Coding your way out of a problem

Jeffrey M Perkel

Forays into information technology can make lab work easier to manage.

From the mid-1980s through the mid-1990s, Greg Wilson worked in high-performance scientific computing in Edinburgh, UK. His job was to help physicists migrate their homemade software onto massively parallel supercomputers. Along the way, he says, he realized he was doing more harm than good.

"The average scientist probably has one undergraduate course on programming and then no other training at all until they hit grad school and somebody says, 'Here [are] 100,000 lines of Fortran. There's a one-of-a-kind supercomputer. Good luck'," says Wilson. That, he says, is the computational equivalent of "taking somebody who just barely knows how to back a family car out of a driveway and sticking them in a Formula-1 race car on the highway, in the dark, in the rain. The results are the same: most people crash."

A wreck can put a serious dent in a scientific career. A mathematical error in a bit of borrowed code used to help solve crystal structures forced the Scripps Research Institute's Geoffrey Chang to retract five papers from *Science*, *Proceedings of the National Academy of Sciences of the United States of America* and *Journal of Molecular Biology*. "I've been devastated," Chang said in 2006 (ref. 1).

To prevent these kinds of accidents, Wilson developed an online programming course for scientists called Software Carpentry, which he ran first at US Los Alamos National Laboratory and later at the University of Toronto. In 2010 he left academia to devote himself to the project full time. Software Carpentry covers everything from basic programming to version control and testing. "As far as I am concerned, a program is a piece of experimental apparatus," Wilson says, and it should be treated as such. Yet many scientists, especially in the life sciences, lack the skills to use, let alone create, these tools. That is a pity, because from anatomy to zoology, there is no shortage of problems that could be eased with a little computational know-how.

Fortunately, there are projects that can make scientists' lives easier without confounding their results (**Box 1**). Here is a series of stories to inspire, starting with those that require the least expertise.

Electronic lab notebooks

In the easiest cases, no programming at all is necessary, just a suitable marriage of software and hardware, plus some convenient timing. When Jeremy Baumberg took on his current role as director of the NanoPhotonics Centre at the Cavendish Laboratory at the University of Cambridge, he used the opportunity to transition to a linked system of electronic laboratory notebooks.

Handwritten lab notebooks are usually chaotic and always unsearchable, facts that were a continuing source of frustration to Baumberg. "Your lab book is one of the most precious things you have as a scientist," he says. "It's your entire knowledge and memory; it's your know-how." Yet frequently, that know-how is lost when people leave a lab and the remaining scientists cannot read their notebooks. Electronic lab notebooks solve some problems, but the ones Baumberg investigated had overly structured templates that would not suit all of his students. So he started searching for a different approach.

Around 2007, Baumberg, then at the University of Southampton, began noticing tablet personal computers, laptop computers (such as the Hewlett-Packard TouchSmart tm2 notebook PC and Lenovo's ThinkPad X Series) that let users write on the screen with a stylus like with a pen on a clipboard. Previously, these computers were slow, clunky and error-prone. But through what he calls "an iterative improvement," their handwriting capture technology grew reliable. That, Baumberg says, was critical: "If you lose the ability to actually sketch or manipulate equations, you're cutting yourself off from your scientific creativity in an enormously bad way," he says.

Baumberg realized these tablets could form the backbone of a paperless lab. But he needed software to make it work. Enter Microsoft OneNote, a free-form notes

Cameron Neylon used his programming skills to code a bridge between his instrumentation and his electronic lab notebook.

BOX 1 ADVICE FROM THE PROS

Do not be afraid. A computer is just a tool, so do not be afraid of it, says C. Titus Brown, a Michigan State University researcher who teaches programming for biologists. Brown does not recommend users necessarily "program fearlessly" on mission-critical computers; instead, use a nonessential machine or rent an Amazon virtual PC. "There really is nothing you can do to break that computer."

Get started. Start with your language's or program's documentation; it is often chockablock with examples and tutorials. Software Carpentry (http://software-carpentry.org/), run by Greg Wilson, offers online video tutorials aimed at scientists, whereas SEQanswers.com, BioStar (http://biostar.stackexchange.com/) and The Hacker Within (http://hackerwithin.org/thw/) provide resources and forums for answering questions. Also recommended is *Practical Computing for Biologists*[3].

Optimize the bottlenecks. A good algorithm is both accurate and efficient. That is especially true in biology. Richard Bowman, a graduate student at University of Glasgow, Scotland, who has written applications to control optical tweezers, cites the 80/20 rule: 20% of the code performs 80% of the work. "If you spend enough time to write that tiny bit of code really well, then it will pay off hugely," he says.

