Scipy 2008 -- Sage

William Stein

**(I'm a Number Theory Professor in the
"Pure" Mathematics Department
at University of Washington, but I was a CS undergrad.)**

Scipy 2008

Acknowledgement:

I would like to profusely thank the organizers of Scipy 2008. They've done a superb job putting together this conference and making it an enjoyable, productive, and exciting experience.

Observation: Scipy/Numpy/Enthought ***does not meet the needs*** of many users of mathematics software (otherwise, a lot less people would use Matlab, Mathematica and Maple).

> From IRC last night:
>
> Sage and the notebook are *very* cool; they beg experimentation in a way that "make sure you have x, y, and z installed also if you want to see any graphs" doesn't.
>
> Making numerical solutions of ODEs as easy (or easier) in Sage as is in Mathematica would be a dream for me and I've been wondering about making the CAS do more work to put things in a form that they can be farmed out to specialized code.
>
> With scipy's numerical solver, you need to put things in a very special form which involves tedious algebra, which is why we have CAS's.

> **Sage Mission Statement:**
> Create a viable open source free

> alternative to Magma, Maple, Mathematica, and Matlab.

Thus our stated goal is to meet the needs of Magma, Maple, Mathematica, and Matlab users. Enthought's approach can **never** meet the needs of this range of users because of their (very puzzling to me) refusal to use GPL'd code. Sage can.

History of Sage

1. **Berkeley 1995-2000:** I did my Ph.D. in Number Theory (Modular Forms) under Hendrik Lenstra, and wrote a massive amount of C++ code, then moved entirely to Magma.

2. **Harvard 2000-2003:** I developed a lot of code in Magma, and convinced many people in my area to use Magma.

3. **Harvard 2003-2005:** I realized that I had made a **major mistake** in embracing the closed Magma system as my core research tool,

4. **Harvard late 2004:** I get tenure at UCSD and start investigating nontrivial options.

5. **Harvard, January 2005:** I release Sage-0.1 and a long lonely year of very hard work follows.

6. **UC San Diego, February 2006:** Sage-1.0 and Sage Days 1. Get some help from David Joyner and David Kohel.

7. **University of Washington, October 2006:** Sage Days 2; Start build a developer community that now numbers well over 100 and is writing **massive amounts of new very high quality code**.

8. **November 2007:** Sage wins first prize in Scientific Category of Trophees du Libre (3000 euros, a laptop, etc.) Developer base grows and matures substantially.
9. **NOW:** An average of 800 new visitors to the website each day. Well over 1000 complete downloads a month.

What is Sage?

1. A **completely free distribution** of the world's best open source mathematical software that builds easily from source on Linux and OS X (and soon on Solaris and Microsoft Windows). Basically Sage **includes nearly everything** you need

standard.

2. Over three hundred thousand lines of **new code and documentation** that implements large amounts of new functionality and provides a graphical interface. A massive automated test suite.

3. **Unified interfaces to Magma, Mathematica, Maple,** Matlab, Axiom, Mupad, Singular, Macaulay2, etc., so it is easier for you to use all these programs together (for example, when migrating to Sage or making sure your new Sage code is faster than anything else out there).

3D Visualization

```
graphs.CircularLadderGraph(12).plot()
```

```
graphs.CircularLadderGraph(12).plot3d(zoom=1.1)
```

```
var('x,y')
f = sin(x^2+y^2)/(x^2+y^2)
icosahedron(size=pi/2.1,color='red') + \
   plot3d(f, (x,-pi,pi), (y,-pi,pi), opacity=0.85, zoom=1.2)
```

```
# 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=Color('#00aa00')) +
IndexFaceSet(F4,V,color=Color('#00aa00'))
Y = Y.rotateX(-1)
Y.show(aspect_ratio=[1,1,1], frame=False, figsize=4)
```

```
@interact
def _(points=selector([5..10], buttons=True, default=7),
     interpolation_points=(15..50),
     mesh=False, raytrace=False, seed=(18,(0..99))):
   print 'This uses Java 2D (jmol) and scipy.interpolate to draw a surface
through some random points.'
   set_random_seed(seed)
   show(list_plot3d(random_matrix(RDF, points), interpolation_type='nn',
        num_points=interpolation_points,
        texture='red', frame_aspect_ratio=[1,1,1/2], mesh=mesh,
        viewer='tachyon' if raytrace else 'jmol'))
```

points 5 6 7 8 9 10

interpolation_points

mesh ☐

raytrace ☐

seed

```
This uses Java 2D (jmol) and scipy.interpolate to draw a surface
through some random points.
```

CPU time: 0.01 s, Wall time: 0.50 s

Cython and Fortran

We make massive use of Cython to implement Sage. Over 100000 lines of the new Sage library code is written in Cython.

```
%cython

cdef inline double f(double x):
   return x*x*x - 3*x

def integrate_f(double a, double b, int N):
   cdef double s = 0
   cdef double dx = (b-a)/N
   cdef int i
   for i in range(N):
       s += f(a+i*dx)
   return s * dx
```

```
print integrate_f(float(1),float(20),10^4)
timeit('integrate_f(float(1),float(20),10^4)')
```

```
39393.7054601
625 loops, best of 3: 72.7 Âµs per loop
```

```
reset()
```

