Numbers in Sage

http://www.sagemath.org

William Stein[1]

[1]Department of Mathematics
University of Washington, Seattle

Sage Days 8 at Enthought, February 29, 2008

Contents

Prelude

Rings and Fields

Polynomials

Matrices

Outline

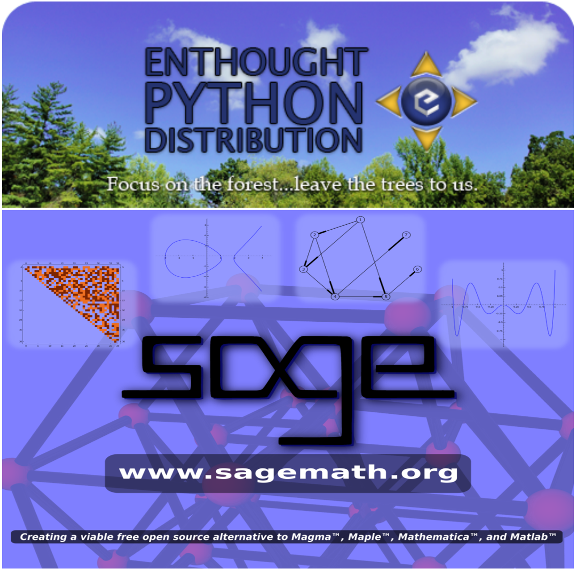
Prelude

Rings and Fields

Polynomials

Matrices

Welcome to Sage Days 8!



Enthought and Sage: Huge Potential for Collaboration

The goal of **Sage** is to create a **viable free open source alternative to Maple, Mathematica, Matlab, and Magma**.

General and Advanced Pure and Applied Mathematics:

> *"Use SAGE for studying a huge range of mathematics, including algebra, calculus, elementary to very advanced number theory, cryptography,* **numerical computation***, commutative algebra, group theory, combinatorics, graph theory, and exact linear algebra."*

This Talk: Sage's Core Arithmetic

- This talk is about **core arithmetic functionality** in Sage that supports the algebraic side of mathematical computation.
- This is the **foundation** of "pure" research mathematics computation, and it is where much Sage development is currently focused.
- It's functionality that up until now **only Magma** did really well (certainly *not* Maple, Mathematica, Matlab, or any single open source program).
- Relevant for numerical computation? Don't know. *Numerical computation is very* relevant to "pure" research mathematics.

Outline

Prelude

Rings and Fields

Polynomials

Matrices

It all starts with...
GMP: Gnu Multiprecision Library

1. GMP is about 150,000 lines of code.
2. Optimized arbitrary precision arithmetic with integers and rationals.
3. GMP currently ships with Mathematica, Maple, Magma, etc. – *nobody has consistently done better than GMP.*
4. Almost all algebraic libraries *depend on GMP*.
5. We might be forced into **forking GMP** because of:
   - their move to LGPL v3,
   - the (un)health of the GMP community (anti-Mac, extorsion for patches, poor release cycle, etc.),
   - impressive progress on fast arithmetic by Sage developers, and
   - others reasons I won't mention here.

   See `http://mpir.org`.

A Note About Benchmarks in this Talk

- By default, benchmarks in this talk are under **OS X 10.5** running 32-bit versions of the relevant software on a **2.6Ghz Core 2 Duo** Macbook Pro laptop.
- I also ran the benchmarks under 64-bit Linux with 64-bit versions of software, and if the timings are drastically different, mention it. (These were run in 64-bit Debian Linux under VMware on the same 2.6Ghz Core 2 Duo, which has VTX and 2GB RAM.)
- I used Sage-2.10.3, Magma V2.14-9, Mathematica 6.0.1, PARI 2.3.3, and Maple 11.

Multiplying Million Digit Integers?: Sage (GMP) is over 50 times Faster than Python

```
sage: n = ZZ.random_element(10^1000000)
sage: m = ZZ.random_element(10^1000000)
sage: time a = n*m
CPU time: 0.13 s,  Wall time: 0.14 s
sage: nn = int(n); mm = int(m)
sage: time a = nn*mm
CPU time: 6.86 s,  Wall time: 6.91 s
sage: time n.gcd(m)
2
CPU time: 4.10 s,  Wall time: 4.12 s
sage: 6.86 / 0.13
52.7692307692308
```

(Note: GMP is over 100 times faster than Python under 64-bit Linux.)

