Get With the Program

DIY tips for adding coding to your analysis arsenal

By Jeffrey M. Perkel | August 1, 2015

© IVAN LUKYANCHUK/SHUTTERSTOCK

Biological science these days is all about Big Data. Whether it's in the form of DNA sequences, photomicrographs, or mass spectra, researchers increasingly need to collect, integrate, manipulate, and interpret enormous pools of information.

For many biologists, that can be pretty intimidating. Traditional training programs tend to focus on scientific fundamentals and experimentation, not computer programming and statistics. As a result, when many researchers find themselves confronted by massive data sets, they have no idea how to tackle them.

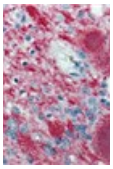
There's no shortage of readily available computational tools to help—many free of charge—but these too can be overwhelming for the uninitiated. Typically, users must interact with these programs through command-line wizardry, rather than through user-friendly graphical interfaces. And doing so often requires a deep knowledge of the underlying algorithms.

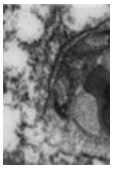
The upshot is that researchers working with big data inevitably have to write at least a little code to handle the information in a reproducible and well-documented way. Yet they must be cautious. It's easy to make a mistake, and if researchers aren't careful, they can end up compromising the data itself.

The Scientist spoke with several scientific programmers about the tools they use, as well as best practices for fledgling coders. Here's what they said.

PICK A LANGUAGE, ANY LANGUAGE

Biologists can choose from among a wide array of programming languages, and for many applications any language will do. But the most popular are probably Python and R. "That's the dynamic duo for science these days," says Vince Buffalo, a bioinformatician and graduate student at the University of California, Davis, and author of the new book *Bioinformatics Data Skills* (O'Reilly Media Inc.). Both are relatively user-friendly, but Python is capable of performing just about any task, whereas R is geared towards statistics. Both languages also have robust communities of scientists who use them, and thus large libraries of prebuilt code for discipline-specific functionality. For instance, the Bioconductor Project





(www.bioconductor.org), built in R, provides modules for handling microarray, sequencing, and microscopy data, among others. Popular scientific libraries for Python also exist, and can easily be installed via the science-focused Python installer, Anaconda (continuum.io/downloads).

Cole Trapnell, an assistant professor in the Department of Genome Sciences at the University of Washington, uses R to help process single-cell genomic data sets. "Single-cell genomics problems involve lots of statistical machine learning, and R is a good language for that."

Another popular option, especially for relatively slow or memory-intensive tasks, is C/C++. Or, check out Julia (www.julialang.org), Trapnell says, an emerging language that combines the syntax of Python, the graphing acumen of R, and the speed of C++. "That means the code is easy to write, but super fast." As a general rule, C. Titus Brown, an associate professor of genetics at UC Davis, advises that you pick the language your colleagues are using, so they can help if you're stuck.

ASSEMBLE YOUR TOOLS

UNIX and Linux machines typically have development software pre-installed, but if not, it's easy to obtain through the operating system package manager. Macs include a Python interpreter and C/C++ compiler out of the box, but R must be installed separately (www.r-project.org). Windows does not by default contain any of the programming languages, so you probably will need to install any you wish to use.

© IVAN LUKYANCHUK/SHUTTERSTOCK

The only other tool programmers really need is a good text editor—that is, a program capable of handling plain-text files, as opposed to proprietary formats such as Microsoft Word. Hard-core programmers typically favor the command-line editors vi or emacs, both of which come preinstalled on Linux and Mac. These are exceptionally powerful and configurable, but difficult for newcomers. "You want to think about the code, not about how to put your work down on paper. So I would recommend using whatever is least in your way," says Trapnell.

Look for an editor capable of colorizing language-specific keywords ("syntax highlighting"), syntax checking (for instance, ensuring that brackets and parentheses are properly paired), code formatting, and handling multiple files. Two popular options are Notepad++ (notepad-plus-plus.org) for Windows users, and Sublime Text (www.sublimetext.com) for Mac, Windows, and Linux. There's also a Mac-friendly version of emacs, called AquaMacs (aquamacs.org). "The key thing I emphasize to students is they should be using the mouse as little as possible," says Karl Broman, a professor of biostatistics and medical informatics at the University of Wisconsin–Madison. "Every time you move your hands away from the keys, you're slowing yourself down."

