ESSAY

A MINIMUM STANDARD FOR PUBLISHING COMPUTATIONAL RESULTS IN THE WEATHER AND CLIMATE SCIENCES

BY DAMIEN IRVING

A procedure for publishing reproducible computational research is described that could be adopted as a minimum standard by journals in the weather and climate sciences.

The rise of computational science has led to unprecedented opportunities in the weather and climate sciences. Ever more powerful computers enable experiments that would have been considered impossible only a decade ago, while new hardware technologies allow data collection in even the most inaccessible places. To analyze the vast quantities of data now available to them, modern practitioners—most of whom are not computational experts—use an increasingly diverse set of software tools and packages. Today's weather or climate scientist is far more likely to be found debugging code written in Python, MATLAB, Interactive Data Language (IDL), NCAR Command Language (NCL), or R than to be poring over satellite images or releasing radiosondes.

This computational revolution is not unique to the weather and climate sciences and has led to something of a reproducibility crisis in published research (e.g., Peng 2011). Most papers do not make the data and code underpinning key findings available, nor do they adequately specify the software packages and libraries used to execute that code. This means it is impossible to replicate and verify most of the computational results presented in journal articles today. By extension (and perhaps even more importantly), it is also impossible for readers to interrogate the data processing methodology. If a reader cannot find out which Python library was used in regridding a particular dataset, how can they build upon that regridding method and/or apply it in their own context?

A movement within the computational science community has arisen in response to this crisis, calling for existing communication standards to be adapted to include the data and code associated with published findings (e.g., Stodden and Miguez 2014). The movement has also been active in producing best

AFFILIATIONS: IRVING—School of Earth Sciences, University of Melbourne, Parkville, Victoria, Australia
CORRESPONDING AUTHOR: Damien Irving, School of Earth Sciences, University of Melbourne, Parkville VIC 3010, Australia
E-mail: irving.damien@gmail.com

The abstract for this article can be found in this issue, following the table of contents.
DOI:10.1175/BAMS-D-15-00010.1

In final form 4 September 2015

practice recommendations to guide scientists and stakeholders (e.g., Prlić and Procter 2012; Stodden 2012; Sandve et al. 2013; Stodden and Miguez 2014), and similar calls and guidelines have appeared in numerous editorials and commentaries in recent years (e.g., Barnes 2010; Merali 2010; Ince et al. 2012). In response to this sustained campaign, there has been a modest but perceptible reaction from funding agencies and academic journals. Agencies like the National Science Foundation now require dataset disclosure and encourage software availability; however, this is not consistently enforced and compliance is largely left to the authors themselves (Stodden et al. 2013). A recent review of journal policies found a trend toward data and code availability but, overall, the vast majority of journals have no data or code policy (Stodden et al. 2013).

Similar to many other computational disciplines, in the weather and climate sciences progress on code availability is lagging behind data availability. The societies behind most of the major journals (the American Meteorological Society, Royal Meteorological Society, American Geophysical Union, and European Geosciences Union) all have official data policies (e.g., Mayernik et al. 2015b); however, only two of the four indicate that code is included under their broad definition of data or metadata. Where code is included, statements regarding code availability consist of brief, vague suggestions that are not enforced by editors and reviewers. New journals such as *Geoscientific Model Development* have arisen for documenting work where code/software is the primary output (e.g., the development of a new climate model), but little progress has been made in documenting the computational aspects of research where code is ancillary to the main focus (i.e., where the code is not of sufficient consequence to require a standalone paper devoted to its description). Given that much of the research conducted by weather and climate scientists is based on previously documented datasets and/or models (e.g., a paper might analyze a reanalysis dataset or the output from running a well-known atmospheric model forced with anomalous sea surface temperatures), ancillary code availability (as opposed to data availability or primary code availability) is the component of the reproducibility crisis common to essentially all research today.

While it is tempting to simply decry the slow response of journals and funding agencies in the face of this crisis, the reality is that examples of reproducible weather and climate research upon which to base new communication standards have only just begun to emerge. For instance, the Max Planck Institute for Meteorology (MPI-M) recently enacted a policy (Stevens 2015b) that requires all primary data (including ancillary code) to be archived, and papers adhering to that policy are now starting to be published (e.g., Stevens 2015a). There are also a limited number of examples from other research disciplines where highly motivated computational scientists have taken a variety of different approaches to publishing reproducible results (e.g., Ketcheson and Ahmadia 2012; Crooks and Hailegiorgis 2014; Bremges et al. 2015; Schmitt et al. 2015).

To stimulate further discussion and progress in this area, this essay describes a procedure for reporting computational results that was employed in a recent *Journal of Climate* paper (Irving and Simmonds 2015, hereafter IS2015). Building on the aforementioned examples, the procedure was developed to be consistent with recommended computational best practices and seeks to minimize the time burden on authors. For the reasons articulated above and the fact that data availability has already been addressed in a recent *BAMS* essay (Mayernik et al. 2015a), the focus of the procedure is on ancillary code availability. It should provide a starting point for weather and climate scientists looking to publish reproducible research, and it is proposed that the procedure could be adopted as a minimum standard by relevant academic journals and institutions.

