Best practices for scientific computing

Mike Jackson
EPCC
michaelj@epcc.ed.ac.uk

Software makes supercomputers useful

- “software development …[is]… often discounted in the scientific community, and programming is treated as something to **spend as little time on as possible**”
- “Serious scientists are **not** expected to carefully test code, let alone document it, in the same way they are trained to properly use other tools or document their experiments”
- “It has been said … that writing a large piece of software is akin to building infrastructure such as a telescope rather than a creditable scientific contribution…”
- “software reflects the **intellectual engine** that **makes the supercomputers useful**, and has scientific value beyond the hardware itself.”
  - Stodden, V., Bailey, D. H., Borwein, J., LeVeque, R.J., Rider, W., Stein, W. “Setting the Default to Reproducible Reproducibility in Computational and Experimental Mathematics”, ICERM 2013, February 2013.

Best practices for scientific computing

Wilson, G., Aruliah, D.A., Brown, C.T., Chue Hong, N.P. and Davis, M. “Best Practices for Scientific Computing”, PLoS Biol 12(1): e1001745, January 2014. doi:10.1371/journal.pbio.1001745.

“Scientists spend an increasing amount of time building and using software. However, most scientists are never taught how to do this efficiently. As a result, many are unaware of tools and practices that would allow them to write more reliable and maintainable code with less effort. We describe a set of best practices for scientific software development that have solid foundations in research and experience, and that improve scientists' productivity and the reliability of their software.”

Best practices for scientific computing

1. Write programs for people, not computers
   a) A program should not require its readers to hold more than a handful of facts in memory at once
   b) Make names consistent, distinctive, and meaningful
   c) Make code style and formatting consistent
2. Let the computer do the work
   a) Make the computer repeat tasks
   b) Save recent commands in a file for re-use
   c) Use a build tool to automate workflows

Best practices for scientific computing

3. Make incremental changes
    a) Work in small steps with frequent feedback and course correction
    b) Use a version control system
    c) Put everything that has been created manually in version control
4. Don't repeat yourself (or others)
    a) Every piece of data must have a single authoritative representation in the system
    b) Modularize code rather than copying and pasting
    c) Re-use code instead of rewriting it

Best practices for scientific computing

5. Plan for mistakes
    a) Add assertions to programs to check their operation
    b) Use an off-the-shelf unit testing library
    c) Turn bugs into test cases
    d) Use a symbolic debugger
6. Optimize software only after it works correctly
    a) Use a profiler to identify bottlenecks
    b) Write code in the highest-level language possible

Best practices for scientific computing

7. Document design and purpose, not mechanics
   a) Document interfaces and reasons, not implementations
   b) Refactor code in preference to explaining how it works
   c) Embed the documentation for a piece of software in that software
8. Collaborate
   a) Use pre-merge code reviews
   b) Use pair programming when bringing someone new up to speed and when tackling particularly tricky problems
   c) Use an issue tracking tool

Ten simple rules for reproducible computational research

Sandve, G.K., Nekrutenko, A., Taylor, J. and Hovig, E. "Ten Simple Rules for Reproducible Computational Research", PLoS Comput Biol 9(10): e1003285, October 2013. doi:10.1371/journal.pcbi.1003285

1. For every result, keep track of how it was produced
2. Avoid manual data manipulation steps
3. Archive the exact versions of all external programs used
4. Version control all custom scripts
5. Record all intermediate results, when possible in standardized formats
6. For analyses that include randomness, note underlying random seeds
7. Always store raw data behind plots
8. Generate hierarchical analysis output, allowing layers of increasing detail to be inspected
9. Connect textual statements to underlying results
10. Provide public access to scripts, runs, and results