Experimenting with reproducibility in bioinformatics

Yang-Min KIM [1,2,3,4,*], Jean-Baptiste POLINE[5] and Guillaume DUMAS [1,2,3,4]

[1]Institut Pasteur, Human Genetics and Cognitive Functions Unit, Paris, France, [2]CNRS UMR 3571 Genes, Synapses and Cognition, Institut Pasteur, Paris, France, [3]University Paris Diderot, Sorbonne Paris Cité, Paris, France, [4]Centre de Bioinformatique, Biostatistique et Biologie Intégrative (C3BI, USR 3756 Institut Pasteur and CNRS), Paris, France, [5]Henry H. Wheeler Jr. Brain Imaging Center, Helen Wills Neuroscience Institute, University of California, Berkeley, California, USA

*To whom correspondence should be addressed.

Abstract

Reproducibility or replication has been shown to be limited in many scientific fields. This question is a fundamental tenet of the scientific activity, but the related issues of reusability of scientific data are poorly documented. Here, we present a case study of our attempt to reproduce a bioinformatics method and illustrate the challenges to use a published method for which code and data were available. From this example, we address the difficulties that pave the way towards reproducibility and propose some recommendations to the research community to improve the reusability of the data.

Availability and implementation: last version of *StratiPy* (Python) is available at GitHub: https://github.com/GHFC/stratipy

Contact: yang-min.kim@pasteur.fr

1 Introduction

The collective endeavour of science depends on researchers being able to replicate the work of others. In a recent interview with 1,576 researchers, 70% of them admitted having difficulty in reproducing experiments proposed by other scientists (Baker, 2016). For 50%, this reproducibility issue even concerns with their own experiments. Despite the growing attention on the replication crisis in science, this controversial subject is far from being new: already in the 17th century, scientists criticized the air pump invented by physicist Robert Boyle because it was too complicated and expensive to build (Shapin and Schaffer, 2011).

While **reproducibility** – here defined as being able to obtain the results using the same software and same data – or **replicability** – here defined as obtaining the same results with different data and software (Peng, 2011) – are key to the scientific progress, it seems that researchers are in general not considering these as priorities. Indeed, it takes great efforts and competence to overcome all the obstacles on the path to replication. The process is costly in resources, both in time and funding. In computational science, there are also many technical barriers ranging from unavailable data to hardware infrastructure. Even when authors provide data and code, the outcome can vary either marginally or fundamentally (Herndon *et al.*, 2014). Tackling irreproducibility in bioinformatics thus requires considerable effort beyond code and data availability (Fig 1). Such effort is nevertheless necessary to increase the robustness of the literature and efficiency of the scientific research process. Indeed, behind reproducibility hides re-usability.

In this case study, we focus on reproducing a promising bioinformatics method (Hofree *et al.*, 2013) and identify and document different issues related to the reproducibility process. First, we tried to re-run the analysis with the code and data provided by the authors. Second, we reimplemented the method in Python to avoid dependency on a MATLAB licence and ease the execution of the code on HPCC (High-Performance Computing Cluster). Third, we assessed reusability of our reimplementation and the quality of our documentation. Then, we experimented with our own software and tested how easy it would be to start from our implementation to reproduce the results, hence attempting to estimate the reproducibility of reproducibility. Finally, in a second part, we propose solutions from this case study and other observations to improve reproducibility and research efficiency at the individual and collective level.

2 Reproducibility in bioinformatics: a case study

2.1 From MATLAB to MATLAB: OS and Environment

Our team studies Autism Spectrum Disorders (ASD), a group of neurodevelopmental disorders well known for its heterogeneity. One of the current challenges of our research is to uncover homogeneous subgroups of patients (i.e. stratification) with more precise clinical outcomes, improving their prognosis and treatment (Bourgeron, 2015; Loth *et al.*, 2016). An interesting stratification method was recently proposed in the field of cancer research (Hofree *et al.*, 2013), where the authors proposed to combine genetic profiles of patients tumors with protein-protein interaction networks to uncover meaningful homogeneous subgroups, a method called Network Based Stratification (NBS).

