Sage for Mathematical and Cryptographic Research

http://www.sagemath.org

William Stein[1] Martin Albrecht[2]

[1]Department of Mathematics
University of Washington, Seattle

[2]Information Security Group
Royal Holloway, University of London

Bristol, 10.November 2007

Outline

Outline

A Quote from Neil Sloane from This Week

Neil Sloane

```
From: N. J. A. Sloane <njas@research.att.com>
Date: 8 Nov 2007 06:28
Subject: Re: dumb question about installing pari-gp with fink

I would like to thank everyone who responded to
my question about installing PARI on an iMAC.

The consensus was that it would be simplest to install sage,
which includes PARI and many other things.

I tried this and it worked!

Thanks!

Neil

(It is such a shock when things actually work!!)
```

A Quote from John Voight's MIT Talk Last Month

John Voight

"Having seemingly eliminated every alternative, *we turn to SAGE.* SAGE includes Pari, so it has number field arithmetic. It uses Python, which is a very friendly modern (object-oriented) programming language. It is free. It incorporates Cython, which easily allows one to write optimized C code for repeated tasks.

Despite being a relatively new system (so some functionality is limited), since it is open source it is easy to contribute yourself. For example, Carl Witty recently wrote a package for fast real root isolation. So even though one must think about issues like how best to coerce between a C int, a Python integer, and a SAGE Integer, there is a very active development community willing to help!

It has the further advantage that there is a package for **distributed computing called DSage**..."

Mission Statement

Mission Statement

Provide an open source, high-quality, and free viable alternative to **Magma, Mathematica, Maple** and **MATLAB** (in that order).

To achieve this do not reinvent the wheel but **reuse** as much **existing building blocks** as possible and make sure the result is **rigorously tested**, **easy to modify** by the end user and **very well documented**.

Also create a **helpful environment** for users to get help (mailinglists, irc-channel, meetings, coding sprints).

What is Sage?

Sage is a mathematics software package developed by a worldwide community of developers.

1. a **distribution** of the best free, open-source mathematics software available (Sage 2.8.12 ships over 50 third-party packages) that is easy to compile or install from binaries.
2. an **interface** to most free and commercial mathematics software packages (e.g. Magma, Mathematica)
3. a **huge new library**, which uniformly covers the widest area of **functionality**, including several new implementations not yet found elsewhere.

Welcome to Sage:

```
| SAGE Version 2.8.12, Release Date: 2007-11-06
| Type notebook() for the GUI, and license() for information.
```

```
sage: 1 + 1
2
```

The Python Programming Language

Python is a powerful modern interpreted programming language.

- "Python is fast enough for our site and allows us to **produce maintainable features in record times**, with a minimum of developers," said Cuong Do, Software Architect, **YouTube.com.**
- "Google has made no secret of the fact they use Python a lot for a number of internal projects. Even knowing that, once **I was an employee, I was amazed at how much Python code there actually is in the Google source code system**.", said Guido van Rossum, **Google**, creator of Python.
- "Python plays a key role in our production pipeline. Without it a project the size of **Star Wars: Episode II** would have been very difficult to pull off. From crowd rendering to batch processing to compositing, **Python binds all things together**," said Tommy Burnette, Senior Technical Director, **Industrial Light & Magic.**

Python & Cython

http://www.python.org and http://www.cython.org

Python

- easy for you to define **your own data types** and methods on it (bitstreams, ciphers, rings, whatever).
- very clean language that results in **easy to read code**.
- easy to learn: e.g., "Dive into Python" http://www.diveintopython.org/
- a **huge number of libraries**: statistics, networking, databases, bioinformatic, physics, video games, 3d graphics, numerical computation (scipy), and serious "pure" mathematics (via Sage)

Cython [See Robert Bradshaw's talk]

- almost Python compiler
- allows to mix C and Python which is crucial for fast execution speed
- makes easy to **use existing C/C++ libraries** from Python.

Outline

Overview I

see http://www.sagemath.org:9001/cando

Commutative Algebra commutative algebra over $\mathbb{F}_{p^n}$ using, basic arithmetic over arbitrary rings, very fast basic arithmetic over $\mathbb{F}_{p^n}$, quotient rings over multivariate polynomial rings, global & local orderings

Linear Algebra fast linear algebra over $\mathbb{Q}$; dense linear algebra over $\mathbb{F}_q$, M4RI (for $\mathbb{F}_2$), and custom code, and sparse linear algebra solver and echelon form over $\mathbb{F}_q$; numerical dense linear algebra; matrix structure visualisation.

Group Theory permutation groups, abelian groups, matrix groups such as classical groups over finite fields; word problem; subgroup enumeration.

Combinatorics many basic functions, many of Sloane's functions are implemented; symmetric functions, Young Tableaux, and combinatorial algebras.

