SAGE: Linear Algebra Plan

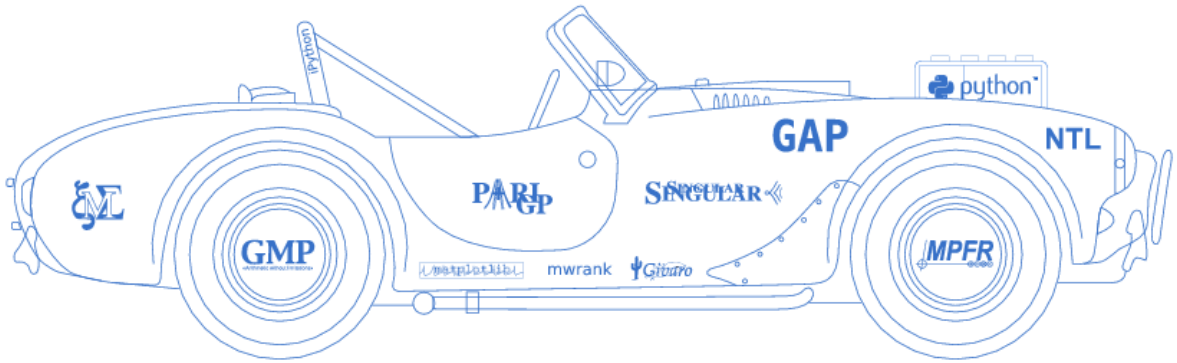
William Stein

February 12, 2007, SAGE week

http://modular.math.washington.edu/sage

SAGE

Building »The Car«



»Every free computer algebra system I've tried has reinvented many times the wheel without being able to build the car.«

The Goal

GOAL: Implement fast optimized linear algebra over exact rings and fields in SAGE.

We have been working on this for what seems like **forever**, and are not done!

```
On Friday 08 April 2005 03:09 am, Kevin Buzzard wrote:
> I had no idea it [SAGE] had got so far. I had
> conjectured that you would still be messing with
> the linear algebra stuff.
```

In particular, at a minimum we need modern fast optimized implementations of algorithms for linear algebra over $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{F}_q$, $\mathbb{Q}_p$, number fields (especially cyclotomic fields), and $k(t)$ for k each of the fields listed before. **Both dense and sparse with identical functionality.**

First Challenge

1. In the world of numerical computation, developing good linear algebra tools has been extremely difficult (because of roundoff error, etc.), but that group of people has succeeded.

2. Unfortunately, for linear algebra over exact rings and fields, the situation is **terrible** in comparison because the main people who have done this (Magma and cryptography researchers) often do not make their tools available to the community (so ask them to!!). But at least they have proved that good results are possible.

3. The **linbox** project has been around for over 5 years now, and is a free C++ library that aims to address this problem:
   `http://www.linalg.org/NEWS-1.1.html`

4. Linbox is now finally getting quite usable, and is in fact **kicks ass** at *certain* things.

High Level Plan for Linear Algebra in SAGE

1. (done) **Initial Python implementation** so what functionality is needed is clear, and a package that *uses* linear algebra in a highly nontrivial way over almost every base ring can be implemented (namely modular symbols).
2. (done) **Rewrite implementation SageX** – compiled language, and restructure implementation to make it possible to systematically create optimized matrix classes for different base rings.
3. **Incorporate linbox** into SAGE (in progress); complain, fix bugs, discover gaps, etc.
4. **Implement advanced optimized algorithms** in the context of SAGE (in progress); fix bugs, etc.

Yardstick

1. The fastest general purpose system in the world for dense linear algebra is MAGMA.
2. Most general purpose math software is *terrible* compared to MAGMA at exact linear algebra. PARI won't save us here:

```
sage: m = random_matrix(ZZ,200)
sage: time k=m*m
CPU times: user 0.05 s, sys: 0.00 s, total: 0.06 s
Wall time: 0.12
sage: g = pari(m)
sage: time h=g*g
CPU times: user 0.71 s, sys: 0.00 s, total: 0.72 s
Wall time: 0.72
sage: time z = g.matker()
CPU times: user 47.64 s, sys: 0.21 s, total: 47.85 s
Wall time: 49.12
sage: n=m.change_ring(QQ)
sage: time w=n.kernel()
CPU times: user 0.14 s, sys: 0.02 s, total: 0.16 s
Wall time: 0.18
```

More precise plan

With linbox having matured enough in the last year to be usable, success is finally within reach! I'm going to work very hard on this during the next two weeks. The plan

1. Create a good set of **benchmarks and challenge problems**. These will come from modular symbols, etc. My project for today.
2. Incorporate **IML (integer matrix library)** into SAGE.
3. Polish, test, and tune new code Robert Bradshaw and I have been writing since this summer in SageX for **fast dense echelon and matrix multiply**. In particular, we must be able to completely turn off linbox with no loss in functionality (since linbox is complicated and this helps with debugging).
4. Incorporate as much of **linbox's** functionality as we can. NOTE: Linbox is tricky to use, buggy in some ways, etc., so this squeezing full value out of linbox is quite hard. But it's well worth it, and everybody benefits.
5. Build **optimized algorithms** out of components available in linbox. For example, for dense echelon I think a wide range of tricks should be used together – there's no one best algorithm.
6. Improve interface to **system solving** over exact base rings.

Linear Algebra in SAGE: Tutorial

1. **Matrices:**
   1. `matrix` **command,** `random_matrix` **command, via a** `MatrixSpace`
   2. **Compute the following associated to** A**: kernel, inverse (if invertible), characteristic polynomial, eigenspaces (and understand the output), the product** AA**, echelon form, Hessenberg form, determinant.**
   3. **Create matrices form given rows or columns of** A**. Make** A **immutable.**
2. **Vector spaces:**
   1. **Choose a field** k**. Create the** 4**-dimensional vector space** $V = k^4$ **over** k**.**
   2. **Create two three-dimensional subspaces** W_1 **and** W_2 **of** V**.**
   3. **Compute the intersection of** W_1 **and** W_2**. Compute their sum.**
   4. **Express an element of** $W_1 \cap W_2$ **in terms of the basis for** W_1 **(using the coordinates method).**
   5. **Do the above 3 steps with** k **replaced by** $\mathbb{Z}$**.**

Tour of Linear Algebra Source Code

1. `docs.py`; **overall layout and design; levels of implementation functionality**
2. **Abstract base class for matrices:** `matrix0.pyx`, `matrix1.pyx`, `matrix2.pyx`, **and** `matrix.pyx`.
3. **Dense and sparse base classses**
4. **Generic dense and generic sparse matrices**
5. **Dense and sparse matrices modulo** n.
6. **Dense matrices over** $\mathbb{Z}$ **and** $\mathbb{Q}$.