Mathematics

http://www.aimspress.com/journal/Math

AIMS Mathematics, 1(3): 261-281
DOI:10.3934/Math.2016.3.261
Received: 27 June 2016
Accepted: 1 September 2016
Published: 28 September 2016

*Review*

# Best practices for replicability, reproducibility and reusability of computer-based experiments exemplified by model reduction software

**Jörg Fehr**[1], **Jan Heiland**[2], **Christian Himpe**[3,*], **and Jens Saak**[2]

[1] Institute of Engineering and Computational Mechanics at the University of Stuttgart, Pfaffenwaldring 9, D-70569 Stuttgart, Germany

[2] Computational Methods in Systems and Control Theory Group at the Max Planck Institute for Dynamics of Complex Technical Systems, Sandtorstraße 1, D-39106 Magdeburg, Germany

[3] Institute for Computational and Applied Mathematics at the University of Münster, Einsteinstrasse 62, D-48149 Münster, Germany

[*] **Correspondence:** Email: christian.himpe@uni-muenster.de

**Abstract:** Over the recent years the importance of numerical experiments has gradually been more recognized. Nonetheless, sufficient documentation of how computational results have been obtained is often not available. Especially in the scientific computing and applied mathematics domain this is crucial, since numerical experiments are often employed to verify the proposed hypothesis in a publication. This work aims to propose standards and best practices for the setup and publication of numerical experiments. Naturally, this amounts to a guideline for development, maintenance, and publication of numerical research software. Such a primer will enable the replicability and reproducibility of computer-based experiments or published results and also promote the reusability of the associated software.

**Keywords:** Replicability; Reproducibility; Reusability; Repeatability; Recomputability; Computer-Based Experiments

## 1. Introduction

Through the years, the value and the extent of Computer-Based Experiments (CBEx) in publications in the fields of applied mathematics, numerical analysis, and scientific computing has risen significantly. For an illustration of this general observation, one may compare the following three scientific articles from 1971, 1986, and 2010, which introduced nowadays commonly applied numerical methods. In *Nitsche*'s 1971 paper [46] on a new variational approach to elliptic PDEs with non-homogeneous Dirichlet conditions, there is no numerical experiment reported. Then, in the important

work [53] by *Saad* and *Schultz* on the *GMRES* algorithm from 1986, two out of 14 pages are devoted to numerical experiments. Finally, the paper [10] on *DEIM* by *Chataranbutat* and *Sorensen* in 2010, consists of more than 30% of numerical examples or reasoning based on numerical experiments.

However, the high standards on analytical findings, namely the requirement of a concise, comprehensible and traceable derivation and documentation, seems not equally adapted to numerical experiments and results in the scientific literature, cf. *LeVeque*'s article on *Top Ten Reasons to Not Share Your Code (and why you should anyway)* [38].

In the same time, the nature of CBEx has changed from a rather deterministic mathematical exercise on a computer towards a scientific experiment [54] with inevitable uncertainties. Thus, a CBEx should be seen in analogy with experiments from natural sciences with the numerical result corresponding to the observations and the hard- and software corresponding to the methods that were used to obtain the observations. And, like experiments in natural sciences, a CBEx should be designed such that it is robust against uncertainties, i.e., such that it can be replicated to give the same results.

Once a replicable experiment has been established, the question of *reproducibility* arises. This principle seems broadly accepted since long, and it has found its formulation in *Popper*'s words: "I refuse to accept the view that there are statements in science which we have, resignedly, to accept as true merely because it does not seem possible, for logical reasons, to test them.", cf. [50, Ch. 1.8]. Note that the demand of testability of the hypothesis does not include a *truth* value as it is implicated by the reproducibility of an experiment. Nevertheless, as *Popper* states, an unreproducible singular discovery would not be published by a researcher.

Reproducibility is commonly accepted as a necessary condition for good scientific practice, and it's absence in some prominent works but also in a statistically significant number of journal publications that has been detected in recent years in, e.g., medicine [15], psychology [31], and computer science [11] has shaped the term of the *reproducibility crisis* that has been broadly covered in scientific, public, and social media [39,7].

Finally, software can be duplicated and dissected so that, besides the results, also parts of the methods can serve as the base of new experiments, which is meant by *reusability*.

In this work, we adapt notions related to replicability, reproducibility and reusability (*RRR*) as they are relevant for CBEx from first principles. We describe conditions for their implementation in research and publications that are general enough to meet particular needs of projects as well as habits of the researchers. To find the balance between a reliable framework and openness towards common practices, we add sections with concrete suggestions – a *best practice guide*. We will refrain from addressing details on code and data layout or licensing and associated copyright issues; work on these topics can be found for example in [59] and [60] respectively. Also, our work is about how CBEx are conducted and documented which can be seen independent from approaches that try to validate numerical results like the notion of *Verification and Validation* [4].

Overall, this work aims to: *Make CBEx replicable in its basic definition and use the potential of software to enable easy reproducibility and even reusability.*

### *1.1. Prior Work and State of the Discourse*

The discrepancy between the potential of CBEx to be easily made RRR and the widespread lack of RRR in CBEx in the scientific literature has stimulated various initiatives and theoretical work on the implementation of RRR principles in scientific computing. We list but a few of the most recent

publications:

The discussion on opening scientific source codes has been more noticeable in the recent years. For example in *Nature*, arguments against open source are refuted [5], more accurate results are predicted [42], partial opened codes are discussed [25], and a code availability section is suggested [44,45]. In *Science* not only the opening and review of research codes is discussed [33,56,32] but it is required by the editorial policies [55] that: "*All computer codes involved in the creation or analysis of data must also be available to any reader of Science*". Mathematical organizations are also discussing open scientific codes; examples are: *AMS* on the maintainability and necessity of open code accompanying publications [34], *ACM* on advantages and disadvantages of releasing scientific codes [41], and *SIAM* on a publication of codes by default and attributable credit [3]. Furthermore, dedicated workshops were held, such as the *ICERM* workshop [1,61,62] or *reproducibility@XSEDE* [30].

The general concept of reproducibility has been taken up in computer-based research in the 1990s [9]. The related notion *reproducible research* [17] refers to computational environments that allowed for simply transferring to and rerunning the experiments on different computers; see [40] for an example from the field of archeology.

Several publications describe abstract software engineering and collaborative development techniques. In [36] basic practices for scientific software development are distilled, while in [22] software management principles are explained. A set of rules, devised in [51], is concerned with the code development but also the user-developer interaction. And the best practices in [65,66] summarize code development fundamentals. General recommendations for reproducibility for CBEx are also given in [2]. Furthermore, the practical reproduction of research results themselves is discussed in [43].

