Why computational reproducibility matters

Published June 20, 2025

Citation
Hinsen, K. (2025, June 20). Why computational reproducibility matters. *Konrad Hinsen's Blog*. https://doi.org/10.59350/vpfzy-wrr35

Thirty years after my first contact with computational (ir)reproducibility, I am happy to note that many things have improved. Reproducibility, computational and otherwise, is increasingly recognized as an important aspect of scientific quality control, and mostly considered worth striving for. However, I also note that more and more people, including reproducibility activists, have lost contact with the day-to-day reality in which reproducibility matters. Reproducibility is becoming an item on a checklist, and its precise incarnation the subject of political bickering aimed at making it easy to check off that item. So let's take a look at *why* computational reproducibility matters for researchers.

If you have read papers from the field of cryptography, you probably know Alice and Bob. They used to invest a lot of effort into communicating privately, making sure that nobody else could listen to them. But then, Alice and Bob discovered Open Science, and now they talk to each other in public spaces, in addition to publishing all their data and code of course. And now we know that they are researchers in computational biophysics!

Let's listen to Alice and Bob as they meet at a conference.

Alice: I have computed the equilibrium distance between the ligand and the active site of our pet protein. **It's 0.9 nm.**

Bob: I have computed the same distance, but **I find 1.1 nm.**

For any Open Science practitioner, the next step should be obvious.

Alice: Uhhh... Well... I will look at your code, and you look at mine. Let's meet again tomorrow.

Bob: OK!

The next day, they meet again.

Alice: I couldn't compile your code. Look at this error message!

Bob: It works for me! You use Debian 12? I still run Debian 9. That's surely what makes the difference. But I also have good news: I managed to run your code on my machine. The only problem is that... I get **0.8 nm.**

Alice: I use `libode` version 3.4. The documentation says it must be compiled with `gcc 10` or later. You probably have an older `gcc`.

Bob: Uhhh... Well... I will have to install a virtual machine with Debian 12, and you with Debian 9. Shall we meet again in a week?

Alice: OK!

A week later, no solution is in sight.

Alice: Under Debian 9, I managed to run your code. I get 1.1 nm, like you do. But I don't understand why! **Your code is unreadable.**

Bob: Under Debian 12, **your code yields 0.85 nm for me.** That's not your value of 0.9 nm. Nor 1.1 nm as I get using my method. I don't understand why!

That's what real life looks like. Whatever reproducibility policies you advocate, they aren't worth much unless they open a way for Alice and Bob to figure out why they get different numbers. Reproducibility needs to support effective debugging.

In terms of the increasingly consensual terminology defined by the US National Academies of Science, Engineering, and Medicine, Alice and Bob have two issues to resolve:

1. Bob cannot *reproduce* Alice's value of 0.9 nm. He finds 0.85 nm instead, in spite of running the same code in what he believes to be the same computational environment: Debian 12.
2. Neither Alice nor Bob can *replicate* the other's finding using their own code. Alice finds 0.9 nm, Bob finds 1.1 nm.

Now let's consider two popular recipes for reproducibility:

1. *Just use Docker.* Both Alice and Bob should package their code plus environment as a container image. They could then exchange their images, and each check that they retrieve the other's value. Problem 1 is solved. However, the investigation of problem 2 becomes nearly impossible. The code inside the container images is a black box. Exploring point 2 would require reading, recompiling, and modifying the code. That's not possible if all you have is a container image.

2. *Just use conda.* For some reason, many people seem to believe that conda has some magical capability to solve reproducibility issues - see this report for example. In reality, it's not fundamentally different from Debian's package manager (and many others), and in practice it is worse because conda users tend to focus on fast-moving bleeding-edge code whereas Debian package maintainers place a high value on stability. If Alice and Bob used conda, they would probably both fail at even compiling the other's code in their own environments, and also fail at trying to reproduce the other's environment.

Neither simple recipe is really helpful. Both containerization and package managers are useful tools in the quest for reproducibility, but there is no "just use..." that really works. A big reason is that neither container nor packaging tools were developed with reproducibility in mind. They are by design *deployment* technologies, meant to facilitate the task of installing, updating, and running software on a computer. Which is an important task of course, but it's not reproducibility.

