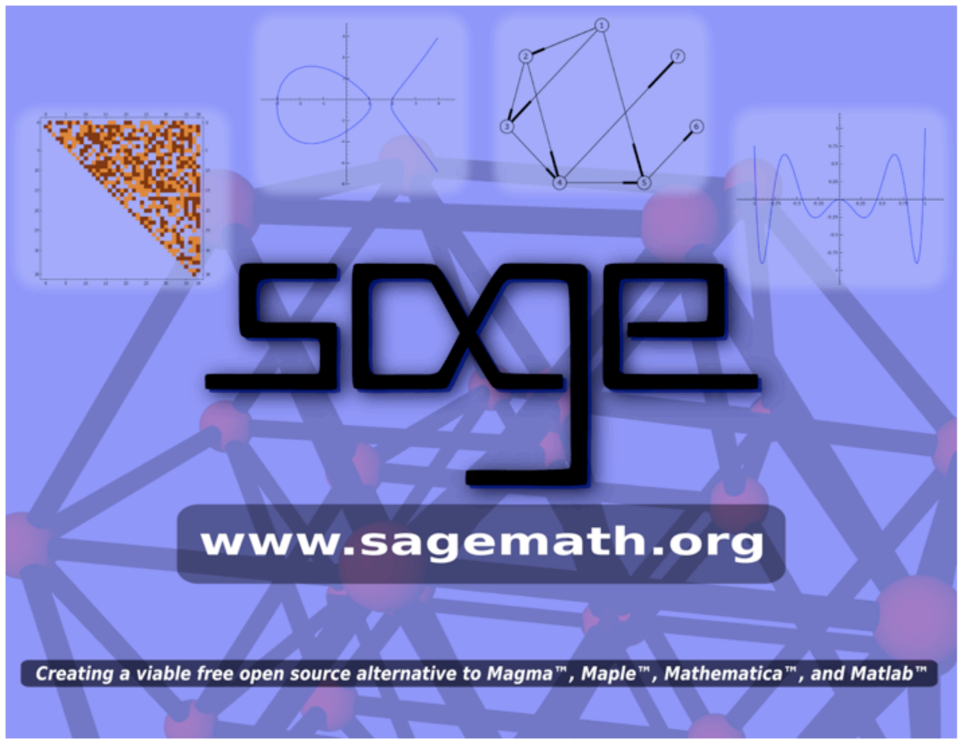
Sage

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

William Stein, Associate Professor, University of Washington



```
BTW: I really, REALLY like sage.  I'm just surprised i haven't heard
                    of it before 5 days ago.

-- Clinton Bowen on the sage-support mailing list Mon, Feb 9, 2009 at 11:03 PM.
```

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.

Sun's Interest in Sage

1. Sage will ***very*** soon (weeks) fully support Solaris. This porting work was fully funded by the Department of Defense, and will continue.

2. Sage is the ***only*** project whose goal is to create a viable open source alternative to all of Mathematica, Maple, Matlab and Magma. It is the analogue of Star Office or Firefox, as alternatives to Microsoft Office or Internet Explorer.

Sparc and x86 Solaris Port Nearly Done!

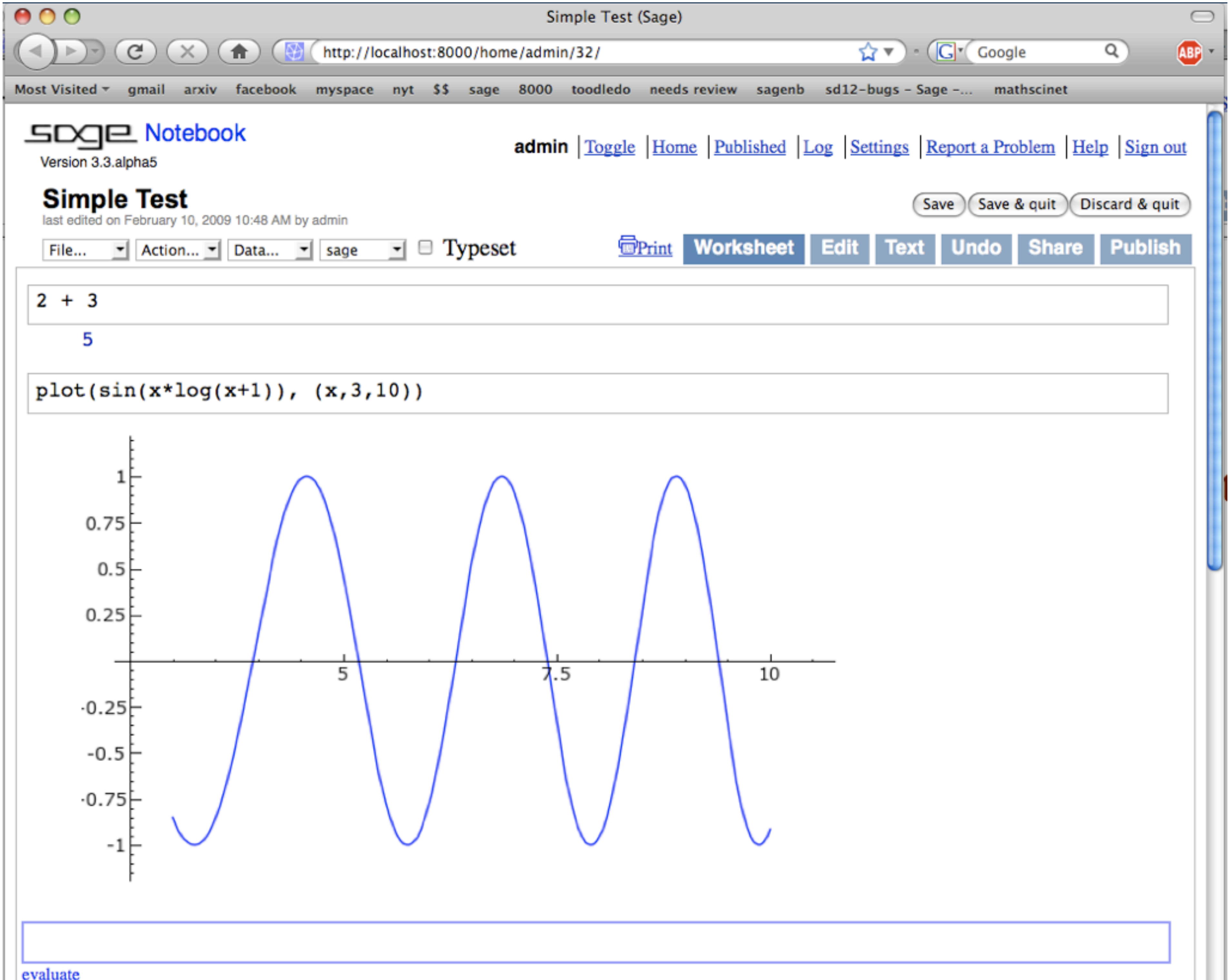
3.3.alpha6 on SOLARIS

Builds out of the box, but does not have the following fixes:

- RANDMAX setting (unclean patch, so not upstream yet)
- Singular pexepct (unclean patch, so not upstream yet)
- Sympow flag fixes (x86 only)

Open issues: There are less than ten issues left to fix and some of them already have fixes that need to be cleaned up. All but three issues are identical on both x86 and Sparc:

- Summetrica interface problem
- Nan vs. nan [has fix, needs to be cleaned up]
- inf vs. Infinity [has fix, needs to be cleaned up]
- coercion oddity - sparc specific
- various pexpect issues
- HeilbronnCremona() strangeness
- large allocation failure fails [change doctest?]
- pow()/axes.py problem
- numerical noise problems [trivial to fix]



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
evaluate

Sage is 100% Free and Open Source

Active user community; **964** members of the sage-support mailing list.

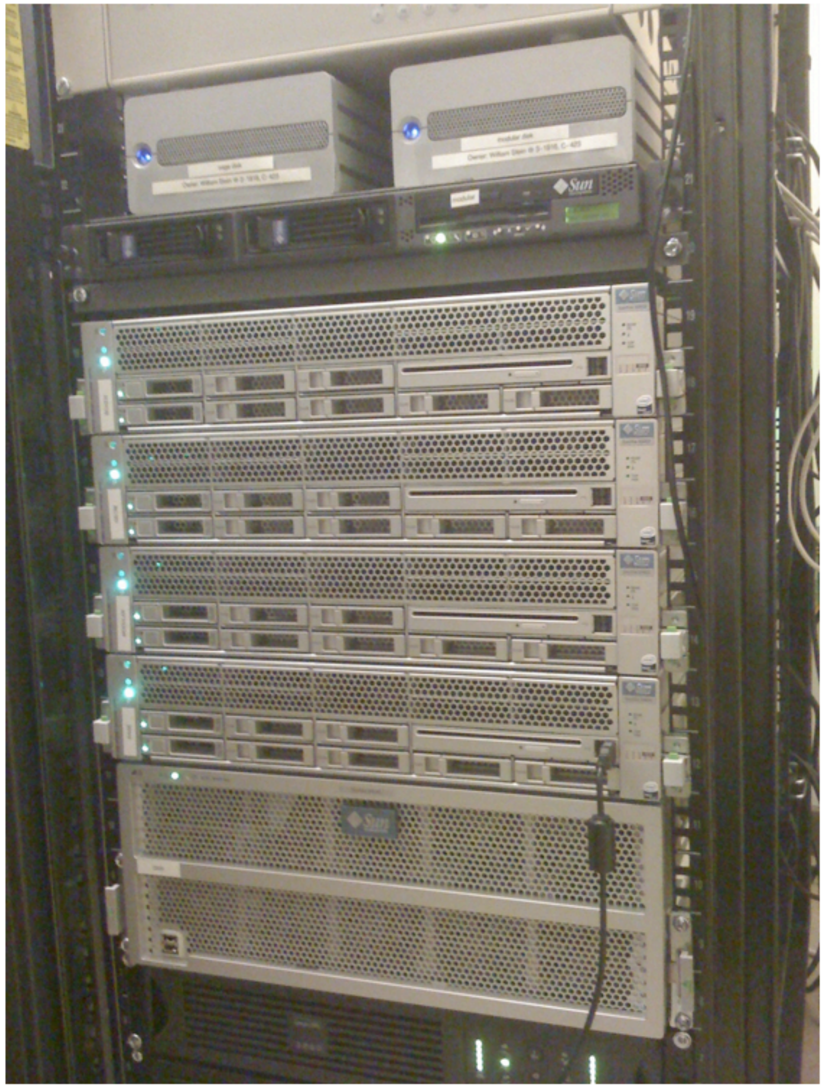
Discussions 8 of 9817 messages view all »

- [Wrong plot in optimisation problem - not tangent]
  By Paolo Crosetto - 8:46am - 3 authors - 3 replies
- [sage-support] Re: ILLEGAL INSTRUCTION sse4_pni
  By William Stein - 7:16am - 2 authors - 3 replies
- Iterators in compiled code?
  By Alasdair - 3:23am - 3 authors - 6 replies
- [sage-support] How to compute half-weight coefficients?
  By William Stein - Jan 30 - 2 authors - 1 reply
- [sage-support] Notebook Plotting
  By William Stein - Jan 30 - 2 authors - 1 reply
- problem with GraphDatabase
  By Jason Grout - Jan 30 - 2 authors - 1 reply
- Factorization
  By Paul Zimmermann - Jan 30 - 1 author - 0 replies
- [sage-support] How can I make a topographic map with Sage?
  By Benjamin J. Racine - Jan 30 - 5 authors - 4 replies

