Reproducible Research in the Mathematical Sciences[1]

D. Donoho, V. Stodden

Department of Statistics, Stanford University
Stanford, CA 94305

Department of Statistics, Columbia University
New York, NY 10027

1 Introduction

Traditionally, mathematical research was conducted via mental abstraction and manual symbolic manipulation. Mathematical journals published theorems and completed proofs, while other sorts of evidence were gathered privately and remained in shadow. For example, long after Riemann had passed away, historians discovered that he had developed advanced techniques for calculating the Riemann zeta function, and that his formulation of the Riemann hypothesis – often depicted as a triumph of pure thought – was actually based on painstaking numerical work. In fact Riemann's computational methods remained for decades after his death far ahead of what was available to others. This example shows that mathematical researchers have been 'covering their (computational) tracks' for a long time.

Times have been changing: on the one hand, Mathematics has grown into the so-called Mathematical Sciences, and in this larger endeavor, proposing new computational methods takes center stage, and documenting the behavior of proposed methods in test cases became an important part of research activity - witness current publications throughout the mathematical sciences, including statistics, optimization, and computer science. On the other hand, even pure mathematics has been affected by the trend towards computational evidence; Tom Hales' brilliant article *Mathematics in the Age of the Turing Machine* points to several examples of important mathematical regularities that were discovered empirically and have driven much mathematical research subsequently, the Birch and Swinnerton-Dyer conjecture being his lead example. This conjecture posits deep relationships between the zeta function of elliptic curves and the rank of elliptic curves, and was discovered by counting the number of rational points on individual elliptic curves in the early 1960's.

We can expect that over time, an ever-increasing fraction of what we know about mathematical structures will be based on computational experiments, either because our work (in applied areas) is explicitly about the behavior of computations, or because (in pure mathematics) the leading questions of the day concern empirical regularities uncovered computationally.

Indeed, with the advent of cluster computing, cloud computing, GPU processor boards and other computing innovations, it is now possible for a researcher to direct overwhelming amounts of computational power at specific problems. With the advent of mathematical programming environments like Mathematica, MATLAB, and Sage, it is possible to easily prototype algorithms which can then be quickly scaled up using the cloud. Such direct access to computational power is an irresistible force. Reflect for a moment on the fact that the Birch and Swinnerton-Dyer conjecture was discovered using the rudimentary computational resources of the early 1960's. Research in the mathematical sciences can now be dramatically more ambitious in scale and scope. This opens very exciting possibilities for discovery and exploration, as explained in EXPERIMENTAL APPLIED MATHEMATICS [VIII.XY].

The expected scaling up of experimental and computational mathematics is, at the same time, problematic. Much of the knowledge currently being generated using computers is not of the same quality as traditional mathematical knowledge. Mathematicians are very strict and demanding when it comes to understanding the basis of a theorem, the assumptions used, the prior theorems on which it depends, and the chain of inference that establishes the theorem. As it stands, the way evidence based on computations is typically published leaves 'a great deal to the imagination' and so computational evidence simply does not have the same epistemological status as a rigorously proved theorem.

Algorithms are becoming ever more complicated. Figure 1 shows the number of lines of

[1]We would like to thank Jennifer Seiler for outstanding research assistance.

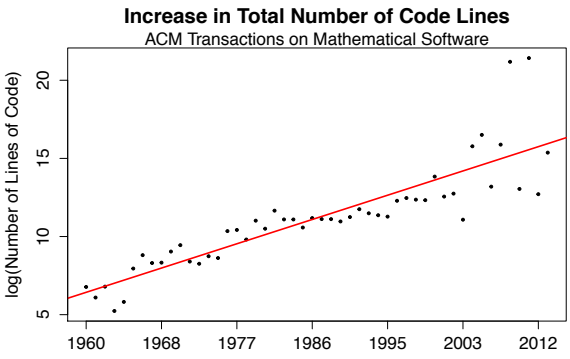

Figure 1: The number of lines of code submitted to ACM Transactions on Mathematical Software (TOMS), 1960–2012, on a log scale. The proportion of publications that submitted their code remained roughly constant at about 1/3, with standard error of about 0.12, and ACM TOMS consistently published around 35 articles each year.

code submitted to the ACM journal Transactions on Mathematical Software (TOMS) from 1960 through 2012. The number of lines submitted has increased exponentially, from 875 lines in 1960 to nearly 5 million in 2012, including libraries. The number of articles with code in TOMS is roughly constant; individual algorithms are requiring ever more code, even though modern languages are ever more expressive.

