Literate Programming and Reproducible Research in Computational Science

Randall J. LeVeque
Department of Applied Mathematics
University of Washington

Supported in part by DOE and NSF

Outline

- What is “Literate Programming”?
- What is “Reproducible Research”?
- Why do research reproducibly?
  Some lessons I’ve learned the hard way.
- Some tools
- Examples from my field

Responses to some questions raised after the talk have been added at the end.

See http://www.amath.washington.edu/∼rjl/lprr.html for more links to literate programming and reproducible research tools.

Literate Programming

Donald Knuth:

I believe that the time is ripe for significantly better documentation of programs, and that we can best achieve this by considering programs to be works of literature. Hence, my title: “Literate Programming.”

Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.

From: "Literate Programming (1984)", by D. E. Knuth, in Literate Programming. CSLI Lecture notes, 1992, pg. 99.

WEB and related tools

Knuth introduced WEB in 1981 for documenting TeX source.

Main components:

TANGLE produces Pascal code from text source,
WEAVE produces TeX'ed documentation.

CWEB: Knuth and Levy, for C and C++.

Various other versions for different languages, and language-independent versions such as noweb and FunnelWeb.

Literate Programming and Reproducible Research often go hand in hand.

Many tools and approaches, for example:

- Doxygen documentation generator is widely used,
- AMRITA James Quirk (Los Alamos), "Cross between a document preparation system, a computational engine, and a programming language",
  Using Acrobat 9.0 with many new features.
- Madagascar for geophysical data analysis

Reproducible Research

A crucial aspect of the Scientific Method

The Scientific Method

From Wikipedia:

“Scientific method” refers to the body of techniques for investigating phenomena, acquiring new knowledge, or correcting and integrating previous knowledge. It is based on gathering observable, empirical and measurable evidence subject to specific principles of reasoning.

Another basic expectation is to document, archive and share all data and methodology so they are available for careful scrutiny by other scientists, thereby allowing other researchers the opportunity to verify results by attempting to reproduce them. This practice, called “full disclosure”, also allows statistical measures of the reliability of these data to be established.

Reproducible Research in Computational Science

"Computation" or "simulation" is a young branch of science, complementing

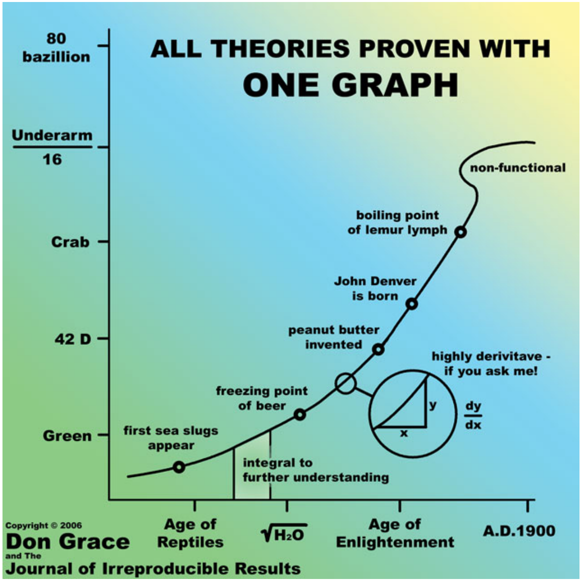
- Theory,
- Experiment (non-computational),
- Observation.

Standards and expectations have not yet matured.

It's an exciting time in computational science!
Progress is rapid, capabilities and techniques evolve quickly.

People in this room should be on the leading edge,
not left behind. Don't become a scientific joke, as in ....

The Journal of Irreproducible Results www.jir.com



Reproducible Research in Computational Science

J. B. Buckheit and D. L. Donoho, *WaveLab and reproducible research*, 1995 (following Jon Claerbout's lead):

An article about computational science in a scientific publication is **not** the scholarship itself, it is merely **advertising** of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures.

Some comments:

- Even more true for talks.

  As in talks on theory or experiments, the paper should contain many details not mentioned in a talk.

- Some papers on computational science should contain all the details. (Others can't.)

Reproducible Research in Computational Science

My own goal:

When a project is complete (e.g. a paper is published), the computational tools and data should be preserved in a manner that allows one to reproduce the final products (e.g., figures, tables, error values) and to later understand the methods used and the implementation (including all parameter values).

The paper should contain all details that can be reasonably included. Other details should be available on-line.

Reproducible Research in Computational Science

Why?

Heart of the “Scientific Method”. Others should be able to attempt to reproduce your results, perhaps from scratch. This is impossible if method is not fully described.

Often many competing methods have been proposed. At some point we must systematically compare them, in order to judge which methods are best for which problems.

Original author often can’t reproduce results at a later time. Parameter values forgotten, codes misplaced, collaborators or students have disappeared, etc.

We should build on past work rather than constantly starting over.

Reproducible Research in Computational Science

Why not?

A lot of extra work. (Especially without proper tools.)

Research is often done piecemeal — some results obtained, code modified, more results obtained (but code no longer capable of producing first set of results), etc.

Computer program is valuable intellectual property with years of work invested. Don't want to share with competitors.

Code isn't pretty enough or sufficiently well organized to share.

Program uses proprietary packages and/or only runs on special hardware.

Some advice, based on 30 years of poor habits:

Begin by practicing RR methodology for your own sake.

- You don't need to make all your codes freely available, but you should archive them for your own later inspection and use.
- Computer codes and input data are generally small by today's standards, can afford to keep many versions.
- Use versioning tools, such as Subversion.
- Keep a "lab notebook" (or electronic equivalent) to keep track of what tests you've done, where to find things, etc.

Some benefits:

- You'll do better work.
- Much less frustration down the road.
- Greater ability to build on your past work (and your students').