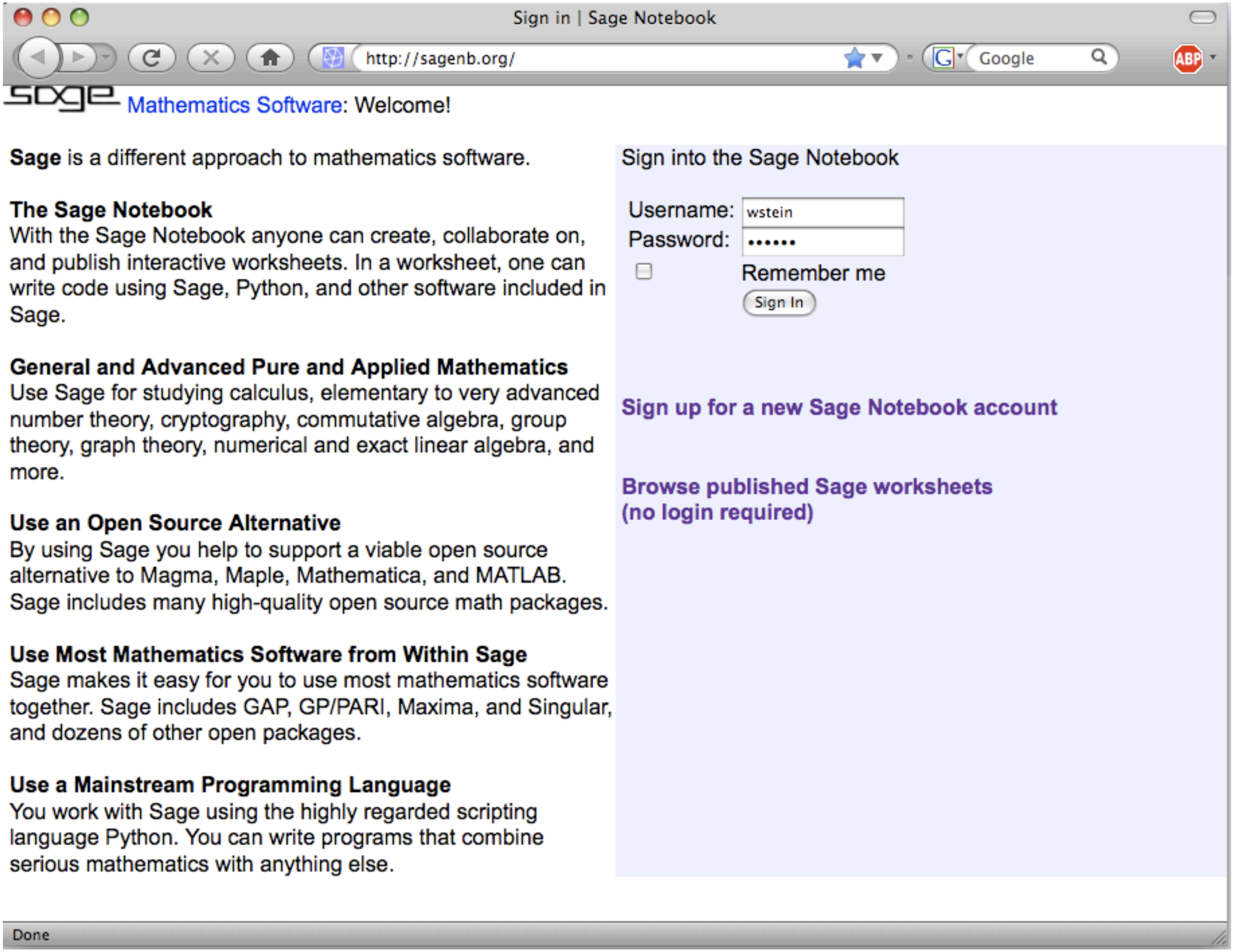
Members 964 members view all » + i

sri Member	indhu Member	bill.pa...@synthesis.anikast.ca Member	nat...@math.ucla.edu Member

sagemath.org



Sage Devel Headquarters: Four 24-core Sun X4450's with 128GB RAM each + 1 Sun X4540 with 24TB disk


Sign in | Sage Notebook
http://sagenb.org/
Google
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

Many Advantages of Sage over the Ma*'s

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

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*** (numpy, scipy, R, and gsl all included)
- Group theory, ***number theory***, combinatorics, crypto.
- ***Calculus***
- 2d and 3d ***plotting***
- ***Range of functionality rivals*** the Ma*'s

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
```

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)
```

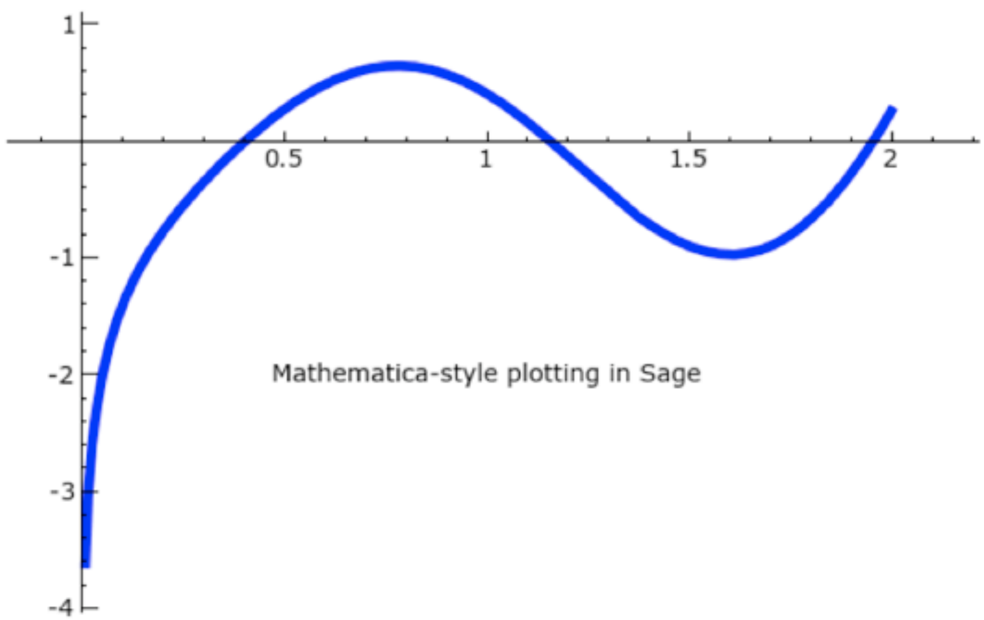
$$x \mapsto x \sin(3x) + \log(x) + \frac{1}{(x+1)^2}$$

Plotting functions has similar syntax to Mathematica:

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

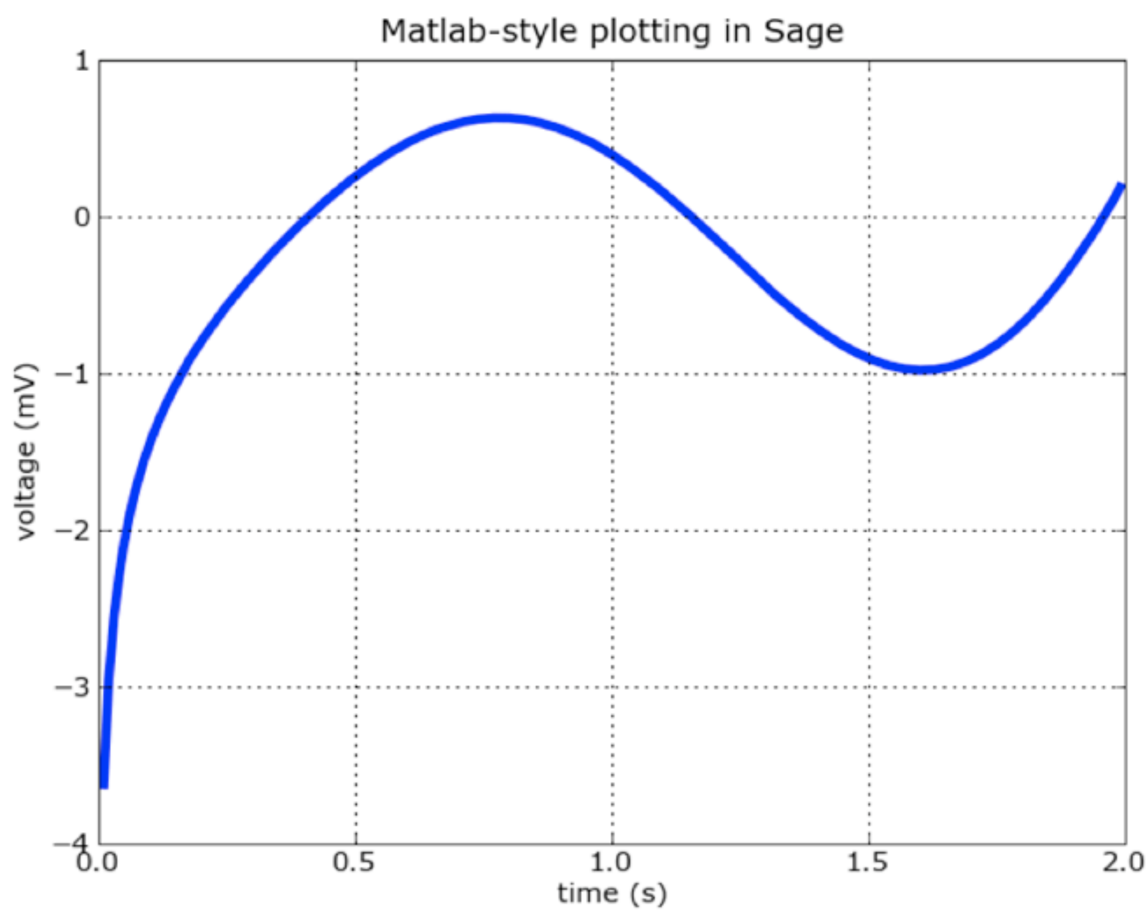
$$x \mapsto \frac{\sin(3x) - 3x\cos(3x)}{9} + x\log(x) - \frac{1}{x+1} - x$$

```
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) = x^3 * sin(x^2) + cos(x*y) - 1/(x+y+z)
```

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

```
    625 loops, best of 3: 709 ns per loop
```

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

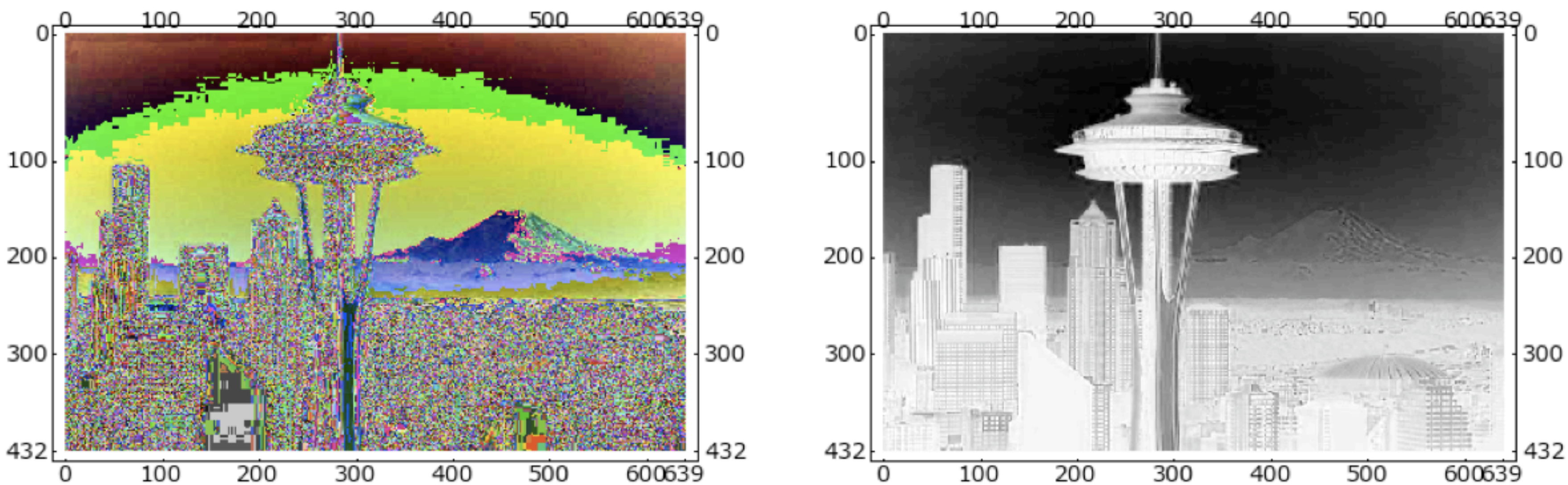
```
timeit('g(4.5r)')
```

```
    625 loops, best of 3: 6.89 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 + 'seattle.png')
graphics_array([matrix_plot(A^(-1)), matrix_plot(1-A[0:,0:,2])]).show(figsize=[10,4])
```