Graph Theory construction, directed graphs, labelled graphs. 2d and 3d plotting of graphs using an optimised implementation of the spring layout algorithm. constructors for all standard families of graphs, graph isomorphism testing, automorphism group computation

Overview II

see http://www.sagemath.org:9001/cando

Number Theory compute Mordell-Weil groups of (many) elliptic curves using both invariants and algebraic 2-descents, a wide range of number theoretic functions, optimised modern quadratic sieve for factoring integers $n = p \cdot q$, optimised implementation of the elliptic curve factorisation method, modular symbols for general weight, character, $\Gamma_1(N)$, and $\Gamma_H(N)$, modular forms for general weight ≥ 2

Elliptic Curves all standard invariants of elliptic curves over $\mathbb{Q}$, division polynomials, etc. , compute the number of points on an elliptic curve modulo p for all primes p less than a million in seconds, optimised implementation of the Schoof-Elkies-Atkin point counting algorithm for counting points modulo p when p is large, complex and p-adic L-functions of elliptic curves. Can compute p-adic heights and regulators for $p < 100000$ in a reasonable amount of time.

p-adic Numbers extensive support for arithmetic with a range of different models of p-adic arithmetic.

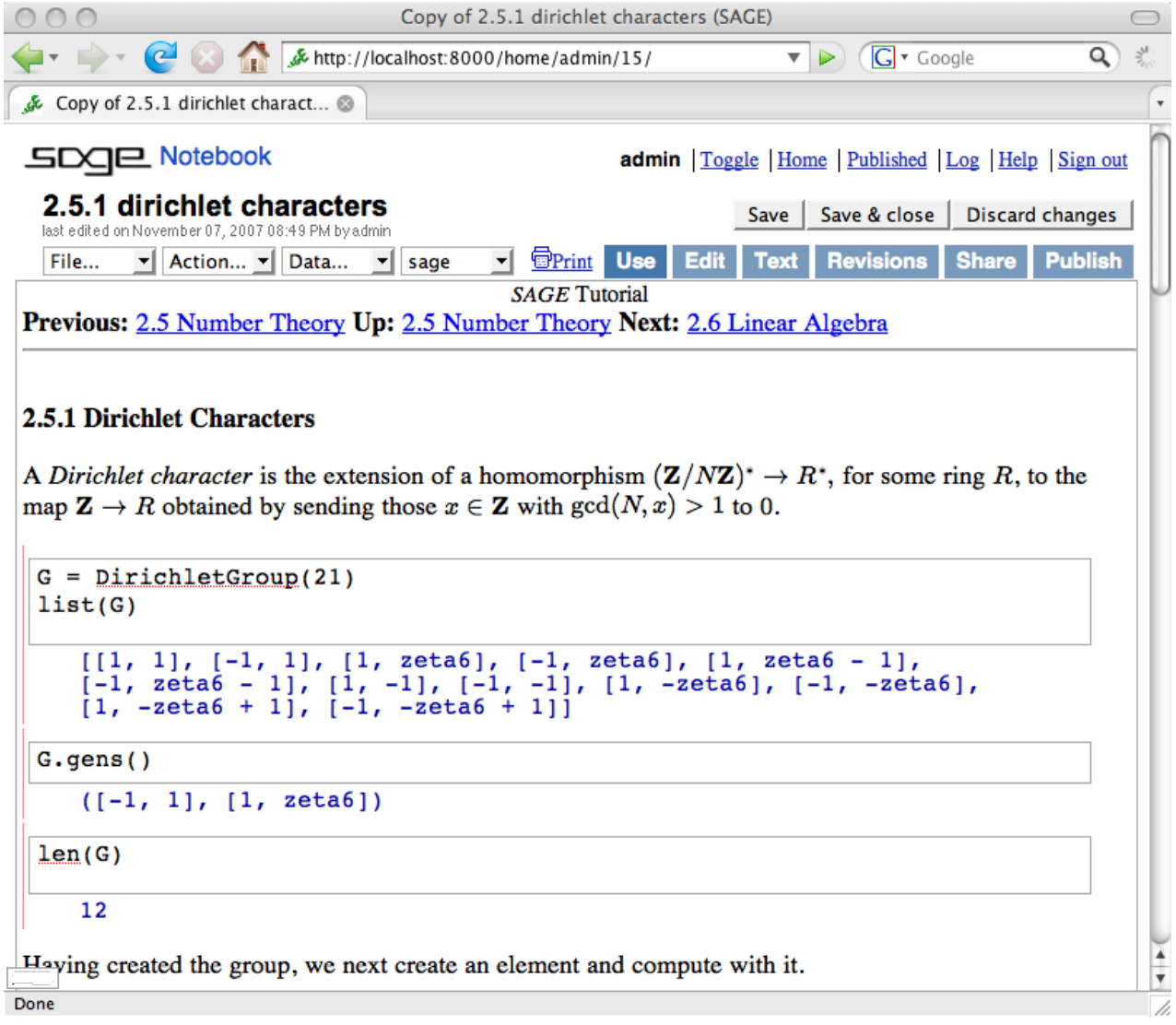
Plotting very complete 2d plotting functionality similar to Mathematica's, limited 3d plotting via an included ray tracer.