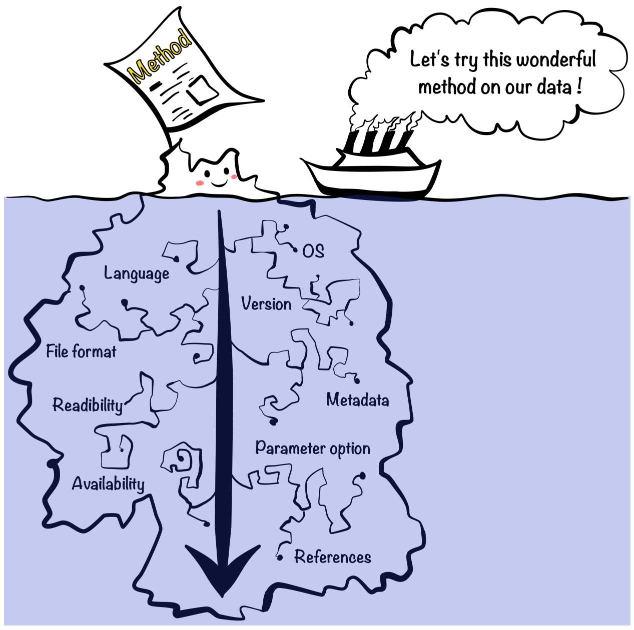

Fig. 1. Hidden reproducibility issues like underwater iceberg. Scientific journals readers have the impression that they can almost see the full work of method. But in reality, articles do not take into account adjustment and configuration for significant replication in most cases. Therefore, there is a significant gap between apparent executable work (i.e. above water portion of iceberg) and necessary effort in practice (i.e. full iceberg).

Before using this NBS method on our data, we studied the method by reproducing results from the original study. We are very grateful to the main authors who kindly provided online all the related data and code, and gave us invaluable input upon request. The authors of this study thus should not be blamed for the difficulty that we experienced in attempting to reproduce and replicate their study, as they did more to help replicate their results than is generally done. Despite their help we experienced a number of difficulties that we document here, hoping that this report will help future researchers to improve the reproducibility of results and reusability of research products.

The first step of our project was to execute the original method code with the given data. The programming code was written in MATLAB, an interpreted language originally developed for linear algebra computations which is easier and faster to write as well as more readable than compiled language such as C, making our reproducibility attempt easier. To improve execution speed, the original authors used a library for MATLAB using executable compiled code MEX file) callable from MATLAB: MTIMESX (Tursa, 2009), a library with compiled code allowing acceleration of large matrix multiplication. MEX files however are specific to the architecture and have to be recompiled for each Operating System (OS). The original MEX file was initially developed for Linux. Since our lab was using Mac OS X Sierra, the compilation of this MEX file into a mac64 binary required a new version of MTIMESX. It was also necessary to install and to configure properly OpenMP (http://www.openmp.org/specifications/), a development library for parallel computing. After this, the original MATLAB code was successfully run in our environment.

These issues are classic, but may not be overcome by researchers with little experience in compilation or installation issues. For these reasons alone, many individuals may turn down the opportunity of reusing code.

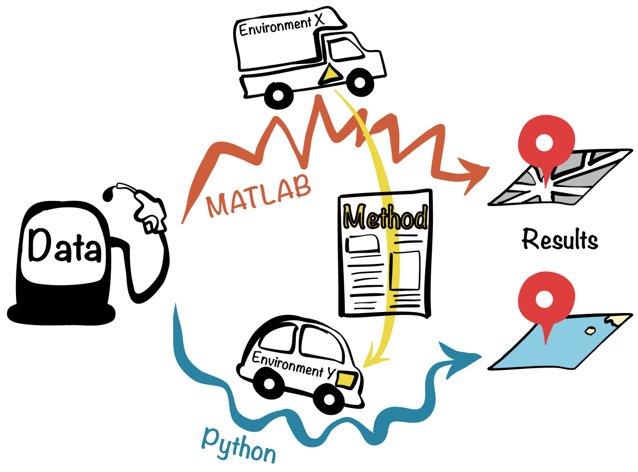

Fig. 2. Analogy between irreproducibility and road transport. The aim is to achieve same output (i.e. to reach the same location) using published methods (i.e. engine). Despite the same input data (i.e. gasoline), we obtained different results due to different programming languages —e.g. MATLAB and Python— (i.e. different roadways) and environments (i.e. different vehicles).

