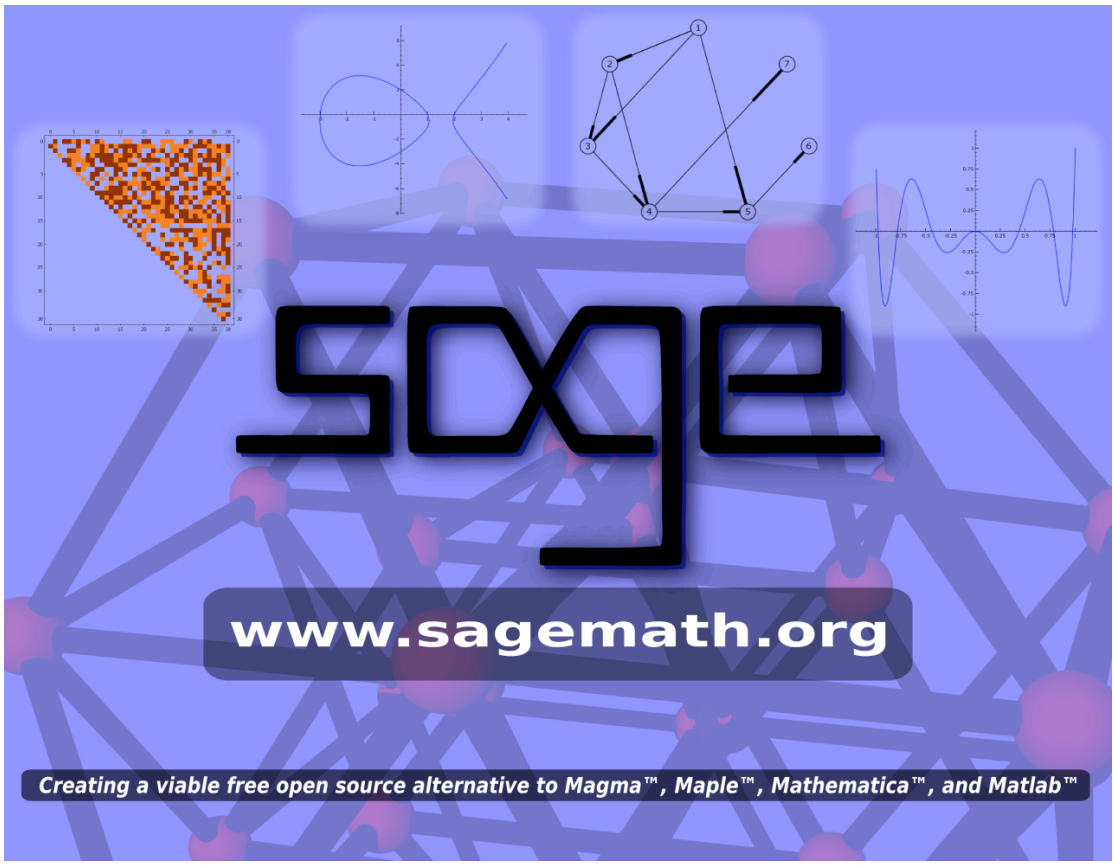
Boeing - June 12, 2008-stein

Sage: Open Source Mathematical Software




```
2 + 3
```

```
5
```

```

```

Features! Features! Features!

Sage can do *a lot*.

1. Numerical linear algebra, optimization (numpy, scipy, gsl)
2. Exact linear and commutative algebra (Linbox, IML, etc.)
3. Group theory
4. Number theory
5. Symbolic calculus
6. Coding theory
7. Cryptography and cryptanalysis
8. Graph theory
9. Combinatorics
10. 2d and 3d plotting
11. Statistics (Sage includes R)
12. Fast compiled code
13. Use Fortran in the notebook
14. Everything is 100% open source and free (just like Linux), so you can **integrate** it into your research environment however you want and change everything.