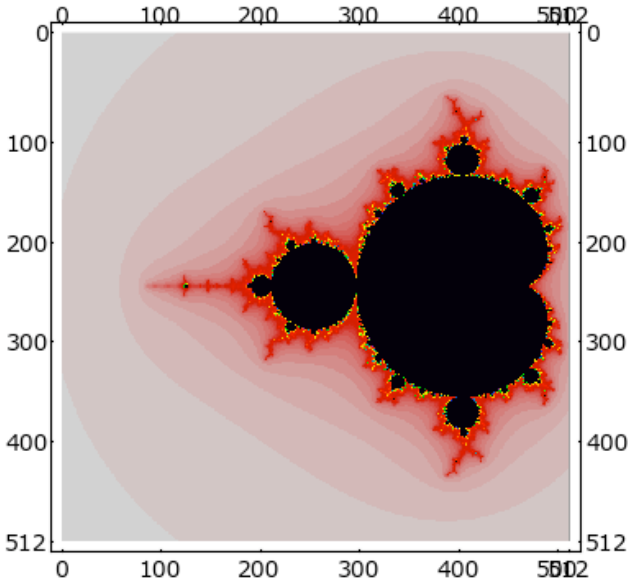
```
%fortran

      SUBROUTINE MANDELBROT(BIT)
      INTEGER I, J
      INTEGER ITICS0, ITICS1
      REAL CPTIM
      REAL BIT(513,513)
cf2py intent(out) bit
      REAL COUNT
      REAL AA, BB, D
      COMPLEX Z, CC, NEWZ
      DATA NEWZ /(0.0,0.0)/, BB /2.5/, D /1.5/

c mandelbrot set test
      AA = 3.0 / FLOAT(512)
      DO 30 I = 1, 512
       DO 20 J = 1, 512
        CC = CMPLX(AA*FLOAT(I)-BB,AA*FLOAT(J)-D)
        Z = NEWZ
        COUNT = 256
  10    Z = Z * Z + CC
        COUNT = COUNT - 1
        IF (COUNT.GT.0.AND.CABS(Z).LT.4) GOTO 10
        BIT(J,I)=COUNT+1
  20   CONTINUE
  30  CONTINUE
      AV = 0.0
      DO 50 I = 1, 512
```

```
      DO 40 J = 1, 512
       AV = AV + ALOG(1+ABS(BIT(I,J)))
  40  CONTINUE
  50 CONTINUE

     CPTIM = FLOAT(ITICS1-ITICS0) / 100.0

     END
```

```
None
```

```
time n = mandelbrot()
n.shape=(513,513)
matrix_plot(matrix(RDF,n),cmap='spectral')
```

```
CPU time: 1.33 s,  Wall time: 2.34 s
```

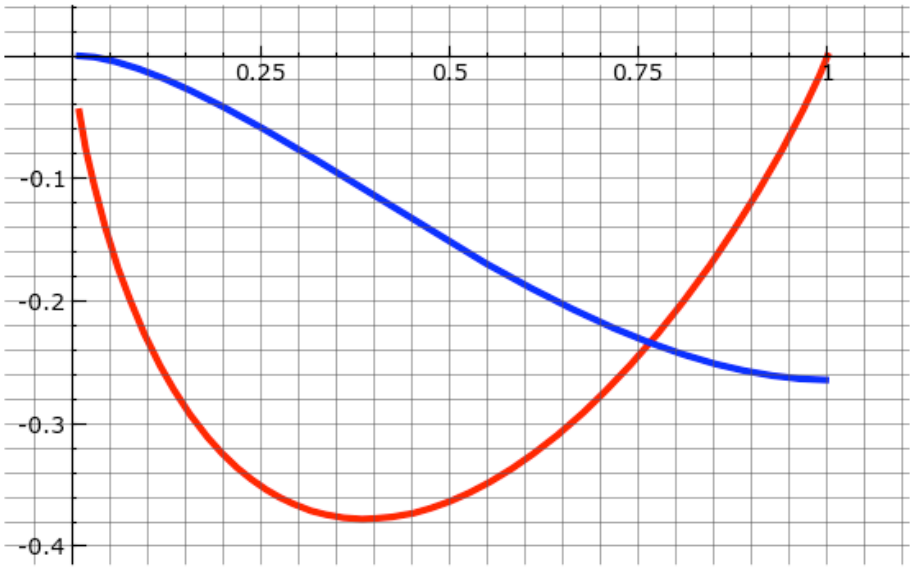


Integrating and Plotting Functions

```
var('x')
f = asin(x) * log(x)
g = integral(f)
show(g)
```

$$\log\left(\begin{matrix} 2\sqrt{1-x^2} \\ |x| \end{matrix} + \begin{matrix} 2 \\ |x| \end{matrix}\right) + \sin^{-1}(x)\,(x\log(x) - x) + \sqrt{1-x^2}\log(x) - 2\sqrt{1-x^2}$$

```
# Sage has a Mathematica-like plotting, built on top of matplotlib.
G = plot(f, (0,1), rgbcolor='red', thickness=3)
G += plot(g - g(0.01), (0,1), thickness=3)
G.show(gridlines='minor')
```




```
# Very fast float evaluation, even of the very complicated expression g above.
fast = g._fast_float_(x)
```

```
z = float(1)
timeit('g(z)')
timeit('fast(z)')
```

```
125 loops, best of 3: 3.09 ms per loop
625 loops, best of 3: 835 ns per loop
```

```
timeit('2 + 2', preparse=False)
```

```
625 loops, best of 3: 35.1 ns per loop
```

```
3.0/(919*10^(-6))
```

```
3264.41784548422
```

```
time integral_numerical(fast, 0, 1)
```

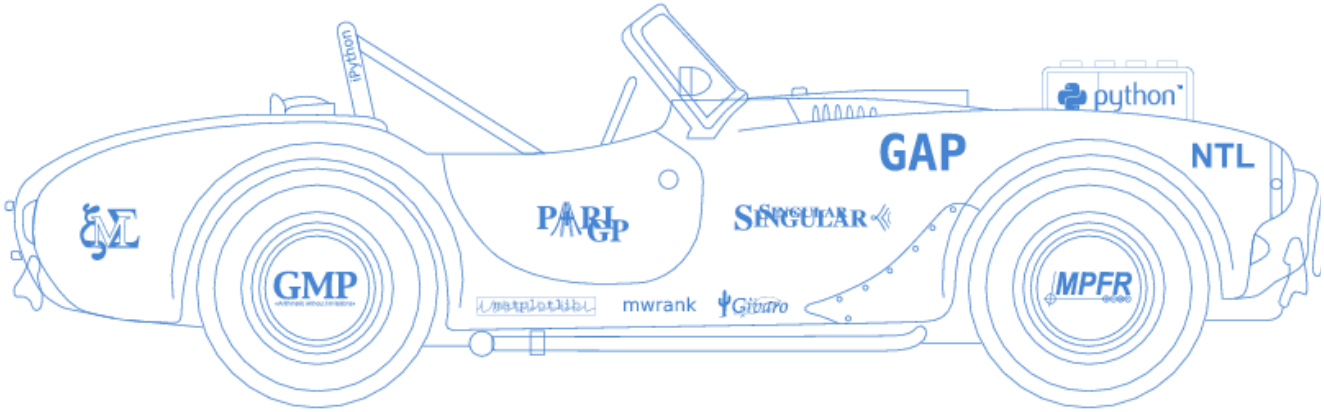
```
(-0.75714832582414338, 2.8943543802338329e-11)
CPU time: 0.00 s,  Wall time: 0.00 s
```

```

```

```

```

Interfaces

Sage has pseudo-tty based interfaces to everything; this make it is possible to make extensive use of Maple, Mathematica, Magma, Matlab, GAP, Maxima, Singular, PARI, and many other systems from Sage.



