Sage Days 24: The Development of Symbolic Calculus in Sage

William Stein

July 17, 2010

Goals for this workshop

"Ask not what Sage can do for you, but what you can do for Sage."

1. **Goals for newbies. Learn to:**
   - Install Sage,
   - Navigate the documentation,
   - Understand basic structure of the source code,
   - Program in Python and Cython, and
   - Improve Sage, then submit patches for review.
2. **Goals for developers:**
   - Get function fields into Sage (trac #9054 and related).
   - Update Sympy and polish the Sage/Sympy interface.
   - Special functions.
   - Piecewise functions.
   - Runtime addition of attributes to symbolic expressions.
   - Clashing of variable names with Maxima.
   - Easy access to ripping apart symbolic expressions
   - C library interface to Maxima.

Part 1: The Past

Background: The Goal of the Sage project

Create a viable free open source alternative to Magma, Maple, Mathematica, and Matlab

A viable alternative will have:

- All the *mathematical features* of Magma, Maple, Mathematica, and Matlab with *comparable (or better) speed*, including sophisticated symbolic calculus, special functions, global fields, easy of use and documentation.
- Beautiful interactive 2d and 3d graphics.
- A notebook interface and an IDE.
- Commercial support (including customized notebook servers)
- Many undergraduate books (e.g., Zimmerman et al.!)

Sage does *not* need to be able to run programs written in the Ma*'s own custom math-only languages. Sage 'aint Octave.

2005: I started the Sage project

SAGE = Software for Arithmetic Geometry Experimentation

- I want an *open* alternative to Magma **(ask me why)**, and I have an extensive research program I've built on it over 6 years. David Joyner (coding theorist) has similar concerns.
- In 2005 – number theory and some coding theory (GAP) – no Symbolic Calculus.

```
sage: E = EllipticCurve('389a'); E
Elliptic Curve defined by y^2 + y = x^3 + x^2 - 2*x
sage: E.gens()
[(-1 : 1 : 1), (0 : -1 : 1)]
```

```
sage: G = matrix(GF(5),4,7,[1,1,1,0,0,0,0,1,0,0,1,1,0,0,0,1,0...
sage: C = LinearCode(G); C
Linear code of length 7, dimension 4 over Finite Field of size 5
sage: C.minimum_distance()
3
```

(Why not Magma?)

1. Commercial: Expensive for my collaborators and students
2. Closed: Magma's implementation of algorithms a trade secret
3. Frustrating: Tight control by John Cannon
4. Static: Users can't define their own classes (data types)
5. Copy protection: A pain in the ass.
6. Language: No `eval`, no exception handling, no namespaces, little new development on the language, math-only language.
7. Developer community: far too small, no public mailing list
8. No graphics, no symbolic calculus, no graphical user interface.
9. No public bug tracker or list of reported bugs
10. No compiler (nothing like Cython)
11. ...

(Similar remarks about Maple.)

2006: Maxima/Sage interface

- I taught Calculus (in San Diego) during this time, and used Maxima + GNUplot for demos, and for plots in lecture notes.
- I integrated Maxima into Sage.
- David Joyner wrote *Sage Constructions* – examples of how to do Calculus using the Sage/Maxima interface.

```
sage: f = maxima('1/sqrt(x^2+2*x-1)')
sage: f.integrate(x)
log(2*sqrt(x^2+2*x-1)+2*x+2)
sage: maxima.plot2d('cos(2*x) + 2*exp(-x)','[x,0,1]',
                    '[plot_format,openmath]')
[[a plot appears]]
```

2007: Bobby Moretti (UW undergraduate)

- 2007: Bobby Moretti (UW undergraduate): wrote a pure Python + Maxima symbolic Calculus package.
- Idea and implementation was Bobby's from start to finish
- Made Sage an option for Calculus classes, but too slow compared to Ma* and not flexible enough.

```
sage: f = 1/sqrt(x^2 + 2*x - 1); f
1/sqrt(x^2 + 2*x - 1)
sage: f.integrate(x)
log(2*x + 2*sqrt(x^2 + 2*x - 1) + 2)
```

2008: Gary Furnish (Utah undergraduate): New Symbolics!!

- 2008: Gary Furnish: another undergrad, tried to completely rewrite symbolic calculus in Cython.
- Wrote a lot of Cython code, which lacked a sufficiently mature foundation, so we didn't end up using it.

> *"Ugly performance enhancing software hacks are often needed to make theoretical algorithms viable."*
> *– Gary Furnish, sage-devel, 2008-05-01.*

2008–2009: GiNAC/Pynac

- July 2008: ISSAC – during the boat trip, Burcin suggested investigating integrating **GiNAC** into Sage.
- Big problem: GiNAC depends on the huge C++ CLN library (basic rings/fields arithmetic), which takes a long time to build, and provides nothing that isn't already in Sage.
- Aug 2008: I wrote the first version of Pynac during an intense two week coding sprint in San Diego.

