Information Security Group

Singular, PolyBoRi and Sage

Martin R. Albrecht

Information Security Group,
Royal Holloway, University of London

July 22, 2009

Table of Contents

Table of Contents

SINGULAR-centric History

Information Security Group

Early 2005	William Stein starts SAGE
22.10.2005	SAGE 0.8.0 includes SINGULAR 3-0-0
14.01.2006	I submit my first patch for the SINGULAR interface :)
13.10.2006	OMALLOC is GPL'd.
07.02.2007	libSINGULAR is born during a visit at UW
08.05.2007	SAGE 2.5 includes libSINGULAR
04.12.2007	SAGE 2.8.15 includes SINGULAR 3-0-4
16.12.2007	SAGE 2.9 includes POLYBORI
...	constant improvements to the interfaces, porting
02.02.2009	SAGE, SINGULAR, POLYBORI in Debian/unstable
18.06.2009	SAGE 4.0.2 includes SINGULAR 3-1-0

What is SAGE?

- A **distribution** of mathematics software. We make sure that e.g. SINGULAR builds and passes tests on Linux, OSX (, Solaris) on x86, x86_64, PPC, Itanium.
- A **unified interface** to mathematics software, i.e. we make sure that e.g. SINGULAR objects can be converted to MAGMA objects via SAGE.
- A **new system** with lots of new code to fill the gaps, e.g. Gröbner bases over more base fields.

Open Culture I

Information Security Group

In SAGE everything is meant to be open and transparent.

- Most discussions happen on public mailing lists like

sage-devel	all things development, ca. 900 members
sage-release	coordination of releases, 21 members
sage-nt	number theoretic development discussions, 64 members.

- Release managers rotate within the community.
- Alpha releases and release candidates are released to the public.

Open Culture II

Information Security Group

- Every known bug is visible on `http://trac.sagemath.org/sage_trac`.
- Every patch submitted for inclusion with SAGE is available on `http://trac.sagemath.org/sage_trac`.
- Every patch going into SAGE receives public peer review.

Things are not static

When developers complained about a lack of transparency w.r.t. to version numbers, [sage-release] was started to make it more transparent.

Developer Friendly

Information Security Group

- Every copy of SAGE contains everything (except a C compiler) needed to get started with development.

  sources sources are installed by default

  build scripts sage −b, sage −docbuild, sage −pkg

  mercurial each user's SAGE installation is a local repository of a distributed revision control system.

- SAGE comes with documentation on how to get involved

  guide `http://www.sagemath.org/doc/developer/`

  wiki `http://wiki.sagemath.org/TracGuidelines`

  talk `http://wiki.sagemath.org/days16`

The languages exposed to the end user (PYTHON & CYTHON) are the languages used to implement SAGE.

Credit I

- Attribution is taken very seriously in SAGE: Each and every person who ever contributed to SAGE is listed in the release notes.
- If not, it is a bug.
- First time contributes are particulary emphasised.
- Every function explicitly calling e.g. SINGULAR is supposed to have ALGORITHM: Uses Singular in its documentation.
- If this is missing, it is considered a bug.

Credit II

Information Security Group

How do I reference Sage?

If you write a paper using Sage, please reference computations done with Sage by including

```
[SAGE], SAGE Mathematical Software, ...
```

in your bibliography .... Moreover, please attempt to track down what components of SAGE are used for your computation, e.g., PARI?, GAP?, Singular? Maxima? and also cite those systems. If you are in doubt about what software your computation uses, feel free to ask on the sage-devel Google group.

— from the SAGE Tutorial

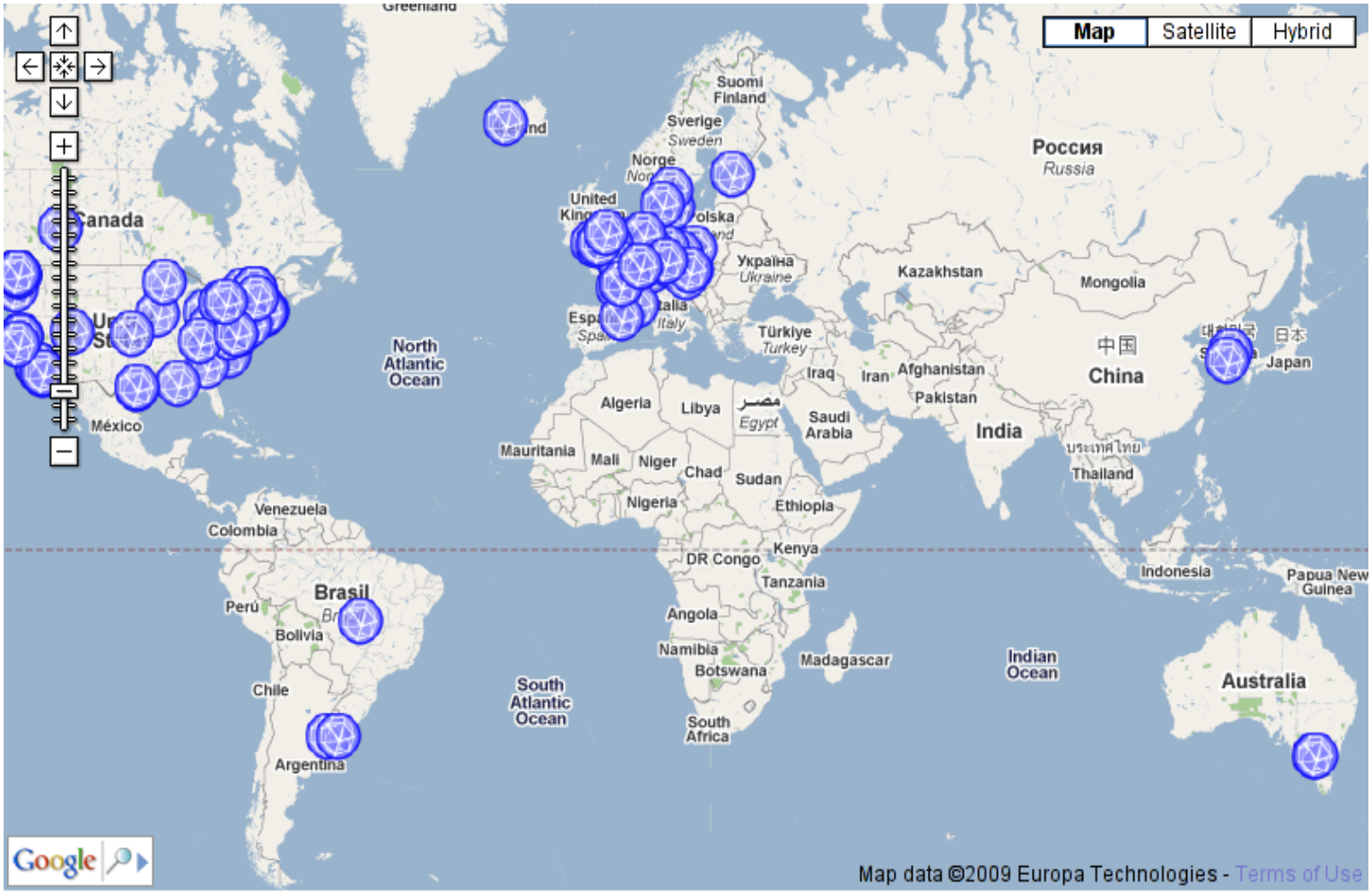
Credit III

Information Security Group

Please let us know if there is anything else we should to do, to ensure that SINGULAR receives proper credit!

