SAGE: Software for Algebra and Geometry Experimentation

William Stein

April 10, 2006

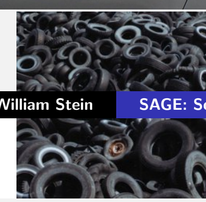
The SAGE Mailing List on Thursday, Feb 2, 2006

Dear SAGE community.

My name is Tiziano and I'm from Italy. I'm writing this mail first of all because I would like to thank you all for SAGE. It's something the world was really missing.

[Every free computer algebra system I've tried has] **"reinvented many times the wheel without being able to build the car."**

Goal of SAGE: **Build the car.**



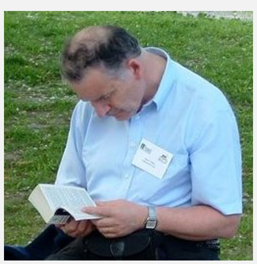
Another Email...

```
Dear William,

[...] I think that you are doing a superb work
with Sage, and thank you for it.

Best,

Henri [Cohen (co-started GP/PARI in 1987)]
```



(But much works remains to be done!)

My Story

- **1997-98: Hecke** and interpreter in C++ (based on other code); for modular forms research with Buzzard and Mazur.
- **1998: D. Kohel:** "too bad you have to write interpreter"; vast amount of Magma code
- **1998: A. Steel:** 2 days in Berkeley teaching me Magma
- **1999-2004:** I wrote **heaps** of Magma code (3 Sydney visits), and tried to convert everyone I met to using it.
- **2000: M. Stoll** –"Magma – Everything under one roof"
- **2004: Frustration**: Magma is closed source, closed development model, and expensive; authorship issues; no user-defined objects; hard to save/load data (no eval command) – not a **mainstream** programming language.

- **S. Hillion** (Berkeley) – Love using Python in my job.
- **Nov 2004: Gonzalo Tornaria (Austin)** – "if I come up with a new algorithm what should I implement it in?"
- **Jan 2005: D. Joyner** – winter AMS meeting; *SAGE is born*
- **One year** of work with many people: David Kohel, *David Joyner*, Iftikhar Burhanuddin, John Cremona, *Martin Albrecht*, Wilson Cheung, *Alex Clemesha*, Neal Harris, Naqi Jaffery, David Kirkby, Jon Hanke, Gregg Musiker, Kyle Schalm, Steven Sivek, Justin Walker, Mark Watkins, Joe Wetherell, Karim Belebas, John Tate, and many others...
- **Feb 4–5: SAGE Days** at UCSD
- *Many more contributors now!* Gonzalo Tornaria, Kiran Kedlaya, Justin Walker, ... and I'm getting new code (and offers of support) from people I've never heard of constantly.

SAGE Days 2006

SAGE Has 3 Distinct Complementary Goals

1. **Distribution** of free open source mathematics software.
2. **New computer algebra system**: structural like Gap and Magma; object-oriented; user extensible; does *not* try to make sense of nonsense like Mathematica, Maple, and PARI do.
3. **Better way to use** all your favorite (**commercial** or free) mathematics software *together*.

1. Distribution of Free Open Source Software

- **Free self-contained** distribution of the very best open source math software that has an *active* community.
- SAGE source tarball: under 50MB; *all* GPL or compatible; **you** can change anything, rebuild, make any changed versions available, etc.
- Type `sage -sdist <version>` to make distro from your local modified version of sage. Type `sage -bdist <version>` to make a binary. Darcs patches.