Reference Manual (over 3000 pages, all new)

A Few Examples...

```
a = random_matrix(ZZ,200)
time a.determinant()
```

```
-89133357130354631419071314971436546985412663987228808818909946512529522249244596160647528631027248561963138672371026803592606454021955619157625512881347758014703226397815191515766045749618376877637483611507388040031694796785423751348994929917917177154246169186600907216608991100943532799316230940777424038448147994523042896425454003419728338126710617488140465528342971655392964422132474573159112945004270729275133364010246895776749601353239922868334741851370
Time: CPU 0.24 s, Wall: 0.25 s
```

```
a = random_matrix(RDF,1000)
time a.LU()
```

```
(1000 x 1000 dense matrix over Real Double Field, 1000 x 1000 dense
matrix over Real Double Field, 1000 x 1000 dense matrix over Real
Double Field)
Time: CPU 1.67 s, Wall: 1.73 s
```

```
factor(902380928340982309489203840283409823094 2304)
```

```
2^5 * 967 * 11339824399261558063 * 25716219628037168507
```

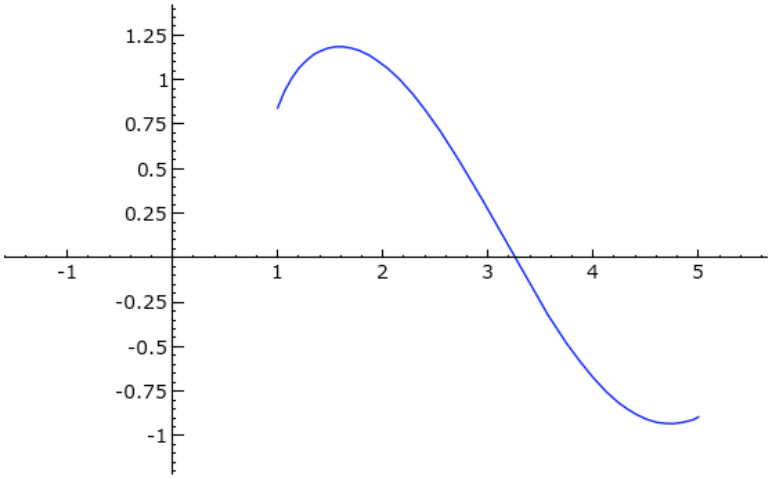
```
time p = number_of_partitions(10^8)
p.ndigits()
```

```
11132
CPU time: 3.83 s,  Wall time: 3.95 s
```

```
var('x')
show( integrate(sin(x) + log(x)/x^2) )
```

$$\frac{-\log(x)}{x} - \cos(x) - \frac{1}{x}$$

```
plot(sin(x)+log(x)/x^2, 1,5)
```




```
# symbolic
a = log(2); a
```

```
log(2)
```

```
# numerical (straight call to c library)
a = math.log(2); a
```

```
0.69314718055994529
```

```
def class_hierarchy(cls, v):
   v.append(str(cls))
   for supercls in cls.__bases__:
       class_hierarchy(supercls, v)

@interact
def _(object=1):
   print '<html>'
```

```
   print '<h1>Inheritance hierarchy of %r</h1>'%object
   print '<font color="#333333"><pre>'
   v = []
   class_hierarchy(object.__class__, v)
   print '\n'.join(['.'*(3*i)+w for i, w in
enumerate(reversed(v))]).replace('<','&lt;')
   print '</pre></font></html>'
```

