Reproducible Science and Modern Scientific Software Development

13th eVITA Winter School in eScience sponsored by Norges forskningsråd
Dr. Holms Hotel, Geilo, Norway
January 20-25, 2013

Best Practices and the Limits of Reproducible Research

Dr. André R. Brodtkorb,
Research Scientist
SINTEF ICT, Dept. of Appl. Math.

Outline

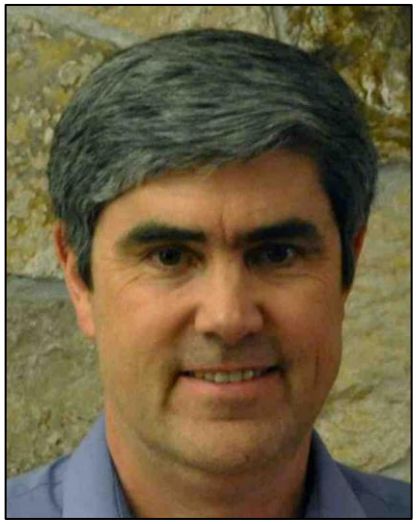
- Why **not** to share code
- Best practices for reproducible research
- Limits of reproducibility

This talk should equip you with:

- Reasons why you should try to be reproducible and share your code
- A check-point list of best practices to consider
- An overview of some situations in which it is difficult to get reproducible results

Top ten list of why not to share code [1]

- A list that would have been better presented by Randy Leveque



Professor Randy Leveque, University of Washington

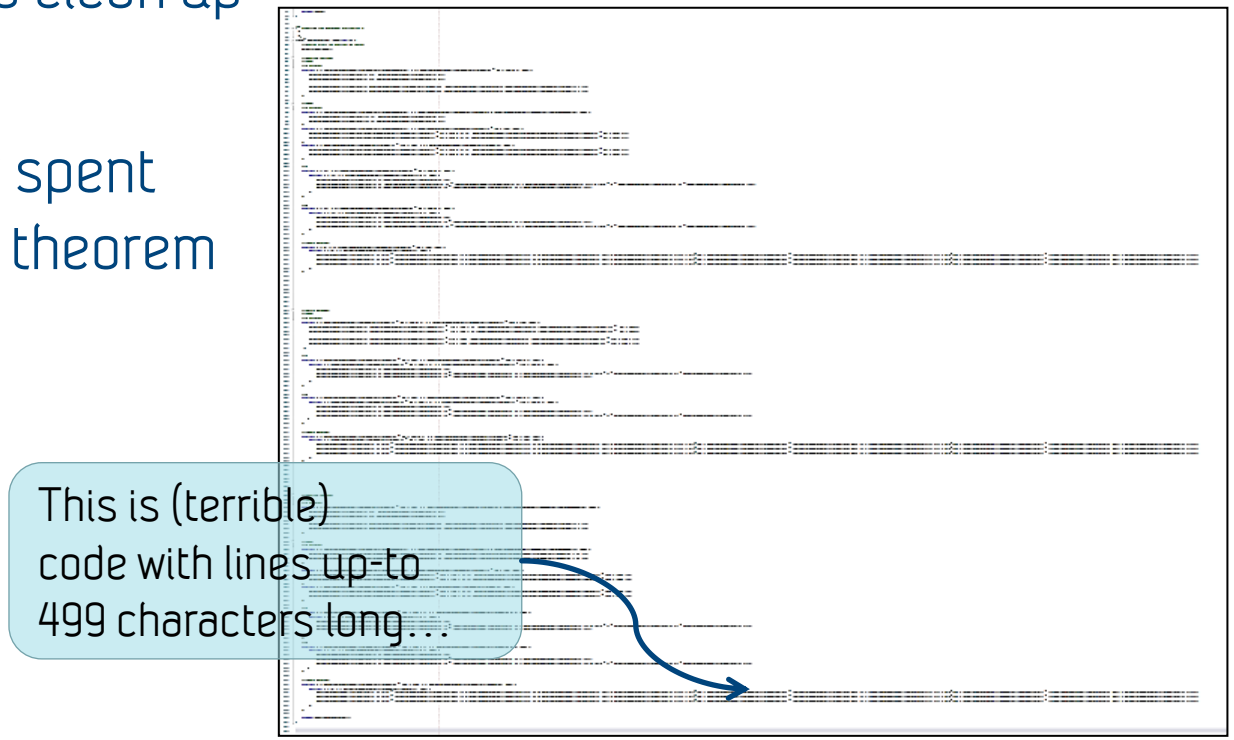
- Imagine a world in which mathematical papers contain the same amount of information as computational papers have today:
  - Papers contain lemmas, theorems, corollaries
  - No proofs are required or expected

- Then some people start demanding proofs to be published.
  What would people say?

