Not Reinventing the Wheel: a Sage Introduction

http://www.sagemath.org

Martin Albrecht (M.R.Albrecht@rhul.ac.uk)

Royal Holloway, University of London

Egham, 1.November 2007

Outline

Outline

Mission Statement (my words)

Provide a free, open-source (GPL-compatible), viable alternative to **Magma, Mathematica, Maple** and **MATLAB** (probably in that order).

To achieve this do not reinvent the wheel but use 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).

What is Sage?

Sage was started by **William Stein** in 2004 and is now developed by a worldwide community of developers (ca. 30 people submit patches each month, ca. 200 people are subscribed to [sage-devel])

1. a distribution of the best free, open-source mathematics software available (Sage 2.8.9 ships roughly 70 third-party packages) that is easy to compile and/or install on Linux, OS X, Solaris (soon) and Windows (via virtualisation)
2. a unified interface to many free and commercial mathematics software packages (e.g. Magma, Mathematica)
3. the mathematics package which covers the widest area of functionality ever (see: 1.) including new implementations not yet found in the open-source world.

Outline

Systems I

Arithmetic	**GMP, MPFR, Givaro**
Commutative Algebra	**SINGULAR** (libSINGULAR)
Linear Algebra	**LinBox, M4RI, IML, fpLLL**
Highlevel Cryptography	**GnuTLS, PyCrypto**
Factoring	**FlintQS, ECM**
Group Theory and Combinatorics	**GAP, Symmetrica**
Graph Theory	**NetworkX**
Number Theory	**PARI, NTL, Flint**
Numerical Computation	**GSL, Numpy, Scipy**
Calculus, Symbolic Comp.	**Maxima, Sympy**
Specialised Math	many C/C++ programs...
Interface	Notebook, **jsmath, Moin wiki, IPython**
Plotting	**Matplotlib, Tachyon, libgd, Java3d**
Networking	**Twisted**
Database	**ZODB, SQLite**, Python Pickles
Programming Language	**Python, Cython** (compiled)

Systems II

In total that makes roughly 4.5 Million physical lines of code (number generated using David A. Wheeler's **SLOCCount**)

ansic	1907386 (39.01%)
p/cython	1258260 (25.73%)
cpp	553701 (11.32%)
fortran	492184 (10.07%)
lisp	340210 (6.96%)
sh	138356 (2.83%)
asm	79260 (1.62%)
perl	60285 (1.23%)

with a development effort estimation of 1,262.90 Person-Years (take this figure with a spoon of salt).

Interactive Math Packages

- All programs/libraries installed by

```
m: cd $SAGE_ROOT
m: make #... drink lots and lots of tea or coffee
```

- You can use every provided program on its own:

```
m: sage -singular
                     SINGULAR                              /   Development
 A Computer Algebra System for Polynomial Computations    /    version 3-0-3
                                                        0<
     by: G.-M. Greuel, G. Pfister, H. Schoenemann         \    May 2007
FB Mathematik der Universitaet, D-67653 Kaiserslautern     \
> 1+1
. ;
2
> ring r = 0,(a,b),dp;
> r;
//   characteristic : 0
//   number of vars : 2
//        block   1 : ordering dp
//                  : names    a b
//        block   2 : ordering C
> quit;
Auf Wiedersehen.
```

Libraries

also all libraries are installed with development headers etc.

```
m: ls $SAGE_ROOT/local/include/NTL/GF2*
/home/malb/SAGE/local/include/NTL/GF2E.h
/home/malb/SAGE/local/include/NTL/GF2EXFactoring.h
/home/malb/SAGE/local/include/NTL/GF2EX.h
/home/malb/SAGE/local/include/NTL/GF2.h
...
```

test.cpp:

```
#include <NTL/GF2.h>

using namespace NTL;
using namespace std;

int main(int argc, char **argv) {
  GF2 n;
  conv(n,1);
  cout << n << endl;
  exit(0);
}
```

```
m$ source $SAGE_ROOT/local/bin/sage-env
m$ g++ -I$SAGE_ROOT/local/include -L$SAGE_ROOT/local/lib -lntl -lgmp test.cpp
m$ ./a.out
1
```

Outline

Why Use Many Math Packages?

