Open science decoded

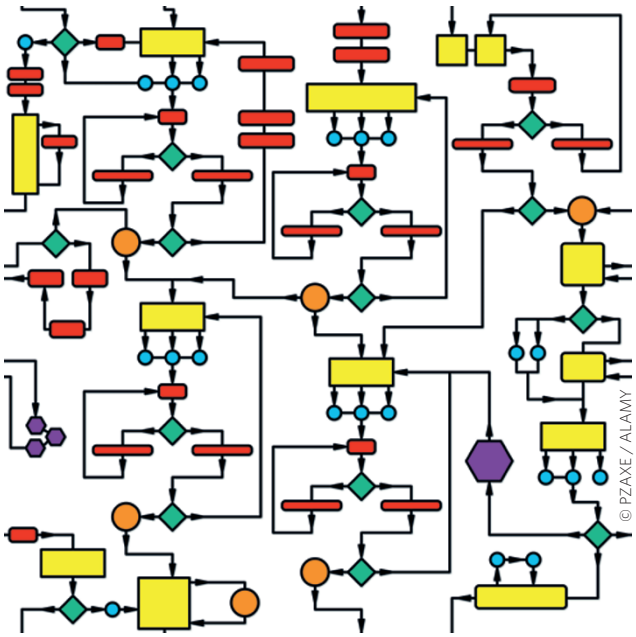
Tony Hey and Mike C. Payne

Granting access to publications and data may be a step towards open science, but it's not enough to ensure reproducibility. Making computer code available is also necessary — but the emphasis must be on the quality of the programming.

In February 2013, the US Office of Science and Technology Policy, in the Executive Office of the President, issued a memorandum requiring that Federal agencies investing in research develop clear policies to support increased public access to the results of their research (http://go.nature.com/OhZNcR). The memo stipulated that "such results include peer-reviewed publications and digital data", where digital data is defined as material necessary to validate other scientists' research. It was soon followed by similar declarations from the Global Research Council in May (http://go.nature.com/qHlgxU) and from the G8 Science Ministers in June2013(http://go.nature.com/TeKjLs).

Clearly, there is now increasing global momentum towards 'open science'. This necessarily requires not only open access to research publications, but also to the metadata and data required to validate and make sense of the results of research. And improving the comprehensibility and reproducibility of computational science is an important step in this endeavour. One approach is through executable papers, which not only provide the text, tables and figures of a conventional paper, but also offer access to the software, data and computer environment used to produce the results. There are now several interesting attempts at providing support for executable papers[1,2]. However, there are still many research challenges in this area and it remains to be seen whether such approaches will become widely adopted.

One might argue that the physics community is leading the charge in response to these obstacles. In the 1980s, forty years after the birth of the electronic computer, Nobel Prize winner Ken Wilson and others argued for the importance of computational science for scientific exploration and discovery. Wilson even went so far as to call computational science the "third paradigm of science, supplementing theory and experimentation"[3]. And indeed, computer simulations are now used effectively to gain understanding of complicated physical systems that are either too complex to be solved analytically or inaccessible to experiment — or both.



© PZAXE / ALAMY

Reproducible research

Computational science presents a challenge to the traditional notion of scientific reproducibility. Several different programs written by different researchers can seek to explore the physics of the same complex system and these may use different algorithms and/or different numerical approximations. Exact numerical agreement between the results of the two different programs is therefore not to be expected, and in practice, reproducibility involves finding very similar quantitative results for key physical parameters. However, all too often, neither the source code nor enough detail of the precise computing environment is included with the published paper to enable other scientists to reproduce the results. In addition, of course, access to the particular high-end supercomputer that obtained the results of the simulation may not be possible. These issues form part of the challenge of 'reproducible research' for computational science[4].