Lastly, we note that various initiatives have been started to promote certain standards in CBEx. Foremost, the *Science Code Manifesto*[1] states five principles (Code, Copyright, Citation, Credit, Curation) for the handling of research software to improve its use in science. Also, the *Recomputation Manifesto*[2] [19] formulates rules to facilitate the repeatable realization of CBEx.

### *1.2. Outline*

This introductory discussion is followed by a more refined analysis of replicability, reproducibility and reusability in Section 2. In Section 3 a technique to document code availability is described. Section 4 summarizes high-level considerations to facilitate RRR, while a minimal documentation for scientific codes and research software is proposed in Section 5. Finally, a sample software project is presented to illustrate the practical implementation of the herein suggested best practices.

## 2. The Three "R"s of Open Science

In this section, taking up the ideas of [64,8], we give a definition of the frequently used terms **Replicability**, **Reproducibility**, and **Reusability** and discuss how these basic scientific principles apply for assessing scientific software.

The distinct notions of replicability and reproducibility are used to qualify research in all fields of science in which experiments play a role, cf., e.g. [63] with a background in biology, [47] from psychology, or [13,17] focusing on scientific computing. In short, replicability refers to a repetition of

[1] sciencecodemanifesto.org
[2] recomputation.org

the experiment with the same results by the same observers in the same environment; reproducibility refers to an independent repetition of the experiment and its outcomes in different circumstances. Reproducibility points to a certain reliability of both, the findings of the experiment and the procedure that was used to obtain the results [37]. Once reliability of a method is established, one can address reusability as the property that enables the use of the method for different setups and purposes. Note that these characteristics should be considered nested, which means reproducibility implies replicability and reusability requires reproducibility.

In what follows, we extend, specify, and adapt these general notions to the case of scientific software and numerical simulations.

### *2.1. Replicability*

The attribute **replicability** describes the ability to repeat a CBEx and to come to the same (in a numerical or statistical sense) results. Sometimes the equivalent term **repeatability** is used for this experimental property. Replicability requires some basic documentation on how to run the software (described in Section 4.5) to obtain replicable results.

Replicability, in turn, is a basic requirement of reliable software as well as of its result as it shows a certain robustness of the procedure against statistical influences and bias of the observer. Also, a replication can serve as a benchmark to which new methods can be compared as pointed out in [64].

### *2.2. Reproducibility*

In its simple definition, **reproducibility** of a CBEx means that it can be repeated by a different researcher in a different computational environment. This can be assured, first, through a documentation that provides enough mathematical and technical detail to set up the CBEx that will provide comparable results, including the software implementation of algorithms; second, through the distribution of a software capable of producing the results on a large variety of machines, or third, any combination of these two extrema – sufficient documentation and available software. If the CBEx depends on hardware, e.g., if runtime is measured, then for reproducibility the hardware needs to be available or sufficiently well documented.

### *2.3. Reusability*

In the sphere of CBEx, **reusability** refers to the possibility to reuse the software or parts thereof for different purposes, in different environments, and by researchers other than the original authors. In particular, reusability enables the utilization of the test setup or parts of it for other experiments or related applications. Although theoretically, any bit of a software can be reused for different purposes, here, reusability applies only for reproducible parts, since a building block of a CBEx that does not define reproducible or even replicable outcomes cannot be reused for a replicable or reproducible CBEx.

## 3. Code Availability Section

Even though availability of the source code associated to a CBEx is not a requirement for replicability and reproducibility (see Section 4), it is essential to open the CBEx to peer scrutiny and highly

recommended by the authors. The availability of the source code itself is necessary for reusability and unconditionally desirable for reproducibility. This section makes the case for a **Code Availability Section** as introduced by *Nature* [12,44,45]. Such a section should by default be included in any publication presenting numerical results like a "Materials and Methods" section in other sciences, and should state if the utilized code is available and if not for what reason, i.e. third-party licenses, non-disclosure agreements, trade secrets, or the thought of keeping competitive advantages.

Different code availability models exist, which will be listed and briefly commented on in the following.

**Open source code, published under a public license** This procedure is probably preferred by most scientists and for some people the only way to do proper science, compare, e.g., [25]. Referees and interested readers can check if the code fulfills the necessary requirements for reproducibility and they can modify and use the code for their own purpose. There are multiple possibilities to gain access to the code. Nowadays, a common and widely used procedure is the provisioning of source code via a publicly readable revision control repository located on a private server e.g. in a *GitLab*[3] instance or a third-party service provider, such as *GitHub*[4], or *Bitbucket*[5]. Alternatively, a download from a collection such as *Netlib*[6] can be provided. A shining example for best practice in the field of open source code in combination with reproducible experiments is the *Image Processing On Line* (IPol) Journal [26]. In this journal each article is supplemented with its source code, with an online demonstration facility and an archive of experiments. Furthermore, the text, as well as the source code, are peer-reviewed.

**Closed source, software available under a non-public license** This less desirable option gives readers and reviewers the opportunity to check, if the proposed numerical procedure or experiment work with their own data, given a license is available. Often, the source code is encoded or obfuscated to protect intellectual properties, which then allows a replication but not a comprehension of results. Matlab code, as an example of an interpreted language, can be encoded via the `pcode` command or compiled into a binary format. However, as stated in the documentation of *MATLAB* Version 2014b [24] "The `pcode` function obfuscates the code but does not encrypt it. While the content in a `.p` file is difficult to understand, it should not be considered secure." For programs written in a compiled language, such as `C++`, only executables or runtime libraries are provided. Hence, for trust reasons it is important, that the software has a-priori passed through a strictly documented verification & validation procedure. By providing and hosting the sources via a version control repository (see Section 4.6) it is possible to provide certain people, i.e. the reviewers, with access to the source code upon request. Alternatively, the source code may be provided directly to an eligible user via physical data volumes, or direct file transfers.

**Software as a Service (SaaS)** The availability of web access to computer programs or computer resources is an emerging strategy. This approach can also be used to enable interested users or reviewers

[3] `about.gitlab.com`
[4] `github.com`
[5] `bitbucket.org`
[6] `netlib.org`

to use the developed software as a service, e.g. to test if the program runs with their own, respectively modified input data. Therefore, SaaS offers many advantages such as a read-without-copying restriction for the source code, time-limited access for users, third-party software dependencies can be resolved, new licensing schemes and so on. It should be noted that, while SaaS enables the use of a CBEx, it does not allow a dissection at a source code level.

**Non-available code** The last and the most undesirable option is the non-availability option: Source code, computer program or required third party software is not available or purchasable to the interested reader. A review is hardly possible and the proposed numerical scheme or ideas need to be described in great detail, to enable reproducibility of the work.