- coverage varies

GAP	group theory
Singular	(non-)commutative algebra
Pari	number theory
Maxima	calculus
...	

- quality/speed of implementations varies (e.g. Pari/NTL/Magma)
- verify results with independent implementations: even Magma has bugs

How Many Languages Can You Handle?

GP/Pari

```
? a=1; for(X=1,100,if(isprime(X),a+=1,a+=2)); a
%1 = 176
```

SINGULAR

```
> int a = 1; int i = 1;
> for(i=1; i<=100; i=i+1) { if(prime(i) == i){  a=a+1; } else {a=a+2;} };
> a;
176
```

Magma

```
> a:=1;
> for i in [1..100] do if IsPrime(i) then a:=a+1; else a:=a+2; end if; end for;
> a;
176
```

GAP

```
gap> a := 1;
gap> for i in [1..100] do if IsPrime(i) then a:=a+1; else a:=a+2; fi;  od; a;
176
```

Maxima

```
(%i1) a: 1;
(%o1)                                   1
(%i2) for i:1 thru 100 do if primep(i) then a: a + 1 else a: a + 2;
(%o2)                                 done
(%i3) a;
(%o3)                                  176
```

Languages of Mathematics Packages

- every package has its own
- languages often a by-product and ad-hoc-ish, developer want to do math not design a programming language
- thus interpreter sometimes really bad and slow
- languages often lack fundamental building blocks: sets, hashtables, object orientation, memory management, inheritance, modularisation
- lack of debugging tools, editor support, references and general purpose libraries (I/O, networking, serialisation)
- . . . it's a pain!

So Use a General Purpose Language: Python

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

- 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)
- easy to **use existing C/C++ libraries** from Python.
- Cython – an almost Python compiler.

Stupid Example Revisited

Instead of learning *n* special purpose languages learn 1 general purpose language to control all systems.

```
sage: a= gap(1) # or gp(1), magma(1), singular(1)
sage: for i in range(100):
...       if gap(i+1).IsPrime():
...         a+=1
...       else:
...         a+=2
...
sage: a
176
```

Note

This code won't break any speed records, because of the way these interfaces work

How the Interfaces Work

Use buffered psuedo-tty, files, and Python objects that wrap native objects. This makes it possible to wrap **all** math software that has a command line interface using similar code.

```
sage: a = gap('1')
```

This fires up one copy of GAP and sends the line `'$sage1 = 1'` to it

```
sage: !ps ax |grep gap
 6995 pts/4    Ss+    0:00 .../local/lib/gap-4.4.10/bin/.../gap -m 24m -l ...
sage: type(a)
<class 'sage.interfaces.gap.GapElement'>
sage: a, a.name()
(1, '$sage1')
sage: a.IsPrime()
false
```

William Stein on [sage-support]:

"Using pseudo-tty's is slow and should be avoided when other options are available. It allows us to very quickly get lots of functionality, whilst not reinventing the wheel. But it's not a panacea."

A More Realistic Example

```
sage: P.<x,y> = PolynomialRing(GF(2))
sage: I = Ideal([x*y + x + 1, x*y + y])
sage: I.groebner_basis? # output reformated slightly for this slide

 Return a Groebner basis of this ideal.

INPUT:
  algorithm -- determines the algorithm to use, available are:
               * None - autoselect (default)
               * 'singular:groebner' - Singulars groebner command
               * 'singular:std' - Singulars std command
               * 'singular:stdhilb' - Singulars stdhib command
               * 'singular:stdfglm' - Singulars stdfglm command
               * 'singular:slimgb' - Singulars slimgb command
               * 'libsingular:std' - libSINGULARs std command
               * 'libsingular:slimgb' - libSINGULARs slimgb command
               * 'toy:buchberger' - SAGEs toy/educational buchberger w.o. strategy
               * 'toy:buchberger2' - SAGEs toy/educational buchberger with strategy
               * 'macaulay2:gb' (if available) - Macaulay2s gb command
               * 'magma:GroebnerBasis' (if available) - MAGMAs Groebnerbasis command

EXAMPLES:
    Consider Katsura-3 over QQ with term ordering 'degrevlex'
    ...
```

