SAGE: System for Algebra and Geometry Experimentation

http://modular.ucsd.edu/sage/

William Stein

Assoc. Professor of Mathematics, UCSD

Sept 15, 2005, SANDPYT

What is SAGE?

SAGE is a *Python* framework for number theory, algebra, and geometry computation. It is open source and freely available under the terms of the GNU General Public License (GPL).

- Mainly written by William Stein (other contributors include David Joyner, David Kohel, Kyle Schalm, Justin Walker, etc.)

- SAGE is a Python library with a customized interpreter. It is written in Python, C++, C, and Pyrex. (**No** use of SWIG or psycho.)

- Interface to open source libraries for computing with elliptic curves (mwrank), number theory (PARI, NTL), commutative algebra (Singular), and group theory (GAP). Not finished.

- Design influenced by the non-free computer algebra program MAGMA. Focuses on a class structure that reflects the mathematical objects that mathematicians think about, rather than symbolic manipulation.

SAGE in Action

```
$ sage
------------------------------------------------------------------------
  SAGE Version 0.6.beta2, Export Date: 2005-09-08-1814
  Distributed under the terms of the GNU General Public License (GPL)
  IPython shell -- for help type <object>?, <object>??, %magic, or help
------------------------------------------------------------------------

sage: p = next_prime(10^15)
sage: p
 _2 = 1000000000000037
sage: q = next_prime(10^17)
sage: n = p*q
sage: factor(n)
 _5 = [(1000000000000037, 1), (100000000000000003, 1)]
sage: a = 1/3
sage: a
 _7 = 1/3
sage: type(a)
 _8 = <type '_rational.Rational'>
sage: a.parent()
 _9 = Rational Field
sage: Q = _
sage: R = PolynomialRing(Q, 'a');  a = R.gen(0)
sage: f = a^2 +a + 1
sage: f^3
```

```
_13 = a^6 + 3*a^5 + 6*a^4 + 7*a^3 + 6*a^2 + 3*a + 1
sage: parent(f)
_14 = Univariate Polynomial Ring in a over Rational Field
sage: S = R.quotient(f, 'b'); b = S.0
sage: S
_17 = Univariate Quotient Polynomial Ring in b over
Rational Field with modulus a^2 + a + 1
sage: b+1
_18 = b + 1
sage: b^2
_19 = -b - 1
sage: V = VectorSpace(Q,4)
sage: V
_21 = Full Vector space of degree 4 over Rational Field
sage: V.random_element()
_22 = (1, 1, 1, 1)
sage: k = GF(23^2, 'a'); a = k.0
sage: k
_28 = Finite field of size 23^2
sage: a^5
_26 = 4*a + 14
```

Target Audience

- **Number theorists:** people who study problems related to Fermat's last theorem, distribution of primes (the Riemann Hypothesis), arithmetic of elliptic curves, etc.

- **Cryptography researchers:** people who improve understanding of cryptosystems. Most software for crypto is non-free, for obvious reasons.

Main Goals

- **Efficient:** Be very fast – comparable to other systems. **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.) Means much non pure Python code.

- **Free and Open Source:** The source code must be available and readable, so users can understand what the system is really doing. Math research papers are easier to verify, extend and use when they rely only on free open software.

- **Well documented:** Reference manual and an API reference with example usage for every function. Make documentation and source a peer reviewed package, so get academic credit like a journal publication.

Existing Mature Systems

- **Mathematica, Maple, and MATLAB:** Number theory / cryptograhy is not the target audience. Mathematica does well at special functions, and both do calculus very well. These systems are *closed source, expensive, for profit*.

- **MAGMA:** *By far* the best software for number theory and cryptography research. It's efficient, comprehensive, and well documented. Great design (but could be better). 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 number theory.

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

All *these system use their own custom programming language.*

SAGE: System for Algebra and Geometry Experimentation

I am creating a system using Python that will hopefully solve the problem. I've been working on this for 8 months (and I've been writing number theory software since 1998, in C++ and MAGMA). Please download and try it, though it is still *not ready* for prime time:

`http://modular.ucsd.edu/sage`.

- *SciPy/Numarray-ish:* but for number theory.
- **Financial Support:** NSF Grant DMS-0400386.
- **Implementation:** C/C++, Python, and Pyrex (easy access to C code from Python, and compiled). Wrapping C++ libraries by wrapping in C and using Pyrex (avoid SWIG for simplicity and performance).
- **Libraries:** GMP, MPFR, PARI, mwrank, NTL, etc. These are all are GPL'd and SAGE provides (or will provide) a unified interface for using them. Not reinventing wheel.
- **Current Platforms:** Linux, OS X, and Windows. Architecture independent, so *no use of psyco*.

Advantages to Using Python

Asside from being open source, building SAGE using Python has several advantages.

- **Object persistence** is easy in Python—but 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**: Python deals well with **circular references**.
- Python has **many packages** available now that may prove useful in number theory/crypto research: numarray/Numeric/SciPy, 2d and 3d graphics, networking (for distributed computation), 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.

Solution: 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 mathematics, 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. This was easy to do using IPython.

More Examples

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

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

```
sage: M = MatrixSpace(Q,3)
sage: M
Full MatrixSpace of 3 by 3 dense matrices over Rational Field
sage: B = M.basis()
sage: len(B)
9
sage: B[1]
[0 1 0]
```

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

Compute with an elliptic curve.

```
sage: E = EllipticCurve([0,0,1,-1,0])
sage: E
     y^2 + y = x^3 - x
sage: P = E([0,0])
sage: 10*P
     (161/16, -2065/64)
sage: 20*P
```

```
     (683916417/264517696, -18784454671297/4302115807744)
sage: E.conductor()
     37
sage: 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]
sage: M = ModularSymbols(37)
sage: D = M.decomposition()
sage: 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]
sage: D[1].T(2)
Linear function defined by matrix:
[0 0]
[0 0]
sage: D[2].T(2)
Linear function defined by matrix:
[-2  0]
[ 0 -2]
```