1

Sage: Creating a Viable Free Open Source Alternative to Magma, Maple, Mathematica, and MATLAB

William Stein[1]

[1] The work was supported by NSF grant DUE-1022574.

1.1 Introduction

The goal of the Sage project (`http://www.sagemath.org`) is to create a viable free open source alternative to Magma, Maple$^{\text{TM}}$, Mathematica®, and MATLAB®, which are the most popular non-free closed source mathematical software systems.[1] Magma is (by far) the most advanced non-free system for structured abstract algebraic computation, Mathematica and Maple are popular and highly developed systems that shine at symbolic manipulation, and MATLAB is the most popular system for applied numerical mathematics. Together there are over 3,000 employees working at the companies that produce the four Ma's listed above, which take in over a hundred million dollars of revenue annually.

By a viable free alternative to the Ma's, we mean a system that will have the important mathematical features of each Ma, with comparable speed. It will have 2d and 3d graphics, an interactive graphical user interface, and documentation, including books, papers, school and college curriculum materials, etc. A single alternative to all of the Ma's is not necessarily a drop-in replacement for any of the Ma's; in particular, it need not run programs written in the custom languages of those systems. Thus an alternative may be philosophically different than the open source system Octave, which understands the MATLAB source language and attempts to implement the entire MATLAB library. Development could instead focus on implementing functions that users demand, rather than systematically trying to implement every single function of the Ma's. The culture, architecture, and general look and feel of such a system would be very different than that of the Ma's.

In Section 1.2 we explain some of the motivation for starting the Sage project, in Section 1.3 we describe the basic architecture of Sage, and in Section 1.4 we sketch aspects of the history of the project.

1.2 Motivation for Starting Sage

Each of the Ma's cost substantial money, and is hence expensive for me, my collaborators, and students. The Ma's are not *owned by the community* like Sage is, or Wikipedia is, for that matter.

The Ma's are closed, which means that the implementation of some

[1] Maple is a trademark of Waterloo Maple Inc. Mathematica is a registered trademark of Wolfram Research Incorporated. MATLAB is a registered trademark of MathWorks. I will refer to the four systems together as "the Ma's" in the rest of this article.

algorithms are secret, in which case you are not allowed to modify or extend them.

"You should realize at the outset that while knowing about the internals of Mathematica may be of intellectual interest, it is usually much less important in practice than you might at first suppose. Indeed, in almost all practical uses of Mathematica, issues about how Mathematica works inside turn out to be largely irrelevant. Particularly in more advanced applications of Mathematica, it may sometimes seem worthwhile to try to analyze internal algorithms in order to predict which way of doing a given computation will be the most efficient. [...] But most often the analyses will not be worthwhile. For the internals of Mathematica are quite complicated.."
– The Mathematica Documentation

The philosophy espoused in Sage, and indeed by the vast open source software community, is exactly the opposite. We want you to know about the internals, and when they are quite complicated, we want you to help make them more understandable. Indeed, Sage's growth depends on *you* analyzing how Sage works, improving it, and contributing your improvements back.

```
sage: crt(2, 1, 3, 5)   # Chinese Remainder Theorem
11
sage: crt?         # ? = documentation and examples
Returns a solution to a Chinese Remainder Theorem...
...
sage: crt??        # ?? = source code
def crt(...):
...
    g, alpha, beta = XGCD(m, n)
    q, r = (b - a).quo_rem(g)
    if r != 0:
        raise ValueError("No solution ...")
    return (a + q*alpha*m) % lcm(m, n)
```

Moreover, by browsing `http://hg.sagemath.org/sage-main/`, you can see exactly who wrote or modified any particular line of code in the Sage library, when they did it, and why. Everything included in Sage is free and open source, and it will foreover remain that way.

"I see open source as Science. If you don't spread your ideas in the open, if you don't allow other people to look at how your ideas work and verify that they work, you are not doing Science, you are doing Witchcraft. Traditional software development models, where you keep things inside a company and hide what you are doing, are basically Witchcraft. Open source is all about the fact that it is open; people can actually look at what you are doing, and they can improve it, and they can build on top of it. [...] One of my favorite quotes from history is Newton: 'If I had seen further, it has been by standing on the shoulders of giants.'"