Ways to Work with Singular from Sage

- You can construct Singular elements directly:

```
sage: r = singular.ring(0,'(a,b)','dp')
sage: i = singular('2*a^3+3*a*b+1,b^2 + a^2','ideal')
sage: i.std().vdim() # note we never wrote a method 'std' or 'vdim'
6
```

- Some Sage objects use Singular behind the scenes:

```
sage: P.<a,b> = PolynomialRing(QQ,order='degrevlex')
sage: I = Ideal([2*a^3+3*a*b+1, b^2 + a^2]) # calls Singular
sage: gb = I.groebner_basis()
sage: Ideal(gb).vector_space_dimension() # calls Singular
6
```

- low-level arithmetic implemented via libSingular:

```
sage: P.<a,b> = PolynomialRing(GF(127),order='degrevlex')
sage: an, bn = a._magma_().name(),b._magma_().name()
sage: t = magma.cputime()
sage: s = magma.eval("for i in [1..10^6] do z:= %s*%s; end for"%(an,bn))
sage: magma.cputime(t)
0.73999999999999999
sage: time for i in xrange(10^6): z = a*b
CPU times: user 0.60 s, sys: 0.05 s, total: 0.65 s
```

Interfaces to Special Purpose Programs

fpLLL by Damien Stehle for floating point LLL-reduction:

```
m: time ./generate u 200 1000  | ./fplll -r 200 -c 200 > /dev/null
real    0m4.336s
```

you can also use if from Sage:

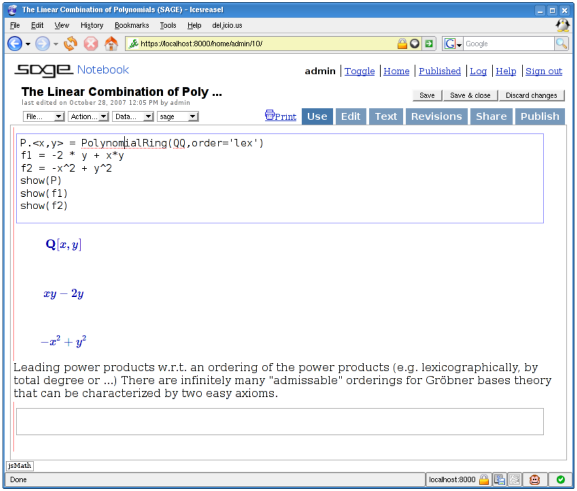
```
sage: from sage.libs.fplll.fplll import gen_uniform
sage: A = gen_uniform(200,200,1000)
sage: time B.LLL()
CPU time: 4.11 s,  Wall time: 4.16 s
sage: AM = A._magma_()
sage: t = magma.cputime(); BM = AM.LLL(); magma.cputime(t)
10.48
```

mwrank by John Cremona for elliptic curves over $\mathbb{Q}$

```
sage: E = EllipticCurve([1, -1, 1, -29372, -1932937])
sage: E.conductor(algorithm="mwrank")
3006
sage: Em = E.mwrank_curve()
sage: Em.conductor()
3006
```

Web-based Notebook Interface

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


The Linear Combination of Polynomials (SAGE) - Iceweasel
File Edit View History Bookmarks Tools Help del.icio.us
https://localhost:8000/home/admin/10/
Notebook
admin | Toggle | Home | Published | Log | Help | Sign out
Save Save & close Discard changes
File... Action... Data... sage
Print Use Edit Text Revisions Share Publish
jsMath
Done
localhost:8000

