington eScience Institute. My research training and experience have been mostly in the field of human cognitive neuroscience. During my postdoctoral training (2011-2015) in Prof. Brian Wandell's group, at the Department of Psychology at Stanford University, I conducted studies of human brain structure and function, using quantitative MRI. A focus of the research program that I started in Brian's lab is the application of ideas from statistical learning theory to measurements of human white matter with diffusion MRI (dMRI).

Workflow



Measurements of dMRI are used as a way to assess the structure of the human brain and its connectivity *in vivo*. Many parameters of the measurement are determined by the

experimenter, and incur trade-offs between sensitivity and signal-to-noise ratio (SNR). Models of the white matter based on different measurements are commonly used to make inferences about connectivity and tissue properties, but there was no extensive study of the fits of these models to the data, and no assessment of the effects of measurement parameters on the model fits. In the study described here, we used cross-validation to evaluate two commonly-used models in a variety of measurement conditions. The work was published in PLoS One

The project started with the collection of MRI data. Six participants were scanned in different experimental conditions. The data were collected in the Stanford Center for Neurobiological Imaging (CNI). The CNI has developed the Neurobiological Image Management System (Wandell, Rokem, Perry, Schaefer, & Dougherty, 2015), which captures the data from the scanner, archives it, and exposes a web interface that allows researchers to control access to the data, and copy it into the lab's data storage, a RAID (redundant array of independent disks) system.

The data were preprocessed using standard procedures (in the sense that any practitioner of MRI would perform these steps on his or her data). This includes correction of motion artifacts, alignment to a common coordinate frame, and tissue type segmentation. These steps were performed once, at the beginning of the study. The code that performs these steps is part of the lab code distribution, `vistasoft`, freely available through GitHub. Preprocessing also relied on freely available software from other labs.

These preprocessed data are publicly available through the Stanford Libraries Stanford Digital Repository (SDR), as two different collections: http://purl.stanford.edu/ng782rw8378 and http://purl.stanford.edu/rt034xr8593. Most of the data was licensed under the Creative Commons Atttribution license, and a small subset was also released under the Public Domain Dedication License, for unencumbered use in methods development.

Subsequent analysis was conducted on these preprocessed data, using a Python library: `osmosis`. This includes implementations of methods for fitting the data, statistical analysis, simulations and visualization, as well as utility functions to handle parallel execution on an HPC (high-performance computing) cluster. The library depends on many components of the scipy stack, including `numpy`, `scipy`, `matplotlib`, `scikit-learn`. In addition, the code depends on components of the neuroimaging in Python libraries. Approximately 30% of the module code was covered by unit tests, with a particular emphasis on core modules and utility functions that were reused. A few end-to-end tests were implemented to track regressions. Development of the software was done openly on GitHub, and it was also released under an attribution license.

Scripts using the module code were developed using the IPython notebook. These scripts were run and edited many times, and as the project evolved a few of these were copied into a documentation folder, with notebooks named "Figure1.ipynb", "Figure2.ipynb", etc, each

corresponding to a figure in the paper. In writing the paper, these figures were saved and additionally manually edited by hand to add labels and annotation, and then integrated into a Word document file, which was used to collaborate on writing with the other authors. The writing process was not versioned throughout, but several versions of the article were submitted to the arXiv preprint server, while the article underwent peer review.

Most of the computations during the development of the project were conducted on a lab multi-core compute server that was running an IPython notebook server. Thus, much of the development of the code was done on a laptop, over a web browser, connected to the server. Some procedures described in the paper would require an inordinate amount of time without the access that we had to an HPC cluster. For example, testing different settings of model regularization parameters required fitting the models hundreds of times. Data was accesible to the cluster through a mount of the lab RAID. Tasks run on the cluster were managed through a queue system (Sun Grid Engine), and a module was developed (`osmosis.parallel`) to facilitate submission of code to the cluster. These scripts could not be used as IPython `ipynb` files, and were separately invoked on the command line, but they are included as part of the code distribution, recording these steps.

The IPython notebook documenting the steps that required parallel execution includes both a 'precomputed' version (where parameters of the analysis are read from precomputed files), and 'complete' versions, which include the code that would have to be run to reproduce these results entirely on a single machine. Precomputed parameter files were not made publicly available, and would have to be recomputed to reproduce the results in these notebooks.

Though reproduction of the results in the paper could, in principle, be achieved using this library, it is not neccesarily useful as a tool for others to work with, and not easy to extend beyond the models that we tested. During the work on this project, I became involved in an open-source project, which develops Python software for the analysis of dMRI data: Dipy. The main ideas in `osmosis` were eventually ported into Dipy, accomodating the application programming interfact (API), documentation and testing requirements of that project. Furthermore, the prediction and cross-validation API implemented in Dipy that I implemented in `Dipy` is designed to be sufficiently general to accomodate new models, and mechanisms to evaluate their performance in fitting dMRI data.

Through Dipy, the code in this project is now also distributed widely through both GitHub and the Python Package Index (PYPI), under the permissive BSD license.

Pain points

One of the main difficulties encountered was the duration of some of the calculations. Some of the models, when fit on the entire brain volume, would require many hours. In particular, using the IPython notebook as a computational environment proved to be limiting, because connection to the kernel is only reliably maintained as long as the computer running the browser is kept on and prevented from sleeping. This also made it hard to perform computations that required a long duration in the notebook. One of the ways to deal with this was the development of caching mechanisms for model fit parameters. The models would be fit using a script, and the paramters cached to file. The model instantiation in the notebook would then know how to load these parameters from file, if the file existed.

Another point of frustration was that as the code in the modules evolved, code that was stored in the notebooks become outdated, and was no longer usable. This meant that as the analysis code itself evolved, new notebooks had to be written. Furthermore, as the writing and review of the article proceeded, figures were moved around in the article, and other figures got added; Figures that had started as appendices were integrated into the body of the article, etc. Thus, it might have been better to wait until the end result was an accepted article, and only then organize the entiree reproducible workflow that led to this result.

Key benefits

Though sometimes cumbersome and effortful, one of the major benefits of the process of producing a reproducible workflow is the level of confidence in the results. There is never a doubt about what code is associated with what result, because the full chain of evidence is documented in the code leading to that result.

Key tools

A specific module (`osmosis.parallel`) was developed to deal with submission of jobs to parallel execution on the HPC cluster. This module would read in a 'template' script, and then create from this template, Python script files that contained the instructions to run the fitting process with different conditions, or on different parts of the same brain. The creation of this module resulted in a highly reproducible process. Consequently reuse of elements of this module produced benefits in time-saving during the development of the analysis methods.

Questions

What does "reproducibility" mean to you?

Reproducibility is a matter of degree, not of kind. It usually depends on the availability of code and data from a scientific study, such that only a reasonable effort would be required to generate the evidence (numbers and visuals) used to support a scientific finding.

Ideally, a small number of commands at the command line would suffice, but in some complex cases, more work could be required. A reasonable amount of effort required might be rather extensive, when large amounts of data storage, or large amounts of computation are needed.

A higher standard, sometimes called 'replicability' would be to require that the same conclusions be reached if another group of researchers were to do the same experiments, and implement the same ideas in their analysis.

Reproducibility does not guarantee replicability (Leek & Peng, 2015). Some may even argue that reproducibility and replicability may sometimes be in conflict, because implementation errors can be propagated in reproduction, but not in replication (Baggerly, Morris, Edmonson, & Coombes, 2005; R. D. Peng, 2009).

Why do you think that reproducibility in your domain is important?

Human neuroscience is a field which is particularly likely to have an abundance of false findings (Ioannidis, 2005): Sample sizes are usually small, particularly in MRI, which is an expensive experimental technique. The standards of the field focus on statistical significance of effects, rather than effect sizes, which tend to be small. Though standards limiting the selection of tested relationships, and limiting the flexibility of experimental and analytic designs are starting to emerge, in practice these are not very strictly limited. Some of the aspects of the field that make it interesting and important, are also pernicious in this regard: the direct application to human health means that there is a perception of potential financial incentives. Finally, it is a burgeoning field, with many groups working on similar questions. Higher standards of reproducibility in this case would mean less false findings, because at least some of these factors would be ameliorated by a full "chain of evidence" to support every finding.

How or where did you learn about reproducibility?

Many of these practices evolved out of laziness. Early on in grad school, I learned that most analyses that are done once eventually need to be redone, and that ultimately I would have to do less work, not more, if I had a script that generated all my figures for every study that I was doing. This also evolved from being rather bad at taking notes about the work I was doing in the lab. I would need programs, and eventually IPython notebooks, just to remember what I did to get from the data to the conclusions.

A huge impact was the mentorship I got from Fernando Perez during graduate school. He was not shy about how little of the research in our field he believed to be true, and this skepticism inspired me to struggle to be more confident in my own research conclusions.

What do you see as the major challenges to doing reproducible research in your domain, and do you have any suggestions?

There are several barriers to wider adoption of reproducible research practices in human neuroscience. The first is that there is very little practical cost to not being reproducible. As mentioned above, there is likely to be a large proportion of false results in the neuroscience literature, and it's more likely to be false if it's not reproducible. Since a false positive result is more likely to result in a publishable unit, there seem to be incentives in place to not be reproducible, slowing down the progress of the entire field.

What do you view as the major incentives for doing reproducible research?

The level of confidence that I have in my results is quite high. That helps me sleep well at night.

Would you recommend any specific resources for learning more about reproducibility?

There are several papers that provide guidelines for reproducibility with a specific focus on neuroimaging. Two recent examples include Gorgoloewksi & Poldrack (2016) and Pernet & Poline (2015).

References

Baggerly, K. A., Morris, J. S., Edmonson, S. R., & Coombes, K. R. (2005). Signal in Noise: Evaluating Reported Reproducibility of Serum Proteomic Tests for Ovarian Cancer. *J Natl Cancer Inst*, *97*, 307–309.

Gorgoloewksi, K., & Poldrack, R. (2016). A practical guide for improving transparency and reproducibility in neuroimaging research. *J Natl Cancer Inst*, *14*(7), e1002506. http://doi.org/http://dx.doi.org/10.1371/journal.pbio.1002506

Ioannidis, J. P. A. (2005). Why Most Published Research Findings Are False. *PLoS Med*, *2*(8), e124. http://doi.org/10.1371/journal.pmed.0020124

Leek, J. T., & Peng, R. D. (2015). Opinion: Reproducible research can still be wrong: Adopting a prevention approach. *Proceedings of the National Academy of Sciences*, *112*(6), 1645–1646. http://doi.org/10.1073/pnas.1421412111

Peng, R. D. (2009). Reproducible research and Biostatistics. *Biostatistics*, *10*, 405–408.

Pernet, C., & Poline, J. B. (2015). Improving functional magnetic resonance imaging reproducibility. *Gigascience*, *4*(15). http://doi.org/http://dx.doi.org/10.1186/s13742-015-0055-8

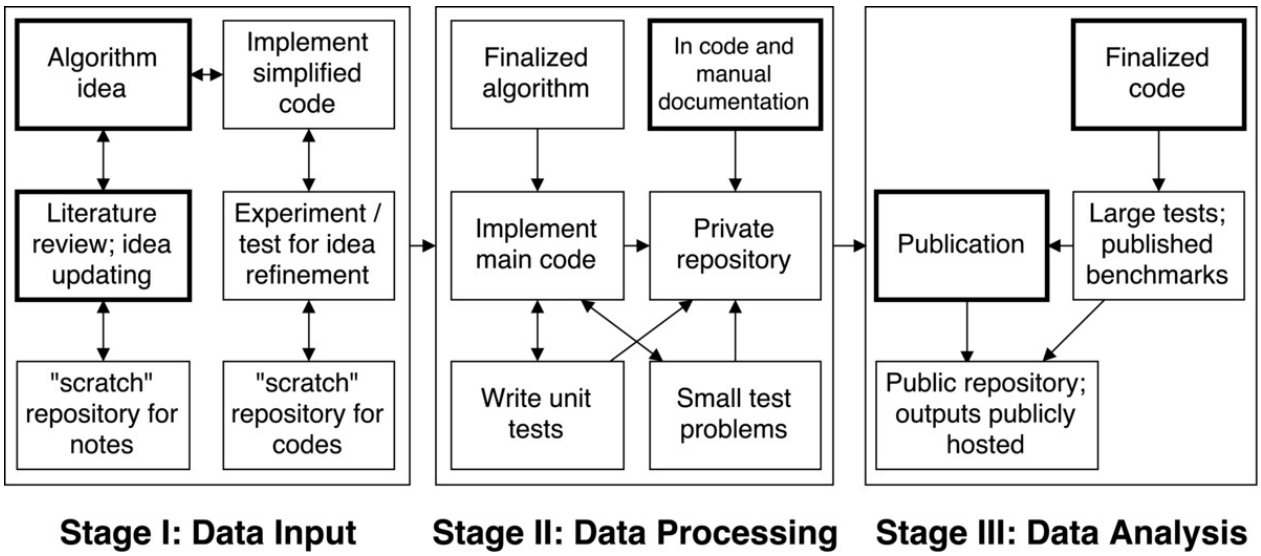
Wandell, B. A., Rokem, A., Perry, L. M., Schaefer, G., & Dougherty, R. F. (2015). Data management to support reproducible research. *ArXiv*, 1502.06900v1.

Reproducible Computational Science on High Performance Computers:

Rachel Slaybaugh

My name is Rachel Slaybaugh and I am an Assistant Professor in the Nuclear Engineering Department at the University of California, Berkeley. I study computational methods for neutron transport: numerical methods for solving the Boltzmann equation applied to neutral particle interactions. The methods I study are both deterministic (e.g. finite difference, etc.) and stochastic (Monte Carlo). I develop these algorithms for reactor design and analysis, radiation shielding, and nuclear nonproliferation applications. Much of my work has an emphasis on high performance computing. (Tagline: intersection of applied math and computational science, informed by nuclear engineering)

Workflow



I tend to think of my workflow for code development as having three fundamental steps: (1) idea generation and refinement, (2) idea implementation and testing, and (3) large scale testing and publication.

Step 1: The starting point of a new project is the development of an algorithm. This comes from a combination of reviewing literature, discussion with colleagues, familiarity with challenges in the field, and so on. As I refine an idea, I find I need to review more literature; as I research the literature, I refine the idea. The algorithm development tends to be collaborative as it is based on discussions with others, but the literature review tends to be independent. I like to write notes while reading papers in one large LaTeX document and

keep that document in a repository with all of my other notes and reviews so all of my notes on a given topic are in one place and I only have one place to look for things I have researched in the past.

Next, I implement a simplified version of the algorithm to make sure that it works at all. For example, I would implement a 0D or 1D version (as opposed to 3D) of a method quickly and simply in Python to use for testing. In this step there can be iteration between the algorithm idea and the test code, informed by additional literature review as necessary. Once satisfied with the experiments with the simple code, the algorithm is considered "final" (though of course it can be adjusted later if needed). I am not sure that this part of the workflow is reproducible in the sense that the process could be exactly replicated, but version controlling everything makes it possible to recover intermediate steps, which in some ways allows the idea refinement to be traced.

Step 2: Once there is a finalized algorithm, it gets implemented in the "real" code that has multiple developers and is written in a compiled language like C++. The repository for the code is typically private because, as mentioned, these codes are not completely open. It is often the case that only one or a very few people are working on this idea, so we make a branch and do the development there. I add unit tests to a testing framework associated with the code as I go (for example GoogleTest); the tests reside in the same repository as the main code. As the code approaches completion, I use small "system" test problems to investigate basic system functionality: does the code get the correct answer for analytical/known solutions? what does basic performance look like? etc. The small tests are also version controlled--either in the same repository as the source code or in a separate one.

Once the unit tests are deemed sufficient and, combined with the small tests, everything indicates that code is correct, I finalize documentation. Throughout development I typically use Doxygen to comment the code. Doxygen automatically generates documentation from source code comments when those comments are made using particular, simple annotations. Doxygen works for languages like C++, Python, Java, and others. Using Doxygen is useful for creating an application program interface quickly and easily. However, some extra work is often required to get the theory down and provide clearer directions for using the new algorithm. All of that is written in LaTeX for incorporation into the user and/or theory manual. The documentation LaTeX files are version controlled, often in a separate documentation directory. At this point the code will be reviewed and merged into the main code base. Once the code is finalized, the unit test and small test results should all be reproducible by the other developers--the people who have access to the developer repository.

Step 3: Once there is "finalized" code, it is time to do the real demonstration testing for publication. This involves running large test problems that demonstrate performance of the new algorithm for problems of interest as well as comparison to benchmarks to demonstrate correctness. The literature review, algorithm description, and results of the large (and sometimes small) tests will go in a final LaTeX document for journal submission. Recording of work for journal publication will also be version controlled, typically in a public GitHub repository. The idea is that, beyond the text writeup, the large test input files will be in the same repository as any scripts required to process data and generate plots, all with corresponding directions. Thus if you have access to the code and the repository with tests, scripts, and results, you can rerun all the calculations and process the data.

Pain points

There are a few pain points: An annoying one is getting the documentation right. It seems like just using Doxygen is not enough. To get something that really is user-manual quality you have to write a lot of things twice, just slightly differently. I try to reuse as much as I can, but if things are replicated there is the challenge of maintaining consistency.

A tough one is ensuring that the version of what is released in the end is actually reproducible. This requires the extra step of documenting which version of the code was used (the results should not change in the future, but it is better to have the version clearly written just in case). In principle one can figure this out from the repositories, but if everything isn't stored together that becomes more challenging. Providing directions about how to run everything and which versions of third party libraries were used is also some extra work.

A final pain point is re-implementing the algorithm from the simple case to the complex case, since the simple code is never really used for anything. However, this is a pretty small issue since the toy code didn't take long to develop.

Key benefits

The largest benefit of this approach is having confidence in the validity of the data that you publish. For me that confidence starts with implementing the methods and their tests at the same time. I think everyone should have a unit testing system; it is difficult for me to see how one could have confidence in the correctness of their software without one. I get very nervous about using code that I write if I haven't written tests to go with it.

Having an up-to-date application programming interface is also very useful. When I'm interfacing or working with a piece of code I wrote a long time ago I would not otherwise remember what it did or how to use it. It is also helpful when interfacing with parts of the program other people wrote. This extends to proper documentation. I personally can't

remember many things. I must write them down for future reference. Keeping a user and theory manual means not only that users and other developers will know what the code does and why, it means next week I will also know what the code does and why.