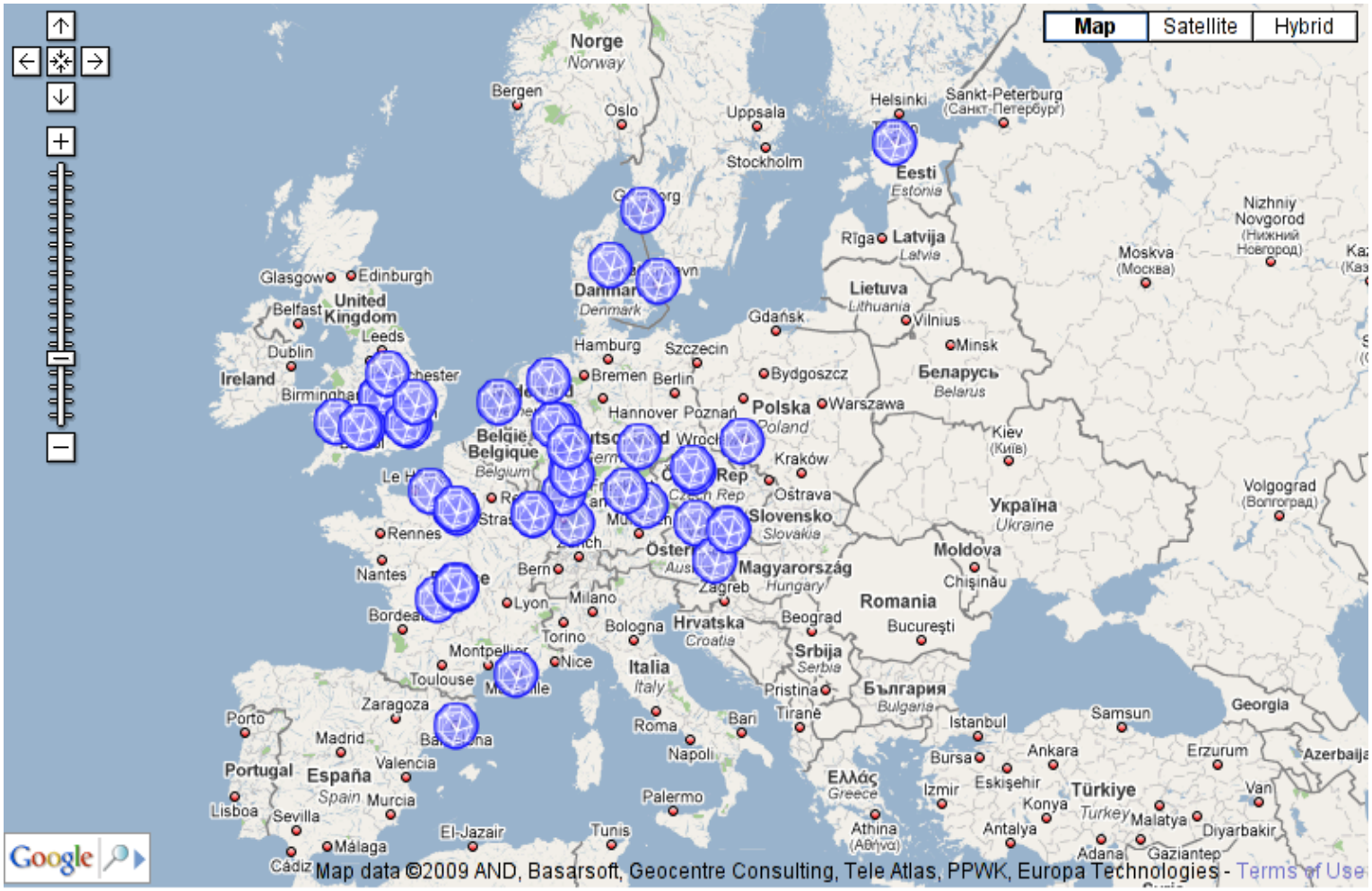
International I

Information Security Group



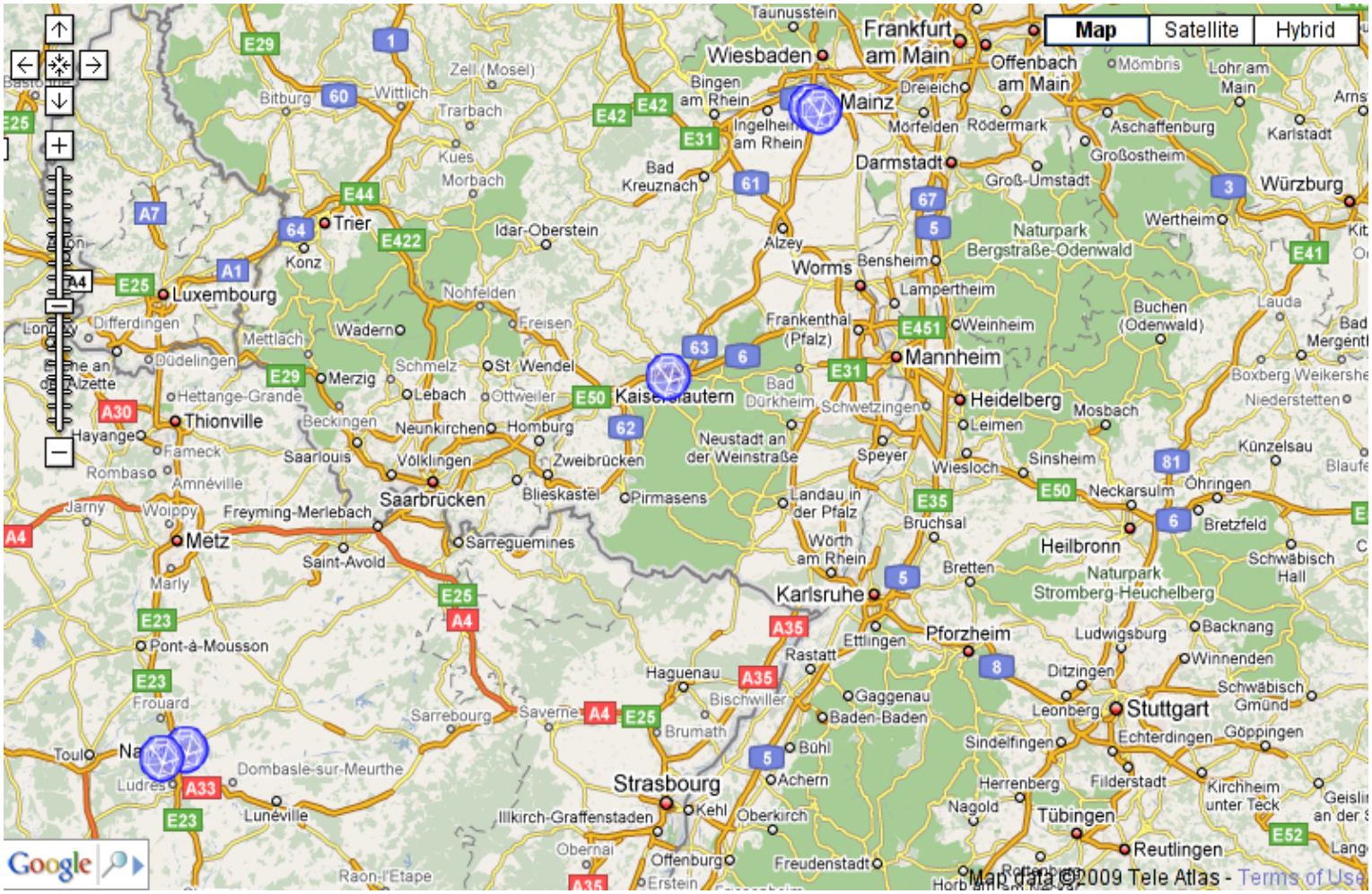
International II

Information Security Group



International III

Information Security Group



Collaborative

Information Security Group

- SAGE didn't invent its own scripting language, it uses PYTHON.
- While CYTHON is developed partly by SAGE developers it is an independent project.
- SAGE uses as much as possible existing packages like SINGULAR, POLYBORI, LinBox, Pari and FLINT (overall more than 90 packages).
- SAGE projects are made idependent projects if it seems beneficial (M4RI, notebook, CYTHON).
- We try to give back to upstream.

Table of Contents

General Comments I

- SAGE is object oriented.
  - PYTHON has multiple inheritance.
  - CYTHON extensions have single inheritance.
- Design is heavily inspired by MAGMA, but SAGE is more object oriented.
- There is no global ring, global modulus etc.
- Most elements (e.g. polynomials) have a parent (e.g. a polynomial ring).
- Big computations in SAGE should be interruptable (user presses Ctrl-C).