»Every free computer algebra system I've tried ha
reinvented many times the wheel without being able to build the car.

Make it ***much easier*** to do computations using multiple math software systems (a good idea; see Alex Marteli's talk).

```
maxima.eval('2 + 3')
```

```
'5'
```

```
a = maxima('5'); a
```

```
5
```

```
a.integrate('x')
```

```
5*x
```

```
a.integrate
```

```
<bound method MaximaElement.integral of 5>
```

```
type(a)
```

```
<class 'sage.interfaces.maxima.MaximaElement'>
```

```
a.name()
```

```
'sage0'
```

```
a = mathematica('5'); a
```

```
5
```

```
type(a)
```

```
<class 'sage.interfaces.mathematica.MathematicaElement'>
```

```
a.name()
```

```
'sage0'
```

```
a.Integrate(x)
```

```
5*x
```

```
x = var('x')
f = sin(x^2) + sqrt(3)/pi^e
show(f)
```

$$\sin\left(x^2\right) + \frac{\sqrt{3}}{\pi^e}$$

```
m = maxima(f); m
```

```
sin(x^2)+sqrt(3)/%pi^%e
```

```
m.name()
```

```
'sage4'
```

```
m = mathematica(f)  # requires mathematica!!
m
```

```
Sqrt[3]/Pi^E + Sin[x^2]
```

```
```

WARNING -- kids, don't try this at home: It's a license violation to use Mathematica from Sage via the notebook, since the EULA for Mathematica explicitly disallows any use of Mathematica via http.

```
os.system('ps ax |grep Mathematica.app')
```

```
type(m)
```

```
m.name()
```

```
m.Integrate(x)
```

```
reset('r')
```

```
z = r('z <- array(1:20, dim=c(4,5))')
z
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20
```

```
type(z)
```

```
<class 'sage.interfaces.r.RElement'>
```

```
z.mean()
```

```
[1] 10.5
```

```
z.sd()
```

```
[1] 1.290994 1.290994 1.290994 1.290994 1.290994
```

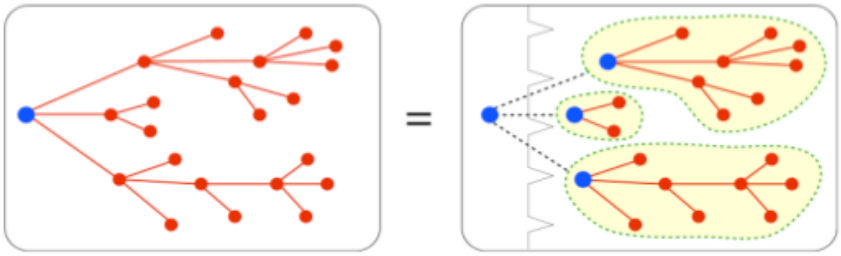
```
M = octave('rand(4)'); M
```

```
      0.230727 0.120852 0.360582 0.596342
      0.94515 0.91126 0.481142 0.481398
      0.188736 0.0326471 0.158264 0.694403
      0.959683 0.763152 0.339965 0.856096
```

```
M.eig()
```

```
      (2.12475,0)
      (-0.00565625,0.252413)
      (-0.00565625,-0.252413)
      (0.0429078,0)
```

Combinatorics



1. Sage is very strong in algebraic combinatorics. **The MuPAD-Combinat project** -- founded by Nicolas Thiery and Florent Hivert -- has officially decided to switch over to Sage, and have already ported much of their code over. Mike Hansen has done a massive amount of work on this.

2. Mike Hansen is porting Aldor-Combinat to Sage.

3. Sage has a **massive amount of graph theory code**, including the only open source implementation of computation of automorphism groups of graphs (Robert Miller's), a large database (Jason Grout and Emily Kirkman), and much much more.

4. Sage makes extensive use of Symmetrica, Gap, Gfan, and NetworkX.

```
G = graphs.CubeGraph(4); G
```

```
G.automorphism_group()
```

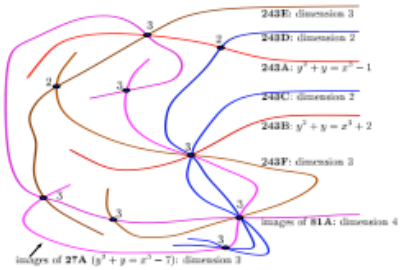
```
show(G)
```

```
G.plot3d(spin=True)
```

```

```

```

```

Number Theory



1. Sage has a wide range of number theory, including algebraic number fields, modular forms, special functions, way more elliptic curve functionality than any other open source program, and many elliptic and hyperelliptic curve functions that are available in no other programs.

2. Sage tightly builds on PARI, NTL, lcalc, Dokchiter, all of Cremona's code (both for curves and modular symbols), sympow, genus2reduction, and other number theory programs and libraries.

