Computational Reproducibility: A Primer from UKRN

Authors (A-Z): Anna Schultze, Louise Saul, Etienne Roesch, Eike Mark Rinke, Osama Mahmoud, Mark Kelson, Lisa DeBruine, Alejandro Coca-Castro, Fiona Booth, Reny Baykova

UKRN Reviewers: Nicolas Rougier

Introduction

The intended audience for this primer is anyone who is involved in research activity that relies on a computer, for any single step of the research process[1]. We describe aspects of reproducibility that depend on the use of a computer, explain what can go wrong and propose mitigation strategies. No technical knowledge is required, and readers wanting to develop their understanding are presented with a selection of references.

We can all agree that, given the exact same data and same computational analyses, we should be able to produce the same results; this is termed "computational reproducibility". You might think that this bare-minimum level of reproducibility would be easy to achieve, but many studies have found it is not, for a variety of reasons. For example, when analytic code is not supplied, analyses described in text may be interpreted in multiple ways. When code is supplied, it may omit necessary resources that are present on the authors' computer, but not the users' computer. Additionally, versions of software may change or packages become unavailable, leading to un-runnable scripts. This primer covers some practices that increase computational reproducibility and points you to sources for further learning.

Our general advice is to strive to understand the methods and tools you are using in as much detail as possible and establish the level of reproducibility and reliability that is appropriate to them. It is also important to note that reproducibility does not imply that a computation is correct, but it allows for others to understand what has been done, in support of the incremental build-up of knowledge – and it's okay to be wrong. A healthy first mental shortcut may be to assess how the computing tools you are using follow the FAIR process, and are findable, accessible, interoperable, and reusable (Lamprecht et al, 2020; Barker et al., 2022).

What is computational reproducibility?

The vocabulary surrounding issues of reproducibility and replicability is, unfortunately, confusing and dependent on historical developments in each academic field (Barba, 2018). For the purpose of this primer, we are following the definition proposed by the Royal Society (Krafczyk et al., 2021) and define computational reproducibility as the ability to obtain

[1] We acknowledge that the severity of threats to computational reproducibility can vary with the complexity of the computational resources involved. However, the issues reviewed in this primer affect all software, including tools that may appear mundane. Accordingly, our remit extends to aspects of computation that researchers may neither directly control nor typically consider susceptible to reproducibility issues. This includes, for example, online reference databases, coding assistants for qualitative research, and general utilities such as web-based visualisation tools, all of which depend on underlying computational layers that may themselves be unstable or non-reproducible. Even seemingly peripheral infrastructure, such as web servers used to deliver dynamic content, is prone to inconsistencies across platforms and software releases. These challenges are further compounded when AI systems are involved.

consistent computational results using the same input data, computational steps, methods, code, and conditions of analysis. We distinguish it from replicability, which is the ability for independent researchers using different methods (e.g., a different programming language, a different software) to address the same hypothesis and hopefully obtain consistent results (Peng, 2011).

Continuum of computational reproducibility

In our definition of computational reproducibility, by using the word "consistent", we mean that the results should be similar enough to allow us to judge that the reproduction was successful. In some contexts, rounding differences will be acceptable. In other contexts, we might expect bitwise reproducibility, that is, that the results are exactly the same, down to the last bit. It is important to keep in mind the conditions that will influence the computational reproducibility of a given piece of work:

Is the time frame explicit? - No computational work will ever be reproducible in perpetuity. As hardware and software update and change over time, code that once worked would no longer function (a phenomenon known as code rot/decay; Eick et al., 2001). Thus, the choice of computational ecosystem (programming language, hardware, operating system, etc) influences the ability to reproduce work in the future.

What is the cost for someone to reproduce your work? - It can take significant investment in time and resources to understand, amend or implement a reproduction of someone else's work. It is the responsibility of the initial authors to reduce this burden, making explicit decisions about the skills and resources needed to reproduce their work.