A sample **Code Availability Section** is enclosed in Figure 1. The linked source code archive should ideally be uniquely identified by a Digital Object Identifier[7] (DOI) which can be obtained for software releases for example from *Zenodo*[8] for scientific codes. Alternatively, the source code can be enclosed in the supplemental materials or deposited at some stable location.

**Code Availability / Licensing Option**
The source code of the implementations used to compute the presented results can be obtained from:

`doi:???????/????????` and is authored by: X Y, A B

Please contact X Y for licensing information

**Figure 1. Sample Code Availability Section.**

Even though a simple statement on the (non-)availability of the source code does neither improve the review process nor the reproducibility (in the sense of Section 2.2), it can at least facilitate replicability through its assurance by the authors. Furthermore, it could be noted if the referees had access to the implementation during the peer review process.

Moreover, due to the important role of computational results, not only in numerical analysis but also in many other sciences, this measure contributes to the basic idea of verifiability in science. If the source code is made available, as a part of the publication, on the one hand, effort invested into an openly available software implementation is made visible and, on the other hand, compels authors to comment on means of the experimental setup. Lastly, a mandatory code availability section raises awareness for RRR.

## 4. Code Guidelines

In this section, based on the previous definitions of replicability, reproducibility, and reusability as well as prior work in other fields such as [27], guidelines for the design, documentation, or publication of CBEx and research software are summarized. The foundation for these guidelines is the interrelation

[7] `www.doi.org`
[8] `zenodo.org`

of RRR: reusability implies reproducibility which implies replicability; and mandatory requirements and optional recommendations are given. **Requirements** are limited to the minimal extent necessary while **recommendations** enable a practical and comfortable realization of the replication, reproduction, or reuse. The interdependence of the requirements and recommendations is to be understood as follows: A requirement for replicability is also a requirement for reproducibility and similarly, a requirement for reproducibility is also a requirement for reusability. Recommendations are optional but strongly encouraged, yet have no dependence on previous recommendations.

We will use the term "source code archive" to refer to the set of source code, build instructions (such as a `makefile`), configuration files and input data[9]. For a summary of the following guidelines see Figure 2.

### *4.1. Replicability Requirement: Basic Documentation*

A fundamental requirement for replicability is a **basic documentation**, which encompasses instructions on how to generate an executable binary program in case of a compiled language, and a description on how to run the program to obtain the results to be replicated (see also Section 5). This documentation is crucial to an experiment's replication as it defines the technical implementation and ensures the practical repetition of the experiment.

Often, the numerically computed results are further processed to facilitate interpretation, for example by a visualization. A documentation of the evaluation of these results, descriptively or algorithmically, is needed to allow for replication, not only of the computational results, but also of their evaluation.

### *4.2. Replicability Recommendation: Automation and Testing*

The **automation** of the experiment enables an easy and reliable check for replicability of a CBEx. This typically means that a single or multiple scripts automatically prepare and run the experiment as well as the post-processing of the results.

Replicability requires replicable behavior of all building blocks of the experiment, for which the setup of particular **tests** is recommended. Commonly, three categories of tests are considered: Unit tests, exam small sections of the source code; integration tests, check major components of the source code; and system tests, assess the whole project [6, Ch. 3]. Tests usually involve a comparison of the computed with analytical results, statistically significant sampling or the conformance to an accepted benchmark problem.

### *4.3. Reproducibility Requirement: Extensive Documentation*

To enable the reproducibility of a CBEx, a sufficiently detailed description of the algorithms, implementation, test setup, and parameters needs to be provided. Here, sufficiency is achieved if the documentation contains all information needed to setup and to run the experiment by a different researcher in a comparable environment. However, to reproduce a CBEx in a different environment, a documentation of the utilized hardware and software is also needed. An essential part of this environment documentation is the listing of other software packages required to perform the CBEx. Documenting these dependencies includes all software, which is not available in a commonly assumed

[9]The source code archive may also include resulting data sets from the authors experiments.

environment including employed variant and version and allows to set up the same or at least a similar software stack.

Depending on the programming language in which the considered CBEx is encoded, different types of dependencies arise. A compiled language requires a compiler and linked libraries to generate an executable file embodying the program computing the results. The variant of the compiler and its version as well as the variants of (statically and dynamically linked) libraries with their versions make up the associated dependencies. Furthermore, a build system, which organizes the compilation and linking may be used and constitutes a dependency. An interpreted language requires an interpreter, which parses and executes the source during runtime. In this case, typical dependencies are the variant of the interpreter in a specific version as well as depending toolboxes or modules with versions.

### *4.4. Reproducibility Recommendation: Availability*

The **availability** of the source code archive is highly recommended for reproducibility because of two main reasons. First, the code itself may serve as documentation of the experiment and is a scientific contribution in itself. Second, the code may be used to accomplish the actual reproduction.

Therefore, the availability of the source code archive from a stable location is vitally important. A location can be considered stable if its main purpose is storing data. This does not imply lasting availability, hence a second backup location is commendable.

The classic method of providing source code access is the bundling with the publication by including the source code archive as supplemental material. This affiliates the code with the publication and is conveniently obtainable together with the publication itself. Yet, a supplemental material section may not be available or only accept certain file types (with a maximum file size).

Recently, depots for scientific source code have been established. For example, *RunMyCode*[10] or *Research Compendia*[11] are services for storing source code archives and linking these to publications.

Alternatively, the source code archive can be published separately through platforms such as *Zenodo*[8] or *Figshare*[12]. An advantage of this method is the assignment of a digital object identifier (DOI) for such a software publications, which can then be stated in the Code Availability Section of the associated publication.

As for the dependencies, reproducibility is not inhibited by closed-source software. However, a statement on the applicability of an open-source variant, if available, of those dependencies is suggested. In any case, those components of the experiments that are not part of the source code, need to be documented as described in Section 4.3.

### *4.5. Reusability Requirement: Accessibility*

A CBEx is reusable if it is accessible in a related or even different context. **Accessibility** encompasses all means to (partially) apply the functionality of the original to another CBEx. The availability of source code fulfills the accessibility for reusability, but also access to a compiled executable and library or a remote service is sufficient to comply.

[10] `www.runmycode.org`
[11] `researchcompendia.org`
[12] `figshare.com`

*4.6. Reusability Recommendation: Modularity, Software Management & Licensing*

To be able to adapt a CBEx to differing environments and settings, the CBEx itself has to allow some parametrization to enable a certain configurability. Furthermore, **modularity**, the separation of experiment and method, enables the utilization of the method in other experiments or conducting the experiment with alternative methods. A more fine-grained modularization can allow, in addition, the exchange of components from the method or experiment such as numerical solvers or service libraries. Modularity necessitates a definition of interfaces which determine the communication between the interchangeable components. The documentation of such an interface is essential for it to fulfill its purpose and involves e.g. a description of protocols, variables, types and function signatures with their arguments and return values.