A REPRODUCIBLE PAPER. *Rationale*. In devising a procedure that might be adopted as a minimum communication standard, it was important to first consider why computational scientists do not currently publish their code. The literature on this topic is relatively sparse; however, a recent survey of the machine learning community found that the top reason for not sharing code was the perceived time required to prepare it for publication, followed by the prospect of dealing with questions from users (Stodden 2010). While many researchers are sympathetic to the ideals of open science and reproducible research, it appears that the practicalities seem too difficult and time consuming, particularly when the pressure to publish is so strong and unrelenting.

The bewildering array of suggested tools and best practices is one reason why an individual working in the weather and climate sciences might place reproducibility in the "too hard" basket. An appropriate solution for any given scientist no doubt exists within that collection of tools and practices, but it is heavily obscured. Consider the "regular" scientist described in the sidebar. In consulting the literature on reproducible computational research, they would

THE REGULAR SCIENTIST

The procedure for documenting computational results adopted by IS2015 was developed with a "regular" weather/climate scientist in mind. A characterization of this scientist is given below and is based on the few documented surveys of how computational scientists do their work (Hannay et al. 2009; Stodden 2010; Momcheva and Tollerud 2015), editorials describing current computational practices (e.g., Easterbrook 2014), and my personal experience teaching at numerous Software Carpentry workshops (Wilson 2016) over the past few years. This regular scientist

- works with publicly available data (e.g., CMIP data, reanalysis data) that are often large in size (e.g., it might be tens or hundreds of gigabytes) but are not so large as to be considered "big data";
- acquired the knowledge to develop and use scientific software from peers and through self-study, as opposed to formal education and training;
- primarily relies on software like Python, MATLAB, IDL, NCL, or R, which has a large user/support base and is relatively simple to install on a Windows, Mac, or Linux computer;
- does most of their work on a desktop or intermediate computer (as opposed to a supercomputer);
- is only writing code for a specific task/paper and is not looking for community uptake; and
- works on their own or in a very small team (i.e., two to three other scientists) and does not have access to professional software developers for support.

Some scientists might be regular in most but not all aspects of their work (e.g., all the points above might apply except they occasionally use a highly specialized software package that does not have a large support base) so this characterization can be thought of as a baseline or minimum level of computation that essentially all weather and climate scientists are engaged in.

be confronted with options including data provenance tracking systems like VisTrails (Freire and Silva 2012) and PyRDM (Jacobs et al. 2014), software environment managers like Docker and Vagrant (Stodden and Miguez 2014), and even online services like RunMyCode.org where your code and data can be run by others (Stodden et al. 2012). These might be fantastic options for small teams of software engineers or experienced scientific programmers dealing with very large workflows [e.g., the postprocessing of thousands of Coupled Model Intercomparison Project (CMIP) model runs], complex model simulations, and/or production style code (e.g., they might be developing a satellite retrieval algorithm that has high reuse potential in the wider community), but a regular scientist has neither the requisite computational experience or a research problem of sufficient scale and complexity to necessarily require and/or make use of such tools.

The procedure employed by IS2015 was designed with this regular scientist in mind. It looked to minimize both the time involved and the complexity of the associated tools, while at the same time remaining faithful to established best practices in scientific computing.

Components. At first glance, the only difference between a regular journal article and that of IS2015 is the addition of a short computation section (see "Example computation section" for more information). That section accompanied the traditional description of data and methods within the article and briefly cited the major software packages used in the research before pointing the reader to three key supplementary items: 1) a more detailed description of the software used, 2) a version controlled and publicly available code repository, and 3) a collection of supplementary log files that captured the data processing steps taken in producing each key result. The repository was hosted at a popular code sharing website called GitHub, while the detailed software description and log files were hosted at Figshare (Irving 2015a), which is a website where researchers commonly archive the "long tail" of their research (e.g., supplementary figures, code, and data).

Software description. There is an important difference between citing the software that was used in a study (i.e., so that the authors get appropriate academic credit) and describing it in sufficient detail so as to convey the precise version and computing environment (Jackson 2015). Recognizing this, IS2015 began their computation section with a high-level description of the software used, which included citations to any papers written about the software. Authors of scientific software are increasingly publishing with journals like the *Journal of Open Research Software*, so it is important for users of that software to cite those papers within their manuscripts. This high-level description is also useful for briefly articulating

what general tasks each software item was used for (e.g., plotting, data analysis, file manipulation). Such an overview does not provide sufficient detail to recreate the computing environment used in the study, so IS2015 provided a link to a supplementary file on Figshare that documents the precise version of each software package used and the operating system upon which it was run (viz., the name, version number, release date, institution, and DOI or URL).

