SAGE: System for Arithmetic Geometry Experimentation

http://modular.fas.harvard.edu/SAGE/

William Stein

Asst. Professor of Mathematics, Harvard University

March 24, 2005, PYCON, Washington, D.C.

Arithmetic Geometry

Arithmetic geometry is about geometric questions that have an arithmetic flavor. Sample famous problems:

- **Fermat's Last Theorem:** $x^n + y^n = z^n$ has no solutions with $n \geq 3$ and x, y, z all nonzero integers. Andrew Wiles proved this in 1995 using elliptic curves and modular forms.

- **The Birch and Swinnerton-Dyer Conjecture:** Discovered from computations in the 1960s. Simple criterion for whether or not for given a, b the elliptic curve $y^2 = x^3 + ax + b$ has infinitely many rational solutions. (Clay \$$10^6$ dollar problem.)

- **The Riemann Hypothesis:** Nontrivial zeros of Riemann Zeta function are on line $\mathrm{Re}(s) = 1/2$. Solution gives deep understanding of distribution of the prime numbers $2, 3, 5, 7, 11, 13, \ldots$. (Clay million dollar problem.)

- **Cryptography:** Factor integers quickly. E.g., Hendrik Lenstra, used elliptic curves to give a new algorithms for this. The number field sieve is a sophisticated algorithm for factoring and uses computation in number fields. Also, cryptosystems come from elliptic curves over finite fields.

The Problem

Create a system for doing computations with the mathematical objects mentioned above. Main Goals:

- **Efficient:** Be very fast – comparable to or faster than anything else available. This is very difficult, since most systems are closed source, algorithms are often not published, and finding fast algorithms is often extremely difficult (years of work, Ph.D. theses, luck, etc.)

- **Open Source:** The source code must be available and readable, so users can understand what the system is really doing and trust the results more.

- **Comprehensive:** Implements enough different things to be really useful.

- **Well documented:** Reference manual, API reference with examples for every function, and at least one published book. Make documentation and source a peer reviewed package, so get academic credit like a journal publication.

- **Extensible:** Be able to define new data types or derive from builtin types, or make code written in your favorite language part of the system.

- **Free:** Must be sufficiently free (at least GPL).

Existing Mature Systems

- **Mathematica, Maple, and MATLAB:** Arithmetic geometers are not their target audience. Mathematica does well at special functions, and both do calculus very well, which is almost never useful in arithmetic geometry. These systems are *closed source, very expensive, for profit*.

- **MAGMA:** *By far* the best software for arithmetic geometry. It's very efficient, comprehensive, and well documented. Great design and class hierarchy. BUT: It's closed source (but non-profit), expensive, and not easily extensible (no user defined types or C/C++-extension code). I've contributed substantially to MAGMA.

- **PARI:** Efficient, open source, extendible and free. But the documentation is not good enough and the memory management is not robust enough. Also, PARI does not do nearly as much as what is needed.

- **Maxima, Octave, etc.:** Open source, but not for arithmetic geometry.

(There's always something else that I don't know about.)

All these system use their own custom programming language.

SAGE: System for Arithmetic Geometry Experimentation

I am creating a system using Python that will hopefully solve the problem. I've been working on this for one year (and I've been writing arithmetic geometry software since 1998, in C++ and for MAGMA). Please download and try it, though it is still *far from ready* for prime time:

`http://modular.fas.harvard.edu/SAGE/`.

- *Like SciPy/Numarray but for arithmetic geometry.*

- **Financial Support:** My NSF Grant DMS-0400386, and hopefully grad students when I go to UC San Diego as a tenured professor in July. I intend to apply for other grants as well. This is a very long-term project.

- **Tools:** Python, Pyrex, GMP, PARI, mwrank, SWIG, NTL. These are all are GPL'd and SAGE provides (or will provide) a unified interface for using them. Also, I'm writing lots of new code, mainly related to my areas of expertise (modular forms, and linear algebra over exact fields). Not reinventing wheel. Using design ideas from MAGMA.

