Parabola Volume 44, Issue 1 (2008)

Sage — Free Software for Mathematics

Bill McLean[1]

A lot of useful mathematical software is freely available via the internet. If you look hard enough you can find a code to solve just about any standard mathematical problem, and for many years professional scientists have downloaded programs and subroutine libraries from repositories like netlib [4]. The trouble is that most of this software is not easy to use, and typically requires specialised programming skills.

For this reason, university mathematics courses mostly use commercial software applications like Maple, Matlab or Mathematica, that provide a simpler user interface. Several free applications [1, 3, 6, 8] provide similar, if less sophisticated, facilities. Some of these mathematics packages have a long history, for example, the earliest version of Matlab dates from the 1970s. In this article I will describe a relatively recent package called Sage [9] (Software for Algebra and Geometry Experimentation). The lead developer of Sage, a mathematics professor from the University of Washington called William Stein, released version 0.1 in February of 2005. At the time of writing, the latest version of Sage is 2.10.1.

The Sage interpreter

The quickest way to get a feeling for how Sage works is by looking at a few examples.

After starting Sage, a command prompt appears,

```
sage:
```

You can then type a command and press enter to have the Sage interpreter carry out that command. For example, rational arithmetic is easy.

```
sage: 3531/83411 − 45134/3132542
3648166864/130644230381
```

Here, Sage has evaluated the typed expression and printed the result on the next line (cancelling any common factors in the numerator and denominator). Sage provides many standard functions, such as factorials and binomial coefficients.

```
sage: factorial(41)
33452526613163807108170062053440751665152000000000
sage: binomial(143, 37)
```

[1]Bill McLean is a Senior Lecturer in the School of Mathematics and Statistics at UNSW

24430124567519572309215443791318020

You also have access to fast integer factorisation and operations useful in number theory, such as finding a greatest common divisor.

sage: $factor(2^{113} + 1)$
$3 * 227 * 48817 * 636190001 * 491003369344660409$
sage: $gcd(2805, 573937)$
17

Sage can perform standard operations from calculus. You can easily define a function, evaluate it at a point, and differentiate or integrate it.

sage: $f(x) = x^3 * sin(x) + log(x)$
sage: $f(e)$
$e^3 * sin(e) + 1$
sage: $f(pi)$
$log(pi)$
sage: $f.derivative()$
$x| - - > 3 * x^2 * sin(x) + x^3 * cos(x) + 1/x$
sage: $f.integral()$
$x| - - > (3 * x^2 - 6) * sin(x) + x * log(x) + (6 * x - x^3) * cos(x) - x$
sage: $f.integral(x, 0, pi)$
$pi * log(pi) + pi^3 - 7 * pi$

Here, the initial statement creates a function object `f`. The expressions `f(e)` and `f(pi)` evaluate the function at the predefined symbolic constants `e` and `pi` which have the obvious meanings from calculus. Notice that `f.integral()`, with no arguments, gives an indefinite integral $\int f(x)\,dx$, whereas `f.integral(x,0,pi)` gives the definite integral $\int_0^\pi f(x)\,dx$. The function object `f` has a long list of *bound methods*; to see a complete list, type `f.` followed by the tab key. To get information about an object, just type its name followed by a question mark, for instance,

sage: $f.diff?$

shows that `f.diff` is an abbreviation for `f.derivative`. (Type "q" to quit the help screen.)
Sometimes a complicated mathematical expression is difficult to read when printed on one line as a plain text string. For an example, consider

sage: $g = derivative(log(f), x)$
sage: g
$(3 * x^2 * sin(x) + x^3 * cos(x) + 1/x)/(x^3 * sin(x) + log(x))$

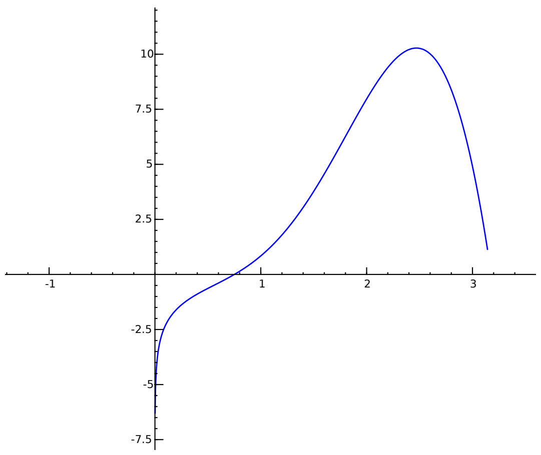


Figure 1: A simple sage plot.

The `print` command displays the result in a more readable form.

sage: $printg$

Even better is the `show` method: doing

sage: g.show()

opens a graphics window that displays a typeset version of g, looking like

$$\frac{3 \cdot x^2 \cdot \sin(x) + x^3 \cdot \cos(x) + \frac{1}{x}}{x^3 \cdot \sin(x) + \log(x)}.$$

Plotting functions or data is easy in Sage. For example, using the function `f` above we can create a plot object, display it and save a copy to an encapsulated postscript file, as follows.

sage: $p = f.plot(0, pi)$
sage: $p.show()$
sage: $p.save('fplot.eps')$

Figure 1 shows the result.