Algorithms are also being combined in ever more complicated processing pipelines. Individual algorithms of the kind that traditionally have been documented in journal articles increasingly represent only a small fraction of the code making up a computational science project. Scaling up projects to fully exploit the potential of modern computing resources requires complex workflows to pipeline together numerous algorithms, with problems broken into pieces and farmed out to be run on numerous processors, and the results harvested and combined in project-specific ways. As a result, a given computational project may involve much infrastructure not explicitly described in journal articles. In that environment, journal articles become simply *advertisements* - pointers to a complex body of software development, experimental outcomes, and analyses, in which there is really no hope that 'outsiders' can understand the full meaning of those summaries.

The computational era seems to be thrusting the mathematical sciences into a situation where mathematical knowledge in the wide sense, including also solidly based empirical discoveries, is broader and more penetrating but far less transparent and far less common 'property' than ever. Individual researchers report that over time they are becoming increasingly uncertain about what other researchers have done and about the strength of evidence underlying the results those other researchers have published.

The phrase *mathematical sciences* contains a key to improving the situation. The traditional laboratory sciences evolved, over hundreds of years, a set of procedures for enabling the *reproducibility of findings* in one laboratory by other laboratories. As the mathematical sciences evolve towards ever-heavier reliance on computation, they should likewise develop a discipline for documenting and sharing algorithms and empirical mathematical findings. Such a disciplined approach to scholarly communication in the mathematical sciences offers two advantages: (1) promoting scientific progress and (2) resolving uncertainties and controversies that spread a 'fog of uncertainty.'

2 Reproducible Research

We fully expect that, two decades from today, there will be widely accepted standards for communication of findings in computational mathematics. Such standards are needed so that computational mathematics research can be used and believed by others.

Today the raw ingredients that could enable such standards seem to be in place. *Problem solving environments* (PSEs) like MATLAB, R, IPython, Sage, and Mathematica, as well as open source operating systems and software, now enable researchers to share their code and data with others. While such sharing is not nearly as common as it should be, we expect that it soon will be.

Randall J. LeVeque describes well the moment we are living through; on the one hand, many computational mathematicians and compu-

tational scientists do not work reproducibly (LeVeque, 2006):

> Even brilliant and well intentioned computational scientists often do a poor job of presenting their work in a reproducible manner. The methods are often very vaguely defined, and even if they are carefully defined they would normally have to be implemented from scratch by the reader in order to test them. Most modern algorithms are so complicated that there is little hope of doing this properly ...

On the other hand, LeVeque continues, the ingredients exist:

> The idea of "reproducible research" in scientific computing is to archive and make publicly available all of the codes used to create the figures or tables in a paper in such a way that the reader can download the codes and run them to reproduce the results. The program can then be examined to see exactly what has been done. The development of very high level programming languages has made it easier to share codes and generate reproducible research. ... These days many algorithms can be written in languages such as MATLAB in a way that is both easy for the reader to comprehend and also executable, with all details intact."

While the technology needed for reproducible research exists today, mathematical scientists don't yet agree on exactly how to use this technology in a disciplined way. As we write this article there is a great deal of activity to define and promote standards for reproducible research in computational mathematics.

A number of publications address reproducibility and verification in computational mathematics; topics covered include: computational scale and proof checking, probabilistic model checking, verification of numerical solutions, standard methods in uncertainty quantification, and reproducibility in computational research. This is not an exhaustive account of the literature in these areas of course, merely a starting point for further investigation.

In this article we review some of the available tools that can enable reproducible research, and conclude with a series of "best practice" recommendations based on modern examples and research methods.

3 Script Sharing Based on PSEs

3.1 PSEs Offer Power and Simplicity

A key precondition for reproducible computational research is the ability for researchers to run the code that generated results in some published paper of interest. Traditionally this has been problematic. Often researchers were unprepared or unwilling to share code, and even if they did share it the impact was minimal, as the code depended on a specific computational environment (hardware, operating system, compiler, etc.) that others could not access. Traditionally, research laboratory computing environments often relied upon proprietary or hardware-specific software.

PSEs like R, Mathematica, and MATLAB have over the last decade dramatically simplified and uniformized much computational science.

Each PSE offers a high-level language for describing computations, often a language that is very compatible with standard mathematical notation. PSEs also offer graphics capabilities that make it easy to produce often quite sophisticated figures for inclusion in research papers. The researcher is gaining extreme ease of access to fundamental capabilities like matrix algebra, symbolic integration and optimization, and statistical model fitting; in many cases a whole research project, involving a complex series of variations on some basic computation, can be encoded in a few compact command scripts.

The popularity of this approach to computing is impressive. Figure 2 shows that the PSEs with the most impact on research - by citations - are the commercial closed-source packages Mathematica and MATLAB, which revolutionized technical computing in the 80's and 90's. However, these systems are no longer rapidly growing in impact; while the recent growth in popularity of R and Python is dramatic.