The next part will focus on code re-implementation, a procedure, which can help understanding the method, but can be even more costly.

2.2 From MATLAB to Python: Language and Organization

To fully master the method, adapt it to our data, and ease its re-use, we developed a complete open source toolkit of genomic stratification in Python. Python is also an interpreted programming language, but contrary to MATLAB is free of use and has a GPL-compatible license (https://docs.python.org/3/license.html). This is particularly interesting for both replicability and scalability. Re-coding in another language in a different environment will lead to be some unavoidable problems such as random initialization, and variation in low level libraries (e.g. glibc): it is likely that the outcomes will vary even if the same algorithm is implemented. In addition, we rely on Python packages to perform visualization or linear algebra computations (e.g. Matplotlib, SciPy, NumPy), and results may depend on these packages versions. Python is currently in a transitional period between two major versions 2 and 3. We chose to write the code in Python 3, which is the current recommendation.

2.2.1 Metadata and File formats

Even if the original code could be run, we had to handle several file formats to check and understand the structure of the original data. For instance the data on patients with cancer data was provided by The Cancer Genome (TCGA, https://cancergenome.nih.gov) and made available in a MATLAB *.mat* file format. Thanks to SciPy, Python can load all versions through v7.2 MATLAB files. To read v7.3 *.mat* files, we however needed an HDF5 Python library. Moreover, the original authors had denoted download dates of patients' data of TCGA, thereby clarifying source of data. But in the absence of structural metadata, it was not always obvious how to interpret patients' dataset variables (e.g. patient ID, gene ID, phenotype).

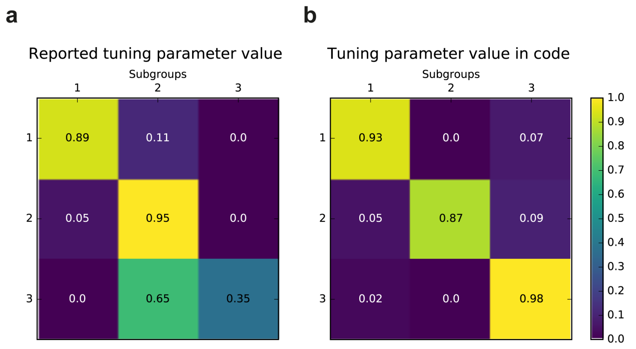

Fig. 3. Normalized confusion matrices between original and replicated results. Before (a) and after (b) applying appropriate value of graph regularization factor on NBS method. Each row or column corresponds to a subgroup of patients (here three subgroups). The diagonal elements show the frequency of correct classifications for each subgroup: a high value indicates a correct prediction.

2.2.2 Codes and parameters

Once the environment and file format issues were resolved, the code was finally executable with genetic data. Unfortunately, several attempts produced error messages. Alternatively, “unexpected” results were obtained (Fig 2): e.g. during the application of hierarchical clustering, we used the clustering tools of SciPy (Eads, 2007) (https://docs.scipy.org/doc/scipy/reference/cluster.hierarchy.html). Both SciPy and MATLAB (MathWorks, http://www.mathworks.com) functions offer seven linkage methods, however, SciPy’s default option (single method) differs from MATLAB’s default option (UPGMA method), which was used in the original study. Another key example is the value of one of the most important parameters of the method, the graph regulator factor, which was not clarified in the original paper. We believed that this factor had a constant value of 1.0 until we found in the code that during iterations, its value was changing and converged to a high optimal value (~1800). Therefore, we obtained different results from the original NBS at the beginning (Fig 3). We observed heterogeneous subgroups instead of obtaining homogeneous clusters. No or little explanation on the parameter choices can explain variability in the results as we explored the possible parameters. Moreover, while attempting the original code to understand the causes of the errors, we realized that some part of the code were not run anymore (e.g. discarded work, remaining traces of debugging) which made the attempt to understand the implementation harder.