Code repository. An important best practice in scientific computing is that of modularizing code, rather than copying and pasting (Wilson et al. 2014). As such, any communication standard for code should expect/encourage authors to provide a whole library of code (i.e., a repository containing many interconnected scripts) as opposed to a single script for each key figure or result. A related best practice is the use of a version control system like Git, Subversion, or Mercurial. These systems are easily linked to an online hosting service such as GitHub or Bitbucket, which is the means by which a code repository can be made publicly available. Most of the examples of reproducible research currently available in the literature produced a pristine GitHub or Bitbucket repository for their paper (i.e., one that contains only the code that is directly relevant to that paper); however, this should not be an expectation for regular scientists who are not looking for broad-scale community uptake of their code. Not only is this a time consuming practice that would likely involve a degree of cutting and pasting, it also goes against the workflow that version control promotes. By providing a link to their everyday repository instead, authors can quickly and easily signal to readers what code was used to produce the results in their paper. The authors may further develop that code in the future, so the reader then also has easy access to the latest version if they would like to use it in their own work and/or suggest improvements (referred to as a "pull request" in version control parlance). There are all sorts of extra bits and pieces of code in the everyday repository that is linked to by IS2015. Readers will probably never look at that code (because it is not referred to in the associated log files), but what is the harm if they do? Students and other scientists could potentially learn from it, and in the best case scenario they would inform the authors of a bug or potential improvement to the code.

In addition to providing a link to the associated Github repository, IS2015 provided (on Figshare) a static snapshot of the repository taken at the time the manuscript was accepted for publication. The motivation for doing this was to ensure that a static version of the code is available in case the associated GitHub repository is ever moved or deleted. Unlike GitHub, archiving sites such as Figshare and Zenodo issue DOIs, which function as a perpetual link to the resource and can only be obtained by agencies that commit to maintain a reliable level of preservation (Potter and Smith 2015). Since this snapshot does not provide a revision history of the code, it would not have been appropriate to provide it instead of the link to GitHub. Not all of the results presented in a paper will necessarily have been generated with the very latest version of the code, hence the need for the revision history (and even if they were generated with the latest version, it is not possible to submit a pull request to Figshare or Zenodo). Both Figshare and Zenodo provide a simple interface for importing code directly from GitHub, so the process is very straightforward.

EXAMPLE COMPUTATION SECTION

The results in this paper were obtained using a number of different software packages. A collection of command line utilities known as the NetCDF Operators (NCO) and Climate Data Operators (CDO) were used to edit the attributes of netCDF files and to perform routine calculations on those files (e.g., the calculation of anomalies and climatologies) respectively. For more complex analysis and visualization, a Python distribution called Anaconda was used. In addition to the Numerical Python (NumPy; Van Der Walt et al. 2011) and Scientific Python (SciPy) libraries that come installed by default with Anaconda, a Python library called xray was used for reading/writing netCDF files and data analysis. Similarly, in addition to Matplotlib (the default Python plotting library; Hunter 2007), Iris and Cartopy were used to generate many of the figures.

To facilitate the reproducibility of the results presented, an accompanying Figshare repository has been created to document the computational methodology (Irving 2015a). In addition to a more detailed account (i.e., version numbers, release dates, web addresses) of the software packages discussed above, the Figshare repository contains a supplementary file for each figure in the paper, outlining the computational steps performed from initial download of the ERA-Interim data through to the final generation of the plot. A version controlled repository of the code referred to in those supplementary files can be found at https://github.com/DamienIrving/climate-analysis.

