University of Hamburg
Department: Computer Science

Essay

Reproducibility

By: Tim Rolff

Supervised by: Dr. Michael Kuhn

1 Introduction

Reproducible science affects everyone, this is one of the reasons why reproducibility is a supporting pillar of science. As for example medical treatment is based on research, and a recent study of Bayer stated, that only 6 of 53 studies about cancer could be reproduced by them [BE12]. Therefore it is necessary to provide results which could be used as a reliable and reproducible source. Since computers became available for most scientists and researchers, it raises the question if they are reliable enough to support facts and theories. Because empirical science rely on the idea that every observation which have been made could be made again, given the tools. As these tools get more difficult and harder to use it is also more challenging to achieve correct results. This may be one of the reasons why some scientists considered science in a "Replication Crisis" around 2010 [Sch14]. Another reason for this could be the often interchangeably used concepts of reproducibility and replicability which are often not defined precisely and therefore depends heavily on the research field and the authors.

In this paper I try to give an overview over the field of reproducible science including - but not limited to - the context of reproducibility in computer science and especially in computer simulations.

2 Background

2.1 What is Reproducibility / Replicatibility

Before starting deeper into the topic, I want to give a brief overview what reproducibility is. First of all it is extremely hard to get the definition by looking it up, as for example the Oxford dictionary gives the following definition: "Able to be reproduced or copied."[1] for the word "reproducible", where the word "replicate" is defined as "Make an exact copy of"[2]. One might argue that the word "exact" states the difference between being reproducible and replicatable, but in my view the term "copy" indicates that scientific experiments are identical to each other. If that's the case, each experiment is nevertheless a replica of each other and therefore not reproducible by definition, because a copy is always an exact replication. But arguing with word definitions is always cumbersome, therefore an often used argumentation is made by J. T. Leek and R. D. Peng [LP15] which define reproducibility as:

[1] https://en.oxforddictionaries.com/definition/reproducible

[2] https://en.oxforddictionaries.com/definition/replicate

"We define reproducibility as the ability to recompute data analytic results given an observed data set and knowledge of the data analysis pipeline."

— J. T. Leek and R. D. Peng, [LP15]

While this definition is very common and a similar definition has been made by [Son07], it has some problems. First of all this definition of reproducibility might result in wrong results, if the data set was measured wrong or got cherrypicked for specific data. This might therefore result in the same wrong recomputed results as the original publication. Consequently having a reproducible experiment with this definition does not imply a correct conclusion. In another paper R. D. Peng defines reproducibility for publications in computer science, in which he defines multiple levels of reproducibility (see figure 1).

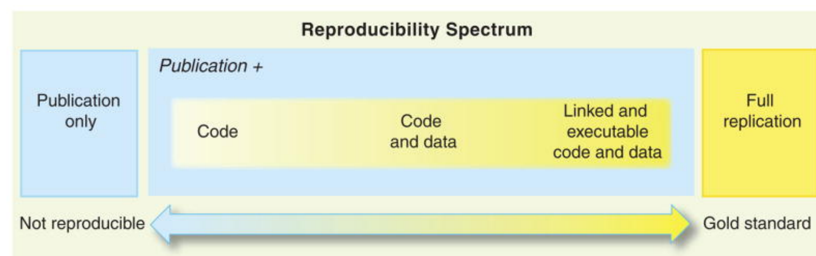

Figure 1: Definition of reproducibility by R. D. Peng [Pen11]

This special definition suffers the same problems as the previous definition and extend it for publications which include source code. The consequence is that the source can be just recompiled or the program which is shipped can be run again with different data. This has the problem, that it may not show any bug or error in the code and therefore does not contribute to reproducible research. But it is arguable that the source code should be checked before running the code again or should have been rewritten to avoid the same errors as the original source code, then this could be also possible with a publication and the data only, which again does not avoid wrong data. Another approach to define reproducibility has been made by C. Drummond [Dru09] where he stated, that his definition of reproducibility is, that the conclusion of a single experiment can be verified with a different experiment. This results in the fact that another researcher doesn't need to use the same experiment to come to the same conclusion as the original author of a paper. He also extends the definition to different levels of reproducibility, where an experiment, which is more different from the original one, is more valuable than one which does

simply use the same tools. He also states, that replication is the weakest form of reproducibility, because it aims to copy the original experiment exactly with all it flaws.