Novel Functionality

- free re-implementation of Nauty's graph isomorphism algorithm
- certain models of arithmetic with p-adic numbers & polynomial rings over them
- task farming distributed computing (DSage)
- modular symbols, modular forms, modular abelian varieties
- computing with Dirichlet characters
- Eisenstein series enumeration
- arithmetic on jacobians of curves
- quaternion algebras
- **p-adic L-functions of elliptic curves in a lot of generality, with proven precision**
- **fast computation of p-adic heights on elliptic curves**
- **Coleman integration**
- Duursma zeta functions of linear codes
- **permanents of rectangular matrices over general rings**

Web-based Notebook Interface

public notebooks available at `http://www.sagenb.org`



- graphical user interface
- plotting
- LaTeX typesetting
- remote access
- worksheet sharing
- interface to 3rd party systems like e.g. Magma

Outline

Basic Finite Field Arithmetic

- finite fields $\mathbb{F}_{p^n}$ for arbitrary sizes of p and $n < 2^{32}$.
- finite extension fields of order $< 2^{16}$ implemented using Zech logs (fast!)
- larger $\mathbb{F}_q$ implemented using Pari's (slow) or NTL's (reasonable) polynomial representation
- another option: mp$\mathbb{F}_q$
  `http://hyperelliptic.org/SPEED/slides/Gaudry_Thome_mpFq.pdf`

```
sage: k.<a> = GF(2^8); k
Finite Field in a of size 2^8
sage: P.<x> = GF(3)['x']
sage: p = P.random_element(degree=5); p
2*x^5 + 2*x^2 + 2*x + 1
sage: p/2 # monic
x^5 + x^2 + x + 2
sage: k.<a> = GF(3^5, modulus=p)
sage: k.modulus()
x^5 + x^2 + x + 2
```

Dense Linear Algebra over Finite Fields

- specialised implementation for prime fields
- uses LinBox's BLAS arithmetic (see Clement Pernet's talk) for some operations
- Todo: extension fields (LinBox supports them!)

```
sage: A = random_matrix(GF(127),2000,2000)
sage: B = random_matrix(GF(127),2000,2000)
sage: %time C = A._multiply_linbox(B)
CPU times: user 4.15 s, sys: 0.00 s, total: 4.15 s
Wall time: 4.19 # time depends on installed BLAS

sage: time A.echelonize()
CPU times: user 4.30 s, sys: 0.03 s, total: 4.34 s
Wall time: 4.51
```

Dense Linear Algebra over GF(2)

- M4RI package by Gregory Bard
- uses packed representation and Gray-Codes
- multiplication & echolonisation $O(n^3/log(n))$
- fastest implementation up to $5.000 \times 5.000 - 10.000 \times 10.000$ matrices

```
sage: A = random_matrix(GF(2),20000,20000)
sage: time A.echelonize()
CPU times: user 48.06 s, sys: 0.04 s, total: 48.10 s
Wall time: 48.46
```

- Strassen ($O(n^{2.7})$) support in the works

Sparse Linear Algebra over Finite Fields

- generic sparse matrix classes and a special $\mathbb{F}_p$ class
- custom code for computing the reduced row echelon form

```
sage: A = random_matrix(GF(127),10000,10000,density=3/10000)
sage: time A.echelonize()
CPU times: user 53.13 s, sys: 0.15 s, total: 53.28 s
Wall time: 59.18
sage: A.rank()
9307
```

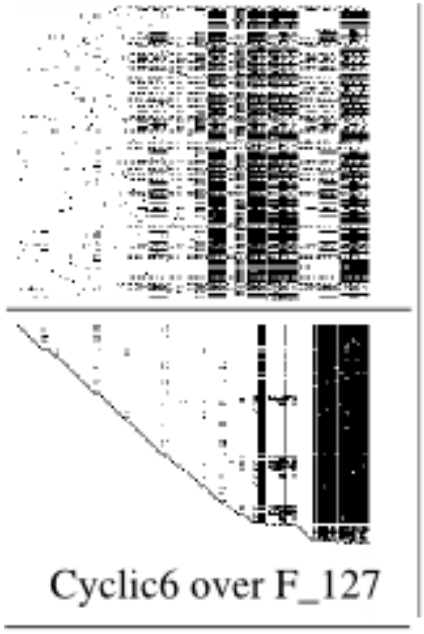
- sparse matrix solver via LinBox (see Clement Pernet's talk):

```
sage: A = random_matrix(GF(127),3000,3000,density=10/3000)
sage: time c = A\b
CPU times: user 35.12 s, sys: 0.59 s, total: 35.71 s
Wall time: 38.75
sage: (A*c) == b
True
```