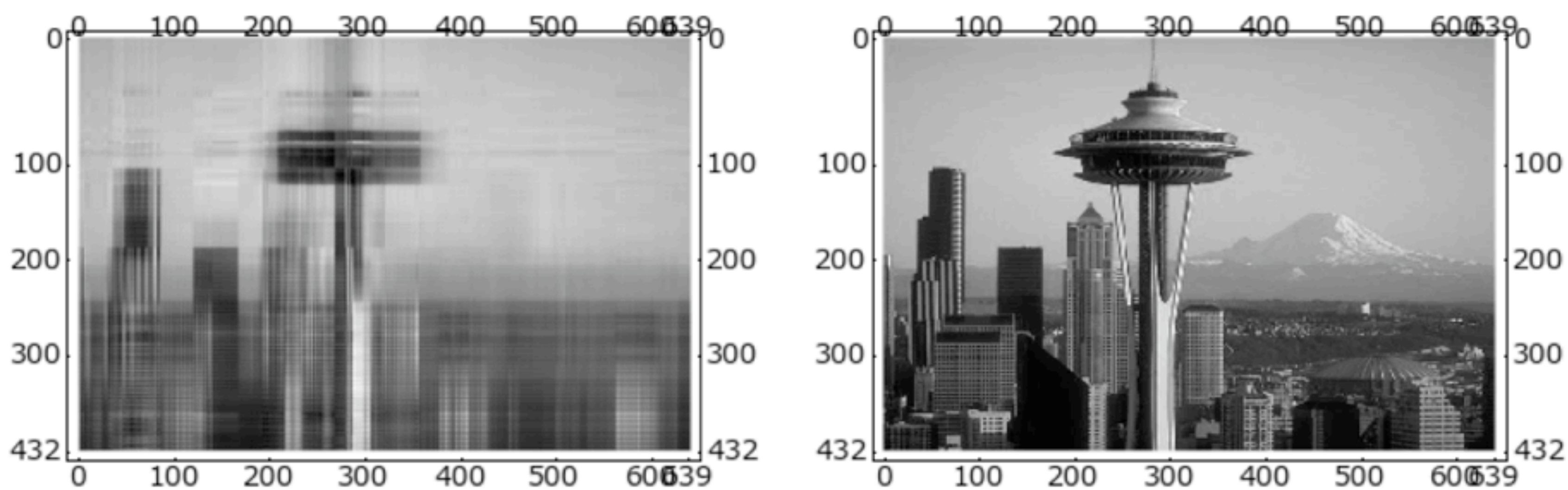


```
import pylab
A_image = pylab.mean(pylab.imread(DATA + 'seattle.png'), 2)
@interact
def svd_image(i=(20,(1..100)), display_axes=True):
    u,s,v = pylab.linalg.svd(A_image)
    A     = sum(s[j]*pylab.outer(u[0:,j], v[j,0:]) for j in range(i))
    g = graphics_array([matrix_plot(A),matrix_plot(A_image)])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

i 6

display_axes ☑

Compressed using 6 eigenvalues

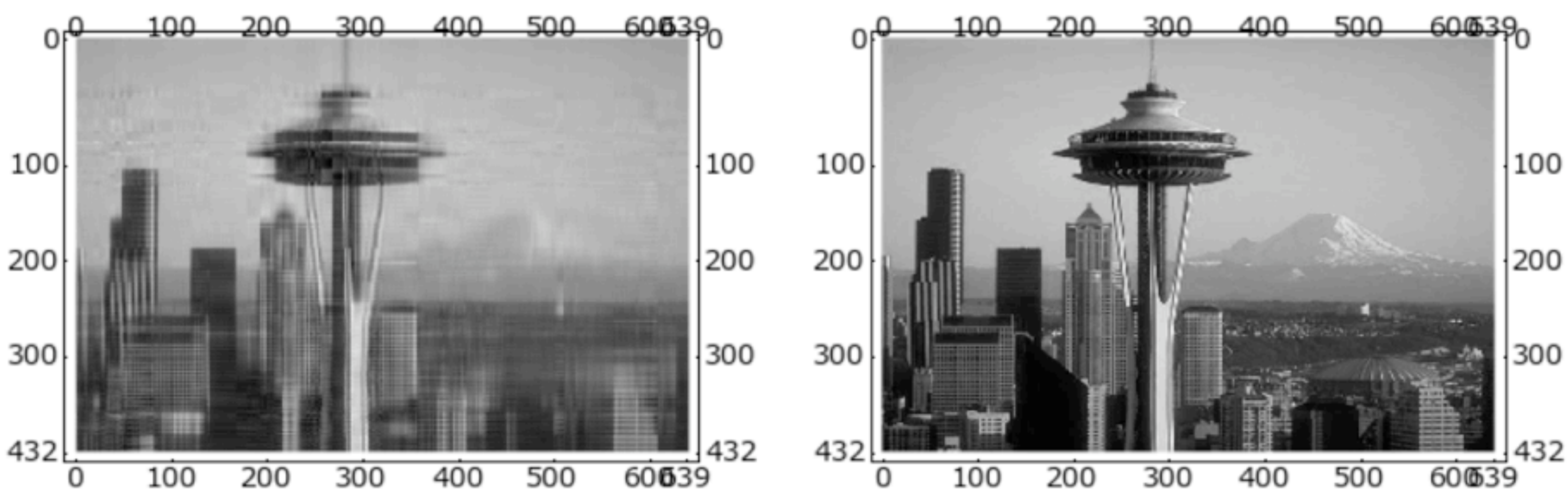


```
import pylab
A_image = pylab.mean(pylab.imread(DATA + 'seattle.png'), 2)
@interact
def svd_image(i=(20,(1..100)), display_axes=True):
    u,s,v = pylab.linalg.svd(A_image)
    A     = sum(s[j]*pylab.outer(u[0:,j], v[j,0:]) for j in range(i))
    g = graphics_array([matrix_plot(A),matrix_plot(A_image)])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