One of the first scientists to recognize the need for reproducibility in computational science was the geophysicist Jon Claerbout. As early as 1990, he set a goal of reproducibility for all the (non-open-access) reports coming out of his Stanford Exploration Project, identifying reproducibility as "a way of organizing computational research that allows both the author and the reader of a publication to verify the reported results" (http://go.nature.com/FMw34h).

Twenty years later, in December 2012, the Institute for Computational and Experimental Research in Mathematics at Brown University hosted a workshop on reproducibility in computational and experimental mathematics. Out of this workshop came a list of the five best publication practices (http://go.nature.com/wtqOLn). The list declares that data and code should both be made available and accessible, in the sense that others can reproduce published findings. It underlines the importance of properly citing data that is not generated exclusively for the study in question. And it stipulates guidelines for granting publishers copyright permissions, and for organizing supplementary materials.

Following such best-practice guidelines in principle gives researchers the information needed to reproduce computational results using the same code and data used by the original researcher. However, there is a related but distinct problem of numerical reproducibility[4]. Round-off error can be exacerbated when simulations are scaled up to run on large parallel systems. This can be problematic for the original authors of the code — and even more so for researchers seeking to reproduce results

using independently written programs that run on different systems. It is clear that the computational science community as a whole still has a long way to go to change the culture of reporting their results.

The large and the small

Computational science, in the sense of performing computer simulations of complex systems, is only one aspect of computing in physics. For many research areas, analysis of the ever-increasing amount of experimental data would be impossible without the use of computers. For example, the raw data from NASA's MODIS satellite undergoes very significant computer processing to generate the 'data products' for analysis by geophysicists. In the case of the experiments at the Large Hadron Collider (LHC) at CERN, the ATLAS and CMS experiments not only must perform much pre-processing to reconstruct the individual particle tracks through their detectors but also need to develop a detailed simulation of their detectors before the physics analysis can begin.

The ATLAS and CMS experiments are examples of 'big science' projects in physics that have significant budgets for software development. For the ATLAS collaboration, the software is written by around 700 physicist postdoctoral researchers and graduate students, not by professional software developers. The software is written in C++ and multiple software modules are glued together using Python scripts. The ATLAS software has more than six million lines of code — 4.5 million written in C++ and 1.5 million in Python. A typical reconstruction project may have on the order of 500 modules. The software starts with raw data of the collision events from around 100 million readout channels and reconstructs the particle trajectories. The process needs a detailed understanding of the geometry, properties and efficiencies of the ATLAS detector. Such understanding requires a very complex Monte Carlo simulation of the detector. Finally, from the calculated values of the energy and momentum four-vectors of the tracks observed in the detector, each event can then be analysed to explore hypotheses of what was going on in the original collision. Each of the 3,000 or so physicists in the ATLAS collaboration is able to access the data from CERN using a global distributed-computing and storage infrastructure called the Worldwide LHC Computing Grid, built by the international particle physics community.

For a project as complex as the ATLAS experiment, serious software management and testing processes are required. Automated integration testing is implemented for each build of the software, sometimes for the whole software chain — from Monte Carlo generation to simulation, reconstruction and analysis. Once these tests have been executed, a 'release' of the build is given to another group to test the code further, by running long jobs, producing plots and making detailed comparison with reference data sets. For bug tracking, ATLAS uses JIRA, a proprietary product that is free for use in academic projects. Each observed difference is investigated and, when all of these have been acted on, the code can then be released for use by the whole collaboration. Version control of the source code is obviously a necessity, and ATLAS uses the open source Apache Subversion (SVN) version-control system. For the software releases, the ATLAS team needs to track over 2,000 packages and uses release management software developed by the collaboration. The CMS software system is similarly complex, but uses the GitHub repository and version-control system.

The twin issues of research reproducibility and open data are clearly very complex in the case of these large physics experiments. At the LHC, research reproducibility is addressed by having multiple experiments — like ATLAS and CMS — producing and analysing their own data using different software. The task of making LHC data available and useful to the public (or even other scientists) is challenging. The new CERN Open Data Portal is now making a start by releasing some data, together with the analysis tools and environment needed to make sense of the data.