Is the computational environment sufficiently described? — Reproducibility is only possible if all the relevant information about a piece of computational work is available. It is the responsibility of the initial authors to understand the environment they are using and describe it appropriately (Gallagher et al., 2024), to envisage what could go wrong in the future, and not simply expect that it will work out of the box.

What level of reproducibility is expected? — Each community will have expectations as to what constitutes an appropriate level of proof of reproducibility, asserting trustworthiness in the results, from accepting a claim that includes rounding differences at significant digits to demonstrating bitwise reproducibility of the computation.

Why prioritise computational reproducibility?

Computational reproducibility is a concern across many disciplines, as long as they rely on computers for any part of their work. This is not just a philosophical concern over best research practices: on occasion, there have been research misconduct allegations and legal issues due to lack of consideration for computational reproducibility (Dyer, 2015). Beyond misconduct, issues of computational reproducibility are often misunderstood and misrepresented by researchers and a source of "questionable" research practices.

Sources of irreproducibility

Provided you have shared your data and analysis code online, potential sources of irreproducibility include:

- A lack of a detailed description of the study e.g. insufficient information on the study protocol (Breznau et al., 2025)
- Insufficient documentation of data and metadata e.g. lack of documentation makes reproducibility by independent researchers impossible (Gertler et al., 2018)
- No human-readable description of data processing steps for data cleaning prior to analysis (Trisovic et al., 2022)
- Incomplete description of implementation of collection, processing, and analysis in the software e.g. missing code section, insufficient code literacy to prepare software for sharing (Herndon et al., 2014)
- Insufficient description of software dependencies and computational environment leading to irreproducible computation (Glatard et al., 2015; Pauli et al., 2016)
- Differences in hardware environment leading to differences in outputs (Vila et al., 2024)

It is not expected that every researcher enrols onto a computer science degree. However, it is reasonable to expect that researchers understand the tools they are using, and are able to describe them to others. When gaps in knowledge exist, researchers should feel empowered to train themselves, especially as they tend to create bespoke workflows and research software. Concurrently, institutions need to promote the adoption of standards and support dedicated roles, such as data stewards or dedicated trainers, and supportive communities of practice for beginners.

How to enable computational reproducibility?

To promote computational reproducibility, steps can be introduced that will result in standardised and/or more explicit workflows. This section is relevant for users who are engaged in research activities and using computers at any phase of their workflow, who want to check someone else's work or reproduce it, or provide a checklist for researchers producing code—basically everyone! We suggest implementing some of the steps below into your workflows, to increase transparency and trust in your data and code.

Where this is not applicable, we suggest the use of alternative methods of recording of the steps: e.g., if a specific piece of software or a particular context does not permit the use of explicit scripts in plain text, maybe requiring the use of “point and click” interfaces, one might consider recording a video, or writing up the steps and asking a colleague to reenact the steps for verification.

Step 0: Decide who you think should be able to reproduce your work

Imagine a single researcher who you think should be able to evaluate and reuse your work (including yourself sometime in the future). Give them a name. What is their technical background? What is the minimal knowledge needed to perform the same steps you did? Table 1 describes practices that you may decide to engage with, depending on your workflow.

Step 1: Pay attention to how you organise your data

Create a plan that reviews every step of the workflow: spanning the gathering and generation of raw information (data), processing the data (if appropriate), analysing them and preparing them for archiving and re-use. Each discipline will have its own set of requirements, and we recommend starting with a Data Management Plan; see: https://dmponline.dcc.ac.uk/ and https://psych-ds.github.io/ (see Michener, 2015).

Step 2: Make sure your data is readable

Provide a data provenance description for your data, placed in a README (www.makeareadme.com) that is linked to the data with a data dictionary associated with the relevant data (Mitchell, 2022). See also: https://datadryad.org/best_practices

Step 3: Use code to pre-process and analyse your data

Using packages and libraries in languages such as R and Python to clean data and to create figures and tables can improve reproducibility through reducing bias (Haider et al., 2020; Omta et al., 2020). Knowledge of the methodology and software used is needed to ensure that computations are correct.