I also find that having little bits of experimental code, the low-dimension test pieces I write at the beginning, are useful to have on hand. This does not particularly impact reproducibility, but it is useful to have chunks of code to start with when playing around with new ideas. Similarly, having a repository with literature review notes is good for remembering past research, speeds up writing papers and documentation, and provides a place to start looking the next time.

Key tools

They key tools I use are Doxygen, git (for version control), LaTeX, and plotting and data manipulation tools (usually in Python).

Questions

What does "reproducibility" mean to you?

The first way I think of reproducibility is "can I/my lab reproduce the results in my paper exactly?" After that, "can an independent researcher, given that they have legal access to the required data and software, reproduce the results?" Nuclear engineering data and codes are often controlled, so for many projects only researchers within my field will have access to the required data and software. Fortunately, such non-open-source codes are typically available at no cost to researchers through a simple licensing process.

Why do you think that reproducibility in your domain is important?

The codes that we write are often used to investigate new nuclear systems and make long-term policy or design decisions based on the results. They are also used to study existing nuclear systems. This is important stuff; the codes need to be right and the results need to be verifiable.

How or where did you learn about reproducibility?

- Mentors: my PhD advisers valued reproducible practices and insisted that we used them
- Student groups: the Hacker Within

- Practice: taking on a project that used good practices was how I really learned many of these skills
- Community exposure: spending time with others who value reproducible practices
- Self-study: looking up things I saw people do that looked helpful

What do you see as the major challenges to doing reproducible research in your domain, and do you have any suggestions?

The biggest challenge is legal: only some people can access the codes and data required. A secondary challenge is access: some of the work I do requires high-performance computing that is not readily available to many.

What do you view as the major incentives for doing reproducible research?

Ethical mandate: I want my work to be right and for others to be able to know that it is right.

Impact: My ideas and products might then be adopted and built upon.

Are there any best practices that you'd recommend for researchers in your field?

Testing, testing, testing.

Would you recommend any specific resources for learning more about reproducibility?

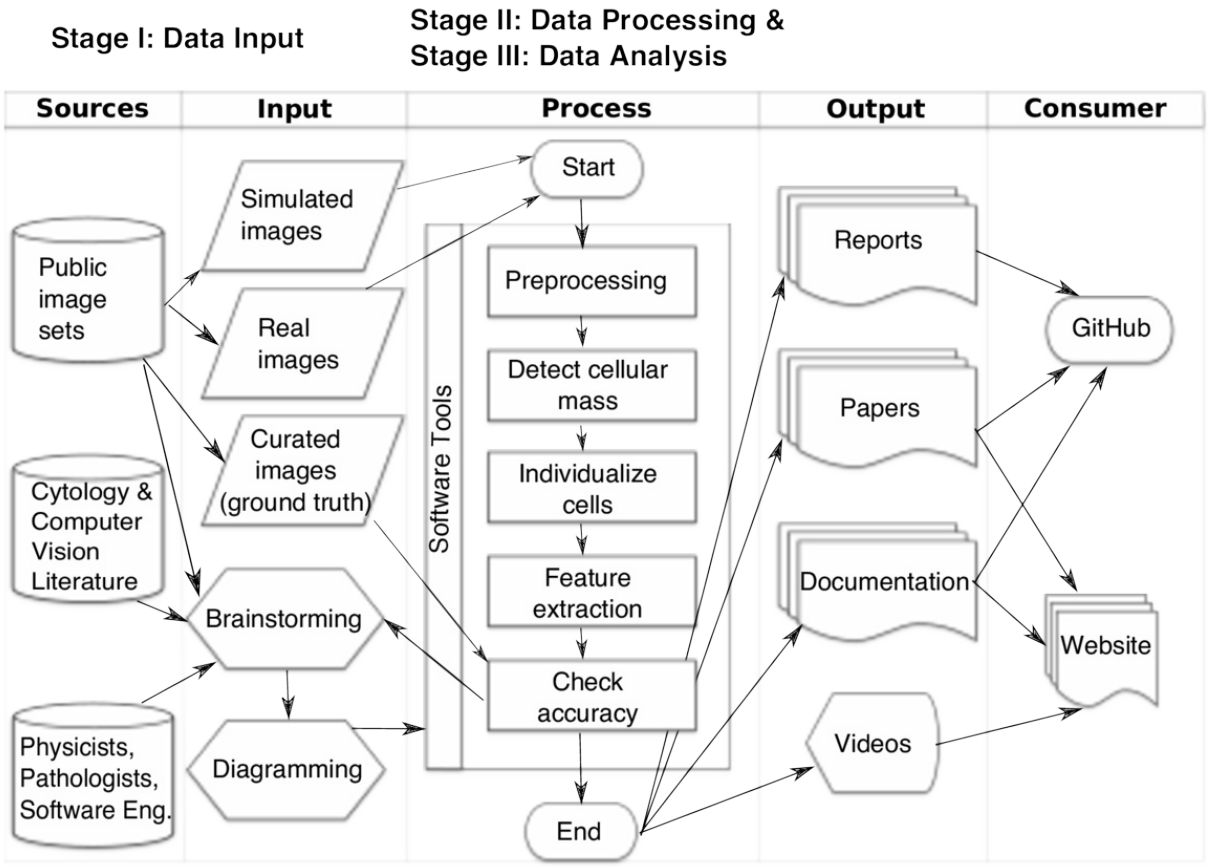
Software Carpentry; the new Scopatz-Huff book.

Detection and Classification of Cervical Cells

Daniela Ushizima

My name is Daniela Ushizima, I am currently a Staff Scientist at the Lawrence Berkeley National Laboratory and a Data Science Fellow for the Berkeley Institute for Data Science at the University of California, Berkeley. Much of my research work is in devising machine vision and pattern recognition algorithms as part of software tools for handling image-centric data, especially those arising from the Department of Energy imaging facilities. The case study I describe here illustrates the core steps in designing a machine vision algorithm to analyze a set of digital images and organize them according to the desired criteria. There are several image processing and analysis frameworks that encapsulate algorithms; this case study refers to ImageJ, a powerful image analysis tool.

Workflow



The workflow diagram follows a data model called SIPOC, which stands for suppliers, inputs, process, outputs, and customers; these correspond to the columns of the table. Here, we adapt SIPOC diagram to better represent our use-case, hence the first column is called sources and the final column represents repositories. The proposed workflow prioritizes the compartmentalization of different processing steps of the analysis system, and hides potential feedback loops that might occur.

This diagram tells the story of research investigations among doctors, pathologists, cytologists, and computer scientists, aiming to design, develop and deploy algorithms for improving the analysis of biomedical images. Some of the tasks include increasing the number of fields under scrutiny, speed up cell counting and recognition, comparison among cells, quantitative description of samples, to name a few.

Historically, this use-case began with brainstorming among pathologists, physicist and software engineers on how to provide scalable computer-aided analysis to the acquired large datasets of biomedical images, containing cervical cells. Communication and diagraming were fundamental *sources* of information to understand how to well characterize types of cells and foresee limitations imposed by the datasets, such as cervical cell lineages, cell fragments, magnification and usable area within the sample.

In order to develop analysis pathways, the team organized the datasets into three main *input* image sets: simulated, real and curated. A simulated image consists of several clipped real cells collated with different levels of overlapping, which facilitates algorithmic validation since the ground-truth is known a priori. A real image consists of a digital picture of a Pap smear slide, obtained at the light microscope -- these images often contain several types of cervical cells and other findings (e.g. bacteria, blood), and may be corrupted by noise and other artifacts, such as staining variations, dirt, hair, etc.

The core of this case study lies in the *process* column that illustrates the main steps in the image analysis. The preprocessing step transforms the samples into more compact and reliable representations of the image, for example, removing areas or eliminating the whole image if it is over-stained. This step includes essential image transformations, such as anisotropic filtering that preserves borders and smooths regions that compose a supposedly homogeneous image partition.

As part of the analysis, the software tool must detect the regions of the interest in each image, i.e., the cellular mass and the rest of the image. This step took time to evolve into an algorithm that could correctly split the image into foreground and background. The next step is to separate the cellular mass into individual cells: by modeling simple biological prior knowledge, such as the relationship between nucleus and cytoplasm, the algorithm is able to quickly estimate cytoplasm boundaries. After identification of the cells, feature extraction takes place, including nucleus-cytoplasm area ratio, convexity of cytoplasm contour, and other parameters that are relevant for identifying cell lineages. Finally, we use simulated and curated datasets to validate results, for example, considering sensitivity and specificity measures based on the number of pixels or the number of cells identified.

An important step of the data processing is keeping track of the *outputs*. The fourth column lists four main outputs of the system: technical reports, scientific papers, documentation about the software tool and educational material about the science problem and algorithm development. Although the diagram omits output of partial results, they are very common and useful throughout the design of the analytic software tools.

The fifth column shows the different outputs being archived in *repositories* to enable access to the research discoveries, for example GitHub and websites. In the context of this case study, it indicates the main hubs of information distribution for the project.

Pain points

The software tool design and testing require intense communication among the team members through reports and presentations. Although part of team used version control, much of the code is still to be made available open-source through GitHub. In addition to commit messages, which tend to be short, we have also maintained an electronic diary of

activities -- these are fundamental to keep the whole team synchronized and up to speed. The painful side of such an electronic lab book has been the unstructured format of the inputs that may require extra-time to parse.

Key benefits

The most reproducible part of this project has been the development of code allied to simulated datasets. This activity improved across the team, particularly due to the participation in code competitions, which forced the whole team to organize data sources and code repositories such that reviewers can quickly reproduce the results. In addition, keeping track of advancements in a common digital lab book helped in preparing manuscripts and other technical reports.

Key tools

An important tool has been ImageJ, a Java-based image processing software program, which was originally developed by Wayne Rasband at the National Institute of Health circa 1997. Although most of the ImageJ plug-ins focus on medical imaging, this framework has been widely used in other applications, such as material sciences.

Questions

What does "reproducibility" mean to you?

In the context of this case study, the work will be computationally reproducible when the software tools our team builds can also be used by the science domain experts, e.g. pathologists, who should be able to transform raw data files into quantifiable patterns, obtaining consistent results with previous/tested analyses. Because algorithmic parameters often change from a dataset to another, it can be challenging to get results with the same accuracy, given different datasets.

Why do you think that reproducibility in your domain is important?

Reproducibility is essential in quantitative microscopic because it can guarantee accurate measurements and improved quality control.

How or where did you learn about reproducibility?

Flow charts and code documentation improved grades during college, and their absence meant dire punishment: reproducibility was a time-consuming protocol to get good grades and spend extra ribbon cartridge with my dot matrix printer. The tech-industry introduced me to version control, and TortoiseSVN and I started working together with several colleagues. Other tools came along the way, such as git and Atlassian tools to enhance usability and extension of the codes. After entering BIDS, reproduciblity turned into a fun activity, a conversation starter and a never-ending code improvement process. Lots of concepts came together more systematically, particularly after attending Software Carpentry classes and becoming a instructor myself.

What do you see as the major challenges to doing reproducible research in your domain, and do you have any suggestions?

The major pitfalls of doing reproducible research in image analysis have been (a) the lack of domain-specific image examples that illustrate applicability of the algorithms, (b) dependency of packages that are not freely available, and (c) absence of documentation that enables understanding how/why the data transformations occur.

What do you view as the major incentives for doing reproducible research?

The major incentives for doing reproducible research are the ability to replicate the experiments later, to fix and/or reuse code for different applications, to easily work in larger teams, and the potential for a broader impact, even with the help of collaborators you have never heard of.

Are there any best practices that you'd recommend for researchers in your field?

While there are no general rules, some tools can only help one to reproduce work: (a) use version control, (b) practice software quality assurance, (c) organize data samples separately from the code, but linked to it somehow.

Would you recommend any specific resources for learning more about reproducibility?

Among the several options out there, Software Carpentry is certainly one of the best resources and the book Making Software by A. Oram and G. Wilson.

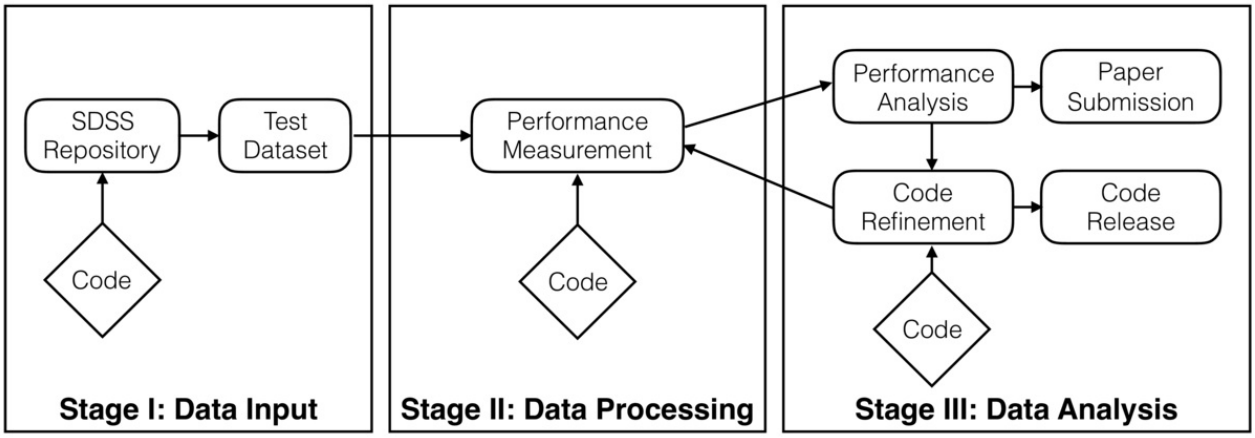
Enabling Astronomy Image Processing With Cloud Computing Using Apache Spark

Zhao Zhang

My name is Zhao Zhang, I am a joint postdoctoral researcher at AMPLab and Berkeley Institute for Data Science, University of California, Berkeley. The theme of my research is to enable data-driven science with computer systems.

This case study describes the process of building Kira, a distributed astronomy image processing pipeline in the cloud environment. The idea of the Kira project is to explore the applicability of cloud computing based software stack in supporting scientific applications. Specifically, we use the SEP (Source Extraction Python) library for domain computation. We choose Apache Spark and Hadoop to build the infrastructure of distributed processing and data storage.

Workflow



We use LaTeX and Slides to track the merit evaluation: why do we need a new system for astronomy image processing, what makes it a better system, and what lessons we can we learn from this research.

We use a private GitHub repository to keep track of solutions for technical barriers such as I/O processing, Spark interaction with C program, Spark system parameter configurations and many others.

The whole system is built with multiple programming languages and tools. At the programming language level, we use Scala, Java, Python, Bash, and C. At the system level, we use Spark for task coordination, HDFS for persistent storage, and the SEP library for actual computation.

The source code of the project is kept in a public GitHub repository to make it open source.

The manuscript is being kept in a private GitHub repository since it is under review.

In system design phase, we decided to use three datasets four development, testing and performance measurements. But we end up with four datasets. A trivial dataset that contains only a few image files is used for development and testing. A small dataset (12GB) is used

for quick verification at scales. A large dataset (65GB) is used for large scale performance measurement. A fourth dataset (1TB) is used to show the data processing capacity of the Kira system as we put up the paper.

All these datasets come from the Sloan Digital Sky Survey. Some of them are from Data Release 2 while some are from Data Release 7. We choose them arbitrarily as we care more about the system capacity rather than the science in this research.

Our collaborators are: Kyle Barbary, Oliver Zahn, Saul Perlmutter are astronomers. Frank Nothaft, Evan Sparks, Michael Franklin, David Patterson are experts about Spark and cloud computing in general. Zhao Zhang has rich experience in HPC community and some experience in cloud computing as well as a bit astronomy background.

We use private GitHub repository for manuscript management and public GitHub repository for project management.

We have summaries for Team Brainstorming and Merit Evaluation phase. System Design's output is in the form of figures and is kept in GitHub repository. Solutions for Technical Barriers are kept in a private GitHub repository. Documents, Source code, system configurations as the products of coding/testing/tuning/measurements are kept in a public GitHub repository. The paper draft is kept in a private GitHub repository.

Before explaining the details of the diagram, I will first briefly review the software and systems that are used in this case study.

- FITS (Flexible Image Transport System) is a widely adopted image format in the astronomy and cosmology community. It is a fixed format with the image metadata as text and the actual image as binary format.
- SEP (Source Extraction Python) is the software that detects light source objects from images. It rewrites the SEXtractor software by exposing primitive functionalities through a library interface with both C and Python.
- Apache Spark is a popular distributed computing framework in cloud computing. It offers implicit parallelism and the lineage-based fault tolerance through the Resilient Distributed Dataset (RDD) abstraction. Spark is built using the Scala programming language which compiles a program that is executable on Java Virtual Machine (JVM).
- JNI (Java Native Interface) provides a method to call existing C libraries inside a Java/Scala program. C and Java/Scala data structures can be used to exchange information between the two runtimes.
- Amazon EC2 (Elastic Compute Cloud) is a public cloud service provided by Amazon. Users can request a number of compute nodes with various hardware and software combination.

- Amazon S3 is a data storage service provided by Amazon. Users can host there dataset on S3.
- NERSC (National Energy Research Scientific Computing Center) is a high performance computing facility operated by Lawrence Berkeley National Lab. It hosts a few supercomputers and clusters.
- SDSS (Sloan Digital Sky Survey) is a large scale sky survey, its data is publicly available online.
- Thread safety is an operating system concept that describes the concurrent execution of multiple threads safely manipulating shared data structures.

The process begins with team brainstorming of how modern computer software and hardware can accelerate the astronomy image processing pipeline. This requires a wide and also deep understanding of the state-of-the-art research and technical solution. In this research, we gather domain expertise (astronomers), cloud computing expertise and high performance computing expertise. We review the existing work and we think using cloud computing software-hardware stack can improve the overall application performance, but we have no idea by how much it can improve. The research is an exploratory process to implement the idea and quantitatively measure the improvements if there is any.

The Team Brainstorming and Merit Evaluation phase happened back and forth as we keep asking why are we building such a project. Detail questions include: What are the existing solutions? How does the new project make difference in terms of performance and usability? Who are the potential users? This procedure lasts for about two weeks, all members of the Kira project are involved in the discussion. The pros and cons of each existing solution was documented, and later used in the paper.