The Linear Combination of Poly ...

last edited on October 28, 2007 12:05 PM by admin

```
P.<x,y> = PolynomialRing(QQ,order='lex')
f1 = -2 * y + x*y
f2 = -x^2 + y^2
show(P)
show(f1)
show(f2)
```

$\mathbf{Q}[x, y]$

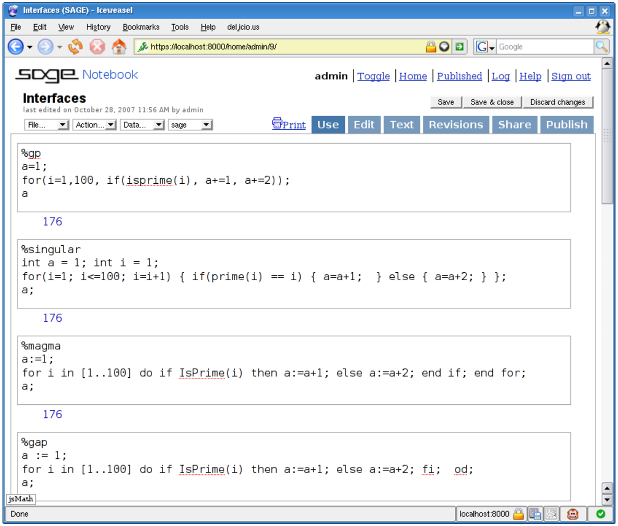
$xy - 2y$

$-x^2 + y^2$

Leading power products w.r.t. an ordering of the power products (e.g. lexicographically, by total degree or ...) There are infinitely many "admissable" orderings for Gröbner bases theory that can be characterized by two easy axioms.

- graphical user interface
- 2d plotting
- LaTeX typesetting
- remote access
- worksheet sharing
- worksheet up-/download
- security (?)

GUI to Many Mathematics Packages


Interfaces (SAGE) - Iceweasel
File Edit View History Bookmarks Tools Help del.icio.us
https://localhost:8000/home/admin/9/
Google
Notebook
admin | Toggle | Home | Published | Log | Help | Sign out
Save
Save & close
Discard changes
Print
Use
Edit
Text
Revisions
Share
Publish
File...
Action...
Data...
sage
jsMath
Done
localhost:8000

Interfaces
last edited on October 28, 2007 11:56 AM by admin

```
%gp
a=1;
for(i=1,100, if(isprime(i), a+=1, a+=2));
a
```

```
    176
```

```
%singular
int a = 1; int i = 1;
for(i=1; i<=100; i=i+1) { if(prime(i) == i) { a=a+1;  } else { a=a+2; } };
a;
```

```
    176
```

```
%magma
a:=1;
for i in [1..100] do if IsPrime(i) then a:=a+1; else a:=a+2; end if; end for;
a;
```

```
    176
```

```
%gap
a := 1;
for i in [1..100] do if IsPrime(i) then a:=a+1; else a:=a+2; fi;  od;
a;
```

Outline

Example: Number of Partitions

```
sage: list(partitions(5))
[(1, 1, 1, 1, 1), (1, 1, 1, 2), (1, 2, 2), (1, 1, 3),
  (2, 3), (1, 4), (5,)]
sage: number_of_partitions(5)
7
```

1. The beginning of the Mathematica tour has an assertion that: "Mathematica computes the number of partitions of 1 billion in a few seconds – a frontier number theory calculation".
2. Sage (and Magma!) would take years to do that, so William Stein posted on [sage-devel]; 72 posts among 15 people followed.
3. Now – thanks to Jon Bobber (U Mich grad student) Sage is faster at this than any other program in the world on some architectures:

```
sage: time len(str(number_of_partitions(10^9)))
CPU times: user 33.32 s, sys: 0.08 s, total: 33.40 s
35219
Mathematica 5.2.1 takes 61.34 seconds.
In[1] := Timing[N[Log[PartitionsP[10^9]]]]
Out[1]= {61.3417 Second, 81092.9}
```