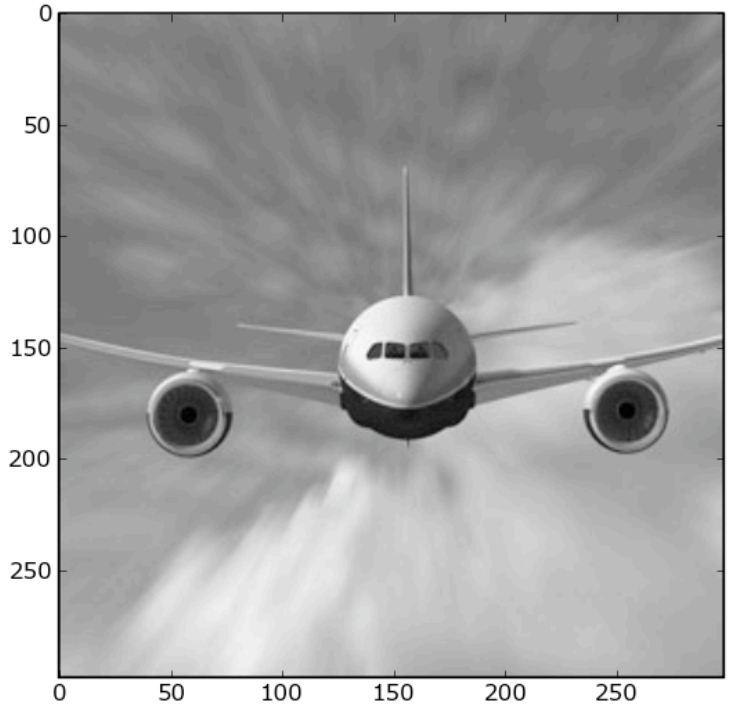
object 1

Inheritance hierarchy of 1

```
<type 'object'>
...<type 'sage.structure.sage_object.SageObject'>
......<type 'sage.structure.element.Element'>
.........<type 'sage.structure.element.ModuleElement'>
............<type 'sage.structure.element.RingElement'>
...............<type 'sage.structure.element.CommutativeRingElement'>
..................<type 'sage.structure.element.IntegralDomainElement'>
.....................<type 'sage.structure.element.DedekindDomainElement'>
........................<type 'sage.structure.element.PrincipalIdealDomainElement'>
...........................<type 'sage.structure.element.EuclideanDomainElement'>
..............................<type 'sage.rings.integer.Integer'>
```

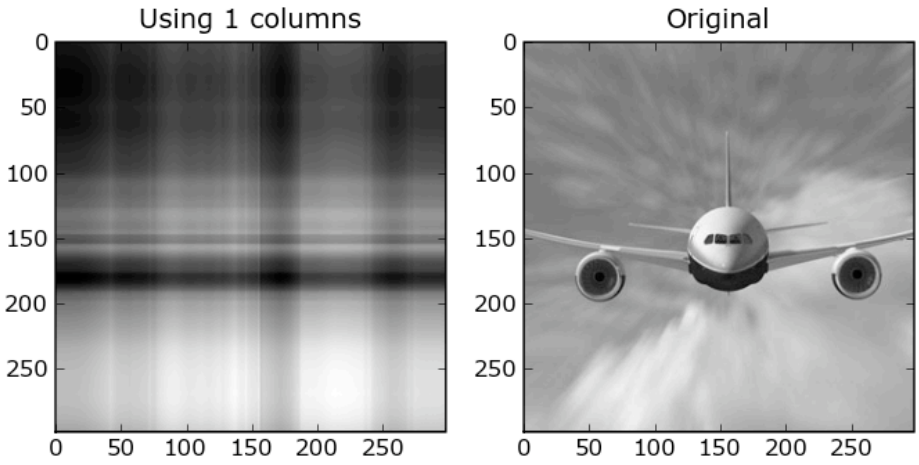
```
#auto
import pylab
A = pylab.imread(DATA + 'boeing.png')
Agray = pylab.mean(A,2)
pylab.imshow(Agray)
pylab.gray()
pylab.savefig('orig.png')
```