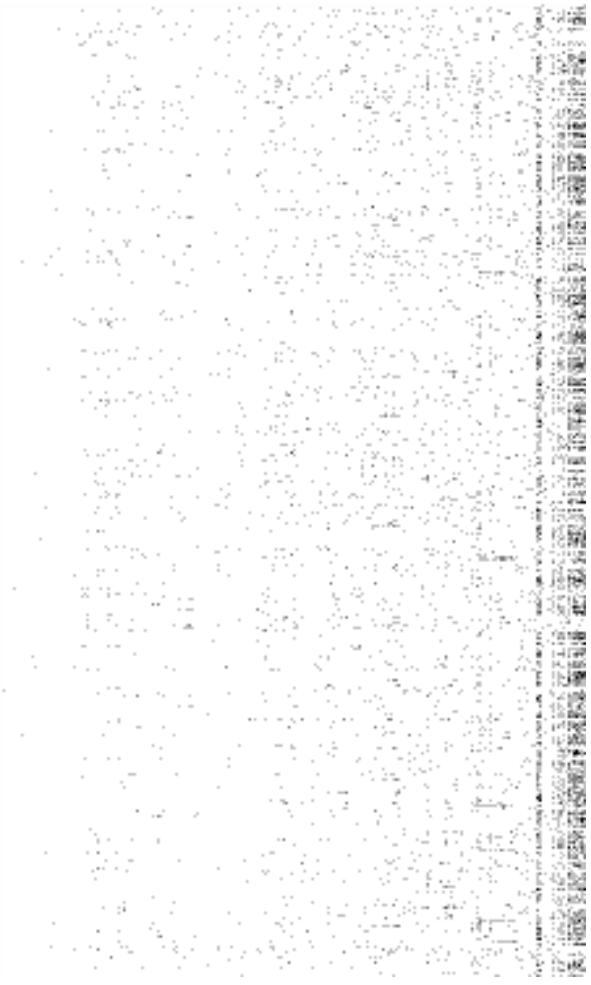
- Ralf Weinmann working on integrating John Cremonas fast g0n sparse matrix code

Matrix Visualisation

A.visualize_structure()



Cyclic6 over F_127



CTC_3,3,3 over F_2

Factoring

factor uses PARI

```
sage: time factor(next_prime(2^40) * next_prime(2^300),verbose=0)
1099511627791 *
203703597633448608626844568840937816105146839366593625063614044935438129\
9763336706183397533
CPU time: 3.77 s,  Wall time: 3.82 s
```

ecm uses GMP-ECM by Paul Zimmermann et al.

```
sage: time ecm.factor(next_prime(2^40) * next_prime(2^300))
[1099511627791, 203703597633448608626844568840937816105146839366593625063614044935438129
CPU time: 0.19 s,  Wall time: 0.62 s
```

qsieve uses Bill Hart's quadratic sieve implementation

```
sage: v,t = qsieve(next_prime(2^90)*next_prime(2^91),time=True)
sage: print v, t[:4]
[1237940039285380274899124357, 247588007857076054979824850 7] 4.00
```

DistributedFactor combines ECM, qsieve and trial division; written by Yi Qiang and Robert Bradshaw.