EXAMPLE LOG FILE

The following is the log file corresponding to Fig. SB1. It shows every command line entry executed in creating the figure from the initial download of the underlying data to the final generation of the figure. Details regarding the software and code referred to in the log file are provided in the "Example computation section."

```
Tue Jun 30 08:57:37 2015: /
usr/local/anaconda/bin/
python /home/STUDENT/
dbirving/climate-analysis/
visualisation/plot_hilbert.
py/mnt/meteo0/data/simmonds/
dbirving/ERAInterim/data/
va_ERAInterim_500 hPa_030day-
runmean_native.nc va /
mnt/meteo0/data/simmonds/
dbirving/ERAInterim/
data/zw/figures/hilbert_
zw_w19_va_ERAInterim_500
hPa_030day-runmean_native-
55S_1986-05-22_2006-07-29.
png 1 2 --latitude -55
--dates 1986-05-22 2006-07-29
--wavenumbers 1 9 --figure_size
15 6 (Git hash: 1a906a7)

Tue Jun 30 07:35:49 2015: cdo
runmean,30 /mnt/meteo0/data/
simmonds/dbirving/ERAInterim/
data/va_ERAInterim_500
hPa_daily_native.nc /mnt/
meteo0/data/simmonds/
dbirving/ERAInterim/data/
va_ERAInterim_500 hPa_030day-
runmean_native.nc

Wed Mar 18 17:17:14 2015: cdo
mergetime va_ERAInterim_500
hPa_daily-2014-06-01-
to-2014-12-31_native.nc ../
data/va_ERAInterim_500 hPa_
daily_native_orig.nc ../data/
va_ERAInterim_500 hPa_daily_
native.nc

Mon Nov 10 17:15:49 2014:
ncatted -O -a level,va,o,c,500
hPa va_ERAInterim_500 hPa_
daily_native.nc

Mon Nov 10 16:31:05 2014:
ncatted -O -a long_
name,va,o,c,northward_wind
va_ERAInterim_500 hPa_daily_
native.nc

Thu Aug 21 10:57:46 2014:
ncatted -O -a axis,time,c,c,T
va_ERAInterim_500 hPa_daily_
native.nc

Thu Aug 21 10:34:46 2014:
ncrename -O -v v,va va_
ERAInterim_500 hPa_daily_
native.nc

Thu Aug 21 10:26:49
2014: cdo invertlat
-sellonlatbox,0,359.9,-90,90
-daymean va_ERAInterim_500
hPa_6hourly_native.nc va_
ERAInterim_500 hPa_daily_native.
nc

Thu Aug 21 10:14:59
2014: cdo mergetime ../
download/va_ERAInterim_500
hPa_6hourly-1979-1988_
native_unpacked.nc ../
download/va_ERAInterim_500
hPa_6hourly-1989-1998_
native_unpacked.nc ../
download/va_ERAInterim_500
hPa_6hourly-1999-2008_
native_unpacked.nc ../
download/va_ERAInterim_500
hPa_6hourly-2009-2014_native_
unpacked.nc va_ERAInterim_500
hPa_6hourly_native.nc

Thu Aug 21 10:13:35 2014:
ncpdq -P upk va_ERAInterim_500
hPa_6hourly-2009-2014_
native.nc va_ERAInterim_500
hPa_6hourly-2009-2014_native_
unpacked.nc

Wed Aug 20 23:16:22 2014:
Initial download of 6 hourly,
500 hPa meridional wind ERA-
Interim data in 5 or 10 year
chunks (e.g., va_ERAInterim_500
hPa_6hourly-2009-2014_native.
nc) from http://apps.ecmwf.
int/datasets/data/interim-
full-daily/
```

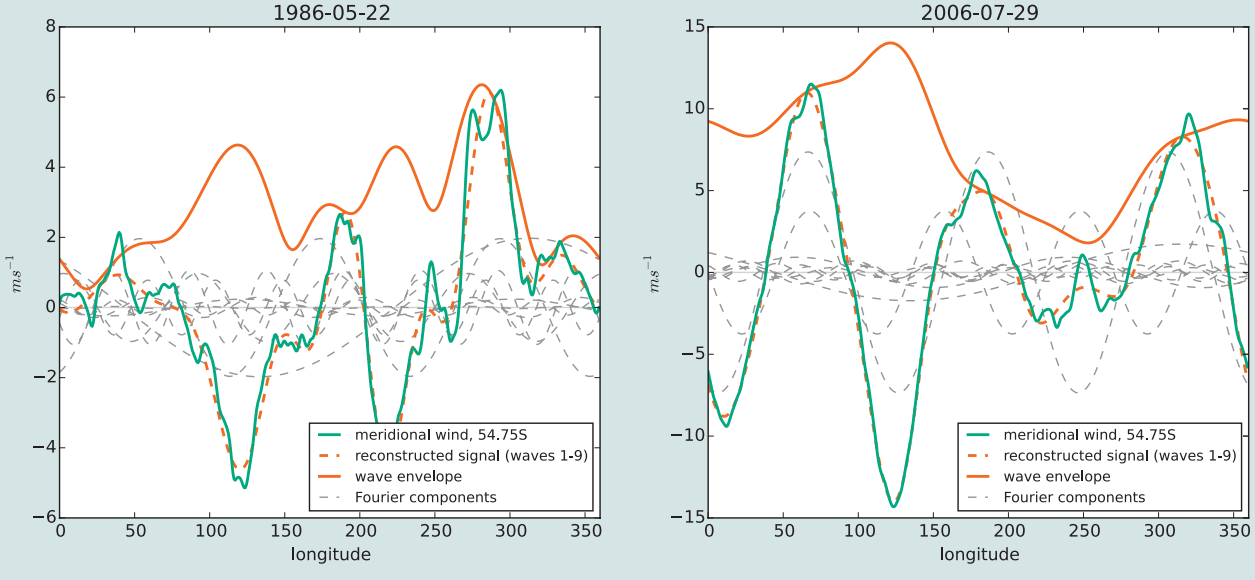

Fig. SB1. Wave envelope of the ERA-Interim, 30-day running mean, 500-hPa meridional wind for 22 May 1986 and 29 Jul 2006 at 54.75°S. See IS2015 for details.

