How to enable inexpensive reproducibility for computational experiments

Samuel Grayson

2024 Oct 03

Problem

Reproducibility of scientific experiments is important for three reasons:

1. The scientific community corrects false claims by applying scrutiny to each others' experiments. Scrutinizing an experiment often includes reproducing it. Therefore, reproducible experiments may be thoroughly scrutinized.
2. Science works by building off of the work of others. Often in extending anothers work, one needs to execute a modified version of their experiment. If reproducing the same experiment is difficult, one would expect executing an extended version to be even more so. Therefore, reproducible experiments may be easier to extend.
3. The end-goal for some science is to be applied in engineering applications. Applying a novel technique on new data involves reproducing a part of the experiment which established the novel technique. Therefore, reproducible experiments may be easier to apply in practice.

Reproducibility also has costs, primarily in human labor needed to explain the experiment beyond that which would be needed to merely disseminate the results. Working scientists balance the cost of reproducibility with the benefits to society or to themselves.

In real-world experiments, there are an infinitude of possible factors that must be controlled to find the desired result; it would be unfathomable that two instantiations of an experiment could give exactly identical results. In contrast, for computational science experiments (from here on, **CS Exp**) on digital computers, while there are still many factors to be controlled, perfect reproduction is quite fathomable. Despite this apparent advantage, CS Exps on digital computers still suffer low rates of reproducibility.

The status quo will not change simply by arguing for reproducibility; those arguments are widely known are already taken into account by the efficient market. Nor do I have the power to change incentives in science funding policy. However, by reducing the cost of reproducibility, scientists may produce more reproducible experiments with the same effort (fig. 1). Reducing the cost is a technical problem this work attempts to solve.

For the purposes of this work, "computational scientist" (from here on, **CS**) should be construed broadly, as anyone who uses research software to carry out some investigation. The term includes people who use research software for data analytics and for simulations, so long as scrutinizability, extensibility, and applicability are motivators. The term applies to professors in academia as well as analysts in a national lab.

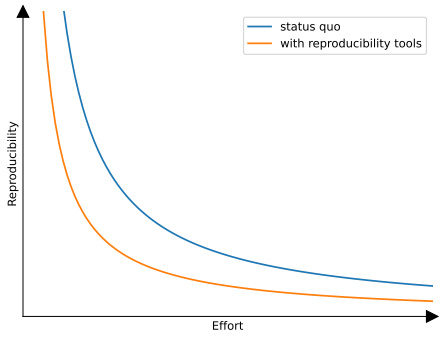

Figure 1: Effort-vs-reproducibility tradeoff curve in the status quo and with reproducibility tools

Background

For the purposes of this work, we use the ACM definition of reproducibility:

> **Reproducibility**: (different team, same experimental setup) The measurement can be obtained with stated precision by a different team using the same measurement procedure, the same measuring system, under the same operating conditions, in the same or a different location on multiple trials.
>
> — (ACM Inc. staff 2020)

This is substantially similar to the National Academy of Sciences:

> **Reproducibility**: obtaining consistent computational results using the same input data, computational steps, methods, code, and conditions of analysis.
>
> —TODO: cite

Naïvely, every CSE would be "reproducible", since there is some, possibly unknown, set of conditions under which another team can make the measurement; i.e., "make your environment bit-wise the same as ours". To prevent the definition from being vacuous, we will only consider *explicitly stated* operating conditions.

This is opposed to **repeatability** (same team, same experimental setup) and **replicability** (different team, different experimental setup).

These definitions derive from *metrology*, the science of measuring, and we will specialize some of the terms "measurement", "measurement procedure", and "operating conditions" in software context for computational science experiments.

- **Measurement procedure (for CSEs)**: The application code and user-interactions required to run the application code (if any).
- **Operating conditions (for CSEs)**: A set of conditions the computer system must implement in order to run a certain program. E.g., GCC 12 at `/usr/bin/gcc` compiled with certain features enabled.

One may over-specify operating conditions without changing the reproducibility status. E.g., one might say their software requires GCC 12, but really GCC 12 or 13 would work. It is quite difficult and often not necessary to know the necessary-and-sufficient set of operating conditions, so in practice, we usually have set of conditions that is sufficient but not necessary to operate the experiment.