Lattice Reduction

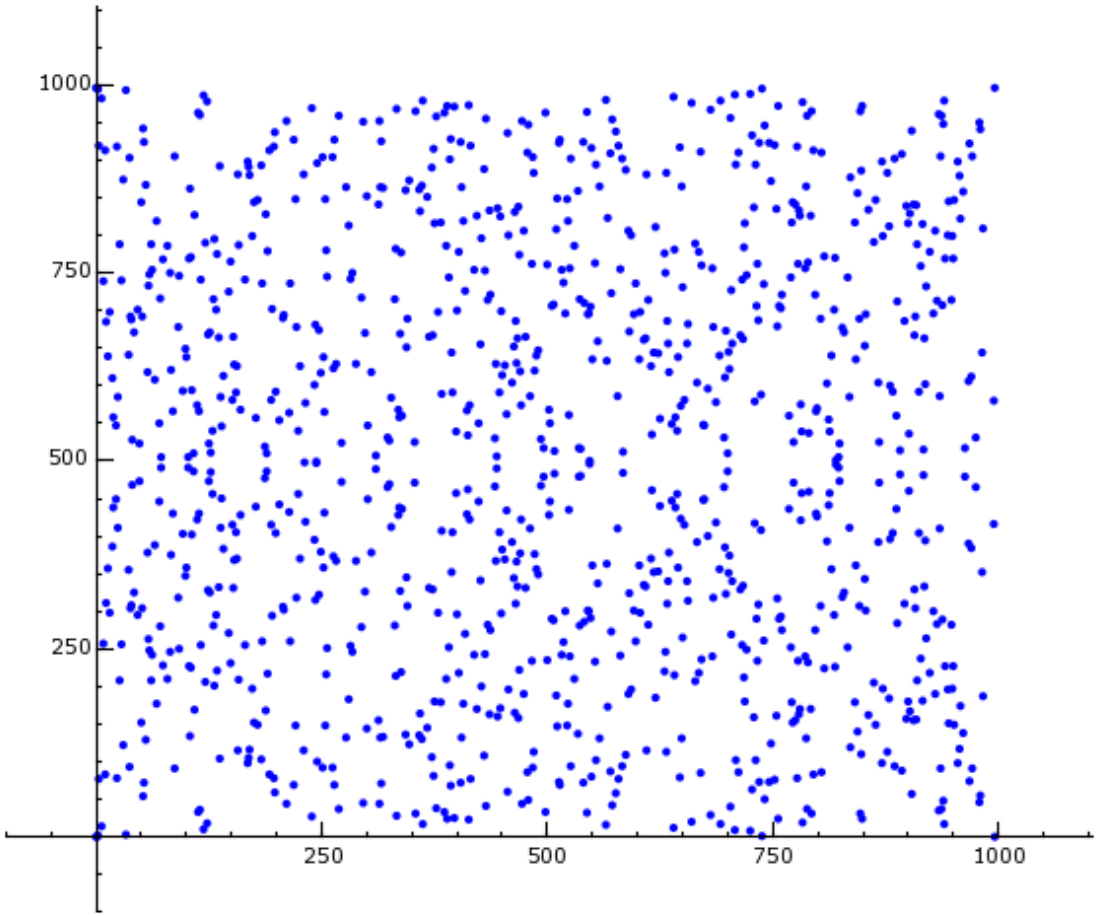
The NTRUEncrypt Public Key Cryptosystem is based on the hard mathematical problem of finding very short vectors in lattices of very high dimension. Generate a ntru-like lattice of dimension (400×400), with the coefficients h_i chosen as random 130 bits integers and parameter $q = 35$:

$$\begin{pmatrix} 1 & 0 & \dots & 0 & h_0 & h_1 & \dots & h_{d-1} \\ 0 & 1 & \dots & 0 & h_1 & h_2 & \dots & h_0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & h_{d-1} & h_0 & \dots & h_{d-1} \\ 0 & 0 & \dots & 0 & q & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & q & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & q \end{pmatrix}$$

```
sage: from sage.libs.fplll.fplll import gen_ntrulike
sage: A = gen_ntrulike(200,130,35)
sage: time B = A.LLL()   # uses fpLLL by Damien Stehle
CPU time: 14.91 s,  Wall time: 15.06
```

Elliptic Curves over $\mathbb{F}_q$ I

```
sage: e = EllipticCurve("37a") # Cremona Label
sage: E = e.change_ring(GF(997))
sage: show(E.plot())
```



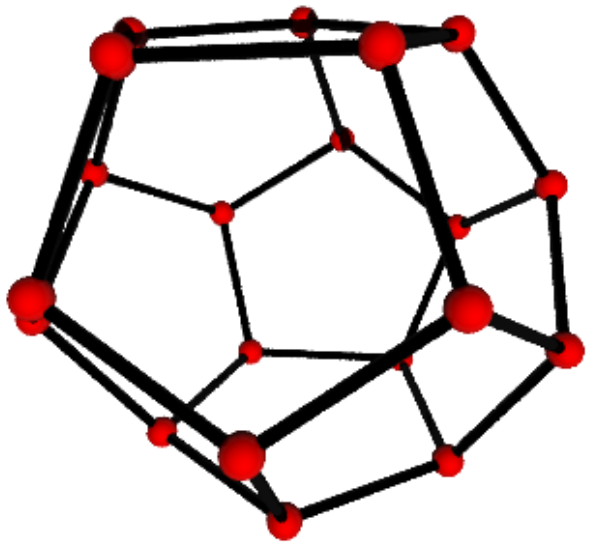
Elliptic Curves over $\mathbb{F}_q$ II

```
sage: k = GF(next_prime(10^7))
sage: E = EllipticCurve(k,
...       [k.random_element(),k.random_element()])
sage: E
Elliptic Curve defined by y^2  = x^3 + 8412640*x + 9921951
over Finite Field of size 10000019
sage: P = E.random_element()
sage: P.order()
5002257
sage: E.cardinality()
10004514
sage: 2*P + P
(251564 : 3681217 : 1)
```

Graph Theory

- builds on NetworkX (Los Alamos's Python graph library)
- graph isomorphism testing – Robert Miller's new implementation
- databases
- 2d and 3d visualization

```
sage: D = graphs.DodecahedralGraph()
sage: D.show3d()
```



```
sage: E = D.copy()
sage: gamma = SymmetricGroup(20).random_element()
sage: E.relabel(gamma)
sage: D.is_isomorphic(E)
True
sage: D.radius()
5
```