– Linus Torvalds.
Listen at `http://www.youtube.com/watch?v=bt_Y4pSdsHw`

The design decisions of the Ma's are not made openly by the community. In contrast, important decisions about Sage development are made via open public discussions and voting that is archived on public mailing lists with thousands of subscribers.

Every one of the Ma's uses a special mathematics-oriented interpreted programming language, which locks you into their product, makes writing some code outside mathematics unnecessarily difficult, and impacts the number of software engineers that are experts at programming in that language. In contrast, the user language of Sage is primarily the mainstream free open source language Python `http://python.org`, which is one of the world's most popular interpreted programming languages. The Sage project neither invented nor maintains the underlying Python language, but gains immediate access to the IPython shell, Python scientific libraries (such as NumPy, SciPy, CVXopt and MatPlotLib), and a large Python community with major support from big companies such as Google. In comparison to Python, the Ma's are small players in terms of language development. Thus for Sage most of the problems of language development are handled by someone else.

The bug tracking done for three of four of the Ma's is currently secret[2], which means that there is no published accounting of all known bugs, the status of work on them, and how bugs are resolved. But the Ma's do have many bugs; see the release notes of each new version, which lists bugs that were fixed[3]. Sage also has bugs, which are all publicly tracked at `http://trac.sagemath.org`, and there are numerous "Bug Days" workshops devoted entirely to fixing bugs in Sage. Moreover, all discussion about resolving a given bug, including peer review of solutions, is publicly archived. We note that sadly even some prize winning[4] free open source systems, such as GAP `http://www.gap-system.org/`, do not have an open bug tracking system, resulting in people reporting the same bugs over and over again.

Each of the Ma's is a combination of secret unchangeable compiled code and less secret interpreted code. Users with experience programming in compiled languages such as Fortran or C++ may find the loss of a compiler to be frustrating. None of the Ma's has an optimizing compiler that converts programs written in their custom interpreted language to a

[2] MATLAB has an open bug tracker, though it requires free registration to view.
[3] See also `http://cybertester.com/` and `http://maple.bug-list.org/`.
[4] Jenks Prize, 2008

fast executable binary format that is not interpreted at runtime.[5] In contrast, Sage is tightly integrated with Cython[6] `http://www.cython.org`, which is a ython-to-C/C++ compiler that speeds up code execution and has support for statically declaring data types (for potentially enormous speedups) and natively calling existing C/C++/Fortran code. For example, enter the following in a cell of the Sage notebook (e.g., `http://sagenb.org`):

```
def python_sum2(n):
    s = int(0)
    for i in xrange(1, n+1):
        s += i*i
    return s
```

Then enter the following in another cell:

```
%cython
def cython_sum2(long n):
    cdef long i, s = 0
    for i in range(1, n+1):
        s += i*i
    return s
```

The second implementation, despite looking nearly identical, is nearly a hundred times faster than the first one (your timings may vary).

```
sage: timeit('python_sum2(2*10^6)')
5 loops, best of 3: 154 ms per loop
sage: timeit('cython_sum2(2*10^6)')
125 loops, best of 3: 1.76 ms per loop
sage: 154/1.76
87.5
```

Of course, it is better to choose a different algorithm. In case you don't remember a closed form expression for the sum of the first n squares, Sage can deduce it:

```
sage: var('k, n')
sage: factor(sum(k^2, k, 1, n))
```

[5] MATLAB has a compiler, but "the source code is still interpreted at run-time, and performance of code should be the same whether run in standalone mode or in MATLAB." Mathematica also has a `Compile` function, but simply compiles expressions to a different internal format that is interpreted, much like Sage's `fast_callable` function.