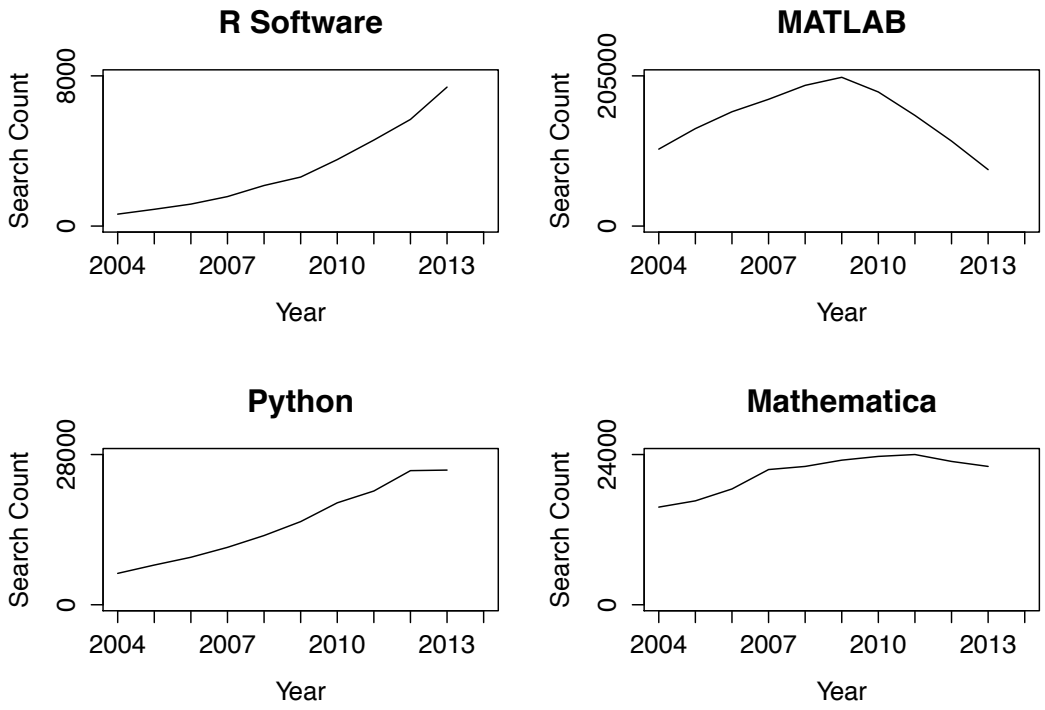

Figure 2: The total number of hits in Google Scholar for each of the four search terms: "R Software," MATLAB, Python, and Mathematica. The search was carried out for each year in the decade 2004–2013. Note that the y-axes are on different scales to show the increase or decrease in software use over time. R and Python are open source, whereas MATLAB and Mathematica are not.

3.2 PSEs Facilitate Reproducibility

As Leveque pointed out in the quote above, a side effect of the power and compactness of coding in PSEs is that reproducible research becomes particularly straightforward, as the original researcher can supply some simple command scripts to interested researchers, who then can rerun the experiment or variations of it privately in their own local instances of the relevant PSE.

In some fields authors of research papers are already heavily committed to a standard of reproducing results in published papers by sharing PSE scripts. In statistics, research papers often seek to introduce new tools that scientists can apply to their data. Many authors would like to increase the visibility and impact of such methodological papers, and are persuaded that a good way to do this is to make it as easy as possible for users to try the newly-proposed tools. Traditional theoretical statistics journal papers might be able to expect citations in the single or low double digits; there are numerous recent examples of articles which were supplemented by easy access to code and that obtained hundreds of readers and citations. It became very standard for authors in statistics to offer access to code using *packages* in one specific PSE, R. To build such a package, authors document their work in a standard LaTeX format, and bundle up the R code and documentation in a defined package structure. They post their package at CRAN, the Comprehensive R Archive Network. All R users can access the code from within R by simple invocations `require("package_name")` which direct R to locate, download, and install the package from CRAN. This process only takes seconds. Consequently, all that a user needs to know today to begin applying a new methodology is the name

of the package. CRAN offered 5519 packages as of May 8, 2014. A side effect of authors making their methodology available in order to attract readers, is of course that results in their original articles may become easily reproducible.[1]

3.3 Notebooks for Sharing Results

A notebook interface to a PSE stores computer instructions alongside accompanying narrative, which can include mathematical expressions, and allows the user to execute the code and store the output, including figures, all in one document. Because all the steps leading to the results are saved in a single file, notebooks can be shared online, which provides a way to communicate reproducible computational results.