Operating conditions can be eliminated by moving them to the measurement procedure. E.g., the program itself contains a copy of GCC 12. For the purposes of this work, the operating conditions are the "manual" steps that the user has to take in order to use the measurement procedure to make the measurement.

- **Measurement (for CSEs)**: Rather than offer a definition, we give some examples:
  - **Crash-freedom**: program produces result without crashing.
  - **Bitwise equivalence**: output files and streams are bit-wise identical.
  - **Statistical equivalence**: overlapping confidence intervals, etc.
  - **Inferential equivalence**: whether the inference is supported by the output.
  - **Others**: domain-specific measurements/equivalences (e.g., XML-equivalence ignores order of attribtues)

In general, it is difficult to find a measurement that is both easy to assess and scientifically meaningful.

Measurement	Easy to assess	Scientifically meaningful
Crash-freedom	Yes; does it crash?	Too lenient; could be no-crash but completely opposite result
Bitwise equivalence	Yes	Too strict; could be off by one decimal point
Statistical equivalence	Maybe; need to know output format	Maybe; need to know which statistics *can* be off
Inferential equivalence	No; need domain experts to argue about it	Yes

Composition of measurement procedures: The outcome of one measurement may be the input to another measurement procedure. This can happen in CSEs as well as in physical experiments. In physical experiments, one may use a device to calibrate (measure) some other device, and use that other device to measure some scientific phenomenon. Likewise, In CSE, the output of compilation may be used as the input to another CSE. One can measure a number of relevant properties of the result of a software compilation.

Compilation measurement	Definition
Source equivalence	Compilation used exactly the same set of source code as input
Behavioral equivalence	The resulting binary has the same behavior as some other one
Bit-wise equivalence	As before, the binary is exactly the same as some other one

E.g., suppose one runs `gcc main.c` on two systems and one system uses a different version of `unistd.h`, which is `#included` by `main.c`. The process (running `gcc main.c`) does not reproduce source-equivalent binaries, but it might reproduce behavior-equivalent binaries or bit-wise equivalent binaries (depending on how different `unistd.h`).

Related work

There are many related works that address reproducibility. We will leverage the definitions and framework discussed above to contextualize each work. Still, there is a significant gap in prior work that this work will exploit. Prior work can be divided into three categories, which we will investigate in turn:

1. Studying empirical characteristics of reproducibility
2. Studying approaches associated with proactively ensuring reproducibility
3. Studying approaches associated with reactively restoring reproducibility

Metastudies

Reproducibility, Replicability, and Repeatability: A survey of reproducible research with a focus on high performance computing by Antunes and Hill https://arxiv.org/abs/2402.07530

Empirical characterization of reproducibility

This group of related work seeks to characterize the degree of reproducibility or a proxy for reproducibility in a sample of CSEs empirically. A proxy variable could be "whether the source is available", since this is a necessary but not sufficient condition for reproducibility.

Publication	N	Subjects	Population	Repro measurement assessed or proxy variable	Level
Vandewalle, Kovacevic, and Vetterli (2009)	134	Article artifacts	2004 ed. of img. proc. journal	Code availability	9%
Zhao et al. (2012)	92	Taverna work-flows	myExp. 2007 – 2012	Crash-free execution	29%
Collberg and Proebsting (2016)	508[1]	Source code	2012 eds. of CS journals	Crash-free compilation	48%
(**gundersen?**)	400	Article artifacts	2013 – 2016 AI journals	Sufficient description	24%[2]
Pimentel et al. (2019)	863,878[3]	Jupyter note-books	GitHub	Same-stdout execution	4%
Krafczyk et al. (2021)	5	Articles artifacts	Journal of Comp. Phys.	Figures and tables semantic equivalence	70%[4]
Wang et al. (2021)	3,740[5]	Jupyter note-books	GitHub	Crash-free execution	19%
Trisovic et al. (2022)	2,109[6]	R scripts	Harvard Dataverse	Crash-free execution	12%

[1] I am considering the total sample to be only those papers whose results were backed by code, did not require special hardware, and were not excluded due to overlapping author lists, since those are the only ones Collberg and Proebsting attempted to reproduce. I am considering OK<30 and OK>30 as "reproducible crash-free building" because codes labelled OK>Author were not actually reproduced on a new system; it was merely *repeated* on the authors system (Collberg and Proebsting 2016).