The System Design phase lays out the programming interface of Kira, the modules and interactions between the modules. In this phase, we also identify some technical barriers of this project. I am listing them below, feel free to contact me if this is hard to understand:

1. Kira I/O, how to make Spark read FITS images
2. Calling C library in Spark, how to make Spark work with existing C code in the SEP library
3. Setting up compilation environment, set Maven to automatically build Kira

As we progress with the code, we notice a few other technical barriers:

1. Thread safety, neither the jFITS library nor the SEP library is thread safe
2. Load imbalance, scheduler tuning for this particular workload

For each of these technical barriers, we seek solutions for them. The solutions come from three sources: colleague expertise, google, and documents. By isolating the barriers, we were able to focus on a single barrier each time and can quickly verify the solution. The resulting code is stored in GitHub, and later merged into the project. This process takes about two weeks.

The Software Coding and Testing phase takes about three weeks, we managed to integrate the SEP library through Java Native Interface with Spark, thus finally implement Kira. I implemented the code, and wrote the documents to make it convenient for my self to repeatedly run experiments. In the meantime, I prepared four datasets for performance measurements. A 24MB (4 files) dataset for sanity check, a 12GB (2,310 files) dataset for small scale test, a 65GB (111,50 files) for medium scale test and a 1TB (176,938 files) for large scale tests. The datasets were initially stored in NERSC shared file system, later I made a mirroring on EC2 S3 service, as most experiments were run on EC2 where S3 has a better transfer bandwidth to.

Performance Measurements and Performance Tuning come in pair and we go back and forth frequently. The key thing in these two steps is that we need a reasonable expected performance before the measurements. If the measurement does not match with our expectation, we need to analyze the reason and tune the system. Our methodology is like this: we started on 1 core on a single machine. We compare the Kira performance against the equivalent implementation to understand the slowdown introduced by Spark and JVM. Then we started to scale up with more cores on the same node, and observe the scaling curve. By doing that we understand the bounding factor of the performance on a single node. Later on, we scale out on multiple nodes by doubling the number of compute nodes in each step and observe the performance scaling. Since Spark hide the scalability complexity in the system, all we need to do here for different scale is to set relevant parameters in the configuration files. The code and documents are kept in GitHub, and the dataset is kept in Amazon S3 service.

With all of the scripts from Merit Evaluation, System Design, and Source code, we put together the paper. Writing the paper is a collaborative process. We used a private GitHub repository to host the paper, and using Pull Request to manage everybody's editing.

Pain points

1. (reproducible results) For the results to be reproducible, the readers should be able to tell and access the computers with the same hardware, the code base used particularly for the experiments, the dataset that was used for performance measurements.

1.1 Hardware access. Since we are using Amazon EC2 resources, the same computer hardware is mostly accessible unless Amazon upgrades the hardware. It happens every few years. A second risk is that for large scale test, the reader might contact Amazon to increase the hardware limit which Amazon uses to limit the quantity of resources each user can posses at the same time.

1.2 Code base. We maintain our code under a public GitHub repository, so it is accessible to all. However, the pain point is that the software evolves and the performance might change with the software evolution. Thus it is important that the authors should let the readers know which version of the software is related to the results that readers care about.

1.3 Dataset. The astronomy image dataset we use is Sloan Digital Sky Survey Data Release 7, which is publicly accessible. As long as the data hosting service is up running, the dataset is available. We also make a copy of the dataset we used in Amazon S3 service with public accessible permission. The pain point is that we have to pay Amazon for maintaining the 1TB dataset, and eventually we will run out of funding, so instead we have to publish the dataset file list as a text file in the code base.

Key benefits

I break this question down to two: non-usable workflow and non-reproducible workflow.

1. Non-usable workflow. I have seen a couple of projects in astronomy, where the authors conducted study on applying new tools to solve the old problems, but the authors failed to publish their source code along with the paper. This gives up the opportunity for people to build solutions upon their work. For Kira, we make the source code available on GitHub, so people can extend this code base for more functionalities.
2. Non-reproducible workflow. There was one experiment I read in a paper that I would like to reproduce, and design a new solution for it. However, the experiment was not reproducible due to the software version evolution. During the Kira building process, we particularly care about this issue, we documented the hardware, code base and dataset that are used for the performance measurement, so any user that follow the documented instructions should be able to reproduce the results.

Key tools

GitHub for code management, and Amazon S3 service for data hosting. We built Kira with Apache Spark which is a highly active open source project, so that we do not have the concern of the computing framework is out of maintenance if our academic funding ends.

Questions

What does "reproducibility" mean to you?

In the context of my case study, reproducibility has several levels of meanings. The very baseline is that users can compile the source code and pass the tests. Secondly, users should be able to configure the computer cluster so the can reproduce the performance in the documents. Since it is impossible to reproduce the exact performance measurement for every single run, a statistical repetition should be fine (average performance with bounded variation). Generally for computer system research that involves data, a public available data source is necessary for the performance to be reproducible.

Why do you think that reproducibility in your domain is important?

As computer system researchers, we build systems assuming people will use them. So it is important that people can follow the instructions in the documents to reproduce the state in which the system works. And it is important that users can reproduce the improvements over existing systems we describe in the paper or documents so they are more likely to adopt our systems. As paper reviewers, it is more convincing if they can reproduce the results in the paper as these are the evidence of the paper's idea.

How or where did you learn about reproducibility?

I learned the reproducible practices since the first time I submitted my homework project in college and ever since. I need to write a README file along with my code so the teaching assistant could compile and run my code to test if my solution is right. The later research experience follows the same path.

What do you see as the major challenges to doing reproducible research in your domain, and do you have any suggestions?

I used to design systems on supercomputers, where many people including paper reviewers have no access to. Thus, it is impossible for the research to be reproducible.

Another major pitfall is due to software version evolving. At the time of writing the paper, some features of a piece of software was working, and the researchers measured and published the numbers. But these numbers are no longer reproducible after a few versions.

What do you view as the major incentives for doing reproducible research?

I break this down to reproducible results and reproducible research process.

4.1. (reproducible results) The systems I build are usually to facilitate scientific research. The systems either expedite the execution of computer programs or provide novel functionalities (e.g. failure diagnosis). To make sense about my research, users should be able to see the performance improvement I documented in the paper or documents. They should be able to use the novel functionalities to ease their research. So reproducibility of the systems is the key to prove the system actually works.

4.2. (reproducible research process) As a whole process, this particular research case is exploratory. We only have a conjecture about the performance before the implementation and measurement. I am not familiar with the tools I am using also (Apache Spark, Scala, Java Native Library, SEP library, Source Extractor C program). I think (not quite sure) the incentive for the reproducible research progress is helpful in my future projects. Once again, if I am facing such situation, I know where to start to tackle a complicated problem. My methodology particularly for this research is: 1) a more-or-less valid hypothesis, 2) a performance profile of the existing solution, 3) isolating the technical barriers, 4) solving the technical barriers, 5) build the new solution, 6) performance measurement and tuning.

Are there any best practices that you'd recommend for researchers in your field?

I would recommend for open source software and related publications. The authors should maintain a version of the software for readers to reproduce the results in the paper. These versions and repository should be included in the paper.

Would you recommend any specific resources for learning more about reproducibility?

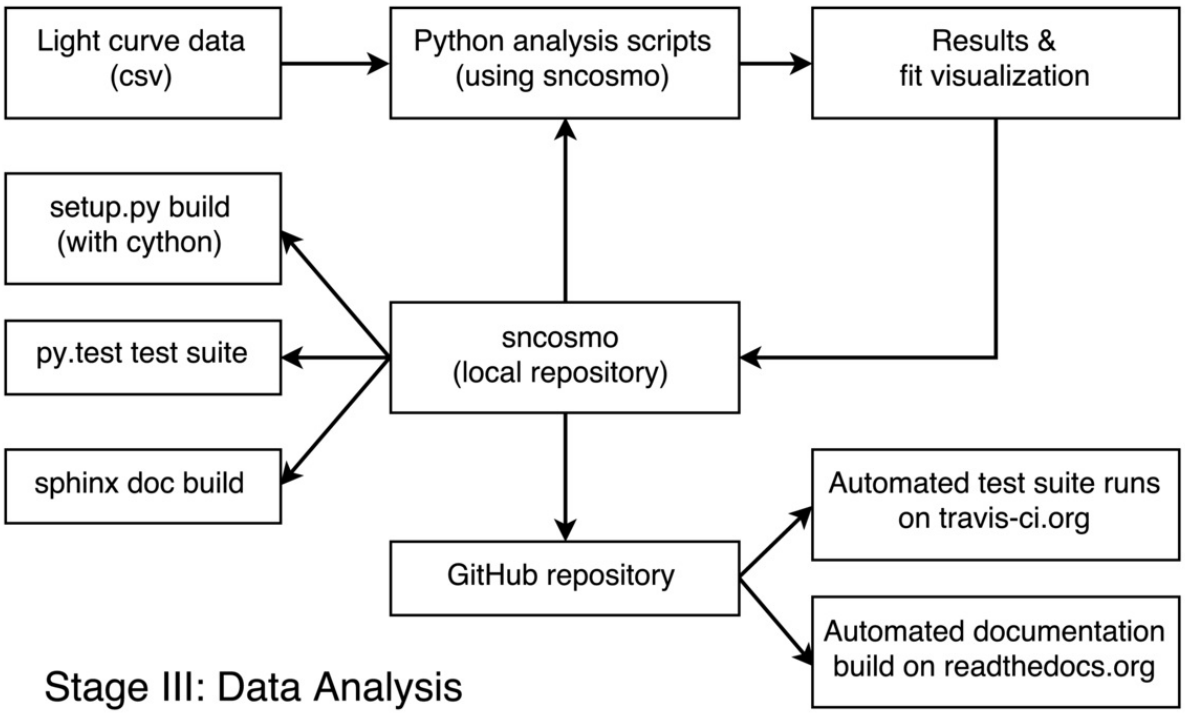
I can only think of GitHub for code management and Amazon S3 for data management right now.

Software for Analyzing Supernova Light Curve Data for Cosmology

Kyle Barbary

My name is Kyle Barbary and I am currently a postdoc in the physics department and a Data Science Fellow in the Institute for Data Science at the University of California, Berkeley. I am an observational cosmologist. More specifically, I use a particular variety of exploding stars, known as Type Ia supernovae, as markers to measure how the universe has expanded over its history. To make this measurement as precisely as possible, it is necessary to combine supernova data from many different surveys targeting different distances. The workflow I describe is about the creation of software tools used to combine and analyze that data in a uniform way.

Workflow



Stage III: Data Analysis

I will describe the development of software for analyzing supernova light curve data. A "light curve" in the parlance of my domain is simply the brightness of a supernova as a function of time. These brightness measurements are derived from images of the same patch of sky spaced in time, ideally showing the supernova growing brighter and then fainter. Analyzing these light curves is a key step in deriving final results for most supernova cosmology

studies. The software in question was originally developed for analyzing data from the Dark Energy Survey, but it can be (and has been) used for analyzing data from other surveys, as I will discuss below.

The analysis starts from reduced light curve data produced by a separate pipeline (not discussed here). A Python script reads the data, performs analysis tasks such as model fitting or parameter sampling, and saves the results or produces plots allowing the user to visualize the results. There are generally multiple scripts for performing different analyses or variations on an analysis, and these can be written by several different scientists on the project. The key aspect of the process is that all commonly useful functionality is split out into a Python *library* (SNCosmo). The top-level analysis scripts contain logic specific to the analysis and to the survey, and the SNCosmo library contains functionality applicable to a variety of surveys and analyses.

The development of the SNCosmo library itself is an iterative process where features of the library are added or refined in response to the needs of various analyses or users. Although there are official release versions of the library, several users stay up-to-date with the development version to keep this feedback loop tighter.

We use git for version control of the library and GitHub to coordinate development, where work is centered around an "SNCosmo" GitHub organization. Users who follow the development version periodically pull changes from the copy of the repository owned by the "SNCosmo" organization. We use two services in conjunction with GitHub. First, continuous integration is done with Travis: every time a change is made to the GitHub repository, this service is triggered. It builds the library and runs the full suite of unit tests for multiple combinations of supported library versions. This allows the developers to catch and fix problems before they are reported by users. Second, automated documentation builds are done by Read the Docs. This service builds the library and runs the documentation builder which produces a set of HTML pages (and also a PDF with the same content). This allows users to see the documentation for the latest development version immediately if needed. These two services are free for open-source projects and are widely used.

Within the repository, we use a number of standard tools: there is a `setup.py` script which can be used to build the library via `setup.py build` or to run the tests using `setup.py test`. The `py.test` package is used internally to run the tests.

Finally, at some point we make an official release version of the library. This is typically done after features have been user-tested for some time and the API is stable enough to be supported in future release versions. This is often a difficult judgement call.

Pain points

- **Feature stability:** There is a trade-off between adding some feature immediately versus waiting until it is obvious whether to include it and what the specific interface should be. In the past I've marked such features as "experimental" with a warning in the documentation that users might have to change their code in the next library release version.
- **Multiple platforms:** I develop on Linux but most users are on Mac OS X day-to-day. This hasn't been a huge problem yet, but it has produced a few headaches. Automated build services are starting to support OS X for free, so this will help.

Key benefits

The separation of common software functionality into a *library* is surprisingly unique in this subfield of supernova cosmology. It is a boon for reproducibility: published results can include the (relatively short) analysis scripts that were used, along with the version of the SNCosmo library used. The fact that the core software is a well-documented library means that readers and practitioners can more easily understand the specifics of the algorithms used.

Questions

What does "reproducibility" mean to you?

To me, reproducibility has two facets: the availability of usable software (preferably under an open-source license), and the availability of data (preferably in both raw and reduced forms). Together, these should give an outsider the ability to reproduce the results of a study from start to finish.

I separate these two aspects because each can be beneficial without the other. For example, even without releasing data, it can still be quite beneficial to release software. If released under an open-source licence, this provides a different flavor of reproducibility - the ability to reproduce an algorithm described in a paper and use and improve that algorithm in subsequent work.

As a side note, in my domain we often settle for a weaker form of full reproducibility, where a "reduced" data product and the software to analyze it is released, but not the raw data and not the software to go from raw to reduced data.

Why do you think that reproducibility in your domain is important?

Efficiency. Reproducibility makes cosmology research more efficient in the following ways:

- Reuse of code. Cosmologists are as guilty as any of reinventing the wheel, particularly when the blueprints for the wheel are not made available.
- Better understanding of algorithms spreads more rapidly. Algorithms are often explained coarsely in papers but without the detail necessary to reimplement them. Allowing the reader to directly read the code (if desired) solves this problem.
- Fewer unexplained conflicting results. Research is often held up or lead down the wrong track by conflicting results from multiple groups. Allowing different groups to reproduce each other's results will help resolve such situations more quickly.

How or where did you learn about reproducibility?

Mainly through working on the AstroPy project, which develops a community astronomy Python package. I got involved in AstroPy when it was started in 2011. Like many other large open-source projects, AstroPy is developed on GitHub and follows typical best practices such as extensive unit testing, automated documentation builds and continuous integration on multiple platforms. In short, I learned these practices by interacting with more experienced programmers also working on the project.

What do you see as the major challenges to doing reproducible research in your domain, and do you have any suggestions?

In astronomy, like other fields, observers have a desire to carefully guard their hard-won data until they have eeked out every possible analysis. I'm sympathetic to this; acquiring the data often requires designing, building and deploying a new instrument or even an entire telescope. It can be a very large fraction of the work that goes into a project. The threat that someone else will download your data and use it to publish a result that you could have published is very real.

I'm less sympathetic about the reluctance to release software. Some of the reasons that I've experienced:

- perceived lack of quality
- perceived extra work to clean it up, maintain and support it
- perceived competitive advantage or that the software is an asset or bargaining chip

Even for those who do wish to release their software under an open-source license, it is often difficult to do so in a fully legal manner through "official" channels due to university or lab copyright. Often, scientists just release the software without official permission.

Finally, one technical issue with releasing data is data volume. Raw imaging data from an entire survey can be many terabytes. Making this data publicly available often requires dedicated servers and support staff.

What do you view as the major incentives for doing reproducible research?

- **Long term project efficiency:** Projects are often carried out over multiple generations of grad students and postdocs. Doing things reproducibly within a collaboration makes the transition between generations much less lossy.
- **Ability to back up claims:** It often happens that two competing research groups make the same measurement and find results that differ by a marginally significant amount. The differences can often be due to specific statistical choices that were made in the analysis. In such disputes, having reproducible research means that you can invite the competing group to inspect your analysis in detail (and hopefully be proven right!).

pyMooney: Generating a Database of Two-Tone, Mooney Images

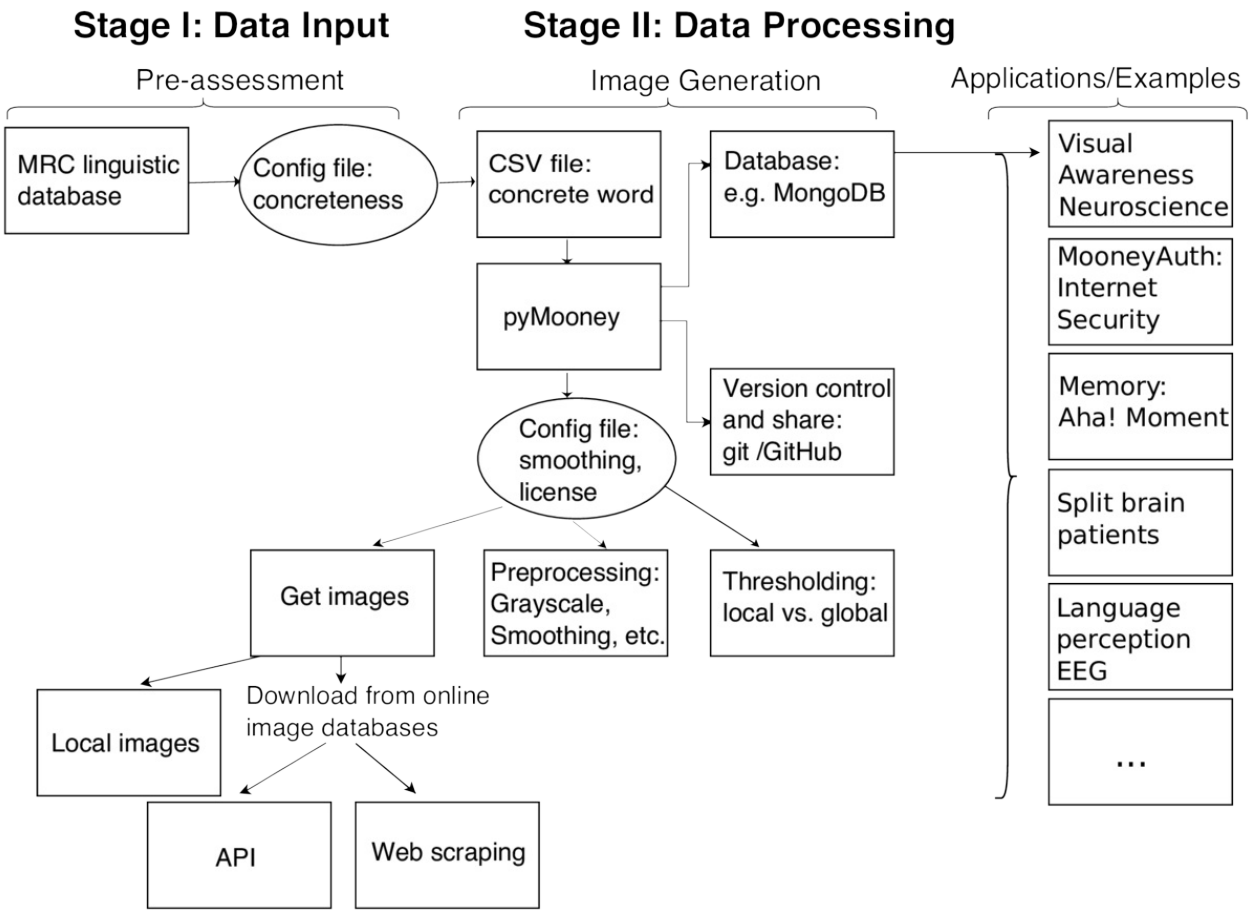
Fatma Deniz

My name is Fatma Deniz, I am a postdoctoral fellow at the International Computer Science Institute, Helen Wills Neuroscience Institute and a Data Science fellow at the Berkeley Institute for Data Science.