Polynomial Arithmetic

- Sage mostly currently uses NTL by default
- FLINT – world's fastest univariate polynomial arithmetic for essentially every bit size and degree (Bill Hart's talk).

```
sage: from sage.libs.flint.fmpz_poly import Fmpz_poly
sage: deg = 31; coeff=64
sage: f=Fmpz_poly([ZZ.random_element(2^coeff) for _ in [1..deg+1]])
sage: g=Fmpz_poly([ZZ.random_element(2^coeff) for _ in [1..deg+1]])
sage: time for _ in xrange(10^5): w = f*g
CPU time: 1.55 s,  Wall time: 1.67 s
```

- PARI takes 9.85 seconds to do the exact same computation.
- Sage wrapping NTL takes 9.24 seconds
- Magma takes 4.68 seconds

Commutative Algebra

- Very fast basic arithmetic for multivariate polynomials over $\mathbb{F}_q$.
- Fast Gröbner basis computation and highlevel ideal operations
- Specialised very fast Gröbner basis computation in

$$\mathbb{F}_2[x_1, \ldots, x_n]/\langle x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle$$

very soon (see Michael Brickenstein's talk)

```
sage: P.<x,y,z> = PolynomialRing(GF(32003),3)
sage: p = (x + y + z + 1)^20 # the Fateman fastmult benchmark
sage: q = p + 1
sage: t = cputime()
sage: r = p*q
sage: cputime(t)
0.13
```

Magma takes 0.360 seconds to do the same calculation

Algebraic Attacks

Sage provides equation systems for algebraic cryptanalysis:

```
sage: sr = mq.SR(2,1,1,4, gf2=True)
sage: F,s = sr.polynomial_system()
sage: s
{k003: 1, k002: 1, k001: 1, k000: 1}
sage: gb = F.groebner_basis()
sage: V = Ideal(gb).variety(); V[0]
...
 k001: 1, k000: 1, k003: 1, k002: 1,
...
sage: sr = mq.SR(10,4,4,8, star=True, aes_mode=True)
sage: F,s = sr.polynomial_system(); F
Polynomial System with 8576 Polynomials in 4288 Variables
sage: F.groebner_basis() # if this terminates => AES broken :-)
```

Outline

How Did Sage Come About?

Sage is:

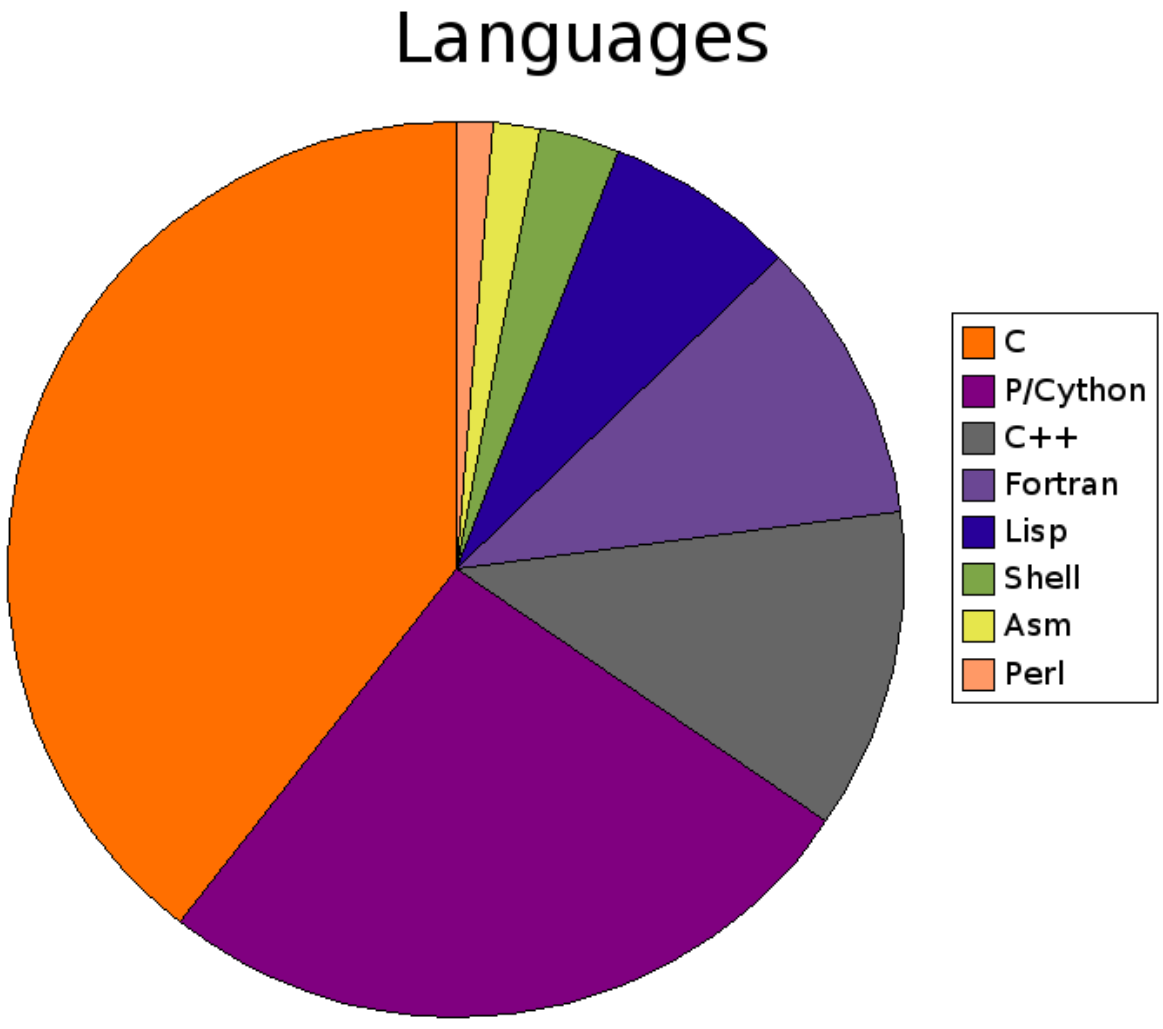
1. A huge amount of *extremely hard mostly volunteer work* and
2. *Refusal to acknowledge reality*, i.e., that Sage is impossible.