SINGULAR interfaces I

Using a pseudo terminal, SAGE can call any SINGULAR function that is callable from the SINGULAR interpreter:

```
sage: r = singular.ring(0,'(x,y,z)','dp')
sage: f = singular('x + 2*y + 2*z - 1')
sage: g = singular('x^2 + 2*y^2 + 2*z^2 - x')
sage: h = singular('2*x*y + 2*y*z - y')
sage: I = singular.ideal(f,g,h)
sage: I.std()
x+2*y+2*z-1,
10*y*z+12*z^2-y-4*z,
4*y^2+2*y*z-y,
210*z^3-79*z^2+7*y+3*z
```

Singular interfaces II

Alternatively, we can pass commands directly:

```
sage: _ = singular.eval('ring r = 0,(x,y,z),dp')
sage: _ = singular.eval('poly f= x + 2*y + 2*z - 1')
sage: _ = singular.eval('poly g = x^2 + 2*y^2 + 2*z^2 - x')
sage: _ = singular.eval('poly h = 2*x*y + 2*y*z - y')
sage: _ = singular.eval('ideal i = f,g,h')
sage: print singular.eval('std(i)')
_[1]=x+2y+2z-1
_[2]=10yz+12z2-y-4z
_[3]=4y2+2yz-y
_[4]=210z3-79z2+7y+3z
```

SINGULAR interfaces III

This functionality is used by several native SAGE objects to perform calculations.

```
sage: P.<x,y,z> = PolynomialRing(QQ,3)
sage: I = sage.rings.ideal.Katsura(P,3)
sage: I.groebner_basis() # calls Singular in background
[x + 2*y + 2*z - 1, 10*y*z + 12*z^2 - y - 4*z, 5*y^2 - 3*z^2 - y + z,
210*z^3 - 79*z^2 + 7*y + 3*z]
```

For longer running calculations, this is a good strategy. However, the communication channel via pseudo ttys and string parsing has too much overhead for short computations like polynomial arithmetic.

SINGULAR interfaces IV

Thus, for low level arithmetic, we link against libSINGULAR directly:

```
sage: P.<x,y,z> = PolynomialRing(QQ,3)
sage: type(P).__name__
'MPolynomialRing_libsingular'
sage: type(x).__name__
'MPolynomial_libsingular'
sage:  x._add_??
cpdef ModuleElement _add_( left, ModuleElement right):
     ...
     cdef poly *_p
     singular_polynomial_add(&_p, left._poly, right._poly,
                             left._parent._ring)
     return new_MP(left._parent, _p)
sage: p = (x + y + z + 1)^10
sage: q = p + 1
sage: %timeit p*q
100 loops, best of 3: 8.49 ms per loop
```

POLYBORI Interface I

Information Security Group

POLYBORI has its own PYTHON interface. However,

- this interface uses the BOOST library,
- it does not follow SAGE conventions and
- it is not well integrated into the SAGE object hierarchy.

Thus Burcin Eröcal and myself re-implemented the POLYBORI PYTHON wrapper in CYTHON.

- The wrapper is relatively thin.
- The wrapper also re-implements POLYBORI's native interface.
- It thus allows to to re-use all of POLYBORI's PYTHON scripts.

POLYBORI Interface II

```
sage: B.<a,b,c> = BooleanPolynomialRing()
sage: type(a).__name__
'BooleanPolynomial'
sage: a._add_??
cpdef ModuleElement _add_(left, ModuleElement right):
    ...
    cdef BooleanPolynomial p = new_BP_from_PBPoly(\
            left._parent, left._pbpoly)
    p._pbpoly.iadd( right._pbpoly )
    return p
sage: p = (a + b + c + 1)
sage: q = p + 1
sage: p*q
0
```

Multivariate Polynomials & Ideals I

Information Security Group

Multivariate Polynomials over the following coefficient rings use SINGULAR via C++ directly.

- $\mathbb{Z}$ – the integers,
- $\mathbb{Q}$ – the rational numbers,
- $\mathbb{Z}/n\mathbb{Z}$ – integers mod n,
- $\mathbb{F}_p$ – finite fields of order $p \leq 2147483629$, p prime,
- $\mathbb{F}_{p^n}$ – finite fields of characteristic $p \leq 2147483629$, p prime and
- $\mathbb{Q}(a)$ – absolute number fields.

Multivariate Polynomials & Ideals II

The following base fields are implemented in SINGULAR but not properly wrapped on a C++ level yet:

- rational function fields,
- $\mathbb{R}$ – real numbers and
- $\mathbb{C}$ – complex numbers.

However, we compute Gröbner bases over these fields using SINGULAR via pexpect.

```
sage: P.<x,y,z> = PolynomialRing(CC)
sage: I = sage.rings.ideal.Katsura(P)
sage: I.groebner_basis() # calls Singular
[z^3 + (-0.376190476190476)*z^2 + 0.0333333333333333*y + ...
```

Multivariate Polynomials & Ideals III

Information Security Group

- longer computations are outsourced to SINGULAR via pexpect (e.g. Gröbner basis computations).
- Multivariate polynomial ideals are mainly implemented via SINGULAR's pexpect interface, but some libSINGULAR function calls exist.
- SAGE exposes all term orderings SINGULAR supports except generic matrix orderings and weighted orderings.

```
sage: T = TermOrder('lex',3) + TermOrder('deglex',3); T
lex(3),deglex(3) term order
sage: P = PolynomialRing(GF(127),6,'x',order=T)
sage: I = sage.rings.ideal.Katsura(P)
sage: P.inject_variables()
Defining x0, x1, x2, x3, x4, x5
sage: x0 > x4^3
True
sage: x3 > x4^3
False
```

Multivariate Polynomials & Ideals IV

Information Security Group

All other coefficient rings use a generic and slow implementation:

```
sage: P.<x,y> = PolynomialRing(GF(2147483629))
sage: type(x).__name__
'MPolynomial_libsingular'
sage: %timeit x*y
1000000 loops, best of 3: 287 ns per loop
```

```
sage: P.<x,y> = PolynomialRing(GF(next_prime(2147483629+1)))
sage: type(x).__name__
'MPolynomial_polydict'
sage: %timeit x*y
100000 loops, best of 3: 10.3 micros per loop
```

Multivariate Polynomials & Ideals V

Information Security Group

Examples of other coefficient rings SAGE supports

- relative number fields
- $\overline{\mathbb{Q}}$ the (exact) field of algebraic numbers
- real interval fields
- p-adic rings with various models

```
sage: Frac(Zp(7))
7-adic Field with capped relative precision 20
sage: P.<x,y,z> = PolynomialRing(Frac(Zp(7)))
sage: I = sage.rings.ideal.Cyclic(P)
sage: gb = I.groebner_basis()
verbose 0 ... Warning: falling back to very slow toy implementation.
sage: gb
[x*y*z + 6 + 6*7 + 6*7^2 + 6*7^3 + ... 6*7^18 + 6*7^19 + O(7^20),
z^3 + 6 + 6*7 + 6*7^2 + 6*7^3 + 6*7^4 + 6*7^5 + 6*7^6 + ... + O(7^20),
x*y + x*z + y*z,
y^2 + y*z + z^2,
x + y + z]
```

Objects Building on Multivariate Polynomials

Information Security Group

fraction fields use multivariate polynomials via PYTHON and are in a quite immature state in SAGE.

quotient rings are also neglected and use multivariate polynomials via PYTHON.

modules over polynomial rings use polnomials via PYTHON.

matrices over polynomial rings use some SINGULAR routines directly, but are mainly implemented quite naively using PYTHON data structures.

