Sage. (Version 3.4) `www.sagemath.org`. Free, with GNU Public License (GPL).

Sage is software for mathematics. To the uninitiated, this statement might sound unimpressive, or even obvious, but the readers of *SIAM Review* clearly recognize the challenges of representing the infinite and the continuous in a machine that is finite and discrete. For example, consider just the vagaries of floating-point arithmetic. A better description, that concisely captures the essence of Sage, comes from project's Mission: "Creating a viable free open-source alternative to Magma, Maple, Mathematica and Matlab." While Sage continues to improve and expand at a dramatic pace, it has come a long way toward meeting its goals. Stable and fast algorithms are provided for much of the mathematical universe, including symbolic, exact, numerical and graphical capabilities. A notebook interface runs in a web browser and provides a convenient and productive environment for using all of Sage's features. The user and developer communities have also expanded dramatically. All of this is based on open-source software, open standards and an open development process.

Borne of his frustration with proprietary programs providing similar functionality, William Stein founded Sage in 2005 and continues to lead the project. He wondered how you could rely on software for research in mathematics if you had little or no knowledge of the algorithms and code producing those results? He believed rapid progress in scientific research had always been predicated on an open exchange of ideas, and so it should be with software for mathematics. Since 2005 the project has attracted a very large user community, as measured by these recent monthly statistics provided by Harald Schilly, the Sage website and forum manager: 2,000 forum posts generated and viewed by the 2,000 forum members, 6,000 downloads of the program and 60,000 visits to the website. Contributors to the code are an international group numbering 150, while at any one time roughly 40 of these developers are working assiduously on projects or improvements. Major funding sources are University of Washington, University of California San Diego, National Science Foundation, Google, Microsoft, Sun and the US Department of Defense.

The genius of Sage is its leveraging of other open-source software projects. There are many mature and stable software projects devoted to relatively narrow areas of mathematics whose authors have released the source code under open licenses, such as the statistics package R and the scientific computing package SciPy. Since Sage is made available with a compatible license, it is allowed to directly integrate this functionality. This brings open, well-tested and very fast algorithms into the project at a cost of simply providing an interface between Sage and the added code. This also allows the vast majority of development work to focus on adding new features and algorithms to the current 300,000 lines of new code, rather than duplicating existing work. Often this work is done by specialists providing tools for their own research and teaching ("scratching their own itch" in open-source parlance). Sage provides the infrastructure, both in terms of a work environment and common basic mathematical functions, that allows specialists to concentrate on algorithms and new features.

Sage's approach has been to add functionality wherever possible through existing packages, and then provide new code where no open-source package exists or speed improvements are possible. An early slogan was, "Building the car, not reinventing the wheel." An early choice of the Sage project was to use Python as the principal language of the project. Many system administrators like to use Python as "glue code" to cobble together different programs and commands into a single script that automates maintenance tasks. Sage exemplifies this philosophy to an extreme. It has been a good choice because many smaller projects, both for mathematics and the associated infrastructure of a large program, have been written in Python, ported to Python, or provide interfaces to C/C++ code via Python. Two examples are the SciPy and NumPy packages, which are used heavily in Sage. NumPy provides critical high-dimension array manipulations, and SciPy builds on NumPy to provide a variety of mathematical tools such as numerical integration, differential equation solvers, and linear algebra routines. While SciPy is an open-source project, it is sponsored by a commercial company, Enthought, that in turn specializes in providing scientific applications to clients based on Python software.

Sage is created and maintained by a world-wide pool of developers, but actual changes are coordinated by a release manager. Much of Sage's stability is due to Michael Abshoff's tenure as sole release manager from January 2008 to May 2009, while more typically the duties rotate through developers such William Stein, Carl Witty, Mar-