- **Current Platforms:** Linux, Solaris, OS X, Windows (cygwin). Architecture independent, so no use of psyco.

Advantages to Using Python

Asside from being open source, building an arithmetic geometry system on Python has several advantages.

- **Object persistence** is very easy in Python—but very difficult in many other math systems.
- Good support for **doctest** and automatic extraction of API documentation from docstrings. Having lots of examples that are tested and guaranteed to work as indicated.
- Memory management: MAGMA also does reference counting but does *not* deal with **circular references**. Python does.
- Python has **many packages** available now that might be useful to arithmetic geometers: numarray/Numeric/SciPy, 2d and 3d graphics, networking (for distributed computation), database hooks, etc.
- Easy to **compile** Python on many platforms.

Standard Python Math Annoyances

Everybody who does mathematics using Python runs into these problems:

- `**` versus `^` – easy for me to get used to, but must define `^` to give an error on any arithmetic SAGE type. (In IPython shell lines can be pre-parsed, so this can be solved nicely.)
- `sin(2/3)` has *much* different behavior in Python than in any standard math system.

I think the best solution is to leave the Python language exactly as is, and write a pre-parser for IPython so that the command line behavior of IPython is what one expects in arithmetic geometry, e.g., typing `a = 2/3` will create `a` as an exact rational number, and typing `a = 3^4` will set `a` to 81. One must still obey the standard Python rules when writing code in a file.

Demo

```
was@form:~$ sage

**********************************************************
*  SAGE Version 0.2                                      *
*  System for Arithmetic Geometry Experimentation        *
*  (c) William Stein, 2005                               *
*  http://modular.fas.harvard.edu/SAGE                   *
*  Distributed under the terms of the GPL                *
**********************************************************
Help: object? -> Documentation about 'object'.

>>> Q
     Rational Field
>>> a = Q("2/3")
>>> a
     2/3
>>> a in Q
     True
>>> type(a)
     <type 'rational.Rational'>
>>> 3^4    # illustration of IPython hack!
     81
>>> 3\^4   # usual Python behavior (xor)
     7
```

Create a matrix as an element of a space of matrices.

```
>>> M = MatrixSpace(Q,3)
>>> M
Full MatrixSpace of 3 by 3 dense matrices over Rational Field
>>> B = M.basis()
>>> len(B)
9
>>> B[1]
[0 1 0]
[0 0 0]
[0 0 0]
>>> A = M(range(9)); A
[0 1 2]
[3 4 5]
[6 7 8]
>>> A.reduced_row_echelon_form()
[ 1  0 -1]
[ 0  1  2]
[ 0  0  0]
>>> A^20
[ 2466392619654627540480  3181394780427730516992  3896396941200833493504] ...
>>> A.kernel()
Vector space of degree 3, dimension 1 over Rational Field
Basis matrix:
[ 1 -2  1]
>>> kernel(A)   # functional notation, like in MAGMA
```

Compute with an elliptic curve.

```
>>> E = EllipticCurve([0,0,1,-1,0])
>>> E
     y^2 + y = x^3 - x
>>> P = E([0,0])
>>> 10*P
     (161/16, -2065/64)
>>> 20*P
     (683916417/264517696, -18784454671297/4302115807744)
>>> E.conductor()
     37
>>> print E.anlist(30)    # coefficients of associated modular form
    [0, 1, -2, -3, 2, -2, 6, -1, 0, 6, 4, -5, -6, -2, 2, 6, -4, 0, -12,
     0, -4, 3, 10, 2, 0, -1, 4, -9, -2, 6, -12]
>>> M = ModularSymbols(37)
>>> D = M.decomposition()
>>> D
[Subspace of Modular Symbols of dimension 1 and level 37,
 Subspace of Modular Symbols of dimension 2 and level 37,
 Subspace of Modular Symbols of dimension 2 and level 37]
>>> D[1].T(2)
Linear function defined by matrix:
[0 0]
[0 0]
>>> D[2].T(2)
Linear function defined by matrix:
[-2  0]
[ 0 -2]
```