The IPython Notebook, illustrated in figure 3, provides an interface to backend computations, for example in Python or R, that displays code and output, including figures, with mathematical notation typeset in LaTeX. An IPython Notebook permits the researcher to track and document the computational steps that generate results, and can be shared with others online using `nbviewer`; see for example `http://nbviewer.ipython.org/url/jakevdp.github.com/downloads/notebooks/XKCD_plots.ipynb`).

4 Open Source Software: A Key Enabler

PSEs and notebook interfaces are having a very substantial effect in promoting reproducibility, but they have their limits. They make many research computations convenient and easy to share with others, but ambitious computations often demand more capability than they can offer. Historically, this would have meant that ambitious projects have to be idiosyncratically coded and difficult to export to new computing environments.

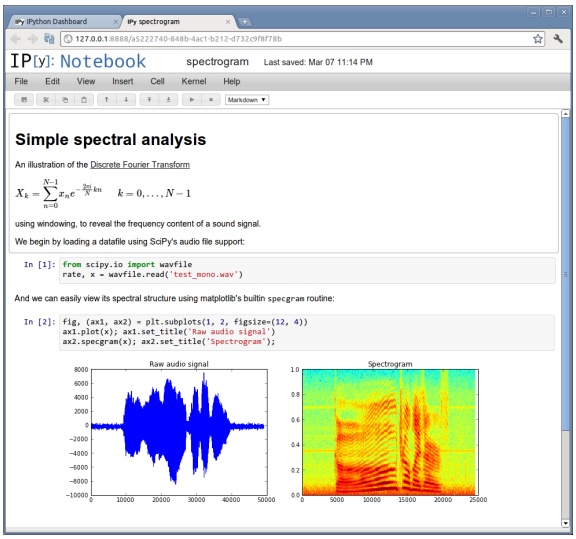

Figure 3: Snapshot of the IPython Interactive Notebook.

The open source revolution has largely changed this. Today, it is often possible to develop all of an ambitious computational project using code that is freely available to others. Moreover, this code can be hosted on an open source operating system (Linux) and run within a standard virtual machine that hides hardware details. The open source 'spirit' also makes researchers more open to sharing code; attribution-only open source licenses may also allow them to do this while retaining some assurance that the shared code will not be misappropriated.

Several broad classes of software are now being shared in ways that we describe in this section. These various classes of software are becoming or have already become part of the standard approaches to reproducible research.

4.1 Fundamental Algorithms and Packages

In Tables 1–10 we consider some of the fundamental problems that underly modern computational mathematics, such as FAST FOURIER TRANSFORMS [II.FFT], LINEAR EQUATIONS [IV.NLA], and NONLINEAR OPTIMIZATION [IV.OPT], and

[1] In fields like statistics, code alone is not sufficient to reproduce published results. Computations are performed on datasets from specific scientific projects; the data may result from experiments, surveys or costly measurements. Increasingly data repositories are being used by researchers to share such data across the internet. Since 2010, arXiv has partnered with the Data Conservancy to facilitate external hosting of data associated with publications uploaded to arXiv. See for example `http://arxiv.org/abs/1110.3649v1`, where the data files are accessible from the paper's arXiv page. Such practices are not yet widespread but they are occurring with increasing frequency.

give examples of some of the many families of open-source codes that have become available for enabling high-quality mathematical computation. The tables include their inception date, their current release number, and the total number of citations these packages have garnered since inception.[2] The different packages within one table may offer very different approaches to the same underlying problem. As the reader can see, a staggering amount of basic functionality is being developed worldwide by many teams and authors in particular subdomains - and made available for broad use. The citation figures in the tables testify to the significant impact these enablers are having on published research.

Table 1: Dense linear algebra

Package	Date	Release	Cites
LAPACK	1992	3.4.2	7600
JAMA	1998	1.0.3	129
IT++	2006	4.2	14
Armadillo	2010	3.900.7	105
EJML	2010	0.23	22
Elemental	2010	0.81	51

Table 2: Sparse-direct solvers

Package	Date	Release	Cites
SuperLU	1997	4.3	317
MUMPS	1999	4.10.0	2029
Amesos	2004	11.4	104
PaStiX	2006	5.2.1	114
Clique	2010	0.81	12

Table 3: Krylov-subspace eigensolvers

Package	Date	Release	Cites
ARPACK	1998	3.1.3	2624
SLEPc	2002	3.4.1	293
Anasazi	2004	11.4	2422
PRIMME	2006	1.1	61

Table 4: Fourier-like transforms

Package	Date	Release	Cites
FFTW	1997	3.3.3	1478
P3DFFT	2007	2.6.1	14
DIGPUFFT	2011	2.4	17
DistButterfly	2013		27
PNFFT	2013		215

Table 5: Fast multipole methods

Package	Date	Release	Cites
KIFMM3d	2003		1780
Puma-EM	2007	0.5.7	32
PetFMM	2009		29
GemsFMM	2010		16
ExaFMM	2011		28

4.2 Specialized Systems

The packages tabulated in the last subsection are broadly useful in computational mathematics; it is perhaps not surprising that developers would arise to create such broadly-useful tools. We have been surprised to see the rise of systems which attack very specific problem areas and offer extremely powerful environments to formulate and solve problems in those narrow domains. We give three examples.