While this definition has the ability to have different experiments which come to the same conclusion, it however has also some problems. What if no other experiment is known to support the conclusion of the first experiment or would be to costly in time or money to prove the results. Then it is less reproducible by the definition of Drummond and is therefore has not same the value as an different experiment which supports the first one. This might be true for experiments which rely on source code only and accordingly could just be run again on another computer. This clearly doesn't offer any value because as argued before, the source code might have bugs or other errors which are not considered and hence lead to the same or another wrong conclusion of the first experiment. But in a more traditional experiment, as in physics or chemistry, there is always the possibility that the measurement utility might have been defect or used incorrectly. As for example in the of the "faster than light neutrinos" experiment, where a loose cable and a slightly faster CPU clock[3] resulted in wrong published results. So having another team rebuild the experiment with different tools should give other results if the first experiment was not executed correctly. Because it is unlikely that the same error made be twice, except if the experiment or the theory behind it is wrong. If that's the case it is clearly necessary to have a supporting or contradicting experiment which show other results, but this leads then to the question which experiment was wrong. Additionally having two experiments, which came to the same conclusion does not indicate that one experiment is correct and reproducible because a false experiment could imply a true conclusion ($a \Rightarrow b$).

For the rest of this paper I want to define replicatibility as reproducing an experiment as an exact copy of on existing one, which results in the same (wrong) results and conclusions. While I want to define reproducibility as a mixture of both papers, where it is necessary to have code and data as a reference as in [Pen11] and [LP15] on the original approach to the specific problem. This should avoid getting a completely different experiment, which is not related to the original one, but not rely on them to be able to rewrite or rebuild the experiment for ground up without using the exact same tools, to avoid the same errors which could have been made by the original authors. Even further it would be ideal to reproduce the same results as C. Drum-

[3] `https://en.wikipedia.org/wiki/Faster-than-light_neutrino_anomaly` last visited at 02.08.2017

mond [Dru09] stated by an different experiment if it is possible to support / contradict the reproducibility of conclusions of the original experiment but this doesn't make a reproduction of the same experiment with different tools less valuable if the theory behind the experiment is well supported by other facts and experiments. Because these experiments might show some errors in the original measurements and therefore guarantee the reproducibility of an specific experiment. The results can therefore be used to support / contradict the reproducibility of conclusions of a theory.

3 Reproducibility in Simulations

While a typical experiment rely heavily on measuring tools to collect data for a conclusion, the line between simulations and traditional experiments is not that sharp anymore with the influence of computers in science such as in physics with e.g. the CERN experiment, where a computer system selects which data should be recorded, because the amount of data is to large to store[4]. While other experiments generate data which can be compared to the real world such as weather / climate simulations. Having such additional components introduce another layer of uncertainty where errors can happen. Additionally there is a difference between run- and compile- time reproducibility, which I want to address in the following sections.

3.1 Runtime (Ir-) Reproducibility

By using a computer as an additional layer for an experiment or as an simulation device and because there is a variety of effects which can result in irreproducibility, such as floating-point errors, non-deterministic algorithms, hardware defects / effects and unknown internals of the hardware itself or the physical influence on these devices as for example by quantum effects, it is challenging to achieve a reproducible program. In this section a want to address these effects which happen mostly at runtime.

