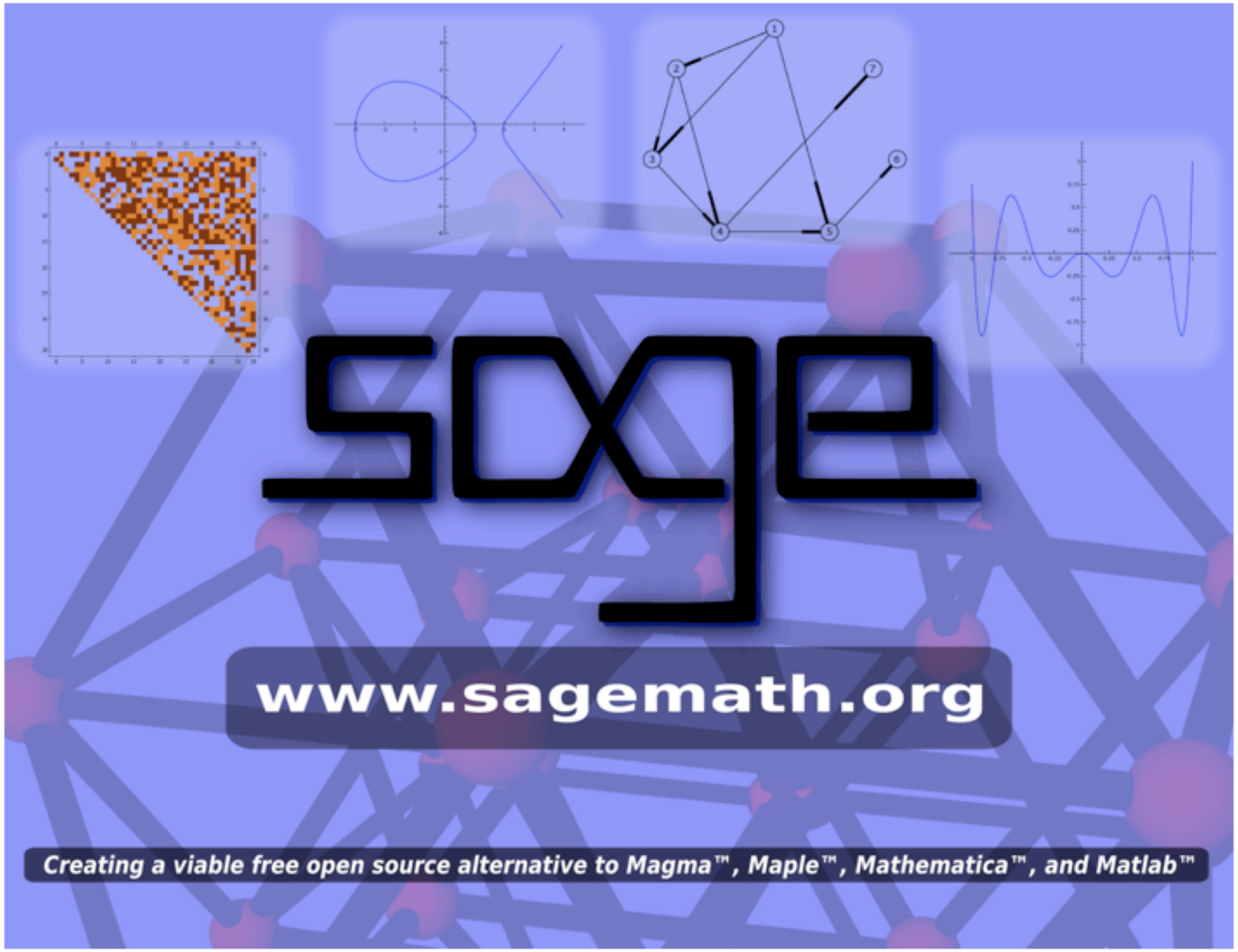
Sage

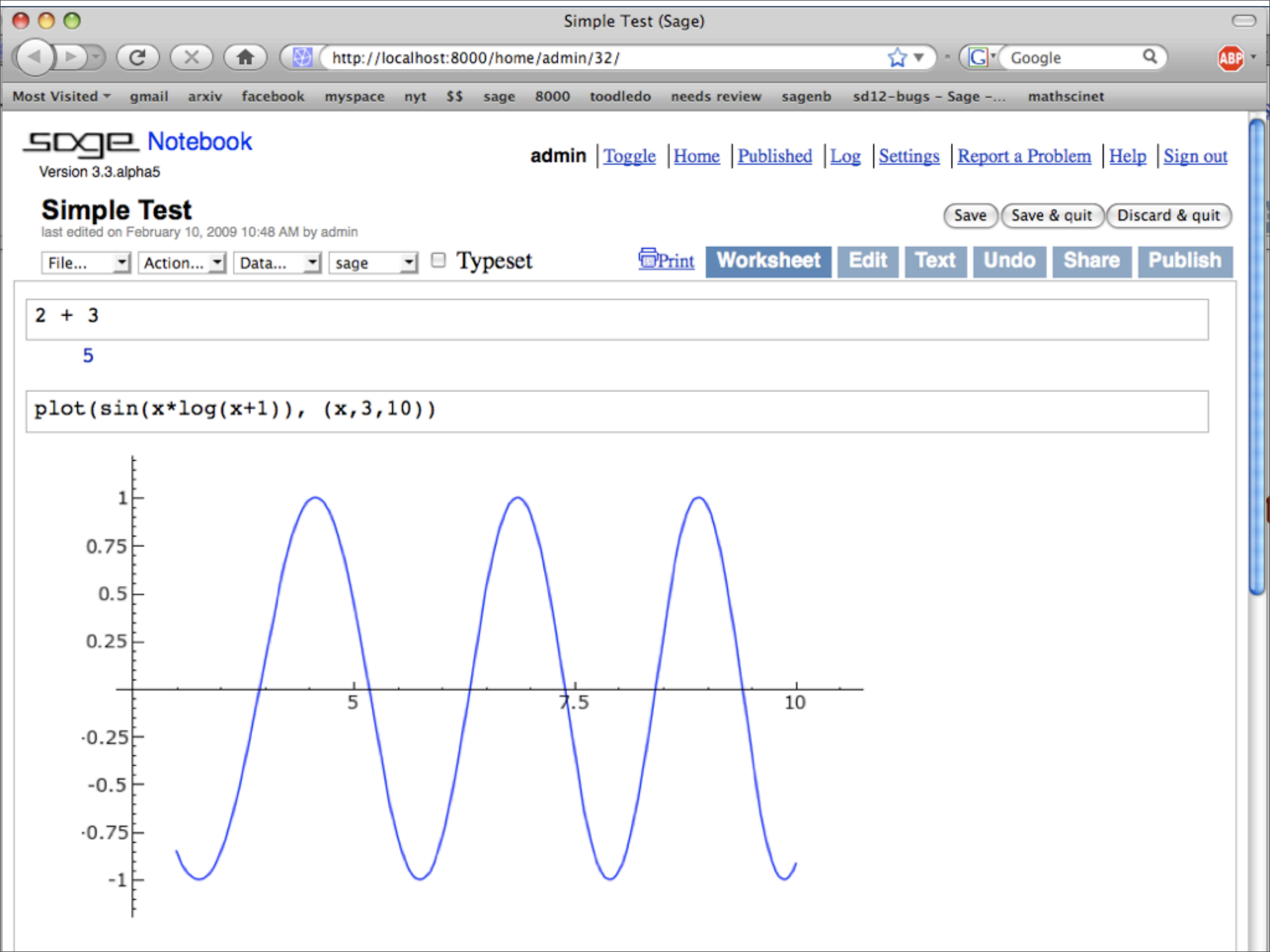
Creating a viable free open source alternative to Magma, Maple, Mathematica and Matlab

William Stein, Associate Professor, University of Washington



History

- *I started Sage* at Harvard in ***January 2005***.
- No existing math software ***good enough***.
- I got ***very annoyed*** that my students and colleagues had to pay a ***ridiculous amount*** to use the code I wrote in Ma*'s.
- Sage-1.0 released ***February 2006*** at Sage Days 1 (San Diego).
- ***Sage Days Workshops*** 1, 2, ..., 12, at UCLA, UW, Cambridge, Bristol, Austin, France, San Diego, Seattle, etc.
- Sage ***won first prize*** in Trophees du Libre (November 2007)
- Funding from ***Microsoft, UW, NSF, DoD, Google, Sun,*** private donations, etc.


Simple Test (Sage)
http://localhost:8000/home/admin/32/
Google
Most Visited gmail arxiv facebook myspace nyt $$ sage 8000 toodledo needs review sagenb sd12-bugs - Sage -... mathscinet
Notebook
Version 3.3.alpha5
admin | Toggle | Home | Published | Log | Settings | Report a Problem | Help | Sign out
Simple Test
last edited on February 10, 2009 10:48 AM by admin
Save
Save & quit
Discard & quit
File...
Action...
Data...
sage
Typeset
Print
Worksheet
Edit
Text
Undo
Share
Publish
2 + 3
5
plot(sin(x*log(x+1)), (x,3,10))
1
0.75
0.5
0.25
5
7.5
10
-0.25
-0.5
-0.75
-1

Sage-support -- 1081 members (join today!)

Discussions 8 of 11344 messages view all »

Installation problems
By MR - 7:35am - 5 authors - 9 replies

Sage Days 16 : Barcelona
By Enrique Gonzalez Jimenez - 6:22am - 1 author - 0 replies

[sage-support] Re: Installation problem
By William Stein - 5:53am - 2 authors - 3 replies

[sage-support] Re: range endpoints
By William Stein - 5:52am - 3 authors - 4 replies

[sage-support] Re: Groebner Basis question
By Martin Albrecht - 5:08am - 3 authors - 4 replies

Get the number of variables of multivariate polynomial
By domingo.domingogomez@gmail.com - 2:06am - 3 authors - 6 replies

[sage-support] Re: Newbie question. How do you get Sage to show a plot?
By William Stein - Mar 31 - 5 authors - 6 replies

[sage-support] Sage for Gentoo Linux
By Alex Ghitza - Mar 31 - 2 authors - 1 reply

Members 1081 members view all »

1eviaw_1...@CO.GUADALUPE.TX.US
Member

a.conn...@galileo.karoo.co.uk
Member

Sage is a free open-source mathematics software system licensed under the GPL. It combines the power of many existing open-source packages into a common Python-based interface.

Mission: *Creating a viable free open source alternative to Magma, Maple, Mathematica and Matlab.*

Donate · Acknowledgments · DevMap · Browse the Code · Questions?

new features since 3.2.3 · become a fan of Sage

Download 3.4
Binary · Source · Packages

Sage Via the Web
Milnix.org · KAIST · [test version]