I work on computational neuroscience. The current case study will only describe one part of my usual research pipeline, i.e., stimulus generation. Every successful neuroscience experiment needs to run several pilot experiments to create the best stimulus that is suitable for one or several questions that the researcher is asking for. Hence, this step is usually tedious, where different parameters are tested until a final set of stimulus is used in the experiment. In order to build upon the work a researcher has already conducted or to create new experiments (with new hypothesis) using the same stimulus set, it is very important to have access to the stimulus that has been used, or to create new stimulus with the same parameters or algorithmic procedure.

In this case study I will focus on an image database that was created for a specific cognitive neuroscience question. Since the experiment was published the database has already been used in several other experiments. The images that I created for the study were two-tone, Mooney images. These images are binary, black and white images, where a single hidden object is only recognizable when the original image has been shown previously to the observer, the hidden object was outlined, or after a certain time when the observer finds hints in the image that follows with recognition. These different steps of recognition makes such stimulus creation very difficult. Using these images in the original experiment (Imamoglu, Kahnt, Koch, & Haynes, 2013) I asked whether functional connectivity in the human brain is altered when the hidden object is recognized vs. when it is not recognized.

Workflow



Generating two-tone, Mooney images starts with a selection of concrete words. There is a database called MRC Psycholinguistic Database where each word has been labeled as concrete or abstract, how frequently it is used, etc. From this database I selected 967 concrete words. The necessary parameters are saved in a config file (e.g. concreteness rating and imageability rating between 550 and 700). These concrete words are saved in a CSV file. Based on these words I downloaded real-world images tagged with these concrete words from an online image database (Flickr) in an automated fashion using custom written **pyMooney** python package. This package is based on a python API (Flickr API) and the scikit-image library. Each project needs to have a config file that specifies whether the Mooney images are created based on images that are stored locally or whether the images should first be downloaded from the image database. If images are downloaded from an online resource licensing information needs to be set in the same config file. Using a smoothing and thresholding process (Otsu, 1979) and prescreening of images, I created a database of 330 two-tone images for the pilot image selection experiment. These images can be stored in a document oriented database (e.g. MongoDB). A database has the advantage that information such as what preprocessing steps have been applied, what license information an images has, what is the average reaction time of the images in specific experiments etc. can be stored among the images. In addition images can be searched and selected according to this information. In this pilot experiment human subjects were presented with the two-tone, Mooney image and were asked to indicate the time when they recognized the hidden object in the image with a key press. They were further asked to

label the name of the object that they think they recognized.

In our functional magnetic resonance imaging (fMRI) experiment we were interested in the question how brain activity changes when subject's recognize the hidden object versus when they not. The two-tone images do not change over the course of the presentation but a subject's perception change over time, and this moment in recognition is associated with a change in brain activity, which we wanted to capture. FMRI image acquisition is relatively slow (every 2 seconds). In addition, as fMRI scans are costly, we are limited by time. Hence, for this fMRI experiment I selected 120 Mooney images (resized to have a 400 x 400 pixel size) that were recognized within 4-10 seconds in the pilot experiment.

The code (written in Python) to create new two-tone, Mooney images are available on GitHub.

Pain points

When images are downloaded from an online resource copyright issues can emerge. This can be avoided by downloading images that are licenced as Creative Common. This change is reflected on the latest version controlled pymooney code and new images can be created with such criteria.

Key benefits

The main benefit of this part of a larger experiment is that these images are now available for further research. The images are currently used in 30 different experiments ranging from clinical set-ups, human memory experiments, vision research, and latest internet security applications.

Key tools

Image processing libraries are important building blocks of this particular case study. In this case the open source python library scikit-image was used. In addition online image database APIs such as FlickrAPI are essential.

Questions

What does "reproducibility" mean to you?

In the context of this study reproducibility means that given the code and a database of images, a new researcher can have access to the images and use the code that was used to create the same images or new images with the same parameters. These images can

then be used to (i) replicate the current findings, (ii) create new questions potentially based on the current findings.

Why do you think that reproducibility in your domain is important?

Not only in my domain but in general I think without reproducible research we waste a lot of professinal time and research money. In my own domain, and in direct connection to this case study I can outline an example. When I came upon the two-tone images in the literature, I contacted several groups with a request to use their images (of course I mentioned that I will properly cite their work), who had used similar images for other behavioral or neuroscience experiments. Unfortunately, the images that these groups used were very limited in number but nevertheless, I never got a response from these groups. Hence, in order to start with my experiment I had to spend almost a year to create the new stimuli.

How or where did you learn about reproducibility?

This was a self taught practice. Hence, the stage it is described here still has some space for improvement, which I will elaborate further below.

What do you see as the major challenges to doing reproducible research in your domain, and do you have any suggestions?

In general, in neuroscience the main pitfalls are sharing of human subject data and the resistance in the field to share code (or images in this case). The majority of the community does not have the mind set to that is similar to open source. Researchers are afraid that someone else can conduct an experiment before they publish their results. Even if they published some results they are sometimes not willing to share the data as they think they can continue to ask new questions.

What do you view as the major incentives for doing reproducible research?

I think reproducible research allows progress not only in the own domain but also makes interdisciplinary projects possible. To see that your own research is further used not only in your own domain but also across domains is very rewarding.

References

Imamoglu, F., Kahnt, T., Koch, C., & Haynes, J.-D. (2013). Changes in functional connectivity support conscious object recognition. *Neuroimage*, *63*, 1909–1917.

Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE Trans. Sys., Man., Cyber.*, *9*(1), 62–66.

Problem-Specific Analysis of Molecular Dynamics Trajectories for Biomolecules

Konrad Hinsen

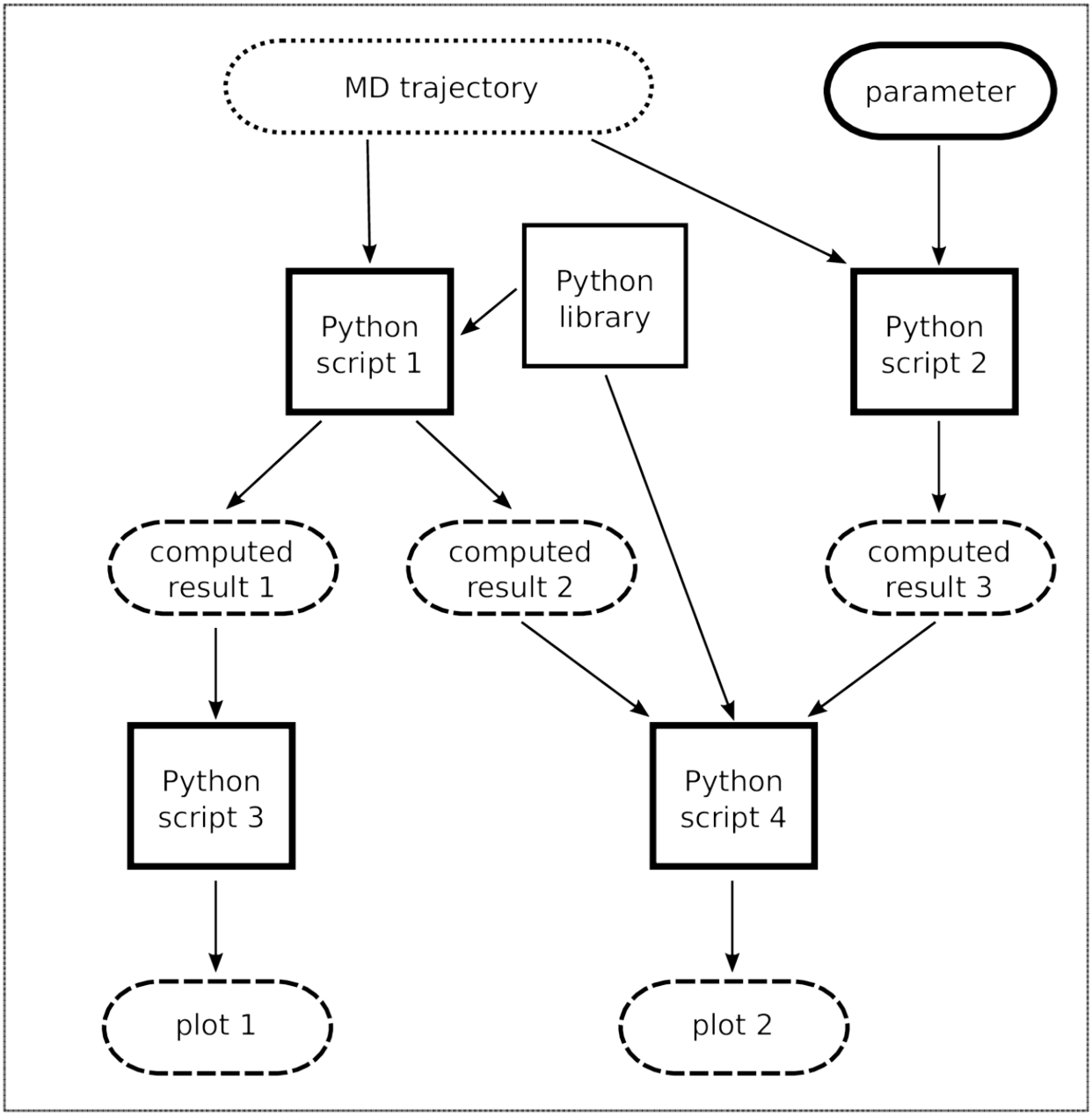
My name is Konrad Hinsen, and I am a researcher at the Centre de Biophysique Moléculaire in Orléans, France. My field of research is molecular biophysics, and in particular the study of the flexibility and dynamics of proteins. All of my work is based on computational approaches, of which the most important ones are elastic network models and Molecular Dynamics (MD) simulations. Moreover, most of my work concerns the development of computational methods rather than the application of already established methods.

This case study is about the extraction of information from MD simulation trajectories, a very common type of work in my field. MD simulations themselves are relatively standard procedures, performed using one of a handful of well-known software packages. They take a few days to a few weeks on a small parallel computer with a few tens of processors, and produce a so-called trajectory file that is one to ten GB in size. Analyzing these trajectories in order to actually learn something about the system that was simulated is a separate step that is much less standardized, meaning that there is a lot of problem-specific code involved. This code is as much a result of the workflow as the plots of the computed quantities.

For reproducible and publishable workflows, there are three specific challenges in this situation:

1. The size of the trajectory files, which are difficult to publish in a citeable way, often being larger than the current upper limits of Zenodo or figshare.
2. There are CPU-intensive tasks that are typically run on a parallel computing cluster in batch mode, and explorative tasks that are done interactively or near-interactively (running short scripts that take about a second) on a desktop machine. Dependency tracking across machines is not supported by most workflow management systems. It doesn't help that computing clusters often have limited network connectivity.
3. The distinction between "software packages" and "workflows" is not useful when most of the code being executed is problem-specific. A more appropriate code structure is "well-established techniques implemented in libraries", "problem-specific scripts" and at the top level "coordination of a small number of scripts". It's the last two levels that must be captured for reproducibility.

Workflow



Stage III: Data Analysis

A published example of the workflow described below can be consulted in the form of two code/data packages (package 1, package 2) and the article describing the study.

The workflow diagram is actually a dataflow graph with attached workflow information. Compared to most approaches to workflow, which place the tools (workflow manager, software packages, Web services, ...) in the center of attention, the approach I describe here focuses on the data and on the way scientists interact with the data. The workflow below is not about "getting a job done" but about "developping and fine-tuning a scientific model".

The dataflow graph shows code in rectangles, and "passive" data in rounded boxes. Code consists of a small number of Python scripts, of which four are shown in the diagram. Data flows from top to bottom, as shown by the arrows, starting with the MD trajectory that is the overall input, and ending in plots showing computed quantities. The three rounded boxes labelled "computed results" are intermediate results, computed by Python scripts 1 and 2 and consumed by Python scripts 3 and 4.

From the point of view of workflow management and reproducibility, the most important distinction among the data items is "human input" (solid outline) vs. "computed data" (dotted outline). It's the human input that represents the scientific model, and thus the main output of this workflow. It consists of code (Python scripts 1 to 4) and numerical parameters (a single one in the diagram), though that distinction is rather arbitrary: every parameter could be turned into a line in a script. Computed data includes the plots that go into the journal article, but also intermediate results. In a fully reproducible workflow, the computed data need not be stored, because it can be recomputed at any time. Nevertheless, it is often preferable to store it explicitly, in particular if recomputation takes a long time. Stored computed data is also more readily available for exploration by scientists who want to gain a better understanding of the method.

The workflow consists of the iterative refinement of the models and methods. The two key tools in processing the workflow are:

- a version control system such as git for keeping track of the changes
- the ActivePapers dependency manager for coordinating the computations

Correspondingly, the state of the project consists of

- a repository under version control, which tracks the changes to the "human input" items as the project advances
- an ActivePaper file, which stores the current state of all data items and the dependency graph

There are two variants of a refinement step: adding a new script or parameter, and modifying existing scripts and parameters. The first kind, which extends the data flow graph, consists of the following user actions:

1. Write the new script.
2. Commit it to version control.
3. Check in the script to the ActivePaper.
4. Run the script via the ActivePapers dependency manager.

The second kind, which preserves the data flow graph, differs only slightly:

1. Edit scripts and parameters.
2. Commit the changes to version control.
3. Check in the modified versions to the ActivePaper.
4. Update the ActivePaper.

The fourth step recomputes all data that is affected by the changes made in step 1. The recomputation is steered by the *dependency graph*, which is obtained from the data flow graph by redirecting arrows that point into a script to point instead to the outputs of the script. The ActivePapers dependency manager computes the dependency graph automatically during the execution of the scripts. Users do not have to deal with (or even know about) either graph explicitly. They write and run scripts as they did before reproducibility became an issue. Similar approaches are used in Sumatra and noWorkflow, but most workflow managers adopt the opposite strategy of letting the user construct a workflow explicitly and then execute it.

A project can be transferred from one computer to another by copying the ActivePaper file and the version control repository. For the common situation in molecular simulations that lengthy computations are off-loaded to a cluster, step 4 in the above procedure is slightly modified: The ActivePaper is sent to the cluster, the "run new script" or "update" operation is performed on the cluster, and the modified ActivePaper file is transferred back to the user's desktop machine. All the tools have a command-line interface, making it easy to use them over an ssh connection.

Method-development projects tend to be small, involving a handful of people. The pitfalls of coordinating modifications to files can easily be avoided by having a single person perform each refinement step, or even all of the refinement steps. Other participants can of course contribute ideas, and inspect the current state of the project for analysis.

At the end of the project, the ActivePaper file(s) can be published, making all of the code and data available to other researchers. The ActivePaper file contains the complete final state of the project (though not its history), meaning that anyone can continue from that state. An ActivePaper file for a new project can re-use items from already published ActivePaper files through a DOI (Digital Object Identifier), allowing other researchers to build on published computational work. The DOI can also be used for citations in journal articles.

Pain points

The main practical difficulty is that most of today's computational scientists grew up with tools and practices that are not compatible with reproducibility. This is particularly true for the field of molecular simulations, where reproducibly published studies are still rare. Working reproducibly requires adopting new tools and habits, and modifying existing software for integration with reprooducble workflows. There is a permanent temptation to give up reproducibility for faster scientific progress.

The immaturity of current workflow tools for reproducible research adds another layer of cognitive overhead. In the workflow described above, this is mainly the use of separate tools for tracking history and dependencies. Today's version control systems, designed for software development rather than computational science, cannot easily be extended by the kind of dependency management required for research. On the other hand, writing new version control software integrated with depenency management represents an effort that is hard to justify at this time.

A major constraint imposed by the ActivePapers system is that all code must be written in Python and all data must be stored in HDF5 datasets. While Python is popular enough for molecular simulation to make the first constraint very acceptable, HDF5 is still a rare choice for data storage, although this is changing thanks to initiatives such as H5MD.

The use of specific tools is rarely sufficient to ensure reproducibility. Tools can only take care of *replicability*, i.e. the technical aspect of tracking all computational dependencies such that a computation can be re-run identically. Reproducibility at the scientific level requires that all steps can easily be understood and verified by fellow scientists. Best practices for reaching this goal remain to be developed. One observation from the applications of the above workflow is the importance of access to intermediate results for human inspection. This suggests an overall structure of many small scripts that each do a well-defined job and communicate via explicitly stored datasets.