[1] from Top Ten Reasons to Not Share Your Code, Randall J. Leveque, 2012
http://jarrodmillman.com/talks/siam2011/ms148/leveque.pdf

Top ten list of why not to share code

1. The proof is too ugly to show anyone else.
   - It's a waste of time to clean up the proof: it's very specific and not worth while to clean up
   - My time is better spent publishing a new theorem



Top ten list of why not to share code

2. I didn't work out all the details.
   - It applies for the examples I use in the paper, that's enough
   - It won't really work for all corner cases, but that's not important

3. I didn't actually create the proof myself, my student did.
   - And the student went into industry so I don't really have it anyway
   - But he was a good student, and I'm pretty sure it's correct

Top ten list of why not to share code

4. Giving the proof to my competitors would be unfair to me.
   - If I give out my proof, anyone can do further research
   - I should be the one to get papers out of this: after all, it's my proof

5. The proof is valuable intellectual property.
   - I would be stupid to give it away:
     I might be able to commercialize it some time in the future

Top ten list of why not to share code

6. Including proofs would make the paper much longer.
   - Journals wouldn't want to publish it, and who would want to read it?

7. Referees would never agree to check proofs.
   - It's already difficult to get reviewers and reviews on time

Top ten list of why not to share code

8. The proof uses sophisticated hardware/software that most readers and referees don't have.
   - If they can't execute the proof, why should they care to get it?

9. My proof relies on other unpublished (proprietary) proofs.
   - It doesn't really help that they have my proof, they don't know if the dependencies are correct anyway

Top ten list of why not to share code

10. Readers who have access to my proof will want user support.
    - And I really don't want to be pestered by people actually using my work

- Are computer codes fundamentally different from mathematical proofs?

Why you should share your code anyway

"An article about computational result is advertising, not scholarship. The actual scholarship is the full software environment, code and data, that produced the result."
--Jon Claerbout [1]

Personally I think people get hung up too much on the fact that it's hard to insure others can run the code, and should focus more on providing a full record of the research methodology.
-- Randy Leveque [2]

[1] WaveLab and Reproducible research, J. B. Buckheit and D. L. Donoho, 1995
[2] Top Ten Reasons to Not Share Your Code, Randy Leveque, 2012

Best Practices

Best Practices Countdown [1]

1. **Write programs for people, not computers**
   - If a code is easy to read, it is easier to check if it is doing what it should
   - Human memory is extremely limited: "a program should not require its readers to hold more than a handful of facts in memory at once"

Bad:
```
def rect_area(x1, y1, x2, y2):
```

Good:
```
def rect_area(point1, point2):
```

   - Human effort is limited: "all aspects of software development should be broken down into tasks roughly an hour long"

[1] Best Practices for Scientific Computing, Greg Wilson et al., 2012, arXiv:1210.0530

Best Practices Countdown

2. Automate repetitive tasks

- even the most careful researcher will lose focus while doing this and make mistakes.
- "use a build tool to automate the scientific workflows"

Best Practices Countdown

3. Use the computer to record history

- Data and source code provenance should automatically be stored
  "history" in Matlab or the Linux command-line, "doskey /history" on windows command line, IPython
- Automatically record versions of software and data, and parameters used to produce results (see also point 2)

4. Make incremental changes

- Do not plan for months or years of development: Plan for one week or so, partitioned into small tasks (which can be solved using one hour long sessions at a time)

Best Practices Countdown

5. Use version control

- Learn how to see the difference (diff) between two versions of the software, and how to revert changes
- Learn how to use version control for collaboration
- "everything that has been created manually should be put in version control"
- Use meta-data to describe binary data

6. Don't repeat yourself

- "every piece of data must have a single authoritative representation in the system"
- "code should be modularized rather than copied and pasted"
- "re-use code instead of rewriting it" (matrix inversion, etc.)