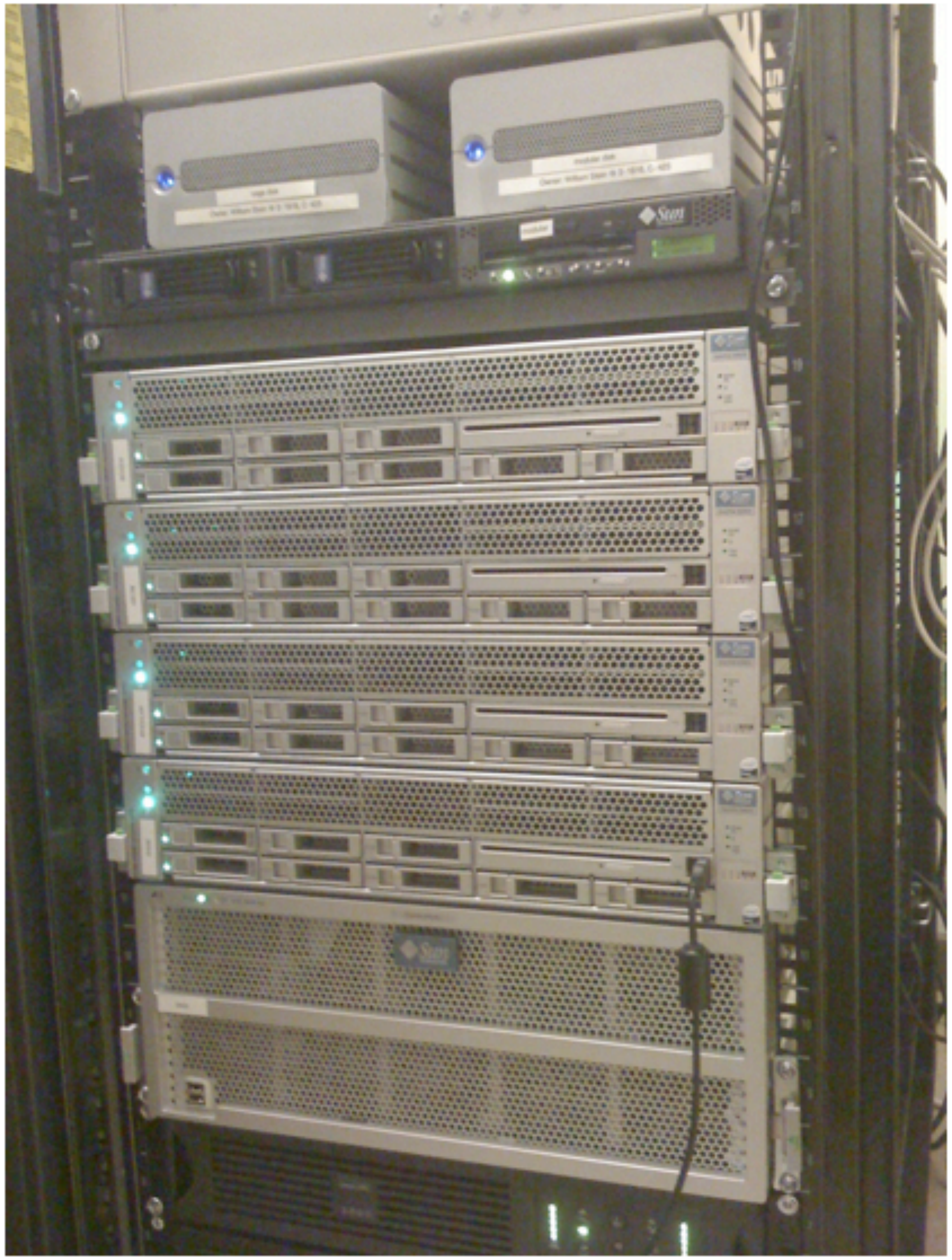
Help
Documentation · Support · Tutorial

Feature Tour
Quickstart · Research · Graphics

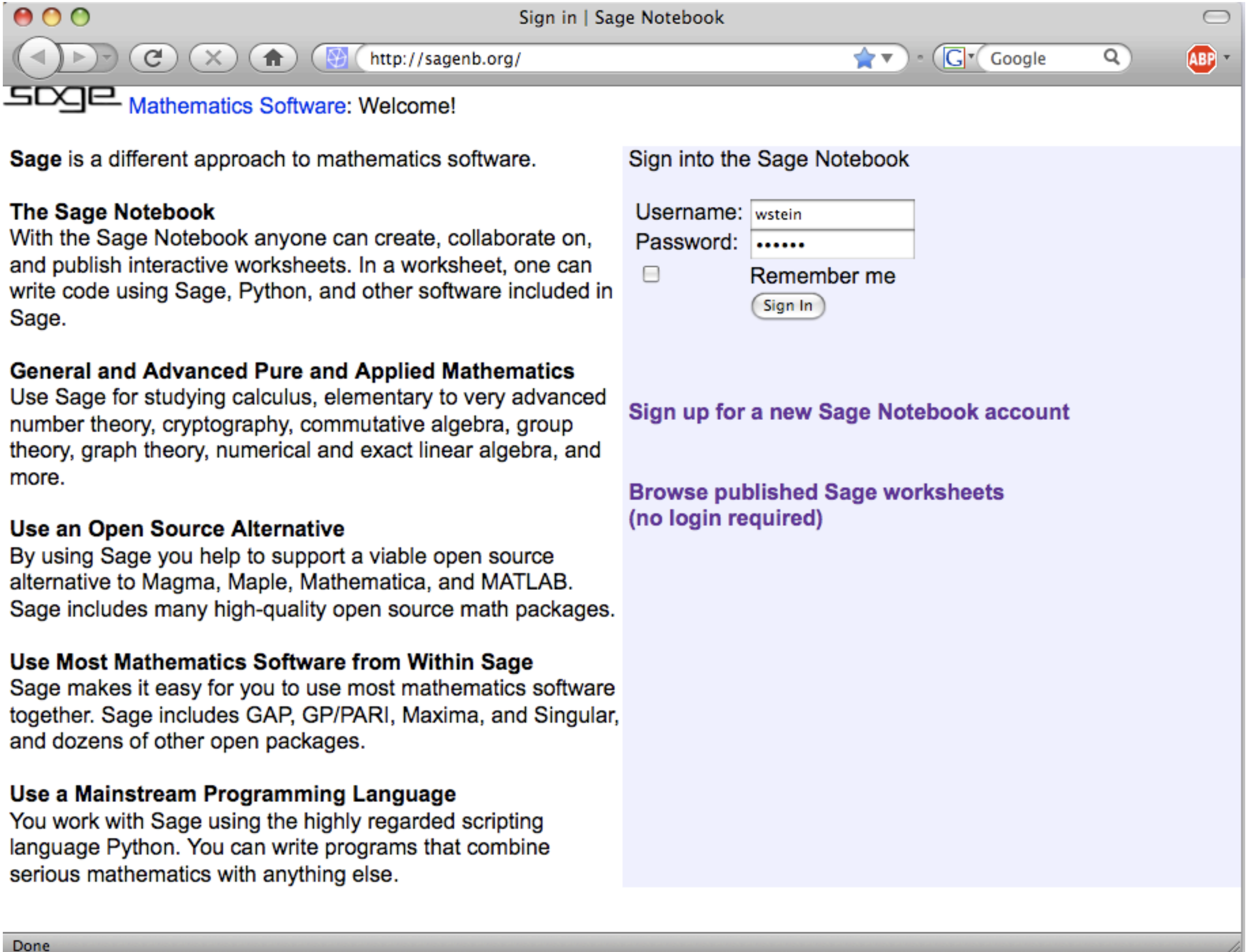
Library
Testimonials · Books · Publications

Search

Random Link: *Success Stories, Testimonials, News Articles, Blogs*



Sage Devel Headquarters: Four 24-core Sun X4450's with 128GB RAM each + 1 Sun X4540 with 24TB disk. Purchased in Nov 2008 using $110K NSF SCREMS Grant.


Sign in | Sage Notebook

http://sagenb.org/

Google

sage Mathematics Software: Welcome!

Sage is a different approach to mathematics software.

The Sage Notebook
With the Sage Notebook anyone can create, collaborate on, and publish interactive worksheets. In a worksheet, one can write code using Sage, Python, and other software included in Sage.

General and Advanced Pure and Applied Mathematics
Use Sage for studying calculus, elementary to very advanced number theory, cryptography, commutative algebra, group theory, graph theory, numerical and exact linear algebra, and more.

Use an Open Source Alternative
By using Sage you help to support a viable open source alternative to Magma, Maple, Mathematica, and MATLAB. Sage includes many high-quality open source math packages.

Use Most Mathematics Software from Within Sage
Sage makes it easy for you to use most mathematics software together. Sage includes GAP, GP/PARI, Maxima, and Singular, and dozens of other open packages.

Use a Mainstream Programming Language
You work with Sage using the highly regarded scripting language Python. You can write programs that combine serious mathematics with anything else.

Sign into the Sage Notebook

Username: wstein
Password: ••••••
Remember me
Sign In

Sign up for a new Sage Notebook account

Browse published Sage worksheets
(no login required)

Done

Unique Advantages of Sage

1. Python-- ***a general purpose*** language at core of Sage; huge user base compared to Matlab, Mathematica, Maple and Magma
2. Cython -- ***write blazingly fast*** compiled code in Sage
3. Free and ***Open source***
4. Excellent ***Peer review*** of Code: "*I do really and truly believe that the Sage refereeing model results in better code*." -- John Cremona

```
email('wstein@gmail.com', 'The calculation finished!')
```

```
Child process 50883 is sending email to wstein@gmail.com...
```

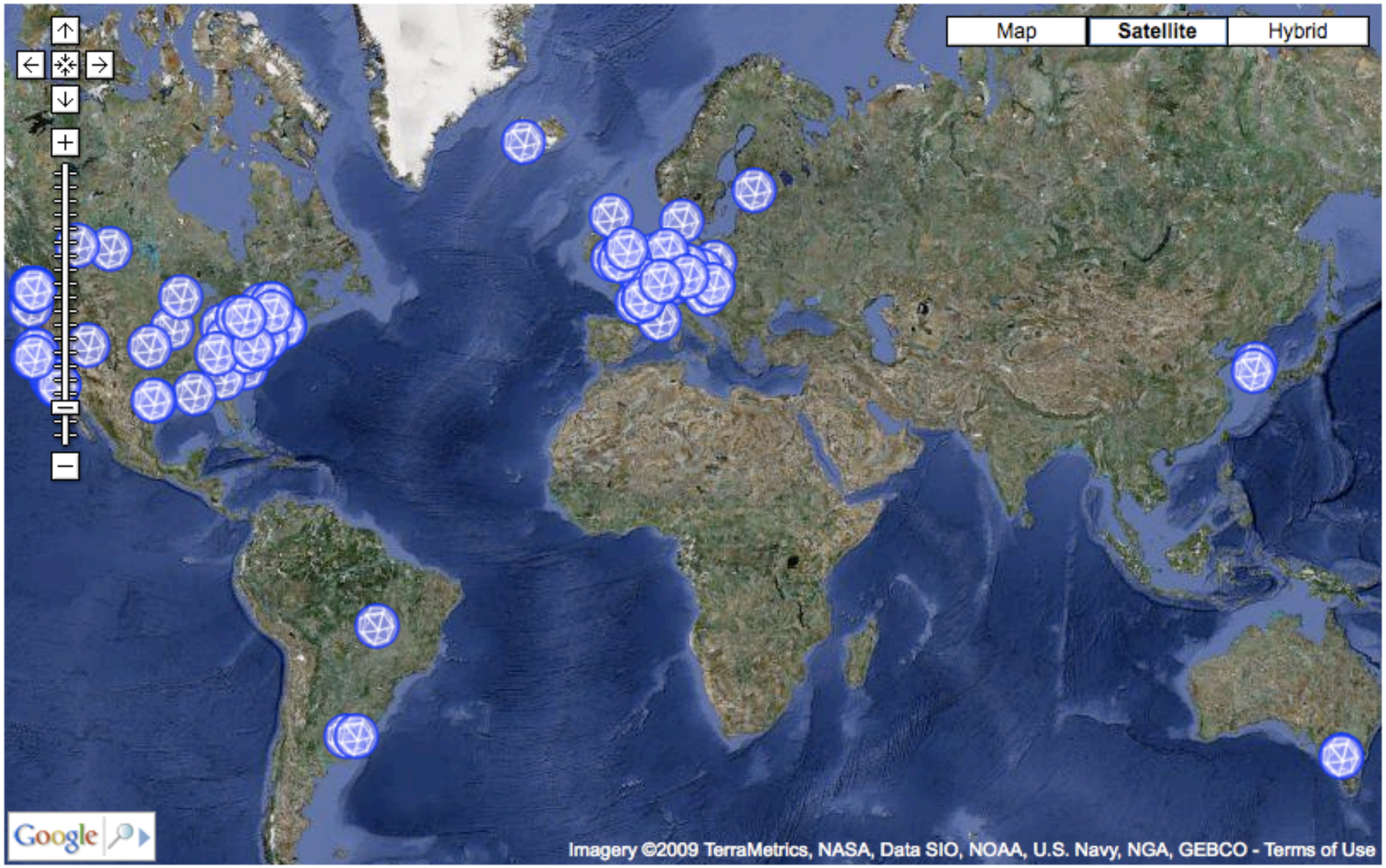
Sage Is...

- Over ***300,000 lines*** of new Python/Cython code
- ***Distribution*** of mathematical software; easy to build from source (over 5 million lines of code).
- About ***150 developers***: developer map
- Exact and numerical ***linear algebra*****,** optimization, ***statistics*** (including R), group theory, combinatorics, cryptography.
- **Number Theory:** elliptic curves, modular forms, *L*-functions -- this is *my research area*
- ***Calculus***
- 2d and 3d ***plotting***
- ***Range of functionality rivals*** the Ma*'s

Sage developers around the world

This is a map of all contributors to the Sage project. There are currently 142 contributors in 85 different places from all around the world.

Map Zoom: Earth - USA (UW, West, East) - Europe - Asia - S. America - Australia



William Stein Tim Abbott Michael Abshoff Martin Albrecht Nick Alexander Maite Aranes Jennifer Balakrishnan Jason Bandlow Arnaud Bergeron Francois Bissey Jonathan Bober Tom Boothby Nicolas Borie Robert Bradshaw Michael Brickenstein Dan Bump Iftikhar Burhanuddin Ondrej Certik Wilson Cheung Craig Citro Francis Clarke Timothy Clemans Alex Clemesha John Cremona Karl-Dieter Crisman Doug Cutrell Tom Denton Didier Deshommes Dan Drake Alexander Dreyer Gabriel Ebner Burcin Erocal Gary Furnish Alex Ghitza Andrzej Giniewicz Amy Glen Daniel Gordon Chris Gorecki Jason Grout Carlo Hamalainen Marshall Hampton Jon Hanke Mike Hansen Bill Hart David Harvey Neal Holtz Sean Howe Alexander Hupfer Wilfried Huss Naqi Jaffery Peter Jipsen David Joyner Michael Kallweit Josh Kantor Kiran Kedlaya Simon King Emily Kirkman David Kohel Ted Kosan Sébastien Labbé Yann Laigle-Chapuy Kwankyu Lee David Loeffler Michael Mardaus Jason Martin Jason Merrill Matthias Meulien Robert Miller Kate Minola Joel Mohler Bobby Moretti Guillaume Moroz Gregg Musiker Tobias Nagel Brett Nakashima Pablo De Nápoli Minh Van Nguyen Andrey Novoseltsev Ronan Paixão Willem Jan Palenstijn John Palmieri David Perkinson Clement Pernet John Perry Pearu Peterson Yi Qiang Dorian Raymer R. Rishikesh David Roe Bjarke Hammersholt Roune Franco Saliola Kyle Schalm Anne Schilling Harald Schilly Jack Schmidt Dan Shumow Steven Sivek Nils-Peter Skoruppa Jaap Spies Blair Sutton Chris Swierczewski Philippe Theveny Nicolas Thiery Igor Tolkov Gonzalo Tornaria John Voight Justin Walker Mark Watkins Georg S. Weber Joe Wetherell Carl Witty Cristian Wuthrich Dal S. Yu Mike Zabrocki Bin Zhang Paul Zimmermann

Examples

Symbolic expressions:

```
x, y = var('x,y')
type(x)
```

```
<class 'sage.calculus.calculus.SymbolicVariable'>
```

```
a = 1 + sqrt(2) + pi + 2/3 + x^y
```

```
show(a)
```

$$x^y + \pi + \sqrt{2} + \frac{5}{3}$$

```
show(expand(a^2))
```

$$x^{2y} + 2\pi x^y + 2\sqrt{2}x^y + \frac{10x^y}{3} + \pi^2 + 2\sqrt{2}\pi + \frac{10\pi}{3} + \frac{10\sqrt{2}}{3} + \frac{43}{9}$$