[2] This figure is the average normalized "reproducibility score", based on whether the method, data, and experiment, were available. If it were 1, all the papers in the sample would be have method, data, and experiment availability, and if it were 0, none would be.

[3] I am considering the total sample to be only notebooks that were valid, pure Python, and had an unambiguous order, since those are the ones Pimentel et al. attempt to reproduce (Pimentel et al. 2019.)

[4] This figure is the average number of computational elements (figures and tables) that were reproduced to semantic equivalence in each paper.

[5] I am considering the total sample to eb only notebooks that had dependency information and used Python 3.5 or later, since those are the only ones Wang et al. attempted to reproduce (Wang et al. 2021).

[6] I am considering the total samples to be the set of *un-repaired* codes, since those are the ones that actually exist publicly. Also, we *include* codes for which the time limit was exceeded; reproduction was attempted and not successful in those conditions.

Note that crash-freedom is prevalent because it is the easiest to automatically assess. Source-availability requires human-intervention to test, so studies on crash-freedom simply begin from a corpus for which source code is available. Note that stdout-equivalence is feasible because Jupyter Notebooks bundle the stdout with the code; otherwise, the stdout is usually thrown away.

Overall, these studies show that reproducibility is a significant problem.

What are common reasons software is not reproducible?

We should address one of those.

Proactively ensuring reproducibility

There are several proposed approach associated with proactively ensuring reproducibility:

Approach	Aspect of reproducibility addressed
Scientific clouds	Reduces non-portable operating conditions
Digital artifact archival	Ensure attainability of operating condition
Workflow managers	Simplify measurement procedure
Provenance	Identifies operating conditions
Record/replay executions	Reduces operating conditions
Version control	Versions and distributes measurement procedure
Virtualization/containerization	Reduces non-portable operating conditions
Package managers	Reduces non-portable operating conditions
Literate programming	Explicates measurements
Seeding inputs	Reduces non-deterministic operating conditions
Coding practices	Reduces non-deterministic operating conditions

Scientific clouds

Scientific clouds aim to address reproducibility by replacing a complex set of operating conditions ("install this, configure that") with a single operating condition: log in to a cloud system. The cloud system hosts a controlled environment with the rest of the operating conditions already met. However, storing the environments and running executions in them is expensive. Either the users have to pay, in which case the computational environment is not "freely accessible", or some institution may sponsor public-access. Previous scientific clouds have a mixed record, with many going defunct in just a few years, failing in their promise of long-term reproducibility.

Scientific Cloud	URL	Lifetime on Archive.org
WholeTale	WholeTale.org	2016 – present
GridSpace2	gs2.plgrid.pl	2015 – 2020
Collage Authoring Environment	collage.elsevier.com	2013 – 2014
RunMyCode	RunMyCode.org	2012 – 2024
SHARE	sites.google.com/site/executablepaper	2011 – 2023
GenePattern	GenePattern.org	2006 – present

TODO: Place the following:

- Chameleon Cloud
- Google Colab
- GalaxyHub.eu
- Gentleman & Temple Lang's Research Compendium
- G. R. Brammer, R. W. Crosby, S. Matthews, and T. L. Williams. Paper mch: Creating dynamic reproducible science. Procedia CS, 4:658–667, 2011.

Metastudies

- Lazaro Costa
  - https://dl.acm.org/doi/10.1145/3641525.3663623

Literate programming

- Jupyter, Amazon Sagemaker, Google Colab, Deepnote, Hex, Databricks Notebooks, DataCamp Workspace, JupyterLab, HyperQuery, Jetbrains Datalore, kaggle, NextJournal, Noteable, nteract, Observable, Query.me, VS Code notebooks, Mode Notebooks, Querybook, Zeppelin, Count, Husprey, Pluto.jl, Polynote, Zepl – https://datasciencenotebook.org/
- Ten Simple Rules for Reproducible Research in Jupyter Notebooks by Rule et al. https://arxiv.org/abs/1810.08055
- Jupyter
- Binder
- S. Li-Thiao-T. Literate program execution for reproducible research and executable papers. Procedia CS, 9:439–448, 2012
- D. Koop, E. Santos, P. Mates, H. T. Vo, P. Bonnet, B. Bauer, B. Surer, M. Troyer, D. N. Williams, J. E. Tohline, J. Freire, and C. T. Silva. A provenance-based infrastructure to support the life cycle of executable papers. Procedia CS, 4:648–657, 2011