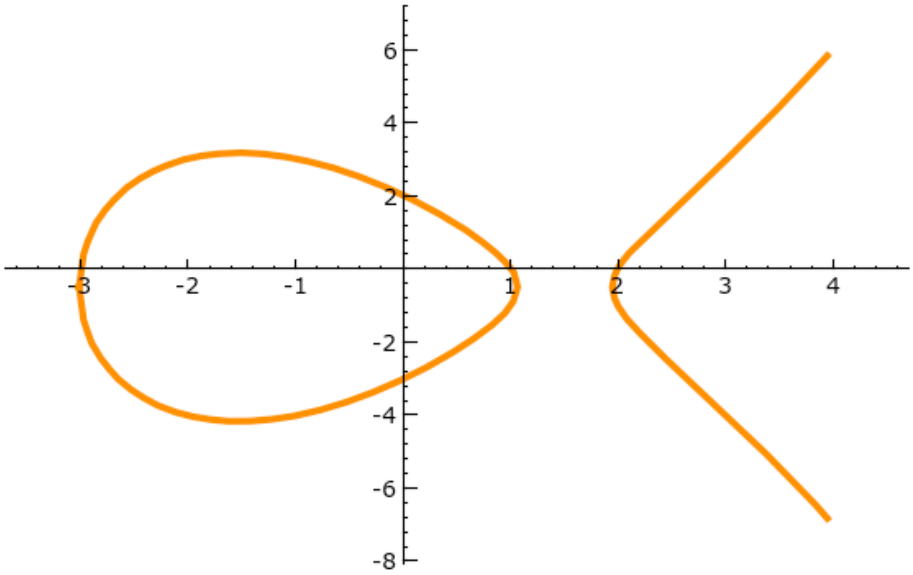
```
show(factor(29308203480928340982340820934820983402348))
```

```
# The only existing open source implementation of "finding
# all integral points on cubics", and is also the fastest money
# can buy!
```

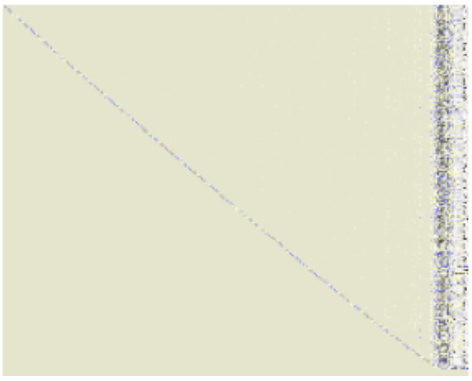
```
E = EllipticCurve('5077a')
html('<h1>$%s$</h1>'%latex(E))
print "ALL Integral Solutions:"
print E.integral_points()   # (new, thanks Nagel, Mardaus, and Cremona!!)
plot(E, thickness=3, rgbcolor='orange').show()
```

$$y^2 + y = x^3 - 7x + 6$$

```
ALL Integral Solutions:
[(-3 : 0 : 1), (-2 : 3 : 1), (-1 : 3 : 1), (0 : 2 : 1), (1 : 0 : 1),
(2 : 0 : 1), (3 : 3 : 1), (4 : 6 : 1), (8 : 21 : 1), (11 : 35 : 1),
(14 : 51 : 1), (21 : 95 : 1), (37 : 224 : 1), (52 : 374 : 1), (93 :
896 : 1), (342 : 6324 : 1), (406 : 8180 : 1), (816 : 23309 : 1)]
```



Exact Linear Algebra



1. Sage has optimized exact dense and sparse linear algebra over the rationals, integers, finite fields (especially GF(2)).
2. Sage contains a large amount of new linear algebra code tightly integrated with Linbox, IML, NTL, Numpy, M4RI, and ATLAS.

```
a = random_matrix(ZZ,200, x=-2^16,y=2^16)  # 16-bit entries
#set_verbose(2)
s = cputime()
d = a.determinant()      # Uses Linbox, ATLAS, IML, and new Sage code.
print d
print "time: ", cputime(s)
#set_verbose(0)
```

```
-27400720853998693305091279018178326437040083072654254754258644334 05\
52108476683749531745317928694598262752028176853601608448721687728805\
71665877393090307312467466400278047826210376409534090623399789725730\
89573252634928782881110671356022567094625616249847649200083905332289\
33590997603525271766307073309945870677719026518249874190487560515274\
86611464547292024855828204019133633588926713994970905086380307817818\
16942718471800299715205635595602103850573562536133856875155283966417\
06563947177911915017656210607392702782106480047360918613946692400165\
81155841494842925956948166029779491965801852940979652249891563674441\
24715524117958351758215551430764126103578376795579191094424385979440\
71214879649332975323126468571827210233434357117544160911789941157882\
00885493001145629586706865517864095731976452004857339497590737757279\
```

```
55546757645523809908989239824417205335883791318560918410421285997213\
85220878279742227101754961977489871899665345269332924009119107734604\
83923492121729525894380688150509719893797937939018007429951180176193\
85877822156911349619233228524771104146271862305041508197859917646360\
0127564131086789
time:  0.328561
```

```
magma.quit()
b = magma(a)
t = magma.cputime()
d2 = b.Determinant()
print 'Time: ', magma.cputime(t)
```

```
Time:  0.7
```

```
d == d2
```

```
True
```

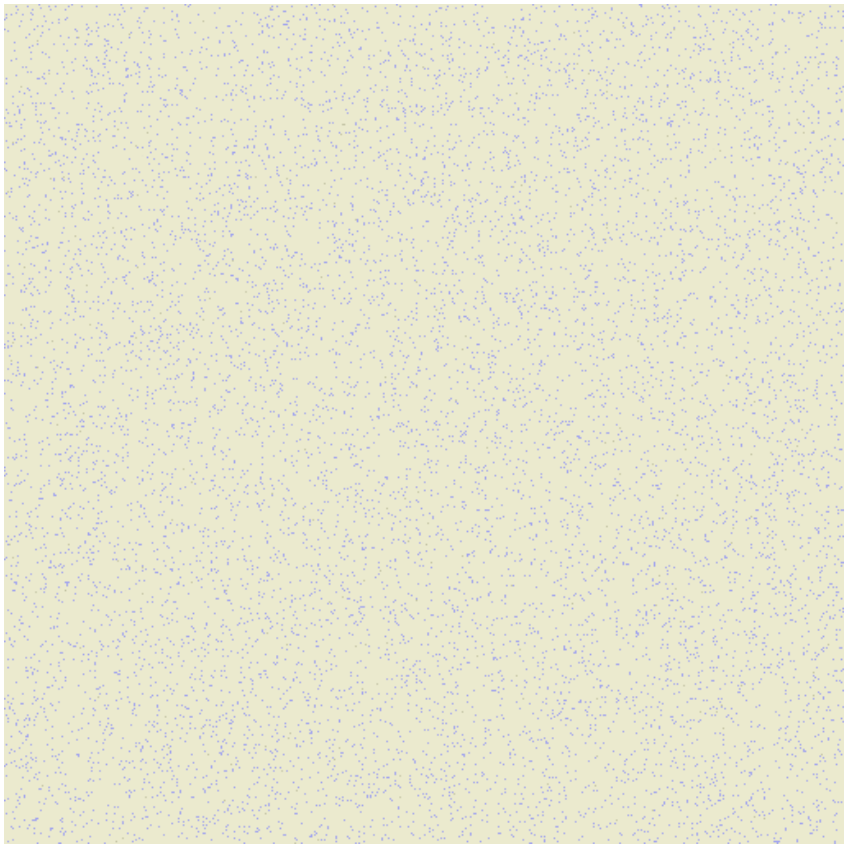
```
b = mathematica(a)
```

```
mathematica.eval('Timing[Det[%s];]'%b.name())
```

```
{13.9828, Null}
```