Boolean Polynomial Rings I

The user can construct POLYBORI rings explicitly but POLYBORI is not use inexplicitly, when quotient rings are constructed.

```
sage: P = PolynomialRing(GF(2),100,'x')
sage: I = sage.rings.ideal.FieldIdeal(P)
sage: Q = P.quotient(I)
sage: f = Q(P.random_element(degree=3,terms=2000))
sage: g = Q(P.random_element(degree=3,terms=2000))
sage: %time _ = f*g
CPU times: user 44.87 s, sys: 0.36 s, total: 45.24 s
```

```
sage: B = BooleanPolynomialRing(100,'x')
sage: f = B.random_element(degree=3,terms=2000)
sage: g = B.random_element(degree=3,terms=2000)
sage: %time _ = f*g
CPU times: user 4.77 s, sys: 0.06 s, total: 4.83 s
```

Boolean Polynomial Rings II

Information Security Group

POLYBORI is not explicitly used by any higher level object, except

- AES equation system can be constructed as POLYBORI system directly and
- polynomial systems over $\mathbb{F}_2$ have some functions directly relying on POLYBORI

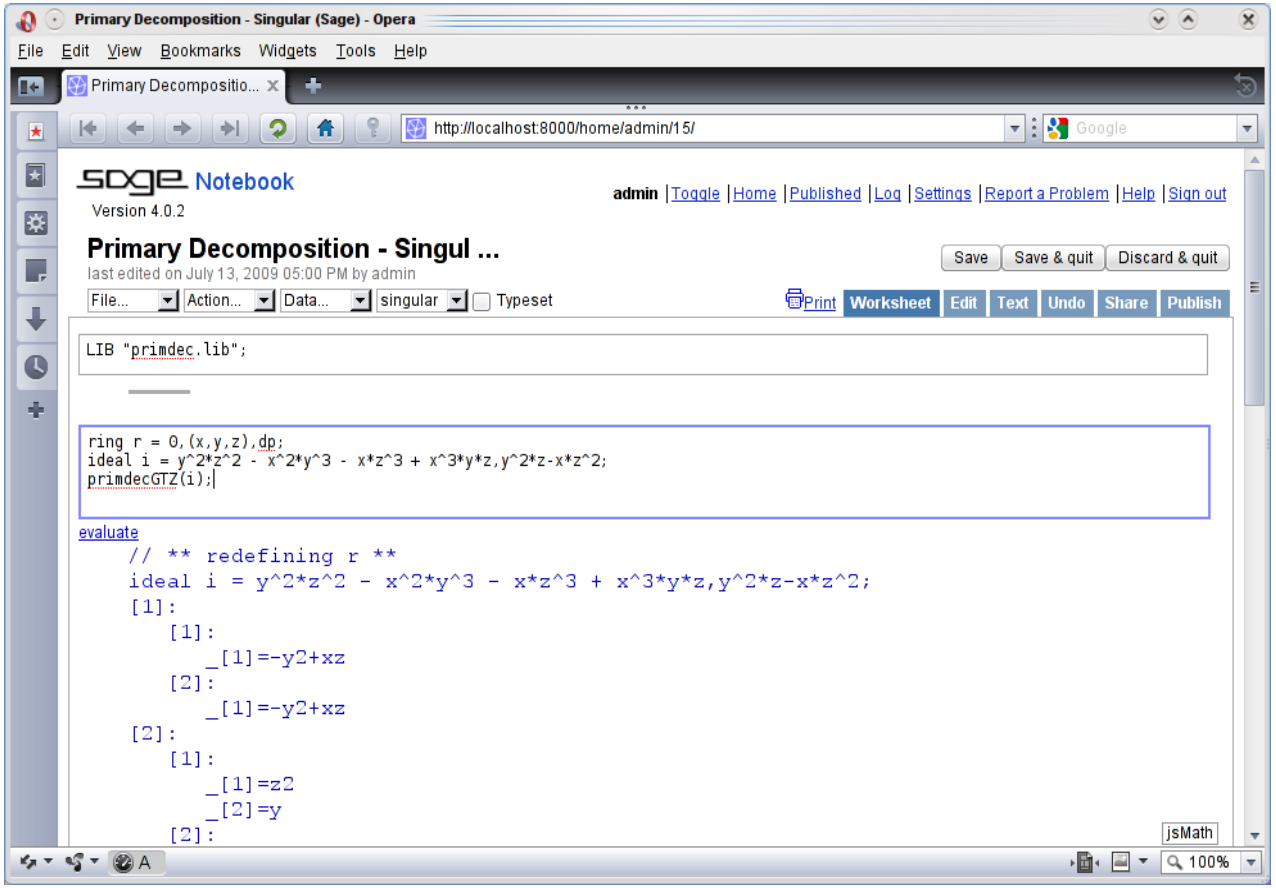
```
sage: MS = MatrixSpace(B,3,3)
sage: MS.random_element()
[   a*c + b*c + b + c + 1    a*b + a + b*c + b + c    a*b + b*c + b + c + 1]
[a*b + a*c + b*c + b + 1    a*c + a + b*c + b + 1      a + b*c + b + c + 1]
[   a*b + a*c + a + b + c    a*b + a*c + a + b + 1      a + b*c + b + c + 1]
```

```
sage: sr = mq.SR(2,2,1,4,gf2=True,polybori=True)
sage: F,s = sr.polynomial_system()
sage: type(F)
<class 'sage.crypto.mq.mpolynomialsystem.MPolynomialSystem_gf2'>
sage: F2 = F.eliminate_linear_variables() # PolyBoRi ll_red_nf
sage: %time gb = F.groebner_basis()
CPU times: user 1.04 s, sys: 0.03 s, total: 1.06 s
Wall time: 1.39 s
```

Table of Contents

SAGE is a Networked GUI for SINGULAR

Information Security Group



The SAGE notebook is becoming an independent project now, so it could be re-branded for SINGULAR in the future.

Coercion and Conversion I

```
sage: P.<x,y> = GF(2)[]
sage: P.<x> = GF(2)[]
sage: type(y)
<type 'sage.rings.polynomial.multi_polynomial_libsingular.MPolynomial_libsi
sage: type(x)
<type 'sage.rings.polynomial.polynomial_gf2x.Polynomial_GF2X'>
sage: x + y
x + y
sage: parent(x + y)
Multivariate Polynomial Ring in x, y over Finite Field of size 2
```

```
sage: P.<x,y> = ZZ[]
sage: P.<y,z> = GF(127)[]
sage: x + z
x + z
sage: parent(x+z)
Multivariate Polynomial Ring in x, y, z over Finite Field of size 127
```

Coercion and Conversion II

```
sage: P.<x,y> = CC[]
sage: f = (3*x^3 + 7*x + 1)*(2*x^2 + 1.5)
sage: f
6.0*x^5 + 18.5*x^3 + 2.0*x^2 + 10.5*x + 1.5
sage: f.factor()
(6.0) * (x - 0.0708196728575949 - 1.53244236809229*I) *
(x - 0.0708196728575949 + 1.53244236809229*I) *
(x- 0.866025403784439*I) * (x + 0.866025403784439*I) *
(x + 0.141639345715190)
sage: singular(f).factorH()
...
TypeError: Singular error:
   ? not implemented
   ? error occurred in general.lib::factorH ...
   ? leaving general.lib::factorH
sage: type(f.univariate_polynomial()).__name__
Polynomial_generic_dense_field
```

Parallelism I

```
from processing import Process, Queue

def pgb(I, algorithm, q):
    singular._start() # new singular instance
    t = singular.cputime()
    gb = I.groebner_basis(algorithm="singular:"+algorithm)
    t = singular.cputime(t)
    print algorithm, "finished in", t, "seconds"
    q.put([t, gb, algorithm])

I = sage.rings.ideal.Katsura(PolynomialRing(QQ,8,'x'))

q = Queue() # for sending back the result
processes = []
for algorithm in ("slimgb","std","stdhilb","groebner"):
   p =  Process(target=pgb, args=(I,algorithm,q))
   p.start()
   processes.append(p)

t, gb, algorithm = q.get()
[p.terminate() for p in processes] # kill the other processes
print t, algorithm, gb[-1]
```

Parallelism II

```
malb@road:$ sage parallel.sage # Katsura-8 over QQ w.r.t. degrevlex
groebner finished in 6.25 seconds
std finished in 6.24 seconds
6.25 groebner x0 + 2*x1 + 2*x2 + 2*x3 + 2*x4 + 2*x5 + 2*x6 + 2*x7 - 1

malb@road:$ sage parallel.sage # Cyclic-7 over GF(32003) w.r.t. degrevlex
slimgb finished in 4.73 seconds
groebner finished in 5.3 seconds
std finished in 5.31 seconds
stdhilb finished in 8.33 seconds
4.73 slimgb x0 + x1 + x2 + x3 + x4 + x5 + x6
```