The discussion above covers only a tiny fraction of the capabilities of Sage, many of which are used for computations in areas of mathematics you would not normally study until university. Here, I have merely described some topics covered in high school mathematics.

Python Powered

How have the developers — a few professors and their students — managed to create Sage in only a couple of years? They have done so by assembling their system out of a large collection of existing free components using a programming language called Python [7] that provides a large part of the software infrastructure needed to run Sage. Instead of creating from scratch a new command interpreter, with all the work that would require, the Sage developers wrote Python classes to provide the objects they wanted. Mostly, these classes do little serious computation, but simply provide convenient and flexible interfaces to fast mathematical routines in existing software libraries. Similarly, a Python library called matplotlib provides the graphics capabilities for Sage.

An essential feature of the software projects used in Sage is a licence that grants access to the source code and freely permits redistribution. The licence also needs to allow modifications to the source code, since the Sage developers sometimes want to make such changes. Much of the software in Sage is licenced under some version of the Gnu Public Licence [2] or GPL.

The Python language is well-suited to developing a system like Sage, and also provides good support for mathematical computing. You can easily learn enough Python to write scripts that carry out a sequence of steps needed to solve a complex problem, or write your own procedures to implement a command that is not already available. However, a couple of features of Python are awkward for symbolic computing. For instance, in Python (as in most programming languages) the statement `n=5/8` gives the variable `n` the integral value zero, because Python has no built-in rational data type and so performs an integer division: 5 divided by 8 equals 0 (with remainder 5). When you type `n=5/8` in Sage, a pre-parser changes this statement behind the scenes to `n=Integer(5)/Integer(8)` before sending it to the Python interpreter. Converting the plain Python integers 5 and 8 into Sage integers has the effect of redefining the operation `/` so that the result is a Sage rational number. The Sage pre-parser also converts an expression like `5^2` into `5**2` because this is a way to say 5^2 in Python.

You can see what the preparser does using a simple command, for instance,

sage: $preparse('2/5 + 8^3')$
$'Integer(2)/Integer(5) + Integer(8) * *Integer(3)'$

The other main way in which Sage differs from Python is the syntax we used earlier to define a function:

sage: $f(x) = x^3 * sin(x) + log(x)$

Compare this with the way a true Python function is defined:

sage: $defg(x):$
.... : $returnx^3 * sin(x) + log(x)$

.... :

Although `f` and `g` give identical numerical values, they are different types of objects.

```
sage: f(3.4)
−8.82001204244046
sage: g(3.4)
−8.82001204244046
sage: type(f)
< class'sage.calculus.calculus.CallableSymbolicExpression' >
sage: type(g)
< type'function' >
```

Asking for, say, `g.derivative()` will result in an error message because a standard Python function has no method called “`derivative`”.

Numerical output

In our discussion so far, I have emphasised features of Sage that are designed for symbolic computation, producing exact results in the form of a string of text, but if desired Sage can convert a symbolic expression to an approximate decimal value. If the function `f` is defined as in our earlier example, then we can find its integral over the interval $(0, \pi)$ to 25 significant digits as follows:

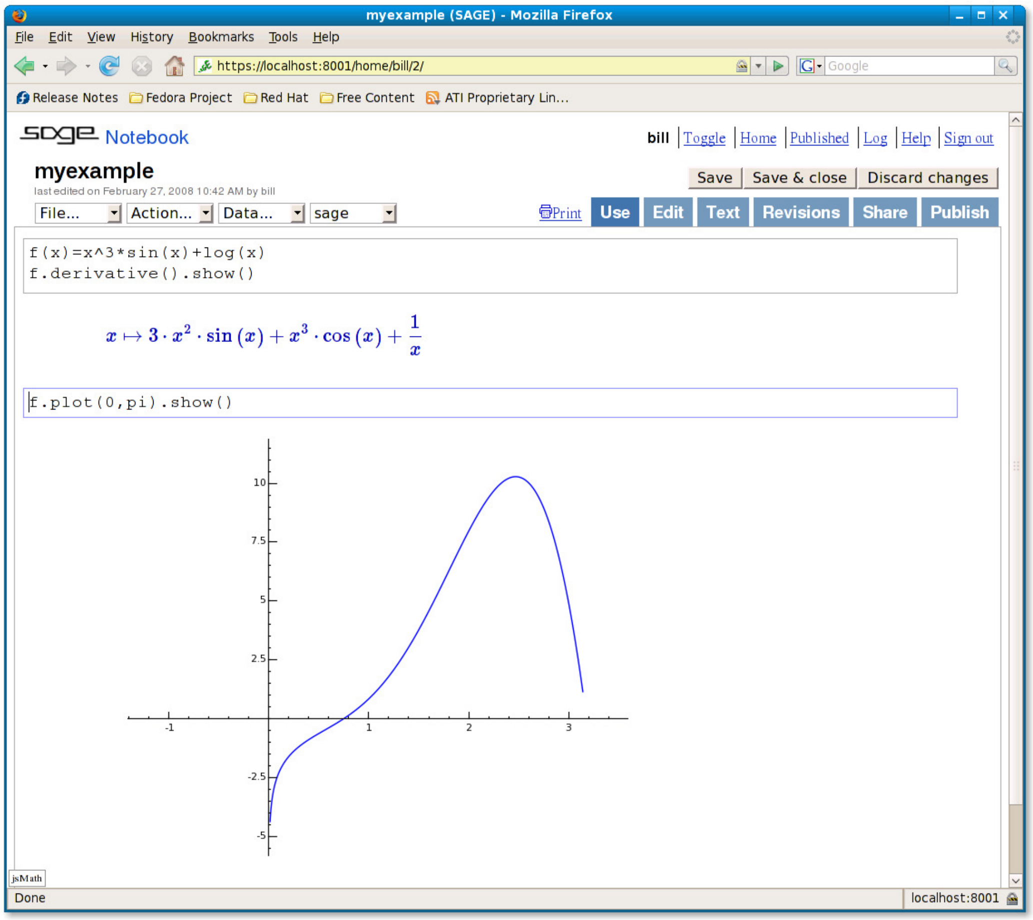
```
sage: ans = f.integral(x, 0, pi)
sage : printans.numericalapprox(digits = 25)
12.61140310490042570432381
```

This type of arbitrary-precision arithmetic can be very useful but it is too slow for large-scale computations of the kind that arise in many areas of applied mathematics. For serious floating-point computations we need to work with standard four- or eight-byte number formats that give a precision of about 7 or 15 decimal digits, respectively. These standard formats are directly supported by the hardware in all modern computers, so that arithmetic operations are much faster than for the corresponding operations on arbitrary-precision numbers, that must be performed by software. We also need facilities for efficiently manipulating very large arrays of data.

I will not try to explain the facilities in Sage for fast floating-point arithmetic, except to say that they are largely provided by another Python library called numpy [5].

The Sage notebook

Instead of interacting with the Sage interpreter via the command shell, you can create a *worksheet* by starting a web application called the Sage *notebook*. The command


myexample (SAGE) - Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://localhost:8001/home/bill/2/
Release Notes
Fedora Project
Red Hat
Free Content
ATI Proprietary Lin...
Notebook
bill | Toggle | Home | Published | Log | Help | Sign out
myexample
last edited on February 27, 2008 10:42 AM by bill
Save
Save & close
Discard changes
File...
Action...
Data...
sage
Print
Use
Edit
Text
Revisions
Share
Publish
jsMath
Done
localhost:8001

```
f(x)=x^3*sin(x)+log(x)
f.derivative().show()
```

$$x \mapsto 3 \cdot x^2 \cdot \sin(x) + x^3 \cdot \cos(x) + \frac{1}{x}$$

```
f.plot(0,pi).show()
```

Figure 2: A Sage worksheet.

sage: *notebook*()

starts a web server on your computer and opens a browser at the Sage notebook page. The first time you run the notebook, you have to set up an admin password and create a user account. In a school setting, Sage can run on a server and a student can use the notebook interface through a web browser without the need to install any software on his or her own computer. After logging in to the Sage notebook, you can create a worksheet consisting of a sequence of *cells*. You can type Sage commands in a cell, press Shift-Enter and see the output. Figure 2 shows a couple of simple cells; notice that graphical output is imbedded directly in the worksheet instead of appearing in a separate window.

A worksheet cell can also contain text with typeset mathematics, so you can compose a documentary record of your work, that is easily saved to a file or shared over the internet.

You can try out a Sage notebook by going to the Sage home page [9]; follow the link 'Online Sage Notebook'. From there, you can sign up for an account and then log in, or just view some published worksheets without logging in.

Installing Sage

Sage runs natively on Linux and on Mac OSX 10.4 (Tiger) and 10.5 (Leopard). The Download page on the Sage website has links to packages for these operating systems. You can also run Sage on Windows using VMware Player [10]. Detailed installation instructions for each of these options is available via the Download page. The Sage packages are quite large: the source tarball is over 200MB, as are the Mac and Linux binaries. The VMware virtual machine is around 500MB.

References

[1] http://freemat.sourceforge.net

[2] http://www.gnu.org/copyleft/gpl.html

[3] http://maxima.sourceforge.net

[4] www.netlib.org

[5] http://numpy.scipy.org

[6] www.octave.org

[7] www.python.org

[8] www.scipy.org

[9] Stein, William, *Sage: Open Source Mathematical Software (Version 2.8.15)*, The Sage Group, 2008, http://www.sagemath.org.

[10] http://www.vmware.com/products/player