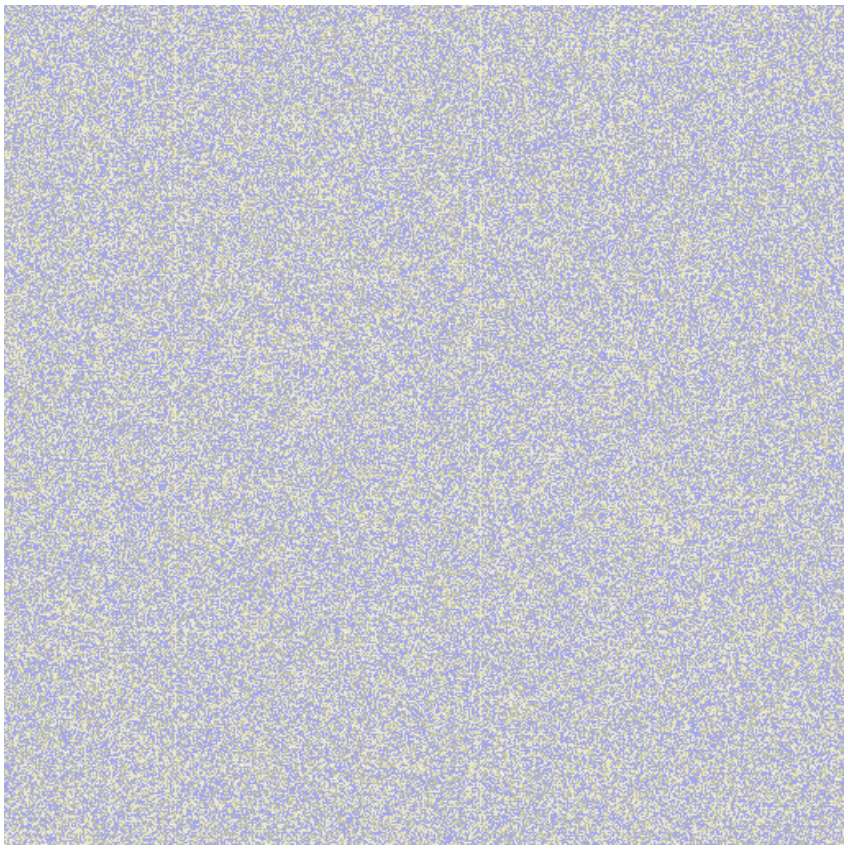
```
A = random_matrix(GF(2),5000, density=0.001)
```

```
A.visualize_structure()
```



```
time B = A^5    # A is a 5000x5000 sparse matrix over GF(2)
B.visualize_structure()
```

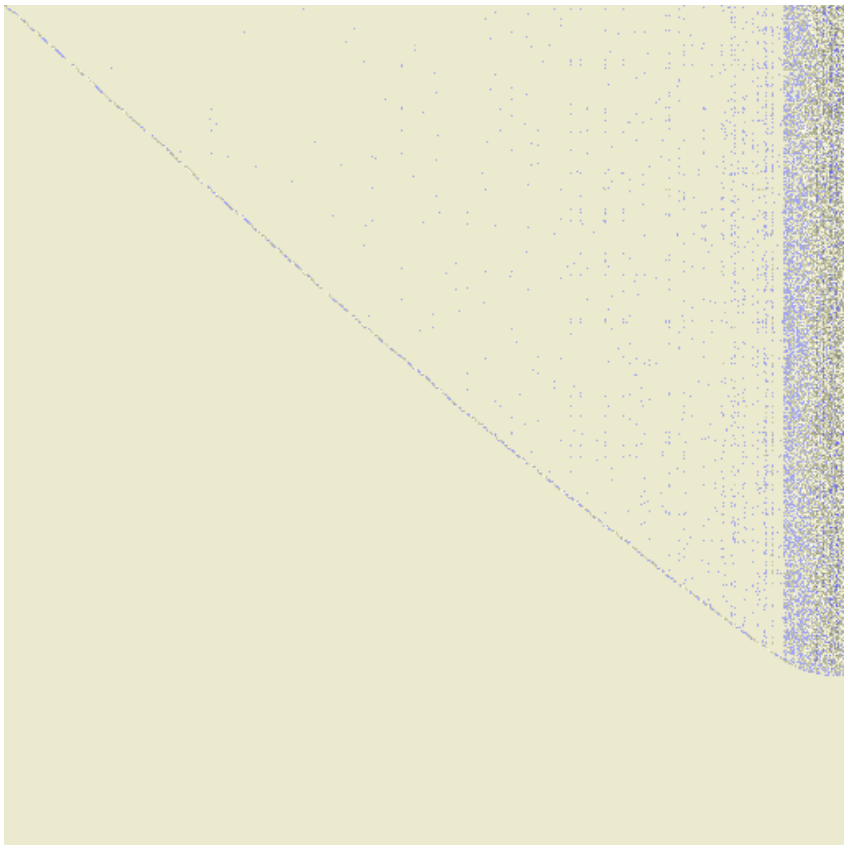
```
CPU time: 6.53 s,  Wall time: 6.88 s
```



```
time B.echelonize()
```

```
    CPU time: 3.52 s,  Wall time: 3.72 s
```

```
B.visualize_structure()
```