Source archival

- Software Heritage Archive
- ACM REP '23 and ACM REP '24 have work on this
- DOI

Workflows

- Taverna
- VizIt
- Galaxy
- WorkflowHub
- Connection between workflow and provenance

Provenance

- What is provenance? Survey of provenance by (Freire)
- How does provenance connect to reproducibility

Record/replay tools

- CDE
- RR
- CARE
- Sciunit (Malik)
- ReproZip
- Sumatra

- Preserving the mess or encouraging cleanliness (Thaine)

Version control

Virtualization and containerization

- How does virtualization and containerization connect to reproducibility
  - Docker
  - Singularity

Package managers

- How do package managers connect to reproducibility
  - Nix
  - Guix
  - Spack

Retroactively restoring reproducibility

- Continuous integration
- Automatic build repair for reproducibility
- Cindy Rubio-Gonzales
- ShipWright
- Jupyter studies

Other aspects of reproducibility

- Relationship to replicability
- Definitions/terms
- (Marwick, 2015) highlighted the challenge posed by computer programs: they act as black boxes. – https://arxiv.org/pdf/2402.07530
- Collberg and Proebsting responses to study
- How to address source code sharing
  - Collberg and Proebsting responses to code sharing
  - Stodden
  - ACM REP '23 and ACM REP '24 have work on this
  - Incentives
    - Artifact badging
    - Vandewalle et al.
- FAIR for RSware
- Why does RSE work matter for reproducibility?
  - Bridging the gaps
- Findability
  - M. Gavish and D. Donoho. A universal identifier for computational results. Procedia CS, 4:637-647, 2011.
- Software citation
- Publishing modes/executable papers
  - Research notebooks
  - Semantic description of hypothesis
  - P. V. Gorp and S. Mazanek. Share: a web portal for creating and sharing executable research papers. Procedia CS, 4:589–597, 2011
- J. Kovacevi (check name). How to encourage and publish reproducible research. In IEEE International Conference on Acoustic Speech Signal Processing, volume IV, pages 1273–1276, Apr. 2007.
- Journals

  - N. Limare. Running a reproducible research journal, with source code inside. In ICERM Workshop, 2012.
  - ReScienceC (Rougier et al. 2017)
  - An increasing number of journals and conferences are concerned with the reproducibility of published articles (Drummond, 2018) (Bajpai et al., 2019).
  - Image Processing On Line (IPOL) [Miguel Colom, Bertrand Kerautret, Nicolas Limare, Pascal Monasse, and Jean-Michel Morel. 2015. IPOL: a new journal for fully reproducible research; analysis of four years development. In 2015 7th International Conference on New Technologies, Mobility and Security (NTMS). IEEE, 1–5.]

- Copyright
  - V. Stodden. The legal framework for reproducible scientific research: Licensing and copyright. Computing in Science and Engineering, 11(1):35–40, 2009.
- Domain-specific testbeds
  - Networking guy from ACM REP '24
  - Triscale
- We found one survey providing state-of-the-art reproducibility in scientific computing (Ivie and Thain, 2018), and several books attempting to do so (Desquilbet et al., 2019) (National Academies of Sciences, 2019) (Randall and Welser, 2018). – https://arxiv.org/pdf/2402.07530
- In his article (Drummond, 2018), Drummond asserted that sharing the source code of an article is unnecessary. He believes that researchers are forced to do so to avoid getting a bad label but that it does not serve science. For him, the reproducible research movement was not based on facts, but only on intuition. He adds that the obligation to provide the source code will lead to papers being accepted based on technically weak criteria and that, according to his opinion, fraud has always existed and never posed a significant problem. However, Drummond supports the concept of open science. – https://arxiv.org/pdf/2402.07530
- Prior work of https://web.archive.org/web/20220119115703/http://reproducibility.cs.arizona.edu/v2/Repeatabili
- http://reproducibleresearch.net/