Factorization, Arithmetic Functions: PARI, FLINT, New code

```
sage: time factor(2^137 - 1)
32032215596496435569 * 5439042183600204290159
CPU time: 0.37 s,  Wall time: 0.41 s

sage: time n = number_of_partitions(10^6)
CPU time: 0.03 s,  Wall time: 0.03 s
sage: len(str(n))
1108

sage: time v = prime_range(10^7)
CPU time: 4.66 s,  Wall time: 6.08 s
sage: len(v)
664579
```

MPFR: Multiprecision Floating Point Reals

1. About 55,000 lines of C code (LGPL, by Paul Zimmerman).
2. Optimized arbitrary precision arithmetic with real numbers.
3. Also **very rigorous** and 100% **platform independent** results.
4. MPFR currently ships with Magma – nobody has done better than MPFR...

Multiplying a Hundred Thousand Digits of π: MPFR is over 4000 times faster than Decimal

```
sage: R = RealField(3.32*10^5)
sage: a = R.pi()
sage: len(a.str())
99942
sage: time b = a*a
CPU time: 0.01 s,  Wall time: 0.01 s

sage: import decimal
sage: PI = decimal.Decimal(a.str())
sage: time b = PI*PI
CPU time: 42.27 s,  Wall time: 42.61 s
sage: 42.27 / 0.01
4227.00000000000
```

Quad Precision Reals

1. About 23,000 lines of C/C++ code, BSD license.
2. Quaddouble is part of Sage – provides real numbers with 216 bits of precision.
3. Simple data structure and fast arithmetic and trig functions.
4. A LAPACK built on this would be of interest to numerical computation, and I think is in the works or done.

```
sage: RQDF
Real Quad Double Field
sage: a = RQDF(pi); a
3.14159265358979323846264338327950288419716939
93751058209749445 90
sage: sin(a/2)
1.000000000000000000000000000000000000000000000
0000000000000000000
```

Multiprecision Floating Point **Complex Numbers**

Sage also has multiprecision floating point **complex numbers** (built on MPFR) along with some multiprecision special functions.

```
sage: K.<I> = ComplexField(200)   # 200 bits of precision
sage: a = K(pi) + I; a
3.1415926535897932384626433832795028841971693993751058209749
+1.0000000000000000000000000000000000000000000000000000000000*I
sage: a.gamma_inc(2+5*I)
1.0486976320988281408799148165081638222108234688452799961068
-1.9239010448433085753898248495800053753881658954589692682543*I
sage: a.zeta()
1.0977390684282594480881320973507592738706227822476680574145
-0.12875465315005267463348616853060115143561047216473997227028*I
sage: a.arcsinh()
1.9046276869706578620372233641527072489817035862403811939698
+0.29558503421162990028572406892213528412745110063135493432292*I
```

MPFI: Multiprecision **Interval Arithmetic**

1. About 7,000 lines of C code (GPL'd); Sagified by C. Witty.
2. Builds on MPFR (which in turn builds on GMP).
3. Arithmetic and special functions with real intervals $[a, b]$, where a and b are multiprecision.
4. Comes to the rescue, e.g., Maxima's symbolic ceil and floor functions can be just plain wrong, and very hard to fix, since Maxima does not have interval arithmetic; Sage's are correct:

```
sage: maxima(factorial(50)/exp(1)).ceiling()
11188719610782480421414879249141773426630319
613740326700720324608
sage: ceil(factorial(50)/exp(1))
11188719610782480504630258070757734324011354
208865721592720336801
```

NOTE: Mathematica does this fine; Maple says "can't do it".