i 16

display_axes

Compressed using 16 eigenvalues

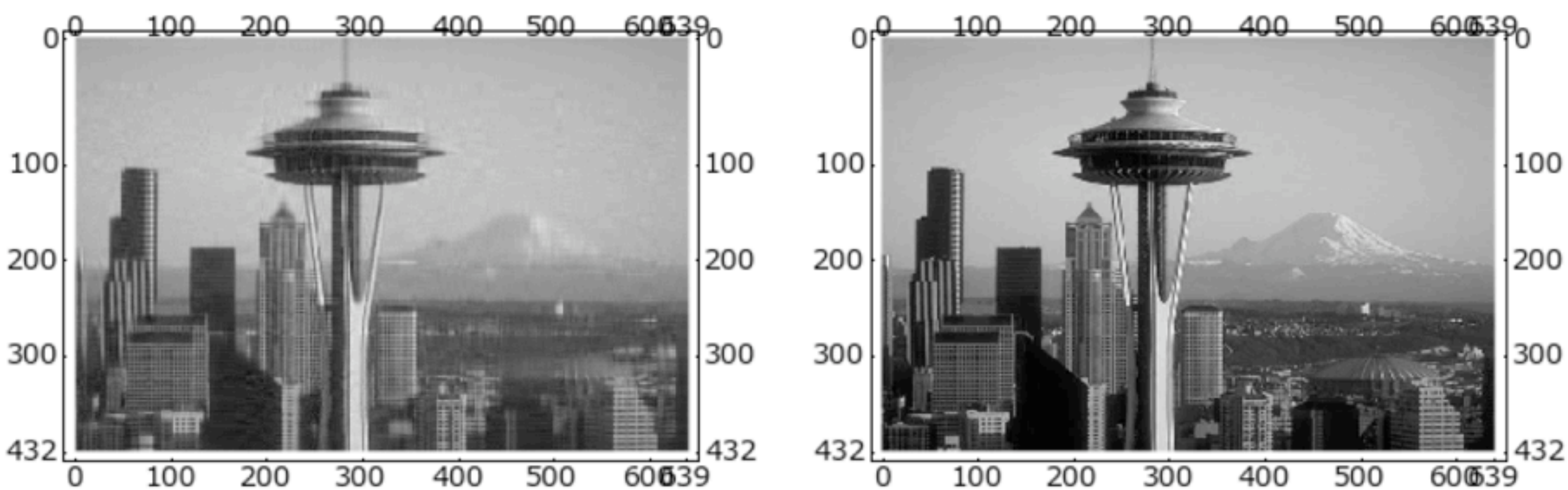


```
import pylab
A_image = pylab.mean(pylab.imread(DATA + 'seattle.png'), 2)
@interact
def svd_image(i=(20,(1..100)), display_axes=True):
    u,s,v = pylab.linalg.svd(A_image)
    A     = sum(s[j]*pylab.outer(u[0:,j], v[j,0:]) for j in range(i))
    g = graphics_array([matrix_plot(A),matrix_plot(A_image)])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

i 34

display_axes ☑

Compressed using 34 eigenvalues

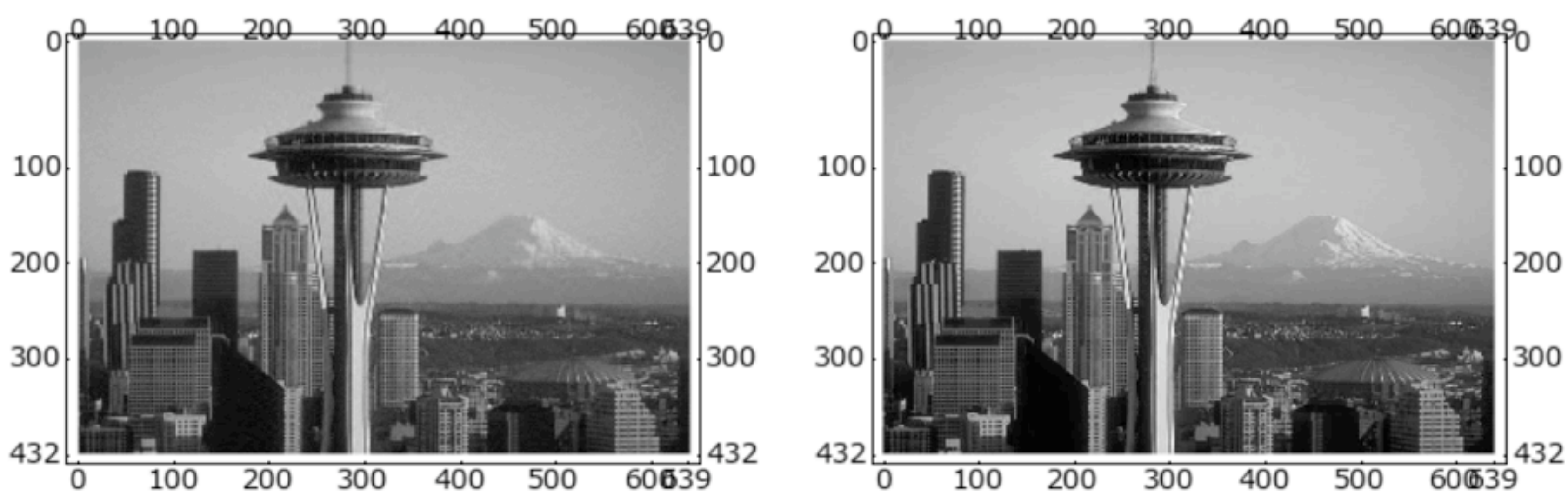


```
import pylab
A_image = pylab.mean(pylab.imread(DATA + 'seattle.png'), 2)
@interact
def svd_image(i=(20,(1..100)), display_axes=True):
    u,s,v = pylab.linalg.svd(A_image)
    A     = sum(s[j]*pylab.outer(u[0:,j], v[j,0:]) for j in range(i))
    g = graphics_array([matrix_plot(A),matrix_plot(A_image)])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