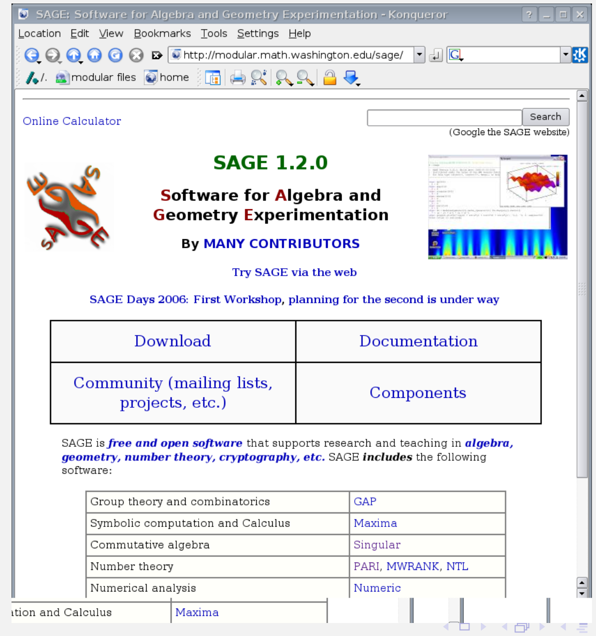
Does Open Source Matter for Math Research?

"You can read Sylow's Theorem and its proof in Huppert's book in the library [...] then you can use Sylow's Theorem for the rest of your life free of charge, but for many computer algebra systems license fees have to be paid regularly [...]. You press buttons and you get answers in the same way as you get the bright pictures from your television set but you cannot control how they were made in either case.

With this situation **two of the most basic rules of conduct in mathematics are violated**: In mathematics **information is passed on free of charge** and **everything is laid open for checking**. Not applying these rules to computer algebra systems that are made for mathematical research [...] means **moving in a most undesirable direction**. Most important: Can we expect somebody to believe a result of a program that he is not allowed to see? "

– J. Neubüser in **1993** (he started GAP in 1986).

The SAGE Website



Not-included With SAGE and Why

1. **NZMATH** – provides inspiration (but not included)
2. **Macaulay2** – supported but not included; working with Dan Grayson right now to make it part of SAGE.
3. **Gnuplot** – screwy license (e.g., I wanted to change C source so paths not hard coded, but this is not allowed!)
4. **KASH** – closed source (but **FREE** and *very* powerful)
5. **Magma** – expensive and closed source (**the dominant system** in arithmetic geometry)
6. **Mathematica / Maple** – expensive and closed source
7. **MATLAB** – no interface, since I don't have it; plan to buy a copy using startup money ($100/year).

2. A New Computer Algebra System

```
$ ls
algebras    databases  __init__.py  modular  schemes
all.py      edu        interfaces   modules  sets
categories  ext        libs         monoids  structure
coding      functions  matrix       plot     tests
crypto      groups     misc         rings    version.py

$ cat */*.py */*/*.py */*.pyx */*/*.pyx |sort|uniq|wc -l
57064

$ cat */*.py */*/*.py */*.pyx */*/*.pyx |sort|uniq|grep "sage
6602         <-------- EXAMPLE INPUT LINES!
```

3. Cooperation – "Everything Under One Roof"

SAGE has many interfaces (**bold included** with SAGE):

- **GAP** (started 1986)– groups, discrete math
- **Singular** (started 1987) – polynomial computation
- **PARI/GP** (started 1987) – number theory
- **Maxima** (started 1967) – symbolic manipulation
- **mwrank, ec, simon, sea** – elliptic curves
- Macaulay2 (started 1993) – commutative algebra
- Gnuplot – 2d and 3d graphics
- KANT/KASH – very sophisticated algebraic number theory
- Magma – vast high-quality research math environment
- Maple – symbolic, educational
- Mathematica – symbolic, numerical, educational
- Octave (started 1992) – numerical analysis

$$-2006 = -1 \cdot 2 \cdot 17 \cdot 59$$

```
sage: (-2006).factor()
-1 * 2 * 17 * 59
sage: gap(-2006).FactorsInt()
[ -2, 17, 59 ]
sage: pari(-2006).factor()
[-1, 1; 2, 1; 17, 1; 59, 1]
sage: maxima(-2006).factor()
-2*17*59
sage: kash(-2006).Factorization()
[ <2, 1>, <17, 1>, <59, 1> ], extended by: ext1 := -1
sage: magma(-2006).Factorization(nvals = 2)   # number of ret
([ <2, 1>, <17, 1>, <59, 1> ], -1)
sage: maple(-2006).ifactor()
-``(2)*``(17)*``(59)
sage: mathematica(-2006).FactorInteger()
{{-1, 1}, {2, 1}, {17, 1}, {59, 1}}
```