3.1.1 Floating-Point Calculation

First of all, a floating-point calculation, which is calculated accordingly to the IEEE 754 standard, is deterministic on the same machine in the context, that it returns the same value for the same input and under the condition, that the calculation happens only in one thread. But because many modern simulations rely heavily on the use of multiple processors, this determinism

[4] `https://home.cern/about/computing/processing-what-record`

is not always given, such as in the following case which K. Diethelm describes in [Die12]. Lets consider we have an program which runs on four cores, each of these cores compute a calculation and return the computed value to the first core which then calculates the sum of all results:

$$\underset{\text{core 1}}{10^{12}} + \underset{\text{core 2}}{-10^{12}} + \underset{\text{core 3}}{10^{-8}} + \underset{\text{core 4}}{10^{-8}} = 2 \cdot 10^{-8}$$

While this results in the correct value now lets consider that the order in which the cores finished is the following:

$$\underbrace{\underset{\text{core 1}}{10^{12}} + \underset{\text{core 3}}{10^{-8}}}_{10^{12}} + \underset{\text{core 2}}{-10^{12}} + \underset{\text{core 4}}{10^{-8}} = 10^{-8}$$

Because the floating-point accuracy is not high enough to calculate $10^{12}+10^{-8}$ correctly, the value get truncated. This leads to a wrong result, which differs from the first value by 50%. As this example show it is fairly simple to get a irreproducible program, without engage a specific use-case or a design strategy. As a solution, it is possible to use one of many algorithms which aims to make floating-point calculations reproducible, such as "Correct Rounding and a Hybrid Approach to Exact Floating-Point Summation" [ZH09] which has the drawback, that it needs an additional array and computing time. Alternatively other approaches can be used such as interval arithmetic where the result is guaranteed to be in between the given upper and lower bound, uncertainty quantification, fixed-point arithmetic or by simply using a higher accuracy through specific libraries, as for example GMP and MPFR.

3.1.2 Non-Deterministic Algorithms

Another source of irreproducibility might be the use of a non deterministic algorithm, as for example if the simulation get driven by some stochastic process or other random processes. But this is not restricted to random processes in the first place, a program might be get irreproducible by changing some algorithms. One might argue, that changing an algorithm is always non deterministic, but I also want to point out, that even changing a formula can result in an non deterministic algorithm. Take for example the formula $a \cdot b + a \cdot c$ it might be better to rewrite this as $a \cdot (b + c)$ to simply gain performance or achieve better readability. This happens to be a non deterministic floating-point transformation where

$$a \cdot (b + c) \neq a \cdot b + a \cdot c$$

is not guaranteed to give the same result[5]. In this case it might be reasonable to avoid using different algorithms, by properly assigning a new version number to the program and avoid using different versioned programs. It might also the case that the real implementation of an algorithm is unknown, because it wasn't described properly in the publication as for example if the original publication uses an stable sorting algorithm, the reimplementation might be an unstable one, which therefore result in wrong results. These kind of irreproducibilties can be avoided by documenting the used algorithms correctly or publish the source code simultaneously with the paper.

3.1.3 Hardware and Environment Effects / Defects

Even if the program itself is perfectly written and avoids irreproducible behavior it might be the case, that the results are irreproducible. This can be caused by defect hardware or by hardware which happens to have a hardware bug, such as the famous Pentium-FDIV-Bug, the Intel Skylake and Kaby-Lake Hyperthreading or the Ryzen FMA3 bug. While such huge bugs are very rare, there a several minor bugs, which are not fixed yet and which might not have been taken into account while running the simulation. As for example in the Kaby Lake architecture[6]. While it is mostly impossible to fix those errors without fixing the microcode or redesign the CPU, these errors can often be avoided, by carefully writing a workaround or using a newer compiler which is aware of those bugs. In addition to the hardware bugs there might be bugs caused by quantum sized effects. As for example in memory, where bits might get flipped by magnetic or electrical influence. While this can be prevented using ECC-memory there are other effects such as leakage of electrons in the semiconductor where an electron tunnel through a transistor and might cause an unwanted switch of the circuit, which could result in irreproducible behavior and can't be prevented easily.