Log files. A code repository and software description on their own are not much use to a reader; they also need to know how that code was used in generating the results presented in the paper. It turns out that in the weather and climate sciences, the answer to adequately documenting the computational steps involved in producing a given result has been staring us in the face for years. As a community we have almost universally adopted a self-describing file format (i.e., a format where metadata can be stored within the file) called network Common Data Form (netCDF), which means we have been able to develop numerous software tools for processing and manipulating data stored in that format. The most well known of these are a collection of command line tools known as the NetCDF Operators (NCO) and Climate Data Operators (CDO). Whenever an NCO or CDO command is executed, a time stamp followed by a copy of the command line entry is automatically placed into the global attributes of the output netCDF file, thus maintaining a history of the provenance of that data (see "Example log file" for examples of NCO and CDO entries).

What is important here is not the specific software tools or file format (there are many researchers who do not use NCO, CDO, or netCDF files) but rather the deceptively simple method for recording previous computational steps. Using any of the programming languages common to the weather and climate sciences, a user can obtain details of the associated command line entry and append such text to the global attributes of a netCDF file (or a corresponding metadata text file if dealing with file formats that are not self-describing). In fact, in all of these languages such tasks can be achieved with just one or two lines of additional code [e.g., see Irving (2015b) for a short lesson on how to do this using Python]. IS2015 provided a log file containing a complete NCO/CDO-style history for each figure; see "Example log file" for details of one of those files and the associated page on Figshare for the complete set.

An important feature of these log files is that they are both readable and writable by any weather and climate scientist. If advanced practitioners are tracking their computational steps with tools like VisTrails then they can certainly submit log files (or equivalent documentation) exported from those tools, but as a minimum standard it is important that elaborate tools are not a requirement. By spelling out every single computational step (i.e., from initial data download to the final plot/result), the log files also ensure that readers do not need to be familiar with build tools like Make or other workflow management tools in order to figure out which computational steps were executed and in what order. Other features of note include the following:

- Besides a slight amendment to the initial download entry of the log file shown in "Example log file" [the default text provided by the European Centre for Medium-Range Weather Forecasts (ECMWF) interim reanalysis (ERA-Interim) data server was not particularly self-explanatory], no manual editing of its contents was done. This means that if a reviewer asked for a slight modification to the figure, for instance, the regeneration of a new log file would be trivial. By resisting the urge to clean up the file (e.g., one might consider removing path details like /mnt/meteo0/data/simmonds/dbirving/ERAInterim/data/) it also doubles as a record that is highly useful to the author in retracing their own steps (e.g., they could use it to recall where they stored the output data on their local machine).
- As previously mentioned, it cannot be assumed that the latest version of any code repository was used to generate all the results in a given paper. The unique version control revision number/hash value was therefore recorded in the log files, wherever a script written by the author was executed. Languages like Python, R, and MATLAB are able to link with version control systems like Git, so the retrieval of the revision number can be automated.
- When more than one input file is passed to an NCO or CDO function, the history of only one of those files is retained in the output file. On occasions where this is not appropriate (i.e., where the histories of the multiple input files are very different), it is important to ensure that the history of all input files is retained. There are a number of examples of this in the log files provided on Figshare.

A NEW MINIMUM STANDARD. *Implications and practicalities.* Before formally proposing a minimum standard for the communication of computational results, it is worth considering the implications and practicalities of adopting a standard based on the approach of IS2015. The reproducibility of published results would presumably improve, which may also lead to increased trust, interest, and citations (Piwowar et al. 2007), but what else would it mean for authors and reviewers? Would there be barriers to overcome in implementing such a standard, and what could be done to make the transition easier?

Authors. As previously mentioned, Stodden (2010) identified the time involved as the largest barrier to sharing code. There are numerous authors who suggest that the best practices adopted by IS2015 (e.g., scripting, version control) save time in the long run (e.g., Sandve et al. 2013; Wilson et al. 2014), which means that once researchers have learned and adopted these practices they may actually save time. In my experience as a Software Carpentry (Wilson 2016) instructor, many weather and climate scientists are comfortable with the idea of scripting, but very few use version control. Learning these new skills is not overly time consuming (Software Carpentry teaches them in a short two-day workshop), but on a local level it requires an individual or institution to take the lead in liaising with Software Carpentry to find volunteer instructors and to coordinate other logistics. A good example is the Australian Meteorological and Oceanographic Society, who have hosted a Software Carpentry workshop alongside their annual conference for the past three years running. Of course, it is also possible to learn these skills by following an online tutorial (e.g., all the Software Carpentry lessons are available online), but there is an added benefit to the social aspect of a workshop. It helps to reduce the shame many scientists have about the quality of their code (i.e., they see that their peers are no "better" at coding than they are), which is an important part of achieving the required cultural shift toward an acceptance of code sharing (Barnes 2010).