Solve equations

```
var('a,b,c,x')
show(solve(x^2 + sqrt(17)*a*x + b == 0, x))
```

$$\left[x = \frac{-\left(\sqrt{17a^2-4b}\right)-\sqrt{17}a}{2},\; x = \frac{\sqrt{17a^2-4b}-\sqrt{17}a}{2}\right]$$

```
var('a,b,c,x')
show(solve(a*x^3 + b*x + c == 0, x)[0])
```

$$x = \left(\frac{-\sqrt{3}i}{2} - \frac{1}{2}\right)\left(\frac{\sqrt{\frac{27ac^2+4b^3}{a}}}{6\sqrt{3}a} - \frac{c}{2a}\right)^{\frac{1}{3}} - \frac{\left(\frac{\sqrt{3}i}{2} - \frac{1}{2}\right)b}{3a\left(\frac{\sqrt{\frac{27ac^2+4b^3}{a}}}{6\sqrt{3}a} - \frac{c}{2a}\right)^{\frac{1}{3}}}$$

```
A = random_matrix(QQ, 500); v = random_matrix(QQ,500,1)
time x = A \ v
```

```
Time: CPU 1.69 s, Wall: 2.20 s
```

```
len(str(x[0]))
```

```
1486
```

Sage is Better than Matlab at Fractions

In Octave:

```
octave:1> format rat;
octave:2> a = [-86/17,40/29,-68/43,-20/11;-24/17,-1/38,-2/25,49/17]
a =
    -86/17      40/29     -68/43     -20/11
    -24/17      -1/38      -2/25      49/17
octave:3> rref(a)
ans =
         1          0   155/2122   -725/384
         0          1   -152/173  -6553/795
```

and in Matlab:

```
>> format rat;
>> a = [-86/17,40/29,-68/43,-20/11;-24/17,-1/38,-2/25,49/17]
a =
    -86/17           40/29           -68/43           -20/11
    -24/17           -1/38            -2/25            49/17
>> rref(a)
ans =
      1               0                13/178          -725/384
      0               1              -152/173         -1426/173
```

```
The truth has little to do with either of the two different outputs above:
```

```
a = matrix(2, [-86/17, 40/29, -68/43, -20/11,   -24/17, -1/38, -2/25, 49/17])
a.echelon_form()
```

```
[                   1                   0    306034/4189705     -404710/214357]
[                   0                   1 -18405604/20948525  -30037214/3644069]
```

```
magma(a).EchelonForm()
```

evaluate

```
[                   1                   0    306034/4189705     -404710/214357]
[                   0                   1 -18405604/20948525  -30037214/3644069]
```

Example: A Huge Integer Determinant

```
a = random_matrix(ZZ,200,x=-2^127,y=2^127)
time d = a.determinant()
len(str(d))
```

```
Time: CPU 3.14 s, Wall: 4.25 s
7786
```

We can also ***copy this matrix*** over to Maple and compute the same determinant there...

```
a[0,0]
```

```
-10495856349494057617178227541504239545
```

```
maple.with_package('LinearAlgebra')
B = maple(a)
t = maple.cputime()
time c = B.Determinant()
maple.cputime(t)
```

evaluate

```
21.9699999999999
```

```
c == d
```

```
True
```

This ability to easily move objects between math software is ***unique to Sage***.

Example: A Symbolic Expression

```
x = var('x')
f(x) = sin(3*x)*x+log(x) + 1/(x+1)^2
show(f)
```

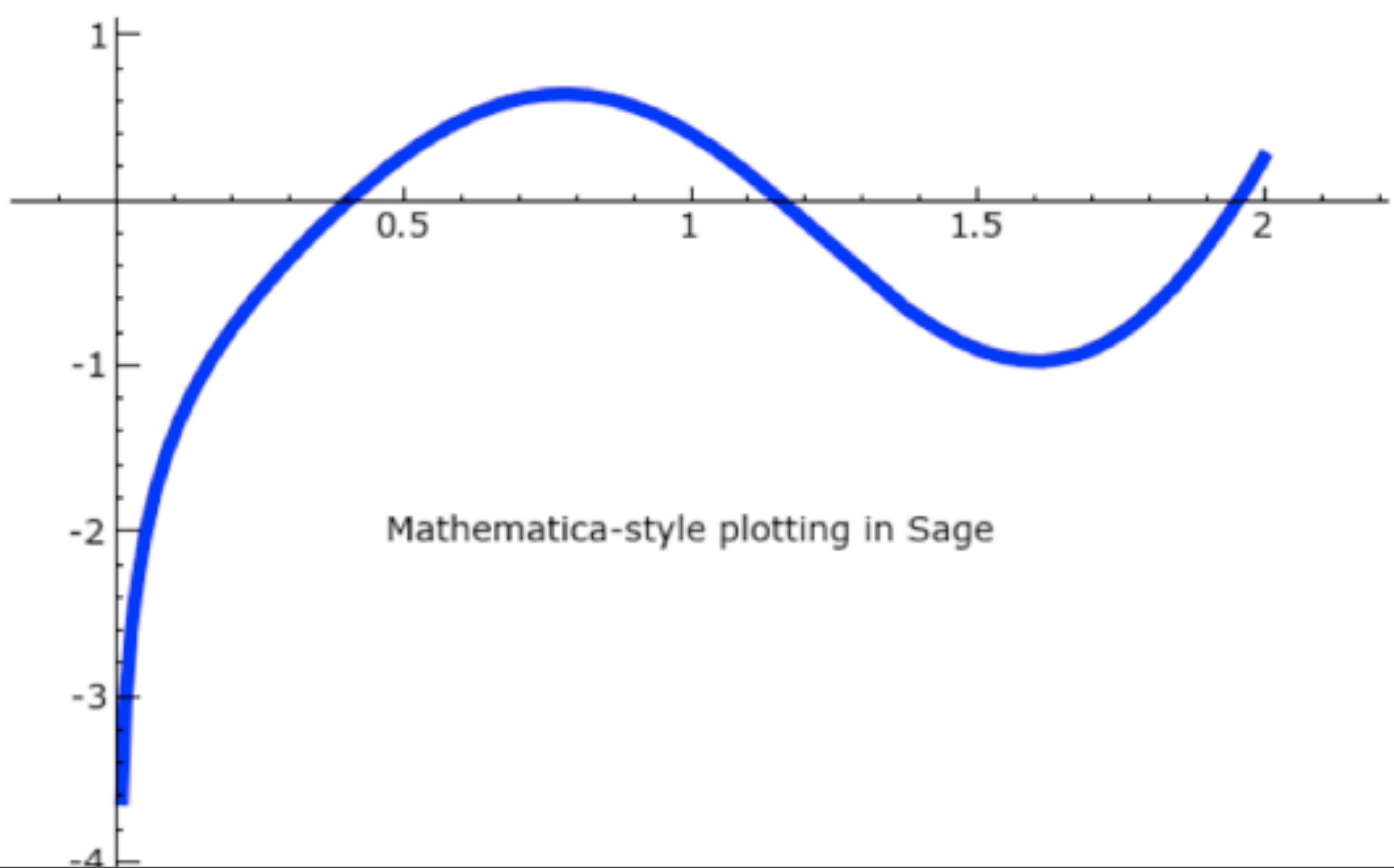
$$x \mapsto x \sin(3x) + \log(x) + \frac{1}{(x+1)^2}$$

Plotting functions has similar syntax to Mathematica:

```
show(f.integrate(x))
```

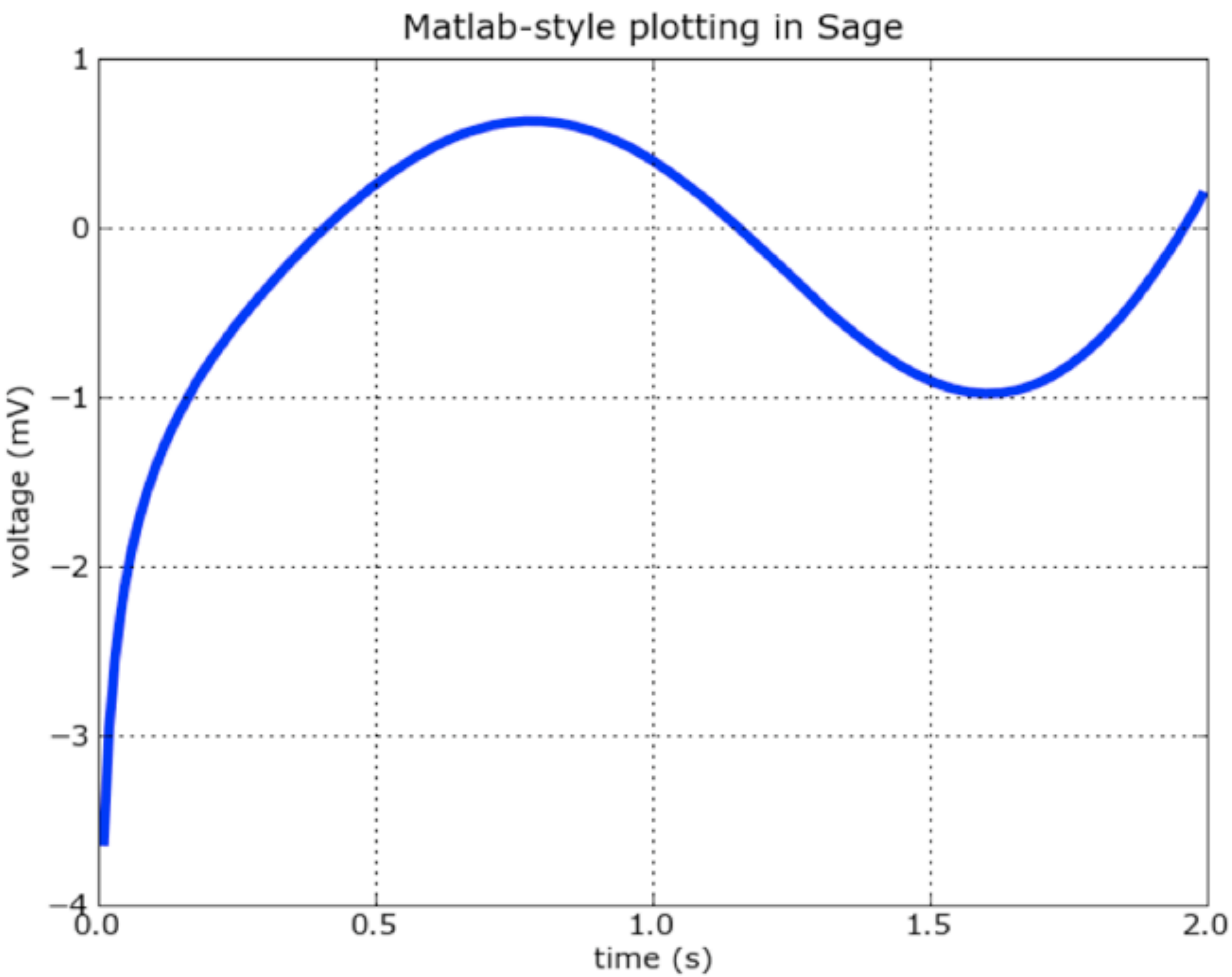
$$x \mapsto \frac{\sin(3x) - 3x\cos(3x)}{9} + x\log(x) - \frac{1}{x+1} - x$$

```
plot(f,(0.01,2), thickness=4) + text("Mathematica-style plotting in Sage", (1,-2), rgbcolor='black')
```



Sage also has 2d plotting that is almost *identical to MATLAB*:

```
import pylab as p
p.figure()
t = p.arange(0.01, 2.0, 0.01)
s = p.sin(2 * p.pi * t)
s = p.array([float(f(x)) for x in t])
P = p.plot(t, s, linewidth=4)
p.xlabel('time (s)'); p.ylabel('voltage (mV)')
p.title('Matlab-style plotting in Sage')
p.grid(True)
p.savefig('sage.png')
```



_fast_float_ yields super-fast evaluation of Sage symbolic expressions -- e.g., here it is *10 times faster* than native Python!

```
f(x,y,z) = sin(3*x)*x + log(x) + 1/(1+x)^2
```

```
g = f._fast_float_(x,y,z)
timeit('g(4.5r,3.2r,5.7r)')
```

```
    625 loops, best of 3: 574 ns per loop
```

```
%python
import math
def g(x): return math.sin(3*x)*x + log(x) + 1/(1+x)**2
```

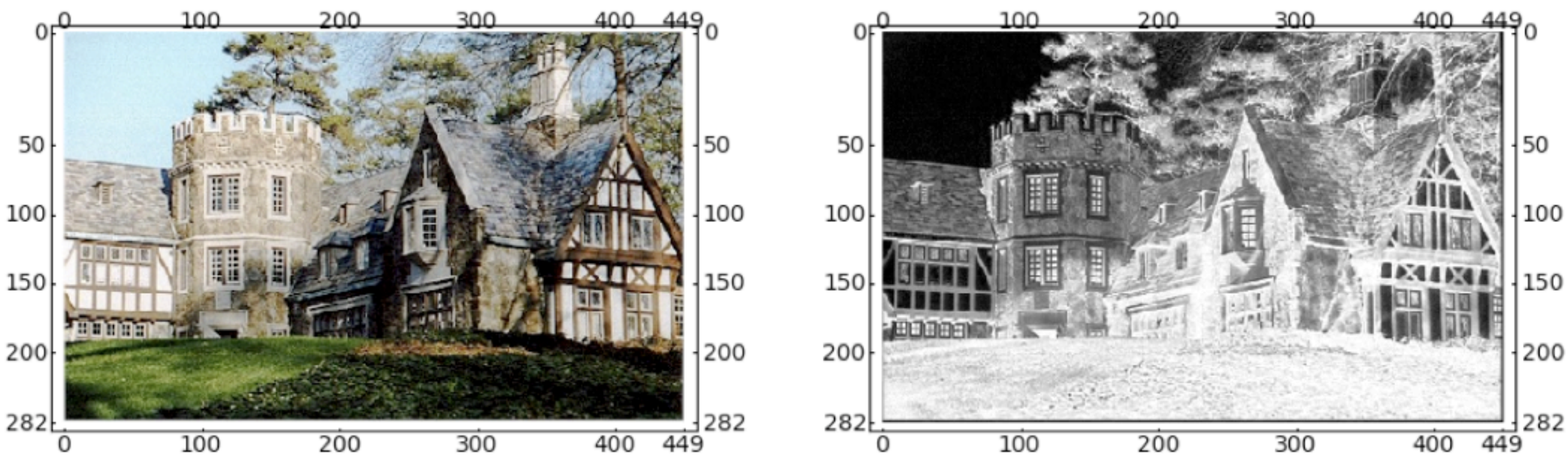
```
timeit('g(4.5r)')
```

```
    625 loops, best of 3: 6.93 µs per loop
```