Step 4: Pay attention to how you organise your code and make sure it is readable

Commenting code will help your future self and others understand what has been done. It is a low-hanging fruit supporting reproducibility that will save a ton of time (Thomas & Hunt, 2019; Benureau & Rougier, 2018; Bednar, 2022).

Step 5: Write down versions of your software and specifications of the hardware you use

Realising that every component of the computational workflow matters even for seemingly simple code, and is described in as much detail as relevant, including hardware (Vallet, Michonneau & Tournier, 2022) and code (Filazzola & Lortie, 2022).

Step 6: Share your data and code

A UKRN primer by Turner et al. (2020) highlights key considerations on why and how open source software and open code complements reproducible research. Following FAIR Principles (findability, accessibility, interoperability and reusability) is a good way to prepare data and code for sharing (https://faircookbook.elixir-europe.org). Licence your data and code so that others know how the materials can be used.

Step 7: Audit your shared materials regularly

As technology evolves and dependencies break, code may rot and become useless pretty quickly (within months, not years). When you share materials online, it becomes your responsibility to maintain them appropriately (Thomas & Hunt, 2019, Test-driven development, p. 261). Using storage platforms and public repositories that allow tracking changes in material over time, i.e. version control, and support access by multiple individuals, makes it possible for code and data to be audited (Braga et al., 2023). See the UKRN primer on version control to get started (Strain et al., 2025). Note that public repositories themselves can disappear, and it is your responsibility to choose a sustainable platform.

Step 8: Assign responsibilities for steps in computational workflows within institutions

Especially for a large codebase or complex workflows, it is important to structure the output around steps for quality assurance. Responsibilities should be assigned to ensure sustained robustness, and relevant practices employed as needed, like test driven development (Thomas & Hunt, 2019).

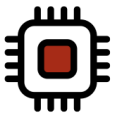
	→ Increasing usefulness (and effort) →			
	Curated data available in a findable source (e.g., OSF)		Curated data permanently archived in a reputable archive	
	Workflow documented in text or video for exact reproduction	Workflow coded or scripted for automatic reproduction	Code/scripts available in a findable source (e.g., GitHub)	Code/scripts permanently archived in a reputable archive
	Software and dependencies mentioned in a README	Specific versions of software and dependencies are mentioned	Software and dependencies provided or linked in permanent archive	Exact software and dependencies provided in a container
	Computational environment (e.g., operating system) mentioned in a README		Exact computational environment provided in a container	
	Hardware requirements described in a README		Choose to use specific hardware as agreed in your community	

Table 1. Five categories of practices to increase computational reproducibility, arranged by increasing usefulness and effort.

Case studies that demonstrate the implementation of computational reproducibility

We invite the reader to read the papers we cite in this primer, which describe a lot of good practices. To see some of these practices in action, refer to:

- Primer on Reproducible Research in R (Siraji and Rahman, 2023)
- Society for Open, Reliable, and Transparent Ecology and Evolutionary Biology: peer code review (Ivimey-Cook et al., 2023)
- British Ecological Society: guide to reproducible code (Croucher et al., 2024)
- Improving reproducibility in geoscientific papers (Coca-Castro et al., 2025)
- CodeCheck (Eglen and Nust, 2025)
- 'ReproduceMe: Lessons from a pilot project on computational reproducibility (Baker et al., 2024)
- Pre-publication reproducibility certification (Baykova, 2024)

References

Baker, D.H. et al. (2024) 'ReproduceMe: Lessons from a pilot project on computational reproducibility', Meta-Psychology, 8. https://doi.org/10.15626/MP.2023.4021

Barba, L. A. (2018). Terminologies for reproducible research. https://arxiv.org/abs/1802.03311

Barker, M., Chue Hong, N. P., Katz, D. S., Lamprecht, A.-L., Martinez-Ortiz, C., Psomopoulos, F., Harrow, J., Castro, L. J., Gruenpeter, M., Martinez, P. A., & Honeyman, T. (2022). Introducing the FAIR Principles for research software. Scientific Data, 9, 622. https://doi.org/10.1038/s41597-022-01710-x

