Adding Spice to Your Research with Sage
Open Source Software for Mathematics

May 10, 2011

Rob Beezer
University of Puget Sound

Perimeter Institute for Theoretical Physics
Waterloo, Canada

1 What is Sage?

- Sage: Open source software for mathematics
- A computer algebra system
- A "distribution" of software for mathematics
- A digital blackboard
- Mission: "Creating a viable free open source alternative to Magma, Maple, Mathematica and Matlab."

2 Who am I?

- "Pure" mathematician – algebra, discrete mathematics
- Teacher at an undergraduate liberal arts college
- Active Sage developer – linear algebra, group theory, graph theory

- Interested in integrating Sage into the classroom (NSF education grant)
- Sage Lecturer, African Institute of Mathematical Sciences, Fall 2010


WARNING
Chubb
ARMED
TEL

3 Components

- Pynac for symbolic expressions

```
var('t')
f(t) = t*tan(t^3)
f

var('s')
f(s)
```

- Maxima for derivatives, antiderivatives

```
fprime(t) = f.derivative()
fprime
```

- matplotlib for 2-D graphics

```
fprime.plot()
```

- GAP (Groups, Algorithms, Programming) for group theory

```
G = DihedralGroup(10)
G.center()
```

- BLAS, ATLAS, LAPACK, NumPy, SciPy for linear algebra

```
A = matrix(QQ,[[ 3,  2, 0,  1],
               [ 2,  1, 1,  0],
               [ 1, -1, 5, -3],
               [-2, -3, 5, -4]])
A.echelon_form()

K = A.kernel()
K

K.dimension()

A.eigenvalues()

B = A.change_ring(RDF)
B.eigenvalues()
```

- JMOL for 3D graphics

```
var('x y')
plot3d(x^2+3*y^2, (x, -2, 2), (y, -2, 2))

var('x y z')
T = RDF(golden_ratio)
p = 2 - (cos(x + T*y) + cos(x - T*y) + cos(y + T*z) + cos(y - T*z) + cos(z - T
r = 4.77
implicit_plot3d(p, (x, -r, r), (y, -r, r), (z, -r, r), plot_points=40).show()
```

- R and GSL for probability and statistics

```
sage: data = [1.2, 3.6, 9.8, 5.4, 7.6]
sage: R_data = r.c([1.2, 3.6, 9.8, 5.4, 7.6])
sage: R_data.parent()

sage: R_data.summary()

sage: python_data = R_data.sage()
sage: python_data

sage: sigma = 2
sage: T = RealDistribution('gaussian', sigma)

sage: T.get_random_element()

sage: T.cum_distribution_function(1)

sage: T.cum_distribution_function_inv(.95)
```

3.1 Other Packages

- Singular - commutative and non-commutative algebra, algebraic geometry, and singularity theory
- Symmetrica - representation theory
- SciPy - optimization, numerical integration, Fourier transforms, signal processing, ODE solvers
- 80 others...number theory, finite fields, number fields, cryptography

3.2 The Glue

- Python is the development language
- Python is the user language
- There is no such thing as a kernel
- *Any* Python package may be simply `import`'ed

```
import scipy
scipy.version.version

scipy.finfo('float128').tiny
```

4 History and Development

- Started in 2005 by William Stein (U of Washington), as reaction to proprietary systems
- 95 packages; 500,000 lines of new code
- Funding: US National Science Foundation, Google, Microsoft, US Department of Defense, ANR, CNRS, academic institutions, and others
- 273 developers, about 50 contribute to any one release
- Regular releases, very stable intermediate releases
- Every change is peer-reviewed, a single "release manager"
- Linux, OSX; Windows port in-progress
- Trivial to build from source, minimal prerequisites
- sage-devel, sage-support Google groups: 3000 members, 1500 messages/-month
- 31 Sage Days Workshops, 3 Sage Educational Days

5 Sage Notebook, a Digital Blackboard

- Web interface, local or remote server
- Every copy of Sage contains a server
- Worksheet management
- Online tab-completion, help, source

```
A = matrix(QQ,[[ 3,  2, 0,  1],
               [ 2,  1, 1,  0],
               [ 1, -1, 5, -3],
               [-2, -3, 5, -4]])
```

- Tab-completion for methods

```
A.
```

- One question mark brings help, with *tested* examples

```
A.jordan_form?
```

- Two question marks brings source code

```
A.jordan_form??
```

- Source code display

```
search_def('jordan')
```

- Low barriers to development
- Revision control log (Interrupt to stop serving)

```
hg_sage.browse()
```

- LaTeXintegration (use typeset button)

```
var('x')
f = x^3*e^-x
f.integrate(x)
```

- More LaTeXintegration

```
latex(A)
```

- Integrated mini-editor, supports LaTeXsnippets (shift-click above)
- Interactive demonstrations, explorations

```
import pylab
A_image = pylab.mean(pylab.imread(DATA + 'mystery.png'), 2)
@interact
def svd_image(i=(1,(1..50)), display_axes=True):
    u,s,v = pylab.linalg.svd(A_image)
    A     = sum(s[j]*pylab.outer(u[0:,j], v[j,0:]) for j in range(i))
    show(matrix_plot(A), axes=display_axes, figsize=(11,5))
    html('<h2>Compressed using %s singular values</h2>'%i)
```