Seriously, how did Sage really come about?

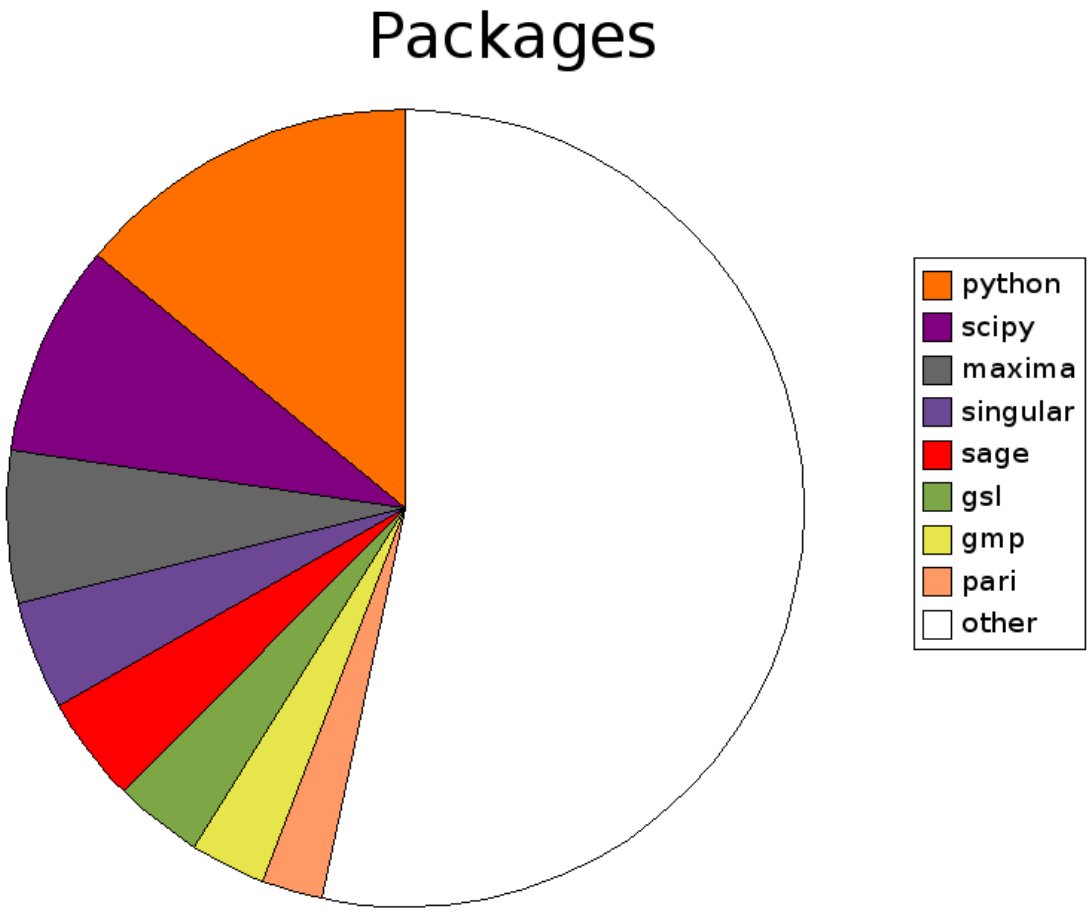
1997–1999	(**Berkeley**) **HECKE** – C++ (modular forms)
1999–2005	(**Berkeley**, **Harvard**) I wrote over 25,000 lines of **Magma** code. **Magma is incredibly powerful!**
	But the languages of Magma, Mathematica, and Maple are **old-fashioned and painful** compared to Python.
	I need to be able to **see inside and change anything** in my software in order to be the best in the world at my research.
	Magma is frustrating and is a **bad longterm investment**.
Feb 2005	I released **SAGE-0.1** — a Python math library.
Feb 2006	**UCSD SAGE Days 1** – SAGE 1.0.
October 2006	**U Washington SAGE Days 2** workshop.
March 2007	**UCLA SAGE Days 3** workshop.
June 2007	**U Washington SAGE Days 4** workshop.
October 2007	**Clay Math Institute SAGE Days 5** workshop.
Now	**U. Bristol and the Heilbronn Institute SAGE Days 6**
Now	**SAGE-2.8.12**; over **100 contributors to SAGE**.