On a smaller scale — in terms both of the software budget and the size of the collaboration — are several 'community' physics software projects. In the UK, such projects are exemplified by the Collaborative Computational Projects (CCPs) supported by a team at the Science and Technology Facilities Council (STFC). The team assists universities in developing, maintaining and distributing computer programs, and in promoting the best computational methods. Each CCP focuses on a specific area of research and they are funded by the UK Research Councils. There are probably more than 1,000 individual researchers and research students supported by the CCPs.

One CCP project is devoted to the study of the electronic structure of condensed matter. The application area is very broad and includes the study of metals, semiconductors, magnets and superconductors from microscopic quantum mechanical calculations. The research activities of the project encompass such highly topical areas as magnetoelectronics, photonics, nanotechnology, high-temperature superconductors and novel wide-bandgap semiconductors. The CASTEP code, for example, is a leading code for calculating the properties of materials from first principles. Using density functional theory, it can simulate a wide range of material properties including energetics, structure at the atomic level, vibrational properties and electronic response properties. The CASTEP source code is available free of charge to all UK academic research groups under the terms of an agreement between STFC, the commercial company BIOVIA (formerly Accelrys) and the UK Car–Parrinello Consortium. Remarkably, the commercial version of the code distributed by BIOVIA effectively comes directly from the academic code repository.

The original CASTEP code was recently rewritten by a small team of physics researchers using modern software-engineering techniques. The aim was to rewrite it as a parallel code, thus removing any distinction between the standard version and one fit to run on a high-performance computer. The code runs on a wide variety of computing platforms from stand-alone workstations, to parallel clusters, to high-end supercomputers. Unfortunately, as is so often the case, a full paper describing the CASTEP code revision in detail was never written. Due to the new modular structure of the revision, the entire CASTEP code can now be made to run efficiently on different hardware architectures by making modifications only in the low-level 'utility' modules.

At the bottom of the pyramid is what is sometimes referred to as long-tail science (http://go.nature.com/zIozIY). Big science only makes up a small fraction of published research. Every day, students and postdoctoral researchers are working away in small groups doing independent research. It is important to recognize that we are not just standing on the shoulders of a few giant projects, but also on those of a multitude of smaller groups of researchers. However, these researchers typically have very little training in software development and certainly do not have large budgets. FORTRAN code is still alive and well in the long-tail and many of these researchers use packages such as MATLAB and tools such Excel for data analysis (http://go.nature.com/1auEJj).

Open source is not a panacea

One can argue about the exact balance between the costs of hardware and software in running a computing centre, but there is no doubt that there are significant costs associated with developing and maintaining software. Our day-to-day experience of Google and Facebook services can all too

easily give rise to the misguided idea that excellent software is free. The open-source software movement further reinforces the idea that all software should be free. There are excellent examples of open-source software projects — such as those supported by the Linux, Apache and Eclipse foundations — and these are typically of high quality, fully documented and supported. Similarly, GitHub now has over three million users and supports over 16 million repositories, most of which contain open-source code. However, much of open-source scientific research software is often of poor quality, inefficient, undocumented and unmaintained. Producing high-quality software that is well documented, debugged and easy to install and use is expensive. Merely making scientific software open source is not a guarantee of quality.

Many developers of scientific software receive their training from other scientists rather than from a traditional computer-science software-engineering course. There is still a mismatch between some standard software-engineering environments and methodologies and the practice of working computational scientists developing parallel codes for high-performance computing systems[5]. However, as can be seen from the example of the ATLAS LHC experiment, significant parts of the physics research community certainly use many of the modern tools of software engineers to develop and manage their large code base.