6 Fast

- Exact determinant of a 500×500 integer matrix

```
A = random_matrix(ZZ, 500)
d = A.determinant()
d
```

```
N(log(abs(d), 10), digits=5)
```

- Precise timings

```
timeit("A.determinant()", number=2, repeat=3)
```

7 Easy

- 4×4 matrix entries

```
entries = [[1, -2, -4, 4], [1, -2, -3, 4], [0, -2, -6, 6], [-1, 2, 1, -4]]
```

- Over the integers

```
A = matrix(ZZ, entries)
A.echelon_form()
```

- Over the rationals

```
A = matrix(QQ, entries)
A.echelon_form()
```

- With floating point double-precision reals

```
A = matrix(RDF, entries)
P, L, U = A.LU()
print L.round(5)
print
print U.round(5)
```

- With elements of a finite field

```
F.<a> = FiniteField(3^2)
F.list()
```

```
A = random_matrix(F, 4, 4)
A
```

```
A.characteristic_polynomial('T')
```

8 All-In-One

- Seamlessly move from one area of mathematics to another

```
G = graphs.CirculantGraph(12, [1, 5])
G.plot()

G.adjacency_matrix()

G.spectrum()

A = G.automorphism_group(); A

A.order()
```

9 Versatile

- Drill down into included libraries
- SciPy Python bindings to FORTRAN LAPACK
- Example by Jason Grout

```
A = matrix(RDF, [[10,2,3],[3,4,6],[2,8,10]])

import scipy
import scipy.linalg
lu, piv, success = scipy.linalg.flapack.dgetrf(A.rows())

# Break apart combined L and U
rows, cols = lu.shape
L = copy(matrix(RDF,rows,cols,1))
U = matrix(RDF,rows,cols)
for i in range(rows):
    for j in range(cols):
        if i>j:
            L[i,j]=lu[i,j]
        else:
            U[i,j]=lu[i,j]

# Construct permutation matrix from permutation
```

```
P = copy(identity_matrix(rows))
for i,j in enumerate(piv):
    P.swap_rows(i,j)

print "\nP: \n", P
print "\nL: \n", L
print "\nU: \n", U
print "\nVerification: \n", P*A - L*U
```

10 Convenient

- Install on any computer (don't need to be root)
- Access from any web browser, including smart phones
- Command line or notebook
- Batch or interactive
- Dedicated applications, such as Android application
- Novel front-ends – one-shot cell phone at `http://math3.skku.ac.kr/wap_html`
- Some folks install Sage *only* to get GAP, R, PARI, SciPy, etc.

11 The Conifold: from "Toric Geometry and Sage" Arnold Sommerfeld Center for Theoretical Physics, Munich, April 26-29, 2011 By Volker Braun, Dublin Institute for Advanced Studies

A singularity that occurs very often is the conifold, which is the simplest non-quotient singularity. In terms of toric geometry, it is defined by the non-simplicial cone over a minimal lattice square at distance 1:

```
# do not evaluate, depends on unofficial code
```

?? ?? ?? ??The conifold is not smooth because the hypersurface equation is not transverse at $\bar{z} = (0, 0, 0, 0)$. More precisely, the singularities are the

variety of the Jacobian ideal

$$\mathrm{Jac}(f) = \left\langle f,\ \frac{\partial f}{\partial z_0},\ \ldots,\ \frac{\partial f}{\partial z_3} \right\rangle \tag{1}$$

For the conifold, it is **?? ??**The most basic invariant of an isolated hypersurface singularity is its *Milnor number*, which is the vector space dimension of $\mathbb{C}[\bar{x}]/\mathrm{Jac}(f(\bar{x}))$. For the conifold, it is one: **??**In fact, the converse is also true: A 3-dimensional isolated singularity of Milnor number one is a conifold. Note, however, that higher Milnor numbers no longer uniquely determine the singularity.

Exercise 9. *Use Sage to compute the Milnor number of the singularity* $\mathbb{C}^2/\mathbb{Z}_n$ *for* $n \in \{2, 3, \ldots, 10\}$.

12 Parallel Processing

- Parallel "decorator" converts function to an iterator
- This example uses the Python multiprocessing library, can also fork, or...
- 2 cores, 4 threads

```
@parallel('multiprocessing', 4)
def summation(multiple):
    sum = 0
    for i in srange(0, 1000000*multiple, multiple):
        sum = sum + i
    print('Finished {0}'.format(multiple))
    return sum
```

- Now pass in a *list* of inputs to create the "jobs."

```
summer = summation([20, 5, 100, 57])
```

- Ask for a list of the jobs (inputs and outputs, in new order).

```
list(summer)
```

- A larger list of inputs. Watch in htop.

```
long_summer = summation(range(100))

list(long_summer)
```