tin Albrecht, Robert Miller, Mike Hansen, Craig Citro, Nick Alexander, Tom Boothby and Minh Van Nguyen. Part of Sage's stability comes from requiring every change to receive a positive review by another Sage developer before being added, not unlike the review of a journal article by a referee. Similar to the Linux kernel development process, all approved changes are then integrated into the official version by a release manager. There is an automated test system for catching unintended effects in other areas, providing more safeguards for the high quality of the code as tests are performed after making changes and interested volunteers test preliminary versions on a variety of hardware. It is important to realize that Sage contains *everything* you need to use all of its features. You can install it anywhere you like — your desktop, a laptop, a USB thumb drive or personal space on a shared server. It has no dependencies and does not require administrative privileges to install or run. The included packages are up-to-date and tested to work with each other. Sage is very serious about compiling, installing and running easily on a wide variety of hardware. Binary distributions are available for eight popular Linux distributions (32- and 64-bit each), Mac OSX (Intel and PowerPC), Windows, plus more exotic configurations such as Sun Solaris, the Intel Atom chip for netbooks, and the Itanium chip. At this writing, the best way to run on Windows is within a virtual machine, and therefore this version is distributed as a VMware image for the freely available player. A native Windows port is an ongoing project. With a relatively current compiler installed (as provided by most Linux distributions and Mac OSX) it is almost as easy to just compile all of Sage from source since it takes just a few simple commands to initiate the compilation. But be prepared to wait several hours, Sage is big. As an extreme example that illustrates the robustness of Sage's build system, Carl Witty, a Sage developer, downloaded the Sage source code to his Google Android G1 cell phone, and after a total of 15 days of compile time and 3 days of automated testing, he eventually had a working command line version.

Sage can be run from its own command line, which is useful for batch processes. However, for a new user or a student, the notebook interface is one of the strongest features. With Sage running in server mode, either on the same computer or remotely, a user may execute Sage commands via a web page. You can experiment with Sage, and this interface, by quickly creating an account on the public and free server located at `sagenb.org` and logging in and working through the tutorial [1]. The notebook is a collection of web pages, which are known as worksheets. Each worksheet is a sequence of input and output cells. An input cell is a sequence of Sage commands which can be evaluated as a group, with the object on the final line being displayed. Print commands can be used to output intermediate calculations. As Python is an interpreted language, the results are immediate, encouraging exploration and experimentation. With tab-completion it is easy to see exactly which commands are available for an object. The statement `A.determinant?` will bring up brief but helpful documentation for the determinant of the matrix A, while `A.determinant??` will bring up the source code for this command.

Because Sage is written in Python, it is most natural to construct programs calling Sage routines in Python, though it is not necessary to know any Python to use Sage effectively. A full working copy of Python is included in the distribution, and from the command line or the notebook, there is no overhead to immediately writing routines in Python. It is even possible that a user will learn some Python syntax without even realizing it. This is in contrast to the proprietary or one-off languages used by similar programs. Sage contains a Fortran compiler, which can be activated with a single line of code, and Fortran routines can be called from Python code with the included f2py utility. A spin-off of Sage is the Cython project, which builds on work of the Pyrex project. This project defines additional syntax for Python that in turn enables the generation of compiled C code and interfaces to C routines. Many new routines in Sage begin as interpreted Python and once stable are modified easily and quickly to Cython for the resultant speed improvement. Sage also includes interfaces to Axiom, Maple, Mathematica, Matlab, MuPad, and Octave, provided you have legitimate licenses for those programs in this list that are proprietary. So you can gain the benefits of the Sage notebook without orphaning any existing code.

Every mathematical object in Sage can be output in the typesetting language LaTeX and the notebook includes the open-source software jsMath for displaying LaTeX properly in a web page via Javascript commands. So it is possible to automatically (via one checkbox) have high-quality output. Or you can output the raw LaTeX output to paste into another document. Addi-

tionally, Sage makes available through Javascript the open-source TinyMCE mini-word-processor so that it is possible to annotate a worksheet. Entering small snippets of LaTeX in the word-processor will cause jsMath to render the mathematics properly upon exiting TinyMCE. Three-dimensional plots are rendered in an open-source Java applet, JMOL, originally built for chemists to visualize molecules, but adapted by Sage to render, rotate, and zoom surfaces, data, or curves in space. Worksheets are designed to be published at publicly accessible URLs or shared among a small group of collaborators. There is simple-to-use infrastructure to create sliders, input fields, and checkboxes in the output of a cell, such that the output responds to changes in the inputs. So for example, a slider might control the degree of a Taylor polynomial and the output would include a plot of both the original function and the approximation (see examples at [2]). Such a demonstration could be accompanied by notes written in the word-processor. The worksheet is a comfortable place to learn Sage (and Python and LaTeX) with obvious applications in education. For the researcher it is a comfortable place to test new applications before scaling up to production runs via the command line.