Best Practices Countdown

7. Plan for mistakes (1/2)

- "add assertions to programs to check their operation"

```
def bradford_transfer(grid, point, smoothing):
        assert grid.contains(point),
                'Point is not located in grid'
        assert grid.is_local_maximum(point),
                'Point is not a local maximum in grid'
        assert len(smoothing) > FILTER_LENGTH,
                'Not enough smoothing parameters'
        ...do calculations...
        assert 0.0 < result <= 1.0,
                'Bradford transfer value out of legal range'
        return result
```

- Assertions are "executable documentation, i.e., they explain the program as well as checking its behaviour"

Best Practices Countdown

7. Plan for mistakes (2/2)

- Use automated testing
  - Regression testing => has something changed
  - Verification testing => does the code produce known correct/analytical solutions?
  - Use an interactive debugger instead of print-statements

8. Optimize software only after it works correctly

- When it works, use a profiler to find out what the bottleneck is
- Software developers write the same amount of code independently of the language: "write code in the highest-level language possible"

Best Practices Countdown

9. Document design and purpose, not mechanics.

- Code should be written for humans and not require any documentation by itself: "document interfaces and reasons, not implementations"
- Remove unreadable code: "refactor code instead of explaining how it works"

10. Collaborate.

- Make others read your code: "code reviews are the most cost-effective way of finding bugs in code"
- "use pair programming when bringing someone new up to speed and when tackling particularly tricky problems"
- Collaborators can interpret results in completely different ways

"**Research suggests** that the **time cost** of
implementing these kinds of tools and approaches
in scientific computing **is almost immediately**
offset by the gains in productivity of
the programmers involved"

Best practices overload

11. Take notes

- Use an issue tracker, blog, wiki or physical lab notebook for notes and ideas
- Ideas come up all the time, and are written on post-its etc. and easily forgotten.

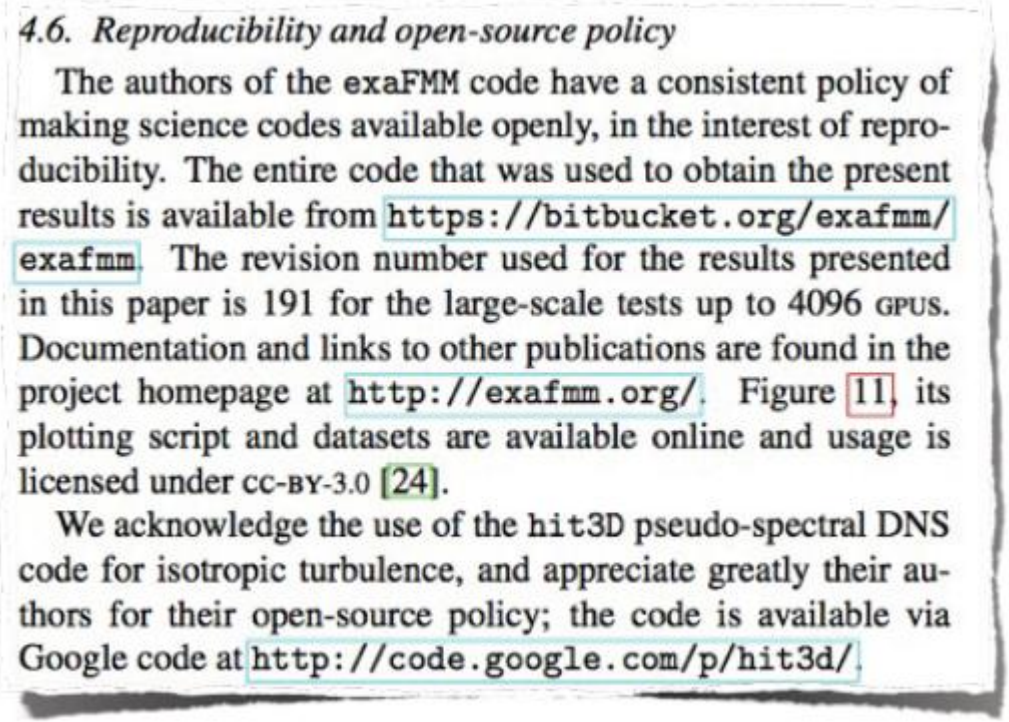
12. Keep it simple, stupid

- Design your code and work flow so "anyone" can repair it using standard tools
- If it's extremely complicated,
  does it really have to be?
- Simplicity in design is a virtue

Best practices overflow

13. **Write statements on reproducibility** [1]

- At the end of papers you publish, write if and how the results are reproducible. Especially if you are unable to publish code: write why!