How Do We Do That? I

Arithmetic	**GMP, MPFR, Givaro**
Commutative Algebra	**SINGULAR** (libSINGULAR)
Linear Algebra	**LinBox, M4RI, IML, fpLLL**
Cryptosystems	**GnuTLS, PyCrypto**
Integer Factorization	**FlintQS, ECM**
Group Theory	**GAP**
Combinatorics	**Symmetrica**
Graph Theory	**NetworkX**
Number Theory	**PARI, NTL, Flint, mwrank**
Numerical Computation	**GSL, Numpy, Scipy**
Calculus, Symbolic Comp.	**Maxima, Sympy**
Lattice Polytopes	**PALP**
User Interface	Sage Notebook, **jsmath, Moin wiki, IPython**
Graphics	**Matplotlib, Tachyon, libgd, Java3d**
Networking	**Twisted**
Databases	**ZODB, SQLite**, Python Pickles
Programming Language	**Python, Cython** (compiled)



Overall VERY roughly 4.5 million lines of source code and estimated several hundred person-years.


Packages
python
scipy
maxima
singular
sage
gsl
gmp
pari
other

SAGE: Lots of new code

Unique lines of code and docstrings:

```
$ cat *.py */*.py ... */*/*/*.pxd |sort |uniq |wc -l
189082
```

Unique input lines of autotested examples:

```
$ cat *.py */*.py ... */*/*/*.pxd | grep "sage:"
              | sort |uniq |wc -l
26711
```

Doctesting coverage:

```
$ sage -coverage .
...
Overall weighted coverage score:  34.3%
Total number of functions:  17424
```

Upcoming Workshops

- Jan 5–9, 2008: Sage Days $6\frac{1}{4}$, AMS meeting in San Diego (booth, sprints)
- Feb 5–9, 2008: Sage Days 7, IPAM (confirmed and funded!)
- Feb 29–March 4, 2008: Sage Days 8, UT Austin
- June 2008: Sage Days 9 in Seattle (tentative)
- August 2008: Sage Days 10 in Vancouver at SFU (tentative)

The Sage Release Process

1. All enhancement proposals, bug reports and tasks are available on `http://trac.sagemath.org`.
2. All discussions happen in the open on public mailing lists and on a public chat channel.
3. One release per week on average: release often, release early.
4. Changes happening to the main repository can be tracked in real time online
5. Code is refereed by a board of editors: **Journal of Sage**.
6. If code is rejected by the release maintainer every developer can appeal to this board of editors and they vote on the inclusion of the patch.

How We Talk

sage-devel	development discussions, 213 subscribers, ca. 800 messages per month
sage-support	support requests, 187 subscribers, ca. 200 messages per month
sage-forum	general discussions, 148 subscribers, low traffic
sage-newbie	basic questions about e.g. programming, 33 subscribers, low traffic
#sage-devel	irc.freenode.net irc channel, very busy during bug squashes, usually at least two Sage developers around

Outline

Shortcomings of Sage

1. There are currently about a thousand users of Sage; our goal is to have ten thousand serious users by 2009.
2. Sage is not robust enough.
3. Sage is sometimes much slower than Magma, Mathematica, etc. (and sometimes faster, to be fair).
4. Sage is new – there are too many bugs.

However, if something is wrong you can fix it, and the Sage mailing lists and irc channel are extremely active and helpful (over 1000 messages a month in all lists!).

Advantages of Sage

1. Sage is the only serious general purpose mathematics software that uses a mainstream programing language (Python).
2. Sage is the only program that allows you to use Maple, Mathematica, Magma, etc., all together.
3. Sage has more functionality out of the box than any other open source mathematics software.
4. Sage has a huge, active, and well rounded developer community: [sage-devel] mailing list has over 200 subscribers, working very hard on everything from highly optimised arithmetic, to high school education, to computing modular forms.
5. Sage development is done in the open. You can read about why all decision are made, have input into decisions, see a list of every change anybody has made, etc. This is different than the situation with Magma and Mathematica. Users love this.

Questions?

Thank You!