Equally to the just discussed hardware effects / defects there a several influences on a program, by its runtime environment, such as bugs in used programs or the operating system. These can influence for example the score of a benchmark or the runtime of a simulation. Because most software rely on the operating system for file input/output or drivers for network it might be the case. By changing these the program might give irreproducible results, as for an example in a benchmark or a runtime measurement. It could also be the case that the topology of a network, or the load of a system, could

[5] `https://gcc.gnu.org/wiki/FloatingPointMath`

[6] `https://www.intel.com/content/dam/www/public/us/en/documents/specification-updates/7th-gen-core-family-spec-update.pdf`

influence these. Especially in the case of a runtime measurement. Even if that is not the case, it is possible that the used tools are not reproducible, or the original paper used another version of a tool, that happen to introduce a irreproduibility over multiple version and therefore don't ensure the same outcome. It is also possible to go further and argue that the system was influenced by / infected with malicious software or a background process such as anti-virus system which interacts with the program in a way, that changes data and therefore result in irreproducible behavior.

3.2 Compiletime (Ir-) Reproduciblity

Another process which causes another layer of uncertainty is the compilation of the source code. Therefore many attempts have been made to support a reproducible build for a specific program, in order to achieve a deterministic environment. One of the main reason to have a reproducible build environment, is to improve the debugging system. Because if the program code is guaranteed to be deterministic, it is possible to reproduce an error on another system, but also to check for bit-reproducibility. As for example in the ICON model (see [Rei15] for more details). Additionally having a deterministic build also provides a way to compute a hash for an executable to confirm its identity. In order to achieve a reproducible build, it is necessary to overcome multiple problems as described in [Lun15] (see list 1).

- Timestamps / Timezone
- Local
- File- order / paths
- UUID (Universally Unique Identifier)
- CPU-specific compilation
- Version information
- Environmental variables
- Compiler search paths

List 1: Common problems in a reproducible build according to [Lun15]

It is fairly obvious that timestamp / timezone, locals and version information which get compiled into the executable can't result in a reproducible

build. Because it is impossible to get the same starting time / date of the original building process without manipulating the system clock. Therefore by compiling them into an executable, it results in a different binary for different users at different conditions at compiletime. This is also the case for environmental variables such as the system name or special paths to libraries and programs. Those get then compiled into the final executable and result in different linked libraries and include directories.

Another common problem for reproducible builds is the compilation for a specific processor architectures, these can cause the generation of different instructions even for the same platform. While this can be prevented using the lowest instruction set for a platform that need to be supported, it is not possible for different processor architectures, such as x86 in contrast to x86-64 or ARMv8. Having such a restriction it may not always be achievable to create a reproducible build across different architectures, because it might be necessary to optimize the program for a specific platform to reduce runtime or memory consumption. But it should be noted, that for a specific architecture it should produce a reproducible build to achieve at least a reproducible build for the given architecture in order to get an overall reproduciblity.

Additionally to the CPU-specific compilation there are processes which results in a non reproducible build by default, as for example with Address Space Layout Randomization (ASLR) or older compilers such as the gcc-3.3.1 with the -fbranch-probabilities compiletime argument set[7]. When this argument is set, it calculates the branch properbilty with the help of a random number which is not deterministic and therefore lead to different compiler outputs. It is also to consider, that compilers are just software and consequently have bugs, which might cause different binary outputs, for a compilation process. This also applies to the other used software as mentioned previously. Also to mention is that some compilers, including the gcc, search in specific paths if they could not find a specified library, which can lead to a wrong linked executable, if there happen to be the required library in the compiler search path. While this isn't problematic for a build, which does not require to be reproducible, it can result in a non reproducible build, because there are a number of difficulties which arise with this. First if the requested library is binary compatible to older / newer libraries and couldn't be found, the compiler can just link an older / newer library which then result in a wrong version. Second, even if the library version is the same, it could be that it was linked with or without some extensions or with

[7] `https://gcc.gnu.org/onlinedocs/gcc-3.3.1/gcc/Optimize-Options.html`

CPU-specific instructions which also conclude in a irreproducible build as mentioned earlier. Luckily the wrong environmental variables and compiler search paths can be prevented by using a proper tool which sets the environmental variables before calling the compiler and jails them in a reproducible environment. A possibility for this might be a virtual machine or a container to avoid wrong search paths.