2.2.3 iPython/Jupyter

During the re-cording process, we used an enhanced Python interpreter to debug: IPython, an interactive shell supporting both Python 2 and 3. Since the dataset is large and the execution takes a significant amount of time, we used IPython to re-run interactively some sub-sections of the script, which is one of the most helpful features. IPython can be integrated in the web interface Jupyter Notebook, offering an advanced structure for mixing code and documentation. For instance, verifying intermediate results by plotting helped us to better understand the original code. While the IPython notebook was therefore initially convenient, it does not scale well and is not well adapted to versioning. However, ability of mixing code with document text is very useful for tutorials: a user of the code can read documentation (docstring), text explanations, and see how to run the code, explore parameters and visualize results in the browser.

2.3 From Python to Python: Replication of Replication

Besides IPython, we used versioning tools like the git code version control system (VCS) to document the development of our Python code. Git is arguably one of the most powerful VCS, allowing easy development of branches and helping us to work together as a distributed team (Paris, Berkeley) on the same project. This project, *StratiPy*, is hosted on GitHub, a web-based Git repository hosting service (https://github.com/GHFC/stratipy). While the original code was not available on GitHub, the main authors shared their code on a website. This should be sufficient for our purpose, but makes it less easy to collaborate on code. While working on our GitHub repository, several researchers from all over the world contacted us about our reproducibility experiment. Not only GitHub supports a better organization of projects, it also facilitates the collaboration of open-source software projects, thanks to several social network functions. We tried to comply with open source coding standards and to learn how to efficiently use Git and GitHub. Both required considerable efforts on the short-term but brought clear benefits on the long-term, especially regarding collaboration and debugging.

We then attempted to re-run and reproduce the results we obtained on another platform. While the Python code was developed under Mac OS X Sierra (10.12) we used an Ubuntu 16.04.1 (Xenial) computer to test the Python implementation. Some additional issues emerged. First, our initial documentation was not complete enough to know which packages were required and how to launch the code. Second, the code was very slow to the extent that it was impractical to run it on a laptop because the Numpy package had not been compiled with BLAS (Basic Linear Algebra Subprograms), low-level routines performing basic vector and matrix operations. Last, there was (initially) no easy way to check whether the results obtained on a different architecture were the expected ones. We added documentation and tests on the results files md5sum to solve this. To summarize, although the re-use and reproducibility of the results of the developed package were improved, these were far from being optimal.

3 Potential solutions: from local to global

3.1 Act locally: simple practices and available tools

Given the observed difficulties, we draw some conclusions on this reproducibility case study experiment and suggest some practices and tools.

3.1.1 Environment

Container technologies such as Docker and Vagrant are becoming a standard solution to installation issues. These rely however on competencies that we think few biologists possess today. Also, while the container will encapsulate everything needed for the software execution, it is hard to develop in a container, limiting the reusability of the code.

3.1.2 Metadata

Standard metadata are vital for an efficient documentation of both data and software. In our example, we still lack the standard lexicon to document the data as well as documenting the software: e.g. using HDF5 file instead of *.mat* file is more suitable to store patient's’ data. We however aim to follow the recommendations by Stodden *et al.* (2016): *“Software metadata should include, at a minimum, the title, authors, version, language, license, Uniform Resource Identifier/DOI, software descrip-*

tion (including purpose, inputs, outputs, dependencies), and execution requirements." The more comprehensive is the metadata description, the more likely the re-use will be both efficient and appropriate.

3.1.3 Write readable code

Anyone who has spent time to understand someone else's code would advise some simple basic rules to help make the code readable and understandable.

First, the structure of the program should be clear and easily accessible. Second, good concise code documentation and naming convention will help readability. Third, the code should not contain left-overs of previously tested solutions. When a solution takes a long time to compute, an option to store it locally can be proposed. Nevertheless, the code to compute, this variable should be given in any case (e.g. inverse of large matrix). Using standard coding and documentation conventions (e.g. PEP 8 and PEP 257 in Python, https://www.python.org/dev/peps) with detailed comments and references of papers makes the code more accessible. When an algorithm from another paper is used, any modification should be explained and discussed in the paper as well as in the code. All these remarks are not necessarily obvious especially if the developer is working on her/his own, and to some extent "writes for her/himself". We advocate for researchers to write code "for their colleagues", hence, the opinion and notice of co-working or partner laboratories should be very helpful. Furthermore, the collaboration between researchers working on different environments can more easily isolate reproducibility problems. In the future, journals may consider review of code as part of the standard review process.