4.2.1 Hyperbolic partial differential equations

Clawpack is an open-source software package designed to compute numerical solutions to hyperbolic partial differential equations (PDEs) using a wave propagation approach. According to system's lead author, Randall J. LeVeque, "[t]he development and use of the Clawpack software implementing [high-resolution finite volume methods for solving hyperbolic PDEs] serves as a case study for a more general discussion of mathe-

[2]Data for citation counts collected via Google Scholar in August 2013. Note that widely used packages such as LAPACK, FFTW, ARPACK and Suitesparse are used in other software (e.g., MATLAB), which do not generate citations directly.

matical aspects of software development and the need for more reproducibility in computational research."

The package has been used in creating reproducible mathematical research. For example the figures for LeVeque's book, *Finite Volume Methods for Hyperbolic Problems*, were generated using Clawpack; instructions are provided for recreating those figures.

Clawpack is now a framework offering numerous extensions including PyClaw, with a Python interface to a number of advanced capabilities, and GeoClaw, developed for TSUNAMI MODELING [V.XY] and modeling of other geophysical flows. Apparently, the open source software practices enabled not only reproducibility but also code extension and expansion into new areas.

Table 6: PDE Frameworks

Package	**Date**	**Release**	**Cites**
PETSc	1997	3.4	2695
Cactus	1998	4.2.0	669
deal.II	1999	8.0	576
Clawpack	2001	4.6.3	131
Hypre	2001	2.9.0	384
libMesh	2003	0.9.2.1	260
Trilinos	2003	11.4	3483
Feel++	2005	0.93.0	405
Lis	2005	1.4.11	29

Table 7: Finite element analysis

Package	**Date**	**Release**	**Cites**
Code Aster		11.4.03	48
CalculiX	1998	2.6	69
deal.II	1999	8.0	576
DUNE	2002	2.3	325
Elmer	2005	6.2	97
FEniCS Project	2009	1.2.0	418
FEBio	2010	1.6.0	32

Table 8: Optimization

Package	**Date**	**Release**	**Cites**
MINUIT/MINUIT2	2001	94.1	2336
CUTEr	2002	r152	1368
IPOPT	2002	3.11.2	1517
CONDOR	2005	1.11	1019
OpenOpt	2007	0.50.0	24
ADMB	2009	11.1	175

Table 9: Graph partitioning

Package	**Date**	**Release**	**Cites**
Scotch	1992	6.0.0	435
ParMeTIS	1997	4.0.3	4349
kMeTIS	1998	1.5.3	3449
Zoltan-HG	2008	r362	125
KaHIP	2011	0.52	71

Table 10: Adaptive mesh refinement

Package	**Date**	**Release**	**Cites**
AMRClaw	1994	4.6.3	4800
PARAMESH	1999	4.1	409
SAMRAI	1998		185
Carpet	2001	4	579
BoxLib	2000		155
Chombo	2000	3.1	198
AMROC	2003	1.1	342
p4est	2007	0.3.4.1	227

4.2.2 Parabolic and Elliptic PDEs: DUNE

The Distributed and Unified Numerics Environment (DUNE) is an open-source modular software toolbox for solving PDEs using grid-based methods. It was developed by Mario Ohlberger and other contributors and supports the implementation of methods such as finite elements, finite volumes, finite differences, and discontinuous Galerkin methods.

DUNE was envisioned to permit the integrated use of both legacy and new libraries. The soft-

ware uses modern C++ programming techniques to enable very different implementations of the same concepts (i.e. grids, solvers, linear algebra, etc.) using a common interface with low overhead, meaning that DUNE prioritizes efficiency in scientific computations and supports high-performance computing applications. DUNE has a variety of downloadable modules including various grid implementations, linear algebra, quadrature formulas, shape functions, and discretization modules.

DUNE is based on the following main principles: the separation of data structures and algorithms by abstract interfaces; the efficient implementation of these interfaces using generic programming techniques; and reuse of existing finite element packages with a large body of functionality. The finite element codes UG, ALBERTA, and ALUGrid have been adapted to the DUNE framework, showing the value of open source development not only for reproducibility but for acceleration of discovery through code reuse.[3]

4.2.3 Computer-Aided Theorem Proving

Computer-aided theorem proving [VII.xy] has been making extremely impressive strides in the last decade. This rests ultimately on the underlying computational tools which are openly available and which a whole community of researchers is contributing to and using. Indeed, one can only have justified belief in a computationally-enabled proof with transparent access to the underlying technology and broad discussion.