Example: Interactive Image Compression

This illustrates pylab (matplotlib + numpy), Sage plotting, html output, and @interact.

```
# first just play
import pylab
A = pylab.imread(DATA + 'emoryimage.png')
graphics_array([matrix_plot(A), matrix_plot(1-A[0:,0:,2])]).show(figsize=[10,4])
```




```
A[0,0,]
```

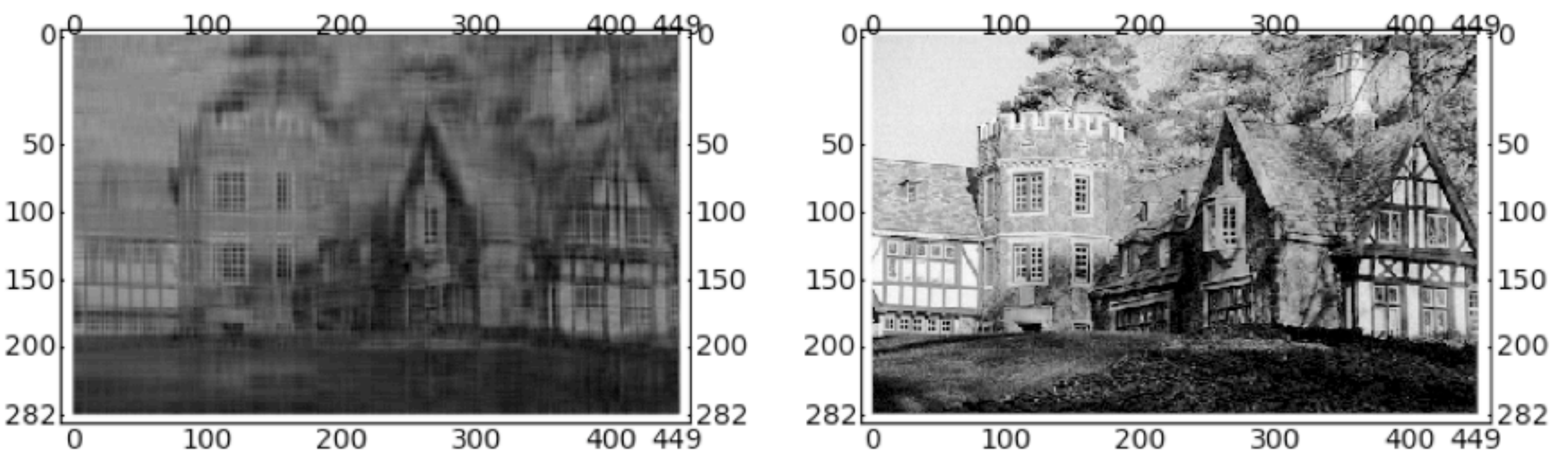
```
array([ 0.76862746,  0.89803922,  0.96470588,  1.        ],
dtype=float32)
```

```
import pylab
A_image = pylab.mean(pylab.imread(DATA + 'emoryimage.png'), 2)
@interact
def svd_image(i=(20,(1..100)), 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))
    g = graphics_array([matrix_plot(A),matrix_plot(A_image)])
    show(g, axes=display_axes, figsize=(8,3))
    html('<h2>Compressed using %s eigenvalues</h2>'%i)
```

i 14

display_axes ☑

Compressed using 14 eigenvalues

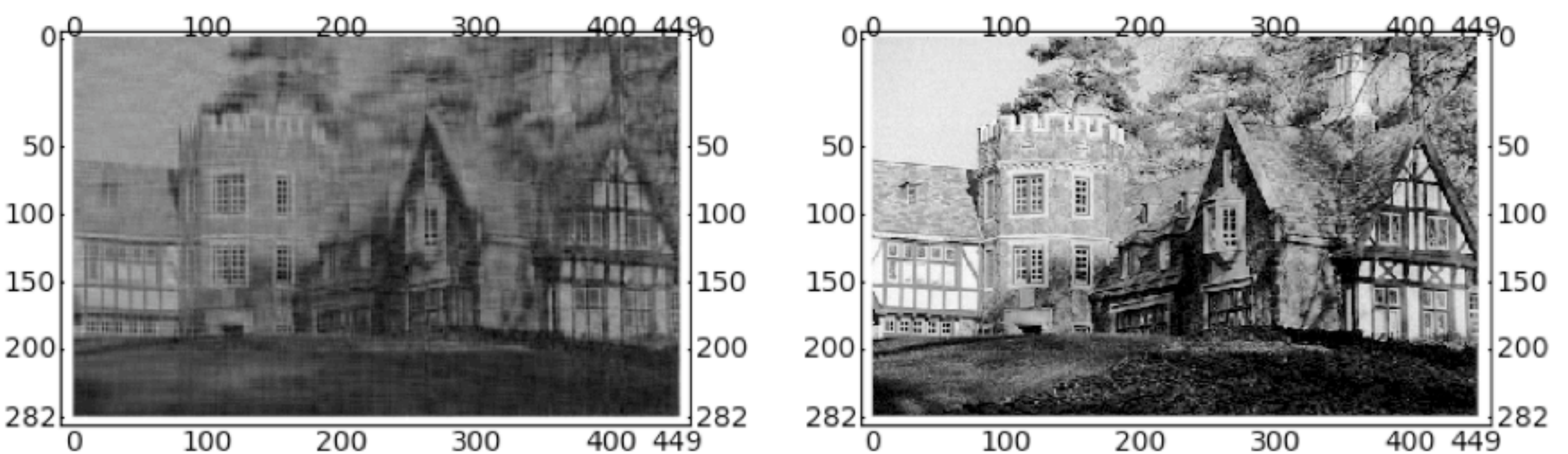


```
import pylab
A_image = pylab.mean(pylab.imread(DATA + 'emoryimage.png'), 2)
@interact
def svd_image(i=(20,(1..100)), display_axes=True):
    u,s,v = pylab.linalg.svd(A_image)
    A     = sum(s[j]*pylab.outer(u[0:,j], v[j,0:]) for j in range(i))
    g = graphics_array([matrix_plot(A),matrix_plot(A_image)])
    show(g, axes=display_axes, figsize=(8,3))
    html('<h2>Compressed using %s eigenvalues</h2>'%i)
```

i 25

display_axes ☑

Compressed using 25 eigenvalues

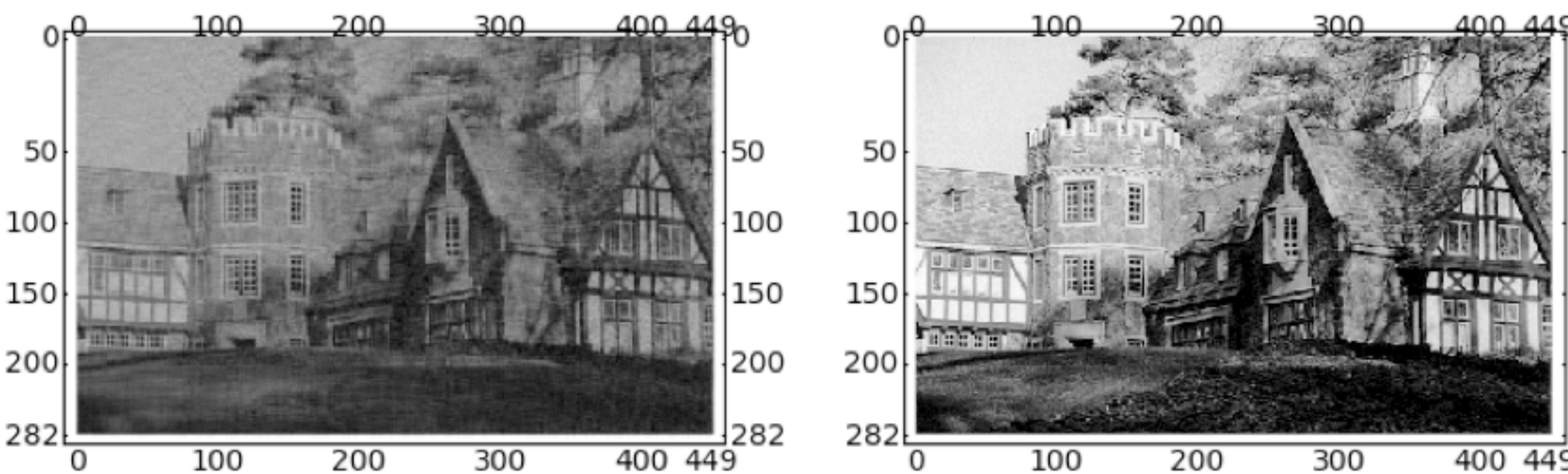


```
import pylab
A_image = pylab.mean(pylab.imread(DATA + 'emoryimage.png'), 2)
@interact
def svd_image(i=(20,(1..100)), display_axes=True):
    u,s,v = pylab.linalg.svd(A_image)
    A     = sum(s[j]*pylab.outer(u[0:,j], v[j,0:]) for j in range(i))
    g = graphics_array([matrix_plot(A),matrix_plot(A_image)])
    show(g, axes=display_axes, figsize=(8,3))
    html('<h2>Compressed using %s eigenvalues</h2>'%i)
```

i 45

display_axes ☑

Compressed using 45 eigenvalues

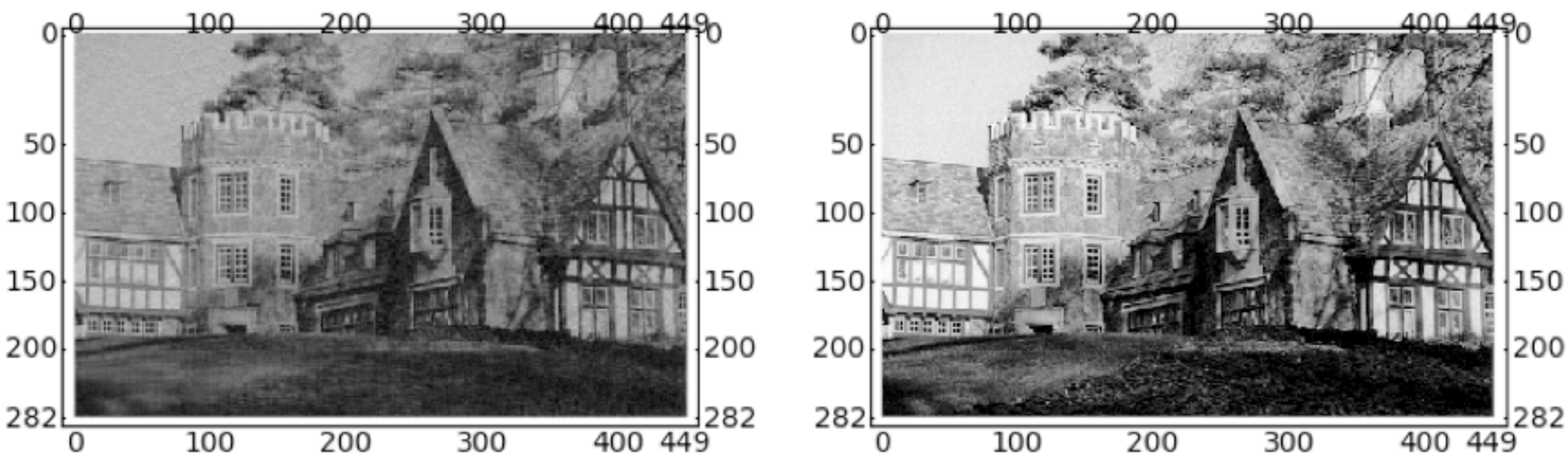


```
import pylab
A_image = pylab.mean(pylab.imread(DATA + 'emoryimage.png'), 2)
@interact
def svd_image(i=(20,(1..100)), display_axes=True):
    u,s,v = pylab.linalg.svd(A_image)
    A     = sum(s[j]*pylab.outer(u[0:,j], v[j,0:]) for j in range(i))
    g = graphics_array([matrix_plot(A),matrix_plot(A_image)])
    show(g, axes=display_axes, figsize=(8,3))
    html('<h2>Compressed using %s eigenvalues</h2>'%i)
```

