Sage and Magma

Sage and Magma

June 2009, William Stein

Part 1: The Sage/Magma Interface

Sage controls Magma through a **pseudo tty interface**:

> The pty driver provides support for a device-pair termed a pseudo terminal. A pseudo terminal is a pair of character devices, a master device and a slave device. The slave device provides to a process an interface identical to that described in tty(4). However, whereas all other devices which provide the interface described in tty(4) have a hardware device of some sort behind them, the slave device has, instead, another process manipulating it through the master half of the pseudo terminal. That is, anything written on the master device is given to the slave device as input and anything written on the slave device is presented as input on the master device.

Demo - magma.eval -- string evaluation in Magma with output a string

```
magma.eval('2 + 3')
```

```
'5'
```

```
magma.eval('a := 5; print a + 3')
```

```
'8'
```

```
magma.eval('a^2')
```

```
'25'
```

Demo -- creating Magma objects in Sage. Contains pointer to object in running Magma session.

```
a = magma('2/3'); a
```

```
2/3
```

```
type(a)
```

```
parent(a)
```

```
a = magma(38493); a
```

```
a.Factorization()
```

Demo -- Conversion of matrices to Magma is (often) blazingly fast. Below we compare two det calculations.

```
m = random_matrix(ZZ,200,x=2^128)
```

```
len(m.str())
```

```
1600399
```

```
time z = magma(m)
```

```
Time: CPU 0.07 s, Wall: 0.33 s
```

```
time d1 = m.det()
```

```
Time: CPU 3.64 s, Wall: 4.22 s
```

```
time d2 = magma(m).Determinant()
```

```
Time: CPU 0.07 s, Wall: 10.38 s
```

```
d1 == d2
```

```
True
```

Demo -- Converting a more complicated object to Magma

```
k.<a> = GF(16); E = EllipticCurve([a,1,a+1,a^2,0]); E
```

```
Elliptic Curve defined by y^2 + a*x*y + (a+1)*y = x^3 + x^2 + a^2*x
over Finite Field in a of size 2^4
```

```
E.points()
```

```
[(0 : 0 : 1), (0 : 1 : 0), (0 : a + 1 : 1), (1 : a^3 : 1), (1 : a^3
+ 1 : 1), (a + 1 : a^3 + 1 : 1), (a + 1 : a^3 + a^2 : 1), (a^2 + a :
a^2 + a + 1 : 1), (a^2 + a : a^3 : 1), (a^3 : 0 : 1), (a^3 + 1 : 0 :
1), (a^3 + 1 : a : 1), (a^3 + a : a + 1 : 1), (a^3 + a : a^2 + a + 1
: 1), (a^3 + a + 1 : a + 1 : 1), (a^3 + a + 1 : a^2 + 1 : 1), (a^3 +
a^2 : a^2 + a : 1), (a^3 + a^2 : a^3 + a^2 + a : 1), (a^3 + a^2 + 1
: a^2 + a + 1 : 1), (a^3 + a^2 + 1 : a^3 + a^2 + 1 : 1), (a^3 + a^2
+ a + 1 : 1 : 1), (a^3 + a^2 + a + 1 : a^3 + a^2 + a + 1 : 1)]
```

```
E.cardinality()
```

```
22
```

```
F = magma(E); F
```

```
Elliptic Curve defined by y^2 + a*x*y + a^4*y = x^3 + x^2 + a^2*x
over GF(2^4)
```

```
magma.eval('#%s'%F.name())
```

```
'22'
```

```
F.Points()
```

```
{@ (0 : 1 : 0), (1 : a^3 : 1), (1 : a^14 : 1), (a^3 : 0 : 1), (a^4 :
a^6 : 1), (a^4 : a^14 : 1), (a^5 : a^3 : 1), (a^5 : a^10 : 1), (a^6
: a^5 : 1), (a^6 : a^11 : 1), (a^7 : a^4 : 1), (a^7 : a^8 : 1), (a^9
: a^4 : 1), (a^9 : a^10 : 1), (a^12 : 1 : 1), (a^12 : a^12 : 1),
(a^13 : a^10 : 1), (a^13 : a^13 : 1), (a^14 : 0 : 1), (a^14 : a :
1), (0 : 0 : 1), (0 : a^4 : 1) @}
```

```