Part of the problem is that the long-tail science researchers are often not well equipped to design, write, test, debug, install and maintain software — having only very basic training in programming. Enter 'software carpentry'. This movement was started by Greg Wilson in 1998 and has now evolved into a worldwide volunteer effort to raise standards in scientific computing. Wilson runs software-carpentry boot camps that focus on four aspects of computational competence[6]. A typical course introduces a handful of basic Unix shell commands, teaches students how to build simple Python programs step by step, emphasizes the benefits of version-control systems and extols the virtues of structured data. As Wilson notes, teaching such basic skills is not very interesting for computer scientists.

Support from above

A key problem in supporting research software development is that funding agencies in many countries do not view software development as an intellectual exercise worthy of a research grant. Instead, in their proposals, scientists have to focus on the research aspects of a particular scientific application and gloss over the fact that software will need to be developed to enable the science. However, both in the USA and the UK, the major research funding agencies have recently changed their policies on funding research software development.

The Advanced Cyberinfrastructure Division of the National Science Foundation (NSF) in the USA has produced a vision and strategy for supporting research software[7]. The plan is to address research software sustainability issues though a tiered approach. First, the NSF will form small groups to design software elements tailored for particular advances in science and engineering. Second, larger teams will be enlisted to build software infrastructure that is accessible to diverse scientific communities. The NSF intends to create hubs of excellence in software infrastructure and technologies. And finally, it plans to provide incentives for individuals and communities to build on existing software frameworks in their software development.

In the UK, the Research Councils have also recently changed their policy on software-development costs. In addition to providing support for the UK Software Sustainability Institute, the Engineering and Physical Sciences Research Council (EPSRC) now issues regular calls for proposals that are focused purely on either developing new and innovative software — adding novel functionality to existing software, or simply making widely used software packages more efficient and/or robust (http://go.nature.com/XFCWXy). The EPSRC also now offers personal fellowships specifically for individuals who specialize in software development. These are encouraging signs, but there is a long way to go before skilled developers of research software achieve parity of recognition with their partner research scientists. ❐

Tony Hey is at the eScience Institute at the University of Washington, Seattle, Washington 98195, USA. Mike C. Payne is in the Cavendish Laboratory at the University of Cambridge, Cambridge CB3 0HE, UK. e-mail: tony.hey@live.com; mcp1@cam.ac.uk

References

1. Nowakowski, P. *et al. Procedia Comput. Sci.* **4,** 608–617 (2011).
2. Koop, D. *et al. Procedia Comput. Sci.* **4,** 648–657 (2011).
3. Bell, C. G. *Comput. Sci.* **1,** 4–6 (1987).
4. Stodden, V. *et al. Setting the Default to Reproducible: Reproducibility in Computational and Experimental Mathematics* (ICERM, 2012); http://go.nature.com/nacwjm
5. Basili, V. R. *et al. IEEE Software* **25,** 29–36 (2008); http://go.nature.com/xTSILw
6. Wilson, G. *F1000Research* **3,** 62 (2014); http://go.nature.com/8V6ui9
7. *A Vision and Strategy for Software for Science, Engineering and Education* (NSF, 2012); http://go.nature.com/M29xjk

Programming revisited

Thomas C. Schulthess

Writing efficient scientific software that makes best use of the increasing complexity of computer architectures requires bringing together modelling, applied mathematics and computer engineering. Physics may help unite these approaches.

For half a century, the performance of computers has been improving exponentially. These impressive developments in scientific computing — which most scientists now take for granted — are documented by the Top500 project (www.top500.org). Since 1993 this website has been monitoring how the fastest computing systems solve dense linear equations with the high-performance LINPACK (HPL) benchmark and the sustained performance for HPL-type problems has been increasing roughly by three orders of magnitude per decade.

This improvement in performance is not limited to high-end supercomputers, but permeates all information technology infrastructures available to scientists today. For example, the current HPL performance of typical smartphones is comparable to that of Cray X-MP supercomputers in the late 1980s. Hard computational problems that originally required unique computing infrastructures become solvable within years on regular computers, and typically a decade or two later can be handled by commodity devices most people around the world carry in their pocket.