i 65

display_axes

Compressed using 65 eigenvalues

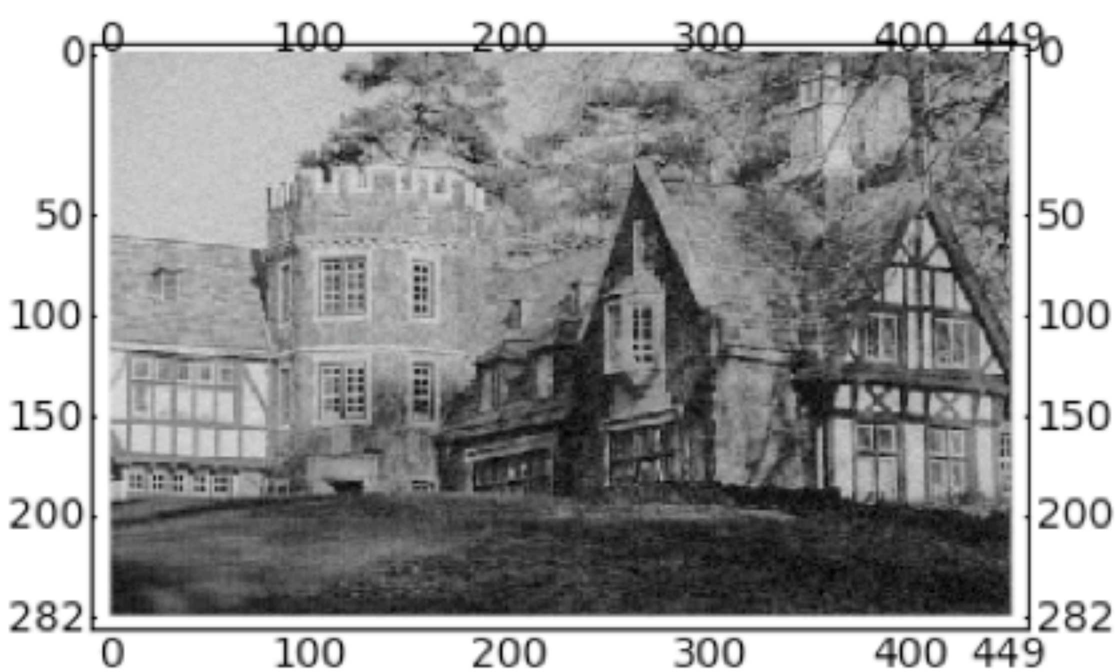


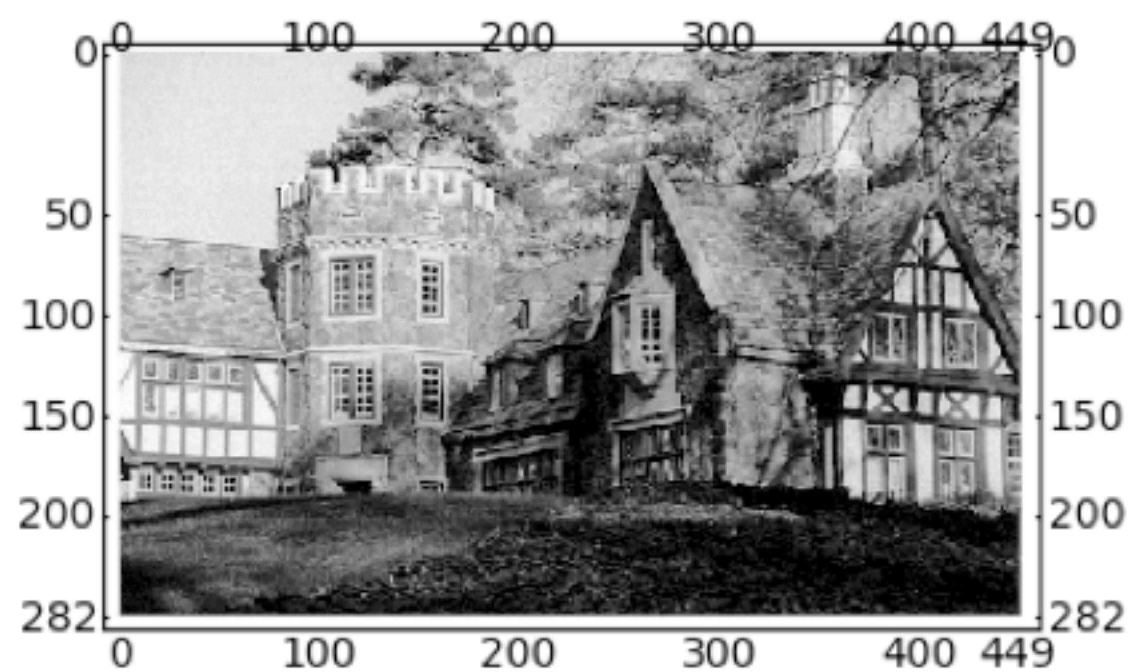
```
import pylab
A_image = pylab.mean(pylab.imread(DATA + 'emoryimage.png'), 2)
@interact
def svd_image(i=(20,(1..100)), display_axes=True):
    u,s,v = pylab.linalg.svd(A_image)
    A     = sum(s[j]*pylab.outer(u[0:,j], v[j,0:]) for j in range(i))
    g = graphics_array([matrix_plot(A),matrix_plot(A_image)])
    show(g, axes=display_axes, figsize=(8,3))
    html('<h2>Compressed using %s eigenvalues</h2>'%i)
```

i 90

display_axes ☑

Compressed using 90 eigenvalues

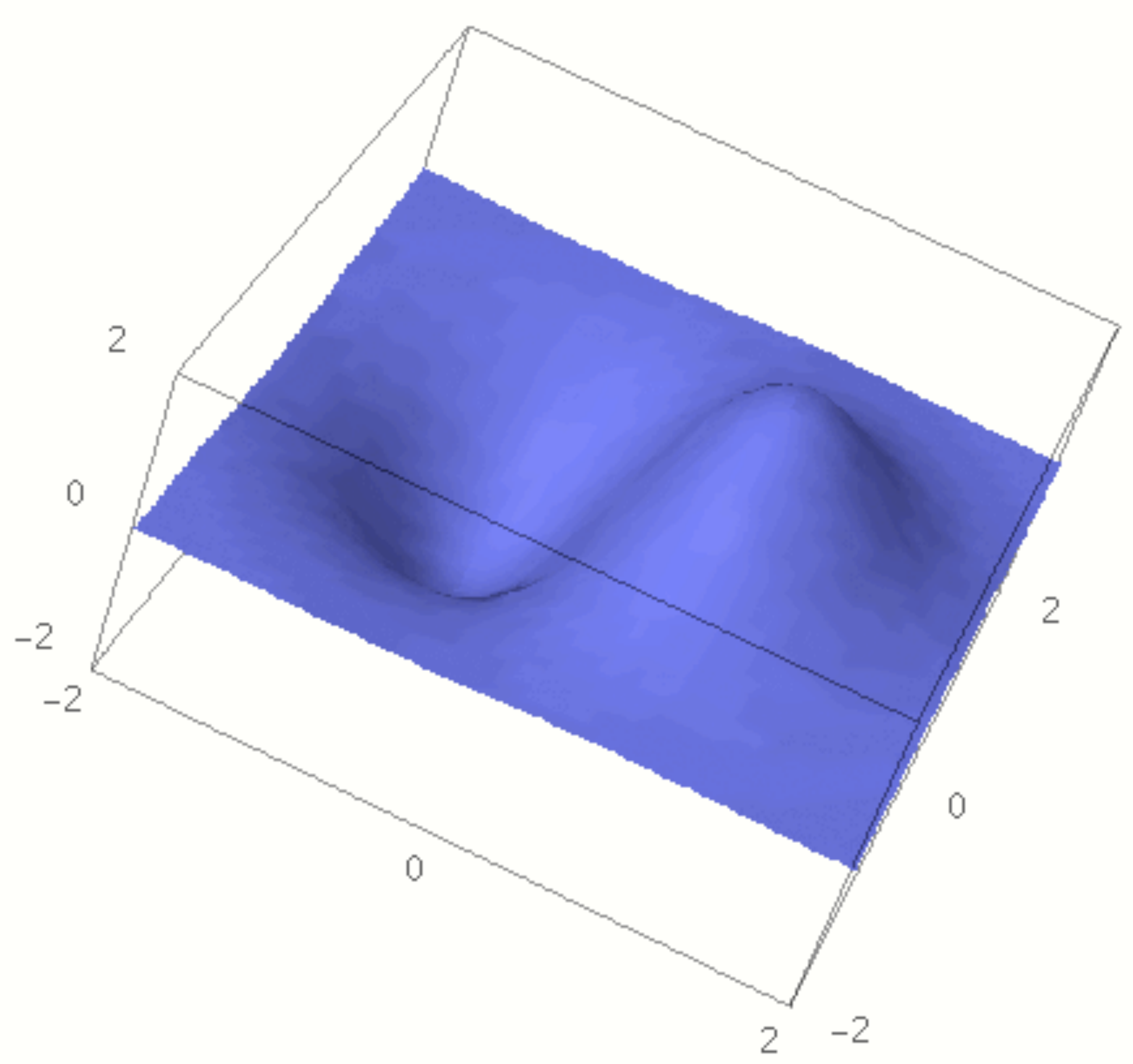




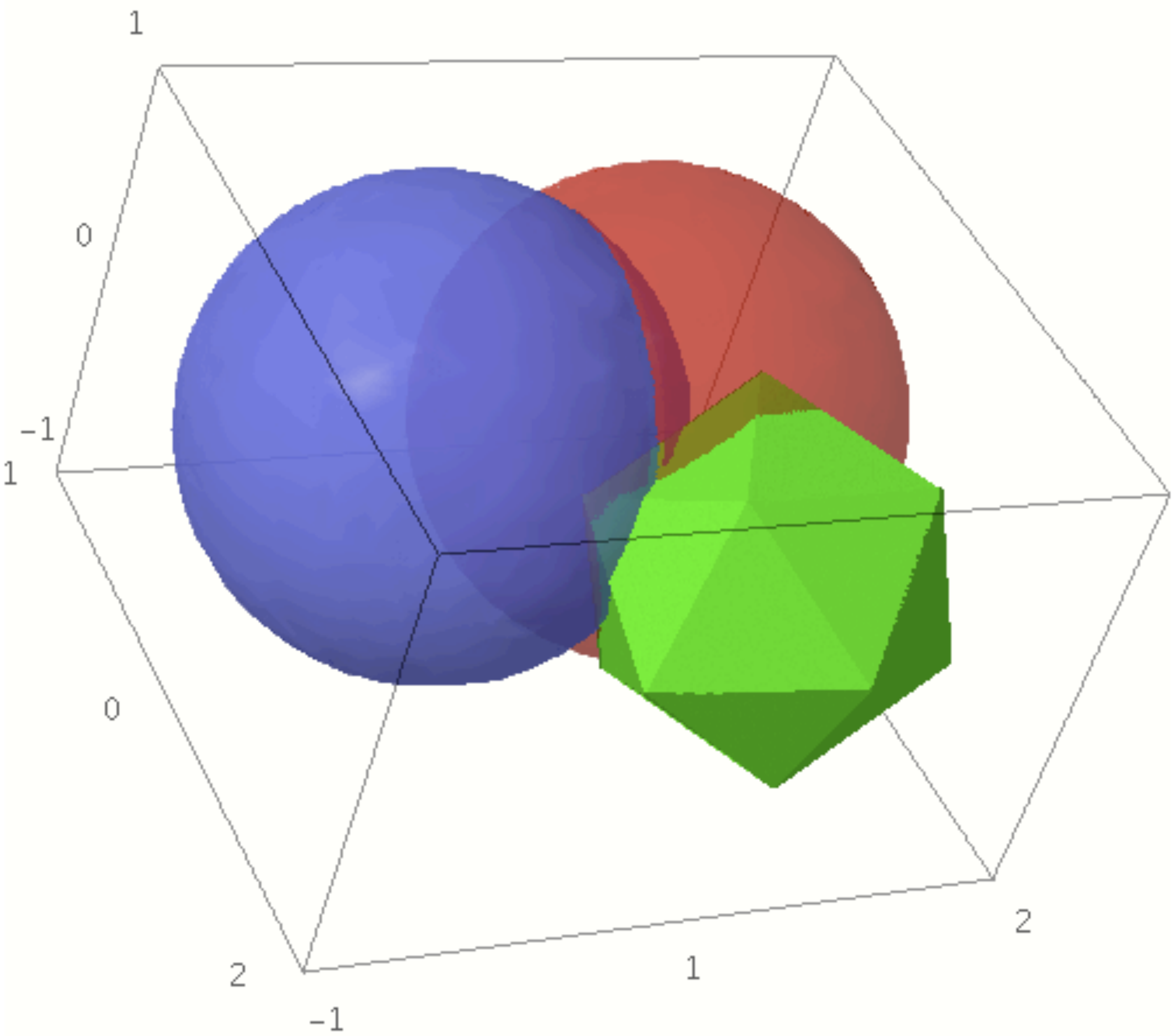
3d Plots

```
var('x y')
plot3d( 4*x*exp(-x^2-y^2), (x,-2,2), (y,-2,2) )
```

evaluate



```
( sphere((0,0,0), opacity=0.7) + sphere((0,1,0), color='red', opacity=0.5)
        + icosahedron((1,1,0), color='green') )
```