There are broadly speaking two approaches to computer-aided theorem proving tools in experimental mathematics. The first type encompasses machine-human collaborative proof assistants and interactive theorem proving systems to verify mathematics and computation, while the second type includes automatic proof checking which occurs when the machine verifies previously completed human proofs or conjectures.

Interactive theorem proving systems include coq, Mizar, HOL4, HOL Light, Isabelle, LEGO, ACL2, Veritas, NuPRL, and PVS. Such systems have been used to verify the Four Color Theorem and to reprove important classical mathematical results. Thomas Hales' Flyspeck project is currently producing a formal proof of the Kepler conjecture, using HOL Light and Isabelle. The software produces machine-readable code that can be re-used and repurposed into other proof efforts. Examples of open-source software for automatic theorem proving include E and Prover9/Mace 4.

[3]See also FEniCS `http://fenicsproject.org` for another example of an open source finite element package.

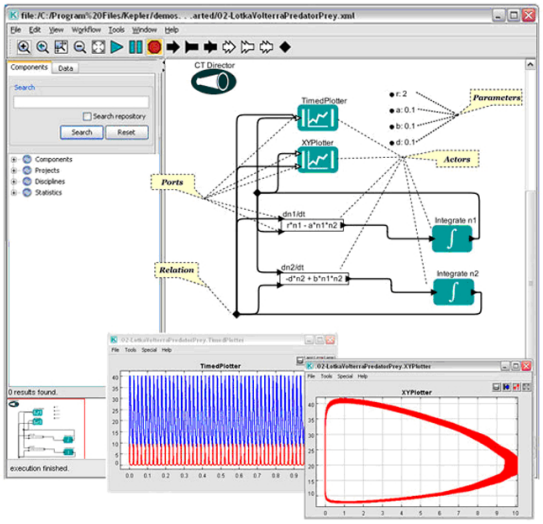

Figure 4: Example of the Kepler interface, showing a workflow solving the classic Lotka-Volterra predator prey dynamics model.

5 Scientific Workflows

Highly ambitious computations today often go beyond single algorithms to combine different pieces of software in complex pipelines. Moreover, modern research often considers a whole pipeline as a single object of study and makes experiments varying the pipeline itself. Experiments involving many moving parts that must be combined to produce a complete result are often called *workflows*.

Kepler is an open source project structured around *scientific workflows* – "an executable representation of the steps required to generate results," or the capture of experimental details that permit others to reproduce computational findings.

Kepler provides a graphical interface that al-

lows users to create and share these workflows. An example of a Kepler workflow is given in Figure 4, solving a model of two coupled differential equations, and plotting the output. Kepler maintains a Component Repository where workflows can be uploaded, downloaded, searched and shared with the community or designated users, and it contains a searchable library with more than 350 processing components. Kepler operates on data stored in a variety of formats, locally and over the internet, and can merge software from different sources such as R scripts and compiled C code by linking in their inputs and outputs to perform the desired overall task.

6 Dissemination Platforms

Dissemination platforms are websites that serve specialized content to interested visitors. They offer an interesting method to facilitate reproducibility; we describe here the platforms Image Processing OnLine (IPOL) project and ResearchCompendia.org.

IPOL is an open source journal infrastructure developed in Python that publishes relevant image processing and image analysis algorithms. The journal peer reviews article contributions, including code, and publishes accepted papers in a standardized format that includes:

1. a manuscript containing the detailed description of the algorithm, its bibliography, and documented examples;
2. a downloadable software implementation of the algorithm;
3. an online demo, where the algorithm can be tested on data sets, for example images, uploaded by the users;
4. an archive containing a history of the online experiments.

Figure 5 displays these components, for a sample IPOL publication.

ResearchCompendia, which one of the authors is developing, is an open source platform designed to link the published article with the code and data that generated the results. The idea is based on the notion of a "research compendium" – a bundle including the article and the code and data needed to recreate the findings. For a published paper, a webpage is created that links to the article and provides access to code and data as well as meta-data, descriptions and documentation, and code and data citation suggestions. Figure 6 shows an example compendium page.