The other potentially time consuming task associated with adopting a minimum standard would be dealing with requests for assistance. One suggested solution to this problem is to make it clear that authors are not obliged to support others in repeating their computations (Easterbrook 2014). This is probably the only feasible solution, but it is worth noting that even if not formally obliged, some authors may fear that refusing requests will make it look like they have something to hide.

Some researchers also face barriers relating to security and proprietary, particularly if they are using large code bases that have been developed for research and/or operations within government laboratories, national weather bureaus, and private companies (Stodden 2010). Such code bases are increasingly being made public [e.g., the Australian Bureau of Meteorology and Commonwealth Scientific and Industrial Research Organisation (CSIRO) host the code for their Climate and Weather Science Laboratory in a public GitHub repository], but any proposed minimum standard would need to allow some flexibility for researchers who are unable to make their code public for these reasons (the code archived by MPI-M is available via request only, which might be an acceptable solution in many cases). For those concerned about getting appropriate academic credit for highly novel and original code, a separate publication (e.g., with the *Journal of Open Research Software*) or software license (e.g., Morin et al. 2012) might also be an option.

Reviewers. If the approach of IS2015 was to be implemented as a minimum standard, it would be important to convey to reviewers that they are not expected to review the code associated with a submission; they simply have to check that it is sufficiently documented (i.e., that the code is available in an online repository and that log files have been provided for all figures and key results). Not only would it be unrealistic to have reviewers examine submitted code owing to the wide variety of software tools and programming languages out there, it would also be inconsistent with the way scientific methods have always been reviewed. For instance, in the 1980s it was common for weather and climate scientists to manually identify weather systems of interest (e.g., polar lows) from satellite imagery. The reviewers of the day were not required to go through all the satellite images and check that the author had counted correctly—they simply had to check that the criteria for polar low identification was adequately documented. This is not to say that counting errors were not made on the part of authors (as with computer code today there were surely numerous errors/bugs)—it was just not the job of the reviewer to find them. Author errors are borne out when other studies show conflicting results and/or when other authors try to replicate key results, which is a process that would be greatly enhanced by having a minimum standard for the communication of computational results. This idea of conceptualizing the peer review of code as a postpublication process is consistent with the publication system envisaged by the open evaluation movement (e.g., Kriegeskorte et al. 2012).

Proposed standards. To assist in establishing a minimum standard for the communication of computational results, it is proposed that the following text could be inserted into the author and reviewer guidelines of journals in the weather and climate sciences (institutions that have their own internal review process could also adopt these guidelines). In places, the language borrows from the guidelines recently adopted by *Nature* (Nature 2014). It is anticipated that a journal would provide links to

examples of well-documented computational results to help both authors and reviewers in complying with these guidelines. The journal could decide to host the supplementary materials itself (i.e., the software description, log files, and static snapshot code repository) or encourage the author to host these items at an external location that can guarantee persistent, long-term access (e.g., an institutionally supported site like MPI-M provides for its researchers or an online academic archive such as Figshare or Zenodo).

Author guidelines. If computer code is central to any of the paper's major conclusions, then the following is required as a minimum standard:

1) A statement describing whether (and where) that code is available and setting out any restrictions on accessibility.
2) A high-level description of the software used to execute that code (including citations for any academic papers written to describe that software).
3) A supplementary file outlining the precise version of the software packages and operating system used. This information should be presented in the following format: name, version number, release date, institution, and DOI or URL.
4) A supplementary log file for each major result (including key figures) listing all computational steps taken from the initial download/attainment of the data to the final result (i.e., the log files describe how the code and software were used to produce the major results).

It is recommended that items 1 and 2 are included in a "computation procedures" (or similarly named) section within the manuscript itself. Any practical issues preventing code sharing will be evaluated by the editors, who reserve the right to decline a paper if important code is unavailable. While not a compulsory requirement, best practice for code sharing involves managing code with a version control system such as Git, Subversion, or Mercurial, which is then linked to a publicly accessible online repository such as GitHub or Bitbucket. In the log files a unique revision number (or hash value) can then be quoted to indicate the precise version of the code repository that was used. Authors are not expected to produce a brand new repository to accompany their paper; an "everyday" repository that also contains code not relevant to the paper is acceptable. Authors should also note that they are not obliged to support reviewers or readers in repeating their computations.

Reviewer guidelines. The reviewer guidelines for most journals already ask if the methodology is explained in sufficient detail so that the paper's scientific conclusions could be tested by others. Such guidelines could simply be added to as follows: "If computer code is central to any of those conclusions, then reviewers should ensure that the authors are compliant with the minimum standards outlined in the author guidelines. It should be noted that reviewers are not obliged to assess or execute the code associated with a submission. They must simply check that it is adequately documented."