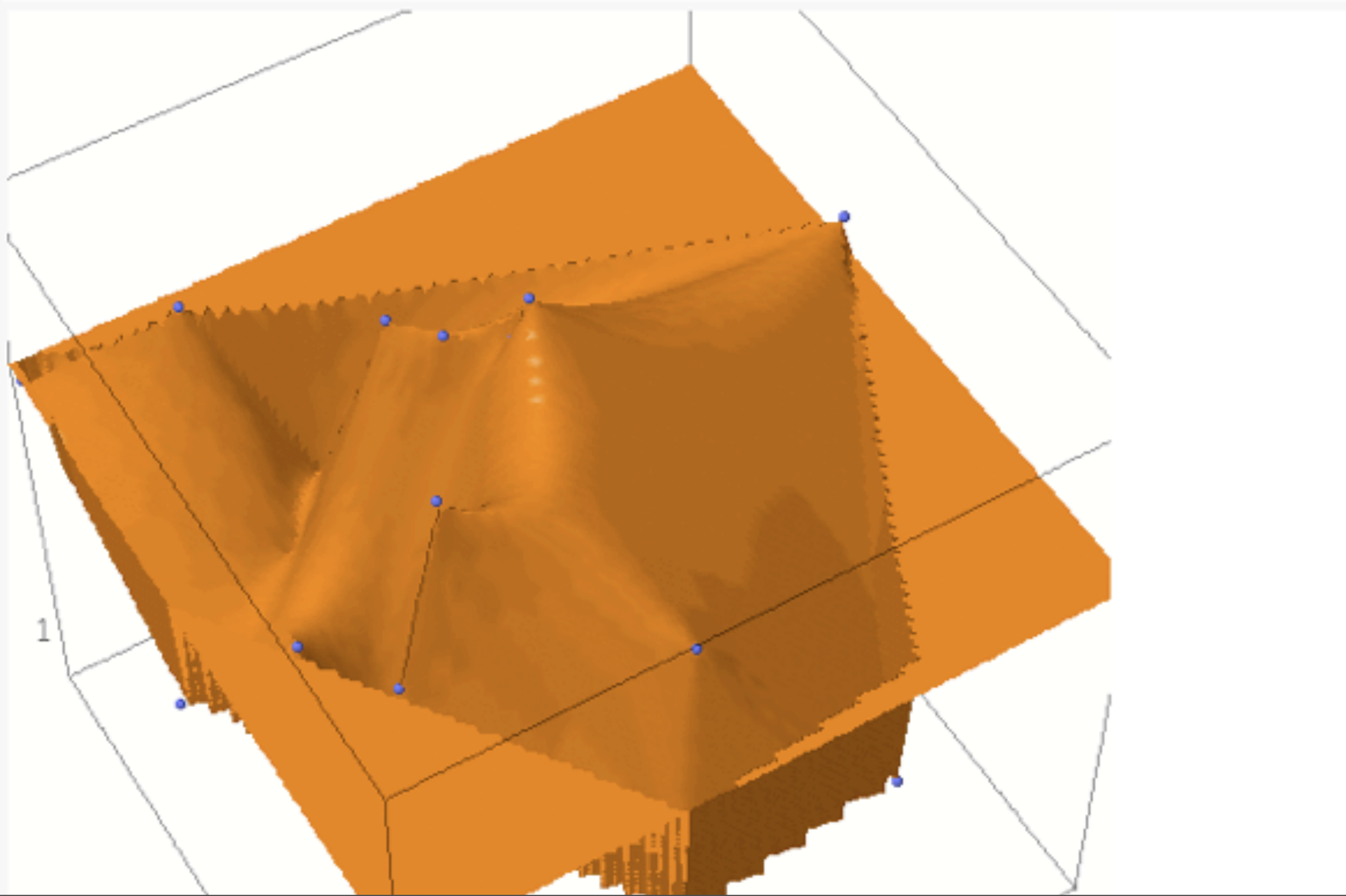


```
L = []

@interact
def random_list(number_of_points=(10..50), dots=True):
    n = normalvariate
    global L
    if len(L) != number_of_points:
        L = [(n(0,1), n(0,1), n(0,1)) for i in range(number_of_points)]
    G = list_plot3d(L,interpolation_type='nn', texture=Color('#ff7500'),num_points=120)
    if dots: G += point3d(L)
    G.show()
```

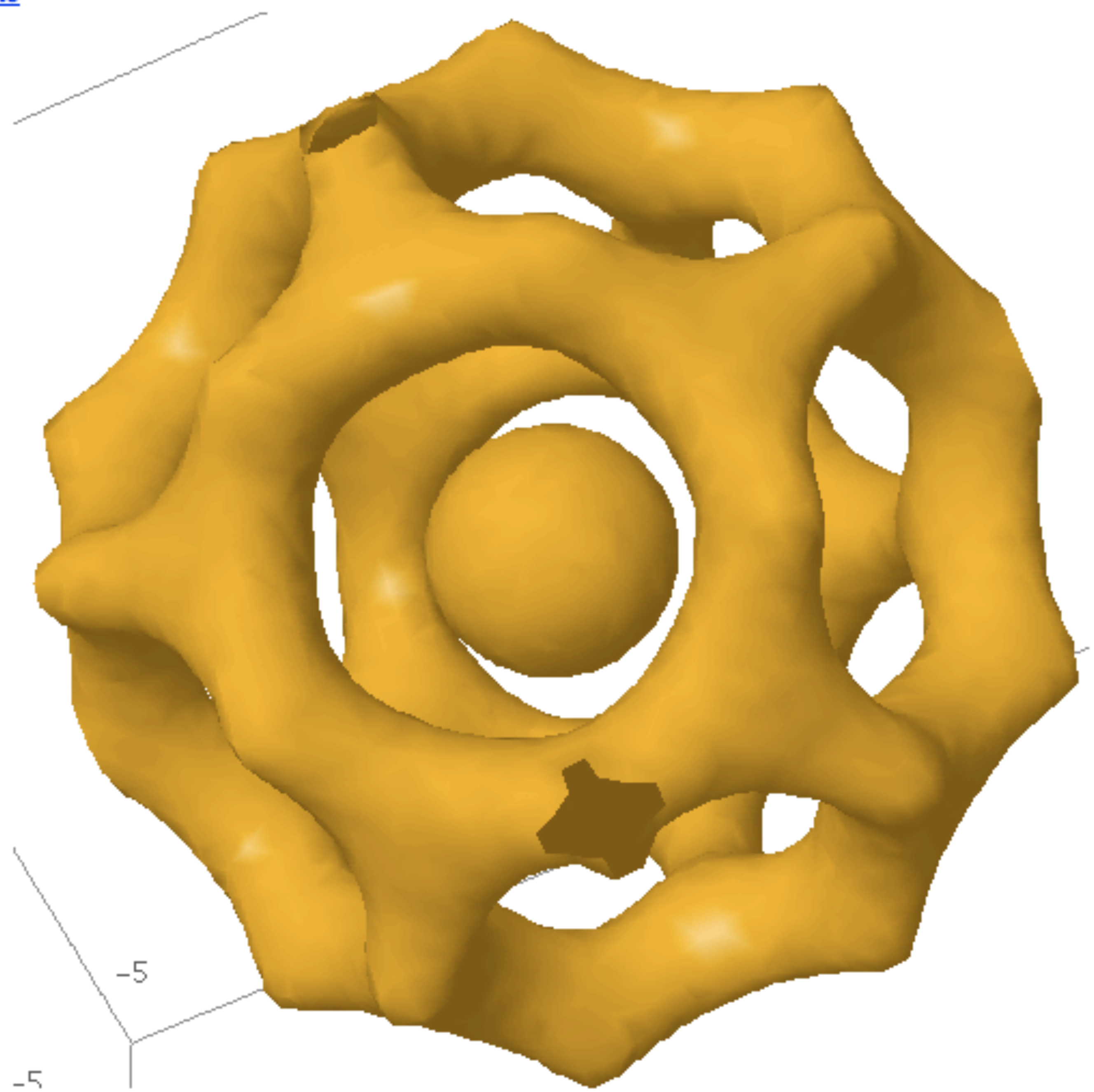
number_of_points 17

dots ☑



```
# implicit_plot3d -- not yet released
# (see http://trac.sagemath.org/sage_trac/ticket/5249)!!
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*x) + cos(z + T*x)))
r = 4.77
implicit_plot3d(p, (-r, r), (-r, r), (-r, r), plot_points=40)
```

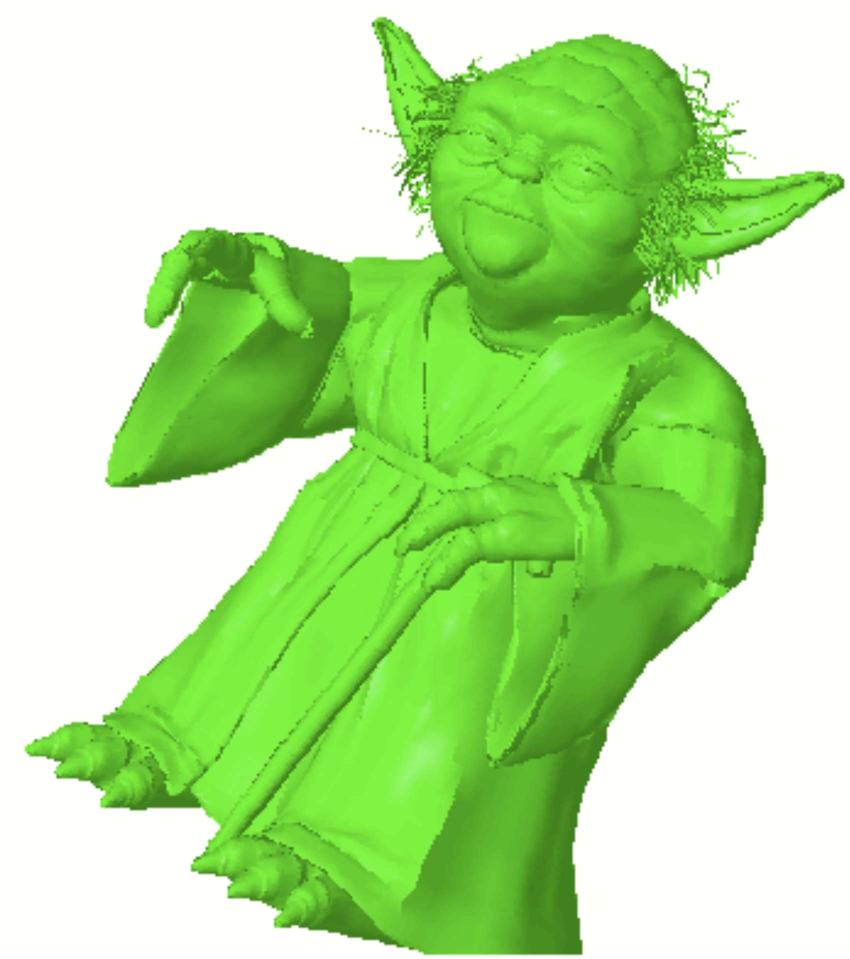
evaluate



3d plotting (using jmol) is fast even though it does **not** use Java3d or OpenGL or require any special signed code or drivers.

```
# Yoda! -- over 50,000 triangles.
from scipy import io
X = io.loadmat(DATA + 'yodapose.mat')
from sage.plot.plot3d.index_face_set import IndexFaceSet
V = X['V']; F3=X['F3']-1; F4=X['F4']-1
Y = IndexFaceSet(F3,V,color='green') + IndexFaceSet(F4,V,color='green')
Y = Y.rotateX(-1)
Y.show(aspect_ratio=[1,1,1], frame=False, figsize=4)
html('"Use the source, Luke..."')
```

"Use the source, Luke..."



Cython: Sage's Compiler

```
to      sage-support
date    Sat, Jan 31, 2009 at 11:15 AM
Hi,

I received first a MemoryError, and later on Sage reported:

uitkomst1=[]
uitkomst2=[]
eind=int((10^9+2)/(2*sqrt(3)))
print eind
for y in srange(1,eind):
 test1=is_square(3*y^2+1,True)
 test2=is_square(48*y^2+1,True)
 if test1[0] and test1[1]%3==2: uitkomst1.append((y,(2*test1[1]-1)/3))
 if test2[0] and test2[1]%3==1: uitkomst2.append((y,(2*test2[1]+1)/3))
print uitkomst1
een=sum([3*x-1 for (y,x) in uitkomst1 if 3*x-1<10^9])
print uitkomst2
twee=sum([3*x+1 for (y,x) in uitkomst2 if 3*x+1<10^9])
print een+twee

If you replace 10^9 with 10^6, the above listing works properly.

Maybe I made a mistake?

Rolandb
```

```
def f_python(n):
    uitkomst1=[]
    uitkomst2=[]
    eind=int((n+2)/(2*sqrt(3)))
    print eind
    for y in (1..eind):
        test1=is_square(3*y^2+1,True)
        test2=is_square(48*y^2+1,True)
        if test1[0] and test1[1]%3==2:
            uitkomst1.append((y,(2*test1[1]-1)/3))
        if test2[0] and test2[1]%3==1:
            uitkomst2.append((y,(2*test2[1]+1)/3))
    print uitkomst1
    een=sum(3*x-1 for (y,x) in uitkomst1 if 3*x-1<10^9)
    print uitkomst2
    twee=sum(3*x+1 for (y,x) in uitkomst2 if 3*x+1<10^9)
    print een+twee
```

```
time f_python(10^5)
```

```
28868
[(1, 1), (15, 17), (209, 241), (2911, 3361)]
[(1, 5), (14, 65), (195, 901), (2716, 12545)]
51408
Time: CPU 0.72 s, Wall: 0.77 s
```

```
time f_python(10^6)
```

evaluate

```
288675
[(1, 1), (15, 17), (209, 241), (2911, 3361), (40545, 46817)]
[(1, 5), (14, 65), (195, 901), (2716, 12545), (37829, 174725)]
716034
Time: CPU 7.14 s, Wall: 7.65 s
```

While waiting to see if f_python(10^9) would finish, I decided to try the ***Cython compiler***. I declared a few data types, put %cython at the top of the cell, and wham, it got ***over 200 times faster***.

```
%cython
from sage.all import is_square
cdef extern from "math.h":
    long double sqrtl(long double)

def f(n):
    uitkomst1=[]
    uitkomst2=[]
    cdef long long eind=int((n+2)/(2*sqrt(3)))
    cdef long long y, t
    print eind
    for y in range(1,eind):
        t = <long long>sqrtl(<long long> (3*y*y + 1))
        if t * t == 3*y*y + 1:
            uitkomst1.append((y, (2*t-1)/3))
        t = <long long>sqrtl(<long long> (48*y*y + 1))
        if t * t == 48*y*y + 1:
            uitkomst2.append((y, (2*t+1)/3))
    print uitkomst1
    een=sum([3*x-1 for (y,x) in uitkomst1 if 3*x-1<10^9])
    print uitkomst2
    twee=sum([3*x+1 for (y,x) in uitkomst2 if 3*x+1<10^9])
    print een+twee
```

__Users_ws..._code_sage214_spyx.c   __Users_ws...de_sage214_spyx.html

```
time f(10^5)
```

```
28868
[(1L, 1L), (4L, 4L), (15L, 17L), (56L, 64L), (209L, 241L), (780L, 900L),
(2911L, 3361L), (10864L, 12544L)]
[(1L, 5L), (14L, 65L), (195L, 901L), (2716L, 12545L)]
2
Time: CPU 0.00 s, Wall: 0.00 s
```

```
time f(10^6)
```

```
288675
[(1L, 1L), (4L, 4L), (15L, 17L), (56L, 64L), (209L, 241L), (780L, 900L),
(2911L, 3361L), (10864L, 12544L), (40545L, 46817L), (151316L, 174724L)]
[(1L, 5L), (14L, 65L), (195L, 901L), (2716L, 12545L), (37829L, 174725L)]
2
Time: CPU 0.03 s, Wall: 0.03 s
```

```
time f(10^9)
```

```
288675135
[(1L, 1L), (4L, 4L), (15L, 17L), (56L, 64L), (209L, 241L), (780L, 900L),
(2911L, 3361L), (10864L, 12544L), (40545L, 46817L), (151316L, 174724L),
(564719L, 652081L), (2107560L, 2433600L), (7865521L, 9082321L),
(29354524L, 33895684L), (109552575L, 126500417L)]
[(1L, 5L), (14L, 65L), (195L, 901L), (2716L, 12545L), (37829L, 174725L),
(526890L, 2433601L), (7338631L, 33895685L), (102213944L, 472105985L)]
2
Time: CPU 25.60 s, Wall: 26.50 s
```

```
7.14/0.03
```

```
238.000000000000
```

This is ***not*** a contrived example. This is a **real world example** that came up last weekend. For C-style computations, ***Sage (via Cython) is as fast as C***.