4.6. Reproducibility and open-source policy

The authors of the exaFMM code have a consistent policy of making science codes available openly, in the interest of reproducibility. The entire code that was used to obtain the present results is available from `https://bitbucket.org/exafmm/exafmm`. The revision number used for the results presented in this paper is 191 for the large-scale tests up to 4096 GPUs. Documentation and links to other publications are found in the project homepage at `http://exafmm.org/`. Figure 11, its plotting script and datasets are available online and usage is licensed under CC-BY-3.0 [24].

We acknowledge the use of the `hit3D` pseudo-spectral DNS code for isotropic turbulence, and appreciate greatly their authors for their open-source policy; the code is available via Google code at `http://code.google.com/p/hit3d/`.

[1] Reproducibility PI Manifesto, Lorena A. Barba, http://faculty.washington.edu/rjl/icerm2012/Lightning/Barba_Manifesto.pdf

Limits of reproducible research

Definitions of reproducible research

[1]Reproducibility in Computational and Experimental Mathematics, Workshop report, 2012

Reviewed Research:

The descriptions of the research methods have been independently assessed and the **results judged credible.** (This includes both traditional peer review and community review, and **does not necessarily imply reproducibility.**)

Replicable Research:

Tools are made available that would allow one to **duplicate the results of the research, for example by running the authors' code** to produce the plots shown in the publication. (Here tools might be limited in scope, e.g., only essential data or executables, and might only be made available to referees or only upon request.)

Confirmable Research:

The **main conclusions** of the research **can be obtained independently** without the use of software provided by the author. (But using the complete description of algorithms and methodology provided in the publication and any supplementary materials.)

Auditable Research:

Sufficient records (including data and software) have been archived so that the **research can be defended later if necessary**. The archive might be private, as with traditional laboratory notebooks.

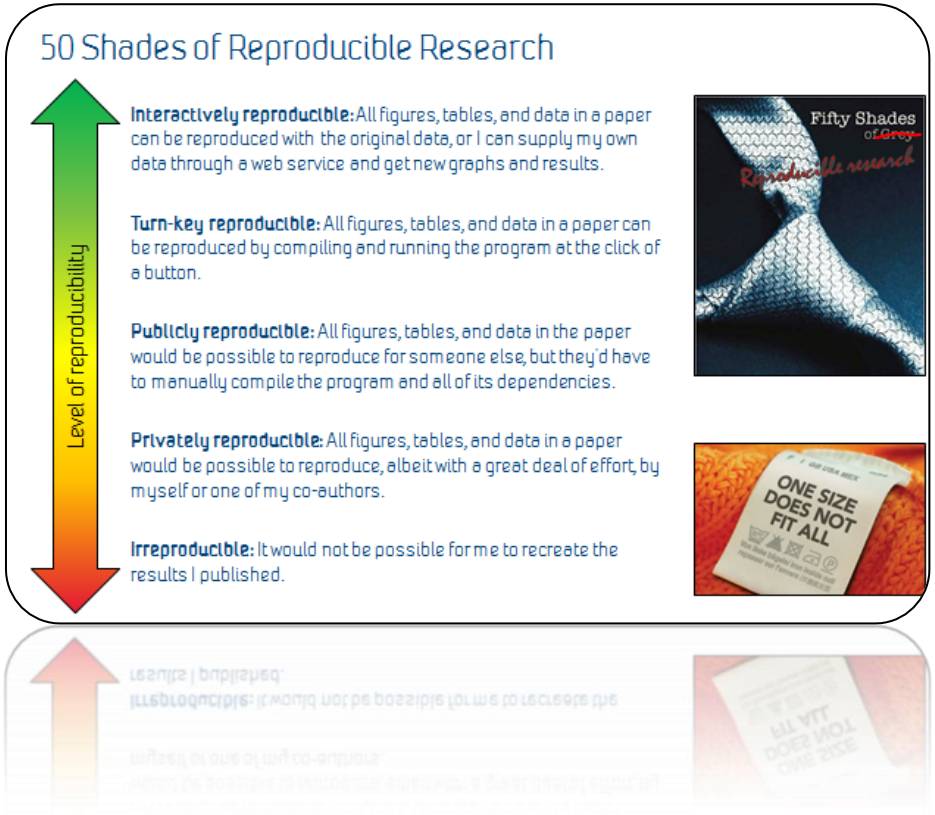
Open Research: Well-documented and fully open tools are publicly available (e.g., all data and open source software) that would allow one to (a) fully audit the computational procedure, (b) duplicate the results of the research, and (c) extend the results or apply the method to new problems.