Parallelism III

```
sage: P = PolynomialRing(QQ,9,'x')
sage: I = sage.rings.ideal.Katsura(P)
sage: def mod_gb(p, I):
...     J = I.change_ring(I.ring().change_ring(GF(p)))
...     return J.groebner_basis('libsingular:std')

sage: %time l = [mod_gb(p,I) for p in prime_range(1000,1100)]
CPU time: 69.48 s,  Wall time: 70.09 s

sage: from sage.parallel.multiprocessing import parallel_iter
sage: my_args = []
sage: for p in prime_range(1000,1100):
...       my_args.append( ((p,I),{}) )
sage: %time v = list(parallel_iter(2, mod_gb, my_args))
CPU time: 10.69 s,  Wall time: 52.04 s
```

Parallelism IV

Information Security Group

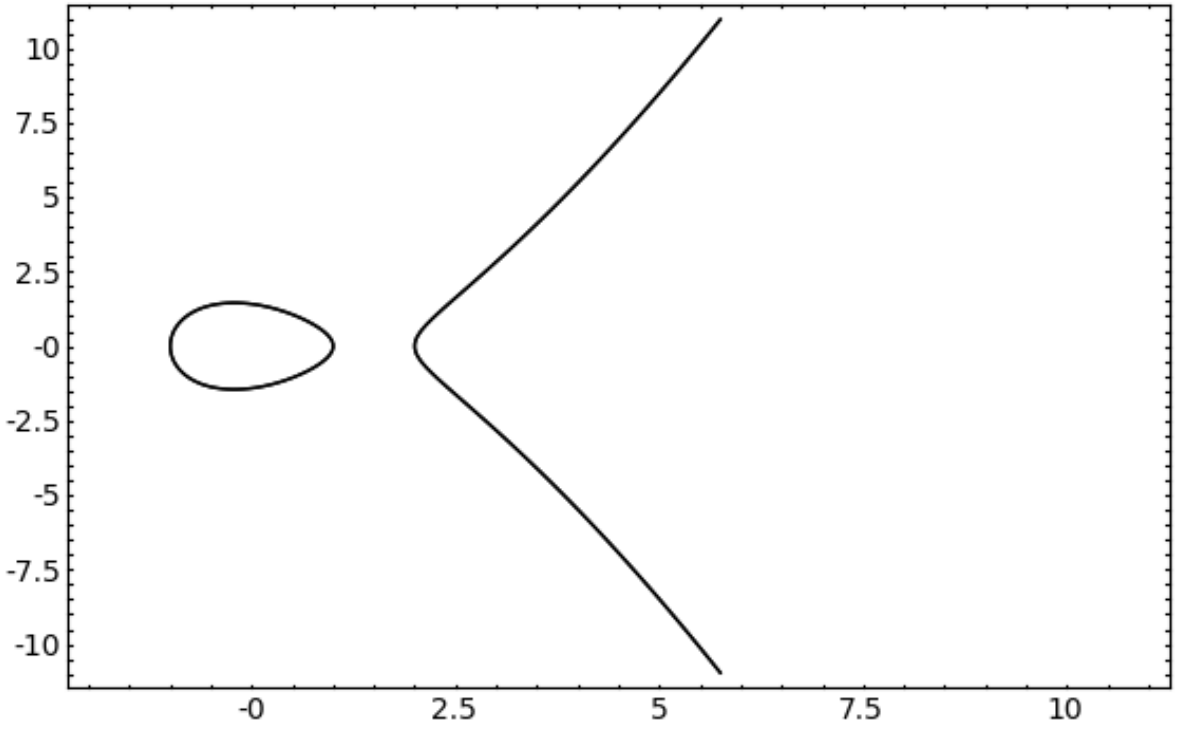
sage.math

```
sage: P = PolynomialRing(QQ,8,'x')
sage: I = sage.rings.ideal.Cyclic(P)
sage: %time l = [mod_gb(p,I) for p in prime_range(1000,1200)]
CPU times: user 1434.89 s, sys: 0.99 s, total: 1435.88 s
Wall time: 1435.97 s
```

```
sage: from sage.parallel.multiprocessing import parallel_iter
sage: my_args = []
sage: for p in prime_range(1000,1200):
....:     my_args.append( ((p,I),{}) )
....:
sage: %time v = list(parallel_iter(4, mod_gb, my_args))
CPU times: user 78.00 s, sys: 1.95 s, total: 79.95 s
Wall time: 427.24 s
```

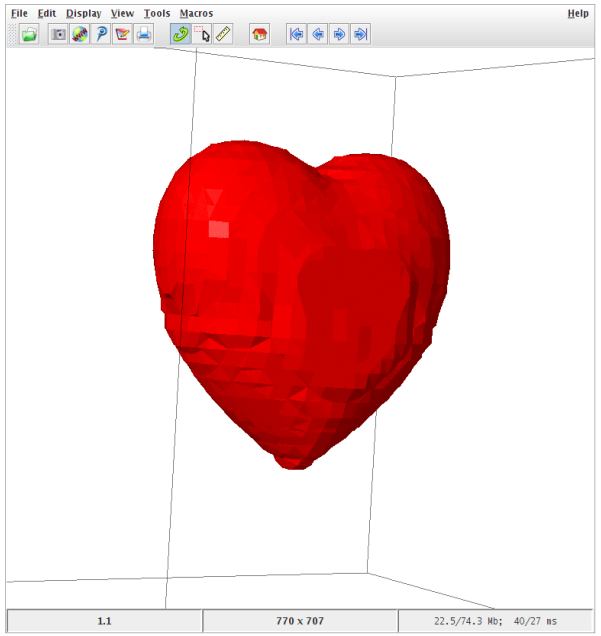
Visualisation I

```
sage: R.<x,y> = PolynomialRing(QQ,2)
sage: I = R.ideal([y^2-(x^2-1)*(x-2)])
sage: I.plot()
```



Visualisation II

```
sage: x,y,z = var('x,y,z')
sage: f=(2*x^2+y^2+z^2-1)^3-(1/10)*x^2*z^3-y^2*z^3 == 0
sage: implicit_plot3d(f,(-2,2),(-2,2),(-2,2),rgbcolor='red')
```



We didn't write the proper wrapper code yet to do this with principal ideals in three variables. It should be easy though.

Visualisation III

We visualise the numerical instability of Gröbner basis algorithms using @interact, SINGULAR and matplotlib.