Numerical Computation and Statistics

1. Sage includes Numpy, Scipy, CVXOpt, R (and rpy), and some new code for quantative finance.

```
import scipy.stats
scipy.stats.std([1..1000])
```

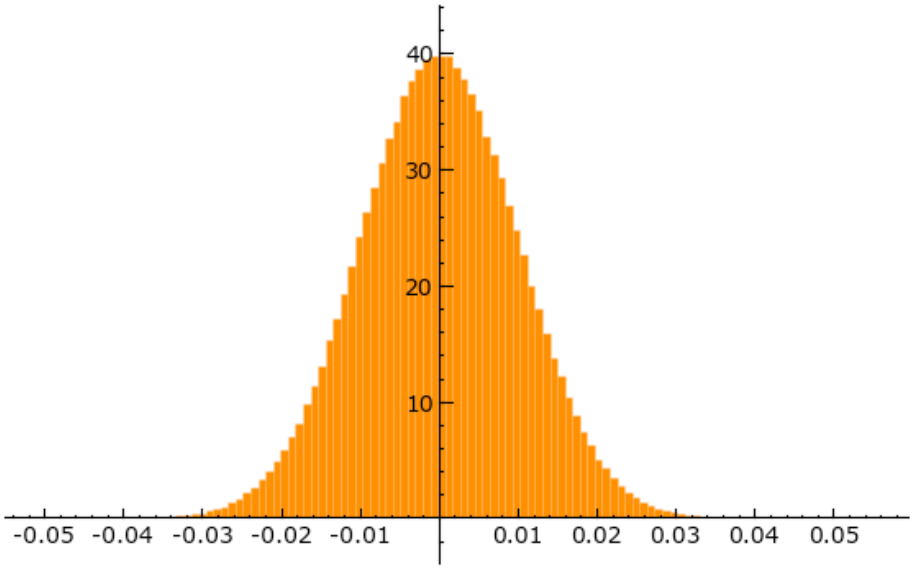
```
finance.TimeSeries([1..1000]).standard_deviation()
```

```
import rpy
print rpy.std([1..1000])
```

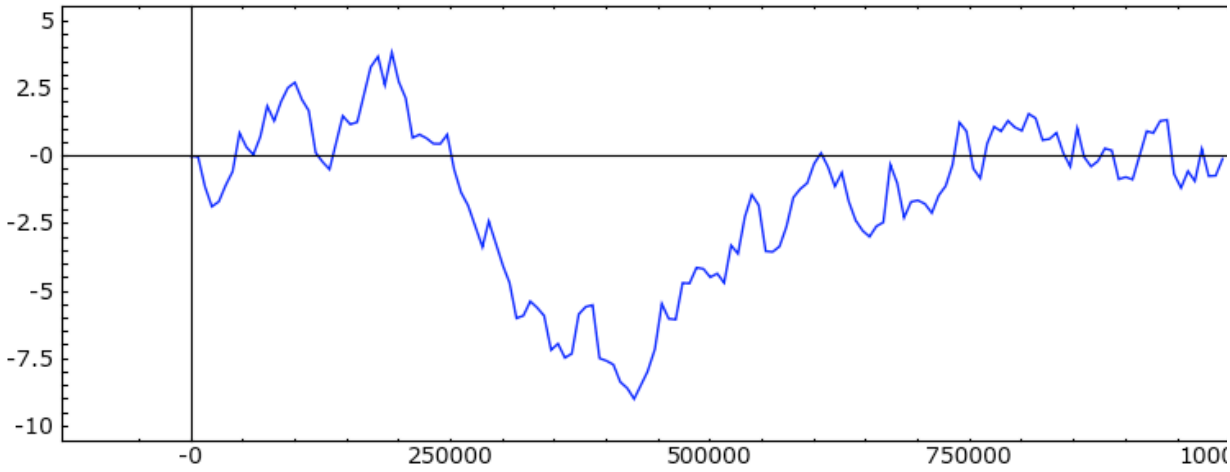
```
# pexpect r interface
sage.interfaces.all.r([1..1000]).sd()
```

```
time v = finance.TimeSeries(10^6).randomize('normal', 0, 0.01)
v.plot_histogram(100, rgbcolor='orange')
```

```
CPU time: 0.58 s,  Wall time: 0.64 s
```




```
v.sums().plot(150).show(frame=True, figsize=[9,3])
```




```
# This combines scipy's fft code and Sage's new stats code.
v = finance.fractional_brownian_motion_simulation(0.8,0.01,300,3)
line3d([(v[0][i],v[1][i],v[2][i]) for i in range(len(v[0]))], color='purple', thickness=3)
```

Nearly Every Imaginable Number System

```
K.<i,j,k> = QuaternionAlgebra(QQ,-1,-1)
print K
i + (i+j+k)^2
```

```
S.<X,Y,Z> = PolynomialRing(GF(9,'a')); S
```

```
# insanely fast (we wrote a C++/Cython interface to Singular, which had
# never been usable in library mode before)
f = (X+Y+Z+1)^20
time g = f*(f+1)
```

```

```

```

```

Interval Arithmetic

We also have rigorous high precision interval arithmetic based on MPFR and MPFI, which works **exactly the same on any OS/hardware**. But our intervals do not work when their are multiple disjoint parts, so we might want to create an optimized Cython-ized version of Stefano's interval package for Sage (For basic arithmetic, it seems that Stefano's interval package is over 50 times slower than Sage's.)

```
RealNumber = float
def f(x,y):
   return ((333.75-x**2)* y**6 + x**2 * (11* x**2 * y**2-121 * y**4 -2) +
              5.5 * y**8 + x/(2*y))
f(77617,33096)
```

```
-1.1805916207174113e+21
```

```
@interact
def _(bits=(53..200)):
   RealNumber = RealField(bits)
   def f(x,y):
       return ((333.75-x**2)* y**6 + x**2 * (11* x**2 * y**2-121 * y**4 -2) +
                  5.5 * y**8 + x/(2*y))
   print f(77617,33096)
```