Saint Graal de la légende du Roi Arthur et des Chevaliers de la Table Ronde, Alfred W Pollard, 1917

The limits of reproducible research

- The limits of reproducible research depends on the type of reproducible research we are discussing

- What hinders
  - "Private reproducibility"?
  - "Public reproducibility"?
  - "Turn-key reproducibility"?
  - "Interactive reproducibility"?



Problems in reproducibility

Supercomputer simulations

- Requires special hardware
- Rerunning experiments not always feasible
- Changing number of nodes changes the domain decomposition, which affects the answer…
- Parallel computing is terribly irreproducible
- High performance computing often comes at the expense of reproducibility

Argonne National Laboratory, IBM Blue Gene P, CC-BY-SA 2.0

Problems in reproducibility

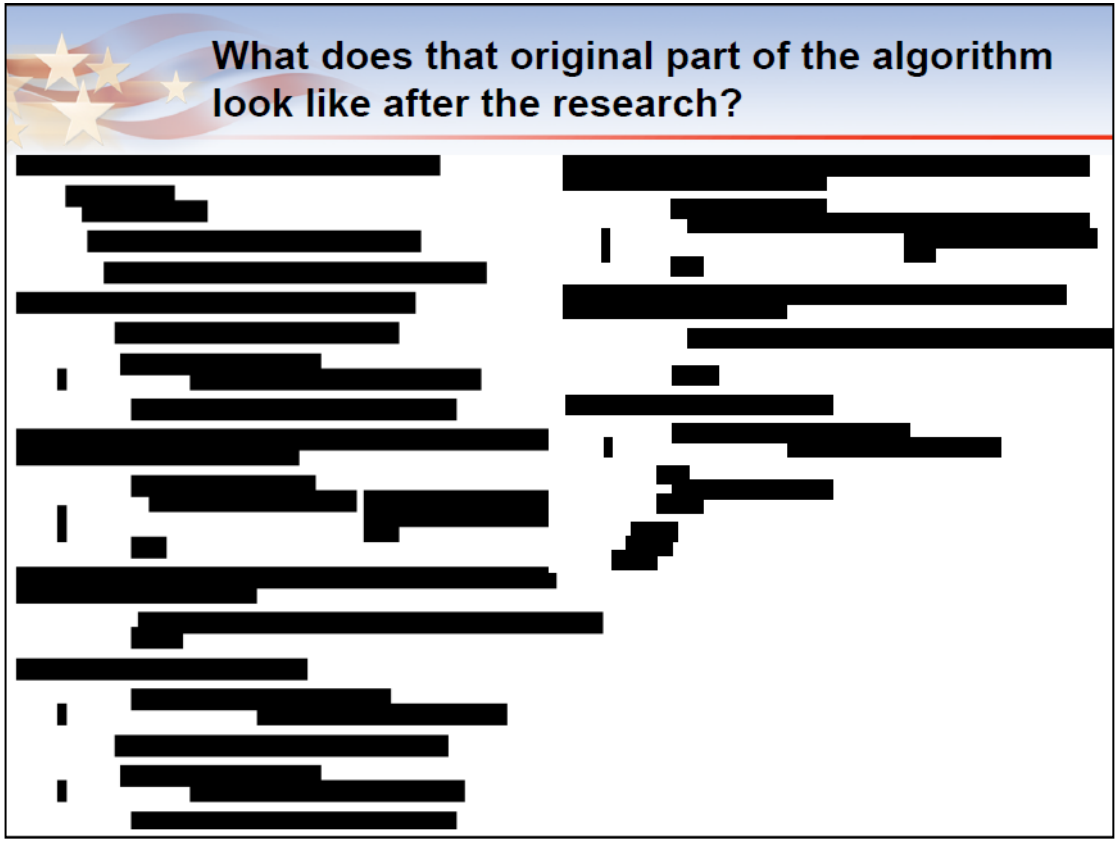
Graphics Processing Units

- GPUs are extremely parallel processors with all the pitfalls of parallel computing
- GPUs change rapidly, and I can't get a five year old GPU anymore
- Programming languages and tools change extremely fast
- A different floating point model than many CPUs (more accurate)

