La reproductibilité dans la recherche assistée par ordinateur

Konrad HINSEN

Centre de Biophysique Moléculaire, Orléans, France
and
Synchrotron SOLEIL, Saint Aubin, France

28 mars 2013

Reproducibility

One of the ideals of science

- Scientific results should be verifiable.
- Verification requires reproduction by other scientists.
- Few results actually are reproduced, but it's still important to make this possible:
    - important for the credibility of science in society (remember "Climategate", cold fusion, ...)
    - important for the credibility of a specific study
      The more detail you provide about what you did, the more your peers are willing to believe that you did what you claim to have done.
- Reproducibility also matters for efficient collaboration inside a team.

Reproducibility in practice

Near-perfect in non-numerical mathematics

No journal publishes a theorem unless the author provides a proof.

Often best-effort in experimental sciences

- Lab notebooks record all the details for in-house replication.
- Published protocols are less detailed, but often clear enough for an expert in the field.
- The main limitation is technical: lab equipment and concrete samples cannot be reproduced identically.

Lousy in computational science

- Papers give short method summaries and concentrate on results.
- Few scientists can reproduce their own results after a few months.

Reproducibility in computational science

We could do better than experimentalists

- Results are deterministic, fully determined by input data and algorithms.
- If we published all input data and programs, anyone could reproduce the results exactly.

But we don't

- Lots of technical difficulties.
- Important additional effort.
- Few incentives.

Goals of the Reproducible Research movement

- Create more awareness of the problem.
- Provide better tools.

Reproducible (computational) Research matters

“Climategate”

In 2009 a server at the Climatic Research Unit at the University of East Anglia was hacked and many of its files became public. One of them describes a scientist’s difficulties to reproduce his colleagues’ results. This has been used by climate change skeptics to discredit climate research.

Protein structure retractions

In 2006, six protein structures (published in Science, Nature, ...) were retracted following the discovery of a bug in the software used for data processing.

For more examples and details:

- Z. Merali, “...Error ... why scientific programming does not compute”, Nature **467**, 775 (2010)
- Science Special Issue on Computational Biology, 13 April 2012

Replicating, reproducing, reusing, ...

Terminology in this field isn't stable yet.

Some people distinguish:

Replication	re-running one's own software with the same input data in order to obtain identical results
Reproduction	verifying results published by someone else using the original authors' input data and the same and/or different software
Reuse	using data and/or software published by someone else to do different studies

Reproducing your own results

I don't remember which version of the code I used to make figure 3.

On my new laptop I get different results.

I thought the parameters were the same, but the curve looks different.

Why did I do that last month?

Complexity

Each number in your results depends on

- the input data
- the source code of the software
- all the libraries the software uses
- compilers/interpreters
- compiler options
- the system software of the computer(s)
- the computer hardware

All of these ingredients change continuously.
Many of them are not under your control.
Hardly anyone keeps detailed notes.

Reproducing published results

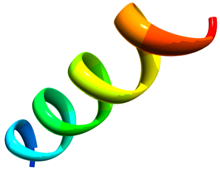
Each number in the published results depends on

- the input data
- the source code of the software
- all the libraries the software uses
- compilers/interpreters
- compiler options
- the system software of the computer(s)
- the computer hardware

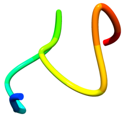
You don’t have most of this information.
You don’t have access to the same hardware and/or software.

A real-life case of (non-)reproducibility (1/5)

Goal: find the most stable gas-phase structure of short peptide sequences by molecular simulation



or



Earlier work on this topic[1] finds that $Ac\,A_{15}\,K + H^+$ forms a helix, but $Ac\,K\,A_{15} + H^+$ forms a globule.

My simulations predict that both sequences form globules.

What did I do differently?