4 Best Practices

After listing these problems, which can cause an irreproducible build, I want to introduce some best practices as proposed by the National Research Council (US) Committee on Responsibilities of Authorship in the Biological Sciences [Cec03] and extended by Stodden et al. [SM14]. These can prevent some of the most common errors, which can occur in any kind of publication. First of all, [Cec03] state that everything which is related to a publication, should be published, to ensure that the publication can be rebuild from scratch. This includes the original measured data with measuring errors. The used algorithms, with there specification and the expected output for a given input, as well as the used programs, instruments and an instruction how the instruments got calibrated. Additionally the data and the source code should be documented extensively to avoid ambiguities between the publication and the data / source code. This includes a proper description, what the function should return for its input space and what happens in case of an error or wrong input data. If possible the source code should include some tests, which ensure the correct functionality of the functions and can be used as a reference how to use these.

When releasing data, it is important to have it as open as possible, to allow other researchers to work on it and check for reproducibiltiy. With a commercial or closed data set this might not be possible at all or only to few other researches, which therefore makes the publication less valuable. The same applies to the source code, a publication which doesn't include the source code or uses a proprietary program, can't be check against programming or logical errors, and is therefore not as valuable as an publication with open source code. This doesn't imply that the source code should be available for everyone. In fact it could still be proprietary and / or commercially distributed for other projects, but this shouldn't be the case for other non commercial research projects. As a way to achieve this, a dual licensing of the source code / data and / or signing a non-disclosure agreement to avoid that the original data / source code gets publicly available, approach can be

used. Also if the data or the source code is from a third party, it should be cited correctly to avoid plagiarism and to ensure that the data was not modified. While Stodden et al. don't forbid modification in general, it should at least be noted what data / source code has changed. This can be simplified with a source control tool to make this a simple process, where others can skip through the changes on the data / source code. An often overseen requirement for the data is to have a stable infrastructure, which can serve the data set over many years. It might take some time before other researchers want to reproduce the results or use the measured data for other research projects. Therefore having a good infrastructure is necessary to avoid the loss of valuable research data.

While it may not be always possible, to achieve a reproducible program, because this process might be to time / memory consuming in runtime as well as in development or simply to expensive, there is another common problem which arise with this. These problem got addressed by [Die12] and arise more often as research software get more available, which results in that these type of software get often used by users without that kind of an computer science background. Because many users without a computer science background have the expectation, that the program is infallible or that for a given input the program should return the same output. If this is not the case, it is then an often made assumption, that the program has a bug. It is therefore important to provide a simplified insight to the program with its data and source code, which allows users to understand the kind of problems, which occur with an irreprodcible program. This has also the advantage that errors in the publication can be found by potentially more people, especially in an interdisciplinary publication.

If the publication is not reproducible, this does not mean that it can't be used for research in general. [Die12] suggest for example that the results can be used as an approximation of the real value and to support a theory. Alternatively the program can be run multiple times and then get evaluated by standard statistical methods such as calculating the minimum / maximum, the mean value and the standard deviation, etc..

5 Conclusion

While it is not always possible to achieve reproducibility for a program it should be the gold standard for publications in general. Because they can affect everyone and many modern achievements such as weather / clima prediction or medicine rely on having reproducible research, as well as other

researchers which build their publications on existing ones. In addition to the initially described problem, that 47 of 53 studies couldn't be reproduced [BE12], there is a discrepancy between the definition of reproducibility between different research fields or even research groups. These need to change to a unambiguous clear definition on which all agree. This avoids that reproducible research is considered to be replicatable research by other research fields / groups and the other way around.

References

[BE12] C. Glenn Begley and Lee M. Ellis. "Drug development: Raise standards for preclinical cancer research". eng. In: *Nature* 483 (7391 2012). Journal Article, pp. 531–533. ISSN: 0028-0836. DOI: 10.1038/483531a. eprint: 22460880 (cit. on pp. 1, 11).

[Cec03] Thomas R. Cech. *Sharing Publication-Related Data and Materials*. Ed. by Sciences, National Research Council Committee on Responsibilities of Authorship in the Biological. 2003. DOI: \url{10.17226/10613} (cit. on p. 9).