Problems in reproducibility

Legal concerns

- Patent laws
- Export limitations
- Intellectual property rights
- Licenses
- Research performed at commercial institutions
- …

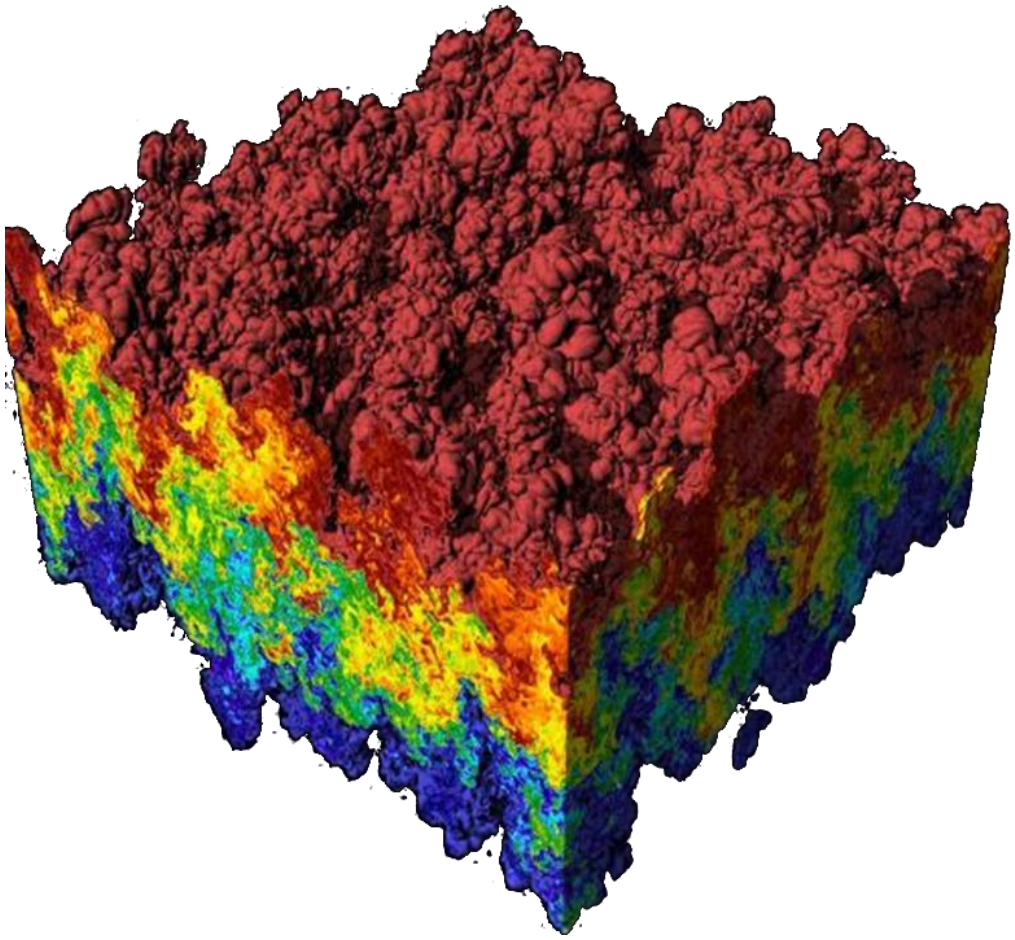


Censored slide from Bill Rider, "What does it take to do reproducible computational science? What stands in our way?", ICERM 2012.