[6] The Cython project has received extensive contributions from Sage developers, and is very popular in the world of Python-based scientific computing.

```
1/6*(n + 1)*(2*n + 1)*n
```

And now our simpler fast implementation is:

```
def sum2(n):
    return n*(2*n+1)*(n+1)/6
```

Just as above, we can also use the Cython compiler:

```
%cython
def c_sum2(long n):
    return n*(2*n+1)*(n+1)/6
```

Comparing times, we see that Cython is 10 times faster:

```
sage: n = 2*10^6
sage: timeit('sum2(n)')
625 loops, best of 3: 1.41 microseconds per loop
sage: timeit('c_sum2(n)')
625 loops, best of 3: 0.145 microseconds per loop
sage: 1.41/.145
9.72413793103448
```

In this case, the enhanced speed comes at a cost, in that the answer is *wrong* when the input is large enough to cause an overflow:

```
sage: c_sum2(2*10^6)   # WARNING: overflow
-407788678951258603
```

Cython is very powerful, but to fully benefit from it, one must understand machine level arithmetic data types, such as long, int, float, etc. With Sage you have that option.

1.3 What is Sage?

The goal of Sage is to compete with the Ma's, and the intellectual property at our disposal is the complete range of GPL-compatibly licensed open source software.

Sage is a self-contained free open source *distribution* of about 100 open source software packages and libraries[7] that aims to address all

[7] See the list of packages in Sage at `http://sagemath.org/packages/standard/`. The list includes R, Pari, Singular, GAP, Maxima, GSL, Numpy, Scipy, ATLAS, Matplotlib, and many other popular programs.

computational areas of pure and applied mathematics. The download of Sage contains all dependencies required for the normal functioning of Sage, including Python itself. Sage includes a substantial amount of code that provides a unified Python-based *interface* to these other packages. Sage also includes a library of new code written in Python, Cython and C/C++, which implements a huge range of algorithms.

1.4 History

I made the first release of Sage in February 2005, and at the time called it "**S**oftware for **A**rithmetic **G**eometry **E**xperimentation." I was a serious user of, and contributor to, Magma at the time, and was motivated to start Sage for many of the reasons discussed above. In particular, I was personally frustrated with the top-down closed development model of Magma, the fact that *several million lines* of the source code of Magma are closed source, and the fees that my colleagues had to pay in order to use the substantial amount of code that I contributed to Magma. Despite my early naive hope that Magma would be open sourced, it never was. So I started Sage motivated by the dream that someday the single most important item of software I use on a daily basis would be free and open. David Joyner, David Kohel, Joe Wetherell, and Martin Albrecht were also involved in the development of Sage during the first year.

In February 2006, the National Science Foundation funded a 2-day workshop called "Sage Days 2006" at UC San Diego, which had about 40 participants and speakers from several open and closed source mathematical software projects. After doing a year of fulltime mostly solitary work on Sage, I was surprised by the positive reception of Sage by members of the mathematical research community. What Sage promised was something many mathematicians wanted. Whether or not Sage would someday deliver on that promise was (and for many still is) an open question.

I had decided when I started Sage that I would make it powerful enough for my research, with or without the help of anybody else, and was pleasantly surprised at this workshop to find that many other people were interested in helping, and understood the shortcomings of existing open source software, such as GAP and PARI, and the longterm need to move beyond Magma. Six months later, I ran another Sage Days workshop, which resulted in numerous talented young graduate students, including David Harvey, David Roe, Robert Bradshaw, and Robert Miller, getting involved in Sage development. I used startup money from University of Washington to hire Alex Clemesha as a fulltime employee to implement 2d graphics and help create a notebook interface to Sage. I also learned that there was much broader interest in such a system, and stopped referring to Sage as being exclusively for "arithmetic geometry"; instead, Sage became "**S**oftware for **A**lgebra and **G**eometry **E**xperimentation." Today the acronym is deprecated.

The year 2007 was a major turning point for Sage. Far more people got involved with development, we had four Sage Days workshops, and prompted by Craig Citro, we instituted a requirement that all new code must have tests for 100% of the functions touched by that code, and every modification to Sage must be peer reviewed. Our peer review process is much more open than in mathematical research journals; everything that happens is publicly archived at `http://trac.sagemath.org`. During 2007, I also secured some funding for Sage development from Microsoft Research, Google, and NSF. Also, a German graduate student studying cryptography, Martin Albrecht presented Sage at the Trophées du Libre competition in France, and Sage won first place in "Scientific Software", which led to a huge amount of good publicity, including articles in many languages around the world and appearances[8] on the front page of `http://slashdot.org`.