MPFI Demo

```
sage: R = RealIntervalField(53); R
Real Interval Field with 53 bits of precision
sage: a = cos(sin(pi^2 + e))^2 - sin(2)
sage: R(a)
[0.090239794310820853 .. 0.090239794310821742]
sage: float(a)
0.090239794310821408
```

Zero or not?

```
sage: b = float(10^(-16)); b
9.9999999999999998e-17
sage: b + 1 - 1
0.0
sage: c = RIF(10^(-16)); c
[9.9999999999999997e-17 .. 1.0000000000000002e-16]
sage: c + 1 - 1
[-0.00000000000000000 .. 2.2204460492503131e-16]
```

Finite Fields: NTL, Givaro, Pari

Now for something **Very Algebraic**.
Integers modulo p; finite fields.

```
sage: k.<a> = GF(3^2); k
Finite Field in a of size 3^2
sage: list(k)
[0, 2*a, a + 1, a + 2, 2, a, 2*a + 2, 2*a + 1, 1]
sage: x = a; y = (2*a+3); x*y
2*a + 2
sage: time for _ in xrange(10^6): c=x*y
CPU time: 0.46 s,  Wall time: 0.46 s
sage: a = int(3); b = int(7); d = int(17)
sage: time for _ in xrange(10^6): c=(a*b)%d
CPU time: 0.33 s,  Wall time: 0.34 s
```

(On Linux the two timings are identical.)

Number Fields: NTL, Pari

1. Number fields are what you get by “adjoining” a root of a polynomial with rational coefficients to $\mathbb{Q}$.
2. A huge deal in number theory.

```
sage: K.<a> = QQ[2^(1/3)]
sage: a
a
sage: a^3
2
sage: (1+a)^10
729*a^2 + 918*a + 1161
```

PARI provides much deep number field related functionality.

p-adic Numbers

1. p-adics are the **number theorist's analogue of the real numbers**, but where the metric is that rational numbers are *close if their difference is divisible by a large power of p*.
2. The Sage p-adics are 100% new code (over 10,000 lines; mostly by David Roe of Harvard).
3. There is a growing field of *p-adic analysis* (Kiran Kedlaya at MIT). Relevant to cryptography...

Playing with p-adics

```
sage: K = pAdicField(5)
sage: a = K(-1); a
4 + 4*5 + 4*5^2 + 4*5^3 + 4*5^4 + 4*5^5 + 4*5^6 + ...
sage: b = a.sqrt(); b
2 + 5 + 2*5^2 + 5^3 + 3*5^4 + 4*5^5 + 2*5^6 + ...
sage: exp(a)
Traceback (most recent call last):
...
ValueError: series doesn't converge
sage: exp(a-4)
1 + 4*5 + 2*5^2 + 5^3 + 3*5^4 + 2*5^6 + 4*5^7 + ...
```

Coercion

1. Problem: make good sense of *a* **plus** *b* (say) without requiring explicit coercions or doing stupid ad hoc things.
2. Sage has a sophisticated **coercion model**, which resulted from months of hard work over several years by many people.
3. Core ideas: *Canonical morphisms*; constructing objects via a sequence of categorical operations.
4. Being actively rolled out right now; is vastly better for arithmetic than Python's builtin naive approach.

```
sage: R.<x> = PolynomialRing(ZZ); R
Univariate Polynomial Ring in x over Integer Ring
sage: f = x + 1/5; f
x + 1/5
sage: parent(f)
Univariate Polynomial Ring in x over Rational Field
sage: parent(2/1 + R(2))
Univariate Polynomial Ring in x over Rational Field
```

Outline

Prelude

Rings and Fields

Polynomials

Matrices

Polynomials

1. In Sage one can define univariate and multivariate polynomials over *any* of the above rings.
2. In some cases arithmetic is blazingly fast; in others it isn't.
3. Factoring multivariate polynomials – a basic problem in computer algebra – is bizarely **embarassingly slow** in many cases in **all open source software**. **Magma is amazingly good.** We are actively working on this.

Univariate Polynomials: NTL, Pari, FLINT

We create a finite field, a polynomial ring over it, and a polynomial ring over that polynomial ring.

```
sage: F.<a> = GF(9)
sage: R.<x> = F[]
sage: S.<y> = R[]; S
Univariate Polynomial Ring in y over Univariate Polynomial
Ring in x over Finite Field in a of size 3^2
sage: (a + x + y)^2
y^2 + (2*x + 2*a)*y + x^2 + 2*a*x + a + 1
```

NOTE: FLINT (part of Sage) does faster arithmetic in $\mathbb{Z}[x]$ than anything else in the world; not "on" in Sage by default yet, but will be soon. This can have a major impact on many other algorithms, including large integer multiplication.