More Examples

list is result of a thread on [sage-devel]

- 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**

Outline

Things Sage Can Do I

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

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

Linear Algebra dense linear algebra over $\mathbb{F}_q$ using LinBox, M4RI (for $\mathbb{F}_2$), and custom code, and sparse linear algebra over $\mathbb{F}_q$ via custom code and LinBox (e.g. sparse solver). Numerical dense linear algebra via Numpy and GSL. Matrix structure visualisation.

Group Theory permutations groups, abelian groups, matrix groups (in particular, classical groups over finite fields)

Combinatorics many basic functions, many of Sloane's functions are implemented.

Things Sage Can Do II

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

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

Number Theory compute Mordell-Weil groups of (many) elliptic curves using both invariants and algebraic 2-descents, a wide range of number theoretic functions, optimized modern quadratic sieve for factoring integers $n = p \cdot q$, optimized implementation of the elliptic curve factorization method, modular symbols for general weight, character, Gamma1, and GammaH, modular forms for general weight ≥ 2

Things Sage Can Do III

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

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, optimized 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.

Cryptography I

Sage ships with **PyCrypto**:

```
sage: from Crypto.Cipher import AES
sage: crypt = AES.new('abcdefghijklmnop', AES.MODE_ECB)
sage: txt = 'ea523a664dabaa4476d31226a1e3bab0'
sage: c = crypt.encrypt(txt)
sage: c
'w\x81\xe3\xdd\x066\x9eY\xc7\xce~O\x9e\xfb\xef\xfa\xb5\x8a\xac/
\x7f\xca\x9fl{\xe5\xfd6\x80\xe3\x81%\xb9
```

Sage also has some cryptography educational code:

```
sage: S = AlphabeticStrings()
sage: E = VigenereCryptosystem(S,14); E
Vigenere cryptosystem on Free alphabetic string monoid on A-Z \
of period 14
sage: K = S('ABCDEFGHIJKLMN'); K
ABCDEFGHIJKLMN
sage: e = E(K); e
ABCDEFGHIJKLMN
sage: e(S("THECATINTHEHAT"))
TIGFEYOUBQOSMG
```

Cryptography II

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 :-)
```

Shortcomings of Sage

1. There are currently probably less than a thousand users of Sage (there are millions of Python users).
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.

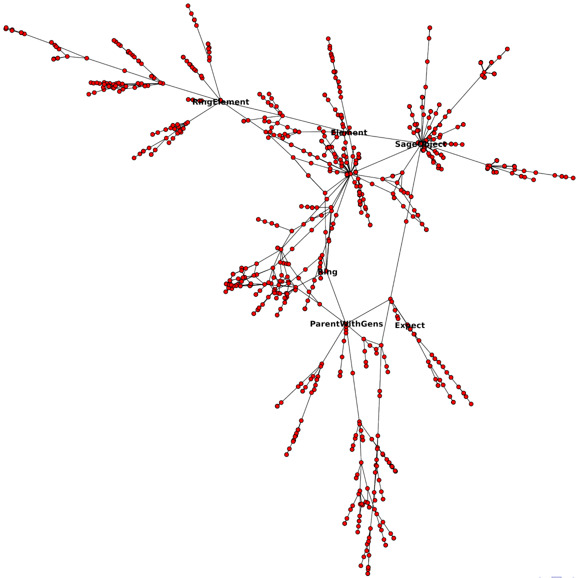
However, if something is wrong you can fix it, and the Sage mailing lists and irc channel are extremely active and helpful (over 1000 messages a month!).

Advantages of Sage
The Final Advertisment Slide

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 optimized 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 totally different than the situation with Magma and Mathematica.

Live Demo

Questions?



Thank You!