Baykova, R. (2024, Jan 30). Pre-submission certification of computational reproducibility. UKRN. https://www.ukrn.org/2024/01/30/pre-submission-certification-of-computational-reproducibility/

Bednar, J. (2022). 8 levels of reproducibility: Future-proofing your Python projects. https://www.anaconda.com/blog/8-levels-of-reproducibility

Benureau, F. C. Y., & Rougier, N. P. (2018). Re-run, repeat, reproduce, reuse, replicate: Transforming code into scientific contributions. Frontiers in Neuroinformatics, Volume 11-2017. https://doi.org/10.3389/fninf.2017.00069

Braga, P.H.P. et al. (2023) 'Not just for programmers: How GitHub can accelerate collaborative and reproducible research in ecology and evolution', Methods in Ecology and Evolution, 14(6), pp. 1364–1380. https://doi.org/10.1111/2041-210X.14108

Breznau, N., Rinke, E. M., Wuttke, A., Adem, M., Adriaans, J., Akdeniz, E., Alvarez-Benjumea, A., Andersen, H. K., Auer, D., Azevedo, F., Bahnsen, O., Bai, L., Balzer, D., Bauer, P. C., Bauer, G., Baumann, M., Baute, S., Benoit, V., Bernauer, J., … Nguyen, H. H. V. (2025). The reliability of replications: A study in computational reproductions. Royal Society Open Science, 12(3), 241038. https://doi.org/10.1098/rsos.241038

Coca-Castro, A. et al. (2025) 'Improving the reproducibility in geoscientific papers: Lessons learned from a Hackathon in climate science', Environmental Data Science, 4, e6. https://doi.org/10.1017/eds.2024.35

Croucher, M. et al. (2024) A guide to reproducible code in ecology and evolution. http://hdl.handle.net/10141/622618

Dyer, C. (2015) 'Duke University settles lawsuits alleging that patients were harmed in chemotherapy trials', BMJ, 350, p. h2559. https://doi.org/10.1136/bmj.h2559

Eglen, S. and Nust, D. (2025) CODECHECKing goes NL - strengthening open science through reproducible publications in the Netherlands, https://codecheck.org.uk/nl/

Eick, S. G., Graves, T. L., Karr, A. F., Marron, J. S., & Mockus, A. (2001). Does code decay? Assessing the evidence from change management data. IEEE transactions on software engineering, 27(1), 1-12. https://doi.org/10.1109/32.895984

FAIRplus and Elixir (2025) FAIR cookbook. https://faircookbook.elixir-europe.org

Filazzola, A. and Lortie, C. (2022) 'A call for clean code to effectively communicate science', Methods in Ecology and Evolution, 13(10), pp. 2119–2128. https://doi.org/10.1111/2041-210X.13961

Gallagher, K., Creswell, R., Lambert, B., Robinson, M., Lei, C. L., Mirams, G. R., & Gavaghan, D. J. (2024). Ten simple rules for training scientists to make better software. PLOS Computational Biology, 20(9), e1012410. https://doi.org/10.1371/journal.pcbi.1012410

Gertler, P., Galiani, S. and Romero, M. (2018) 'How to make replication the norm', Nature, 554(7693), pp. 417–419. https://doi.org/10.1038/d41586-018-02108-9

Glatard, T. et al. (2015) 'Reproducibility of neuroimaging analyses across operating systems', Frontiers in Neuroinformatics, 9. https://doi.org/10.3389/fninf.2015.00012

Guo, D. (2025) README 101. https://www.makeareadme.com/

Haider, S.N., Zhao, Q. and Meran, B.K. (2020) 'Automated data cleaning for data centers: A case study', in 2020 39th Chinese Control Conference (CCC). IEEE, pp. 3227–3232. https://doi.org/10.23919/CCC50068.2020.9189357