i 83

display_axes ☑

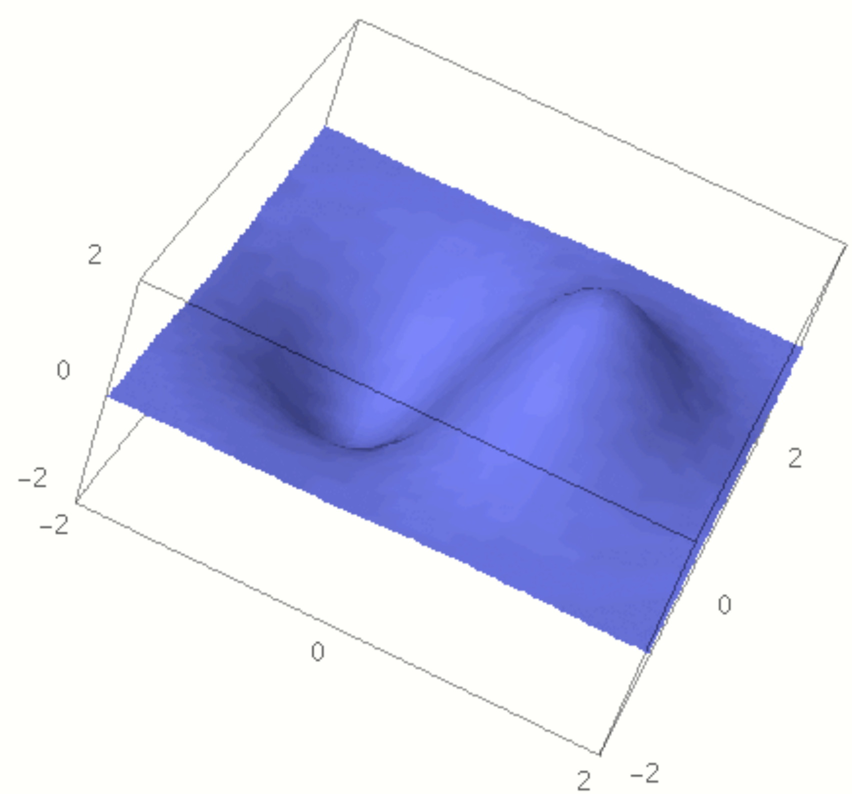
Compressed using 83 eigenvalues



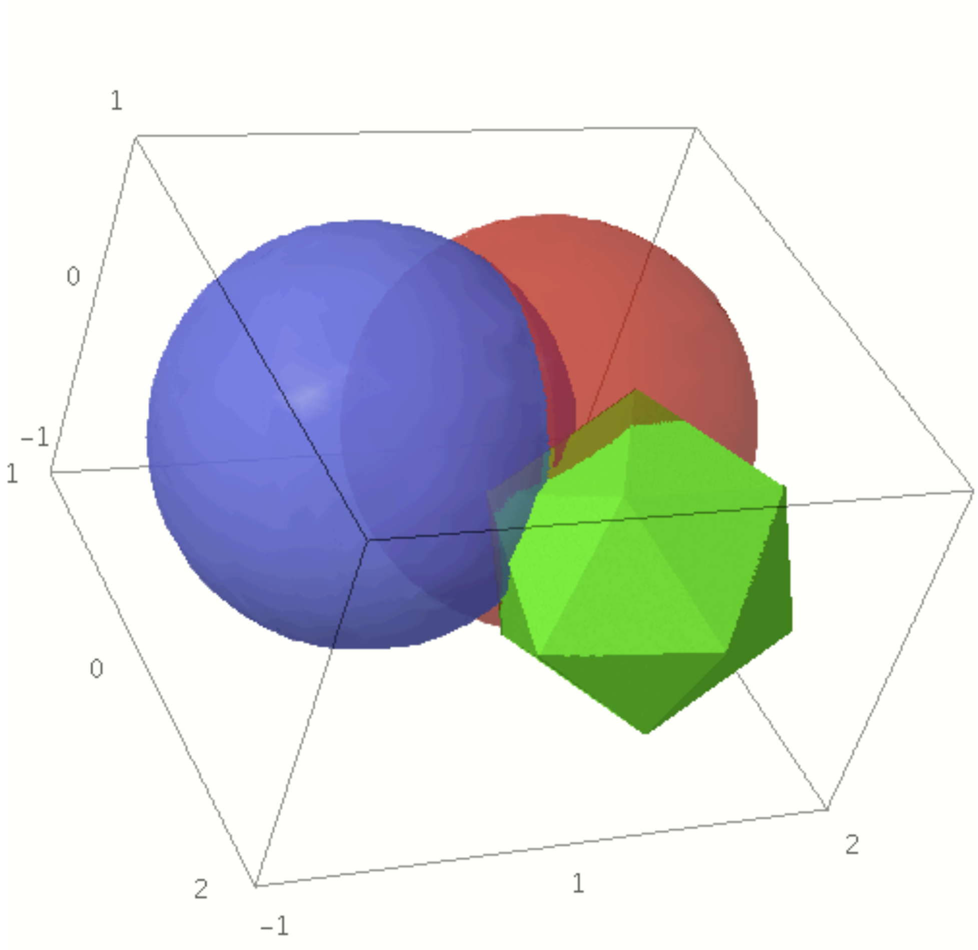
3d Plots

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

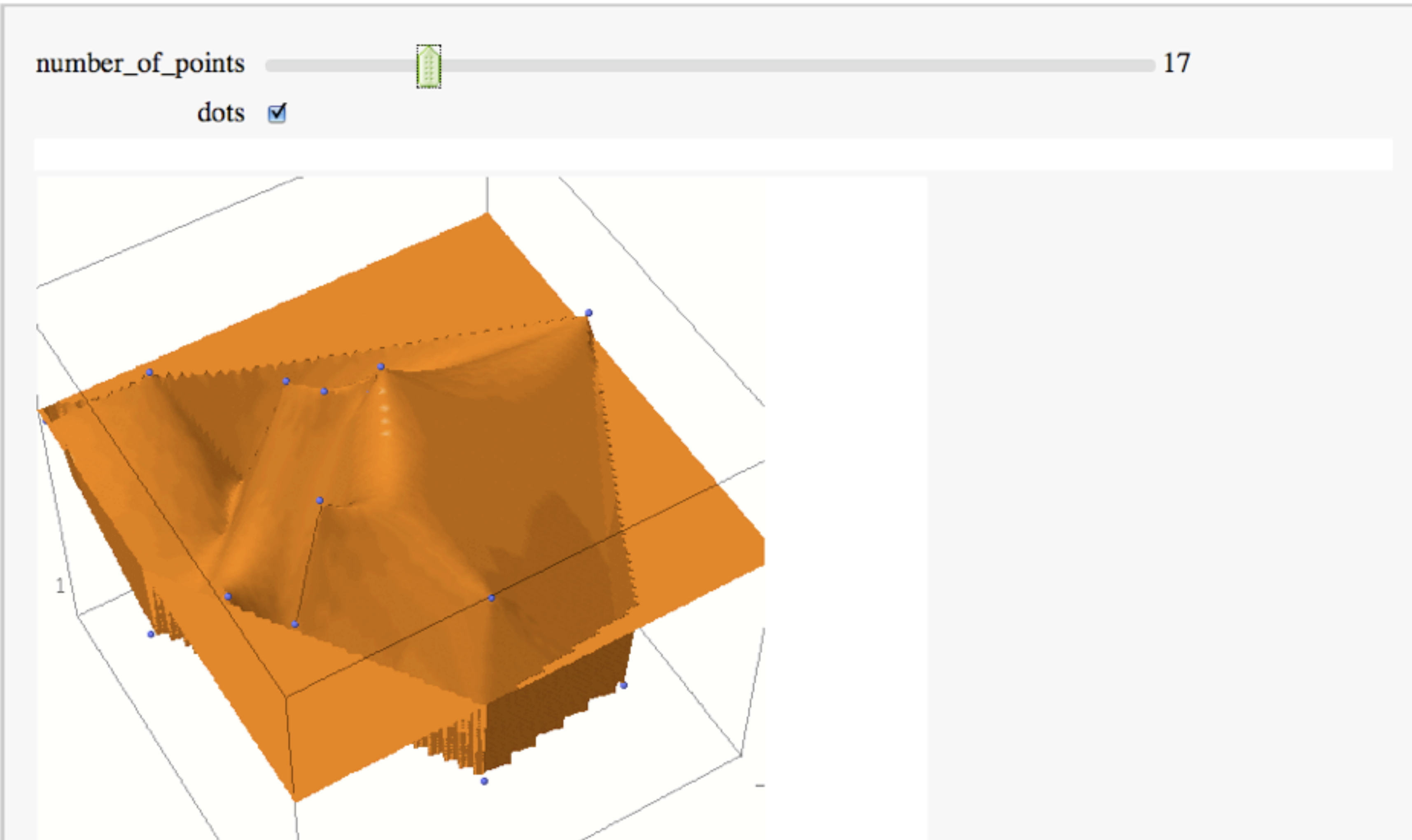
evaluate



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



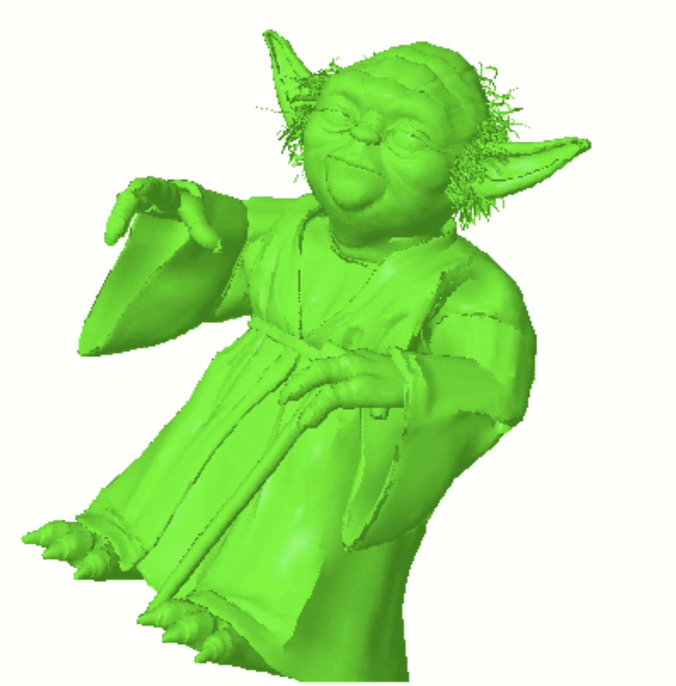
```
L = []

@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()
```



