Reproducibility of mathematical data: a cheatsheet to take away

(Alex Elzenaar, 2/8/22)

We have colour coded some things: in **bold red** are lessons you can learn for your own papers to make things easier for readers. In *slanted fuchsia* are some questions to think about. On the separate sheet you will find in emerald a practical example to work on as you read this sheet. Everything is also available online, at `https://aelzenaar.github.io/tchng/repro.html`

1 Work out what results you want to reproduce, and find them.

We need some results to reproduce. Usually of course we are coming from a paper, so hopefully the authors included a clear link to a place where we can find their software and assorted pieces of working for their published results, in a place that still exists; then we need to hope that the software is actually in a form that we can run. (Sometimes this will involve trawling through the Wayback Machine.)

Lesson: (1) Always put your software somewhere that has some kind of backing behind it, won't vanish without warning, and is easy to find via search engines (e.g. `GitHub`). (2) Make sure your paper links to the place you put your data, and vice versa.

*What kinds of bibliographic data should you include in your paper? Is it enough to just put a URL? What are some **bad** ways to include mathematical code on your website?*

Remark. Once you know what you want to reproduce, there are actually two things to do:

1. verify that the method and tools (e.g. computer programmes) which were used by the author(s) do indeed produce the desired result(s);
2. verify that the process by which the results were produced are in fact correct (e.g. that the computer tools were correct).

In general (2) is much harder: for computers, this involves checking the entire stack of software and hardware[1] used as well as the theoretical basis. (In practice, one does this by redoing everything from scratch completely independently on an entirely different computer setup, without looking at the original software.) We restrict ourselves to (1).

2 Get the tools that the authors used.

There are two kinds of tools: off-the-shelf tools, and bespoke tools. We need to get hold of any software platforms and packages used. Different hardware and software combinations might produce different results with the same input, so we should make sure that we are using the same version of the software[2], if possible.

Lesson: Document everything, including software versions, the computer you used (especially if you wrote your own low-level code that depends explicitly on hardware features), and make sure it is always kept next to your software: best practice is in a file that is *inside* the compressed file you are putting up for download on your website/other repository.

[1] Even the basic arithmetic hardware found in your computer might not be trustworthy: `https://en.wikipedia.org/wiki/Pentium_FDIV_bug`

[2] To be more rigorous we would need to also match the hardware architecture, but practically speaking this is probably not an issue except in niche cases.

3 Modify the tools so that they are in a form which we can actually use.

Maybe it wasn't possible to get the original tools that were used by the authors, for instance because they wrote their software in a programming language that we can't compile easily, or an old version of the software platform which we don't have access to. Then we need to be more clever, and the problem becomes a software engineering problem. Document any changes you need to make to the original software, especially if you plan to publish or teach something which depends on the paper in question.

Lesson: try to pick software platforms with a large community around them that have been around for a while (years). This is not a guarantee of continued accessibility but it gives future readers a fighting chance. Of course this needs to be balanced against having specialty features.

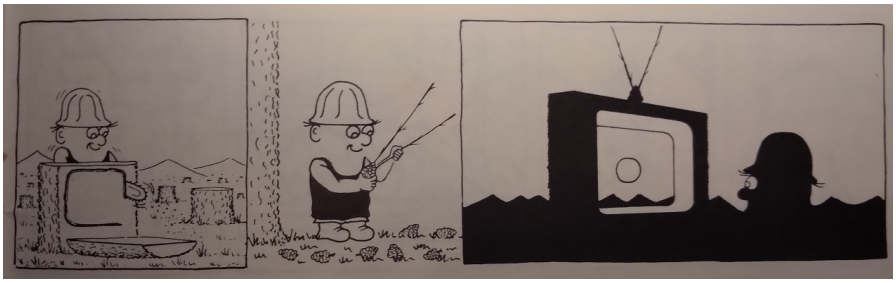
Here are some languages which various popular books use. Which were good choices and which weren't, taking into account the tradeoffs we mentioned and anything else you can think of?

1. `Macaulay2` *[Eisenbud, Commutative Algebra]*
2. `MATLAB` *[Waldron, An Introduction to Finite Tight Frames]*
3. *Fortran and BASIC (unspecified versions) [Strang, Linear Algebra and its Applications]*
4. `MIX` *[Knuth, Fundamental Algorithms (TAoCP I)] (←this one might be a trick question)*
5. `APL` *[Braun, Differential Equations and Their Applications]*
6. `FORTRAN`*-like pseudocode [Mumford, Series, and Wright, Indra's Pearls]*

4 Actually reproducing the results.

Now comes the 'easy' part: just run the code and make sure that it produces the same result. In practice there are a few issues:

- The software might produce a mound of data, which you can't check manually against the original. This is now a software engineering problem, probably.
- The software might be producing data with some amount of randomness (for instance, it might do a Monte Carlo integration). You might need to run a statistical analysis.
- The software might need additional input which isn't documented. **Lesson: It is not enough to publish your computer programme, you also need all the information that's needed to run it—including any additional parameters that you used.**
- *What else might go wrong?*



To try to reproduce Bogor's experience in the third panel, you need three things: (1) the same off-the-shelf or foundational platform (the wood and the saw); (2) the thing that is produced with that platform specifically for the project (the TV); and (3) the right *environment* (i.e. the landscape and skyscape). If any of these were missing, the picture would probably look similar in some ways but would not be a reproduction. [Picture: Burton Silver, *Bogor 1977–78*, p. 27. Whitcoulls Ltd, Christchurch (1977).]

5 What if you can't reproduce the results?

1. **Document everything you did.** *What kinds of things, explicitly, do you need to write down?*
2. Are the results you are getting actually theoretically possible, or are they total rubbish?
3. Try to narrow down the problem to a particular place in the stack:

theory ⟶ hardware ⟶ off-the-shelf software platform ⟶ author software ⟶ changes you made

4. Can you ask the original authors for help?
5. If you are sure that the results are wrong, what to do next depends on many factors: Are the original authors still around? Was the original data published in a journal or other venue backed by an editorial board or institute who you can contact? Can you correct the result or is it irrecoverable? Etc.