In 2008, I organized 7 Sage Days workshops at places such as IPAM (at UCLA) and the Clay Mathematics Institute, and for the first time, several people besides me made releases of Sage. In 2009, we had 8 more Sage Days workshops, and the underlying foundations of Sage improved, including development of a powerful coercion architecture. This *coercion model* systematically determines what happens when performing operations such as `a + b`, when `a` and `b` are elements of potentially different rings (or groups, or modules, etc.).

```
sage: R.<x> = PolynomialRing(ZZ)
sage: f = x + 1/2; f
x + 1/2
sage: parent(f)
Univariate Polynomial Ring in x over Rational Field
```

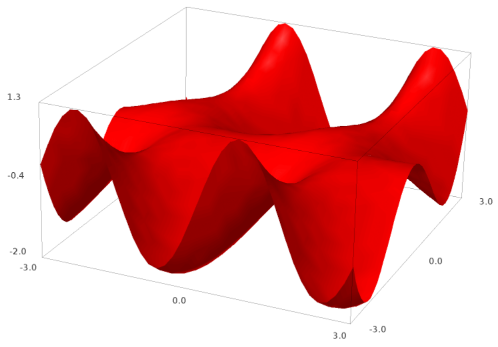
We compare this with Magma (V2.17-4), which has a more ad hoc coercion system:

[8] For example, `http://science.slashdot.org/story/07/12/08/1350258/Open-Source-Sage-Takes-Aim-at-High-End-Math-Software`

```
> R<x> := PolynomialRing(IntegerRing());
> x + 1/2
      ^
Runtime error in '+': Bad argument types
Argument types given: RngUPolElt[RngInt], FldRatElt
```

Robert Bradshaw and I also added support for beautiful browser-based 3D graphics to Sage, which involved writing a 3D graphics library, and adapting the free open source JMOL Java library (see `http://jmol.sourceforge.net/`) for rendering molecules to instead plot mathematical objects.

```
sage: f(x,y) = sin(x - y) * y * cos(x)
sage: plot3d(f, (x,-3,3), (y,-3,3), color='red')
```



In 2009, following a huge amount of porting work by Mike Hansen, development of algebraic combinatorics in Sage picked up substantial momentum, with the switch of the entire MuPAD-combinat group to Sage (forming sage-combinat `http://wiki.sagemath.org/combinat`), only months before the formerly free system MuPAD®[9] was bought out by Mathworks (makers of MATLAB). In addition to work on Lie theory by Dan Bump, this also led to a massive amount of work on a category theoretic framework for Sage by Nicolas Thiery.

In 2010, there were 13 Sage Days workshops in many parts of the world, and grant funding for Sage significantly improved, including new NSF funding for undergraduate curriculum development. I also spent much of my programming time during 2010–2011 developing a number theory library called psage `http://code.google.com/p/purplesage/`, which is currently not included in Sage, but can be easily installed.

[9] MuPAD is a registered trademark of SciFace Software GmbH & Co.

Many aspects of Sage make it an ideal tool for teaching mathematics, so there's a steadily growing group of teachers using it: for example, there have been MAA PREP workshops on Sage for the last two years, and a third is likely to run next summer, there are regular posts on the Sage lists about setting up classroom servers, and there is an NSF-funded project called UTMOST (see `http://utmost.aimath.org/`) devoted to creating undergraduate curriculum materials for Sage.

The page `http://sagemath.org/library-publications.html` lists 101 accepted publications that use Sage, 47 preprints, 22 theses, and 16 books, and there are surely many more "in the wild" that we are not aware of. According to Google Analytics, the main Sage website gets about 2,500 absolute unique visitors per day, and the website `http://sagenb.org`, which allows anybody to easily use Sage through their web browser, has around 700 absolute unique visitors per day.

For many mathematicians and students, Sage is today the mature, open source, and free foundation on which they can build their research program.