bits

```
1.1726039400531786318588349045202
```

```
@interact
def _(bits=(100..130)):
   RealNumber = RealIntervalField(bits)
   def f(x,y):
       return ((333.75-x**2)* y**6 + x**2 * (11* x**2 * y**2-121 * y**4 -2) +
                  5.5 * y**8 + x/(2*y))
   print f(77617,33096)
```

bits

```
-0.827396059946821368141165095479816292 0?
```

```
RealNumber = float
def f1(x):
   return sqrt(x+1) - sqrt(x)
f1(9876543.21)
```

```
RealNumber = RealIntervalField(53)
def f1(x):
   return sqrt(x+1) - sqrt(x)
f1(9876543.21)
```

```
reset('RealNumber')
```

```
random_matrix(RIF, 10)
```

```
[-2.0000000000000000?              0.?e-17  1.0000000000000000?
0.?e-17 -2.0000000000000000?              0.?e-17
1.0000000000000000?  2.0000000000000000?  2.0000000000000000?
2.0000000000000000?]
[ 1.0000000000000000? -1.0000000000000000?  1.0000000000000000?
-1.0000000000000000?  1.0000000000000000?  2.0000000000000000?
-2.0000000000000000?              0.?e-17              0.?e-17
-1.0000000000000000?]
[             0.?e-17  1.0000000000000000?              0.?e-17
2.0000000000000000?  1.0000000000000000? -2.0000000000000000?
-1.0000000000000000?  2.0000000000000000?  1.0000000000000000?
-2.0000000000000000?]
[             0.?e-17  2.0000000000000000?              0.?e-17
-2.0000000000000000?  2.0000000000000000?  2.0000000000000000?
-1.0000000000000000?  2.0000000000000000? -2.0000000000000000?
2.0000000000000000?]
[-1.0000000000000000?              0.?e-17 -2.0000000000000000?
0.?e-17 -1.0000000000000000?              0.?e-17
-1.0000000000000000? -1.0000000000000000? -1.0000000000000000?
0.?e-17]
[-2.0000000000000000?  2.0000000000000000? -2.0000000000000000?
0.?e-17 -2.0000000000000000?  1.0000000000000000?
-2.0000000000000000?  2.0000000000000000? -1.0000000000000000?
1.0000000000000000?]
[-1.0000000000000000? -1.0000000000000000? -2.0000000000000000?
-2.0000000000000000? -1.0000000000000000? -1.0000000000000000?
```

```
2.0000000000000000?              0.?e-17 -1.0000000000000000?
-2.0000000000000000?]
[ 1.0000000000000000? -1.0000000000000000?              0.?e-17
2.0000000000000000?              0.?e-17 -2.0000000000000000?
1.0000000000000000?  1.0000000000000000?  2.0000000000000000?
0.?e-17]
[ 1.0000000000000000?              0.?e-17 -2.0000000000000000?
0.?e-17 -1.0000000000000000? -1.0000000000000000?
2.0000000000000000? -1.0000000000000000? -2.0000000000000000?
-1.0000000000000000?]
[             0.?e-17              0.?e-17 -2.0000000000000000?
-1.0000000000000000?  2.0000000000000000?  2.0000000000000000?
-2.0000000000000000?  2.0000000000000000?              0.?e-17
-1.0000000000000000?]
```

Hidden Markov Models

The next version of Sage will include GHMM with new Cython bindings, for very efficient computation with generalized hidden Markov models. GHMM is a C library developed by the Algorithmics group at the Max Planck Institute for Molecular Genetics over the last ten years. It's extremely useful in computational biology, machine learning, etc.

```
@interact
def foo(phrase = 'numpy scipy matplotlib sage', states=(3,(1..40)),
       output = (1000,(100..2000)),
       show_model=False, seed=(0..100)):
   print "Type in a some text.  This will train a Hidden Markov Model with %s states to generate
similar text."%states
   set_random_seed(seed)
   A = random_matrix(RDF, states).apply_map(abs)
   B = [[random() for _ in range(27)] for _ in range(states)]
   letters = [chr(97+i) for i in range(26)] + [' ']
   H = hmm.DiscreteHiddenMarkovModel(A,B, [1]*states, letters)
   H.normalize()
   phrase = (' ' + phrase + ' ').lower()
   phrase = [x for x in phrase if x in letters]
   try:
       H.baum_welch(phrase)
   except Exception, msg:
       pass
   print "Here is the text:\n\n"
   print ''.join(H.sample(output))
   print '\n'*2
   if show_model:
       show(matrix_plot(H.transition_matrix()))
       show(matrix_plot(H.emission_matrix()))
       dx = dict([ (i, [j for j in range(A.ncols()) if abs(A[i,j])>0.2])   for i in
range(A.nrows())])
       show(plot(Graph(dx)))
```

phrase: numpy scipy matplotlib sage

states

output

show_model

seed

```
Type in a some text.  This will train a Hidden Markov Model with 29
states to generate similar text.
Here is the text:


 numatplotplotplotplotplotplotlib satplotplotplotplotplotlipy sage
numage numpy scib mpy scipy matlib mpy scipy spy scipy spy scipy mpy
scib mpy scipy mpy scib matlib mpy scib mage spy scipy spy scib mpy
scipy sage numatlipy spy scipy matplotplotplotplotlib mage numatlipy
```

```
mpy scipy matlipy satplotlib matplotplotplotplotplotlipy mpy scib
mpy scib mage numage mpy scib spy scipy matlib mpy scipy matplotlipy
mage numpy scipy spy scipy mpy scib matplotlipy matlib
matplotplotlipy mpy scipy sage numatplotplotplotlipy mage numage
numpy scipy mpy scipy mpy scib matlipy spy scipy mage nusage
numatplotlib mpy scipy sage matlipy mage numatlib mpy scipy mage
numpy scib matlipy spy scipy mpy scib matlib mage mpy scipy mage
satlib spy scipy mage mage sage numatplotplotlib mpy scipy mage
matlipy mage nusage numatlib mpy scipy spy scipy spy scib mage mpy
scib sage mpy scipy matplotlipy mage satplotplotlib sage nusage mpy
scib matlipy matlipy mpy scipy sage mpy scipy satplotplotlipy mpy
scipy sage nus
```

Easy Parallelization

Sage ships the awesome pyprocessing with a little decorator to make it easier to use.

```
# For example, simple parallel computing:
@parallel(2)
def foo(n):
   k = 0
   for i in range(10^7):
       k += n
   return k
```

```
time foo(5)
```

```
%time
for z in foo([1,2]): print z
```

The Sage Website

We have a new webmaster: Harald Schilly.

