SAGE: open source mathematical software (also) for cryptanalysts

Ralf-Philipp Weinmann
<weinmann@cdc.informatik.tu-darmstadt.de>

TECHNISCHE UNIVERSITÄT DARMSTADT

"Building the Car Instead of Reinventing the Wheel"

What is SAGE?

TECHNISCHE UNIVERSITÄT DARMSTADT

- A computer algebra system
- Free software
- has both command-line and GUI
- integrates many other computer algebra systems
- aims to be “best of breed”
  (cherry-picks the best algorithm implementations)
- Available for Linux and MacOS X
- VMware images available for Windows
- Solaris port is underway

Why use SAGE?

TECHNISCHE UNIVERSITÄT DARMSTADT

- ... or rather: what's wrong with other computer algebra systems?
- each CAS has its own language with its own idioms: entry barrier!
- Closed-source software (Magma, Maple, Matlab, Mathematica):
  - not easily extensible
  - can't look at intermediate results
- SAGE:
  - uses a general-purpose scripting language [Python] for user interaction
  - allows introspection
- Trust:
  - "Can we expect somebody to believe a result of a program that he is not allowed to see?" (Joachim Neubüser [founder of GAP], 1993)

Example: Expression simplification

TECHNISCHE UNIVERSITÄT DARMSTADT

```
sage: var('x')
sage: f = sin(x)^2+cos(x)^2
sage: f
sin(x)^2 + cos(x)^2
sage: f.simplify_trig()
1
sage: limit((tan(sin(x)) - sin(tan(x)))/x^7, taylor=True, x=0)
1/30
```

Above functionality is based on Maxima, developed 1982-2001 by William Shelter. Maxima in turn is based on Macsyma, the first computer algebra system ever, created at MIT in the 1960s.

Example: Integer Factorization

TECHNISCHE UNIVERSITÄT DARMSTADT

```
sage: bound = 2^100
sage: p = next_prime(ceil(random()*bound)+bound)
sage: q = next_prime(ceil(random()*bound)+bound)
sage: time factor(n, algorithm='pari')
CPU times: user 1.49 s, sys: 0.20 s, total: 1.69 s
Wall time: 2.64
210015172260555796984629 7 * 22456053712774325380055 87
sage: time qsieve(n)
CPU times: user 0.00 s, sys: 0.00 s, total: 0.00 s
Wall time: 20.10
([210015172260555796984629 7, 22456053712774325380055 87], '')
```

Above functionality is based on Pari and FlintQS. Pari is a CAS developed by Henri Cohen and co-workers. FlintQS is an implementation of the quadratic sieve developed by David Harvey and Bill Hart.

Example: Linear algebra

TECHNISCHE UNIVERSITÄT DARMSTADT

```
sage: dimension=2000
sage: MatSpace = MatrixSpace(GF(2), dimension, sparse=True)
sage: M = MatSpace.random_element(density = 0.05)
sage: time M_echelon = M.echelon_form()
CPU times: user 3.16 s, sys: 0.01 s, total: 3.17 s
Wall time: 3.64
sage: A = random_matrix(GF(127),2000,2000)
sage: B = random_matrix(GF(127),2000,2000)
sage: time D = A * B
CPU times: user 13.72 s, sys: 0.24 s, total: 13.96 s
Wall time: 28.30
sage: time C = A._multiply_linbox(B)
CPU times: user 6.64 s, sys: 0.25 s, total: 6.89 s
Wall time: 11.82
```

SAGE has both its own linear algebra implementation (row echelon form however does not work for sparse matrices over extension fields yet) and is able to make use of LinBox (which in turn bases some of its linear algebra on ATLAS and GSL).

Example: Lattice reduction

TECHNISCHE UNIVERSITÄT DARMSTADT

```
sage: M = random_matrix(ZZ,300)
sage: time M2=M.LLL()
CPU times: user 7.35 s, sys: 0.08 s, total: 7.43 s
Wall time: 10.66
sage: M_magma = magma(M)
sage: time M2_magma = M_magma.LLL()
CPU times: user 0.00 s, sys: 0.00 s, total: 0.00 s
Wall time: 6.27
```

The above benchmark is Stehlé vs. Stehlé. Damien Stehlé implemented both the Magma LLL and the library fpLLL which is underlying SAGE's LLL command.

Example: Systems of equations for AES

TECHNISCHE UNIVERSITÄT DARMSTADT

```
sage: SR = mq.SR(4,2,2,4,gf2=True)
sage: flag = False
sage: while flag == False:
....:         try:
....:             flag = True
....:         plaintext = SR.random_element()
....:         key = SR.random_element()
....:         ciphertext = SR(plaintext,key)
....:     except ZeroDivisionError:
....:             flag = False
....:
sage: F,s = SR.polynomial_system(plaintext,key)
sage: print F
Polynomial System with 672 Polynomials in 240 Variables
```

Above functionality written by Martin Albrecht for SAGE in pure Python. More general framework for this is forthcoming (SAGE Days 6).

Example: Polynomial system solving

TECHNISCHE UNIVERSITÄT DARMSTADT

```
sage: R = PolynomialRing(GF(5), 4, ["a","b","c","d"])
sage: a,b,c,d = R.gens()
sage: I = (a+b+c+d, a*b+a*d+b*c+c*d, a*b*c+a*b*d+a*c*d+b*c*d,
a*b*c*d-1)*R; I
Ideal (a + b + c + d, a*b + b*c + a*d + c*d, a*b*c + a*b*d +
a*c*d + b*c*d, a*b*c*d - 1) of Multivariate Polynomial Ring in
a, b, c, d over Finite Field of size 5
sage: B = I.groebner_basis()
sage: B
[a + b + c + d, b^2 + 2*b*d + d^2, b*c^2 + c^2*d - b*d^2 - d^3,
b*c*d^2 + c^2*d^2 - b*d^3 + c*d^3 - d^4 - 1, b*d^4 + d^5 - b -
d, c^3*d^2 + c^2*d^3 - c - d, c^2*d^4 + b*c - b*d + c*d - 2*d^2]
```

SAGE currently uses Singular to compute Gröbner bases. In the near future, SAGE will also include a fast implementation of Faugère's F4 algorithm (written by me).

SAGE Internals

TECHNISCHE UNIVERSITÄT DARMSTADT

- Written in Python
- wrappers (“glue code”) written for libraries SAGE builds upon
- programs without bindings are tied to SAGE using pipes
- Cython used to compile critical code paths into C
- Source code contains known-answer tests (“doctests”)
- Source lines of code for sage-2.8.9.rc1 [Michael Abshoff]:
  - Total Physical Source Lines of Code (SLOC) = 4,817,399
  - Development Effort Estimate, Person-Years = 1,472.26
    - Ansi C:1907386 (39.59%)
    - Python: 1186092 (24.62%)
    - C++: 553701 (11.49%)
    - Fortran: 492184 (10.22%)
    - Lisp: 340210 (7.06%)
    - Shell script: 138356 (2.87%)

Python example

TECHNISCHE UNIVERSITÄT DARMSTADT

```
def InsertionSort(A):
    for j in range(1, len(A)):
        key = A[j]
        i = j - 1
        while (i >=0) and (A[i] > key):
            A[i+1] = A[i]
            i = i - 1
        A[i+1] = key
```