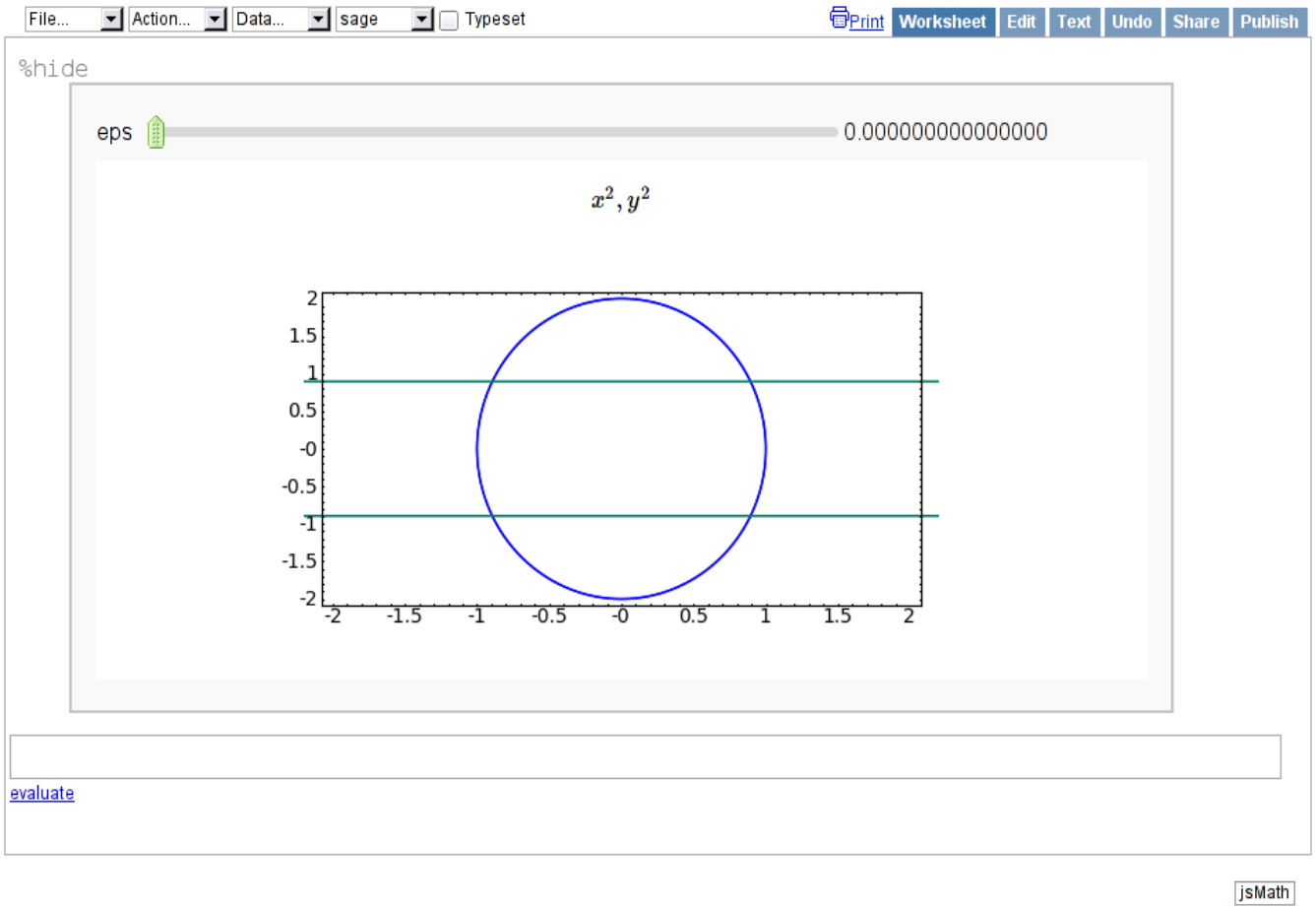
```
@interact
def _(eps=slider(0.00,10.0,step_size=0.5)):
    P.<x,y> = RR[]

    f1 = 4*x^2 + y^2 - 4

    f2 = 4*eps*x*y + 15*y^2 - 12 + 0.001*x

    p = Ideal(f1).plot(cmap='winter') + Ideal(f2).plot(cmap='summer')
    gb = Ideal([f1,f2]).groebner_basis()
    lm_str = ", ".join([latex(f.lm()) for f in gb])
    html('<center>$%s$</center>'%(lm_str))
    html('<center>')
    p.show(xmin=-2,xmax=2,ymin=-2,ymax=2,aspect_ratio=2,figsize=(3,5))
    html('</center>')
```

Visualisation IV



Visualisation V

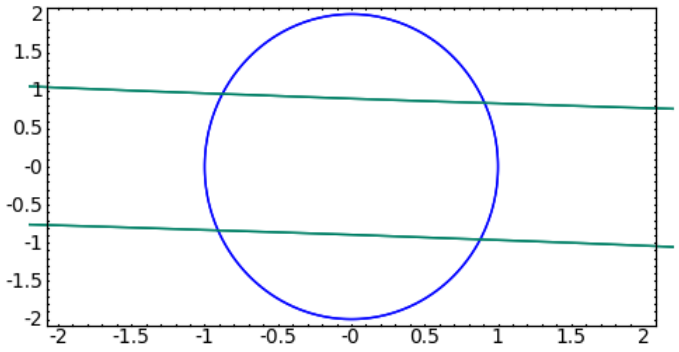
Information Security Group

File... Action... Data... sage Typeset Print Worksheet Edit Text Undo Share Publish

```
%hide
```