DISCUSSION. To combat the reproducibility crisis in published computational research, a simple procedure for communicating computational results has been demonstrated (see IS2015) and its rationale discussed. The procedure involves authors providing three key supplementary items: 1) a description of the software packages and operating system used, 2) a (preferably version controlled and publicly accessible) code repository, and 3) a collection of supplementary log files that capture the data processing steps taken in producing each key result. It should provide a starting point for weather and climate scientists (and perhaps computational scientists more generally) looking to publish reproducible research and could be adopted as a minimum standard by relevant academic journals.

The procedure/standard was developed to be consistent with recommended computational best practices and seeks to minimize the time burden on authors, which has been identified as the most important barrier to publishing code. In particular, best practice dictates that at a minimum weather and climate scientists should be 1) writing data analysis scripts so they can rerun their analyses, 2) using version control to manage those scripts for backup and ease of sharing/collaboration, and 3) storing the details of their analysis steps in the global history attribute of their netCDF data files (or following an equivalent process for other file formats) to ensure the complete provenance of their data. To make their published results reproducible, it follows that the minimum an author would need to do is simply make those history attributes available (via log files) along with the associated code repository and a description of the software used to execute that code. The attainment of this minimum standard would involve a slight change to the workflow of many regular weather and climate scientists (e.g., most do not use version control); however, the standard has been designed to only require skills that can be learned very quickly (e.g., at a two-day Software Carpentry workshop).

While widespread adoption of this minimum standard would be a great starting point for reproducible research, it is worth noting that as a community we should ultimately aim much higher. By way of analogy, minimum standards in the construction industry ensure that buildings will not fall over or otherwise kill their inhabitants, but if everyone only built to those minimum standards our cities would be hugely energy inefficient. The proposed minimum standard for computational research ensures that published results are reproducible (which is a big improvement on the current state of affairs), but recreating workflows from the log files and daily code repositories of even just moderately complex analyses would be a tedious and time consuming process. Once comfortable with the skills and processes required to meet the minimum standard, authors should seek to go beyond them to improve the comprehensibility of their published computational results, in the same way that builders should strive for a five-star energy rating. The precise tools and methods used in this endeavor will vary from author to author; basic analyses might only require the inclusion of informative REAMDE files that explain in plain language how to execute the code, while others might choose to prepackage their code/software for ease of installation (e.g., for inclusion in the Python Package Index), make their code available to run on an online platform like RunMyCode.org, and/or provide informative flow diagrams exported from provenance tracking systems like VisTrails. As previously mentioned, it would not be appropriate to include these many varied (and often complex) options in any minimum standards, but they represent an excellent next step for scientists who have mastered the basics and will hopefully see more uptake as the computational competency of the community improves over time.

REFERENCES

Barnes, N., 2010: Publish your computer code: It is good enough. *Nature*, **467**, 753, doi:10.1038/467753a.

Bremges, A., and Coauthors, 2015: Deeply sequenced metagenome and metatranscriptome of a biogas-producing microbial community from an agricultural production-scale biogas plant. *Gigascience*, **4**, 33, doi:10.1186/s13742-015-0073-6.

Crooks, A. T., and A. B. Hailegiorgis, 2014: An agent-based modeling approach applied to the spread of cholera. *Environ. Modell. Software*, **62**, 164–177, doi:10.1016/j.envsoft.2014.08.027.

Easterbrook, S. M., 2014: Open code for open science? *Nat. Geosci.*, **7**, 779–781, doi:10.1038/ngeo2283.

Freire, J., and C. T. Silva, 2012: Making computations and publications reproducible with VisTrails. *Comput. Sci. Eng.*, **14**, 18–25, doi:10.1109/MCSE.2012.76.

Hannay, J. E., C. MacLeod, J. Singer, H. P. Langtangen, D. Pfahl, and G. Wilson, 2009: How do scientists develop and use scientific software? *Proc. ICSE Workshop on Software Engineering for Computational Science and Engineering*, Vancouver, BC, Canada, IEEE, 1–8, doi:10.1109/SECSE.2009.5069155.

Hunter, J. D., 2007: Matplotlib: A 2D graphics environment. *Comput. Sci. Eng.*, **9**, 90–95, doi:10.1109/MCSE.2007.55.

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

Irving, D. B., 2015a: A novel approach to diagnosing Southern Hemisphere planetary wave activity and its influence on regional climate variability: Supplementary metadata. Figshare, accessed 4 September 2015, doi:10.6084/m9.figshare.1385387.

——, 2015b: Data management in the ocean, weather and climate sciences. Zenodo, accessed 10 December 2015, doi:10.5281/zenodo.35095.

——, and I. Simmonds, 2015: A novel approach to diagnosing Southern Hemisphere planetary wave activity and its influence on regional climate variability. *J. Climate*, **28**, 9041–9057, doi:10.1175/JCLI-D-15-0287.1.

Jackson, M., 2015: How to cite and describe software. [Available online at http://software.ac.uk/so-exactly-what-software-did-you-use.]