Source code usually undergoes some evolution over time during which errors are fixed, and new features are introduced. Hence, **software management** methods, such as version control, are recommended for the organization of this development process. A reusable software project is recommended to obey some versioning procedure. A version scheme allows a unique identification of different chronological stages of the project. Usually, such a version consists of at least two numbers delimited by a dot, describing the major and minor iteration of changes. More fine-grained versioning can be applied with further numbers. A release of a new version can be fixed by assigning a DOI. To record the evolution of the source code a version control system, such as *git*, *mercurial* or *bazaar*, is an important tool. A version control system tracks changes for each controlled file and allows a well-defined collaborative work on the (plain text) source files. The set of all files under version control makes a repository, a set of changes to a single or multiple files constitute a revision of the repository, and a set of revisions defines a new version. A history of the revisions can also augment the documentation of the CBEx if the changes are recorded with comprehensive descriptions.

A **license** assigned to the source code archive, which governs the rights and duties associated with its use and reuse as well as indicating copyrights, is practically necessary for reusability. If an open-source license is selected, certain characteristics should be considered: The license should be approved by the Open-Source-Initiative[13] and the Free-Software-Foundation[14] as well as being compatible with the GNU-General-Public-License[15]. Generally, a central requirement for scientific software should be an attribution clause requiring the future inclusion of the copyright information, which usually notes authors and contributors. Note, that a non-permissive license may inhibit the reusability of the software in non-open projects, cf. [58]. To select a license, the service *Choose-A-License*[16] can be of help, and for an explanation of the selected license, a service like *tl;dr Legal*[17] provides short summaries of the license's legal implications.

## 5. Basic Documentation

In terms of research software, it is important that the accompanying documentation enables usage and reproducibility of results. To this end, certain information on the tested hardware and software should be documented. Following, a basic form of documentation is proposed, which includes the

[13] opensource.org
[14] fsf.org
[15] opensource.org/licenses/gpl-license
[16] choosealicense.com
[17] tldrlegal.com

- **Replicability**

Required: Basic Documentation
Recommended: Automation & Testing

- **Reproducibility**

Required: Extensive Documentation
Recommended: Availability

- **Reusability**

Required: Accessibility
Recommended: Modularity, Software Management & Licensing

**Figure 2. Coding guidelines overview.**

essential information to facilitate RRR.

A simple form of documentation is providing basic information in plain text files. These should be sequential files containing only printable ASCII characters [28] and consequently using a US-ASCII file encoding. If it is necessary to also use non-ASCII characters, a modern encoding with good cross-platform support, like UTF-8, should be used. Recently, these text files have been decorated with *commonmark*[18] mark-down code[19], which rather improves readability then inhibiting it and are considered an unofficial standard due to the widespread use for example by *GitHub*[4]. Since typically scientific publications are composed in the English language, so should be these text file.

Certain default filenames are established to indicate the file's contents, such as `README`, `LICENSE`, `AUTHORS` and `CHANGELOG`. Additionally, further files of relevance to the academic environment have been suggested such as `CITATION` and `CODE`. This work proposes two more files, namely `RUNME` and `DEPENDENCIES` to facilitate replicability.

### *5.1. README*

The bare minimum of any code package, source code repository or source code archive should be a `README` file. To uniquely identify this text file it should state the name of the associated software project along with its version and the release date. Normally, also a brief description of the code functionality and its contents are expected.

Often, the `README` file also includes a manual for the compilation or installation of the project. In the case that these procedures are more elaborate, a separate `INSTALL` file can be used and referenced inside the `README`. The same holds for the authors and contributors to the project, which can be listed in the `README` or in an additional `AUTHORS` file. Relevant information for the `README` includes a project website, a (stable) download location, contact information and sample usage (for example referencing the `RUNME` file) of the associated software. Furthermore, the license and the `LICENSE`

[18] `commonmark.org`
[19] Usually indicated by the file extension `.md`.

file[20], a record of the history of changes in the CHANGELOG file, a set of frequently asked questions in a FAQ file and a documentation can be referenced.

In case the replicability of an experiment is targeted, the specific software stack and hardware environment should be documented, as well as all configurations, parameters and arguments defining the CBEx. For reproducibility, related publications should be cited, and for reusability, links to technical documentation, e.g. interfaces, or a version control repository could be listed. Generally, a README file can also act as a table of contents to the remaining files associated with the source code archive.

Preferably, the README presents the necessary information to start using the software in a quick and comprehensive way. Therefore, the general recommendation is to make it as detailed as necessary while at the same time keeping it as brief as possible. For in-depth discussions of the further details, a reference to the actual software documentation should be preferred.

### *5.2. RUNME*

To facilitate replicability, an additional file called RUNME is proposed in this work, that lists the steps required to replicate results. This can be an executable script file, which upon execution automatically performs all steps necessary to replicate the results of an associated publication. In case, multiple environments are supported, the respective environment can be highlighted by a file extension, for example RUNME.linux or RUNME.win. Alternatively, the RUNME file can describe these stages in pseudo-code or, in general, not machine readable language.

### *5.3. CITATION*

The concept of a CITATION file has first been used by the *R-Project* [52] and has also been adapted by *GNU Octave* [14]. This file contains information on how to cite the associated software project in other works. Besides a sample citation, a suggested BibTeX code is often provided in this file.

### *5.4. DEPENDENCIES*

Modern software stacks encompass multiple layers of intermediary software on which a project may depend upon. To be able to build and use a provided source code package such dependencies must be locally available. For projects with few dependencies, it is sufficient to list those in the README file, yet for projects with many dependencies it is suggested to include a DEPENDENCIES file that lists these necessary (third-party) software components including the required versions. Dependencies encompass, but are not limited to: runtime environments, libraries, toolboxes, modules, source code archives or executable files.

### *5.5. CODE*

The purpose of the CODE file is the listing of key meta-data on the associated software project. Initially, the idea of bundling code meta-data[21] was proposed in [57] and formalized in [35]. The main intended purpose of this proposal was the assignment of transitive credit in software stacks utilized for scientific work. In publications, about a software project this meta-data also helps as a unique

[20]The LICENSE file holds the full license text the copyright holders and the release year

[21]See also: [18] and schema.org/SoftwareApplication.

identification, as for example in the *SoftwareX* journal[22]. Another important reason for code meta-data is the classification and organization of scientific software, which facilitates reproducibility and reusability. This information could and should also be enclosed in the `README` file, yet the focused `CODE` file is machine-readable and allows automatically generated directories.