If you plan to program mostly in one particular language on a single platform (such as Mac or Windows), you might try an integrated development environment (IDE). IDEs integrate text editing, syntax highlighting, version control, help, build tools, and debugging in one interface, simplifying development. Mac C/C++ programmers can use the free Xcode (developer.apple.com/xcode), while Windows users can use Microsoft Visual Studio (www.visualstudio.com). For R programming, a popular choice is RStudio (www.rstudio.com). Eclipse IDE is a modular tool supporting multiple languages (eclipse.org/ide). Basic versions of all these platforms are free to download; additional features for advanced users also are available.

EMBRACE THE COMMAND LINE

Not every challenge requires programming per se. A lot can be done with simple scripts written in the command-line language called bash. "Bash is designed for working with files," explains Brown, "moving files around, iterating over files in a directory, mass moving and renaming of files." Using bash, researchers can assemble prebuilt tools into automated workflows, for instance to filter the high-quality reads from a set of sequencer runs, concatenate them into a new file, and pass that file to an analysis program using a predefined set of parameters. There's a good command line primer here: cli.learncodethehardway.org/book/.

BEFRIEND A DEBUGGER

Newbie programmers often eschew complicated debugging tools in favor of ad hoc approaches, such as peppering their code with print statements that describe the state of the program at a given point (e.g., "Now entering this block of code with value X").

Debuggers, though, allow programmers to halt software execution at a particular line of code to see what's happening under the hood and then advance the code line by line to see where things went wrong. That can be critical when trying to track down subtle bugs. "Every language has a debugger, and it's always worth learning how to use it," says Kevin Thornton, an associate professor of ecology and evolutionary biology at UC Irvine. Online tutorials are typically available, "so Google is your best friend there," he adds.

© IVAN LUKYANCHUK/SHUTTERSTOCK

USE VERSION-CONTROL SOFTWARE

Rachel Slaybaugh, an assistant professor in the Department of Nuclear Engineering at UC Berkeley, uses C++ and Python to develop computational methods for modeling neutron transport. She is still developing the best coding practices for her lab, she says, but one will surely be that everything in the lab has to be version-controlled.

Version-control software, such as the popular Git (git-scm.com), is like Word's track-changes feature, only amped up for software development. Using it, programmers can see precisely what changed from one program version to another, and critically, roll back those changes if necessary. Free and open source, Git also facilitates project collaboration, providing mechanisms to share code, "fork" development to test new algorithms without destroying existing code, track and merge differences, and so on. The Git hosting service, GitHub, provides a no-cost mechanism to share, distribute, and document code online, and also provides a stable link you can include in publications to point to your software. Upgraded plans are available starting at $7/month for private projects with their own access-restricted repositories.

Although Git is admittedly a complicated piece of software, it is of "paramount importance," Buffalo says. "More often than not, it'll save our ass at some point. If I introduce a bug, I can go to an old version and see if it is there. And then I can move forward version by version to see when it arrived and address it."

Some researchers prefer the alternative version-control system Mercurial (mercurial.selenic.com) and its hosting service BitBucket (bitbucket.org). It's free for up to five users, including unlimited private repositories.

AUTOMATE AND DOCUMENT

According to Brown, who coauthored a recent primer on scientific computing (*PLOS Biol*, 12(1):e1001745, 2014), one key to success is automation. "Automate everything," he says. In other words, rather than entering parameters manually each time you run your program, plug them into a command-line script that executes the program automatically. That will allow you to manage the manipulation and movement of data from file to file without errors. It also enables reproducibility.


© IVAN LUKYANCHUK/SHUTTERSTOCK

Suppose you're running a home-grown modeling algorithm, Brown explains. "If you're going to run that 10 times, write a shell script that runs it 10 times. Don't do it manually because you will forget what you ran, and you will forget how to reproduce it." Those scripts can be version-controlled as well, providing a mechanism for comparing results with those from a few weeks ago.

And don't forget to document everything, Slaybaugh says. Add comments to your code to tell others (and remind yourself) what it's doing. If you plan to share your code with colleagues, document it extensively using a tool like "doxygen" (www.stack.nl/~dimitri/doxygen), which creates software documentation automatically from properly commented source code in a variety of languages. Alternatively, researchers can use the Jupyter Project (jupyter.org, formerly the IPython Project) to interleave code, data, and documentation in a kind of online annotated notebook for Python, R, and other languages.