CPU time: 0.74 s, Wall time: 1.73 s




```
%hide  #code to implement using SVD
def svd_image(Agray,i):
siz = pylab.shape(Agray)
       u,s,v = pylab.linalg.svd(Agray)
       Anew = s[0]*pylab.outer(u[0:,0],v[0,0:])
for j in range(i):
           uu = u[0:,j+1]; vv = v[j+1,0:]
           op = pylab.outer(uu,vv)
           Anew = copy(Anew) + pylab.dot(s[j+1],op)
       pylab.figure()
       pylab.subplot(121)
       pylab.imshow(Anew)
       pylab.gray()
       pylab.title('Using '+str(i+1)+ ' columns')
       pylab.subplot(122)
       pylab.imshow(Agray)
pylab.gray()
       pylab.title('Original')
pylab.savefig('orig.png')
       return Agray, Anew
@interact
def energy_in_svd(i=slider(0,100,1)):
   svd_image(Agray,i)
```

i

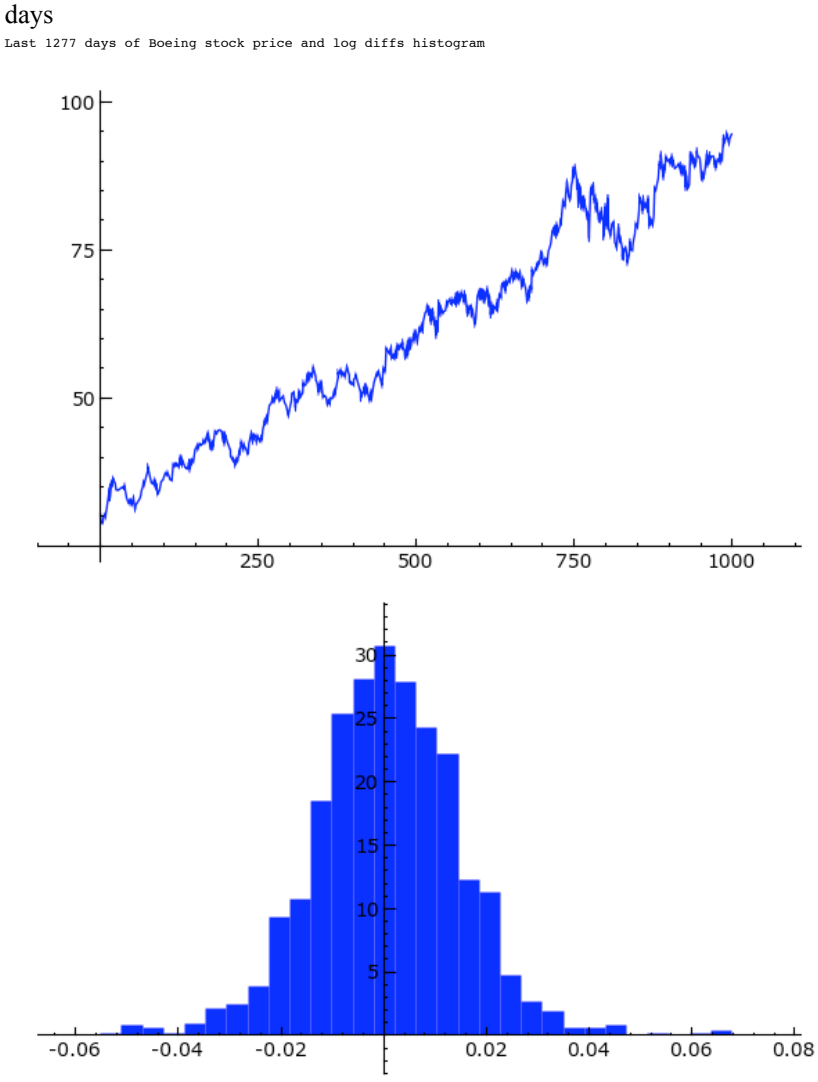



```
#  Steiner surface/Roman's surface with embedded sphere
u, v = var('u,v')
fx = sin(2*u) * cos(v) * cos(v); fy = sin(u)*sin(2 * v)
fz = cos(u) * sin(2*v)
(sphere((0,0,0), 0.7, opacity=0.5) +
parametric_plot3d([fx, fy, fz], (u, -pi/2, pi/2), (v, -pi/2,pi/2),
frame=False, color="red"))
```

```
# 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)
```

```
#S = finance.Stock('ba').close()
S = load(DATA+'ba.sobj')
```

```
@interact
def _(days=(1300,(20..len(S)))):
   print "Last %s days of Boeing stock price and log diffs
histogram"%days
   S[-days:].plot().show()
   S[-days:].log().diffs().plot_histogram(30).show()
```