eps 0.500000000000000

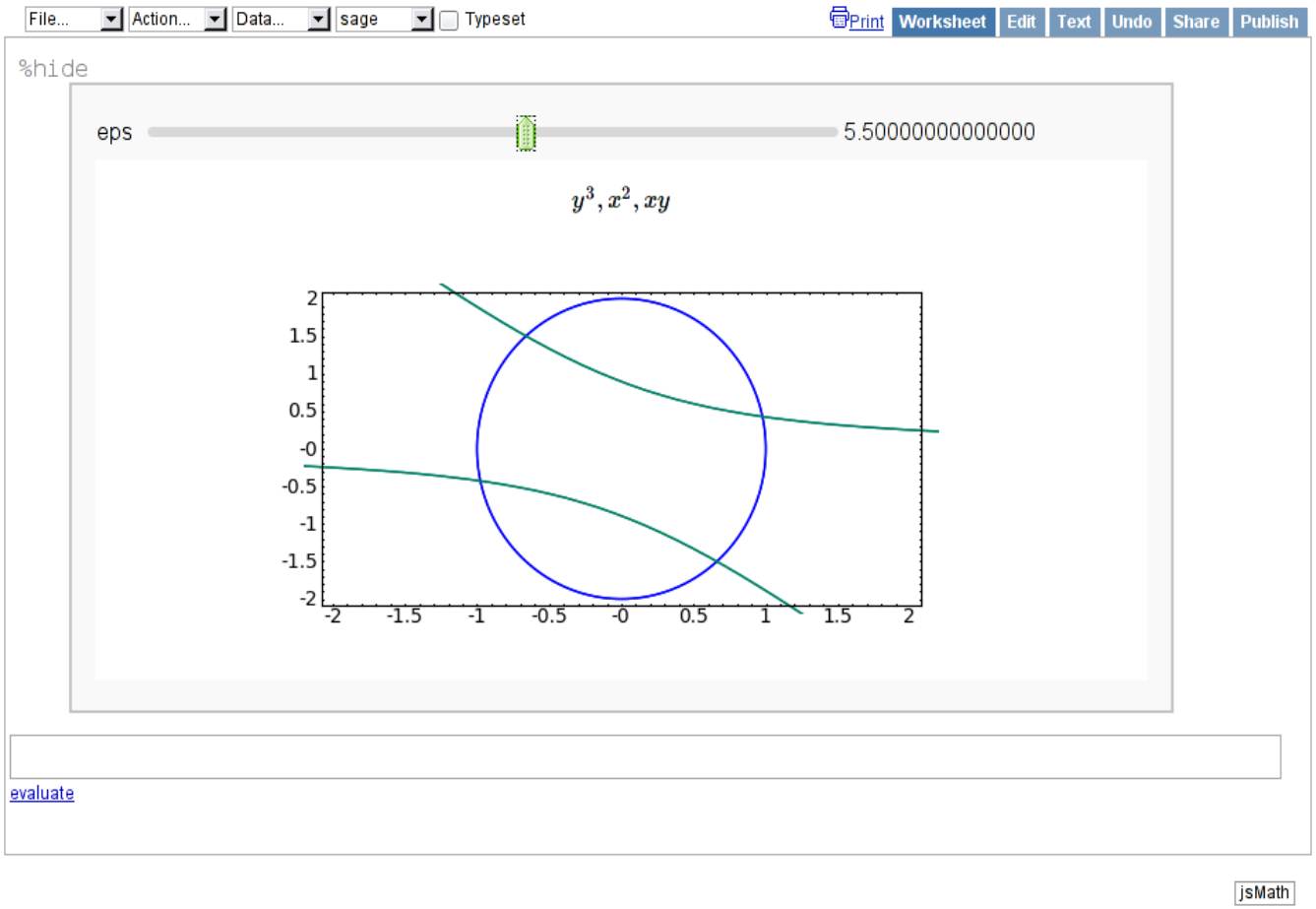
y^3, x^2, xy



evaluate

jsMath

Visualisation VI

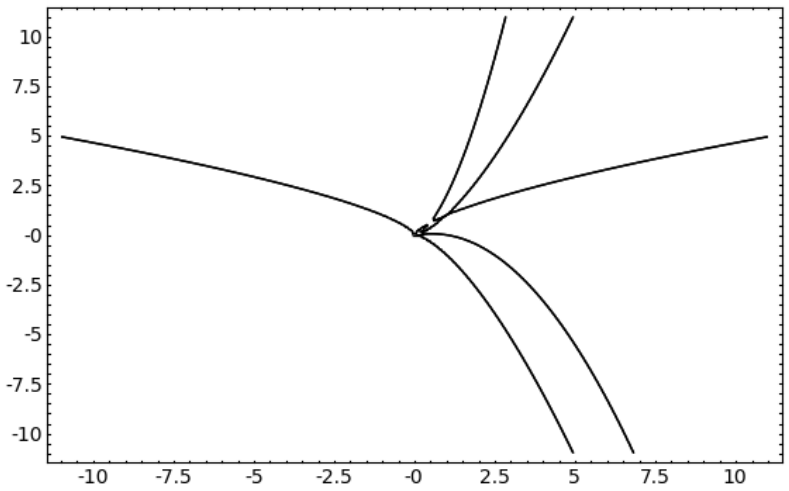


Interaction with Other Packages I

Information Security Group

We construct a highly singular curve and plot it using matplotlib.

```
sage: R.<x,y> = QQ[]
sage: f1 = (y^2 - x^3)^2 - 4*x^5*y - x^7
sage: f2 = y^2 - x^3
sage: f3 = y^3 - x^2
sage: Ideal(f1*f2*f3).plot()
```



Interaction with Other Packages II

Information Security Group

Then, we convert it to SINGULAR.

```
sage: g = singular(f1*f2*f3); print g
x^10*y^3-x^12-x^9*y^3+4*x^8*y^4-x^7*y^5+x^11-4*x^10*y+x^9*y^2+3*x^6*y^5-\
4*x^5*y^6-3*x^8*y^2+4*x^7*y^3-3*x^3*y^7+3*x^5*y^4+y^9-x^2*y^6
```

and use SINGULAR to compute the the **incidence matrix** M of the Enriques diagram:

```
sage: singular.load('alexpoly.lib')
sage: M = g.proximitymatrix()[3]; M
```

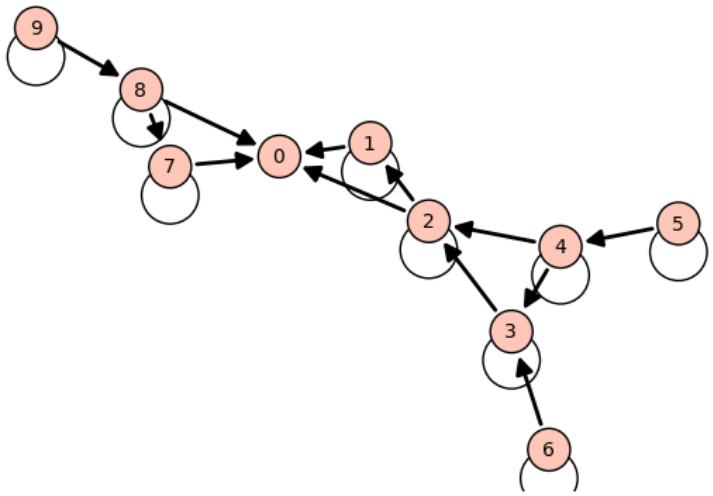
$$
\begin{matrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & -1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & -1 & 4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 5 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 4 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
-1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 3
\end{matrix}
$$

Interaction with Other Packages III

Information Security Group

Finally, we move the matrix back to SAGE and plot the Enriques diagram:

```
sage: G = DiGraph(M.sage(), incidence_matrix=True)
sage: G.plot(scaling_term=0.15)
```



Interaction with Other Packages IV

Information Security Group

Next, we compute and plot the Newton polytope of $f_1 \cdot f_2 \cdot f_3$ using cddlib:

```
sage: p = (f1*f2*f3).newton_polytope()
sage: p.show()
```

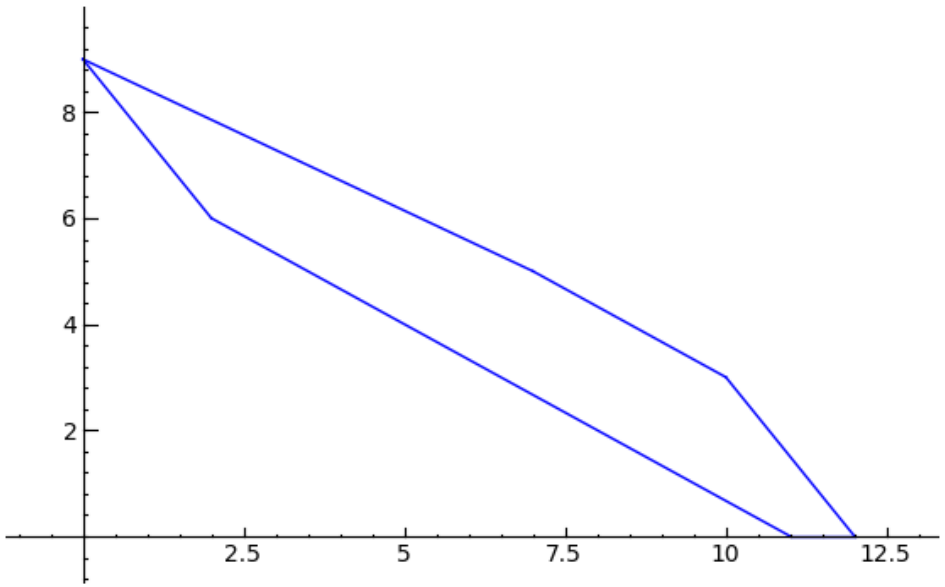


Table of Contents

More Users

In the three months April, March and June 2009 the statistics are as follows:

- The SAGE website had 100,000 unique vistors.
- SAGE was downloaded at least 12,000 times.
- The majority downloaded the virtual machine for Windows (5668).
- Downloads by country:

United States	5017	34.94%
Germany	1402	9.77%
France	693	4.83%
UK	668	4.65%

The term "Singular" was mentioned in 402 and 1,260 different postings on [sage-support] and [sage-devel] respectively. The term "PolyBoRi" appears in 57 and 473 postings respectively.

Bug Reports & Testing

- Before each release, we build and test SAGE on a variety of platforms: Debian, Ubuntu, Redhat Linux, SuSE Linux, Solaris, OSX on Itanium, x86, x86_64 and PPC.
- We run thousands of tests which includes hundreds of tests which call SINGULAR and POLYBORI somehow.
- Our users also build and test SAGE on a wide variety of platforms.
- We report bugs upstream, if possible with a suggestion on how to fix it, to save you guys time.

Porting to New Platforms

Information Security Group

Sage developers ported a wide variety of packages to new platforms. Examples are:

OSX 64-bit	Michael Abshoff, Martin Albrecht
OSX PPC	Michael Abshoff
Solaris x86	Michael Abshoff, David Kirkby
Linux Itanum	Michael Abshoff
FreeBSD x86	Michael Abshoff
Cygwin x86	Michael Abshoff, Martin Albrecht

Debian & Ubuntu I

```
Package: singular
Priority: optional
Section: math
Installed-Size: 9708
Maintainer: Tim Abbott <tabbott@mit.edu>
Architecture: amd64
Version: 3-0-4-3.dfsg-2
Depends: libc6 (>= 2.7-1), libgcc1 (>= 1:4.1.1), libgmp3c2, libncurses5
(>= 5.6+20071006-3), libntl-5.4.2, libreadline5 (>= 5.2),
libstdc++6 (>= 4.2.1), libsingular-3-0-4-3
Filename: pool/main/s/singular/singular_3-0-4-3.dfsg-2_amd64.deb
Size: 3403138
MD5sum: 1280797acc6db0cabfd04aede03327fa
SHA1: 7a2bea47e0a75eacde0a7eabfbe602d0ed194f14
SHA256: 9c40b125991e9154181efed985354f0b80d8fa1a330b1928b197599d5c40071d
Description: A commutative algebra system
 SINGULAR is a Computer Algebra System for polynomial computations
 with special emphasis on the needs of commutative algebra,
 algebraic geometry, and singularity theory.
 .
 Main computational objects: ideals/modules over very general
 polynomial rings over various ground fields.
Homepage: http://www.singular.uni-kl.de/
Tag: uitoolkit::ncurses
```