Key benefits

The traditional workflow of changing scripts and running the interactively in a shell is extremely prone to mistakes. The most frequent one is forgetting to re-run a script after its input data has changed because of an update to another script. Before I adopted reproducibility support tools, I regularly found myself looking at a data file and wondering which exact sequence of script executions had produced it. The question typically comes up when writing a paper. Even for today's minimal "materials & methods" sections, it is typically necessary to look up parameters and other choices in the scripts when writing the documentation, and that's often the moment when one discovers what a mess they are. This is no longer an issue when the complete final project state is available for inspection, and guaranteed to be complete and coherent by software tools.

Key tools

The key tool in my workflow is the ActivePapers toolset, which was in fact designed specifically for supporting reproducibility in the context of atomistic and molecular simulations. It supports in particular

- computations on large datasets by storing them efficiently in HDF5 files with the dependency information stored as HDF5 metadata
- dependency tracking across machines by storing all datasets and their dependency graph in a single HDF5 file that can be copied easily from one machine to another

The only other reproducibility-enabling tool in the workflow is a version control system.

Questions

What does "reproducibility" mean to you?

Given that my work is 100% computational, my long-term goal is full reproducibility, starting from a specification of the simulation and ending with the plots that go into a journal article. This goal is unrealistic at the moment because the simulation software packages do not support reproducibility. One problem is the accumulation of numerical roundoff errors, which are insufficiently standardized across processors and compilers to be reproducible. Another problem is the widespread use of random number generators without user control over the random seed.

For this reason, I have been setting myself a more modest goal for this case study: reproducibility of the trajectory analysis step, using the MD simulation trajectories as input as if they were experimental data outside of my control. This is a useful compromise because the development of trajectory analysis techniques is the central scientific topic of this work.

Why do you think that reproducibility in your domain is important?

Most MD simulation studies are so complex that in the absence of reproducibility, it is impossible to be sure what was really computed. Most mistakes do not lead to a recognizably wrong result, but to a somewhat different one that could well be correct.

How or where did you learn about reproducibility?

I developed them myself, having found nothing suitable for the specific needs of molecular simulations after a careful survey of existing technology and practices.

What do you see as the major challenges to doing reproducible research in your domain, and do you have any suggestions?

The main challenges are human and social. Most of my colleagues have experienced the problems that non-reproducibility creates, but few are willing to invest the extra effort to do a better job, and many remain unaware of the tools and practices for reproducibility that already exist. Journals in my field generally don't require the publication of software or data, and do not in any way encourage reproducibility. Technical challenges exist in that the most popular software packages do not support reproducibility, but the technical challenges could be met with little effort if there were sufficient motivation in the community.

What do you view as the major incentives for doing reproducible research?

- Feeling more confident about the correctness of my results.
- Being able to build safely on earlier work (by myself or others)

Are there any best practices that you'd recommend for researchers in your field?

I'd already be happy if publishing software and data became the norm in my field. It's hard to recommend any more elaborate practices before the basics become standard.

Developing an Open, Modular Simulation Framework for Nuclear Fuel Cycle Analysis

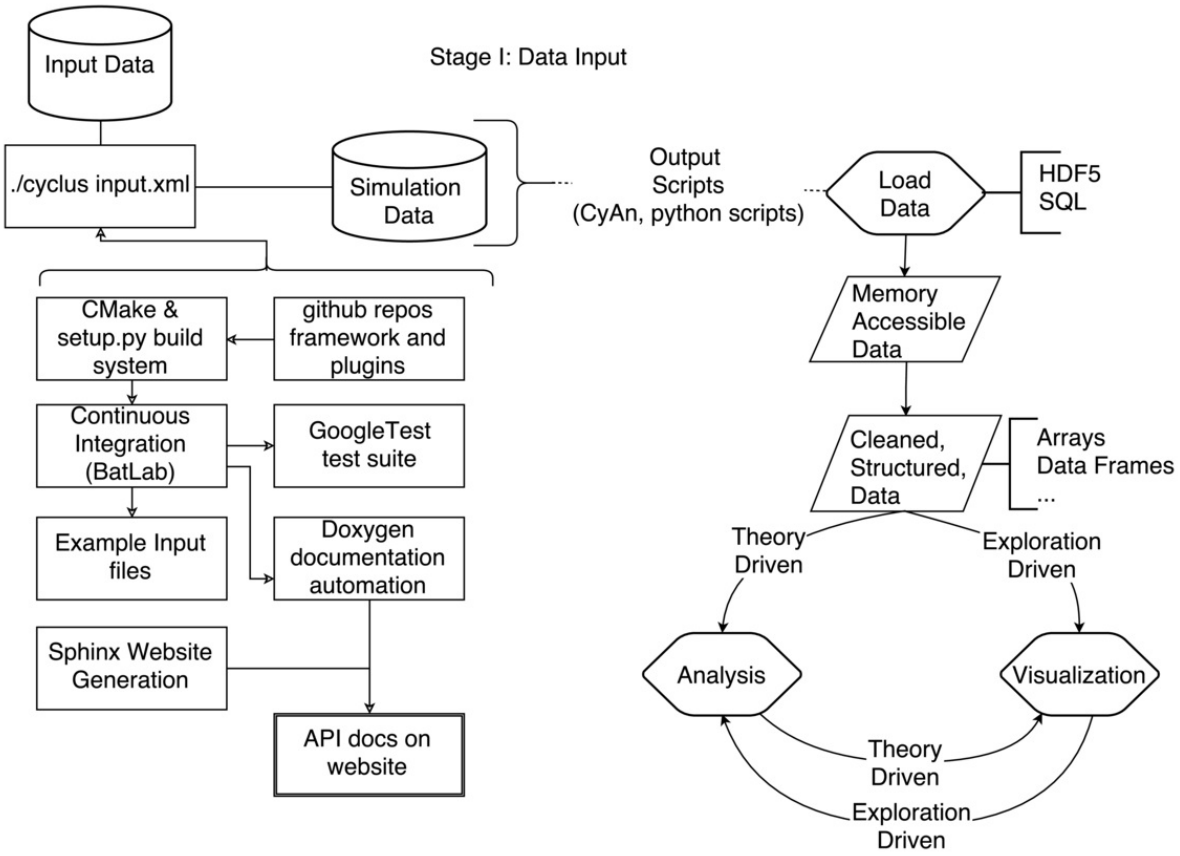
Kathryn Huff

My name is Kathryn (Katy) Huff, and I am a Nuclear Science and Security Consortium postdoctoral scholar in the Nuclear Engineering Department and a Data Science Fellow with the Berkeley Institute for Data Science. My research includes computational nuclear fuel cycle analysis and computational simulation of coupled, transient, nuclear reactor physics.

Improving the safety and sustainability of nuclear power requires improved nuclear reactor designs, fuel cycle strategies, and waste-disposal concepts. The systems are sufficiently complex that breakthrough advancements may emerge when modern data methodologies are applied to their simulation. In particular, faithful assessments of potential nuclear fuel cycles require dynamic, discrete facility, discrete-material simulations of the mining, milling, transmutation, reprocessing, and disposal of nuclear materials as well as the production of energy and movement of capital.

This case study is an overview of the workflow behind the Cyclus nuclear fuel cycle simulation framework -- a tool for exactly that kind of modeling, simulation, and analysis. The workflow described used to create a software tool that other nuclear engineers can use easily, modify quickly, and contribute to when they need to customize behavior or model a different technology.

Workflow



I and a group of geographically-dispersed researchers (graduate students and professors) collectively develop and maintain an agent-based simulation framework called Cyclus. We also develop and maintain plug-in models representing the agents in the simulation. These agents model the mining, milling, fabrication, transmutation, and disposal of nuclear material in the *nuclear fuel cycle*.

Cyclus is a C++ code base. The configuration and build system is created from a combination of Python and CMake (a crossplatform automatic makefile configuration system) and supports both Linux and MacOS operating systems. Our input validation library accepts either xml or json input files. The simulator accordingly conducts a simulation which generates an output database in either SQL or HDF5 format which can be traversed by a separately developed graphical user interface.

As we develop this software, we rely on a number of best practices to ensure reproducibility.

When a large-scale enhancement is needed, a Cyclus Enhancement Proposal (CEP) is proposed and discussed among the development team. Smaller enhancements are discussed as issues in GitHub. Once approved, the enhancement is implemented and a pull request is made in GitHub. Our automated continuous integration server (BatLab) runs the full suite of unit, integration, and regression tests. Before a proposed change is allowed into Cyclus, it must be covered by a test and all tests must pass.

Unit tests cover code units like functions and are implemented using the GoogleTest framework. Integration and regression tests are performed by running sample simulations and verifying that results match predictions or previous results. A set of standard input files are run, then the output is inspected and compared via Nose, a unit testing framework in Python.

Similarly, API changes must be documented as required by the Cyclus documentation CEP. The documentation for the current stable branch and the development branch are both provided on the Cyclus website using Doxygen and Sphinx, which are both automatic documentation systems that rely on the code comments in the C++ and Python code, respectively.

Finally, we use the Google C++ style guide to make our code as consistently formatted as possible.

When the change is made, a developer begins to conduct a particular analysis by creating an input file. That input file is provided to the Cyclus framework and validated by its input validation framework. According to the input file, a simulation is run. The ouput database that is produced contains important metadata about the simulation. It contains:

- a complete copy of the input file
- the commit hash of the current version of the Cyclus code
- commit hashes for all necessary plugins retrieved from the Cyclus ecosystem

That database, containing both data and metadata, can then be analyzed by the user. When analyzing the database, a choice is made by the user about how to interact with the data. The Cyclus development team has provided a GUI and a Go library (called CyAn) with which the database (in either SQL or HDF5 format) can be accessed and brought into memory for vizualization and analysis. Additionally, many user-developers have their own set of Python scripts that can do this stage of tasks. Given the universal nature of these database formats, most common scripting languages can be used to extract the data and metadata efficiently, so many options exist.

In summary, the research workflow in this framework has the following steps :

- If necessary, a developer proposes a change to support their analysis
- The change is made including passing tests and satisfactory documentation
- It is reviewed and pulled into the master branch
- The software is rebuilt and installed using our build system
- A simulation is defined in json or xml

- The input file is run and an HDF5 or SQL database results
- The database is analyzed with a separate GUI, python scripts, or a Go library
- A collaborative paper is created in LaTeX on GitHub
- All input files contributing to the analysis are contained in the repository holding the document

All of these steps are conducted in the context of git and GitHub.

Pain points

Build systems are painful. In particular, cross platform configuration and builds are an enormous time-sink for our research group. There are a number of reasons for this.

First, supporting C++ builds on Windows is sufficiently difficult that we abandoned supporting that platform.

Also, due to the physics-based solvers and optimization calculations in our simulations, external library dependencies are essential to Cylcus. We rely on libraries like Boost and LibXML2 to facilitate development, and we rely on libraries like Blas, Lapack, and COIN for mathematical solvers. For this reason, new developers spend a non-trivial bulk of their spin-up time building and installing the dependencies necessary to install Cyclus on their particular platforms.

Finally, our continuous integration system relies on our ability to create scripts that build, install, and test Cyclus. For this, we use a set of servers at the University of Wisconsin called the BatLab. Unfortunately, BatLab has a few problems. Because of the proprietary nature of MacOSX, it cannot run truly MacOSX instances. It runs, instead, Darwin servers that mimic the behavior of MacOSX. For this reason, idiosyncratic failures apparent in Mavericks and Yosemite but not Darwin cannot be caught before entering the code-base. Additionally, BatLab is somewhat unpredictable and inflexible. Since the behavior of BatLab undergoes a lot of churn, our continuous integration suite is sometimes rendered completely useless.

Key benefits

The Cyclus Enhancement Proposal (CEP) strategy was a bright workflow choice that was inspired by the analogous strategy in the Python community (PEPs). I recommend this to any research group that values strategic planning, consensus, and thoughtful development. A discussion of our workflow around these proposals can be found here.

Fundamentally, a CEP is :

> a design document providing information to the Cyclus community, or describing a new feature or process for Cyclus and related projects in its ecosystem. The CEP should provide a concise technical specification of the feature and a rationale for the feature.

CEPs document major new features, community discussions, and documentation of theory or design not captured by the in-code documentation. Because they are maintained alongside the website source code in a version controlled repository, provenance of the discussion surrounding their acceptance is maintained.

Key tools

We use CMake to configure and build our software. Much more human readable than the configuration files within the GNU autotools suite, CMake makes our lives easier.

The continuous integration system, though difficult to implement due to build issues, has decreased development time. It would not be possible without CMake, GoogleTest, and Nose.

Questions

What does "reproducibility" mean to you?

A reproducible research product is one that has been sufficiently documented, well-constructed, and preserved for its results to be recreated by some external researcher or group.

Why do you think that reproducibility in your domain is important?

My domain, nuclear engineering, is one where precision and accuracy are both of utmost importance to both human and environmental outcomes. Any conclusions drawn by science can only make an impact in the real world if they can meet the standards set out by the Nuclear Regulatory Commission. For this, reproducibility is paramount.

How or where did you learn about reproducibility?

I learned these practices primarily from my advisor, Paul P.H. Wilson at the University of Wisconsin, Madison. I also learned from colleagues in The Hacker Within, the Scientific Python community, and Software Carpentry.

What do you see as the major challenges to doing reproducible research in your domain, and do you have any suggestions?

One major problem is export control. Making software and data open source is restricted by the US Department of Energy, in some cases.

What do you view as the major incentives for doing reproducible research?

- **Fear.** The fear of retractions due to faulty software or data can be reduced by enforcing transparent reproducible practices, which tend to reduce the likelihood of being accused of scientific fraud.
- **Surprise.** Six months after a paper is submitted, the surprise of no longer recalling your own thought process is unpleasant. To avoid it, reproducible practices can help you reproduce your present work in the future.
- **Ruthless Efficiency.** The automation inherent in reproducible workflows, makes tweaking and re-running of simulations and analysis very efficient.

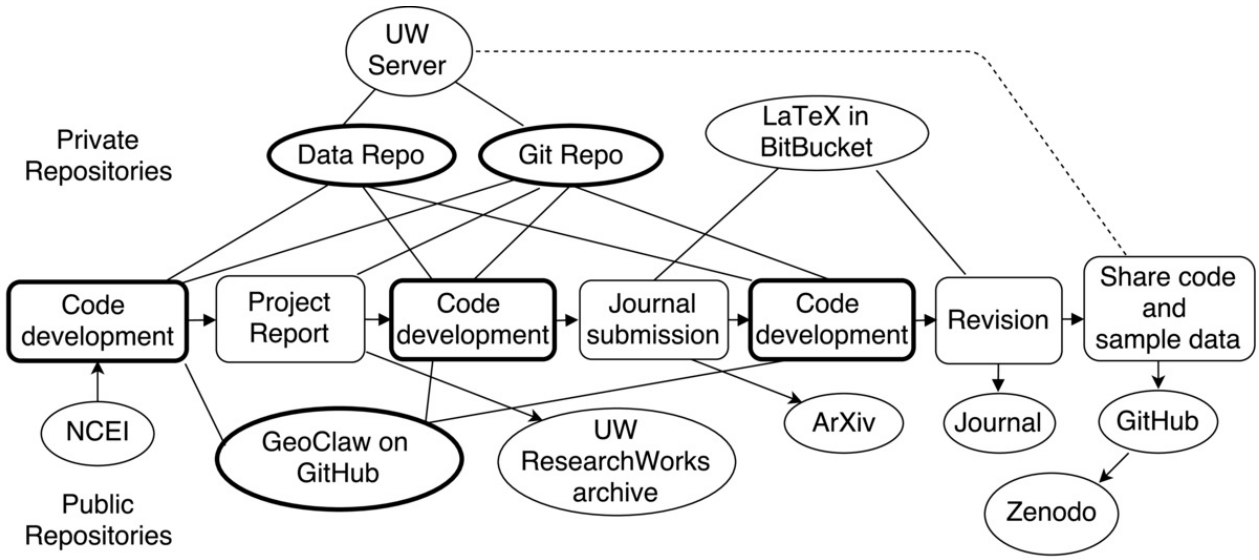
Producing a Journal Article on Probabilistic Tsunami Hazard Assessment

Randall J. LeVeque

My name is Randy LeVeque and I am a Professor of Applied Mathematics and one of the core developers of the open source GeoClaw software package for modeling tsunamis and other geophysical flows. Recently we have been using this software to study new approaches to probabilistic tsunami hazard assessment (PTHA), in which the goal is to take some probability distribution of possible future earthquakes that might cause large tsunamis and produce a probabilistic hazard map for a particular community, indicating which regions are most at risk and estimating the annual probability of flooding to a given depth at each point. This is complicated by the fact that the depth of flooding by a particular hypothetical tsunami depends on whether it arrives at low tide or high tide, and we have developed ways to incorporate this uncertainty.

The workflow I will describe relates to a journal publication on this topic. Much of the research was originally performed as part of a consulting contract funded by a private firm, as part of a broader pilot study funded by the Federal Emergency Management Administration (FEMA). We used version control for the code developed as part of this project that was initially in a private repository, along with the results of many tsunami simulations. A final project report based on this work was made available in our institutional repository, but was not published in a journal. We later improved the description of the methodology and performed additional computational experiments in the process of writing up a portion as a journal article. A variety of different private and public repositories were used in the course of this work, along with several platforms for sharing code, data, and the report and journal article.

Workflow



We first created a new account on the University of Washington (UW) campus computing system dedicated to this project that could be shared by the three collaborators, with sufficient storage for accumulating simulation results (and securely backed up by campus services). On this account we created a git repository that we could all access via ssh to use as our master repository for developing code, and eventually for writing the project reports. We did not use GitHub since we wanted a private repository and did not need the web features of GitHub (or Bitbucket) for this phase of the project.

This project required using some large datasets that are openly available from the National Centers for Environmental Information (NCEI), in particular topography and bathymetry data for running the tsunami model and tide gauge data. We downloaded and archived some of this data on the UW account, but did not put it in the git repository since these did not need to be under version control. Instead we wrote shell scripts to rsync this data to each collaborator's laptop or other computers as needed. (rsync is a utility on unix-like systems to transfer and synchronize files). These scripts were kept in the git repository. Similarly we wrote scripts to rsync simulation results from the computer where the simulation was performed back to this account, along with some metadata. The new methods being developed for tidal uncertainty were implemented in Python code used for postprocessing the simulation results. One collaborator was doing most of the simulation runs, on several different computers, while another was developing and testing the postprocessing code, so rsync'ing the necessary data between laptops via the campus account was convenient and insured that it results were archived as we went along.