Multivariate Polynomials: Singular

For multivariate polynomials, we use a C-library interface that Martin Albrecht (a Sage developer) wrote to the computer algebra system Singular:

```
sage: R.<x,y,z> = QQ[]; R
Multivariate Polynomial Ring in x, y, z
over Rational Field
sage: time f = (1+x+y+z)^50
CPU time: 0.45 s,  Wall time: 0.45 s
sage: len(str(f))
862380
sage: len(f.monomials())
23426
```

Groebner Basis

1. Groebner basis are the core algorithmic operation behind much of commutative algebra and algebraic geometry.
2. *Like echelon form* but for commutative algebra.
3. Solves ideal membership: *is a polynomial in an ideal?*
4. Finds **all common solutions** to polynomial equations.
5. Magma usually computes GB's faster than everything else. Sage (=Singular) is faster than everything but Magma (?).

```
sage: P.<a,b,c> = PolynomialRing(QQ,3, order='lex')
sage: I = sage.rings.ideal.Katsura(P,3); I
(a + 2*b + 2*c - 1, a^2 - a + 2*b^2 + 2*c^2, 2*a*b + 2*b*c - b)
sage: I.groebner_basis()
[c^4 - 10/21*c^3 + 1/84*c^2 + 1/84*c,
 b + 30*c^3 - 79/7*c^2 + 3/7*c,
 a - 60*c^3 + 158/7*c^2 + 8/7*c - 1]
sage: I.groebner_basis(algorithm="magma:GroebnerBasis")
[a - 60*c^3 + 158/7*c^2 + 8/7*c - 1,
 b + 30*c^3 - 79/7*c^2 + 3/7*c,
 c^4 - 10/21*c^3 + 1/84*c^2 + 1/84*c]
```

Arbitrary Precision Root Finding

1. Sage can find roots of polynomials to arbitrary precision.
2. Sage can quickly **isolate all real roots** of a polynomial over $\mathbb{Q}$ using a new algorithm/implementation of Carl Witty.

```
sage: R.<x> = QQ[]
sage: f = (x^97 + x - 1) * (x^15 - 5) * (x^37 + x^5 + 1)
sage: f.degree()    # NOTE: plotting f is USELESS
149
sage: time f.real_root_intervals()
[((-21/20, -9/10), 1), ((9/10, 39/40), 1), ((21/20, 9/8), 1)]
CPU time: 0.03 s,  Wall time: 0.03 s
sage: time f.roots(RealField(30))
[(-0.95705222, 1), (0.96579942, 1), (1.1132636, 1)]
CPU time: 0.05 s,  Wall time: 0.07 s
sage: time f.roots(RealField(60))
[(-0.95705222094784803, 1), (0.96579942473411068, 1), (1.1132635
CPU time: 4.63 s,  Wall time: 4.87 s
sage: time f.roots(RealField(100))
[(-0.957052220947848025347745796640, 1), (0.9657994247341106786643
CPU time: 13.02 s,  Wall time: 13.48 s
```

Outline

Prelude

Rings and Fields

Polynomials

Matrices

Matrices in Sage

I will focus on **dense matrices with EXACT entries**, though Sage also has sparse matrices, and some numerical matrices.

1. Many algorithms and questions in exact linear algebra are **very different** than in numerical linear algebra.
2. No exact analogue of BLAS and LAPACK (maybe C. Pernet will change that?)
3. **Magma has for years been vastly superior** to everything else in the world. Sage is gaining ground (sometimes winning).
4. The whole point of matrices in Sage is different than numpy ndarrays; this often causes confusion. In Sage, matrices are **algebraic objects** – not data structures.

Exact Linear algebra libraries in Sage

1. **Linbox:** about 70,000 lines of C++; LGPL; C. Pernet is lead developer; very good at some things, but there is a lot of broken code – Sage picks and choses only the things that work and does much external testing.
2. **IML:** a GPL'd C library that solves $Ax = b$ over $\mathbb{Q}$
3. **M4RI:** *excellent* linear algebra over $\mathbb{F}_2$ (field of order 2).
4. **PARI and NTL:** very naive linear algebra algorithms; ignores most progress on exact linear algebra from the last decade.
5. We have written a lot of new code (about 15,000 lines).