Numerical Matrix Algebra

```
a = random_matrix(RDF, 3); show(a)
```

$$\begin{pmatrix} -0.898388350554 & -0.245114654439 & 0.630541996552 \\ 0.654081247919 & 0.696119569249 & -0.957945963168 \\ 0.00662634420862 & -0.704817828713 & -0.517608714138 \end{pmatrix}$$

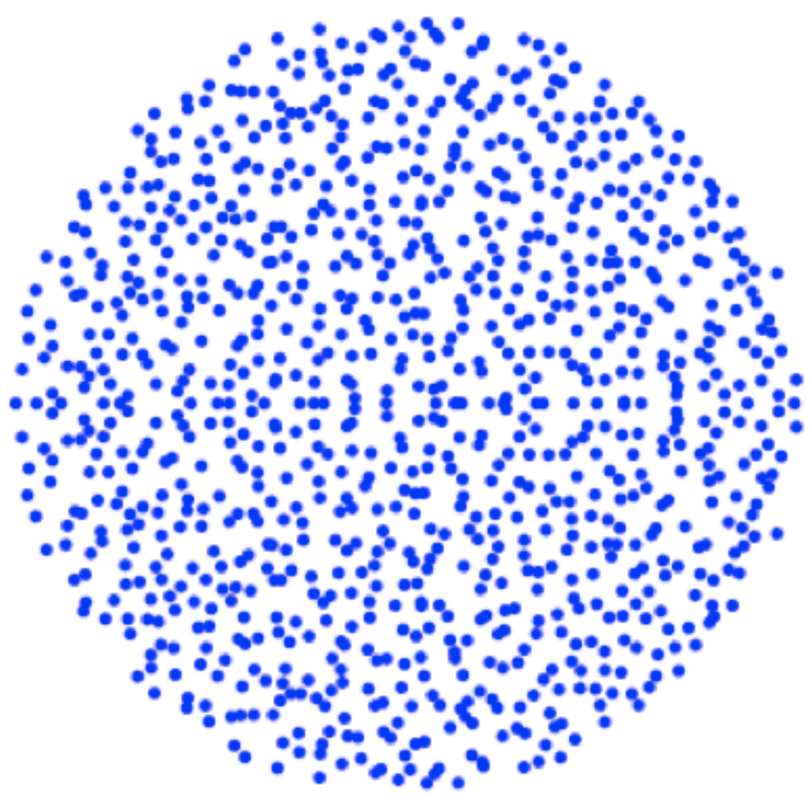
```
show(a.eigenvalues())
```

$[0.961673429437, -0.4808854176, -1.20066550728]$

```
a = random_matrix(RDF, 1000)
time v = a.eigenvalues()
```

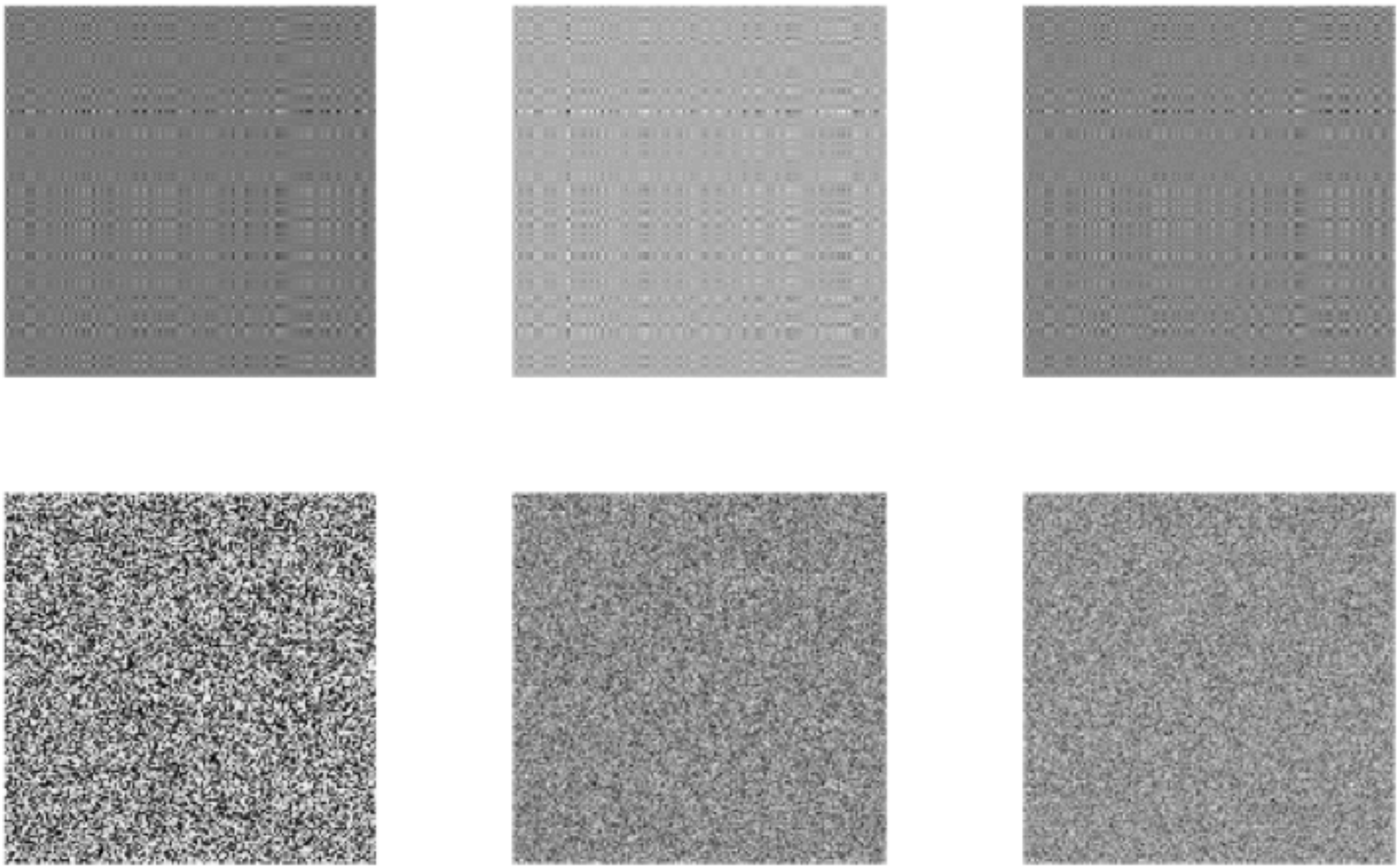
```
Time: CPU 5.63 s, Wall: 7.68 s
```

```
show(points(v), axes=False, aspect_ratio=1, figsize=4)
```



```
a = random_matrix(RDF, 200)  # read doubles in [-1,1]
G = graphics_array([matrix_plot(a^i) for i in [-3,-2,-1,1,2,3]], 2, 3)
show(G, axes=False)
```

evaluate



Part 2: An Extended Example from Computational Number Theory

The Birch and Swinnerton-Dyer Conjecture

Nonsingular Plane Curves

A nonsingular plane algebraic curve is the set of solutions to a (nonsingular) polynomial:

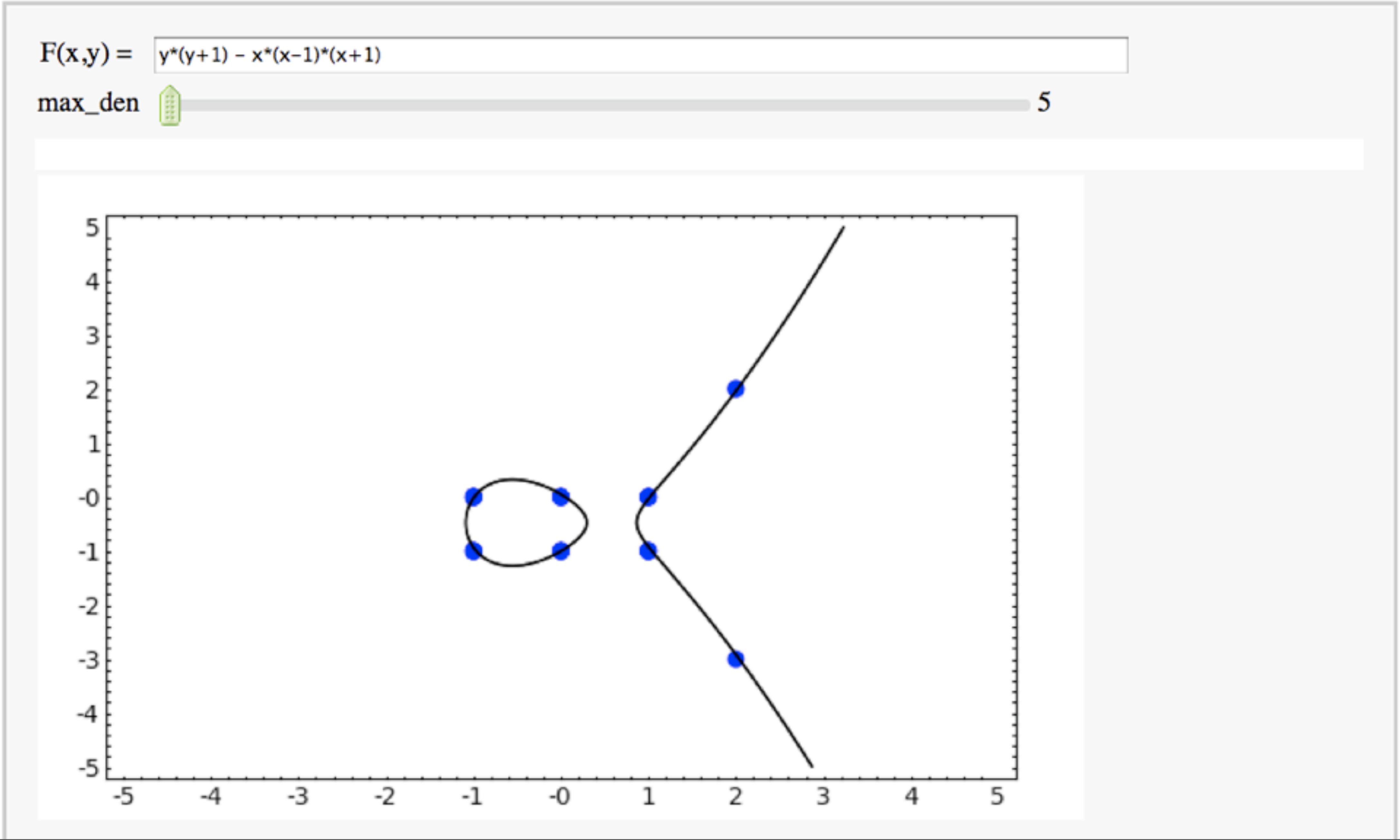
$$F(X, Y) = 0$$

A *rational point* is $(x, y) \in \mathbf{Q} \times \mathbf{Q}$ such that $F(x, y) = 0$.

- **Ancient Theorem:** A curve of **degree** ≤ 2 has no rational points ($x^2 + y^2 = -1$) or infinitely many rational points ($x^2 + y^2 = 1$), and there is a way to decide which and enumerate all solutions.
- **Faltings Theorem (1985):** A curve of **degree** ≥ 4 has finitely many rational points.
- **Birch and Swinnerton-Dyer Conjecture (1960s):** A curve of **degree 3** has either finitely many rational points ($x^3 + y^3 = 1$) or infinitely many rational points $y^2 + y = x^3 - x$. The BSD Conjecture provides a way to decide which and enumerate all solutions.