ResearchCompendia assigns a Digital Object Identifier (DOI) to all citable objects: code, data, compendium page, in such a way to enable bi-directional linking between related digital scholarly objects, such as the publication and the data and code that generated its results (see `http://www.stm-assoc.org/2012_06_14_STM_DataCite_Joint_Statement.pdf`). DOIs are established and widely used unique persistent identifiers for digital scholarly objects. There are other PSE-independent methods of sharing such as GitHub.com (which can now assign DOIs to code `https://guides.github.com/activities/citable-code`) and supplementary materials on journal websites. A DOI is affixed to a certain version of software or data that generates a certain set of results. For this reason among others, VERSION CONTROL [VIII.VC §2] for scientific codes and data is important for reproducibility.[4]

7 Best practices for reproducible computational mathematics

Best practices for communicating computational mathematics have not yet become standardized. The workshop "Reproducibility in Computational and Experimental Mathematics", held at the Institute for Computational and Experimental Research in Mathematics (ICERM) at Brown University in 2012, recommended the following for every paper in computational mathematics.

- A precise statement of assertions made in the paper.
- A statement of the computational approach, and why it constitutes a rigorous test of the hypothesized assertions.

[4] Other reasons include: good coding practices enabling re-use; assigning explicit credit for bug fixing and code extensions or applications; efficiency in code organization and development; and the ability to join collaborative coding communities such as GitHub.com.

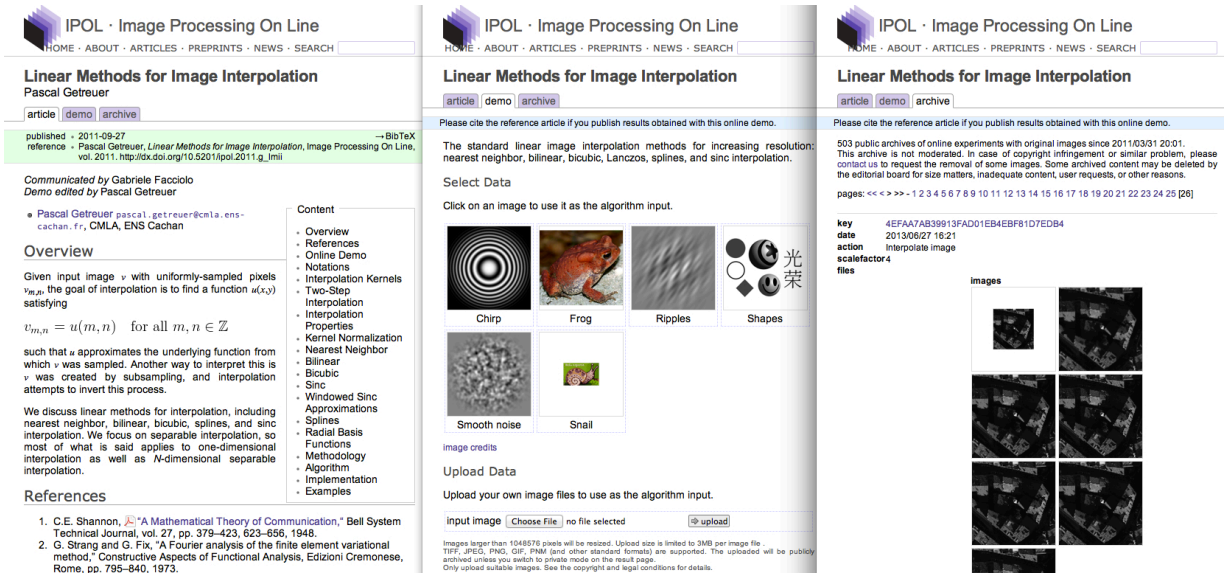

Figure 5: An example IPOL publication. The three panels from left to right include the manuscript, the cloud-executable demo, and the archive of all previous executions.

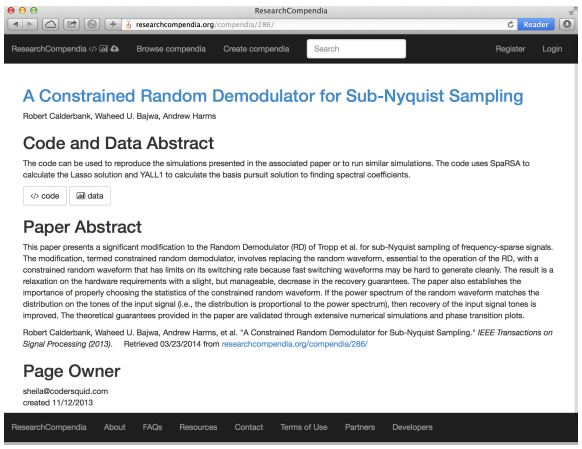

Figure 6: An example compendium page on ResearchCompendia.org. The page links to a published article and provides access to the code and data that generated the published results.