Form collaborations. Sean Davis, a staff scientist in the genetics branch at the US National Cancer Institute and a member of the core development team on Bioconductor (an open-source bioinformatics project), has seen many 'newbies' graduate to fairly sophisticated tasks. Nevertheless, making mistakes is easy, and someone who can spot errors and serve as a sounding board can speed even straightforward projects. "It's never a bad idea to have a collaborator," he says.

Treat computation like an experiment. "Don't trust what the computer spits out just because the computer spits it out," says Brown. Instead, run a search that should yield a negative answer or vary parameters to see how the answer changes when it should not. "Do you get the same answer? If you do, that means it's more robust."

Do not reinvent the wheel. Quite a bit of programming, says Brown, is "rote, cookbook-style stuff." There is plenty of free code available, just copy and paste; the trick is tweaking it. "One of the most important skills to learn is how to just take the plethora of open-source solutions out there and apply them to your own problem."

Keep good records. If there is one lesson Wilson wants students to get out of Software Carpentry, it is version control. "It's the equivalent of a lab notebook for computational work," he says. Several free and commercial options exist; Wilson's recommendation: Subversion (http://subversion.apache.org/), which is reliable and free.

application that is part of the Office for Windows package. Unlike structured software, OneNote allows users to handwrite, type or paste content anywhere on a page, enabling him to develop an ad hoc, virtual lab notebook.

Initial testing was promising, so Baumberg decided to reboot the lab's documentation strategy when he moved his lab to Cambridge. "You can only do that when you're setting something up from scratch, so we just decided, that's it, no more paper." Each lab member, some 100 researchers including collaborators and students, was issued a tablet PC (each costing about $1,200) running OneNote, whose pages are stored on a central server that all lab members could access over an internal network.

These pages can be searched, linked, reordered, archived, edited, tagged, annotated and bundled in virtual 'notebooks' representing different projects. During group meetings, previous weeks' notes can quickly be recalled, tasks assigned and to-do lists compiled. The tablet "is an amazing enhancing tool compared to a regular notebook," says Maximilian Bock, a third-year doctorate student who joined the lab just after the move to Cambridge. He plans to adopt the same strategy when he starts his own lab.

As lab director, Baumberg can view and comment on everything from his office, the local pub or an airport lounge on the other side of the world, whether on his tablet or with an Apple iPhone. So can all his researchers, a fact that enhances collaboration. The tablet also lets Baumberg make other connections, functioning as an ever-expanding, portable, personal encyclopedia. "I can actually search for when I last saw a talk about this particular area, and it will find in my handwritten notes a conference that I went to two years ago," he says.

Other impromptu hardware-software combinations are possible. Though Apple's iPad does not include a built-in stylus, many third-party options are available. Microsoft has released a version of OneNote for iOS, Apple's mobile operating system. Other iPad options that accept handwriting input include Notes Plus (from Viet Tran) and Note Taker HD (Software Garden Inc.).

Cookbook algorithms

Another example involves slightly more sophistication: automating the process of microscopy data analysis. Simon Watkins, a professor and vice chair of cell biology and director of the Center for Biologic Imaging at the University of Pittsburgh, says his microscopes can churn out terabytes of data per day. Just a few years ago such a torrent of data would have been unmanageable. Today, microscopists no longer need to be expert programmers to handle and process these data. "In nearly every situation, with the high-end image processing packages, all you have to do is tie the tools together," Watkins says.

Software such as Molecular Dynamics' Metamorph, Nikon's Elements and the

A lab move allowed Jeremy Baumberg to transition his entire group to a linked system of electronic laboratory notebooks that provides remote access to information and enhances collaboration.

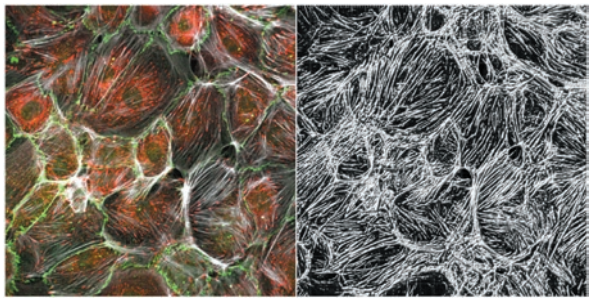
free ImageJ package let researchers piece together complex data-processing algorithms, both within and between applications, using simple scripting languages that are easy to learn. Sometimes the process is as simple as keystroke logging. Algorithms assembled in this way can contain 50 or more steps. But all have in common the same basic processes: data collection, image processing, segmentation (isolating features of interest based on classifiers such as brightness, shape or size) and analysis.

