Running a Reproducible Research Journal, with Source Code Inside

Nicolas Limare

ICERM Workshop — December 10-14, 2012

My experience with reproducible research is based on Image Processing On Line (IPOL), a journal I contributed to create in 2010 and still co-manage. IPOL is online at http://www.ipol.im/ and defines itself as "*[a] journal of image processing and image analysis. Each article contains a text describing an algorithm and source code, with an online demonstration facility and an archive of online experiments. The text and source code are peer-reviewed and the demonstration is controlled.*"

Two Years of a "Reproducible Research Journal"

In two years, IPOL published 40 articles by authors from a dozen of collaborating research labs. Each article comes with the main PDF manuscript, an implementation of the algorithm in C/C++ and a web interface to test this implementation in real time on user-submitted data, everything under free licenses and open access policies[1].

This journal started as an experiment because we knew no place to publish, share and test research software in a manner similar to the way research papers are handled in traditional journals. It grew into a stable project involving multiple research labs in the collaboration. I summarize hereafter the rationale behind the journal, some experience gathered while the concept was being refined over these two years, and a few critical points which would help publishing research software.

No Computational Science Without Software

"Reproducibility" has different meanings and implications in different research communities, depending on what is to be reproduced. I have the impression that the main message so far has been that the results published in a research paper must be reproducible, *ie* that independent researcher must be able to obtain the same results from identical data and computation.

But I am working in a field where we do not publish "results". A substantial part of image processing research is the elaboration of "methods", or algorithms. These algorithms are exposed in research articles and illustrated with some computational results as illustrations, trying to convince the audience of the qualities of this algorithm. But the science here is not in the results, it is in the algorithms. Signal processing might not be an exception here, I think econometrics also produce methods, and probably other fields too.

Then, our algorithms are rarely, if ever, completely specified by the description given if articles. Missing parameters is the first problem, pre- and post-processing also matter but are rarely detailed.

[1] For example, the latest article published as of December 8th, 2012, is online at http://www.ipol.im/pub/art/2012/g-ace/ with PDF and source code, and the algorithm can be tested at http://demo.ipol.im/demo/g_ace/

Articles chiefly focus on the mathematical objects making the algorithm, but not on their numerical counterpart embodied in the implementation. Computational efficiency can also sometimes make all the difference between two algorithms, and this efficiency is highly dependent on implementation details. In short, the source code is the ultimate documentation on an algorithm, and as such we consider no algorithm should be published and described in a classic text-formulas-and-figures manuscript without providing a reference implementation.

Moreover, software serves two additional purposes. As a computational science linked to human perception, image processing methods not only based on math but also evaluated on data, and no one is convinced by a couple of test images. Providing an implementation of every published algorithm allows readers/users to process their own data and verify the robustness of the method. And, last not least, the source code is the natural medium to share our computational research, just like demonstrations are provided and shared for every proven theorem.

By publishing implementations for every computational method exposed in papers, IPOL papers should not only be reproducible, but also verifiable and reusable.

Publishing Software

Publishing software, in the sense that it is released by a research journal like other articles are, is more than just making some program available as a supplementary material. Simply collecting source code without a controlled and enforced editorial policy doesn't define a research journal, only a code repository, which are two complementary but different things. If the software is published for the documentation and evaluation of algorithms, then it implies a few editorial rules.

The software must be provided in source form, otherwise it does not document internal details of the algorithm. This implies that the code must be readable and sufficiently documented to allow other researchers to follow the algorithm and verify that the implementation matches the description in every detail. In other words, software should be prepared for readers just like articles are edited from scribbled drafts to clean and detailed manuscripts following instructions from the journal. Of course, most authors are not prepared to do the same with their code, they have never been asked to. But it seems this is only a matter of training and culture, and can be gradually solved over time with good instructions.

The software must also be usable in other researchers' computing environment, otherwise it can not be tested and compared. This implies at least Windows/UNIX portability, and the free availability of every building block. It is achieved in IPOL with a requirement to be provided in C/C++ source code, portable, using only a few external libraries, and under an open license[2]. This policy reflects what we believe is possible in image processing and desirable for research software, but it needs to be adapted to the needs and habits of every research community: cryptography, particle physics and genetics, for example, will have different requirements.

Reviewing Software

The editorial rules covering software have to be controlled to guarantee the respect of the journal policy, and this is done by reviewing the software just like any article manuscript. Reviewers completely read the source code, or at least the algorithmic part of it, and verify it is clear and match the description given in the manuscript, as well as the respect of journal guidelines. They also run the software and use it on carefully chosen input data to check it performs as announced. This validation is simplified by the web interface provided by IPOL for every algorithm and code;

[2]IPOL Software Guidelines are detailed at https://tools.ipol.im/wiki/ref/software_guidelines/.

with this facility, reviewers and editors can run their tests and they can see tests run by others, because every experiment conducted over the web is archived and browsable.

The software review is a manual process, not a formal software validation. It provides no guarantee that the code is correct, only that informed researchers have examined and approved it. Bugs have been found after the publication of some IPOL articles, and they were handled just like any error found in an article, by publishing an erratum and an updated version approved by the editors.