Debian & Ubuntu II

```
Package: python-polybori
Priority: optional
Section: python
Installed-Size: 11376
Maintainer: Tim Abbott <tabbott@mit.edu>
Architecture: amd64
Source: polybori
Version: 0.5~rc1-1
Provides: polybori
Depends: libboost-python1.34.1 (>= 1.34.1-8), libc6 (>= 2.7-1), libgcc1 (>=
libstdc++6 (>= 4.2.1), ipython, libpolybori-dev, python (<< 2.6), python (>=
python-central (>= 0.6.7)
Filename: pool/main/p/polybori/python-polybori_0.5~rc1-1_amd64.deb
Size: 3388996
MD5sum: 9ac0323c00cc5b04e7c0591f92269a99
SHA1: 15d18762a2bba51e47479ad931e5ebe1ab61644d
SHA256: 97abfc82efcfe0f74a463aee9770493d7291f4fba04757965667806 9f9f245ad
Description: Polynomials over Boolean Rings, Python module
 The core of PolyBoRi is a C++ library, which provides high-level data
 types for Boolean polynomials and monomials, exponent vectors, as
 well as for the underlying polynomial rings and subsets of the
 powerset of the Boolean variables. As a unique approach, binary
 ...
Homepage: http://polybori.sourceforge.net/
```

Interfaces to Other Systems I

```
sage: P.<a,b,c,d,e,f> = PolynomialRing(GF(32003))
sage: I = Ideal([P.random_element() for _ in range(6)])
sage: singular(I)

3739*a*c-1901*d^2-3983*a*e-2720*e*f+4962*b,
10238*d*e+1638*e^2-6317*d*f-7668*c+6349*e,
-13387*c^2-4816*c*d+7527*b*f+11346*d*f+10400*d,
6074*c*d-12486*d^2+6859*b*f-174*c*f+7662*a,
10209*b*d-7274*c*d-6890*a*e-9251*d*f+2002*e*f,
11249*c^2-11407*d^2-11235*d*e-14036*c*f+9445*e*f
sage: magma(I)

Ideal of Polynomial ring of rank 6 over GF(32003)
Order: Graded Reverse Lexicographical
Variables: a, b, c, d, e, f
Basis:
[
3739*a*c + 30102*d^2 + 28020*a*e + 29283*e*f + 4962*b,
10238*d*e + 1638*e^2 + 25686*d*f + 24335*c + 6349*e,
18616*c^2 + 27187*c*d + 7527*b*f + 11346*d*f + 10400*d,
6074*c*d + 19517*d^2 + 6859*b*f + 31829*c*f + 7662*a,
10209*b*d + 24729*c*d + 25113*a*e + 22752*d*f + 2002*e*f,
11249*c^2 + 20596*d^2 + 20768*d*e + 17967*c*f + 9445*e*f
]
```

Interfaces to Other Systems II

Information Security Group

Thus SAGE allows SINGULAR to exchange data with MAGMA and the other way around.

```
sage: fs = singular(I.gens()[0])
sage: fs
3739*a*c-1901*d^2-3983*a*e-2720*e*f+4962*b
sage: magma(P(fs))
3739*a*c + 30102*d^2 + 28020*a*e + 29283*e*f + 4962*b

sage: sage: magma(P(fs)).Factorisation()
[
<a*c + 23118*d^2 + 7377*a*e + 16142*e*f + 532*b, 1>
]
```

libSingular I

Information Security Group

How it all began:

```
On Wed, 07 Feb 2007 16:47:54 -0700, Martin Albrecht wrote:
|  gcc -c -ggdb -I./kernel -I/home/malb/sage-2.0-singular/local/include/test.cc
|
|  gcc Singular/Singular test.o -o test
|
|   ./test
|
| //   characteristic : 0
| //   number of vars : 1
| //        block   1 : ordering lp
| //                  : names    a
|
| it works!

WOW!  You're using singular in library mode!? Frickin' cool.

William
```

libSINGULAR II

Information Security Group

- libSINGULAR is the C/C++ interface we use to link into the SINGULAR kernel directly.
- On top of that we have a thin CYTHON wrapper.
- This is also like a reference implementation how to use SINGULAR as a C/C++ library.
- The main difficulty was to figure out, what the assumptions of various functions are, which function to call and when the global ring is needed and when not.
- libSINGULAR does not support calling SINGULAR library code yet.
- Other packages start to use it (e.g., GFAN) or are considering it (e.g. GIAC).

A Faster Frontend

```
sage: P.<x,y> = PolynomialRing(GF(127))
sage: t = cputime()
sage: f = 0
sage: for i in range(10^5): f = f + x*y
....:
sage: cputime(t)
0.15397700000000025
```

```
sage: r = singular(P)
sage: r
//   characteristic : 127
//   number of vars : 2
//        block   1 : ordering dp
//                  : names    x y
//        block   2 : ordering C
sage: t = cputime()
sage: t = singular.cputime()
sage: singular.eval("poly f=0; int i=0; for(i=0;i<10^5;i=i+1){f=f+x*y;}");
sage: singular.cputime(t)
2.8300000000000001
```

M4RI and POLYBORI I

Information Security Group

M4RI is an independent library for dense linear algebra over $\mathbb{F}_2$ which would not exist in this form if it wasn't for SAGE:

- The main authors of M4RI are Gregory Bard, Clément Pernet, Bill Hart and myself.
- It was started by Gregory Bard in 2006.
- We invited Greg to SAGE Days 3 in Los Angeles and convinced him to release his code under the GPL. During the following months I improved the code etc.
- In May 2008 Bill Hart – a SAGE developer – and myself made a major push to make it more efficient. All discussions happened on the [sage-devel] mailing list.
- Later in 2008 Clément Pernet jumped in to implement asymptotically fast LQUP factorisation. We met Clément at a SAGE related workshop for the first time too.

M4RI and POLYBORI II

Information Security Group

- Tim Abbott fixed up the library to get into Debian, Jean-Guillaume Dumas implemented linear system resolution, David Harvey implemented parallel parity function, Peter Jeremy fixed up the M4 macros. All this work was related to SAGE.
- M4RI is used in POLYBORI for elimination of dense systems.
- Michael reported a 100× speed-up for some problems when switching to M4RI.
- Michael provided examples of interest to him and we made sure to use them to optimise our implementation.
- The current implementation is at least 10–20x faster than the original implementation of the same algorithm. Furthermore, we now have asymptotically fast algorithms.

M4RI would not exist in this form without SAGE.

Table of Contents

SAGE Side

- We want to implement rational function fields on top of libSINGULAR directly.
- We want to implement quotient rings on top of libSINGULAR directly.
- We want to implement modules on top of libSINGULAR directly.
- We want to expose the non-commutative algebras in SINGULAR (PLURAL)
- We want to expose more of SINGULAR's highlevel ideal operations
- We want to implement multivariate power series on top of libSINGULAR.

SINGULAR's Interpreter

- It would be nice, to be able to call the SINGULAR interpreter for one line of code from libSINGULAR.
- This would save us the overhead of interprocess communication.
- libGAP implements this strategy:

```
sage: a = libgap("NormalSubgroups")
sage: b = libgap("SymmetricGroup(4)")
sage: a(b)
[ Group(()), Group([ (1,4)(2,3), (1,3)(2,4) ]), \
  Group([ (2,4,3), (1,4)(2,3), (1,3)(2,4) ]), \
  Sym( [ 1 .. 4 ] )
```

PyObject Coefficient Rings

- PYNAC is a GINAC fork which accepts PyObjects as coefficients.
- Maybe, we could patch SINGULAR to allow the same.
- This would allow arbitrary coefficient rings at least for basic arithmetic.

GCDs and Factoring

- These two functions are missing in all of the open-source world.
- They are constantly demanded.
- But they are either much slower than MAGMA or
- they are not reliable and give wrong results.

Thank You!

Information Security Group

Fin

Table of Contents