3.1.4 Test the code

To check if the code is yielding a correct answer, software developers associate test suites (unit tests or integration tests) with their software. While we developed only a few tests in this project, we realize that this has a number of advantages, such as checking if the software installation seems correct, check if updates in the operating system impact the results, etc. This does not in general validate the method, but at least provides a basic check. In our case, we propose to check for the integrity of the data and for the results of some key processing.

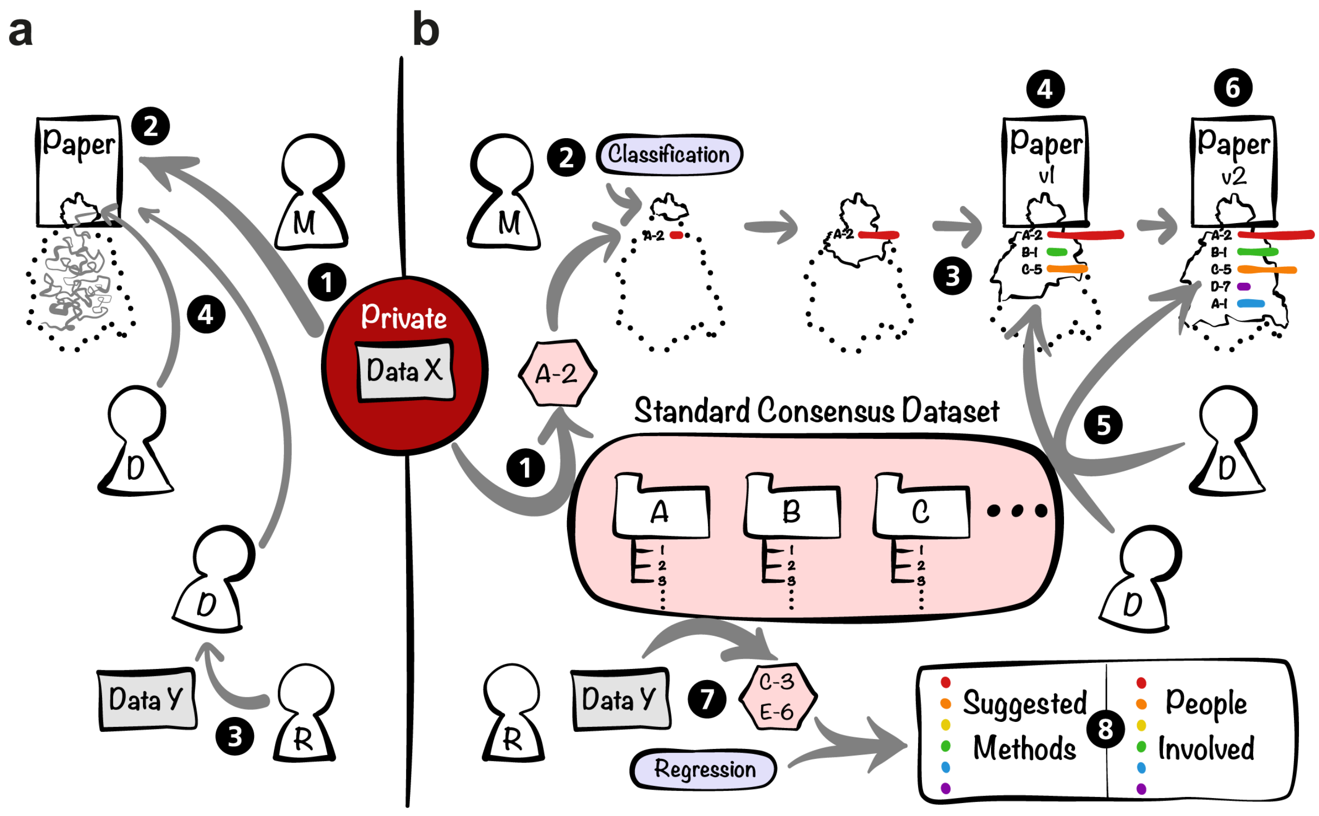

Figure 4: Working principles of Workflow system with private data. Figure 4a shows a classical workflow: (a.1) Methodologists (M) take private data; (a.2) M publish their method and corresponding outputs/results; (a.3) Scientists (requesters = R) (e.g. biologists, doctors) having their own data ask developers (D) (e.g. bioinformaticians) for handling them; (a.4) D find a relevant paper and will be lost in the labyrinth of reproducibility. Figure 4b shows workflow with standard consensus dataset: (b.1) If Methodologists (M) work with their own data, they must identify corresponding standard data tag(s) (e.g. A-2). They can use only standard data at the beginning of the project. If there is no appropriate data, they have to suggest a new data standard; (b.2) M choose a method category (e.g. "Classification"); (b.3) Reproducibility profile with several standard data is progressively built with method upgrade. Color corresponds to method category depending score and bar length corresponds to progression of replication test. Programming code with guide should be accessible although M do not publish; (b.4) Publication of the first version method by M; (b.5) Developers (D) can test proposed method with other data standards and thus participate to enhancement of the reproducibility profile. D then obtain automatically credit for this activity of validation; (b.6) Thanks to the collective work on testing, the method could be optimized and M can upgrade their initial paper (versioning); (b.7) Requesters (R) can now input corresponding tags (e.g. C-3 and E-6) of their data and search available methods in the category of their problem (e.g. "Regression"); (b.8) Following the request, a list of suggested methods and people involved (M, D) is obtained.

3.2 Think globally: from education to community standards

3.2.1 Training the new generation of scientists to digital tools and practices

Unlike theoretical and academic courses and projects, software testing systems are well developed in industry. For a student, discovering and learning this core system of reproducibility, possibly during an internship in cooperation with industry, is a great opportunity for her/his future. Furthermore, as Internet applications in science are growing, networks of scientists and developers are forming and provide learning opportunities on the development practices. For instance, software developers have recently adopted "agile" practices and fast prototyping, test based development, etc. Some of these ideas and practices can —and should— be adapted to scientific software development.