The shared campus account was also used to host webpages so that the visualizations produced from each simulation could be viewed by all collaborators. These webpages were also eventually used to share results with the project sponsors and reviewers of our preliminary report.

The private git repository was also used for writing the final report in LaTeX and collecting the final figures to go into the report. The third collaborator, who was less involved in the coding, was not completely comfortable with git and so we also used Dropbox for sharing and commenting on drafts of the report, but all changes to the LaTeX were made in the git version.

The final report was made available to the public by depositing it in the UW ResearchWorks Archive.

When we started working on the journal paper, we created a private Bitbucket git repository for collecting the code specific to the paper and for the LaTeX file. Bitbucket is similar to GitHub but offers free private repositories, requiring a paid account only for public repositories. By contrast, GitHub offers free public repositores, and charges for private repositories. The interfaces are similar and it is easy to transfer a git repository between them, or maintain copies on both services, so it is often convenient to use both for different purposes. The submitted preprint was also posted on the arXiv, a widely-used preprint server from which it was available with open access.

The referees requested changes to the paper and some figures needed to be redone, which was easy to do since we had produced all figures with scripts in the git repository. The revised paper was developed in this same repository, along with the authors' responses to the referees.

After the revised paper was accepted by the journal, we created a new public GitHub repository for the code and small datasets needed to reproduce some of the figures in the paper that illustrated the basic methodology. This repository was also linked to Zenodo, and a GitHub release was performed that triggered automatic archiving of a zip file of all the code at that stage, and assignment of a DOI.

In addition to the test problem for which we shared code, the final paper also contained some figures with results from the overall project, the probabilistic maps produced for Crescent City, CA using this methodology. Reproducing these results would require running roughly 100 tsunami simulations. We are fairly confident that we have all the code and data to reproduce these results if required, but we have not made this publicly available.

Pain points

Using rsync for large datasets worked fine once we figured out a good workflow and scripts, but is not ideal. A version control system like git that works well for large quantities of data would have been very useful.

Some data could not be shared and we also had to be careful about sharing preliminary results since emergency managers and the agencies involved are very sensitive about publicizing risk maps for specific communities before they have been properly vetted and agreed on.

Key benefits

This workflow proved to be very valuable for this long-term project in which many parts of the code and methodology were evolving. The initial project was followed by additional funding for a Phase II, in which the focus was on studying current velocities rather than only flow depth. This required re-running all the tsunami simulations with a modified version of GeoClaw. Having done all the initial work via scripts archived under git, it was relatively painless to redo these runs. In the meantime other changes had also been made to the GeoClaw code, and having both our code and GeoClaw under version control was very useful when comparing results.

Questions

What does "reproducibility" mean to you?

There were two distinct aspects of reproducibility important in this work. The original development of new techniques was performed in the context of a project that went on for several years and required running many tsunami simulations with the GeoClaw code for the probabilistic studies, each of which took several hours of computing time and produced large quantities of output data. During this time frame the GeoClaw software was evolving, along with our methodologies. GeoClaw is openly developed on the GitHub site. We needed to be able to compare new results with those computed previously, and be able to identify what changed in the software or our code in between, if necessary. For this aspect the goal was not to openly share all of our work or the results (nor were we allowed to, due to the nature of the project), but we needed to be able to reproduce results ourselves if necessary and keep the code under version control to identify changes.

The other aspect is that we wanted the particular new method written up in the journal paper to be accompanied by the Python code that implements the method and a sample set of data that was used to produce some of the figures in the paper. In this context we wanted the figures to be reproducible by a reader using this code, in hopes that this would aid others in understanding the methodology and adapting the code to solve their own problems.

Why do you think that reproducibility in your domain is important?

For researchers who develop new methods and algorithms, it is often important to be able to see the details that are in the code but don't make it into the paper, both to better understand the work and to find potential errors. It also facilitates comparing different methods for the same problem.

In natural hazards modeling, the simulation results may be used by engineers or policy makers to make decisions with public safety implications. Transparency and reproducibility are important aspects of accountability.

How or where did you learn about reproducibility?

My interest in the topic came out of frustration with the publications in numerical analysis (including my own) where it was impossible to reproduce published results or fully understand the implementation of new algorithms they describe. I became proficient with git initially through involvement in open source software projects.

What do you see as the major challenges to doing reproducible research in your domain, and do you have any suggestions?

- Convincing collaborators to learn and use a common set of tools is sometimes a challenge, and some researchers are more willing to share code and data than others.
- Some input data and/or results can not be shared publicly, so it may be necessary to selectively share data and perhaps have both private and public repositories.

What do you view as the major incentives for doing reproducible research?

- Ability to easily modify and build on past work.
- Ability to compare new approaches or software with past versions and determine what changes made a difference in results.
- Facilitates collaboration.

Are there any best practices that you'd recommend for researchers in your field?

- Using version control of some sort is the single most important first step.

- Make a habit of cleaning up code used to produce final results so that it's well documented and all the necessary steps are clearly laid out. Then run through them from scratch if possible to insure that it works. Even if you don't plan to share it with others, your future self will thank you.
- If you do share code and/or data, do so in an archival repository that issues a DOI, and attach a license.

Would you recommend any specific resources for learning more about reproducibility?

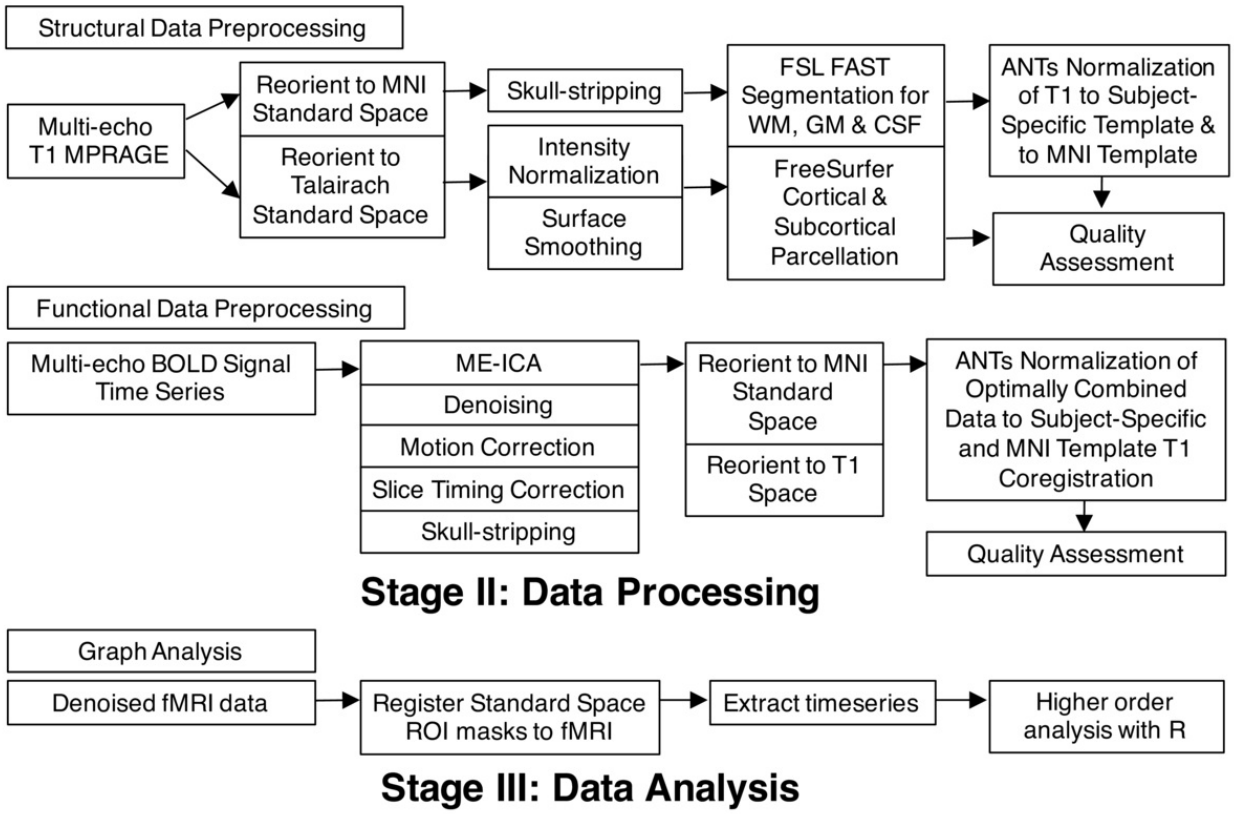
- The UW eScience Institute Reproducibility and Open Science Working Group has developed some guidelines.
- The 2012 ICERM Workshop on Reproducibility in Computational and Experimental Mathematics resulted in a final report with recommendations and additional links.

A Reproducible Neuroimaging Workflow using the Automated Build Tool make

Tara Madhyastha, Natalie Koh and Mary K. Askren

We are Tara Madhyastha, Natalie Koh and Mary K. Askren, affiliated with the Integrated Brain Imaging Center (Radiology). In this project, we use functional magnetic resonance imaging (fMRI) to interrogate the function of the brain in elderly individuals to understand how physiological changes relate to mild cognitive impairment and might be predictive of dementia. Specific challenges to processing fMRI data are that the number of steps involved are complicated and can have significant impact on the results, and to achieve the best results, components from many different software packages must be combined. To do this, we need to structure our own pipelines and quality assurance. Data produced using this workflow are visualized in Bach et al. (2016).

Workflow



One of the major advances of systems neuroscience in the last two decades is the discovery that, at rest, the correlations of the blood oxygen level-dependent (BOLD) signal in cortical regions of the brain (measured non-invasively using fMRI) describe large-scale networks. These networks are altered in a variety of psychiatric and neurodegenerative disease; however, accurate measurement of networks is difficult in part because of a variety of artifacts, including subject motion.

The purpose of this workflow is to preprocess fMRI data for 54 subjects, examining the quality of the data, and generating time series of specific regions of interest. An earlier version of this pipeline (with less exhaustive quality assurance and with a different algorithm for aligning subjects to a standard template) was used to compare a traditional and an improved method for noise removal, generating time-varying correlation matrices for input to an exploratory visualization paradigm. The current preprocessing pipeline supports many primary analyses that are in progress.

Preparation of data for functional connectivity analyses involved the preprocessing of both structural and functional data using a combination of open source neuroimaging toolkits that run in the UNIX environment (Debian Wheezy). These programs include FSL , FreeSurfer, Advanced Normalization Tools (ANTs) and Analysis of Functional Neuroimages (AFNI).

While the details of this processing, described below, are not necessarily important to understanding this case study, the important point is that best practice processing of neuroimaging data requires multiple steps to be performed using different software packages, and a complete reporting of the details involved in these steps is crucial for reproducibility.

Structural data was processed in two ways. First, we used the high resolution structural image to align the lower-resolution functional image (anatomical-functional co-registration). We also used ANTs nonlinear registration to create a study-specific template for all structural images. Second, we used FreeSurfer to create a cortical/sub-cortical parcellation of the brain. Functional data was preprocessed using multi-echo independent components analysis, or ME-ICA (using AFNI's meica.py script; see Kundu, Inati, Evans, Luh, & Bandettini (2012)). Removal of sources of noise in fMRI data is a huge issue, and the ME-ICA uses fMRI images acquired with different parameters (echo times) to automatically classify sources of variation in the fMRI data as BOLD-related or noise, producing a denoised dataset. It also produces an "optimally combined" image that can be denoised using more traditional techniques. ME-ICA also performs the standard steps of skull-stripping (removal of non-brain tissue from the image) and correction for motion and timing of the acquisition of different slices. The optimally combined output from ME-ICA was aligned to Montreal Neurological Institute (MNI) standard space via the structural registrations and study-specific template defined earlier. Preprocessing steps are fully automated using Make (Askren et al., 2016). Make is a UNIX utility which takes an expression of workflow in the form of target files and their dependencies (a "Makefile") and creates a graph describing what work needs to be done to "make" a target file.

Quality assessment (QA) of preprocessed data involved the manual checking of reports for each study subject comprising images, animated GIFs (data concatenated temporally) and statistics that were generated at both the intermediary and final steps of the workflow. This QA was performed by Natalie Koh, and took approximately 20 minutes per subject. Aberrations and outliers in data were logged in a spreadsheet, and efforts were sometimes made to re-process data that had been flagged for poor quality. Special attention was given to making sure that the skull-stripping for both structural and functional had been performed properly, that registrations were acceptable, and that data had been adequately corrected for motion. Motion, in particular, can significantly bias estimates of time series correlations. As such, motion parameters, statistics and graphs plotting framewise displacement (FD) and delta variation signal (DVARS) over time were looked at carefully to ensure that data was acceptable for further analyses. Ultimately, one subject had to be excluded due to the mean displacement of the subject's data greatly exceeding the absolute threshold of 2mm. For this subject, motion correction using ME-ICA also failed to correct for movement and the variation in BOLD signal.

Finally, after QA, we extracted the BOLD time series from specific pre-selected regions of interest (ROIs), using the denoised fMRI data and the precomputed and checked spatial normalization to the MNI template. These time series were combined in a comma separated value file and moved to a separate computer for Tara Madhyastha to process using R scripts.

Although this processing pipeline is well set up for us to replicate our own analysis, it is less straightforward to share. Raw data is not currently online, although it will ultimately be archived as part of the Adult Changes in Thought study. All main software to execute the pipeline is available online. However, there are some minor scripts that we have written that are not available online. To fully replicate the pipeline off-site, we would need to supply these scripts and the versions of all software that we used, along with the Makefiles. It would be easiest to do this by supplying a clean copy of the working analysis directory for the subjects from our site, so that the remote site could edit pathnames to specific packages as necessary and rerun our workflow.

Documentation for Makefiles is embedded in our Makefiles either using comments (because we have relatively standardized target names and file naming conventions). We have been recently trying to adopt standards for documentation of naming conventions and help systems. Thus far, however, processing of primary targets has been relatively intuitive and this has not been as important as developing extensive QA and provenance.

Processing is not currently online because the workflows are too computationally expensive to run on a single multicore server, and we often have to trade off disk storage for processing power when deciding where and how to parallelize. We also lack programming resources to develop web-based interfaces for these pipelines. Writing and running the makefiles is less difficult than determining what they should look like, so web-based sharing of workflows has not been a priority.

Pain points

An important part of neuroimaging workflow is checking the quality of automated processing steps. However, when manual corrections are necessary it is less clear how to record them in such a way that they can be completely replicated. We currently maintain a spreadsheet with this information, but clearly corrections introduce problems with reproducibility.

Generating figures for papers can involve significant handwork. For example, to assemble a montage of brain slices that show statistically significant results might involve generating several screen dumps of different coordinates, setting the minimum and maximum manually to be consistent across images that will share a common colorbar. Once the researcher has decided upon the images and coordinates to include, this process cannot always be scripted, and the relevant parameters must be carefully recorded.

Tools such as Rmarkdown and pandoc (or Sweave, odfweave) allow fabulous integration of statistical analysis and text. However, researchers in many labs rely heavily upon Microsoft Word's "track changes" feature to edit papers, making it difficult to entirely couple paper generation and statistical analysis. Thus, there is an unnatural separation of generation of methods text and tables from assembly and editing of the final paper.

Key benefits

Our workflow is unique compared to state-of-the-art in the neuroimaging field because we use Make to describe dependencies. This ensures that only the parts of the workflow that need to be re-executed (because of a failure, or the change to an intermediate result) will be rerun. Practically, avoiding unnecessary computation time is important in neuroimaging workflows that can take many hours or days to run on small clusters of computers. Using scripts written in languages that do not inherently express dependencies (such as bash or Matlab) it is easy to introduce errors when modifiying scripts to execute only uncompleted work.

There are two key advantages of Make over other neuroimaging workflow systems (such as nipype and LONI Pipeline) which also support a dependency graph model. First, Make does not require the core neuroimaging programs to be "wrapped", or surrounded by interface code, that adds development time to the process of designing a workflow and slows adoption of new versions of software programs as they become incompatable with their wrappers. Second, Make builds the dependency graph implicitly from target files, rather than requiring the graph to be drawn or programmed explicitly.

Key tools

There are two key tools that support reproducibility in our workflows. The first is Make, described above. The second is R Markdown, which we use here in combination with Make to generate complex QA reports including statistics, QA images, and graphs generated using R. In workflows not described we use R Markdown to generate data provenance reports suitable for inclusion in a methods section, combining English text describing the workflow with parameters obtained automatically that include software versions and software and scanner acquisition parameters.

Questions

What does "reproducibility" mean to you?

Reproducibility in this context means that given the specifics of the software versions that we are running, our workflow, and the source data, a scientist can obtain the same results from our data that we can.

Why do you think that reproducibility in your domain is important?

Many results in neuroimaging are highly dependent upon the methods used to process the data. It is often the case that scientists discover that some source of noise introduces artifactual findings, or that data that were previously dismissed as noise contain information. Therefore, it is vital to maintain a programmatic description of how data are processed so that findings can be replicated with the same data.

How or where did you learn about reproducibility?

The ideas behind make are from computer science classes. Most other practices described here have been developed together in IBIC based on experience.

What do you see as the major challenges to doing reproducible research in your domain, and do you have any suggestions?

Cognitive neuroscience is an inherently interdisciplinary field, which means that the level of technical skill as well as core domain knowledge varies across researchers within a team. Practices encouraging reproducibility (e.g., scripted workflow) that are second nature to members of one field may be completely novel and a bit intimidating to members of another field. This can lead to slow adoption of best practices.

What do you view as the major incentives for doing reproducible research?

One of the major incentives is that it saves our future selves time trying to figure out what we did after we've forgotten.

Are there any best practices that you'd recommend for researchers in your field?

We recommend automated, scripted workflow as much as possible with minimal hand-editing to avoid human-induced bias (e.g., in boundary editing). Time invested in developing clear, consistent, and maintainable workflow is rarely misspent.

Would you recommend any specific resources for learning more about reproducibility?

The Organization for Human Brain Mapping (OHBM) recently formed a Committee on Best Practices in Data Analysis and Sharing (COBIDAS) to identify best practices of data analysis and data sharing in the brain mapping community. The committee is expected to publish a final report on these practices in the near future, and this may prove to be a useful source for individuals interested in learning about reproducibility in the context of neuroimaging research.

References

Askren, M., McAllister-Day, T., Koh, N., Mestre, Z., Dines, J., Korman, B., … Madhyastha, T. M. (2016). Using make for reproducible and parallel neuroimaging workflow and quality assurance. *Frontiers in Neuroinformatics*, *10*(2).