Sage is a big and fast-moving target. Not surprisingly in a young open-source project, documentation tends to lag. However, right in the source code for a command there is almost always a fairly complete explanation of input options and an explanation of the output, along with a collection of examples. Extending this coverage to 100% is one of the current priorities. One can construct an object (e.g. a function, matrix, ring,...) and use tab-completion to see just the functions possible for this object, and this is often a quick and easy way to get started experimenting with a new area within Sage. Even better are the very active online forums, including: sage-support for help with routine questions, sage-edu for discussions of educational applications, and sage-devel for technical and design discussions, in addition to specialized forums for number theory and combinatorics and active Internet Relay Chat (IRC) channels for both support and development. Developers, including William Stein and the release managers, frequent the support areas and are quick to offer assistance or recognize bugs that need attention. It is a civil, enthusiastic and helpful community, with flame wars distinguished mostly by their extreme rarity. Well-formed questions are often handled quickly and accurately. Suggestions for new packages or functionality are welcome.

Sage is big and ambitious. What parts are of most interest to readers of *SIAM Review*? First, Sage does not have an easily discernible bias between symbolic, exact and numerical arenas. Symbolic manipulation is provided by a variety of different packages and some of the more notoriously difficult areas are a current focus of Sage development. With research in number theory as an early motivation, support for exact mathematics is impressive. Sage strives to represent mathematics as a mathematician views it. For example, in *Mathematica* the command `NullSpace[A]` will return a list of basis vectors for the null space of the matrix A, rather than a vector space. In Sage, the command `W = A.kernel()` will return a *vector space* `W`, which is a variable that can be further inspected with commands appropriate to a vector space, such as `W.basis()` returning a list of basis vectors, if that is what is desired. It is a subtle, but very important distinction. Similarly, in Sage you are allowed, or required (depending on your perspective), to specify the base ring or field you are working over. For example, the syntax `R.<x> = ZZ[]` defines the ring R, of polynomials in x over the integers, while `R.<x> = QQ[]` defines R over the rationals, and `R.<x> = CC[]` defines R over the complex numbers. Now define a polynomial in x by `p = 2*x^3+x^2+2*x+1`. Depending on the definition of R, the command `p.factor()` will return $(2x+1)(x^2+1)$ over the integers, $2(x+\frac{1}{2})(x^2+1)$ over the rationals and $2(x+\frac{1}{2})(x-i)(x+i)$ over the complexes. This can be a source of confusion for the novice. However, it accurately mirrors actual practice in mathematics and in the long-run the necessary precision and clarity make it possible for Sage to more directly and easily provide correct answers in more complicated situations.

Arbitrary precision computations are equally at home in Sage. For example a matrix could be simply specified over a "field" of real numbers with 200 bits of precision as `C = matrix(RealField(200), [[1,2],[3,4]])`. Consistent with Sage's approach, this field employs the routines of the open-source MPFR package for multi-precision floating-point numbers with correct rounding. For a random 1000×1000 matrix of double-precision reals (the field RDF in Sage syntax) a determinant is computed via the SciPy, NumPy and BLAS packages in about 0.1 seconds on $500 hardware. When the ring is changed to the integers, the

open-source IML library for integer matrix computations is employed instead. Then the roughly 3,0000 digits of the determinant of a random 1000×1000 matrix are computed *exactly*, though the computation time grows to about 12 seconds.

A few examples of useful open-source packages for applied mathematicians that are integrated into Sage are: SciPy and NumPy for numerical and scientific computing with Python that captures much of the functionality of Matlab; the complete R program for statistical analysis; CVXOPT for linear programming and similar optimization problems; the GNU Scientific Library (GSL) for fast Fourier transforms and numerical differential equations; ATLAS, BLAS, LAPACK, Linbox, M4RI and NumPy all for linear algebra.