Non-math SAGE Components

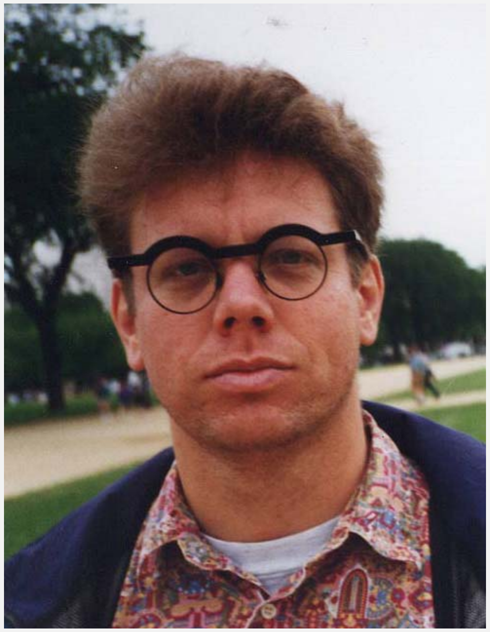
1. **IPython**: Wonderful Interactive Shell
2. **Python**: A **Mainstream** Programming Language (many books; numerous excellent tutorials; constantly being improved by dozens of developers)
3. **Pyrex**: **Compiled** Python-Like Extension Language
4. Saving and Loading Objects (**cPickle** and **ZODB**)

1. IPython: Wonderful Interactive Shell

Under very active development (especially parallel version); widely used by applied math/physics people.

2. Python: A Mainstream Programming Language

- started 1991 by Guido van Rossum (who is now at Google)
- **Numerous libraries** available for networking, graphics, video game programming, numerical analysis, etc.
- A **"gluing language"** (unlike many languages), i.e., easier to use code from other languages.
- **Easy to read** other people's code (unlike, e.g., Perl, C++)
- **Free** and **open source** (unlike, e.g., Java)
- From Python Advocacy FAQ:
  - Run Web sites
  - Write GUI interfaces
  - Control number-crunching code on supercomputers
  - Build test suites for C or Java code



Guido van Rossum

3. Pyrex: Compiled Python-Like Extension Language

```
def factorial(n):
    cdef mpz_t f
    cdef int i
    cdef char* s

    mpz_init(f)
    mpz_set_si(f, n)

    for i from 2 <= i <= n:
        mpz_mul_si(f, f, i)

    s = mpz_get_str(NULL, 32, f)
    r = int(s,32)
    free(s)
    mpz_clear(f)
    return r
```


Pyrex is CRUCIAL to Success of SAGE

1. Written by **Greg Ewing** of New Zealand.
2. Code converted to C code that is **compiled by a C compiler**.
3. Can use any Python functions and objects from Pyrex and any C libraries.
4. **Time-critical** SAGE code gets implemented in Pyrex, which is (as fast as) C code, but easier to read (e.g., since all variables and scopes are explicit).