Herndon, T., Ash, M. and Pollin, R. (2014) 'Does high public debt consistently stifle economic growth? A critique of Reinhart and Rogoff', Cambridge Journal of Economics, 38(2), pp. 257–279. https://doi.org/10.1093/cje/bet075

Ivimey-Cook, E.R. et al. (2023) 'Implementing code review in the scientific workflow: Insights from ecology and evolutionary biology', Journal of Evolutionary Biology, 36(10), pp. 1347–1356. https://doi.org/10.1111/jeb.14230

Krafczyk, M.S. et al. (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). https://doi.org/10.1098/rsta.2020.0069

Lamprecht, A.-L., Garcia, L., Kuzak, M., Martinez, C., Arcila, R., Martin Del Pico, E., Dominguez Del Angel, V., van de Sandt, S., Ison, J., Martinez, P. A., McQuilton, P., Valencia, A., Harrow, J., Psomopoulos, F., Gelpi, J. Ll., Chue Hong, N., Goble, C., & Capella-Gutierrez, S. (2020). Towards FAIR principles for research software. Data Science, 3(1), pp. 37–59. https://doi.org/10.3233/DS-190026

Michener, W.K. (2015) 'Ten simple rules for creating a good data management plan', PLOS Computational Biology, 11(10), e1004525. https://doi.org/10.1371/journal.pcbi.1004525

Mitchell, S. N., Lahiff, A., Cummings, N., Hollocombe, J., Boskamp, B., Field, R., Reddyhoff, D., Zarebski, K., Wilson, A., Viola, B., Burke, M., Archibald, B., Bessell, P., Blackwell, R., Boden, L. A., Brett, A., Brett, S., Dundas, R., Enright, J., … Reeve, R. (2022). FAIR data pipeline: Provenance-driven data management for traceable scientific workflows. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 380(2233), 20210300. https://doi.org/10.1098/rsta.2021.0300

Omta, W.A. et al. (2020) 'PurifyR: An R package for highly automated, reproducible variable extraction and standardization', Systems Medicine, 3(1), pp. 1–7. https://doi.org/10.1089/sysm.2019.0007

Pauli, R. et al. (2016) 'Exploring fMRI results space: 31 variants of an fMRI analysis in AFNI, FSL, and SPM', Frontiers in Neuroinformatics, 10. https://doi.org/10.3389/fninf.2016.00024

Peng, R.D. (2011) 'Reproducible research in computational science', Science, 334(6060), pp. 1226–1227. https://doi.org/10.1126/science.1213847

Siraji, M.A. and Rahman, M. (2023) 'Primer on reproducible research in R: Enhancing transparency and scientific rigor', Clocks & Sleep, 6(1), pp. 1–10. https://doi.org/10.3390/clockssleep6010001

Strain, G., Lee, C., Boneham, S., Shen, X., Coca-Castro, A., & UK, R. N. (2025). Software version control using Git: A primer from UKRN. https://doi.org/10.31219/osf.io/vqru4_v1

Thomas, D., & Hunt, A. (2019). The pragmatic programmer, 20th anniversary edition: Journey to mastery. 2nd ed. Addison-Wesley.

Trisovic, A. et al. (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

Turner, A. et al. (2020) ‘Open code and software: A primer from UKRN’. https://doi.org/10.31219/osf.io/qw9ck

Vallet, N., Michonneau, D. and Tournier, S. (2022) ‘Toward practical transparent verifiable and long-term reproducible research using Guix’, Scientific Data, 9(1), 597. https://doi.org/10.1038/s41597-022-01720-9

Vila, G., Medernach, E., Gonzalez Pepe, I., Bonnet, A., Chatelain, Y., Sdika, M., Glatard, T. and Camarasu Pop, S. (2024) ‘The Impact of Hardware Variability on Applications Packaged with Docker and Guix: a Case Study in Neuroimaging’, in Proceedings of the 2nd ACM Conference on Reproducibility and Replicability. New York, NY, USA: ACM, pp. 75–84. https://doi.org/10.1145/3641525.3663626