Various file formats to encode this meta-data are surmisable. Among others there are: `ini` (Initialization File), `xml` (Extensible Markup Language), `yaml` (YAML Ain't Markup Language) and `json` (Javascript Object Notation), of which the latter is suggested in [57,35]. Basic requirements for such a file are a plain text encoding and a human readable formatting. Additionally, a simple syntax[23] as well as the availability of parsing facilities should be considered. Due to its renownedness and easy readability for human and machine, the authors suggest to use the `ini` file format, as the more elaborate grammars of `xml`, `yaml` and `json` require sophisticated parsers.

There is no standard defining the `ini` format, yet its widespread use establishes a quasi-standard: Each line in an `ini` file holds a single key-value pair, which is delimited by a colon. The other formats also provide hierarchies for its components, which allow nesting of fields, for example grouping an author's properties under a common author key, but these hierarchies introduce an impediment for the automatic parsing of contents. To resolve the former example of multiple authors, in the case of the `ini` file, a comma separated list can be used as the value.

Due to the wide range of possible meta-data across the sciences utilizing software, no one-size-fits-all list of keywords is given, but a list of suggestions which applies to most research software projects.

- **name** The primary identifier of the software project.
- **shortname** An alias or the name of the main executable.
- **version** A unique state of the project, usually symbolized by numbers separated by decimal points indicating the major and minor revisions.
- **release-date** The date this version has been released in the ISO-8601 format: `YYYY-MM-DD` [29].
- **id** An identifier fixing a software release at a stable location.
- **id-type** The type of identifier used, e.g.: DOI.
- **authors** The list of authors.
- **orcids** The list of ORCID[24] identifiers corresponding to the list of authors.
- **associated** An identifier of an associated document, like a publications (DOI), book (ISBN) or documentation (URL).
- **topic** A basic categorization[25] of the project.
- **type** The type of software, for example a program, library, toolbox or module.
- **license** The license under which the software is released.
- **license-type** A distinction between open and propriety licenses.
- **repository** The link to the project's source code repository.
- **repository-type** The type of version control software of this repository.
- **languages** This field is supposed to contain a comma separated list of utilized programming languages in the software project. For larger projects, listing the major used languages will be sufficient. Since programming languages evolve over time, the versions or standards of the employed

[22] `www.journals.elsevier.com/softwarex`
[23] This is understood as a small set of rules.
[24] `orcid.org`
[25] e.g. MSC (`msc2010.org`), ACM (`www.acm.org/about/class`) or PACS (`www.aip.org/publishing/pacs`).

languages or dialects should also be provided.
- **dependencies** A list of software required to use the project, such as runtimes, libraries, toolboxes or modules.
- **systems** A list of compatible[26] operating systems or computational environments.
- **website** If the CBEx is part of an enclosing research software project and has a website, the URL (Uniform Resource Locator) can be provided in this field to guide users to the available resources.
- **keywords** A list of descriptive terms.

An example of such a code meta data `ini`-file, from `emgr` - empirical gramian framework [23], is shown in Figure 3.

```
name:  Empirical Gramian Framework
shortname:  emgr
version:  3.9
release-date:  2016-02-25
id:  10.5281/zenodo.46523
id-type:  doi
authors:  Christian Himpe
orcids:  0000-0003-2194-6754
topic:  Model Reduction
type:  Toolbox
license:  2-Clause BSD
license-type:  Open
repository:  github.com/gramian/emgr
repository-type:  git
languages:  Matlab
dependencies:  GNU Octave >= 3.8, MATLAB >= 2011b
systems:  Linux, Windows
website:  gramian.de
keywords:  empirical gramians, cross gramian, combined reduction
```

**Figure 3. Sample `CODE ini`-file for version 3.9 of the empirical gramian framework.**

### *5.6. Source Code File Headers*

Apart from the text files enclosed with the project, every source code file should state in its first lines, the so-called header:

1. the associated project,
2. the authors and contributors,
3. and the purpose of the file.

[26]It should be noted, that compatibility does not guarantee reproducible results; see for example [20].

This establishes the affiliation of this source file to the project. The header can optionally also include license and version information. Additionally, this file header can hold citations to works used to compose the following source code or keywords categorizing the contents.

## 6. A Practical Example

In this section, we discuss a very rudimentary and simple implementation of the iteratively corrected rational Krylov algorithm for $\mathcal{H}_2$ model reduction proposed by *Gugercin*, *Antoulas* and *Beattie* [21]. The implementation of the algorithm was made as an exercise in a lecture about model reduction. The common denominator of the authors is the fact that their research is within the area of model order reduction, but, their backgrounds, scientific computing, mathematics, control or engineering, is different. Nevertheless, in our opinion, the sharing of code, good documentation, and modular programs which can be reused is essential for the further success of model order reduction. The intention of this best practice example is to show exemplarily the files and rules for good CBEx's and provide a template for other research. During implementation, we particularly paid attention to follow the guidelines given in this work. In a first step, the IRKA algorithm [21] is chosen because the algorithm is widely used, heavily cited algorithm, but also has a well-documented examples section where the numerical experiments used to verify the behavior of the algorithm are described including the model. Also, the outcome of the algorithm is for many examples deterministic. Therefore, replicability of the results of [21] is achieved. The minimum requirement for replicability is the basic documentation, which documents the `RUNME.m` file and every single function. Two example files are given. In the first example, `RUNME.m`, the IRKA algorithm automatically produces the plots shown in Figure 4. The second example file, `EXAMPLES.m`, can be used to test the algorithm with different examples and on various system architectures with different programs and different program versions. Documentation in the header, which architectures and programs work with the algorithm and the test examples, is as recommended. Furthermore, standardized benchmarks from the *Oberwolfach Benchmark Collection*[27] are used to allow reproducibility of the results for other users.

Finally, demonstrating the advantages of the reusable part of the implementation is based on the work of *Panzer* [48]. Since the source code of *Panzer* [48] is published under an open-source license, a reuse of his work is possible. We can modify and use the code for our own purpose. Consequently, for a further reuse of the source code, this implementation is also published under a public license. The code is released through a *GitLab* archive[28] and uniquely identified and archived via a *Zenodo*[8] entry with a DOI [16], the availability of the source code is depicted in our Code Availability Section below. Nevertheless to show the possibility to combine open source code with closed source code the function `calculateFrequencyResponse.p` is given in the obfuscated p-code version.

The results shown in Figure 4 use *Penzl*'s "FOM" benchmark example (see [49, Sec. C.3.1]) and apply our implementation of the method from [21]. In the reported test the initial shift parameters and the reduced order have been chosen such that the progress of the IRKA iteration becomes visible. Larger reduced orders would allow for smaller error norms, while more clever choices of the initial shift could lead to less overall iterations. Both are however beyond the scope of this presentation.