Let's tackle the question from the other end: what would Alice and Bob need for debugging?

To investigate the reproducibility issue, they need to understand why the same code, compiled and run on two different installations of Debian 12, yields different results. Alternatively, to eliminate the reproducibility issue, they would need a more precise way to describe and reconstruct an environment than the label "Debian 12". And to understand their replicability issue, they need their own code plus as much as possible of its dependencies to be

understandable, and easy to modify and run, because exploring replicability invariably leads to tinkering with each other's code.

The fundamental reason for the reproducibility issue is that "Debian 12" is not a precise specification for a computational environment. Packages get constantly updated in between two Debian releases. To make it worse, the order in which packages are installed can also make a difference. If you want to understand why, follow the MOOC, in particular its second module, "Managing software".

If Alice had provided a precisely specified computational environment, rather than just saying "Debian 12", then the reproducibility issue in the above story would simply disappear. Bob would have run Alice's environment, confirmed the result of 0.9 nm, and then they would have moved on to their replicability issue, which is scientifically more relevant. Reproducibility issues are nothing more than a nuisance. They are about results being different due to differences in computational environments that most computer users have no control over, and don't really care about until they have to.

So how do you provide a precisely specified computational environment? One possible answer is a container image. But as I explained above, if all you have is a container image, you can't explore replicability issues any more. What you want is a precisely specified *and reproducible* environment. Meaning that you *can* run it as-is, and get *exactly* the same results, bit for bit, but you can *also* change what's inside and see what difference that makes.

My experience from many talks, courses, and informal discussions I have had over the last years on the topic of reproducibility is that as soon as I say "bit for bit", a resistance forms in parts of the audience. Just over the last month, I have been called a "reproducibility extremist" twice. The counter-argument I hear is always the same: we don't really *need* bit-for-bit identical results. The tacit assumption is that going for a less strict goal, somewhere in between "Debian 12" and a precise specification, would be both easier and sufficient. And that's where I disagree. Something less rigorous may or may not be sufficient in a specific situation. But it is definitely not *easier*.

In fact, I have no idea how Alice could come up with a specification for her computational environment that would promise Bob a result between, say, 0.88 nm and 0.92 nm. I *do* know, however, how Alice can provide a bit-for-bit reproducible specification. It's actually very easy in theory. Computers are deterministic machines. If you give them the same inputs, they produce the same outputs. Bit-for-bit reproducibility is a matter of bookkeeping: keeping track precisely of all the steps that were performed in constructing a computational environment. And bookkeeping is something that computers are quite good at. In contrast, good-enough-for-me reproducibility can only be evaluated through case-by-case domain expertise, and cannot be designed at all.

Unfortunately, what is easy in theory turns out to be difficult in practice. The computational infrastructure that we all use, i.e. our operating systems, package managers, compilers and other software build tools, containerization tools, etc., was not designed with reproducibility in mind. Outside of science, approximately nobody cares about reproducibility. The one exception

I know of is cybersecurity experts, who require reproducibility (bit-for-bit again) as a guarantee that a compiled program was really derived from the source code that it pretends to be generated from, without the secret addition of malware. Most computer users need no more in terms of software management than installing and updating programs. That's what today's infrastructure makes easy.

Nevertheless, it is possible to make a computational environment bit-for-bit reproducible, at least when run on identical hardware. If you want to learn how to do it, follow the MOOC. We show how to do it using Debian snapshots, staying in the context of a well-known and widely used Linux distribution. We also show how to do it using Guix, a next-generation package manager (and Linux distribution) that is one of the very few tools that *was* designed with reproducibility in mind. Neither path is as simple as I would like it to be, because they both involve tools that lack user-friendly interfaces for now. It's bit like using `git` for version control: it does the job, but it can be a pain to use. More work is clearly required. But it will only happen if larger parts of the scientific community agree that it is worth doing, and direct funding towards this task.

My conclusion is that bit-for-bit reproducibility is something that we can solve once and for all and push into the infrastructure, such that Alice and Bob needn't worry about it any more. If instead we go for good-enough-for-me reproducibility, it will remain a nuisance for generations of scientsts to come. If that is an extremist opinion, so be it.