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)
```

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

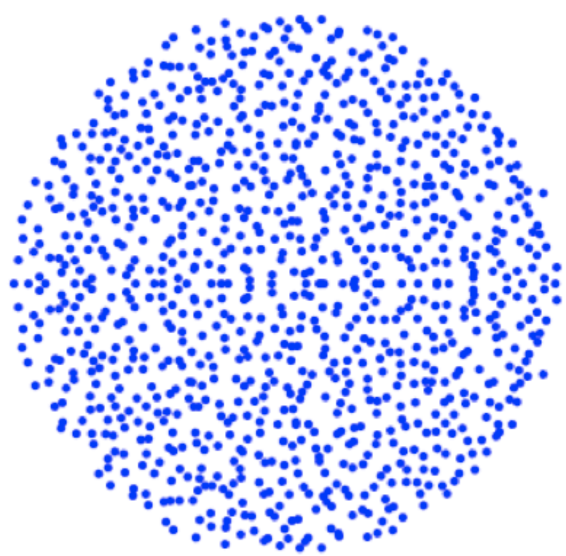
```
show(a.eigenvalues())
```

$$[0.961673429437, -0.4808854176, -1.20066550728]$$

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

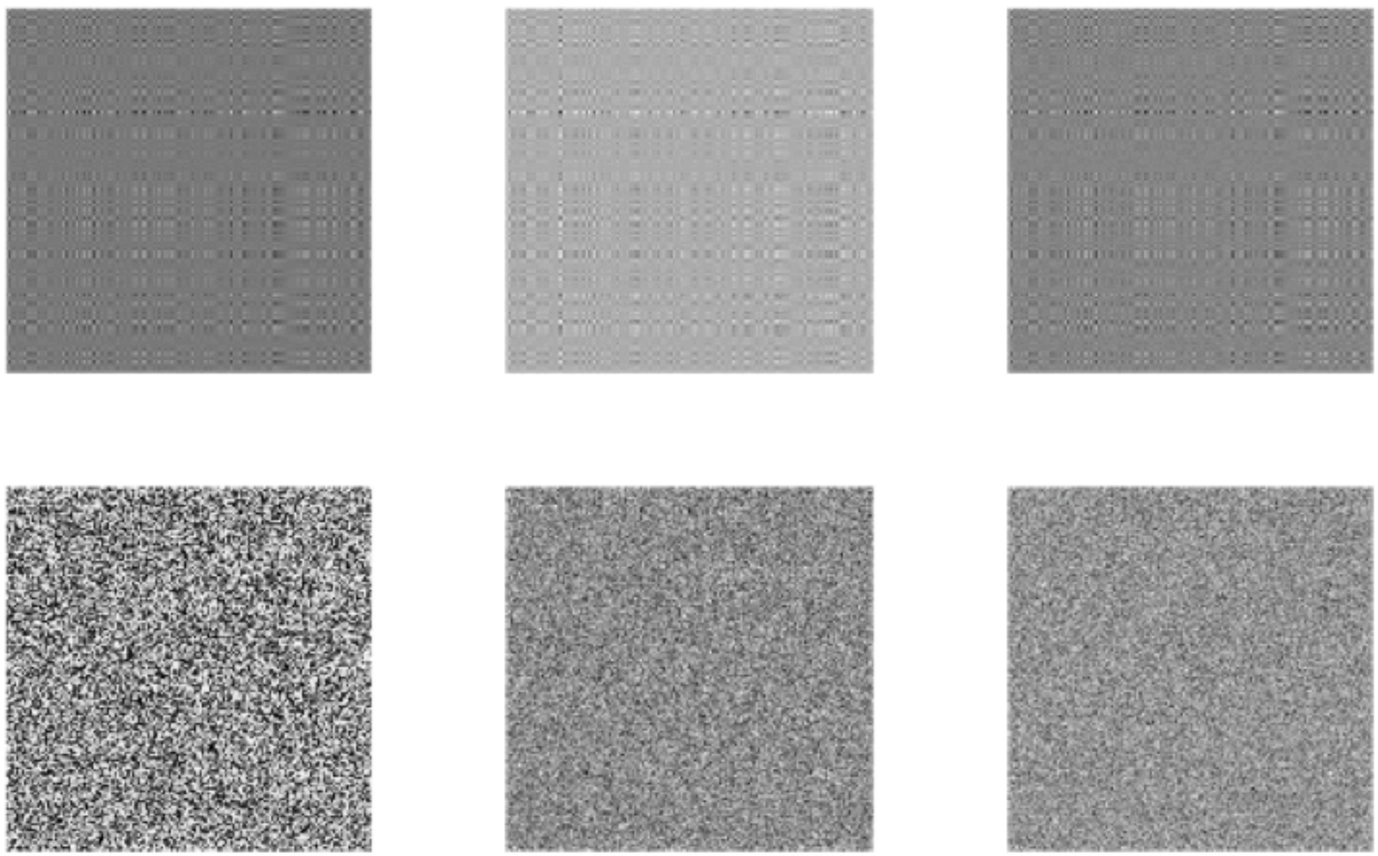
```
Time: CPU 5.63 s, Wall: 7.68 s
```

```
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



Questions?