My research strategy

Irreproducible science (end problem) -> computational irreproducibility (cause) -> source inavailability (cause), crash-free computational irreproducibility (proxy) -> {pro-active activities with limited cost, re-active responses}

source inavailability (cause) -> cost of supporting reproduction

pro-active -> PROBE archive

re-active -> PROBE with automated environment exploration

Translational CS

Completed work

Lifetime of workflows in sample

Lifetime analysis

Literature review and evaluation of record/replay tools

Record/replay benchmarks (done)

System-level prov tracer

Proposed work

Prov tracer (aka Povenance & Replay OBservation Engine)

Re-executable provenance RO-crates

Using liftetime analysis for CI testing

Using PROBE in ECMF

Script -> Workflow conversion

User studies

PROBEd environment manipulation

Intellectual merit

What is the importance of the activity to advancing knowledge or understanding?

- Validate theoretical model
- Theoretical model systematizes knowledge
- Unique treatment of reproducibility as cost/benefit
- Encourage use of provenance (transformative concept)
- Uniting OS background with TAM and user studies

Expected impact

What impact can be expected in terms of particular research communities and on society in general?

- "Push button" reproducibility
- Reproducibility helps inclusivity in research in grad school and globally

Feasibility

How likely are the stated goals to be achieved by the candidate?

Timeline

Bibliography

Notes

Important dimension to reproducibility approaches: time. For how long does a particular approach yield reproducibility? A Dockerfile that is more than a few months old might not even build.

Converting sys level to human level

Explicitly state what is the same

Robustness to software environment changes

Empirical studies of repro

- Send schehdule
- Send expectations
- Send draft
- Email Tim re PROBE
- Find prior art definitions

ACM Inc. staff. 2020. "Artifact Review and Badging." August 24, 2020. https://www.acm.org/publications/policies/artifact-review-and-badging-current.

Collberg, Christian, and Todd A. Proebsting. 2016. "Repeatability in Computer Systems Research." *Communications of the ACM* 59 (3): 62–69. https://doi.org/10.1145/2812803.

Krafczyk, M. S., A. Shi, A. Bhaskar, D. Marinov, and V. Stodden. 2021. "Learning from Reproducing Computational Results: Introducing Three Principles and the Reproduction Package." *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 379 (2197): 20200069. https://doi.org/10.1098/rsta.2020.0069.

Pimentel, João Felipe, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. 2019. "A Large-Scale Study About Quality and Reproducibility of Jupyter Notebooks." In *Proceedings of the 16th International Conference on Mining Software Repositories*, 507–17. MSR '19. Montreal, Quebec, Canada: IEEE Press. https://doi.org/10.1109/MSR.2019.00077.

Trisovic, Ana, Matthew K. Lau, Thomas Pasquier, and Mercè Crosas. 2022. "A Large-Scale Study on Research Code Quality and Execution." *Scientific Data* 9 (1): 60. https://doi.org/10.1038/s41597-022-01143-6.

Vandewalle, Patrick, Jelena Kovacevic, and Martin Vetterli. 2009. "Reproducible Research in Signal Processing." *IEEE Signal Processing Magazine* 26 (3): 37–47. https://doi.org/10.1109/MSP.2009.932122.

Wang, Jiawei, Tzu-yang Kuo, Li Li, and Andreas Zeller. 2021. "Assessing and Restoring Reproducibility of Jupyter Notebooks." In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, 138–49. ASE '20. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3324884.3416585.

Zhao, Jun, Jose Manuel Gomez-Perez, Khalid Belhajjame, Graham Klyne, Esteban Garciacuesta, Aleix Garrido, Kristina Hettne, Marco Roos, David De Roure, and Carole Goble. 2012. "Why Workflows Break — Understanding and Combating Decay in Taverna Workflows." In *2012 IEEE 8th International Conference on E-Science (e-Science)*, 9. Chicago, IL: IEEE. https://doi.org/10.1109/eScience.2012.6404482.