Pyrex Works

Point of the following is to illustrate using Pyrex. There are more sophisticated algorithms for computing factorials (by balancing the multiplies to take advantage of fast asymptotic arithmetic, and by using prime tables).

```
sage: time n = factorial_pure_python(100000)
CPU times: user 78.72 s, sys: 2.64 s, total: 81.37 s
sage: time v = pari('prod(n = 1,100000,n)')
CPU times: user 18.89 s, sys: 0.19 s, total: 19.08 s
sage: time n = factorial_ZZ(100000)
CPU times: user 8.56 s, sys: 2.22 s, total: 10.79 s
sage: time n = factorial(100000)     # Pyrex (first try)
CPU times: user 6.93 s, sys: 0.00 s, total: 6.93 s
```


4. Saving and Loading Objects

Almost any individual object in SAGE can easily be loaded and saved in a compressed format, as can sessions. This requires **little** programmer support, even for very complicated objects.

```
sage: E = EllipticCurve([1,2,3,4,5])
sage: time v = E.anlist(10^5)
CPU times: user 1.03 s, sys: 0.22 s, total: 1.25 s
Wall time: 1.59
sage: E.save('E')
sage: quit
Exiting SAGE (CPU time 0m1.45s, Wall time 0m25.36s).

$ sage
sage: F = load('E')
sage: time v = F.anlist(10^5)
CPU times: user 0.00 s, sys: 0.00 s, total: 0.00 s
sage: save_session, load_session, ...
```


Help System

1. `function?` gives documentation about function (extracted from source code)
2. `function??` gives the **source code** of function
3. Because Python is so readable, `function??` is incredibly useful and users frequently use it.
4. `help(module or object)` gives man-page like docs
5. **TO DO:** full text search

Attribution

- Whenever possible, files, function docs, and the reference manual **state clearly who the author is**.
- All new code submitted to SAGE must be under a GPL compatible license. Author may optionally keep copyright.
- **Citation:** William Stein and David Joyner, *SAGE: System for algebra and geometry experimentation*, Communications in Computer Algebra (SIGSAM Bulletin) (July 2005), `http://sage.sourceforge.net/`.
- **VERY Important!** Always cite the underlying backends used by SAGE for your work, e.g., GAP, Singular, PARI, Kash, etc. Ask in SAGE forum and/or use `function??` to view source.

Example 1: Bernoulli Numbers

Easy to compare timings in different systems...

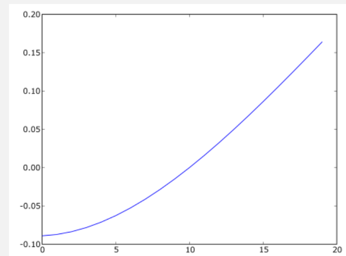
```
sage: a = maple('bernoulli(1000)')     # Wall time: 9.27
sage: a = maxima('bern(1000)')         # Wall time: 5.49
sage: a = magma('Bernoulli(1000)')     # Wall time: 2.58
sage: a = gap('Bernoulli(1000)')       # Wall time: 5.92
sage: a = mathematica('BernoulliB[1000]') #W time: 1.01
  calcbn (http://www.bernoulli.org)    #      Time: 0.06
sage: a = gp('bernfrac(1000)')         # Wall time: 0.00?!
```

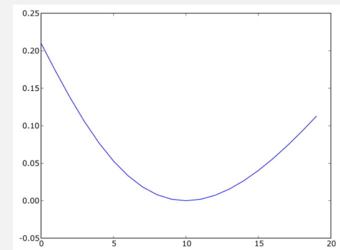
The above led to a paper I'm writing with Kevin McGown...

(NOTE: Mathematica 5.2 is much faster than Mathematica 5.1 at computing Bernoulli numbers, and the timing is almost identical to PARI (for $n > 1000$), though amusingly Mathematica 5.2 is *slow* for $n \leq 1000$!)