```

```

```

```

```

```

```

```

```

Demo - Converting a Nested Ring to Magma

```
R.<x> = QQ[]
S.<Y> = R[]
T.<Z,x> = S[]      # note that I use x again, with no bad impact
below!!
T
```

```
Multivariate Polynomial Ring in Z, x over Univariate Polynomial Ring
in Y over Univariate Polynomial Ring in x over Rational Field
```

```
magma(T)
```

```
Polynomial ring of rank 2 over Univariate Polynomial Ring in Y over
Univariate Polynomial Ring in x over Rational Field
Order: Graded Reverse Lexicographical
Variables: Z, x
```

```
magma(Y) + magma(Z)
```

```
Z + Y
```

```
magma(R.gen()) + magma(x)
```

```
x + x
```

Interface Improvements (Summary)

Most rings and matrices now convert very efficiently.

This involved a CCR-funded *substantial* redesign of the interface.

Now it is far far more robust than it was before (e.g., memory management, nested references, multiple copies of the same thing -- tons of subtle issues addressed)

How to implement a Sage --> Magma conversion

Just define **_magma_init_** method -- look at the many examples in Sage:

```
R.<x> = QQ[]
```

```
R._magma_init_??
```

```
<bound method PolynomialRing_field._magma_init_ of Univariate
Polynomial Ring in x over Rational Field>
```

```
search_src('_magma_init_')
```

How to implement a Magma --> Sage conversion

Add a new function like the following to the file basic.m in SAGE_ROOT/data/extcode /magma/sage

```
intrinsic Sage(X::RngUPolElt) -> MonStgElt, BoolElt
{}
  return Sprintf("%o(%o)", Sage(Parent(X)), Sage(Coefficients(X))), false;
end intrinsic;
```

See the file basic.m mentioned above for tons of examples.

Part 2: What can Magma do that Sage can't?

This part of the talk is entirely on *functionality* present in Magma that is not available in Sage. We will mostly ignore questions of efficiency.

We follow the magma "Handbook" Version 2.15.

http://magma.maths.usyd.edu.au/magma/htmlhelp/MAGMA.htm

For each section of that handbook we list areas where Magma is functionally ahead of Sage in a significant way. We're not getting bogged down in details, but explaining in rough strokes the main functional areas where Magma has capabilities that Sage doesn't. In many cases we silently don't mention sections that Sage already covers.

I. The Magma Language

Sage can't interpret programs written in the Magma language yet. Fortunately, Python is fairly close to Magma as a language.

"There's a nice lexical analyzer/parser package in python called PLY (Python Lex-Yacc) which is GPL'ed, that might be nice to include in Sage, if, in the future, people might be motivated to write translators from other languages to SAGE." -- Victor Miller

"In a nutshell, PLY is nothing more than a straightforward lex/yacc implementation. PLY doesn't try to do anything more or less than provide the basic lex/yacc functionality. In other words, it's not a large parsing framework or a component of some larger system." -- from http://www.dabeaz.com/ply

PROJECT: Write a parser that could run any Magma program, assuming the functionality it requires is available in the Sage library. Robert Bradshaw, lead developer of the Cython compiler, has expressed to me a strong interest in writing such a parser once he finishes his Ph.D., if funding were available.

II. Sets, Sequences, and Mappings

* I think Sage has everything that Magma has here, and more.

III. Basic Rings and Linear Algebra

* Sage still doesn't implement automatic coercions between finite fields of characteristic p. Recent work of Lenstra, de Smit, etc., on this problem suggests a good theoretical framework in which to design a good algorithm. (The algorithm in Magma is unclear.)

PROJECT: Implement automatic coercions between finite fields.
* Multivariate polynomial rings have an optimized F4 in Magma, but not in Sage.

PROJECT: Implement an optimized F4 over QQ, number fields, and finite fields. (Yes, everybody and their dog has tried and failed, but Allan Steel did it back in 2002, so somebody else can.)

* Fast Groebner basis over Z and Z/nZ. This will be in Sage 4.1, since the Singular team made that a major priority.

IV. Lattices and Quadratic Forms

* Sage has no database of lattices

PROJECT: Create a database of lattices.
* There is no Lattice data type in Sage, though most functionality for lattices in Magma is also available in Sage, e.g., in the quadratic froms and free modules code.

PROJECT: Create a **QuadraticLattice** class in Sage. (Jon Hanke is implementing this now.)