> **Key Idea:** Rip out the CLN dependence of GiNAC and replace it by Sage/Python. Amazingly, this worked.

- Pynac is a low level C++ library that depends on Sage/Python and provides fast symbolic arithmetic. No integration, series, or other high level algorithms.

Aug 2008 – May 2009: Build complete new symbolic system in Sage on Pynac

- Finish Pynac and build a new fast symbolic system on top: Burcin (mostly) + Mike Hansen + Carl Witty + Robert Bradshaw + Nick Alexander + me + ⋯
- This is the main new feature of Sage-4.0.
- Maxima is still used for some things, e.g., symbolic integration, some comparisons, etc.

```
sage: f = 1/sqrt(x^2 + 2*x - 1); f
1/sqrt(x^2 + 2*x - 1)
sage: timeit('f*f')
625 loops, best of 3: 83.5 µs per loop
sage: g = maxima(f)
sage: timeit('g*g')
125 loops, best of 3: 1.41 ms per loop
sage: 1.41/.0835
16.8862275449102
```

May 2009 - today: Further development

- May 2009 - today: Burcin (mostly) has further developed symbolics in Sage
- Pynac has been very stable (hardly any changes needed).
- Nils Bruin (and Robert Bradshaw): A new C library interface to Maxima.

Sympy: a standalone pure Python symbolic calculus library

Sympy is a Python project mainly by Physics, Engineering, and Science students...

- 2005: Sympy project started by Ondrej Certik (physics undergraduate at the time)
- 2007?: Sympy included in Sage
- They have many very hard working developers.

```
sage: f = 1/sqrt(x^2 + 2*x - 1); f
1/sqrt(x^2 + 2*x - 1)
sage: g = f._sympy_()
sage: import sympy; sympy.pretty_print(g)
        1
__________________
   _______________
  /              2
\/  -1 + 2⋅x + x
```

Unflattering Demo of using Sympy in Sage

```
sage: var('x')
sage: f = 1/sqrt(x^2 + 2*x - 1); f
1/sqrt(x^2 + 2*x - 1)
sage: time f.integrate(x)
log(2*x + 2*sqrt(x^2 + 2*x - 1) + 2)
Time: CPU 0.01 s, Wall: 0.02 s
sage: f.integrate(x, algorithm='sympy')
Traceback (most recent call last):
AttributeError: 'Integral' object has no attribute '_sage_'

sage: from sympy import sqrt, integrate, var
sage: x = var('x')
sage: f = int(1)/sqrt(x**2 + 2*x - 1); f
(-1 + 2*x + x**2)**(-1/2)
sage: time f.integrate(x)
Integral((-1 + 2*x + x**2)**(-1/2), x)
Time: CPU 0.64 s, Wall: 0.65 s
```

Part 2: The Future

Roadmap of Sage project

- Sage-5.0 (2010)
- Sage-6.0 (2011)
- Sage-7.0 (2012)

Roadmap of Sage project

- Sage-5.0 (August 31, 2010)
  - Windows port via Cygwin
  - Upgrade PARI to the latest SVN version
  - Upgrade MPIR to version 2.x
  - Get doctest coverage to 90% (currently at 84.6%)
- Sage-5.x (rest of this year...)
  - Function fields (Hess's algorithms, 2-descent, L-functions)
  - Special functions: support everything in the DLMF:
    - arbitrary precision numerical evaluation (mpmath)
    - series expansion
    - differentiation
    - shifts (difference analogue of differentiation)
  - Symbolic summation
  - Symbolic integration (Risch-Norman algorithm)
- Sage-6.0 (2011)
- Sage-7.0 (2012)

Roadmap of Sage project

- Sage-5.0 (August 31, 2010).
- Sage-6.0 (2011), some potential goals:
  - Noncommutative symbolic computation (Plural, Weyl Algebras, Graded algebras, Lie algebras).
  - Much improved Sage notebook (scalability, customizability)
  - Textbooks, interacts, etc., integrated with the sage distribution
  - Commercial support (custom notebook servers)
  - Fully switch to using C library interfaces for GAP and Maxima.
  - Get doctest coverage to 100%
- Sage-7.0 (2012)

Roadmap of Sage project

- Sage-5.0 (August 31, 2010).
- Sage-6.0 (2011):
- Sage-7.0 (2012), some potential goals:
  - Fast Groebner basis computation that is competitive with Magma/Maple
  - Vast improvements in Sage for Science and Engineering (documentation, diff'eq, data workflows, reproducible research, instrument support, data formats like HD5)
  - Statistics: something Pythonic/Cythonic and built on top of R + scipy.stats + GSL, which competes with SAS, etc.
  - Switch to Python 3.x
  - Much randomized and unit testing that goes beyond doctesting

... World Domination

Any questions?