13 Cython

- Convert Python to compiled C
- Originally Pyrex, forked by Sage to Cython

```
def summation_python():
    sum = 0
    for i in range(1000000):
        sum = sum + i
    return sum

timeit("summation_python()")
```

- Declare integer variables
- Modify the for-loop

```
%cython
def summation_cython():
    cdef i, sum
    sum = 0
    for 0 <= i < 1000000:
        sum = sum + i
    return sum

timeit("summation_cython()")
```

- Speedups by factors of hundreds and thousands are not uncommon
- f2c (Fortran to C) is also included

14 SageTeX

- Include Sage instructions in a LaTeX document
- SageTeX isolates the commands, evaluates them, pastes in results
- Also creates and inserts plots
- Formats Sage code

3.4 The Conifold

A singularity that occurs very often is the conifold, which is the simplest non-quotient singularity. In terms of toric geometry, it is defined by the non-simplicial cone over a minimal lattice square at distance 1:

```
sage: conifold = toric_varieties.Conifold()
sage: conifold.is_smooth()
False
sage: conifold.is_orbifold()
False
sage: square_cone = conifold.fan().generating_cone(0)
sage: square_cone.rays()
(N(0, 0, 1), N(0, 1, -1), N(1, 0, 1), N(1, 1, 1))
sage: patch = conifold.affine_algebraic_patch(square_cone)
sage: patch
Closed subscheme of Affine Space of dimension 4 over Rational
    Field defined by:
  z0*z2 - z1*z3
```

The conifold is not smooth because the hypersurface equation is not transverse at $\vec{z} = (0,0,0,0)$. More precisely, the singularities are the variety of the Jacobian ideal

$$\mathrm{Jac}(f) = \left\langle f, \frac{\partial f}{\partial z_0}, \ \ldots, \ \frac{\partial f}{\partial z_3} \right\rangle \tag{37}$$

For the conifold, it is

```
sage: Jac = patch.Jacobian()
sage: Jac
Ideal (z0*z2 - z1*z3, z2, -z3, z0, -z1) of Multivariate Polynomial
    Ring in z0, z1, z2, z3 over Rational Field
sage: A4 = patch.ambient_space()
sage: origin = A4.subscheme(A4.gens())  # the origin (0,0,0,0)
sage: A4.subscheme(Jac) == origin
True
```

The most basic invariant of an isolated hypersurface singularity is its *Milnor number*, which is the vector space dimension of $\mathbb{C}[\vec{x}]/\mathrm{Jac}(f(\vec{x}))$. For the conifold, it is one:

```
sage: Jac.vector_space_dimension()
1
```

In fact, the converse is also true: A 3-dimensional isolated singularity of Milnor number one is a conifold. Note, however, that higher Milnor numbers no longer uniquely determine the singularity.

Exercise 9. *Use Sage to compute the Milnor number of the singularity* $\mathbb{C}^2/\mathbb{Z}_n$ *for* $n \in \{2, 3, \ldots, 10\}$.

15 Philosophy

Mathematica: "Why You Do Not Usually Need to Know about Internals" In the online Mathematica Documentation Center.

> You should realize at the outset that while knowing about the internals of Mathematica may be of intellectual interest, it is usually much less important in practice than you might at first suppose.
>
> . . .
>
> Indeed, in almost all practical uses of Mathematica, issues about how Mathematica works inside turn out to be largely irrelevant.
>
> . . .
>
> Particularly in more advanced applications of Mathematica, it may sometimes seem worthwhile to try to analyze internal algorithms in order to predict which way of doing a given computation will be the most efficient. And there are indeed occasionally

> major improvements that you will be able to make in specific computations as a result of such analyses.
>
> But most often the analyses will not be worthwhile. For *the internals of Mathematica are quite complicated* [emphasis added], and even given a basic description of the algorithm used for a particular purpose, it is usually extremely difficult to reach a reliable conclusion about how the detailed implementation of this algorithm will actually behave in particular circumstances.

J. Neubüser (Founder of GAP): "An invitation to computational group theory."
In C. M. Campbell, T. C. Hurley, E. F. Robertson, S. J. Tobin, and J. J. Ward, editors, Groups 93 Galway/St. Andrews, Volume 2, volume 212 of London Mathematical Society Lecture Note Series, pages 457-475. Cambridge University Press, 1995.

> You can read Sylow's Theorem and its proof in Huppert's book in the library without even buying the book and then you can use Sylow's Theorem for the rest of your life free of charge, but... for many computer algebra systems license fees have to be paid regularly for the total time of their use. In order to protect what you pay for, you do not get the source, but only an executable, i.e. a black box. You can press buttons and you get answers in the same way as you get the bright pictures from your television set but you cannot control how they were made in either case.
>
> With this situation two of the most basic rules of conduct in mathematics are violated: In mathematics information is passed on free of charge and everything is laid open for checking. Not applying these rules to computer algebra systems that are made for mathematical research... means moving in a most undesirable direction. Most important: Can we expect somebody to believe a result of a program that he is not allowed to see? Moreover: Do we really want to charge colleagues in Moldava several years of their salary for a computer algebra system?

- Available at `http://buzzard.ups.edu/talks.html`