V. Global Arithmetic Fields

* Computing Galois groups of number fields of arbitrary degree

PROJECT: Implement this new Galois groups algorithm. Is it published?
* Class field theory -- Sage doesn't do anything in particular (e.g., compute ray class groups and ray class fields), though PARI does, so this is likely mainly a matter of exposing more functionality from PARI.

PROJECT: Expose Pari's class field theory functionality.

* Algebraic function fields -- Sage can do almost nothing (even robust computation of Riemann-Roch spaces isn't there). I don't think Sage eve has optimized arithmetic in any algebraic function fields. Florian Hess did a Ph.D. (in German) on the algorithms for function fields.

PROJECT: Implement computation of bases of Riemann-Roch spaces.

PROJECT: Implement Hess's algorithm(s),

* Class field theory for algebraic function fields. Since Sage doesn't have algebraic function fields, it also doesn't have class field theory for them.

PROJECT: Implement class field field for function fields.

* Artin representations -- there is a small (?) package in Magma for computing with representations of Gal(Qbar/Q) into GL_n(C) with finite image. Sage doesn't have a nice object like this.

PROJECT: Create a similar class in Sage. Make sure to hook into Dokchitser's L-functions code, plotting code, etc.

VI. Local Arithmetic Fields

* Factorization over extension of p-adic fields isn't in Sage yet.

PROJECT: Implement factorization over general p-adics (David Roe is currently working on this.)

* Sage doesn't do Newton polygons.

PROJECT: Implement Newton Polygons (David Roe has mostly done this, but not included it in Sage yet)

* Sage doesn't do arbitrary p-adic extensions, though it supports some extensions (this will come once the previous two are done)

* Local class field theory (built upon the previous features: this is a ways out).

* Magma's linear algebra over p-adics is better than in Sage.

PROJECT: Design and implement good algorithms for linear algebra over p-adics (subtle and interesting research area). Kiran Kedlaya points out that "there is room for improvement in Magma's handling of p-adics:

```
> F := Qp(3, 10);
> M := Matrix([[F!1, F!2, F!1], [F!1, F!5, F!4], [F!4, F!8, F!1]]);
> CharacteristicPolynomial(M);
S.1^3 - (7 + O(3^10))
*S.1^2 - (3^3 + O(3^10))
*S.1 + 3^2 + O(3^10)
```

This is valid but not best possible.

The best way to handle this seems to be using Newton polygons, which will have to wait for Roed's p-adic polynomials code to come online. After that, there is a nice project waiting here,

which has *important applications to computing zeta functions of varieties using p-adic cohomology* (in which it is crucial to use as little working precision as you can get away with, for efficiency)."

* Galois rings

PROJECT: Implement Galois rings over p-adic fields. Probably not too hard.

* Lazy multivariate power series with possibly fractional exponents

PROJECT: Implement them. We've tried to design these before, but got tangled up in details, so following the Magma interface and definitions closely is probably a good idea.

VII. Modules

* Sage has nothing for modules over general Dedekind domains (except over ZZ): this is an important building block for other algorithms (e.g., arithmetic in quaternion algebras over number fields), so needs to get implemented. I recently wrote code for general modules over ZZ, but it isn't in Sage yet.

PROJECT: Finish modules over ZZ, optimize, especially in the sparse case using Linbox (for

algebraic topologists).

PROJECT: Extend modules over ZZ to modules over a PID. (David Loeffler implemented general HNF last week -- trac 6178, and I implemented code to compute over general PID's given HNF and SNF.)

PROJECT: Implement modules over Dedekind domains (via pseudobasis).

VIII. Finite Groups

* GAP and Magma seem pretty similar for group theory, and Sage exposes much of GAP functionality here.

PROJECT: Rewrite the C kernel of GAP so that Sage can directly manipulate GAP objects at the C level, thus completely avoiding pexpect. Basically this might mean re-implementing the GAP interpreter in C/Cython.

PROJECT: Implement native group theory functionality in Cython.

IX. Finitely-Presented Groups

* Sage doesn't have finitely presented groups.

PROJECT: Implement a finitely-presented group class in Sage. Nathan Dunfield (a group theorist): "This is in GAP, so it's just a matter of wrapping it."

X. Coxeter Groups; Lie Groups

* Dan Bump: "Sage has algorithms for computing with weight lattices, and characters of Lie groups. This means that we can do things like compute the character of a representation, see how it restricts to a smaller subgroup, construct the Weyl group and work with its action on the weight lattice. [...] *Magma has additionally provisions for working in the group itself*, and these are general enough that one can work with groups of Lie type over any field, including a finite field. ...