- Complete statements of, or references to, every algorithm employed.
- Salient details of auxiliary software (both research and commercial software) used in the computation.
- Salient details of the test environment, including hardware, system software and the number of processors utilized.
- Salient details of data reduction and statistical analysis methods.
- Discussion of the adequacy of parameters such as precision level and grid resolution.
- Full statement (or at least a valid summary) of experimental results.
- Verification and validation tests performed by the author(s).
- Availability of computer code, input data and output data, with some reasonable level of documentation.
- Curation: where are code and data available? With what expected persistence and longevity? Is there a site for future updates, e.g. a version control repository of the code base?
- Instructions for repeating computational experiments described in the paper.
- Terms of use and licensing. Ideally code and data "default to open," i.e. a permissive reuse license, if nothing opposes it.

- Avenues of exploration examined throughout development, including information about negative findings.

- Proper citation of all code and data used, including that generated by the authors.

These guidelines can, and should, be adapted to different research contexts but the goal is to provide readers with the information (such as meta-data including parameter settings and workflow documentation), data, and code, they require to independently verify computational findings.

8 The Outlook

The recommendations of the ICERM workshop listed in the previous section are the least we would hope for today. They commendably propose that authors give enough information so that readers can understand at some high level what was done.

They do not actually require sharing of all code and data in a form that allows precise re-execution and reproduction of results. Hence, these recommendations are very far from where we hope to be in twenty years.

One can envision a day when every published research document will be truly reproducible in a deep sense, where others can repeat published computations utterly mechanically. The reader of such a reproducible research article would be able to deeply study any specific figure: for example, viewing the source code and data which underlie the figure, recreating the original figure from scratch, examining input parameters that define this particular figure and even changing their settings, to study the effect on the resulting figure.

Reproducibility at this ambitious level would enable more than just individual understanding – it would enable meta-research. Consider the "dream applications" mentioned in Gavish and Donoho (2012), where robots automatically crawl through, reproduce and vary research results. Reproducible work can be automatically extended and generalized: it can be optimized, differentiated, extrapolated and interpolated. A reproducible data analysis can be statistically bootstrapped to automatically place confidence statements on the whole analysis.

Coming back to earth, what is likely to happen in the near future? We confidently project increasing computational transparency and increasing computational reproducibility in coming years. We suppose that PSEs will continue to be very popular, and authors will increasingly share their scripts and data, if only to attract readership. Specialized platforms like Clawpack and Dune will come to be seen as standard platforms for whole research communities who will naturally then be able to reproduce work in those areas. We expect that as the use of cloud computing grows and workflows become more complex, researchers will increasingly document and share the workflows that produce their most ambitious results. We expect that code will be developed on common platforms and will be stored in the cloud, enabling the code to run for many years after publication.

We expect that over the next two decades such practices will become standard, and will be based on tools of the kind discussed in this article. The direction of increasing transparency and increasing sharing seem clear, but it's still unclear which combinations of tools and approaches will come to be standard.

Further Reading

1. Hales, T., 2013 Mathematics in the Age of the Turing Machine. `http://arxiv.org/abs/1302.2898`. To appear in Turing's Legacy, ASL Lecture Notes in Logic, editor Rodney G. Downey.
2. LeVeque, R., 2006 Wave propagation software, computational science, and reproducible research. *Proceedings of the International Congress of Mathematicians* Madrid, August 22-30: invited lectures.
3. Bailey, D. H., Borwein, J., and Stodden, V. 2013 Set the Default to 'Open' *Notices of the AMS* June/July, 679–680.
4. AMS Workshop 2011 Reproducible Research: Tools and Strategies for Scientific Computing. July 13-16. `http://stodden.net/AMP2011`
5. Donoho, D., Maleki, A., Shahram, M., Ur Rahman, I., and Stodden, V. 2009 Reproducible Research in Computational Harmonic Analysis. *IEEE Computing in Science and Engineering* **11** (1), 8–18.
6. Stodden, V. 2009 The Legal Framework for Reproducible Scientific Research: Licensing and

Copyright. *Computing in Science and Engineering* **11** (1), 35–40.

7. Stodden, V. 2009 Enabling Reproducible Research: Licensing for Scientific Innovation. *International Journal of Communications Law and Policy* **13**, 1–25.
8. Stodden, V., Guo, P., and Ma, Z. 2013 Toward Reproducible Computational Research: An Empirical Analysis of Data and Code Policy Adoption by Journals. *PLoS ONE* **8** (6).
9. Gentleman, R., and Temple Lang, D. 2007 Statistical analyses and reproducible research *Journal of Computational and Graphical Statistics* **16**, 1–23.
10. Buckheit, J., and Donoho, D. 1995 Wavelab and reproducible research. In: Antoniadis A, editor. Wavelets and Statistics. New York, NY: Springer, 55–81.
11. Gavish. M., and Donoho, D. 2012 Three Dream Applications of Verifiable Computational Results. *Computing in Science and Engineering* 14(4), 26–31