Sage is Fast: Cython

Robert Bradshaw will talk more about this later...

```
def mymean(n):
   s = 0
   for m in range(n):
       s += m
```

```
   return float(s) / n
```

```
time mymean(10^6)
```

```
    499999.5
    CPU time: 1.20 s,  Wall time: 1.22 s
```

```
%cython
def mymean2(int n):
   cdef double s = 0
   cdef int m
   for m from 0 <= m < n:
       s += m
   return s / n
```

__Users_wa..._code_sage269_spyx.c __Users_wa...age269_spyx.pyx.ht

```
time mymean2(10^6)
```

```
    499999.5
    CPU time: 0.00 s,  Wall time: 0.00 s
```

```
timeit('mymean2(10^6)')
```

```
    625 loops, best of 3: 1.19 ms per loop
```

```
1.2/(1.2*10^(-3))
```

```
    1000.00000000000
```

```
v = finance.TimeSeries([1..10^6])
timeit('v.standard_deviation()')   # straight Cython -- way faster
than R/scipy/matlab...
```

```
    125 loops, best of 3: 3.3 ms per loop
```

```
import scipy.stats
w = v.numpy()
timeit('scipy.stats.std(w)')
```

```
    25 loops, best of 3: 35.1 ms per loop
```

```
```

```
```

You Can Use Java From Sage

```
from jpype import java, startJVM
startJVM('/System/Library/Frameworks/JavaVM.framework/Versions/1.5.0/L
```

```
java.lang.System.out.println('Hello Boeing!')
```

```
Hello Boeing!
```

This uses the JNI (Java Native Interface).

For more, see http://jpype.sourceforge.net...

History

- *I started Sage* at Harvard in January 2005, after working in computational number theory for nearly a decade using C++ and Magma.
- No existing mathematical software was technically good enough for what I **needed** for my research and teaching.
- Sage-1.0 released February 2006 at Sage Days 1 (San Diego).
- Sage Days 2, 3, 4, 5, 6, 6.5, 7, 8. At UCLA, UW, Cambridge, Bristol, Austin, etc. Sage DAys 9,10,11, 12 coming up fast.
- Funding from NSF, DoD, Microsoft, private donors, etc. About 100 contributors.

Sage from an Engineer's Perspective

- Sage does math **ALSO**
- Sage is easily integrated with ***anything***
  - Python was designed as a glue language
  - Open Source - Hack the code
  - Cython makes short work of wrapping C/C++ (and .o, .so, etc.)

Glue It All Together With Python

Guido van Rossum
CNRI
1895 Preston White Drive
Reston, VA 20191
Email: guido@cnri.reston.va.us, guido@python.org

Position paper for the OMG-DARPA-MCC **Workshop on Compositional Software Architecture** in Monterey, California, January 6-8, 1998.

Introduction

Python is an advanced scripting language that is being used successfully to glue together large software components. It spans multiple platforms, middleware products, and application domains. Python is an object-oriented language with high-level data structures, dynamic typing, and dynamic binding. Python has been around since 1991, and has a very active user community. For more information, see the Python website http://www.python.org.

Python

- "Python is fast enough for our site and allows us to *produce maintainable features in record times*, with a minimum of developers," said Cuong Do, Software Architect, YouTube.com.

- "Google has made no secret of the fact they use Python a lot for a number of internal projects. Even knowing that, once *I was an employee, I was amazed at how much Python code there actually is in the Google source code system*.", said Guido van Rossum, Google, creator of Python.

- "Python plays a key role in our production pipeline. Without it a project the size of *Star Wars: Episode II* would have been very difficult to pull off. From crowd rendering to batch processing to compositing, *Python binds all things together*," said Tommy Burnette, Senior Technical Director, Industrial Light and Magic.