Jacobs, C. T., A. Avdis, G. J. Gorman, and M. D. Piggott, 2014: PyRDM: A python-based library for automating the management and online publication of scientific software and data. *J. Open Res. Software*, **2**, e28, doi:10.5334/jors.bj.

Ketcheson, D., and A. Ahmadia, 2012: Optimal stability polynomials for numerical integration of initial value problems. *Commun. Appl. Math. Comput. Sci.*, **7**, 247–271, doi:10.2140/camcos.2012.7.247.

Kriegeskorte, N., A. Walther, and D. Deca, 2012: An emerging consensus for open evaluation: 18 visions for the future of scientific publishing. *Front. Comput. Neurosci.*, **6**, doi:10.3389/fncom.2012.00094.

Mayernik, M. S., S. Callaghan, R. Leigh, J. Tedds, and S. Worley, 2015a: Peer review of datasets: When, why, and how. *Bull. Amer. Meteor. Soc.*, **96**, 191–201, doi:10.1175/BAMS-D-13-00083.1.

——, M. K. Ramamurthy, and R. M. Rauber, 2015b: Data archiving and citation within AMS journals. *J. Climate*, **28**, 2529–2530, doi:10.1175/2015JCLI2222.1.

Merali, Z., 2010: Computational science: …Error. *Nature*, **467**, 775–777, doi:10.1038/467775a.

Momcheva, I., and E. Tollerud, 2015: Software use in astronomy: An informal survey. [Available online at http://arxiv.org/abs/1507.03989.]

Morin, A., J. Urban, and P. Sliz, 2012: A quick guide to software licensing for the scientist-programmer. *PLoS Comput. Biol.*, **8**, e1002598, doi:10.1371/journal.pcbi.1002598.

Nature, 2014: Code share. *Nature*, **514**, 536, doi:10.1038/514536a.

Peng, R. D., 2011: Reproducible research in computational science. *Science*, **334**, 1226–1227, doi:10.1126/science.1213847.

Piwowar, H. A., R. S. Day, and D. B. Fridsma, 2007: Sharing detailed research data is associated with increased citation rate. *PLoS One*, **2**, e308, doi:10.1371/journal.pone.0000308.

Potter, M., and T. Smith, 2015: Making code citable with Zenodo and GitHub. [Available online at www.software.ac.uk/node/1720.]

Prlić, A., and J. B. Procter, 2012: Ten simple rules for the open development of scientific software. *PLoS Comput. Biol.*, **8**, e1002802, doi:10.1371/journal.pcbi.1002802.

Sandve, G. K., A. Nekrutenko, J. Taylor, and E. Hovig, 2013: Ten simple rules for reproducible computational research. *PLoS Comput. Biol.*, **9**, e1003285, doi:10.1371/journal.pcbi.1003285.

Schmitt, C., S. Rey-Coyrehourcq, R. Reuillon, and D. Pumain, 2015: Half a billion simulations: Evolutionary algorithms and distributed computing for calibrating the Simpoplocal geographical model. *Environ. Plann.*, **42B**, 300–315, doi:10.1068/b130064p.

Stevens, B., 2015a: Rethinking the lower bound on aerosol radiative forcing. *J. Climate*, **28**, 4794–4819, doi:10.1175/JCLI-D-14-00656.1.

——, 2015b: Good scientific practice at the MPI-M. [Available online at www.mpimet.mpg.de/en/science/publications/good-scientific-practice.html.]

Stodden, V., 2010: The scientific method in practice: Reproducibility in the computational sciences. MIT Sloan Research Paper 4773-10, 33 pp., doi:10.2139/ssrn.1550193.

——, 2012: Reproducible research: Tools and strategies for scientific computing. *Comput. Sci. Eng.*, **14**, 11–12, doi:10.1109/MCSE.2012.82.

——, and S. Miguez, 2014: Best practices for computational science: Software infrastructure and environments for reproducible and extensible research. *J. Open Res. Software*, **2**, e21, doi:10.5334/jors.ay.

——, C. Hurlin, and C. Perignon, 2012: RunMyCode.org: A novel dissemination and collaboration platform for executing published computational results. [Available online at http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2147710.]

——, P. Guo, and Z. Ma, 2013: Toward reproducible computational research: An empirical analysis of data and code policy adoption by journals. *PloS One*, **8**, e67111, doi:10.1371/journal.pone.0067111.

Van Der Walt, S., S. C. Colbert, and G. Varoquaux, 2011: The NumPy array: A structure for efficient numerical computation. *Comput. Sci. Eng.*, **13**, 22–30, doi:10.1109/MCSE.2011.37.

Wilson, G., 2016: Software Carpentry: Lessons learned. *F1000Research*, **3**, 62, doi:10.12688/f1000research.3-62.v2.

——, and Coauthors, 2014: Best practices for scientific computing. *PLoS Biol.*, **12**, e1001745, doi:10.1371/journal.pbio.1001745.