The key to tying algorithms together, Watkins explains, is figuring out how to mathematically define the features you want while filtering out the ones you do not. For Watkins, it is an exercise in relaxation, "like doing a jigsaw puzzle or a crossword," and it's one he uses to kill time on planes.

While returning from a lab course at the Mount Desert Island Biological Laboratory last year, Watkins worked out how to extract actin stress fiber volumes from confocal images of oxygen-deprived lung cells. He perfected the steps en route from Bangor, Maine, USA to Pittsburgh. First, deconvolve the confocal images in Autoquant (from Media Cybernetics) to correct for the axial point spread function. Then, isolate the cells' basal planes and segment the long planar actin filaments in Metamorph, and finally extract the volumetric data with Imaris (Bitplane). The results[2], published in March, indicated that actin volume increased under hypoxic conditions, an analysis Watkins could not otherwise have performed, he says.

Since 1997, Greg Wilson has been committed to teaching scientists how to use and build software through his Software Carpentry project.

Anyone can do this kind of work, Watkins says. It is like cooking: "You have to know what the ingredients of the recipe are to get good food out at the end. And the same goes with image processing: you have to know the ingredients; you have to know how they're going to implement your algorithm such that you get high-quality reproducible data at the back end."



Immunostaining for focal adhesion proteins paxillin (red), beta-catenin (green) and actin (white) (left). Volume-rendered image of actin (right). Simon Watkins developed the algorithm to calculate the actin volume while en route from Maine to Pittsburgh.

Easy interfaces

With the exception of some childhood tinkering, Cameron Neylon, an experimental biophysicist at the UK-based Science and Technology Facilities Council, was never much into programming. When he began working with collaborators who were developing electronic lab notebooks, he initially decided not to do any coding at all, reasoning that all tools should be usable by non-coders, like himself.

But when he realized the programmers lacked resources to implement certain features he believed were important, Neylon changed his mind and dove in. Using freely available open-source code, online resources and a bit of help from Software Carpentry's Wilson, Neylon taught himself the scripting language Python. It took about a month, Neylon says.

His first project was a 'robot' for Google's Wave collaboration tool. The robot could take a bit of text containing chemical information, parse the text, identify standard compound numbers, conduct a web search to determine the corresponding compound names, retrieve relevant information from the ChemSpider service and paste the information back into the document.

That project "didn't get much further than proof of concept," Neylon says. He got more mileage from his second project. Neylon's research group uses both an electronic lab notebook system, a blog-based system designed by Jeremy Frey's team at the University of Southampton, called LabTrove, and a separate computer system for controlling laboratory instruments and processing data. "What I needed to do was to connect the two together to keep a proper record of what had happened in the data processing," he says.

Neylon built a graphical user interface that lets users select specific raw data files from a small angle neutron scattering instrument, assemble the associated data files in one location, process them and upload the resulting data to the electronic lab notebook. He even built a job queue handler so tasks could be handled as a batch.

"What the software does is make it easy to get things into the notebook rather than having to do a manual upload," Neylon says. More importantly, it provides a reliable record of how a processed data file was generated. "Being able to check whether things were done right later down the track has a great potential to save headaches."

Neylon estimates he spent three to four weeks over six or seven months developing the software. Now, he has his eye on other lab inefficiencies. For instance, he would like to build a simple website that would help scientists calculate how to prepare buffers, and make sure that information gets stored in their electronic lab notebooks.

Indeed, Neylon says most of his programming has been "to glue together things that already exist." His advice for other budding programmers: start small. Learn to script macros in your favorite software. "That's a stepping stone to full-blown programming," he says. And it does not have to be difficult, he adds: "Most of the problems people are trying to solve are relatively simple mechanization problems. So look for simple solutions rather than complicated ones."

1. Miller, G. *Science* **314**, 1856–1857 (2006).
2. Bernal, P.J. *et al. Am. J. Physiol. Lung Cell Mol. Physiol.* published online 4 March 2011 (doi:10.1152/ajplung.00328.2010).
3. Haddock, S. & Dunn, C. *Practical Computing for Biologists* (Sinauer Associates, Inc., 2010).

Jeffrey M. Perkel is a freelance writer based in Pocatello, Idaho, USA (jeff@jeffreyperkel.com).