The training in coding is still too limited for biologists. Often, it is self-training, from searching answers on Stack Overflow or equivalent. Despite efforts by organizations such as software (https://software-carpentry.org) or data carpentry (http://www.datacarpentry.org) and the growing demand for 'data scientists' in life science, university training on coding practices is not enough generalized. The difficulty to access and understand code may lead to applying code blindly without checking the validity of the results: often, scientists may prefer to believe that the results are correct because of the time that would be needed to check the validity of the results. Mastering a package such that results are truly understood can take a long time, as it was the case in our experiment.

3.2.2 Standard consensus dataset and workflow system

We propose here that bioinformatics methods publications are systematically accompanied with a test dataset, code source and some basic tests. As the method is tested on new datasets, the number of tests of the method would increase in number and cover a wider range of applications. We give a first example with our NBS re-implementation. We develop below how this could generalize and what would be the benefit for the scientific community. In a sense, we propose to use the software development test framework idea but apply it to the scientific context.

A schematic overview of workflow system is shown in Figure 4. The core of this system would be a standard consensus dataset used to validate methods. Data could be classified in general categories such as binary, text, image (A, B, C in Fig 4b), and with sub-categories to introduce criteria such as size, quantitative/qualitative, discrete/continuous using a tagging system (e.g. A-2, B-1, C-5 in Fig 4b). Dataset could be issued from simulations or from acquisition, and would validate a method on a particular component. This workflow system will help scientists that cannot release their data because of privacy issues (although these can often be overcome) but also give access to data and tests to a wide community.

Roughly, we divide those who interact with scientific software or analysis code in three categories. First, the methodologists who propose a method and need to verify its validity and usefulness with public and/or their own - often private – data ("M" in Fig 4). Second, the developers or engineers who need to test and evaluate the proposed methods with other data ("D" in Fig 4). Third, expert and non-expert users who need to have a good understanding of the results obtained on various types of data ("R" as requester in Fig 4).