[1] M.F. Jarrold, Phys. Chem. Chem. Phys. **9**, 1659 (2007)

A real-life case of (non-)reproducibility (2/5)

From the paper:

> *Molecular Dynamics (MD) simulations were performed to help interpret the experimental results. The simulations were done with the MACSIMUS suite of programs [31] using the CHARMM21.3 parameter set. A dielectric constant of 1.0 was employed.*

The URL in ref. 31 is broken, but Google helps me find MACSIMUS nevertheless. It's free and comes with a manual! ⌣̈

I download the "latest release" dated 2012-11-09.
But which one was used for that paper in 2007?

A real-life case of (non-)reproducibility (3/5)

MACSIMUS comes with a file `charmm21.par` that starts with

```
! Parameter File for CHARMM version 21.3 [June 24, 1991]
! Includes parameters for both polar and all hydrogen topology files
! Based on QUANTA Parameter Handbook [Polygen Corporation, 1990]
! Modified by JK using various sources
```

Did that paper use the “polar” or the “all hydrogen” topology files?

I can find only one set of topology files named charmm21, and that’s with polar hydrogens only, so I guess “polar”. But that’s not the choice I would have made...

Now I have the parameters, but I don’t know the rules of the CHARMM force field, nor can I be sure that MACSIMUS uses the same rules as the CHARMM software.

A real-life case of (non-)reproducibility (4/5)

From the paper:

> *A variety of starting structures were employed (such as helix, sheet, and extended linear chain) and a number of simulated annealing schedules were used in an effort to escape high energy local minima. Often, hundreds of simulations were performed to explore the energy landscape of a particular peptide. In some cases, MD with simulated annealing was unable to locate the lowest energy conformation and more sophisticated methods were used (see description of evolutionary based methods below).*

I might as well give up here...

My point is not to criticize this particular paper.
The level of description is typical for the field of biomolecular simulation.

A real-life case of (non-)reproducibility (5/5)

What I would have liked to get:

- a machine-readable file containing a full specification of the simulated system:
  - chemical structure
  - all force field terms with their parameters
  - the initial atom positions
- a script implementing the annealing protocol
- links to all the software used in the simulation, with version numbers

All that with persistent references (DOIs).

Tools for reproducible research

We are living in the pioneer phase of reproducibility:

- Most scientific software was not written with reproducibility in mind.
- Very few supporting tools exist . . .
- . . . and none of them deals with all the aspects.
- But we can use many tools originally developed for different purposes.

Today’s tools for Reproducible Research (1/2)

- Version control for source code and data

  Mercurial, Git, Subversion, ...

  Great for source code, usable for small data sets
- Literate programming tools

  Lepton, Emacs org-mode, Sweave...

  Combine code, data, results, and documentation into a coherent document.
- Electronic lab notebooks

  Mathematica, IPython notebook, ...

  Keep track of computational procedures with input and output data.

Today’s tools for Reproducible Research (2/2)

- Provenance trackers

  Sumatra, VisTrails, ...

  Keep track of how exactly results were generated.
- Workflow management systems

  VisTrails, Taverna, Kepler, ...

  Preservation of computational procedures and provenance tracking,
- Publication tools for computations

  Collage, IPOL, myExperiment, PyPedia, RunMyCode, SHARE, ...

  Experimental and/or domain-specific

Missing pieces (1/3)

Integration of version control and provenance tracking

Record for reproduction that dataset X was obtained from version 5 of dataset Y using version 2.2 of program Z compiled with gcc version 4.1.

Version control for big datasets

Version control tools are made for text-based formats.

Provenance tracking and workflows across machines

If you do part of your computations elsewhere (supercomputer, lab's cluster, ...), existing tools won't work for you.

Tool-independent standard file formats

If Alice wants to reproduce Bob's results, she needs to use Bob's tools for version control and workflows/notebooks.

Missing pieces (2/3)

Reproducible floating-point computations

- Reproducibility for IEEE float arithmetic requires an exact sequence of load, store, and arithmetic operations.
- Performance optimization requires shuffling around operations depending on processor type, cache size, memory access speed, number of processors, etc.
- High-level programming languages don't let programmers specify all the details of floating-point operations, in particular not the order of evaluation.