One particular example might highlight the application of Sage in an area of applied mathematics. Ahmed Fasih is Ph.D. student at Ohio State University in the Department of Electrical and Computer Engineering studying radar signal processing algorithms and automatic signal analysis. His current work involves tracking moving vehicles with synthetic aperture radar data, specifically computing bounds for the minimum covariance of estimators of position, velocity, and complex amplitude of an electromagnetic scatterer in radar. In the course of this work, integrating functions with maximums of e^{-1000} or smaller became impossible to compute in Matlab. Then he found Sage and the included open-source Mpmath package, providing the necessary support for both arbitrary precision arithmetic and quadrature integration. Mpmath is another example of a specialized project that is an important component of Sage, and a recent NSF-funded joint project has strengthened this relationship. Converting his Matlab routines for Sage, Ahmed employed native Sage support for parallelizing the integrations on subintervals to submit 128 simultaneous jobs on 4-core nodes without any license infringements, and cutting his runtime by a factor of 500.

Ahmed obtained more accurate results, and better insights into his research application. He also reports the advantages of Sage containing all the bits he needed, packaged together to work together, and the freedom from the limits and hassles of licensing limits and needing administrative privileges on target machines. There is even a popular open-source distributed version-control system, Mercurial, that is packaged with Sage and was useful for his research group. He reports the roughest patch was initially understanding all the data type conversions for numbers and vectors between Sage, Python and Mpmath, but in the spirit of open-source development, he plans to use his experience to help the author of mpmath to simplify the situation for others. Presently, for a course in parallel computing architectures, he is experimenting with the PyCUDA package from within Sage, which is a Python interface to the CUDA C framework. This is an initiative to employ nVidia graphics cards as computational engines for intensive parallel computations (besides just computer games).

I resolved at the start of the 2008-09 academic year to learn Sage by using it every reasonable chance I got, abandoning a 20-year investment in *Mathematica*. The straightforward interface to the open-source package GAP (Groups, Algorithms and Programming) was uncomplicated enough to allow me to integrate a computational approach to group theory into my introductory course for the first time. The exact linear algebra routines are useful as I extend my linear algebra textbook, especially being able to simply cut and paste the LaTeX output from Sage to the book. In multivariable calculus the 3-D plots have been invaluable, such as a plot of the degree 16 two-variable Taylor polynomial approximating $f(x, y) = \sin(x)\cos(y)$ on $[-\pi, \pi] \times [-\pi, \pi]$, with Sage computing the 154 necessary partial derivatives symbolically. I've been trading similar worksheets for this course with Jason Grout at Iowa State University and Robert Mařík at Mendel University in the Czech Republic, often by publishing worksheets off the Sage public server. Sage will see significant action as I finish an integral calculus course this term with infinite series and Taylor polynomials. Preparing an upcoming presentation will give me an excuse to learn more about Sage's graph theory routines.

Sage is big, and there is much to explore and use in your professional activities as a mathematician. It is an impressive concentration and unification of mathematical knowledge. The reliance on mature open-source packages and open standards provides a measure of confidence and future-proofing. There are a few rough edges as the project matures, but this also provides the opportunity to get involved and influence development. But see for yourself by experimenting at the public server (`sagenb.org`) along with the over 5,000 others who have accounts there, or simply install your own copy on your favorite hardware. Either way, its free.

Acknowledgments This review has benefited greatly from the help of the Sage community, specifically Michael Abshoff, Robert Bradshaw, Craig Citro, Ahmed Fasih, Jason Grout, Mike Hansen, David Joyner, Josh Kantor, Nancy Neudauer, Harald Schilly, and William Stein. Their assistance is greatly appreciated.

References

[1] Sage Tutorial, `http://sagemath.org/doc/tutorial/index.html`.

[2] Sage Wiki Interactions, `http://wiki.sagemath.org/interact`

Robert A. Beezer
University of Puget Sound

An edited version of this review will appear in *SIAM Review* as part of the Book Review section.