[27] `portal.uni-freiburg.de/imteksimulation/downloads/benchmark`
[28] `gitlab.mpi-magdeburg.mpg.de/saak/best_practice_IRKA.git`

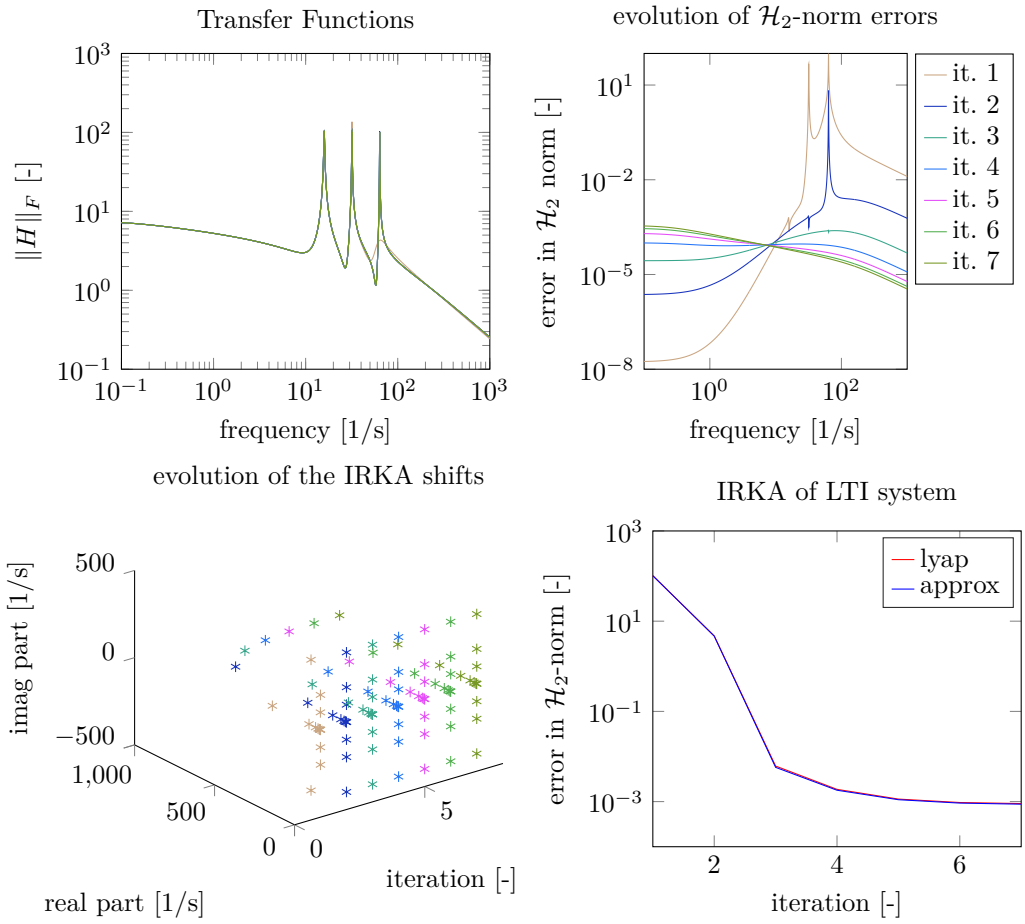


**Figure 4. Example IRKA results for the "FOM" model by Penzl and reduced order 10.**

## 7. Closing Remarks

In this contribution the notions of replicability, reproducibility and reusability are discussed and classified by requirements and recommendations. The issue of code availability and the implied reflection on the artifacts of associated CBEx is exemplified, and simple formats of documentation and meta-data provisioning are described.

This set of best practices is meant as a practical guideline for scientists to establish or enhance RRR for their CBEx, whereas in e.g. [61] many high-level recommendations, like a cultural change of researcher, funding agencies, publishers, and educators, are given. Therefore, this work can be seen as an evolution of the work of *Stodden* et al. [61] applied to a specific domain. Specifically, this work supports the theses of [61] (and also others e.g.: [5,38,3]), and several recommendations overlap, for instance, the advice on providing details on computational work [61, Sec. 2, Appendix D] conforms to the results from Section 5, yet in the work at hand the respective guidelines are particularized and exemplified. Furthermore, novel advice of `RUNME` and `DEPENDENCIES` files, proposed in Section 5, is highlighted here, too. Other important issues [61, Appendix B] relating to hardware and software, such as rounding, parallelization, and distributed computing, are not investigated here.

The proposed best practices in this work improve scientific validity of CBEx, but also aim to spark a discussion on RRR in this context. By no means are the suggested techniques to be understood as a strict rulebook with everlasting validity. The authors emphasize that the proposed practices, which are based on practical experience and standards as well as on general considerations of abstract concepts, are subject to change over time. Nonetheless, the herein demonstrated strategies do enhance replicability, reproducibility & reusability and thus, also in the absence of other general solutions or approaches, merit their consideration for scientific CBEx in general and numerical CBEx in particular.

### Code Availability

The source code of the implementations used to compute the presented results can be obtained from:

`doi:10.5281/zenodo.55297` and is authored by: Jörg Fehr and Jens Saak

Please contact Jörg Fehr and Jens Saak for licensing information

### Acknowledgements

This work was supported by the Deutsche Forschungsgemeinschaft, DFG EXC 1003 Cells in Motion – Cluster of Excellence, Münster; the Center for Developing Mathematics in Interaction, DEMAIN, Münster; and the Deutsche Forschungsgemeinschaft, DFG EXC 310/1 Simulation Technology at the University of Stuttgart.

### Conflict of Interest

All authors declare no conflicts of interest in this paper.