Problems in reproducibility

Visualization

- Many visualization tools are used interactively, and therefore hard to use reproducibly



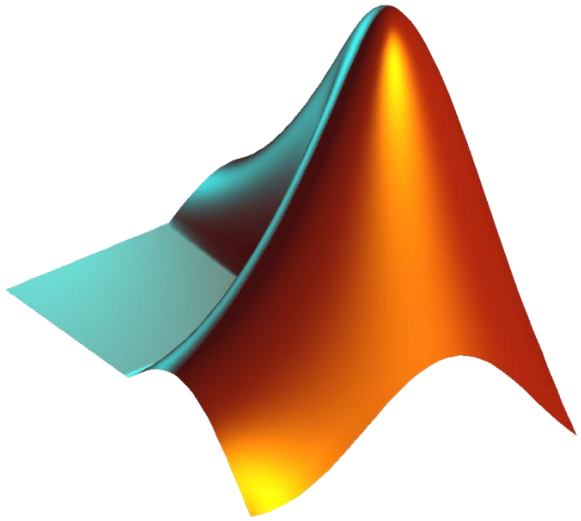
Problems in reproducibility

Data archiving

- We have version control for text-like documents
- We have very little for managing data sets
- Data sets must have meta-data which is manually entered

Archives, Archivo-FSP, CC-BY-SA 3.0

Problems in reproducibility



Software licenses

- Difficult to combine with virtual machines & the cloud

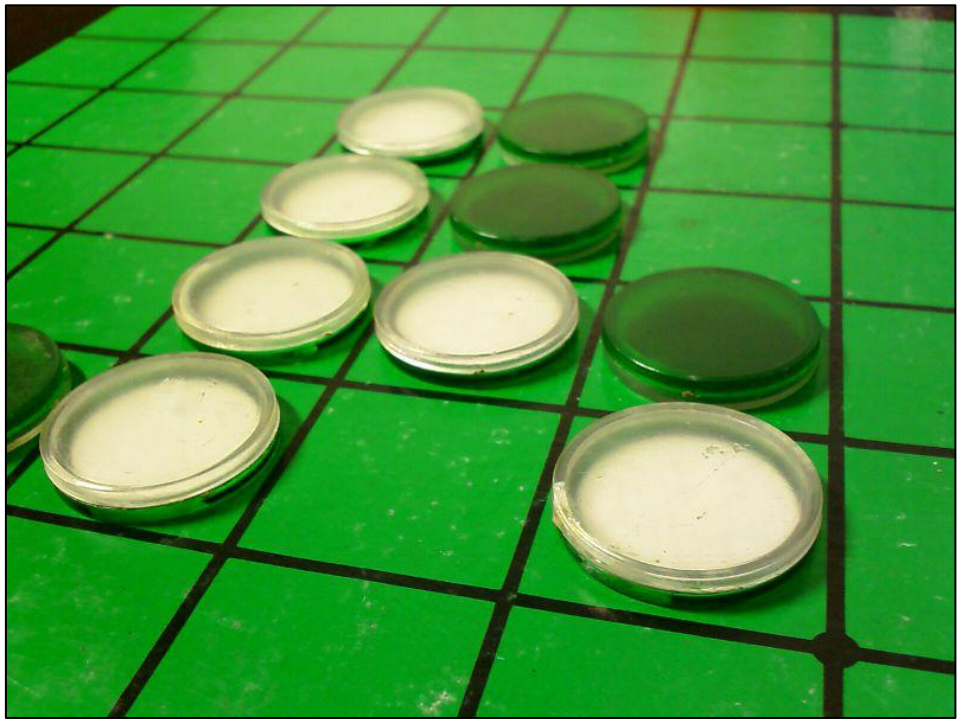
Old software

- My software only run under AIX / Windows NT / … and I can't get hold of hardware that it will install under…
- My software relies on a specific behavior only found in GCC v. 2.81
- My software requires a commercial compiler which only runs on Windows NT / AIX / …

Problems in reproducibility

Floating point

- Floating point is like chess:
  it takes minutes to learn, and
  a lifetime to master
  (or, at least it's quite complex
  for such a simple definition)



A game of Othello, Paul 012, CC-BY-SA 3.0

Summary

- Computational codes are a lot like mathematical proofs and we should aim at publishing whenever possible
- The essence of the best practices is: "Be methodical, be thorough, be honest".
- Different situations have different requirements to disclosure and reproducibility
- It can be difficult to be reproducible in some situations.

Further Reading

- Randy Leveque, Top Ten Reasons to Not Share Your Code (and why you should anyway), Randy Leveque, 2012, http://faculty.washington.edu/rjl/pubs/topten/

- Reproducibility in Computational and Experimental Mathematics, Workshop report, 2012 [to appear]

- 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 (Submitted on 1 Oct 2012 (v1), last revised 29 Nov 2012 (this version, v3)) http://arxiv.org/abs/1210.0530