Main issue: conflict between reproducibility and performance

Missing pieces (3/3)

Reproducible parallel computations

- The most popular parallel computation model in science (message passing) is not deterministic.
- Results are in general not reproducible even between two runs of the same program.
- Even in carefully written software, the combination of floating-point non-associativity and parallel non-determinism is a constant source of trouble.

Main issue: conflict between reproducibility and performance

Software engineering techniques (1/2)

Version control

- Use version control for all software development . . .
- . . . including small scripts . . .
- . . . and perhaps also parameter files.

You will never lose the precise version you used for a particular computation.

Testing

- Write tests for all aspects of your software (individual functions, complete applications, ...)
- Distribute the tests with your software.
- Make the tests easy to run on any machine.

Verify that at least some uses of your code are reproducible.

Software engineering techniques (2/2)

Code review

- Have someone else read your source code critically.
- In a team, review each other's code.

Your source code will be more readable and more reliable.

Documentation

- Document what your software does, i.e. describe the scientific approach behind it.
- Document how to use the software.
- In particular, document limitations and assumptions not explicitly checked in the code.

Even people who can't or don't want to understand your code should be able to check if it is used correctly.

Data management

Good data formats

- Formal definitions (data models, ontologies)
- Documentation

Avoid proprietary formats at all costs.

File management

- Directory structure
- File naming conventions
- Avoid overwriting data from previous software runs

Provenance tracking

Goal: Record which result (figure, table, . . .) was generated by which program based on which input files and parameters.

Practice has shown that manual provenance tracking (keeping a log of program runs) is unreliable. Provenance tracking requires dedicated tools.

We will present one such tool (Sumatra) tomorrow.

Publishing

Ideally, there should be data formats for publishing packages of related data, software, and documentation. The traditional paper belongs to the “documentation” category.

At the moment, we have to publish separately:

- a traditional article
- the software
- the datasets

Publishing: code repositories (1/2)

Github

- based on the Git version control system
- encourages open collaboration
- private repositories only for paying customers

http://github.com/

Bitbucket

- Mercurial or Git for version control
- public and private repositories for free

http://bitbucket.org/

Publishing: code repositories (2/2)

SourceForge

- Mercurial, Git, or Subversion
- only Open Source projects

http://sourceforge.net/

SourceSup (Renater)

- reserved for French research/education
- Git or Subversion
- public or private projects
- extensive tool support
- lots of paperwork to get in

http://sourcesup.renater.fr/

Publishing: data repositories

Figshare

- accepts any file
- archives and publishes “forever”
- delivers a DOI

`http://figshare.com/`

Dryad

- works with journal editors, respects journal policies
- accepts data and software that are supplementary material for a published article
- delivers a DOI

`http://datadryad.org/`

Publishing: hosting sites

RunMyCode

- Proposes “companion sites” to a traditional paper
- Stores code for downloading or on-site execution
- Stores example data sets
- Currently accepted languages: R, MATLAB, C++, Fortran, Rats.

`http://www.runmycode.org/`

myExperiment

- Archives and publishes workflows and other files
- On-site execution of Taverna workflows

`http://www.myexperiment.org/`

Recommended reading

- Best Practices for Scientific Computing

  Greg Wilson, D. A. Aruliah, C. Titus Brown, Neil P. Chue Hong, Matt Davis, Richard T. Guy, Steven H. D. Haddock, Katy Huff, Ian M. Mitchell, Mark Plumbley, Ben Waugh, Ethan P. White, Paul Wilson

  `http://arxiv.org/abs/1210.0530`

- Workflows for reproducible research in computational neuroscience

  Andrew Davison (UNIC, CNRS Gif)

  `http://rrcns.readthedocs.org/en/latest/index.html`