TEST YOUR CODE

Slaybaugh offers another piece of advice for rookie programmers: develop a test suite. A test suite is simply a set of functions and sample data that can be used to ensure the program is doing what you expect. That way, as you develop your software you can ensure you haven't altered the underlying

algorithm, introducing a hard-to-track bug. "You run the test system to make sure it still passes," she explains. The process is no different than using positive and negative controls at the bench, Brown says.

Buffalo's book provides a simple example. Suppose you have a function ("add") that sums two numbers. You might write a function called "test_add" to ensure that, despite your software changes, 2 + 3 still equals 5.

TREAT DATA AS "READ-ONLY"

Use an abundance of caution when working with your hard-won data, Buffalo says. For instance, "treat data as read-only." In other words, don't work with original copies of the data, make working copies instead. "If you have the data in an Excel spreadsheet and you make a change, that original data is gone forever," he says.

© IVAN LUKYANCHUK/SHUTTERSTOCK

GET HELP

Newbie programmers can find an abundance of resources online to get started and find help with whichever coding language they choose. The Software Carpentry project (software-carpentry.org) provides free workshops on scientific programming, and most of their teaching materials are available online. According to Brown, who is an instructor for the organization, these workshops can be arranged at any site "on a cost-neutral basis," meaning you need only foot the bill for instructor travel and lodging. There are also online tutorials through sites like Codecademy (www.codecademy.com) as well as online college courses (e.g., lifehacker.com/tag/lifehacker-u). The StackOverflow Web site (stackoverflow.com) is another great place to find answers, or you could simply ask your colleagues.

FIND A GOOD PROJECT

Finally, remember that programming is not a skill employed in a vacuum. It's intended to solve a particular problem. Thus, says Broman, when first sitting down to learn programming, be sure to have a good problem in mind to solve. Otherwise, progress will come slowly, and motivation may be lacking. If nothing else, says Buffalo, take a crack at adding missing features to open-source software you may be using. Broman agrees: "Having ideas for things you want to do and challenging yourself to do them in this new language, that's really how one learns."

Tags

techniques, software, data analysis and computer programming

64

Add a Comment

You

Sign In with your LabX Media Group Passport to leave a comment

Not a member? Register Now!

Comments

Ian Percy
Posts: 1

August 11, 2015

Excellent advice. When it comes to Big Data, success is always dependent on the quality of the software. Unfortunately, as Capers Jones succinctly put it, "software has one of the highest failure rates of any product in human history due primarily to poor quality." Jones, unarguably the world's leading researcher in software quality, noted that, on average, for every 100,000 lines of code there are 750 faults with 250 of them very capable of producing erroneous conclusions or shutting the system down completely. The *very* best of current

testing tools still leave 5 percent or 38 of those faults undetected. In science "good enough" isn't good enough.

Clearly fault-free software is mission-critical in the research arena. Do not listen to those in the IT establishment who have declared "fault-free" to be an unattainable Holy Grail. There *is* a new technology that combines proprietary and public semantic and logical processes in a mathematically rigourous way that has not been done before. This enables the identification and location of ***all*** faults in the software source code. What's exciting is that this capability can reduce cost and time of software-related projects by 40 to 60 percent.

It's inappropriate to be promotional in this space, so I won't mention the company. We're early-stage and close to having a fully functional prototype and are *very* eager to bring this technology to the bio-sciences. Anyone with interest in this developing technology is invited to comment here.

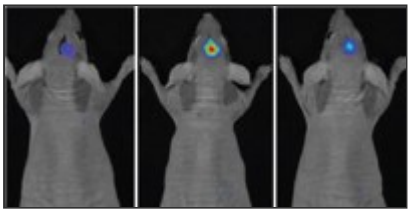
Sign in to Report

Related Articles

Mass Resignation from *Scientific Reports*'s Editorial Board

By Catherine Offord

Nineteen researchers have stepped down after the journal decided not to retract a paper that they say plagiarized the work of a Johns Hopkins biomedical scientist.



Caught in the Act

By Devika G. Bansal

Molecular probes for imaging in live animals

Scientists Who Developed Cryo-Electron Microscopy Win Nobel Prize

By Diana Kwon

Chemistry Nobel goes to Jacques Dubochet, Joachim Frank, and Richard Henderson.