Nicolas Thiery has a large patch ... quite a bit of stuff for Coxeter groups (including Weyl groups) in the combinat queue.

There is a strong parallel between constructions in combinatorics and constructions in Lie theory. The explanation for this fact has to do with quantum groups, which give a dictionary between representations of Lie groups and combinatorial facts such as the Robinson-Schensted correspondence. In Sage the development is being done by mainly by combinatorialists. So what is likely to emerge will have a lot of functionality but a different emphasis than Magma. *I think it is going to be very good*."

* Lie algebras/Lie groups. Magma has facilities for working directly in the Lie algebra and Lie groups. I don't think this is implemented directly in Sage but there are some facilities for this in Gap which could be better exposed.

PROJECT: Expose all functionality in GAP related to Lie Algebras and Lie Groups.

XII. Algebras

* Generic algebras -- we have them, but they are not optimized, and little functionality is supported.

PROJECT: Create a fast optimized Cython implementation of free algebras and their quotients. Basically, rewrite sage/algebras/free_algebras. Make sure that all functionality in Magma for these is also present in Sage.

* Ideal theory of quaternion algebras over number fields. (This is high on my todo list. We have very fast basic arithmetic, but need ideal theory over Dedekind domains, etc.)

PROJECT: Implement quaternion algebras over over number fields, following the algorithms and strategy John Voight used for Magma.

* Quantum groups, Lie algebras, Universal enveloping algebras -- we don't have this in Sage.

XIII. Representation Theory

* Characters of finite groups; GAP has this, of course, but I don't think Sage nicely wraps it.

PROJECT: Wrap more of GAP's functionality, in particular characters of finite groups.

XIV. Commutative Algebra

* Modules over multivariate polynomial rings. This is a gap in functionality in Sage.

PROJECT: Singular can do this well, so wrap it.

* Differential rings.

PROJECT: Add support for differential rings, using Singular. Brickenstein: "The data structure is same, only the rings are constructed differently. So, what is missing is mainly a concept, how to use generic wrapping of Singular's poly type to get polynomials, vectors, noncommutative polynomials... There is a team of Frank Seelisch, Oleksandr Motsak and Alexander Dreyer working on a similar task in C++, to generate a clean object oriented wrapper of the Singular

functionality. So they might be the experts for these design issues."

* Invariant theory: I think we have nothing in Sage.

PROJECT: Add support for invariant theory. GAP + Singular.

XV. Algebraic Geometry

* General algebraic curves are far more well developed in Magma.

PROJECT: Finish improving doctest coverage of algebraic curves in Sage. Fill in all the generic missing functionality. (Alex Ghitza has been working on this.)

* Resolution of singularities of algebraic curves -- we don't have this (?).

PROJECT: Implement this or wrap functionality from Singular.

* Algebraic surfaces -- Sage has nothing special

PROJECT: Implement same algorithms as from Magma for algebraic surfaces.

XVI. Arithmetic Geometry

* Rational curves and conics -- Sage doesn't have any special code for them.

PROJECT: These are mostly "easy", but with a couple of important algorithms, which I think are already in Sage (as part of eclib and Simon's gp code). So this could just be wrapped.

* Elliptic curves -- Sage doesn't have 3-descent, 4-descent, 8-descent; Heegner points are not as well developed yet. Sage doesn't have its own 2-descent.

PROJECT: Implement 2-descent, 3-descent, 4-descent, and 8-descent in Sage. (Robert Miller just implemented most of 2-descent and the results are amazing -- much faster than mwrank and Magma at initial benchmarks.)

John Cremona: "4-descent in Magma was developed first by my student Tom Womack, though later tidied up quite a lot in Sydney. I have all Tom's code."

PROJECT: Implement a package for computing with and doing research on Heegner and higher Heegner points. Include functionality related to Gross-Zagier L-functions. (Robert Bradshaw is doing this as part of his Ph.D. thesis.)

* Sage doesn't have all pairings for elliptic curves over finite fields, e.g., some subset of Weil, Tate, EtaT, Etaq, Ate, ReducedAte, Ateq is missing.

PROJECT: Fill in missing pairings.

* Elliptic curves over function fields -- we don't have anything nontrivial at all.

