STAT259 Summary 1 Mitch Negus
9/7/2017

Developing open source scientific practice

Millman, K. and Perez F.

Summary

One of the fundamental tenets of science is that it is reproducible. An experiment performed under the same circumstances as the original ought to produce comparable results. This week's article addresses how the practice of science in today's world also almost universally requires some form of computation; however, without the proper systems in place, reproducing computational science can be prohibitively difficult, if not impossible. For science to remain reproducible, the article stresses that scientists must adopt tools and strategies similar to those used by large, open-source software projects. Among these suggestions, the authors stress that scientists must become familiar with using version control for documents and code, should become accustomed to automating workflows wherever possible, and ought to make themselves comfortable using collaborative devices to enhance communication and productivity. The authors argue that many open-source software projects have already developed the underlying framework for executing transparent, reproducible, and collaborative products. While there are some challenges that are clearly apparent (such as authorship claims and first-to-publish recognition), basing the future practice of science on these methods will allow science to be both thorough and accessible as it transitions into a computational age.

Exploration

I found the section in the article discussing how most tools commonly used by scientists create discontinuities across the scientific life-cycle to be fascinating. While I had felt this phenomena on my own (in fact it has been a large influence on my interest in the field of reproducible scientific practices) I had not been able to summarize it so succinctly. I had always noted that certain programs did not "play nicely" and was thrilled when I found some that could work well together (or better yet, executed several steps in the scientific process), but I had not recognized explicitly that these were each covering only distinct portions of the scientific life-cycle. I'm curious to now apply this idea to some of the codes I am working with currently in my graduate research. Some of those codes include functionality to proceed from project inception through to visualization in publication-ready visuals, while others require more manual work to proceed through the scientific cycle. I think that being able to identify programs that leave gaps in this process will cause me to become a better developer and work to avoid these pitfalls, instead creating far more integrated, and therefore valuable and reproducible, scientific computing aids.

Notes

–COMPUTATIONAL SCIENCE–

REPRODUCIBLE SCIENCE
- for good science, it must be reproducible
- open source software community has developed this framework/workflow
- though feasibility of reproduction may be limited, the possibility of reproduction must be demanded

LIFE CYCLE
*individual
*collaboration
*production
*publication
*education

-Most common tools create discontinuities across stages of workflow
-Results are considered separate from process, rather than a unified science product
-Joining tools and stages requires both technical and social changes

Reproducibility must be a commitment from the start

OPEN SOURCE
-moving science forward requires computational literacy–¿ science is open source –¿ contributing to science will require open source practices
challenges exist with making science fully open source (author recognition, first to publish, etc.)

–PRACTICE–

VERSION CONTROL
-files stored in repositories, require commits (w/ message)
-allow branching and merging
-modern systems allow data integrity verification (cryptographically fingerprinting)
-also limited when dealing w/ large binary files (solutions being developed in this regard)

AUTOMATED EXECUTION
-reproducibility should extend to process; best to automate all steps when possible -still should be able to be understood by people
-make files facilitate this process

TESTING -testing should accompany product development (test-driven-design) -allows focus on use (rather than details) -TDD prevents "getting lost" in tangled code web
READABILITY
-you and others will read your code (esp. to verify results)

-self-doc code reduces external documentation by being clear and forward
-use the right level of abstraction when writing mathematical expressions (don't simply too much, but don't avoid it entirely)
-comments may be uncoupled from code (one changes, the other is not updated)
-use of docstrings allows coupling of docs to code–¿ then autogeneration for web

INFRASTRUCTURE
-hosted version control allows group collaboration
-continuous integration to automatically execute test-suite

PULL REQUEST
-pull request akin to peer review
-anyone can chime in, lasting document of decisions
-private branches (maintain credit, history, and privacy while allowing transparency after integration

-linear algebra book by Rob Beezer (U. Puget Sound)