Rational Points on Plane Curves

```
@interact
def f(F = ('F(x,y) = ', 'y*(y+1) - x*(x-1)*(x+1)'), max_den=(5..100)):
    R.<x,y> = QQ[]
    try: F = R(F.lower())
    except: print "Enter a polynomial with rational coefficients."; return
    g = F._fast_float_()
    show(implicit_plot(F, (x,-5,5), (y,-5,5), plot_points=200) +
     points([(a/d,b/d) for a in [-5..5] for b in [-5..5] for d in [1..max_den]
            if g(a/d,b/d) == 0], pointsize=40))
```



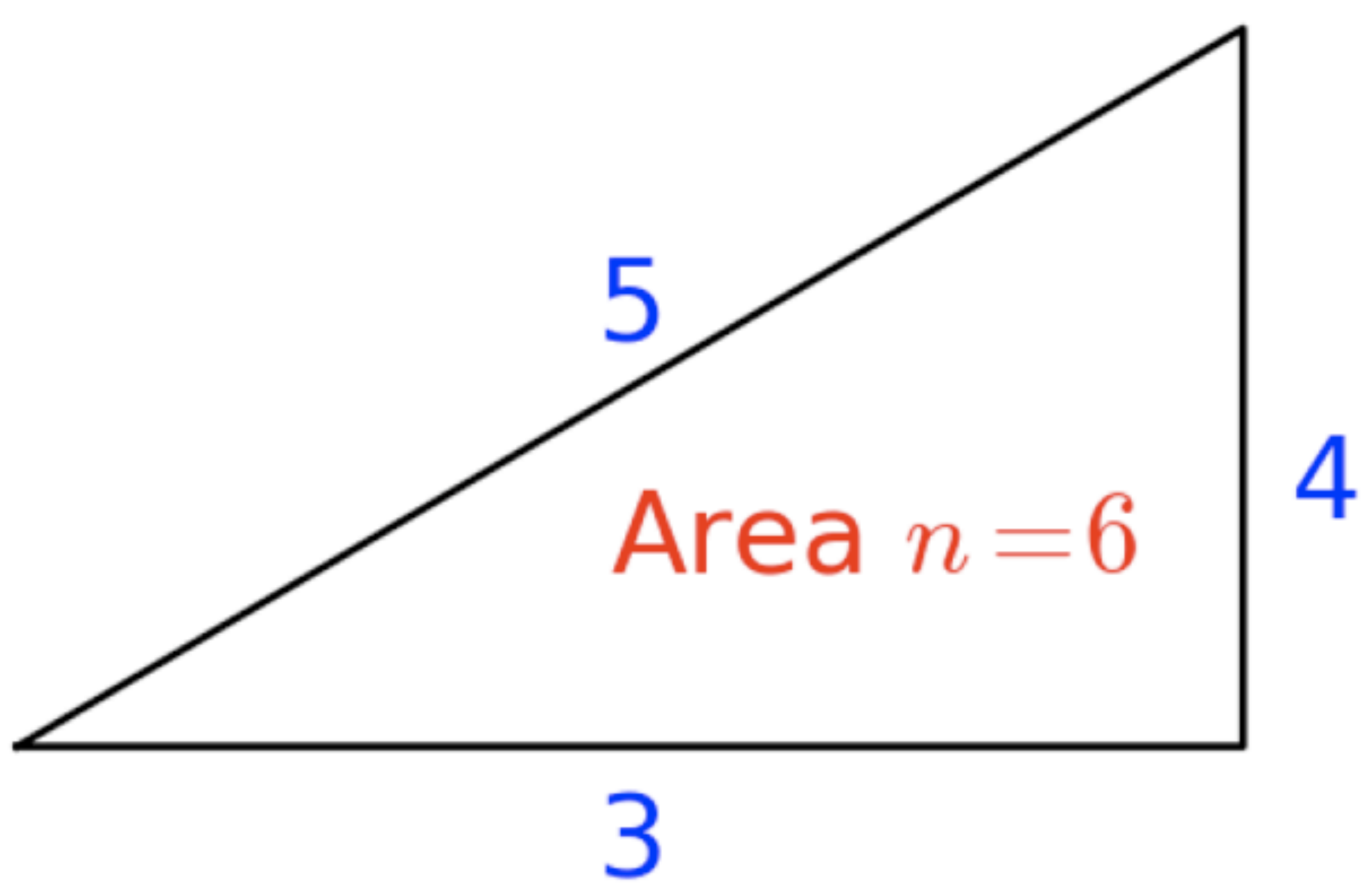
The Congruent Number Problem

Definition: An integer n is a **congruent number** if n is the area of a right triangle with rational side lengths.

Open Problem: Give an algorithm to decide whether or not an integer n is a congruent number.

This is a ***1000-year old open problem***, perhaps the *oldest* open problem in mathematics.

```
T = line([(0,0), (3,0), (3,4), (0,0)],rgbcolor='black',thickness=2)
lbl = text("3",(1.5,-.5),fontsize=28) + text("4",(3.2,1.5),fontsize=28)
lbl += text("5",(1.5,2.5),fontsize=28)
lbl += text("Area $n = 6$", (2.1,1.2), fontsize=28, rgbcolor='red')
show(T+lbl, axes=False)
```



Congruent Numbers and the BSD Conjecture

Theorem: *A proof of the Birch and Swinnerton-Dyer Conjecture would also solve the congruent number problem.*

Proof: Suppose n is a positive integer. Consider the cubic curve $y^2 = x^3 - n^2x$. Using algebra (see next slide), one sees that this cubic curve has infinitely many rational points if and only if there are rationals a, b, c such that $n = ab/2$ and $a^2 + b^2 = c^2$. The Birch and Swinnerton-Dyer conjecture gives an algorithm to decide whether or not any cubic curve has infinitely many solutions.

Explicit Bijection

In fact, there is a bijection between

$$A = \left\{ (a, b, c) \in \mathbf{Q}^3 \ : \ \frac{ab}{2} = n,\ a^2 + b^2 = c^2 \right\}$$

and

$$B = \left\{ (x, y) \in \mathbf{Q}^2 \ : \ y^2 = x^3 - n^2 x,\ \text{ with } y \neq 0 \right\}$$

given explicitly by the maps

$$f(a, b, c) = \left(-\frac{nb}{a+c},\ \frac{2n^2}{a+c} \right)$$

and

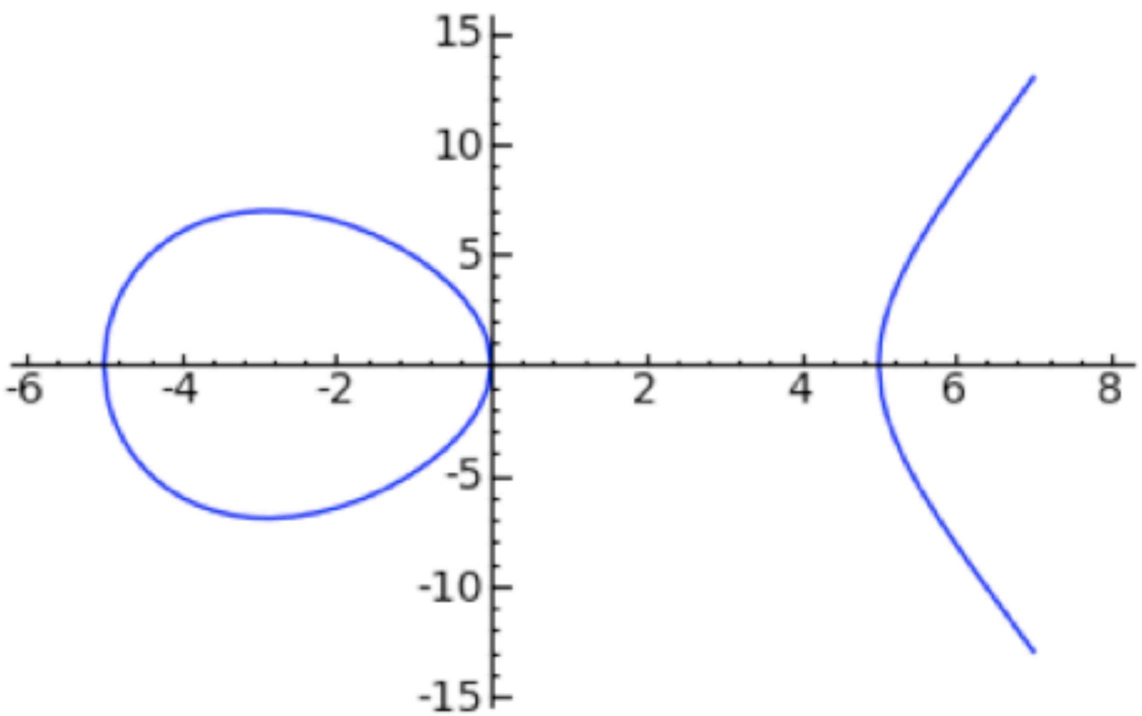
$$g(x, y) = \left(\frac{n^2 - x^2}{y},\ -\frac{2xn}{y},\ \frac{n^2 + x^2}{y} \right).$$

5 is a Congruent Number

```
n = 5; x,y = var('x,y')
C = EllipticCurve(y^2 == x^3 - n^2 * x); C
```

```
Elliptic Curve defined by y^2 = x^3 - 25*x over Rational Field
```

```
show(C.plot(), figsize=4)
```




```
P = C.gens()[0]
print P
print "order of P = ", P.order()
```

```
(-4 : 6 : 1)
order of P =  +Infinity
```

```
(-62279/1728)^2 == (1681/144)^3 - 25*(1681/144)
```

evaluate

```
True
```

1 is *Not* a Congruent Number

```
n = 1
x,y = var('x,y')
C = EllipticCurve(y^2 == x^3 - n^2 * x)
C
```

```
Elliptic Curve defined by y^2 = x^3 - x over Rational Field
```

```
C.gens()
```

```
[]
```

Finding Explicit Rational Right Triangles

```
@interact
def _(n=2009, triangles=(1..10)):
    x,y = var('x,y')
    C = EllipticCurve(y^2 == x^3 - n^2*x)
    G = C.gens()
    html("rank = %s\n\n"%len(G))
    if len(G) == 0: print "%s is not a congruent number"%n; return
    def g(x,y,n): return ((n^2-x^2)/y, -2*x*n/y, (n^2+x^2)/y)
    P = G[0]
    for i in [1..triangles]:
        a,b,c = g((i*P)[0], (i*P)[1], n)
        html("a=%s, b=%s, c=%s\n"%(a,b,c))
```

n 2009

triangles 2

```
rank = 2


a=280/3, b=861/20, c=6167/60

a=-3526873/105720, b=-8669040/71977, c=950998057921/7609408440
```

The L-function

Let C be a cubic curve (+ a technical condition I'm not mentioning). For each *prime number* p, let N_p be the number of solutions to the cubic modulo p.

Definition: For any cubic curve C, let $a_p = p - N_p$.

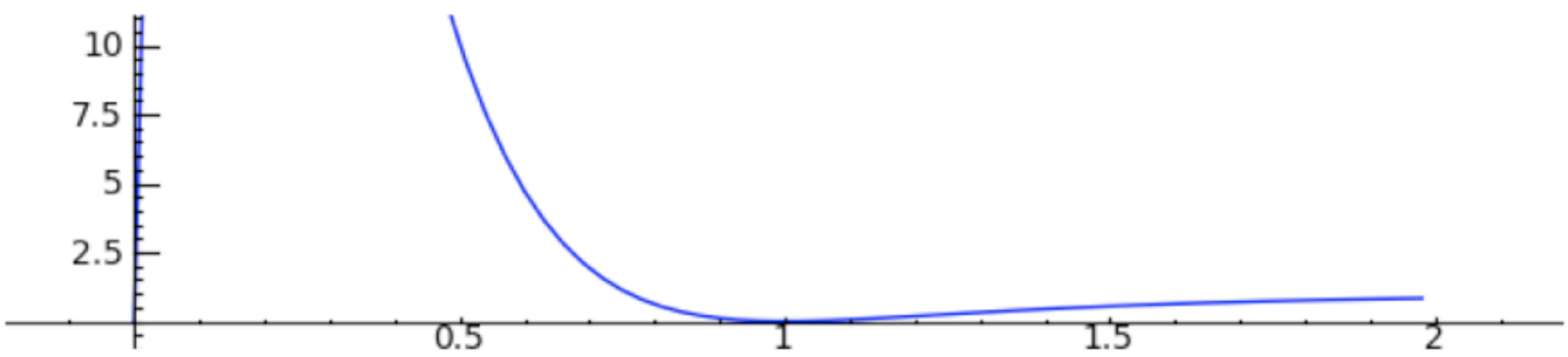
Theorem (Hasse): $|a_p| < 2\sqrt{p}$.

Theorem (Wiles et al.): The function

$$L(C,s) = \prod_p \left(\frac{1}{1 - a_p p^{-s} + p^{1-2s}} \right)$$

extends to an entire complex-analytic function on $\mathbf{C}$.

```
L = EllipticCurve([-2009^2,0])._pari_().elllseries
show(line([(i,L(i)) for i in [0,0.03,..,2]]), figsize=[7,1.5], ymax=10)
```



The Birch and Swinnerton-Dyer Conjecture

Heuristic Observation: If C has infinitely many rational points, then the numbers N_p will tend to be "large". Since $L(C,1)$" = "$\prod_p \frac{p}{N_p}$, the number $L(C,1)$ will tend to be small.

Theorem (Mordell): There is a finite set $P_1, \ldots, P_r$ of rational points on C so that all (non-torsion) rational points can be generated from these using a simple geometric process (chords and tangents).

We call the smallest r in Mordell's theorem the ***rank*** of C.

Conjecture (Birch and Swinnerton-Dyer):

$$\mathrm{ord}_{s=1} L(C,s) = \mathrm{rank}(C)$$

This problem, exactly as stated, is the Clay Math Institute Million Dollar prize problem in number theory. We proved above that its solution would also resolve the 1000-year old congruent number problem.

Examples of the BSD Conjecture

```
@interact
def _(n=2009):
    x,y = var('x,y')
    C = EllipticCurve(y^2 == x^3 - n^2*x)
    show(C)
    print "rank = ", C.rank(), "\n"
    L = C.lseries()
    print "L-series = ", L.taylor_series(1,53, 4)
```

n 2009

$$y^2 = x^3 - 4036081x$$

```
rank =  2

L-series =  3.56988714561815e-24 + (-1.08215677130278e-23)*z +
8.21552435757629*z^2 - 24.9041074709231*z^3 + O(z^4)
```

The Kolyvagin -- Gross-Zagier Theorem

Theorem: *If* $\operatorname{ord}_{s=1} L(C, s) \leq 1$ *then the Birch and Swinnerton-Dyer conjecture is true for* C.

The proof involves Heegner points, modular curves, Euler systems and Galois cohomology.

My Current Research

- Study the mathematical structures (Heegner points, modular curves, Euler systems, etc.) that appear in the proof of the Kolyvagin-Gross-Zagier theorem in order to understand how to generalize *anything* to cubic curves with $\mathrm{ord}_{s=1} L(C, s) \geq 2$.

- This involves a combination of *technical theory* and *explicit machine computation*.