A Matrix Example

```
sage: a = matrix(QQ, 2, [1, 2/3, 5/3, 1/2]); a
[  1 2/3]
[5/3 1/2]
sage: parent(a)
Full MatrixSpace of 2 by 2 dense matrices
over Rational Field
sage: a^(-1)
[ -9/11  12/11]
[ 30/11 -18/11]
sage: a.charpoly()
x^2 - 3/2*x - 11/18
```

Examples of Matrices over various rings

```
sage: a = random_matrix(GF(2),3); a
[0 0 1]
[1 0 0]
[1 1 0]
sage: a^10
[1 0 1]
[1 1 0]
[1 1 1]
sage: R.<x,y> = GF(7)[]
sage: a = matrix(R, 2, [y*x, x+3*y^3, x-y, x^3]); a
[      x*y 3*y^3 + x]
[    x - y       x^3]
sage: time b = a^30
CPU time: 0.02 s,  Wall time: 0.02 s
sage: len(str(b[0,0]))
14073
```

The rest of this talk... benchmarks

1. It is *really easy* to implement the standard exact linear algebra algorithms naively in a way that is **insanely slow**.
2. The challenge is coming up with much better **asymptotically fast algorithms** and implementing and optimizing them.
3. The rest of this talk will thus just discuss some **benchmarks**.
4. Conclusions of the benchmarks will be:
   - **Magma** is extremely good when the numbers are small
   - **Sage** is quite good when numbers are large
   - **Mathematica** is terrible compared to Magma and Sage
   - **PARI** isn't very good either
   - **Maple** is also poor at hard exact linear algebra (or I simply don't know how to use Maple!)
   - **Matlab**: doesn't do exact linear algebra?

Benchmark: Matrix Multiplication over $\mathbb{Z}$

Let A be a random 500x500 integer matrix with entries between -9 and 9. Compute $B = A \cdot A$.

1. Magma: 0.17 seconds
2. Sage (our own code – not Linbox): 0.66 seconds
3. Mathematica: 2.23 seconds
4. PARI: 11.97 seconds
5. Maple: 25.3 seconds

Benchmark: Computing the Nullspace over $\mathbb{Z}$

Let A be a random 301x300 integer matrix with entries between -2^{32} and 2^{32}. Compute the (left) **nullspace** of A. (This is equivalent to solving $Ax = b$.)

1. Sage (uses IML's p-adic nullspace): 1.107 seconds
2. Magma: 15.2 seconds.
3. PARI: Gave up after 4 minutes. (101x100 takes 12.8 seconds)
4. Mathematica: Gave up after several minutes (uses a LOT of memory). (101x100 takes 3.55 s and Sage takes 0.12 s)

(1) Under Linux Sage is only 4 times faster than Magma; (2) Clement Pernet: For really large entries Linbox is much faster than IML.

Benchmark: Hermite Normal Form over $\mathbb{Z}$

Let A be a random 300x300 integer matrix with entries between -2^{32} and 2^{32}. Compute the **Hermite normal form** of A.

1. Sage (new code from Sage Days 7): 8.03 seconds
2. Magma: 34.370 seconds (under Linux, Magma takes only twice as long as Sage)

Benchmark: Characteristic Polynomial over $\mathbb{Z}$

Let A be a random 100x100 integer matrix with entries between -2^{32} and 2^{32}. Compute the **characteristic polynomial** of A.

1. Sage (via C. Pernet's code in Linbox): 1.5 seconds
2. Magma: 3.64 seconds
3. Mathematica: 38.62 seconds

NOTE: Under Linux the situation is quite different!

1. Sage (Linux): 2.88 seconds
2. Magma: 0.61 seconds (!)
3. Mathematica: 69.93 seconds (!)

This is only 100×100. In my research I care about much bigger matrices...

Wrap Up

- And that's my talk. There are many mathematical objects and dozens of algorithms that I didn't mention.
- **Moral:**
  - This talk was about how Sage tackles much different problems than either Scipy/Numpy or Maple/Mathematica, and is the *only program besides Magma to take this challenge seriously*.
  - Magma has limited numerical capabilities, so the combination of Sage with Numpy/Scipy/R may have **a broad impact on pure research mathematics**.
- This talk is about "algebraic stuff", but that is *not* all that Sage is about. Sage is about **creating a viable alternative to Magma, Maple, Mathematica, and Matlab,** with the help of a wide range of contributors from all over the mathematical sciences.