Example 2: Elliptic Curves L-series

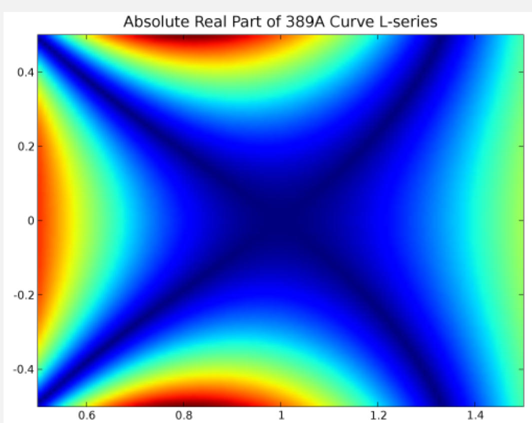
```
sage: E = EllipticCurve('37a'); E
Elliptic Curve defined by y^2 + y = x^3 - x over Rational Field
sage: E.Lseries(1.1)  --> 0.032330449021518493
sage: E.Lseries(1)    --> 0.00000000000000000
sage: plot([float(E.Lseries(float(n)/20)) for n in range(10,30)])
sage: L = EllipticCurve('389a').Lseries
sage: plot([float(L(float(n)/20)) for n in range(10,30)])
```





Example 2 (continued): Complex L-series

```
sage: E = EllipticCurve('389a')
sage: E.Lseries_extended(1+I,  50)
-0.33297168182616760 + 0.37317660446124179*I
sage: E.Lseries_extended(1+0.2*I,  50)
-0.029679202996999034 + 0.0034727623999086183*I
```



Example 3: Birch and Swinnerton-Dyer

```
sage: E = EllipticCurve('37a')
sage: E.sha_an()            --> 1
sage: E.non_surjective()    --> []
sage: E.sha_an()            --> 1
sage: E.regulator()         --> 0.051111408239999996
sage: E.gens()              --> [(0 : 0 : 1)]
sage: E.heegner_discriminants(50)  --> [-3, -4, -7, -11, -40,
sage: E.heegner_index(-7)  # Kolyvagin ==> Sha trivial
[0.999990645298, 1.00000935475]
sage: E.q_expansion(5)
            --> q - 2*q^2 - 3*q^3 + 2*q^4 + O(q^5)
sage: E.simon_two_descent ()
(1, 1, [(0 : 108 : 1)])
sage: E.sea(next_prime(10^30))
1000000000000001426441464441649
```

Summary: Cool Features of SAGE

1. **Mainstream** programming language
2. **Save and load** individual data and sessions
3. DVI and HTML **logging**
4. Easy-to-use **compiled** extension language (can easily use C libraries).
5. **attach, load**; even works with compiled code.
6. All **examples** in documentation **tested**

General To Do

MUCH is left to do. I hope **YOU** will help!

1. Much **new code** still needs to be written for plotting, algebraic geometry, linear algebra, number theory, etc., especially when no open source implementations exist.
2. **Optimization** — parts of SAGE are currently very slow.
3. Many excellent **free packages** need to be included, e.g., genus2reduction, sympow, Rubinstein's L-functions package.
4. **Documentation!** Examples! More Documentation! Even more examples!
5. **Package Distribution:** rpm, msi, deb, pkg, etc. Need user support. The `sage-mindist-*.*.*.tar` package is supposed to make this easier.
6. **STR: Sage Technical Reports:** (Unusual?) Journal; refereed, widely mirrored, subsequent traditional journal publication.

Modular Forms To Do

1. **Optimize modular symbols:** Modular symbols over $\mathbb{Q}$ and cyclotomic fields are implemented for weight $k \geq 2$ and $\Gamma_0(N)$, $\Gamma_1(N)$, and character, but *not optimized*.
2. **Modular forms:** Not even a good design in place.
3. **Dirichlet characters:** Need more documentation and example code. Faster enumeration of characters (to get Eisenstein series).
4. **Bernoulli numbers:** Finish implementation of fast algorithm.
5. **Method of graphs:** Need to implement.
6. **Quaternion algebras:** Half-way done.
7. **Modular abelian varieties:** Nothing in place yet.

Specific Goals at UW

1. **Seminar:** Start a SAGE seminar (weekly talks)
2. **Students:** Have up to 5 undergrads (or grads) working part time on SAGE at a given time
3. **Workshops:** 1-2 international workshops per year