Each method belongs to a general category of methods (e.g. classification, regression) and could have a reproducibility profile, which will progressively be built by methodologists and developers (Fig 4b.3, 4b.5). The information of which method does or does not work with a standard/public data is a crucial information for the scientist. Developers who test and approve reproducibility on original or new data could be credited and recognized by the scientific and developer communities (i.e. Stack Overflow, GitHub). When scientists search about the appropriate methods according to their type of data (Fig 4b.7), they would also be interested in obtaining information about the people who invested heavily in test and optimization of suggested methods (Fig 4b.8). This workflow system could facilitate the gathering of diverse users of the science community.

4 Conclusion and perspective

Across the scientific fields, the reproducibility issue is seen as a growing concern. Before reusing a published method, we attempted to reproduce the initial results and recoded the method to have a deep understanding of it. The investment in time to verify a previously published method can be more important than the work needed to publish a new paper. Despite the willingness of the authors to share their tool and help us in our work, we have faced reproducibility problems due to compatibility between environments, programming languages and software versions, choice of parameters, etc. In addition to individual effort to write well documented and readable code, we recommend to use online repositories and tools to help other scientists in their exploration of the method: Docker for environment standardization, GitHub for code version management, and Jupyter notebooks for demonstration and tutorial. At the community level, we should enhance the cooperation between academic education and industry to foster a new generation of well-trained scientists in software development. We propose a workflow system where the community uses standard datasets to validate tools. The proposed method success on data profile will be evaluated continuously with new datasets. Eventually, data and software can be versioned and cited to give credit to the individuals who have contributed to these building blocks of Science. This workflow is not merely a reproducibility validation tool, it is an attempt to make research product more reusable by the community using an online platform, beyond the publication process. Some top-down initiatives already provide some incentives for such a process i.e. Horizon 2020 (H2020, https://ec.europa.eu/research/press/2016/pdf/opendata-infographic_072016.pdf) project of the European Commission (EC) mandates open access of research data, while respecting security and liability. H2020 supports OpenAIRE (https://www.openaire.eu/edocman?id=749&task=document.viewdoc), a technical infrastructure of the open access, which allows the interconnection between projects, publications, datasets, and author information across Europe. Thanks to common guidelines, OpenAIRE interoperates with other web-based *generalist* scientific data repositories such as Zenodo, hosted by CERN, which allows combining data and GitHub repository using digital object identifiers (DOI). The Open Science Framework also hosts data and software for a given project (Foster and Deardorff, 2017). Respecting standard guidelines to transparently communicate the scientific work is a key step towards tackling irreproducibility and insures a robust scientific endeavor.

References

Baker,M. (2016) 1,500 scientists lift the lid on reproducibility. *Nat. News*, **533**, 452.

Bourgeron,T. (2015) From the genetic architecture to synaptic plasticity in autism spectrum disorder. *Nat. Rev. Neurosci.*, **16**, 551–563.

Eads (2007) Hierarchical clustering (scipy.cluster.hierarchy) — SciPy v0.19.0 Reference Guide.

Foster,E.D. and Deardorff,A. (2017) Open Science Framework (OSF). *J. Med. Libr. Assoc. JMLA*, **105**, 203–206.

Herndon,T. *et al.* (2014) Does high public debt consistently stifle economic growth? A critique of Reinhart and Rogoff. *Camb. J. Econ.*, **38**, 257–279.

Hofree,M. *et al.* (2013) Network-based stratification of tumor mutations. *Nat. Methods*, **10**.

Loth,E. *et al.* (2016) Identification and validation of biomarkers for autism spectrum disorders. *Nat. Rev. Drug Discov.*, **15**, 70–73.

Peng,R.D. (2011) Reproducible research in computational science. *Science*, **334**, 1226–1227.

Shapin,S. and Schaffer,S. (2011) Leviathan and the Air-Pump: Hobbes, Boyle, and the Experimental Life (New in Paper) Princeton University Press.

Stodden,V. *et al.* (2016) Enhancing reproducibility for computational methods. *Science*, **354**, 1240–1241.

Tursa (2009) MTIMESX - Fast Matrix Multiply with Multi-Dimensional Support - File Exchange - MATLAB Central.