## References


1. D. H. Bailey, J. M. Borwein, and V. Stodden. Set the default to "open". *Notices of the AMS*, 60(6): 679–680, 2013. URL http://dx.doi.org/10.1090/noti1014.
2. D. H. Bailey, J. M. Borwein, and V. Stodden. Facilitating reproducibility in scientific computing: Principles and practice. In H. Atmanspacher and S. Maasen, editors, *Reproducibility: Principles, Problems, Practices, and Prospects*, pages 205–232. Wiley, 2016. ISBN 9781118864975. URL http://dx.doi.org/10.1002/9781118865064.ch9.
3. W. Bangerth and T. Heister. Quo vadis, scientific software? *SIAM News*, 47(1), 2014. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.636.5594. Accessed: 2016-09-22.
4. L. A. Barba. Why should i believe your supercomputing research? http://medium.com/@lorenaabarba/why-should-i-believe-your-supercomputing-research-a7cbf4cbc6b4#.anqkh5o3s, 2016. URL http://archive.is/2Zgx4.
5. N. Barnes. Publish your computer code: it is good enough. *Nature*, 467:753, 2010. URL http://dx.doi.org/10.1038/467753a.
6. P. Bourque and R. E. Fairley, editors. *Guide to the Software Engineering Body of Knowledge (SWEBOK), Version 3.0*. IEEE Computer Society, 2014. URL http://swebok.org.
7. B. Brembs. Earning credibility in post-factual science? bjoern.brembs.net/2016/02/earning-credibility-in-post-factual-science, 2016. URL http://archive.is/zniPL.
8. C. T. Brown. Replication, reproduction, and remixing in research software. ivory.idyll.org/blog/research-software-reuse.html, 2013. URL http://archive.is/2myPk.
9. J. B. Buckheit and D. L. Donoho. Wavelab and reproducible research. In A. Antoniadis and G. Oppenheim, editors, *Wavelets and Statistics*, volume 103 of *Lecture Notes in Statistics*, pages 55–81. Springer, New York, 1995. URL http://dx.doi.org/10.1007/978-1-4612-2544-7_5.
10. S. Chaturantabut and D. C. Sorensen. Nonlinear model reduction via discrete empirical interpolation. *SIAM Journal of Scientific Computing*, 32(5):2737–2764, 2010. URL http://dx.doi.org/10.1137/090766498.
11. C. Collberg, T. Proebsten, and A. M. Warren. Repeatability and benefaction in computer systems research. Technical report, University of Arizona, 2014. URL http://reproducibility.cs.arizona.edu/v2/RepeatabilityTR.pdf. Accessed: 2016-09-22.
12. Scientific Data. Editorial and publishing policies. www.nature.com/sdata/for-authors/editorial-and-publishing-policies#code-avail, 2016. URL http://archive.is/c0BBk.
13. S. M. Easterbrook. Open code for open science? *Nature Geoscience*, 7:779–781, 2014. URL http://dx.doi.org/10.1038/ngeo2283.

14. J. W. Eaton, D. Bateman, S. Hauberg, and R. Wehbring. GNU Octave version 4.0.3 manual: a high-level interactive language for numerical computations. http://www.gnu.org/software/octave/octave.pdf, 2016. Accessed: 2016-09-22.

15. T. M. Errington, E. Iorns, W. Gunn, F. E. Tan, J. Lomax, and B. A. Nosek. An open investigation of the reproducibility of cancer biology research. *eLife*, 3:e04333, 2014. URL http://dx.doi.org/10.7554/eLife.04333.

16. J. Fehr and J. Saak. Iterative rational Krylov algorithm (IRKA). DOI 10.5281/zenodo.49965, 2016. URL http://dx.doi.org/10.5281/zenodo.49965.

17. S. Fomel and J. F. Claerbout. Guest editors' introduction: Reproducible research. *Computing in Science & Engineering*, 11(1):5–7, 2009. URL http://dx.doi.org/10.1109/MCSE.2009.14.

18. Interoperability Solutions for European Public Administrations. Asset Description Metadata Schema for Software, 2012. URL http://joinup.ec.europa.eu/asset/adms_foss/asset_release/admssw-100. Accessed: 2016-09-22.

19. I. P. Gent. The recomputation manifesto. cs.GL 1304.3674, arXiv, 2013. URL http://arxiv.org/abs/1304.3674.

20. T. Glatard, L. B. Lewis, R. F. da Silva, R. Adalat, N. Beck, C. Lepage, P. Rioux, M.-E. Rousseau, T. Sherif, E. Deelman, N. Khalili-Mahani, and A. C. Evans. Reproducibility of neuroimaging analyses across operating systems. *Frontiers in Neuroinformatics*, 9, 2015. URL http://dx.doi.org/10.3389/fninf.2015.00012.

21. S. Gugercin, A. C. Antoulas, and C. A. Beattie. $\mathcal{H}_2$ Model reduction for large-scale linear dynamical systems. *SIAM Journal on Matrix Analysis and Applications*, 30(2):609–638, 2008. URL http://dx.doi.org/10.1137/060666123.

22. M. A. Heroux and J. M. Willenbring. Barely sufficient software engineering: 10 practices to improve your CSE software. In *ICSE Workshop on Software Engineering for Computational Science and Engineering*, pages 15–21, 2009. URL http://dx.doi.org/10.1109/SECSE.2009.5069157.

23. C. Himpe. emgr - **Em**pirical **Gr**amian framework (Version 3.9). gramian.de, 2016. URL http://dx.doi.org/10.5281/zenodo.46523.

24. The Mathworks Inc. *Matlab, Product Help, Matlab Release 2014b*. Mathworks Inc., Natick MA, USA, 2014.

25. D. C. Ince, L. Hatton, and J. Graham-Cumming. The case for open computer programs. *Nature*, 482:485–488, 2012. URL http://dx.doi.org/10.1038/nature10836.

26. ipol. IPOL Journal - Image Processing On Line. URL http://www.ipol.im. Accessed: 2016-09-22.

27. D. Irving. A minimum standard for publishing computational results in the weather and climate sciences. *Bulletin of the American Meteorological Society*, 2015. URL http://dx.doi.org/10.1175/BAMS-D-15-00010.1.

28. *ISO 646 - Information technology – ISO 7-bit coded character set for information interchange*. ISO, International Organization for Standardization, 1991. URL http://www.iso.org/cate/d4777.html. Accessed: 2016-09-22.

29. *ISO 8601 - Data elements and interchange formats – Information interchange – Representation of dates and times*. ISO, International Organization for Standardization, 2004. URL http://www.iso.org/iso/iso8601. Accessed: 2016-09-22.

30. D. James, N. Wilkins-Diehr, V. Stodden, D. Colbry, and C. Rosales. Standing together for reproducibility in large-scale computing. In *XSEDE14 Workshop*, volume reproducibility@XSEDE, 2014. URL http://xsede.org/documents/659353/d90df1cb-62b5-47c7-9936-2de11113a40f. Accessed: 2016-09-22.

31. L. K. John, G. Loewenstein, and D. Prelec. Measuring the prevalence of questionable research practices with incentives for truth telling. *Psychological Science*, 23(5):524–532, 2012. URL http://dx.doi.org/10.1177/0956797611430953.

32. L. N. Joppa, D. Gavaghan, R. Harper, K. Takeda, and S. Emmott. Optimizing peer review of software code – response. *Science*, 341(6143):237, 2013a. URL http://dx.doi.org/10.1126/science.341.6143.237-a.

33. L. N. Joppa, G. McInerny, R. Harper, L. Salido, K. Takeda, K. O'Hara, D. Gavaghan, and S. Emmott. Troubling trends in scientific software use. *Science*, 340(6134):814–815, 2013b. URL http://dx.doi.org/10.1126/science.1231535.

34. D. Joyner and W. Stein. Open source mathematical software. *Notices – American Mathematical Society*, 54(10):1279, 2007. URL http://www.ams.org/notices/200710/tx071001279p.pdf. Accessed: 2016-09-22.