Bach, B., Shi, C., Heulot, N., Madhyastha, T., Grabowski, T., & Dragicevic, P. (2016). Time Curves: Folding Time to Visualize Patterns of Temporal Evolution in Data. *IEEE Transactions of Visualization and Computer Graphics*, *22*(1).

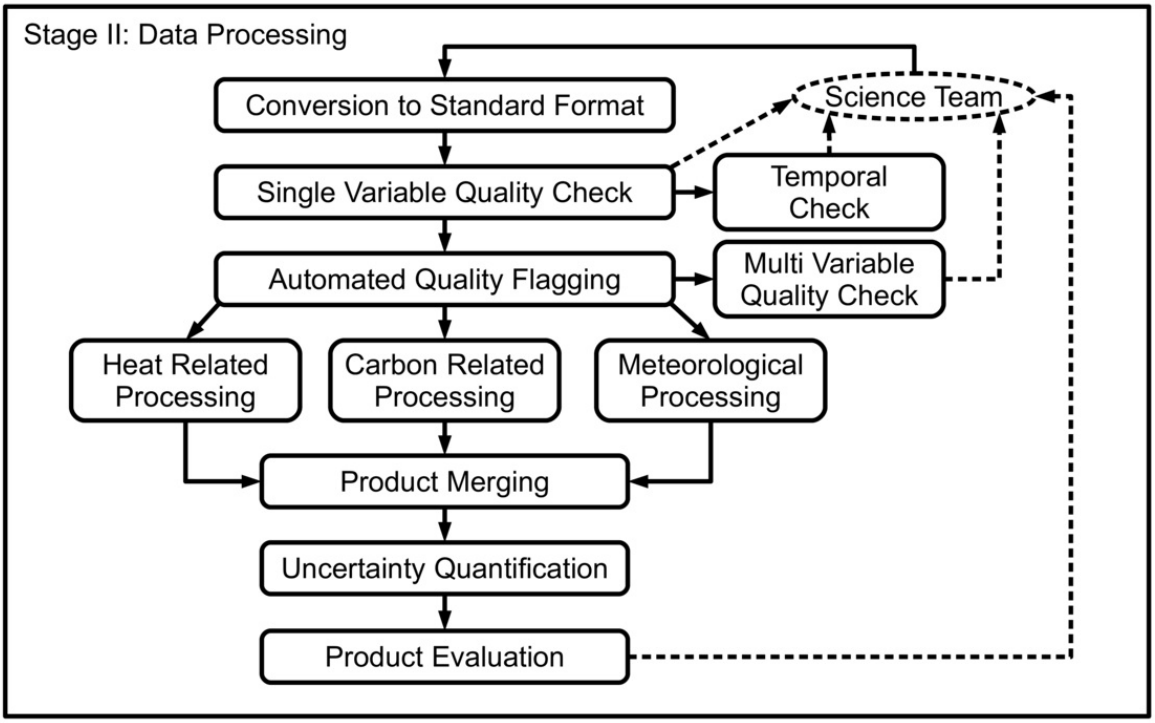
Kundu, P., Inati, S., Evans, J., Luh, W.-M., & Bandettini, P. (2012). Differentiating BOLD and Non-BOLD Signals in fMRI Time Series Using Multi-Echo EPI. *Neuroimage*, *60*(3), 1759–1770.

Generation of Uniform Data Products for AmeriFlux and FLUXNET

Gilberto Pastorello

I'm Gilberto Pastorello, a Research Scientist at Lawrence Berkeley National Laboratory, doing research on life-cycle management of scientific data, encompassing data and metadata structures and linkages, data quality and data uncertainty quantification, and end-to-end data systems. The part of my work described here involves development of data processing pipelines and data management solutions within the environmental domain. This work is done for the AmeriFlux and FLUXNET research networks.

Workflow



Multiple Science Teams collect carbon, water, and energy fluxes from over 800 field sites across the world. Currently, more than 400 of these sites share their data with regional networks such as AmeriFlux, allowing the creation of data products with a global scope for

the FLUXNET network, such as the FLUXNET2015 dataset. These sites are operated independently and methods for data collection, processing, and data quality control by the Science Teams can vary significantly. Our workflow aims at processing these heterogeneous datasets to generate data products that are comparable across these sites. A general view of the steps in our workflow is shown in the figure. In this context, reproducibility is strongly related to identifying and documenting data quality control checks, parameterizations for processing, sequences of correction steps, and data filtering.

The pipeline is executed every time new data are sent to us by the Science Teams. We keep track of multiple submissions with a combination of simple incremental counters for versions and timestamps -- logs of data transfers are stored in a relational database. The frequency of data submissions can range from daily to yearly updates from a Science Team. All executions of the pipeline generate a version, with successful executions being made public after being vetted by the Science Teams.

The first few steps are related to data quality and aim at identifying serious quality issues and making data quality more uniform across sites. Specialized algorithms and visualization methods are used in these steps. Automated generation of flags and manual inspection of datasets are done first, followed by the compilation of metrics about the data and checks. Any decision to change the datasets is first shared with the Science Team and confirmed before being executed. Major issues are identified and solutions are developed in collaboration with Science Teams. A custom issue tracking system developed by us is used to keep track of interactions with Science Teams and the issues being addressed in the datasets. The information in this issue tracking system is private and accessible only to our team and the Science Team for the field site providing the data, but reports and status information from this system can be made available with a published data product. Changes to data are also documented in log files and quality flags that are added to the datasets themselves, these being made available along with data products. Changes and corrections generate a new version of the input data and the pipeline starts again from this new version.

The central portion of the workflow includes the processing steps for heat, carbon and micrometeorological variables. The parameters used to configure these executions are stored with the datasets, and specialized checks are also executed within each step. The results from most of these checks are stored as quality flags added to the datasets. Versions of the code used for these steps are also recorded in the dataset's metadata.

The product merging step reformats the data into common structures and combines quality information into quality flags at a higher level of abstraction, to simplify using the datasets. The uncertainty quantification step generates quantiles representing uncertainty intervals originating from each of the processing steps, also becoming part of the data products.

The more multiple team members understand and can generate products, it is much more likely that problems will be identified early, questions from external members of the community will be answered more easily, and funding agencies will have documented product releases (data and software), which can be combined with other types of publications in assessments of scientific impact.

Pain points

The execution of this pipeline can happen several times before acceptable results are reached, and it also includes several interactions with the Science Teams potentially resulting in changes to the input datasets. The changes can be applied by the Science Teams or by our team. With multiple iterations, the changes that are needed to make a dataset correct are often spread across many versions of the input data. Consolidating these changes is particularly difficult, especially when the changes were applied to versions leading to unsuccessful runs.

More specifically, one version related challenge is keeping versions consistent across multiple processing instances. We use mapping of versions of inputs to outputs, but with outputs influencing what a new version of the dataset looks like, these mappings are not always straightforward.

Another challenge is related to the multi-source nature of our datasets. Many of these datasets span over a decade of data collection, processing, and curation by multiple people. Since the teams are distributed and follow potentially different data collection and processing protocols, fully automated reproducibility is difficult to be achieved. A workaround has been to document the choices in collection and processing data in such cases.

Finally, combining reproducibility issues with credit issues for datasets is another challenge. Giving proper credit to data providers and data processors/curators is an essential part in large data sharing efforts. Comprehensive data sharing policies help with assigning proper credit. Multiple versions of the datasets are created over time, and close communication with the Science Teams is important to allow tracking all contributors.

Key benefits

Without the use of versioning and issue tracking coordinated between code and data, it would be unfeasible to fully document choices for the datasets, many of which affect important aspects such as data quality or uncertainty.

Key tools

Data and software versioning are certainly the main practices adopted in this case study. We are currently assigning versions to data and software in a coordinated way, to allow assessment of changes to data and code. We use a private Subversion server for code version control. We developed a custom Web-based system called FIT (Flux Issue Tracking) for tracking the interactions with Science Teams and data quality issues for datasets. FIT was later also adapted for tracking code related issues. It was implemented in Django/Python using PostgreSQL databases.

For the processing pipeline, we combine specialized code written multiple languages. Automated quality checks and data product generation steps mostly use C implementations, with a couple of steps implemented in Python and MATLAB; visual data quality checks are implemented in MATLAB and Python; and driving code and generation of final data products is done using Python, with extensive use of the NumPy and SciPy packages.

Questions

What does "reproducibility" mean to you?

For this case study, reproducibility means being able to apply standard methods to process heterogeneous datasets to generate comparable data products. The data sources for our processing are distributed across very different ecosystems, with data acquired, processed, and quality checked by different teams. The data products we generate from these datasets need to be in the same scales and have comparable levels of quality, representativity, etc.

Are there any best practices that you'd recommend for researchers in your field?

While version control is widely adopted for software code, this is not necessarily the case for data. Consistently keeping track of versions of a dataset at a minimum helps with data changes and is well worth the extra effort. Another lesson we learned early was that issue/bug tracking ideas and tools can simplify data management activities, and can also be valuable in building the history of a dataset and generating its documentation.

Would you recommend any specific resources for learning more about reproducibility?

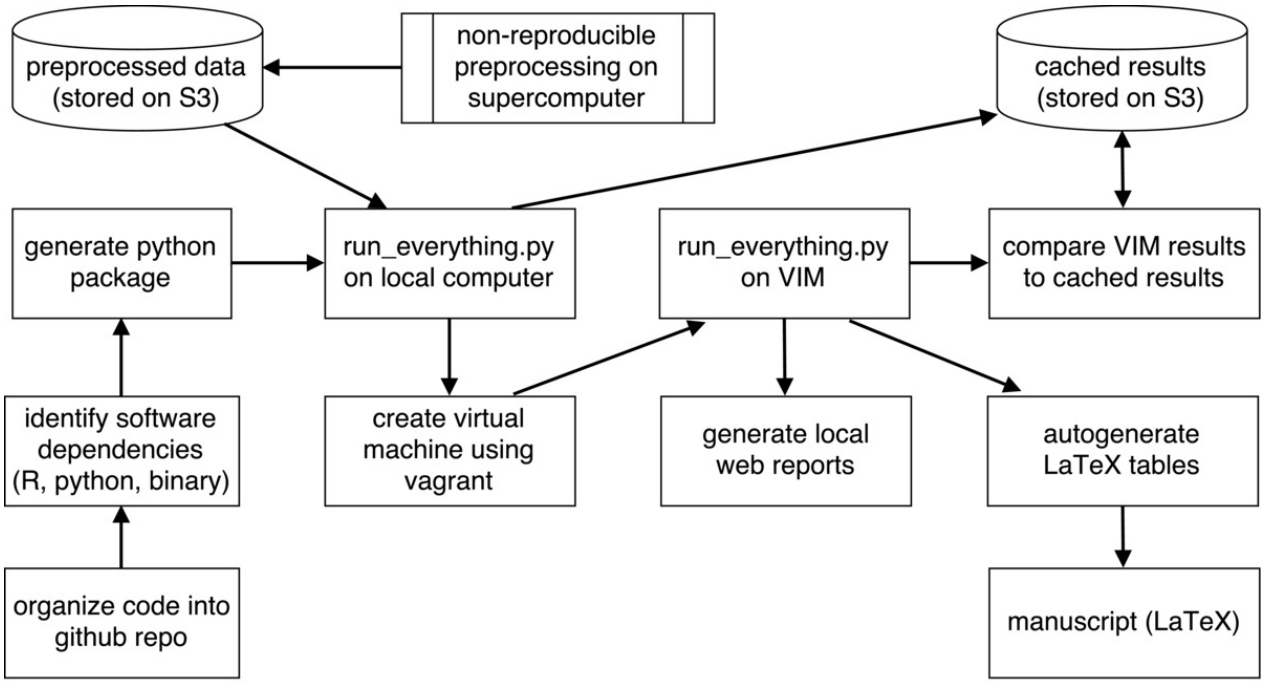
While not always featuring fully developed practices or tools, reproducibility scientific events (and their published proceedings) often showcase new and interesting ideas. Events featuring latest research in provenance include, for instance, the International Provenance and Annotation Workshop (IPAW) and the International Conference on eScience.

Developing a Reproducible Workflow for Large-scale Phenotyping

Russell Poldrack

My name is Russell Poldrack, and I am a professor in the Department of Psychology at Stanford University. My work uses neuroimaging, genomics, and behavioral studies to examine the brain systems involved in decision making and executive control. Many of our workflows use high-performance computing due to the large data and complex nature of the workflows. This particular case study focuses on analysis of a study known as the "MyConnectome study", which involved intensive data collection from a single individual over the course of 18 months, including neuroimaging, genomic, metabolomic, and behavioral data. This large heterogenous dataset raised a number of new challenges for reproducible data analysis.

Workflow



Stage III: Data Analysis

This workflow is meant to outline the analysis of a complex dataset including neuroimaging, behavioral, transcriptomic, and metabolomic data (http://www.myconnectome.org). The data were collected over the course of 18 months from a single individual, and will be made fully available online upon publication of the manuscript via the OpenFMRI project. The data are

released under a public domain dedication, meaning that anyone can do anything they wish with the data, with no restrictions on redistribution or requirements of attribution. I chose this approach because I feel that it will provide greatest degree of utility for the data.

The data processing stream was initially built in a non-reproducible manner on a single laptop. After completing this, I became interested in generating a reproducible version of the workflow so that other researchers can exactly reproduce the analyses. A challenge of reproducible analysis in this study is that the raw data are very large (several TB including the raw genomic and neuroimaging data). These data are being made available to any who wishes to use them, but it is not possible to easily provide reproducible workflows for these operations because they require large-scale supercomputing resources. For the processes that require supercomputing (such as genome alignment for RNA-sequencing data and surface-based parcellation for MRI data), we have shared most of the code used to complete these workflows. Another complication of full reproducibility is that some of the preprocessing operations were performed by another laboratory, using code that they are not currently willing to share openly. Thus, we made the decision to focus on building an open reproducible workflow that encompasses as much as possible of the processing stream, using preprocessed data downloaded from an online archive.

The goal of this process was to create a completed automated analysis stream that requires no manual intervention. The workflow uses a number of tools, including python, R, MATLAB, and the Connectome Workbench (a domain-specific software tool for analysis of neuroimaging data). I started by generating scripts to perform each of the operations, but ultimately decided to generate a single python package to coordinate the entire workflow (available at https://github.com/poldrack/myconnectome). The first operation of this package is to download the preprocessed data from Amazon S3, using the boto package in python. We then perform additional processing of each of the different data types.

For the neuroimaging data, we developed a set of python functions to extract and summarize connectivity measures between different brain regions. These analyses included assessment of connectivity between regions using both standard correlation measures as well as regularized partial correlation (using the R QUIC package). Network analyses were performed using the Brain Connectivity Toolbox, and visualized using the Cytoscape software package.

For the transcriptome and metabolomic data, analyses were performed using a set of Rmarkdown scripts executed using R. These analyses included the identification of coexpression networks using the Weighted Gene Coexpression Network Analysis (WGCNA) package; eigengenes identified from these network were then used in subsequent analyses. The gene networks were annotated using DAVID. Metabolomic measures were clustered using affinity propagation, and annotated using IMPALA.

The outcomes of each of the foregoing analyses were saved and used to compute time series correlations between each of the measures across all domains (for a total of more than 20,000 statistical tests), using the R forecast package. These are then summarized in a web report generated using Rmarkdown. Currently the only test is one that compares the results of the full workflow to a set of results cached on S3. Documentation of the code is minimal.

After implementing the workflow on a single system, I then implemented it on a virtual machine in order to allow anyone anywhere to run it. I used the Vagrant software package to provision a virtual machine with all necessary requirements. Once installed, the user can run the entire workflow with a single command. In addition to running the entire data analysis workflow, the virtual machine also includes a web server that provides access to the results of all of the analyses, along with a data browser for the detailed results. This system is identical to the one exposed publicly at http://results.myconnectome.org. Documentation for installing and running the software is evolving.

Pain points

A number of pain points were encountered in the development of a reproducible analysis workflow. First, there were a number of processing stream operations that could not be implemented in this manner. In particular, some of the preprocessing operations required high-performance computing resources, which could not be generalized due to specifics of job submission systems. Second, there was a substantial amount of extra work necessary to generalize the code to work on an arbitrary system, primarily involving the identification and resolution of software dependencies. A third pain point involved the identification of appropriate technologies for sharing of a reproducible workflow. We have used a VM provisioned using Vagrant, but there are many other approaches that one might use. Finally, we struggled with identifying what level of user at whom we were targeting the project. We ultimately decided to make it easy to use for non-power-users, which required substantial extra work.

Key benefits

The reproducible workflow provides a number of important benefits. First, it provides a degree of detail that could not be feasibly included in the publication. Second, it increases the degree of trust amongst others in the field in the results that are presented in the publication. Third, it provides an example for others in our subfield of how to implement reproducible shared workflows.

Key tools

I have used Vagrant to allow any user to easily provision a virtual machine that includes all of the necesssary dependencies to run the workflow (see https://github.com/poldrack/myconnectome-vm).

Questions

What does "reproducibility" mean to you?

In the context of my case study, "reproducibility" means the ability to exactly reproduce the analysis workflow that was used to obtain the results reported in a manuscript. More generally, I take the term to also encompass the consistency of results across different workflows or datasets.

Why do you think that reproducibility in your domain is important?

The workflows used for neuroimaging data are highly complex with a great degree of analytic flexibility, which raises concerns regarding the reproducibility of results. We desperately need a greater degree of transparency in order to make research in our domain more reproducible.

How or where did you learn about reproducibility?

I think I primarily learned from negative examples; that is, from seeing other researchers whose research findings rely upon code that is not openly available and data that are not shared.

What do you see as the major challenges to doing reproducible research in your domain, and do you have any suggestions?

Human subjects issues and concerns about scooping are often raised here, but I think these are red herrings. The primary pitfall is that reproducible research practices make it harder to obtain splashy findings that get high-profile publications.

What do you view as the major incentives for doing reproducible research?

Increasing trust in one's research.

Are there any best practices that you'd recommend for researchers in your field?