PROJECT: Implement package for elliptic curves over function fields and Drinfeld modular curves. (Sal Butt whose Ph.D. was on elliptic curves over function fields with Fernando R-Villegas, will be my postdoc for the next few years.)

John Cremona: "Elliptic curves over function fields: quite a bit of what Magma has came from my student David Roberts, whose thesis in 2007 was about this. I have his code."

* Models of genus one curves -- Sage has nothing useful here, which is very annoying.

PROJECT: Implement all the standard models (as defined by what Magma supports), and transitions between them.

John Cremona: "Other descents on elliptic curves, and "genus one models": mostly written by Tom Fisher who has not (yet!) started to use Sage; but he is implementing algorithms developed jointly by himself, me Stoll, Simon and others, and may well be willing to donate his code to Sage."

* Hyperelliptic curves -- Sage has nothing really for jacobians over number fields or effective Chabauty. Sage also doesn't implement the analytic Jacobians package from Magma.

PROJECT: Make the group law on hyperelliptic Jacobians very fast. (Nick Alexander has done a lot already (trac 6218)).

PROJECT: Implement effective Chabuaty.

PROJECT: Implement 2-descent on Jacobians of Hyperelliptic curves over QQ.

PROJECT: Implement analytic Jacobians. (Nick Alexander has done "half" of this already, and is working on the other half.)

XVII. Modular Arithmetic Geometry (?)

* Arithmetic fuchsian groups and Shimura curves -- John Voight's code; Sage doesn't have this at all.

PROJECT: Implement a package for computing with Shimura curves in Sage.

XVIII. Geometry

* Weight 1 modular forms

PROJECT: Implement standard algorithm(s) for computing weight 1 modular forms. See Tate's algorithm from Springer Lecture Notes 1585 and an algorithm of Kevin Buzzard.

* Hilbert modular forms -- requires quaternion algebras over real fields.

PROJECT: Implement computation of Hilbert modular forms following Lassina Dembele's papers. This has to wait for quaternion algebras over number fields.

XIX. Combinatorics

* Enumerative Combinatorics: Sage essentially has all that Magma has (and much more)

* Partitions, words and young tableaux: Sage is missing the plactic and tableau monoids (nothing hard).

* Symmetric functions: Sage has some rough edges to be smoothed out (general interface, plethysm, inner product), but other than has all that Magma has.

* Graphs/multigraphs/networks: Sage has everything Magma has except the max flow, min cut

algorithm.

POROJECT: Implement "max flow, min cut" (Ford-Fulkerson algorithm) in Sage [I looked on Wikipedia, and the article on the algorithm there just happens to have a 30-line implementation of the algorithm in *Python*!]

* Finite planes and incidence geometry -- I think Sage doesn't have them. (?)

PROJECT: Add support to Sage for finite planes and incidence geometry.

* Incidence structures and designs -- Sage's capabilities are still pretty primitive.

PROJECT: Dan Gordon: "The designs directory under combinat has a basic functionality, and takes a database of Witt designs and Hadamard matrices from GAP, but Magma also has a database of difference sets, and many more operations; equivalence testing, invariants, resolutions, ... Most of this wouldn't be too difficult to replicate, and I've thought about doing some myself, particularly difference sets."

XX. Coding Theory

* Algebraic-Geometric codes -- We're missing many interesting AG codes, LDPC codes,

Quantum codes, and codes over finite rings.

PROJECT: Implement all these codes. Requires more algebraic curve functionality, some of which isn't available in the open source world.

PROJECT: Fully implement weight distribution and minimum distance functions. Robert Miller is working on this. Joyner, a coding theorist, says "once Robert Miller finishes the weight distribution code, I would classify his automorphism group + weight distribution implementations for arbitrary linear codes over finite fileds as herculean."

* Quantum codes -- not in Sage.

XXI.Cryptography

Strangely, Sage has much more crypto functionality than Magma. The word encryption doesn't appear in any nontrivial way in the Magma reference manual.

XXII. Optimization

We have a lot more in Sage...

Concluding Remarks

I good systematic and verifiable approach to this project would be to make the goal to port every single example in the Magma Handbook to Sage. Or, even better, to directly run every single example automatically via the parser (step I above).

... so why use Sage and not just Magma? Besides being free and open source, Sage can do much that Magma can't, e.g., serious numerical linear algebra, graphics, symbolic calculus, and everything available in any Python library written ever (which is a lot!). And Sage is much faster than Magma at much basic arithmetic -- I will talk about this on Friday.