35. D. S. Katz and A. M. Smith. Transitive credit and JSON-LD. *Journal of Open Research Software*, 3(1), 2015. URL http://dx.doi.org/10.5334/jors.by.

36. D. Kelly, D. Hook, and R. Sanders. Five recommended practices for computational scientists who write software. *Computing in Science & Engineering*, 11(5):48–53, 2009. URL http://dx.doi.org/10.1109/MCSE.2009.139.

37. S. Krishnamurthi and J. Vitek. The real software crisis: Repeatability as a core value. *Communications of the ACM*, 58(3):34–36, 2015. URL http://dx.doi.org/10.1145/2658987.

38. R. J. LeVeque. Top ten reasons to not share your code (and why you should anyway). *SIAM News*, 46(3), 2013. URL http://archive.is/eAr7z.

39. G. Marcus. The crisis in social psychology that isn't. www.newyorker.com/tech/elements/the-crisis-in-social-psychology-that-isnt, 2013. URL http://archive.is/yBJy1.

40. B. Marwick. Computational reproducibility in archaeological research: Basic principles and a case study of their implementation. *Journal of Archaeological Method and Theory*, pages 1–27, 2016. URL http://dx.doi.org/10.1007/s10816-015-9272-9.

41. D. McCafferty. Should code be released? *Communications of the ACM*, 53(10):16–17, 2010. URL http://dx.doi.org/10.1145/1831407.1831415.

42. Z. Merali. Computational science: ...error. *Nature*, 467:775–777, 2010. URL http://dx.doi.org/10.1038/467775a.

43. O. Mesnard and L. A. Barba. Reproducible and replicable CFD: it's harder than you think. physics.comp-ph 1605.04339, arXiv, 2016. URL http://arxiv.org/abs/1605.04339.

44. nature14. Code share. *Nature*, 514:536, 2014. URL http://dx.doi.org/10.1038/514536a.

45. nature15. Ctrl alt share. *Scientific Data*, 2, 2015. URL http://dx.doi.org/10.1038/sdata.2015.4.

46. J. Nitsche. Über ein Variationsprinzip zur Lösung von Dirichlet-Problemen bei Verwendung von Teilräumen, die keinen Randbedingungen unterworfen sind. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 36(1):9–15, 1971. URL http://dx.doi.org/10.1007/BF02995904.

47. Open Science Collaboration. Estimating the reproducibility of psychological science. *Science*, 349 (6251), 2015. URL http://dx.doi.org/10.1126/science.aac4716.

48. H. K. F. Panzer. *Model Order Reduction by Krylov Subspace Methods with Global Error Bounds and Automatic Choice of Parameters*. Dissertation, Technische Universität München, 2014. URL http://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:bvb:91-diss-20140916-1207822-0-0.

49. T. Penzl. LYAPACK Users Guide. Technical report, Sonderforschungsbereich 393 *Numerische Simulation auf massiv parallelen Rechnern*, TU Chemnitz, 2000. URL http://www.netlib.org/lyapack/guide.pdf. Accessed: 2016-09-22.

50. K. R. Popper. *The Logic of Scientific Discovery*. Classics Series. Routledge, 2002. ISBN 9780415278447.

51. A. Prlić and J. B. Procter. Ten simple rules for the open development of scientific software. *PLoS Computational Biology*, 8(12):1–3, 2012. URL http://dx.doi.org/10.1371/journal.pcbi.1002802.

52. R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2014. URL http://R-project.org.

53. Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986. URL http://dx.doi.org/10.1137/0907058.

54. T. C. Schulthess. Programming revisited. *Nature Physics*, 11(5):369–373, 2015. URL http://dx.doi.org/10.1038/nphys3294.

55. science. Publication policies: Data and materials availability after publication. www.sciencemag.org/authors/science-editorial-policies#publication-policies, 2016. URL http://archive.is/e4GT7.

56. P. Sliz and A. Morin. Optimizing peer review of software code. *Science*, 341(6143):236–237, 2013. URL http://dx.doi.org/10.1126/science.341.6143.236-b.

57. A. M. Smith. JSON-LD for software discovery, reuse and credit. www.arfon.org/json-ld-for-software-discovery-reuse-and-credit, 2014. URL http://archive.is/BgMsx.

58. V. Stodden. Enabling reproducible research: Open licensing for scientific innovation. *International Journal of Communications Law and Policy*, pages 1–55, 2009a. URL http://ssrn.com/abstract=1362040. Accessed: 2016-09-22.

59. V. Stodden. The legal framework for reproducible scientific research: Licensing and copyright. *Computer in Science & Engineering*, 11(1):35–40, 2009b. URL http://dx.doi.org/10.1109/MCSE.2009.19.

60. V. Stodden and S. Miguez. Best practices for computational science: Software infrastructure and environments for reproducible and extensible research. *Journal of Open Research Software*, 2(1), 2014. URL http://dx.doi.org/10.5334/jors.ay.

61. V. Stodden, D. H. Bailey, J. Borwein, R. J. LeVeque, W. Rider, and W. Stein. Setting the default to reproducible: Reproducibility in computational and experimental mathematics. Technical report, ICERM Report, 2013a. URL http://icerm.brown.edu/tw12-5-rcem/icerm_report.pdf. Accessed: 2016-09-22.

62. V. Stodden, J. Borwein, and D. H. Bailey. "Setting the default to reproducible" in computational science research. *SIAM News*, 46:4–6, 2013b. URL http://archive.is/ESi5J.

63. D. L. Vaux, F. Fidler, and G. Cumming. Replicates and repeats—what is the difference and is it significant? *EMBO reports*, 13(4):291–296, 2012. URL http://dx.doi.org/10.1038/embor.2012.36.

64. J. Vitek and T. Kalibera. Repeatability, reproducibility, and rigor in systems research. In *Proceedings of the 9th ACM International Conference on Embedded Software*, pages 33–38, 2011. URL http://dx.doi.org/10.1145/2038642.2038650.

65. G. Wilson, D. A. Aruliah, C. T. Brown, N. P. C. Hong, M. Davis, R. T. Guy, S. H. D. Haddock, K. D. Huff, I. M. Mitchell, M. D. Plumbley, B. Waugh, E. P. White, and P. Wilson. Best practices for scientific computing. *PLoS Biology*, 12(1), 2014. URL http://dx.doi.org/10.1371/journal.pbio.1001745.

66. G. Wilson, J. Bryan, K. Cranston, J. Kitzes, L. Nederbragt, and T. K. Teal. Good enough practices in scientific computing. cs.SE 1609.00037, arXiv, 2016. URL http://arxiv.org/abs/1609.00037.


AIMS Press