[Die12] Kai Diethelm. "The Limits of Reproducibility in Numerical Simulation". In: *Computing in Science & Engineering* 14 (1 2012), pp. 64–72. ISSN: 1521-9615. DOI: 10.1109/MCSE.2011.21. (Visited on 06/07/2017) (cit. on pp. 5, 10).

[Dru09] Chris Drummond. *Replicability is not Reproducibility: Nor is it Good Science*. 2009. URL: http://www.csi.uottawa.ca/~cdrummon/pubs/ICMLws09.pdf (visited on 05/22/2017) (cit. on pp. 2, 4).

[LP15] Jeffrey T. Leek and Roger D. Peng. "Opinion: Reproducible research can still be wrong: Adopting a prevention approach". In: *Proceedings of the National Academy of Sciences* 112 (6 2015). aAssociate Professor of Biostatistics and Oncology and jtleek@jhu.edu bAssociate Professor of Biostatistics, Johns Hopkins University, Baltimore, MD, pp. 1645–1646. ISSN: 1091-6490. DOI: 10.1073/pnas.1421412111. URL: http://www.pnas.org/content/112/6/1645.full (cit. on pp. 1–3).

[Lun15] Lunar. *How to make your software build reproducibly. Provide a verifiable path from source to binary*. 2015. URL: https://reproducible.alioth.debian.org/presentations/2015-08-13-CCCamp15-outline.pdf (visited on 06/09/2017) (cit. on p. 7).

[Pen11] Roger D. Peng. “Reproducible Research in Computational Science”. eng. In: *Science (New York, N.y.)* 334 (6060 2011). Journal Article, pp. 1226–1227. ISSN: 1095-9203. DOI: 10.1126/science.1213847. eprint: 22144613 (cit. on pp. 2, 3).

[Rei15] D. Reinert, G. Zängl, Prill F., A. Fernandez del Rio, R. Potthast, D. Rieger, Schröter, J., Förstner, J., C. Walter, R. Ruhnke, and Vogel B. *Working with the ICON Model. Practical Exercises for NWP Model and ICON-ART*. 2015. URL: https://www.earthsystemcog.org/site_media/projects/dcmip-2016/ICON_tutorial.pdf (visited on 06/18/2017) (cit. on p. 7).

[Sch14] Jonathan W. Schooler. “Metascience could rescue the ‘replication crisis’”. In: *Nature News* 515 (7525 2014), p. 9. DOI: 10.1038/515009a (cit. on p. 1).

[SM14] Victoria Stodden and Sheila Miguez. “Best Practices for Computational Science: Software Infrastructure and Environments for Reproducible and Extensible Research”. en. In: *Journal of Open Research Software* 2 (1 2014). vcs@stodden.net Columbia University shekay@gmail.com. ISSN: 2049-9647. DOI: 10.5334/jors.ay. URL: https://openresearchsoftware.metajnl.com/articles/10.5334/jors.ay/ (cit. on p. 9).

[Son07] Sören Sonnenburg, Mikio L. Braun, Cheng Soon Ong, Samy Bengio, Leon Bottou, Geoffrey Holmes, Yann LeCun, Klaus-Robert Müller, Fernando Pereira, Carl Edward Rasmussen, Gunnar Rätsch, Bernhard Schölkopf, Alexander Smola, Pascal Vincent, Jason Weston, and Robert Williamson. “The Need for Open Source Software in Machine Learning”. In: *Journal of Machine Learning Research* 8 (Oct 2007), pp. 2443–2466. ISSN: ISSN 1533-7928. URL: http://www.jmlr.org/papers/volume8/sonnenburg07a/sonnenburg07a.pdf (cit. on p. 2).

[ZH09] Yong-Kang Zhu and Wayne B. Hayes. “Correct Rounding and a Hybrid Approach to Exact Floating-Point Summation”. In: *SIAM Journal on Scientific Computing* 31 (4 2009), pp. 2981–3001. ISSN: 1064-8275. DOI: 10.1137/070710020 (cit. on p. 5).