This is not a problem because publishing the code in a research journal is not a software release. The article should not be mistaken for the software project home page, and the journal must not be confused with an attempt to compile a definitive software library. Codes will keep being developed and improved out of the journal, like other research efforts don't freeze when they are published.

We had no major problem finding reviewers willing to examine the source code, usually young researchers proficient in programming. The quality and level of detail of the reviews varied a lot according to the competence and interest of the reviewers. During the review process, changes and improvements were always asked and obtained by the reviewers. In that sense, the peer-review help improve the software quality, like some sort of pair-programming.

Strategy and Impact

The publishing record of IPOL shows that it is possible to *require* authors to provide software as a journal policy. However, authors still need to be convinced to do the extra work since this task is usually not required for publishing in other journals. Our solution to this problem is not to replace traditional journals; IPOL articles are complementary to classic papers published without code, and one can even publish in IPOL an implementation of a classic algorithm created by other researchers. Our partnership with SIIMS\footnote{SIIMS is the SIAM Journal on Imaging Science, see http://www.siam.org/journals/siims.php.) is a good illustration: authors publish their work in a well-known classic journal, and with a some extra effort they can have a complementary article in IPOL with the source code. Both articles are interlinked, authors get two references, IPOL has some endorsement from a recognized journal, and SIIMS gets some software verified and available with the articles.

We observed that some of the codes published in IPOL have been reused in other projects. Some other codes were mentioned in software development communities as examples and documentations about an algorithm or technique, and sometimes the code published in IPOL becomes the reference implementation which would have been missing otherwise. These are examples of the impact of publishing software. This impact can also be measured by the number of source code download, or by looking at the demo archive size. Popular algorithms have thousands of users, and the diversity of input images reveals interest from diverse research fields and industries. But we can not go further and measure an academic impact in term of citations; IPOL is probably still too small, too young, too unusual, or all these together.

What's Missing? What's Next?

Publishing research software will require a constant effort to convince potential authors until this principle is adopted by big players and journals, and researchers get used to it and anticipate the life cycle of their code early in their software developments.

The academic publishing industry also needs to reform the assumption that and article is a PDF file. This assumption is everywhere in the review, publishing and indexation tools and services. A modern research article should be a set of files, including but not limited to PDF files emulating

paper articles in a digital form. A common standard could be established to pack together all the files constituting an article.

Every time we publish a code using a software library, there is a risk of this library becoming obsolete, unmaintained or unusable. We need standardized and stable APIs for every essential family of libraries[3], so that core functions can be provided by different libraries and new ones can be developed to replace those hit by bit rot.

To protect research software against unwarranted obsolescence, we also need properly specified programming languages. Languages only defined by their current implementation, and updated twice a year[4], put every code at risk of being unusable in the short term. A stable, complete and formal definition of this language would permit alternative implementations or compatibility layers to take the place of a failing provider.

Software should also be stored and identified by reliable, persistent, vendor-independent handlers. The DOI system could be expanded with the collaboration of software hosting services to add a notion of version, so that one can say "*our software was built using the* `libfoo` *library version 42*" and provide a link that will *always* point a location where `libfoo` can be downloaded in version 42 and later developments can also be found.

Some work is needed, too, on research data storage solutions not restricted to an institution, a country or a research topic. We would also like to see more software testing tools and services, to help authors identify improvements needed in their code and relieve reviewers from all the validation tasks that can be automated.

Finally, patent regulation is a constant source of annoyance, with international heterogeneity and inconsistent legal opinions on whether publishing research implementations of an algorithm described in a patent application is a patent infringement. If software patents are here to stay, we would at least want a clear, complete and worldwide exemption of patent regulation for research and experimentation purposes.

Nicolas Limare
nicolas.limare@cmla.ens-cachan.fr
http://limare.perso.math.cnrs.fr/
CMLA, CNRC UMR 8536, ENS Cachan, France

References

[1] Image Processing On Line (IPOL), ISSN 2105-1232, http://dx.doi.org/10.5201/ipol, http://www.ipol.im/.

[2] Limare, Nicolas, and Jean-Michel Morel, "The IPOL Initiative: Publishing and Testing Algorithms on Line for Reproducible Research in Image Processing," *Procedia Computer Science* (2011) Proceedings of the International Conference on Computational Science (ICCS 2011), http://dx.doi.org/10.1016/j.procs.2011.04.075.

[3] Limare, Nicolas, "Reproducible Research, Software Quality, Online Interfaces and Publishing for Image Processing" (PhD diss., École Normale Supérieure de Cachan, 2012).

[4] Limare, Nicolas, Laurent Oudre, and Pascal Getreuer, "IPOL: Reviewed Publication and Public Testing of Research Software," *8th IEEE International Conference on eScience, First Workshop on Maintainable Software Practices in e-Science* (2012).

[3] The BLAS standard is an example that should inspire us to establish a vendor-agnostic interface for every basic building bloc, such as (for image processing) Fourier transform, convolutions and blur, morphology operators, ...

[4] MATLAB is the most prominent example.