Some thoughts on making codes available:

Objection: Code only runs with proprietary software or special hardware.

Response: Code still provides a record of exactly what method was implemented, what parameter values used, etc.

Objection: Code isn't pretty or well organized.

Response: Most research codes aren't.
(Most commercial codes aren't either!)
They need not be beautiful to be useful.

Some thoughts on making codes available:

Objection: Valuable intellectual property.

Response:

Not so easy for others to use/modify even when you try to create user-friendly software.

If others are expected to provide code for publications too, it will be clear if they've used your code.

Most computational experiments raise more questions than answers, sharing can lead to new collaborations.

Your work will be better recognized if your methods are widely adopted by others.

Some tools

Versioning software, e.g.
CVS, Subversion, Mercurial

Python: Object oriented scripting language with many uses,

- Matlab style experimentation (NumPy, SciPy)
  See www.enthought.com
- Open source graphics, visualization
  (matplotlib, VisIt, MayaVi)
- file manipulation (including url's, ftp), text manipulation,
- CGI scripting, web interfaces
- interfacing between different languages (Swig, f2py)
- interfaces to and between legacy codes

Some tools we're developing

mathcode2html.py Modest attempt at literate programming.

html and simple latex commands embedded in comments,
Displayed in web browser using jsMath.
Stays up to date with source code via "make" command.

clawtools module to allow manipulating data, running code, processing output, creating html or pdf's, etc.

For use with CLAWPACK (Conservation Laws Package)
Fortran code for hyperbolic partial differential equations

EagleClaw: Easy Access Graphical Laboratory for Exploring Conservation Laws.   Web interface to CLAWPACK.

Some applications where CLAWPACK has been used

- Aerodynamics, supersonic flows
- Seismic waves, tsunamis, flow on the sphere
- Volcanic flows, dusty gas jets, pyroclastic surges
- Ultrasound, lithotripsy, shock wave therapy
- Plasticity, nonlinear elasticity
- Chemotaxis and pattern formation
- Semiconductor modeling
- Multi-fluids, multi-phase flows, bubbly flow
- Combustion, detonation waves
- Astrophysics: binary stars, planetary nebulae, jets,
- Magnetohydrodynamics, plasmas, relativistic flow
- Numerical relativity — gravitational waves, cosmology

Demos

EagleClaw: preliminary version can be tested soon at
www.clawpack.org

clawtools for running a set of experiments.

clawcode2html used to convert source files to become part of documentation.

Test problem:
One-dimensional advection equation $q_t + uq_x = 0$
True solution: $q(x,t) = q(x - ut, 0)$

Python script to run a series of tests

```
import clawtools

data = clawtools.ClawData()

data.mx = 50
data.order = 1
data.write('claw1ez.data')
od = 'output_mx50_order1'
pd = 'plots_mx50_order1'
clawtools.runclaw(outdir=od)
clawtools.plotclaw(outdir=od, plotdir=pd)

# repeat for other values of mx, and for order = 2
```

Python script to run a series of tests

```
import clawtools

data = clawtools.ClawData()

for order in [1,2]:
   for mx in [50, 100, 200, 400]:
      data.order = order
      data.mx = mx
      data.write('claw1ez.data')
      od = 'output_mx%s_order%s'  % (mx, order)
      pd = 'plots_mx%s_order%s'   % (mx, order)
      clawtools.runclaw(outdir=od)
      clawtools.plotclaw(outdir=od, plotdir=pd)
```

Final comments

Computational science is rapidly evolving.
Higher expectations are coming in journals,
from grant agencies.

Connections to Verification and Validation (V&V) and Uncertainty Quantification (UQ)

Tools are rapidly evolving, stay informed.

Tools may become obsolete, e.g. Python 3.X will break backward compatibility.

May need to archive programming languages, environment as well as codes, data.

Will archives be maintained and readable in the future?

Some questions raised after the talk:

One question concerned regression testing, the idea of having a set test problems that a software package is tested on after any changes are made to insure that a bug fix or enhancement hasn't broken something else. Large ongoing software development projects often do regression tests every night.

It's a good idea to do something along these lines even for small scale projects and tools such as Python can make this easier.

It was pointed out that the development of most scientific computing programs is funded by federal funds, another argument for making them freely available at some point rather than considering them proprietary tools of one research group or the basis for commercialization.

NIH now requires papers published based on funded research to eventually pass into the public domain, and perhaps grant agencies in the future will require that of software.

Some questions raised after the talk:

Other concerns were raised about making code available:

Concern: The authors are then expected to provide maintenance and support for others using it.

Response: One of the first open source software projects was netlib (see www.netlib.org), a repository for numerical analysis software. The disclaimer there states anything free comes with no guarantee.

You might consider putting a similar statement on your webpage, along with a disclaimer to the effect that you are not legally responsible for anything computed with the code, as for example in the GPL (http://www.gnu.org/licenses/).

People generally realize that most of the free things they find are the web are not supported.

On the other hand, if people discover bugs in your code that affect the validity of what is computed, this is presumably useful to know for your own work!

Some questions raised after the talk:

Concern: Others may use your code incorrectly and then publish papers claiming their results are superior based on this work.

Response: This is a danger also (perhaps more so) if they try to compare with your method by implementing it themselves.

In the long run giving access to your code will probably lead to it gaining the reputation it deserves (one way or the other).

Some additional pointers:

Another open source software project that may be of interest is SAGE (www.sagemath.org), which collects a large number (> 100) of open source mathematics packages together with a unified interface based on Python. Symbolic manipulation, numerical methods, and graphics are all included.

A web-based notebook interface makes it easy to use as a literate programming tool. You can try it out online without downloading or installing.

See http://www.amath.washington.edu/∼rjl/lprr.html for more links to literate programming and reproducible research tools.