SAGE: **S**ystem for **A**lgebra and **G**eometry **E**xperimentation*

William Stein
University of California at San Diego
wstein@ucsd.edu

David Joyner
Mathematics Department, US Naval Academy
wdj@usna.edu

SAGE is a framework for number theory, algebra, and geometry computation that is initially being designed for computing with elliptic curves and modular forms. The current implementation is primarily due to William Stein. It is open source and freely available under the terms of the GNU General Public License (GPL).

SAGE is a Python library with a customized interpreter. It is written in Python, C++, and C (via Pyrex). Python is an open source object-oriented interpreted language, with a large number of libraries, e.g., for numerical analysis, which are available to users of SAGE. Python can also be accessed in library mode from C/C++ programs.

SAGE will provide a unified interface to several important open source libraries, including John Cremona's MWRANK library for computing with elliptic curves, the PARI library for number theory, and Shoup's number theory library NTL. There is also a tentative plan to incorporate an interface to SINGULAR (for commutative algebra) and GAP (for group theory).

The design of SAGE is heavily influenced by the carefully thought out and mature class structure of the closed source computer algebra program MAGMA. When possible, classes have similar names, constructors, and print representations as in MAGMA, and similar functions are available. However, SAGE is not meant to be an exact clone of MAGMA.

The main longterm goals for SAGE:

- Efficient: Be very fast—comparable to or faster than anything else available. This is very difficult, since many systems are closed source, algorithms are sometimes not published, and finding fast algorithms is often extremely difficult (years of work, Ph.D. theses, luck, etc.).
- Free and open source: The source code must be freely available and readable, so users can understand what the system is really doing and more easily extend it. Just as mathematicians gain a deeper understanding of a theorem by carefully reading or at least skimming the proof, people who do computations should be able to understand how the calculations work by reading documented source code.
- Easy to compile: SAGE should be relatively easy to compile from source for Linux and OS X users. This provides more flexibility in modifying the system.

*This material is based upon work supported by the NSF under Grant No.0400386.

- Comprehensive: Implement enough algorithms to be really useful.
- Tools: Provide robust interfaces to some of the functionality of PARI, GAP, GMP, SINGULAR, MWRANK, and NTL. These are all are GPL'd and SAGE provides (or will provide) a unified interface for using them.
- Cross-platform: SAGE runs under Linux, OS X, Windows (cygwin). (Solaris is not supported yet.)
- Well documented: Reference manual, API reference with examples for every function, and an extensive tutorial.
- Extensible: Be able to define new data types or derive from builtin types, and make code written in your favorite language (including C/C++) part of the system.
- User friendly: The hope is to eventually attain a high level of user support. (The *GAP Forum* email list is an ideal example of the support it is hoped that SAGE can attain.)

As mentioned above, SAGE runs on many platforms and operating systems (Windows, Mac OS X, FreeBSD, and Linux—both 32-bit and 64-bit). Moreover, it can be downloaded both as a "binary" and as source code that you compile yourself. You can download SAGE, an installation guide and a tutorial, from its webpage `http://sage.sourceforge.net`. The windows version has an easy installer and the linux compilation is very straightforward if you have the tools (make, gcc, etc.). Although SAGE uses Python and PARI and other packages, keep in mind that it is not necessary to have this software preinstalled on your computer. The installation of SAGE is really designed to be relatively painless, but if you have any problems, please ask. Moreover, if you have installed SAGE once, upgrading to the newest version is especially easy with the "make web-update" command (this is described on the website given above and assumes you have wget installed and an internet connection).

In this brief space, we only give one example of SAGE. Here is an example of computing with an elliptic curve. The `EllipticCurve` command has several forms:

- `EllipticCurve([`a_1`,`a_2`,`a_3`,`a_4`,`a_6`])`: Elliptic curve $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$, where the a_i's are coerced into the parent of the first element. If all are integers, they are coerced into the rational numbers.
- `EllipticCurve([`a_4`,`a_6`])`: Same as above, but $a_1 = a_2 = a_3 = 0$.
- `EllipticCurve(label)`: Returns the elliptic curve over $\mathbf{Q}$ from the Cremona database with the given label. The label is a string, such as `"11A"` or `"37B2"`.
- `EllipticCurve(R, [`a_1`,`a_2`,`a_3`,`a_4`,`a_6`])`: Create the elliptic curve over a ring R with given a_i's as above.

Some examples using this command:

```
sage: EllipticCurve([0,0,1,-1,0])
      Elliptic Curve defined by y^2 + y = x^3 - x over Rational Field
sage: EllipticCurve([GF(5)(0),0,1,-1,0])
      Elliptic Curve defined by y^2 + y = x^3 - x over Finite field of size 5
sage: EllipticCurve(GF(5), [0, 0,1,-1,0])
      Elliptic Curve defined by y^2 + y = x^3 - x over Finite field of size 5
```

Here `GF(5)(0)` denotes the zero element of the field with 5 elements. As indicated above, the last two commands are essentially equivalent.

The elliptic curves over the complex numbers (denoted `CC` in SAGE) are parameterized by the j-invariant. SAGE can compute these j-invariants:

```
sage: E = EllipticCurve([CC(0),0,1,-1,0])
sage: E
      Elliptic Curve defined by y^2 + y = x^3 - x over Complex Field
sage: E.j_invariant()
      2988.97297297297297297
```

Obviously $x = y = 0$ is a point on the elliptic curve $E:\ y^2 + y = x^3 - x$. To create this point in SAGE type `E([0,0])`. SAGE can add points on such an elliptic curve (recall elliptic curves support an additive group structure where the "point at infinity" is the zero element and three co-linear points on the curve add to zero):

```
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)
```

Finally, SAGE can use the Cremona database for elliptic curves[1].

```
 sage: for E in curves(prime_range(100)):
   ...:        print "%s\t%s"%(E.cremona_label(), E.rank())
   ...:
11A1    0
11A2    0
11A3    0
17A1    0
```

[1]All these commands have been tested using the 6-2-2005 version of SAGE with the Cremona database `db-elliptic-curves-0.3`. However, only the commands in the "`for` loop" require the database, which must be installed separately (as explained on the SAGE website). Note also that the `curves` command name may change to `elliptic_curves` in a future version of SAGE.

```
17A2    0
17A3    0
17A4    0
19A1    0
19A2    0
19A3    0
37A1    1
37B1    0
37B2    0
37B3    0
43A1    1
53A1    1
61A1    1
67A1    0
73A1    0
73A2    0
79A1    1
83A1    1
89A1    1
89B1    0
89B2    0
```

This is just a quick illustration of the many elliptic curve computations that SAGE can already do. For further examples (using linear algebra, modular forms, and so on), please see the SAGE tutorial.

SAGE is in an early stage of development, but is actively growing, and is already usable. There is a SAGE discussion board, bug-tracker, and wish list. If you are interested in using such a system, or in contributing in any way to its development, please visit `http://sage.sourceforge.net` or email William Stein at `wstein@ucsd.edu`.