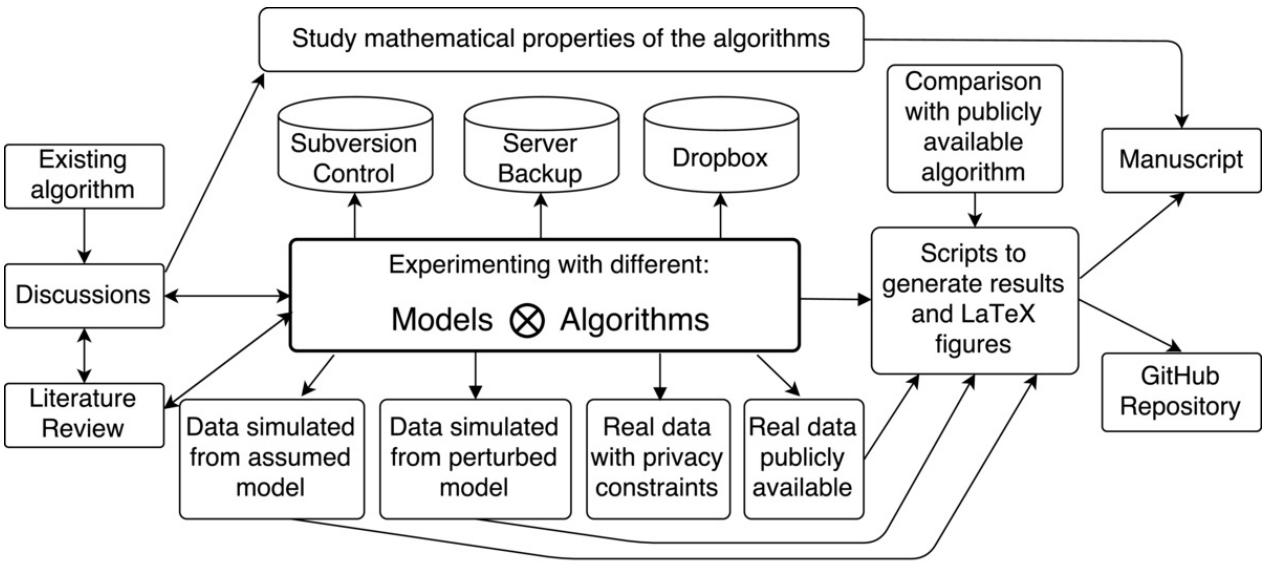
Using version control software, and open sharing of data.

Developing and Testing Stochastic Filtering Methods for Tracking Objects in Videos

Valentina Staneva

My name is Valentina Staneva and I work as a data scientist at the eScience Institute at University of Washington. I am an applied mathematician who develops methods to extract information from diverse datasets. Most of my experience is in the domain of image processing and its biomedical applications. This case study describes the workflow of a particular project whose goal was to develop and test new algorithms for tracking objects in videos which aim to preserve the original structure of the objects. This research was done while I was a graduate student at the Center for Imaging Science, at Johns Hopkins University, and was motivated by the task of tracking heart motion in cardiac images. I believe it reflects a typical experience of an applied mathematician working on biomedical imaging problems.

Workflow



Stage II: Data Processing

This work follows a typical flow for problems in my field: motivated by an existing algorithm, we aimed to extend it to processing sequences of images (as opposed to a single image). Usually the process involves experimenting with different models of the data and different

inference algorithms (intertwined with discussions with my advisor and literature reviews). The various combinations of these models and algorithms can be tested on three main types of datasets:

- a dataset simulated from the assumed model: since the dataset is coming from the "correct model", this experiment is mainly testing the performance of the inference algorithms
- a dataset simulated from a model which deviates from the assumed model (in some interpretable way): this experiment is testing the robustness of the algorithms
- a 'real' dataset: a video of a moving object; this tests the applicability of this methodology in practice

This results in exploring quite a lot of different setups and most of the research time is spent at this stage (it can take from weeks to several months to implement and test a specific formulation). The work was performed on an account on a university server, which was sequentially backed up. I also used Subversion for version control and stored my files in a Dropbox folder (which has its own version control). Luckily, I never lost a file, even when our server got hacked.

In general the evaluation of these algorithms on real data is difficult. There are no standard testing datasets, and it is hard to design ones as different image sequences describe different processes, and some algorithms perform well in some situations and poorly in others. One usually considers a range of typical tracking hurdles and checks whether a given algorithm can overcome them. Since there is no one final metric to submit this requires storing all the results from all the experiments. In the end I saved all the code (written in MATLAB), data, and experiments in a folder, which provides everything necessary to regenerate the results in our manuscript with a few simple commands. When reviewers requested an additional plot to be included in our article, I could easily obtain it from the original data. We also selected a journal which does not prevent us from posting the preprint of the article elsewhere and stored all the supplementary materials on GitHub.

The workflow also contains a parallel path in which one studies the mathematical properties of the models and algorithms. For example, we aimed to develop algorithms which preserve the topology of the tracked objects, and we proved that our framework ensures that, thus eliminating the need to test this property in multiple cases. Sometimes it is possible to guarantee the performance of algorithms even without implementing them, which makes reproducibility of mathematical research quite easy!

Pain points

1) Private data: some of the motivation for this project was driven by the need to process a specific cardiac dataset and perform statistical analysis on the obtained results. I initially tested the algorithms on this dataset, however, eventually I was not allowed to use it in my publication due to some privacy concerns. I resorted to searching for a public cardiac dataset which turned out extremely difficult to find: a website which aimed to maintain a public database of cardiac images was permanently down as the creator left the field. One good source for public biomedical datasets is MICCAI conference challenges: they contain datasets on which to assess methods for solving very specific problems. The drawback is that sometimes the datasets are not complete, as they are designed to solve a particular challenge: for example, the dataset I obtained from an image segmentation challenge did not contain ground truth relevant to the tracking problem.

2) Volume (Storage): when processing video sequences, an issue simply arises from the size of the produced outputs. If I generate many experiments (which is inevitable when performing MCMC simulation) I have to store many videos. Without a ground truth, I could not store just a small measure of mismatch instead of the whole sequence. GitHub's policy does not accept files larger than 100MB. This caused difficulty when trying to upload even only the input data to the repository. This makes it hard to keep code and data together and easily accessible.

3) Randomness: working with Monte Carlo simulations results in different outputs every time the algorithms are applied. This requires the extra step of forcing the random number generators to produce the same sequence of random numbers. This procedure is not so simple when parallelization is involved: MATLAB (and similar scripting languages) usually do not have control over the order in which the separate threads are started and that results in extra randomness in the output. Further, attempting to generate multiple random streams simultaneously results in producing identical sequences of pseudo-random numbers (the seed is based on the current CPU time) which corrupts the Monte Carlo algorithm. One solution is to generate all random sequences that would be needed in the parallel threads in advance, but this requires modification of the programs themselves. Working with random outputs also makes it hard to generate unit tests: we only have asymptotic results of what the outputs should be, so it is difficult to set the confidence intervals for the outputs even with simulated data.

4) Backup: I used Subversion for version control of this project. I wanted to use the integration with the Nautilus file manager that Subversion was providing. It turned out it was buggy (it was a new feature at that time) and not all commits were recorded through the graphical interface: quite dangerous! I learned that it is more reliable to use explicit terminal commands with version control systems.

Key benefits

One of the main advantages of this workflow is that all the code was written in one language without resorting to external libraries and toolboxes. Usually core language functionality changes much more rarely than add-on packages, which makes software better sustainable in the long run and across platforms.

Our approach of encoding certain mathematical properties into the developed algorithms also makes the research more robustly reproducible under deviations of the original set-up.

Questions

What does "reproducibility" mean to you?

"Reproducibility" has two meanings for me:

(1) "Exactly reproducible" - when a result can be regenerated exactly as suggested given the same set of inputs and parameters. For example: a manuscript is "exactly reproducible" when one can provide some scripts and environment which with a press of a button (or an explicit set of instructions) can generate all the figures and calculations in the manuscript.

(2) "Approximately reproducible" - when a result or similar performance can be generated with similar or different methods than the one proposed on the same or possibly slightly different data. Often in science, the goal is to test a hypothesis and the methods to achieve this do not matter, it is actually better if the same hypothesis is supported through different approaches. Further, the data on which the study was performed might never be observed again, so it is not so important to reproduce the results on these data, but it is important to produce similar results on similar data. We are interested in the robustness of the methods and the conclusions, and a better term may be "robustly reproducible".

This case study directly addresses the first type of reproducibility, but it explores also a bit of the second interpretation.

Why do you think that reproducibility in your domain is important?

I find two main reasons for the importance of reproducibility in my domain:

- Personal: working with image data is often quite involved, and one does not want to do things twice, but it is often necessary, and in that case it is better to have an automated process to repeat experiments.
- Public: there is overabundance of algorithms and studies but it is hard to use them in practice, because there is no simple way to reproduce the results (one usually needs to reimplement the algorithms or redo the studies). So if one wants their research to be

useful outside their own group, they should first ensure it is reproducible.

How or where did you learn about reproducibility?

I have been learning by myself. I believe some short reproducibility workshops would have improved my experience substantially (for example, learning about git/GitHub, virtual environments and light virtual containers).

What do you see as the major challenges to doing reproducible research in your domain, and do you have any suggestions?

Working with biomedical data one often faces privacy and storage challenges. Another problem which I did not encounter but I know is persistent in the field is the use of too many external software packages to preprocess the data: some of them are supported only by specific operating systems, or require manual operation. This makes it challenging to automate the workflow. In an attempt to improve performance on large datasets, researchers often use elaborate C++ programs which are hard to interpret and extend.

What do you view as the major incentives for doing reproducible research?

I think the incentives should be personal and based on the understanding that this would improve the workflow and this is how research should be done. Unfortunately, the time and efforts spent on creating reproducible research are not very well awarded.

Are there any best practices that you'd recommend for researchers in your field?

Be reproducible every day! It is much easier to perform reproducible research than making your research reproducible (after it was already performed).

Would you recommend any specific resources for learning more about reproducibility?

Some useful resources have been compiled by the eScience Reproducibility working group: http://uwescience.github.io/reproducible/

Other resources: https://github.com/Reproducible-Science-Curriculum

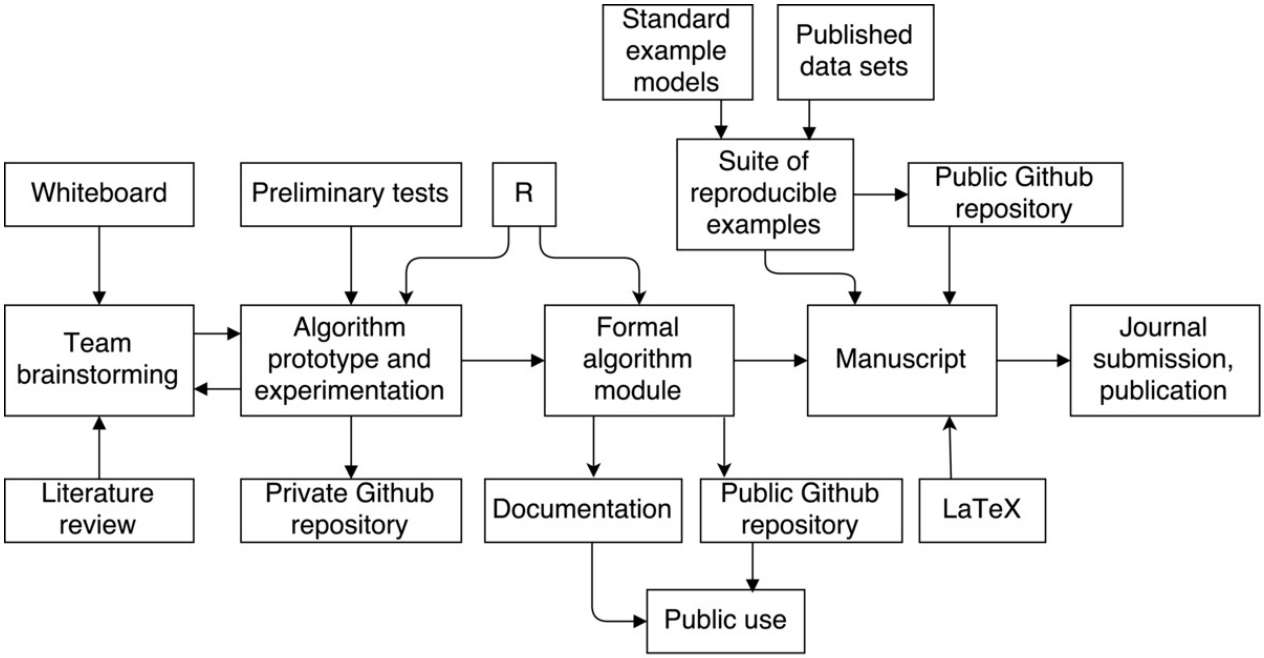
Coursera Class: https://www.coursera.org/course/repdata

Developing, Testing, and Deploying Efficient MCMC Algorithms for Hierarchical Models Using R

Daniel Turek

My name is Daniel Turek, and I'm an Assistant Professor in the Department of Mathematics and Statistics at Williams College. My area of research is computational statistics and algorithms, frequently with applications in ecological statistics. The workflow I describe is the process, from idea inception to publication, of creating an automated procedure to improve the sampling efficiency of Markov chain Monte Carlo (MCMC) sampling. MCMC is an accessible and commonly used statistical technique for performing inference on hierarchical model structures.

Workflow



Stage III: Data Analysis

The process begins with team brainstorming of how an automated procedure for improving MCMC efficiency could work. This is arguably the most fun part of the entire process. This involves anywhere from two to four people actually hitting the whiteboard to discuss ideas. Each of several sessions lasts a few hours. We review theory and literature between these sessions, too. This initial exploration occurs over one or two weeks.

A plausible idea is hatched, and now must be prototyped to assess effectiveness. The project lead implements the algorithm in R, since our engine for doing MCMC runs natively there. This works well for our team, since everyone is comfortable in R, and code may be shared and reviewed easily. We create a private GitHub repository where members of our team write/review/modify the algorithm. This is a private repo amongst us, since it's entirely experimental at this point, and not intended for the public. There is little (or no) documentation at this point.

Multiple iterations are possible at this stage, whereby ideas are implemented and undergo preliminary testing. Depending on the results of each iteration, we go back to the drawing board several times to figure out where the previous algorithm failed, or how it can be improved. Once again, we implement an improved version and test it using a small number of tests we've devised. This part of the process is time consuming, and potentially frustrating, as many dead-ends are reached. The path forward is not always clear. This process of revising and testing our algorithm may take three to six months.

Eventually, this process converges to a functional algorithm. All members of our team are satisfied with the results, and agree the algorithm is ready for a more professional implementation and hopefully publication.

One or two team members (those closest with the MCMC engine) do a more formal implementation of our algorithm. This implementation is added to an existing public GitHub repository, which contains the basis of the MCMC engine for public use. This step should only take a few weeks, since the algorithm is well-defined and finalized. Appropriate documentation is also written in the form of R help files, which are also added to the public repo.

The next goal is to produce a published research paper describing the algorithm and results. Towards this end, we assemble a suite of reproducible examples. These come from known, standard, existing models and published datasets, which are chosen as being either "common" or "difficult" applications of MCMC. A new public GitHub repository is created, and these example models and datasets are added in the form of R data files. Additionally, bash scripts for running our new algorithm on these examples are added, and also a helpful README file. The sole purpose of this repo is to be referenced in our manuscript, as a place containing fully automated scripts for reproducing the results presented in the manuscript.

Our reproducible examples include fixing the random number generator seed in the executable scripts, thus we can guarantee the same sampling results for each MCMC run (otherwise, a stochastic algorithm). However, the exact *timing* of each MCMC run will vary between runs and computing platforms, and hence the final measure of efficiency will vary, too. Thus, the exact results are not perfectly reproducible, but vary approximately 5% between runs.

Team members jointly contribute to drafting a manuscript describing our new algorithm, which presents the results from the suite of example models. This is jointly written by team members using LaTeX. The manuscript specifically references the repository of reproducible examples, and also explains the caveat in exact reproduction of the results — namely, that they will vary slightly from those presented, and why. The reviewers are nonetheless thrilled with the algorithm and reproducible nature of our research, and readily accept the manuscript for publication.

Pain points

The iterative process of devising and testing our algorithm is not well-documented or particularly reproducible. The only saving grace is that GitHub is used for versioning control, so in theory we could look backwards at previous work, if necessary. But in practice, the commit messages are short and not very descriptive, since everything is experimental at this point. No less, there's basically no documentation accompanying our code. It would be difficult to actually review previous versions of the algorithm or results, if it were necessary.

In addition, the fact that our set of "reproducible" examples are not perfectly reproducible is a small point of contention. We are conflicted to call these examples reproducible, since the results presented in our manuscript cannot actually be recreated. Team members agree that this appears to be unavoidable. We explain this in the manuscript, and call our results "reproducible" nonetheless.

For preparation of our manuscript, numerical results are manually typed into a tex document. Tools such as knitr and sweave exist for automating this process, which automatically incorporate numeric and graphical results directly from R into LaTeX. We opted not to use these tools to automate the interaction between R and our manuscript, since not all team members are familiar or comfortable using these tools. Preparation of the manuscript would have been simpler and less error-prone had we used these tools, which probably would have been a wise decision, but the learning curve deterred our team from doing so.

Key benefits

The most notably reproducible aspect of this project is the public repo containing input data and scripts for re-running all analyses appearing in our manuscript. This includes individual bash scripts for running each particular analysis, as well as a single "master" script which re-runs all analyses. A reviewer can easily reproduce (to within a small margin of error) all numerical results appearing in the manuscript, and researchers reading the ensuing publication have an easy path forward to using the algorithm themselves.

Questions

What does "reproducibility" mean to you?

In the context of my case study, reproducibility means that users / reviewers can re-create the results (improvements in MCMC efficiency) presented in our manuscript. However, the results will not match exactly due to small differences in algorithm runtime.

Why do you think that reproducibility in your domain is important?

Reproducibility is important so that others may verify the results given in our publication. This ensures that the results are genuine, and also gives a clear path forward for others to use our algorithm.

How or where did you learn about reproducibility?

Mostly through the use of GitHub, from colleagues at the University of California, Berkeley, and through general programming experience. No specific classes or workshops come to mind.

What do you see as the major challenges to doing reproducible research in your domain, and do you have any suggestions?

In the area of computational statistics, there are not many barriers to reproducible research aside from ignorance or technical inability. However, this case study does highlight one genuine obstacle: that of performance differences between various machines and computing platforms, which will affect algorithm runtime, which factors into our measure of efficiency.

What do you view as the major incentives for doing reproducible research?

Primarily so that others may actually (and easily) verify our results, if they so choose.