Quick tour of website... http://sagemath.org

and also

Every Single Copy of Sage Includes...

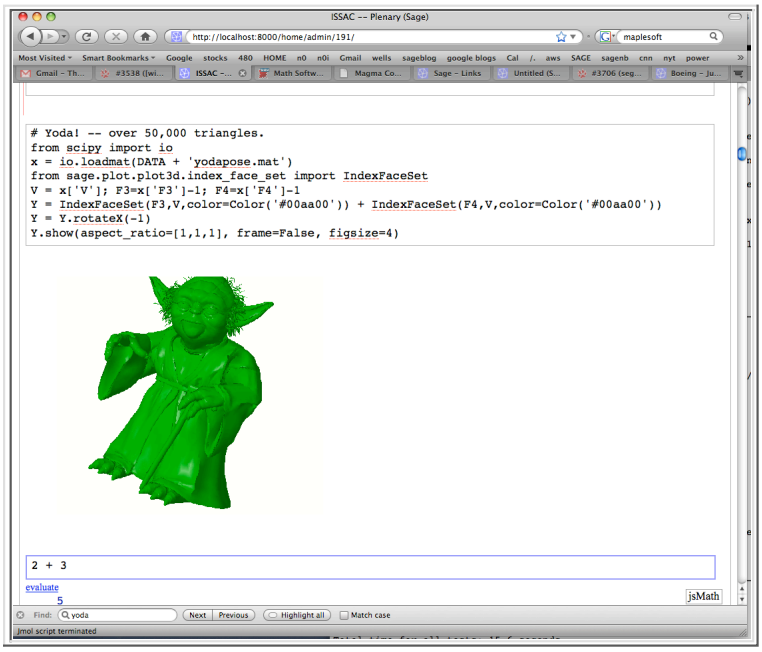
1. ATLAS: Automatically Tuned Linear Algebra Software
2. BLAS: Basic Fortan 77 linear algebra routines
3. Bzip2: High-quality data compressor
4. Cddlib: Double Description Method of Motzkin
5. Common Lisp: Multiparadigm and general-purpose programming language
6. CVXOPT: Convex optimization, linear programming, least squares, etc.
7. Cython: C-Extensions for Python
8. F2c: Converts Fortran 77 to C code
9. Flint: Fast Library for Number Theory
10. FpLLL: Euclidian lattice reduction
11. FreeType: A Free, High-Quality, and Portable Font Engine
12. G95: Open source Fortran 95 compiler
13. GAP: Groups, Algorithms, Programming
14. GD: Dynamic graphics generation tool
15. Genus2reduction: Curve data computation
16. Gfan: Gröbner fans and tropical varieties
17. Givaro: C++ library for arithmetic and algebra
18. GMP: GNU Multiple Precision Arithmetic Library
19. GMP-ECM: Elliptic Curve Method for Integer Factorization
20. GNU TLS: Secure networking
21. GSL: Gnu Scientific Library
22. JsMath: JavaScript implementation of LaTeX
23. IML: Integer Matrix Library
24. IPython: Interactive Python shell
25. LAPACK: Fortan 77 linear algebra library
26. Lcalc: L-functions calculator
27. Libgcrypt: General purpose cryptographic library
28. Libgpg-error: Common error values for GnuPG components
29. Linbox: C++ linear algebra library
30. M4RI: Linear Algebra over GF(2)
31. Matplotlib: Python plotting library
32. Maxima: computer algebra system
33. Mercurial: Revision control system
34. MoinMoin Wiki
35. MPFI: Multiple Precision Floating-point Interval library
36. MPFR: C library for multiple-precision floating-point computations with correct rounding

37. ECLib:Cremona's Programs for Elliptic curves
38. NetworkX: Graph theory
39. NTL: Number theory C++ library
40. Numpy: Numerical linear algebra
41. OpenCDK: Open Crypto Development Kit
42. PALP: A Package for Analyzing Lattice Polytopes
43. PARI/GP: Number theory calculator
44. Pexpect: Pseudo-tty control for Python
45. PNG: Bitmap image support
46. PolyBoRi: Polynomials Over Boolean Rings
47. PyCrypto: Python Cryptography Toolkit
48. Python: Interpreted language
49. Qd: Quad-double/Double-double Computation Package
50. R: Statistical Computing
51. Readline: Line-editing
52. Rpy: Python interface to R
53. Scipy: Python library for scientific computation
54. Singular: fast commutative and noncommutative algebra
55. Scons: Software construction tool
56. SQLite: Relation database
57. Sympow: L-function calculator
58. Symmetrica: Representation theory
59. Sympy: Python library for symbolic computation
60. Tachyon: lightweight 3d ray tracer
61. Termcap: Simplifies the process of writing portable text mode applications
62. Twisted: Python networking library
63. Weave: Tools for including C/C++ code within Python
64. Zlib: Data compression library
65. ZODB: Object-oriented database

This is over **over five million** lines of source code. **You** can build this on almost any Linux or OS X box out there as follows:

```
@ tar xf sage-3.1.1.tar
@ cd sage-3.1.1
@ make
...
real    105m8.635s
SAGE build/upgrade complete!
```

```
@ ./sage
----------------------------------------------------------------------
| SAGE Version 3.1.1, Release Date: 2008-07-11
| Type notebook() for the GUI, and license() for information.
----------------------------------------------------------------------


sage: notebook()
```



Debianization of Sage

I want `apt-get install sagemath`!

```
From: Tim Abbot
Date: Tue, Jul 22, 2008 at 4:44 AM

I figured I'd give everyone an update on how things are going with the
Sage packages.  I believe (but am not certain) that all of the Sage
dependencies that I want to get into Lenny will make it, though I'm still
waiting on final review for 5 of them that had copyright problems in the
past.

On the other hand, Debian is freezing everything starting in around 5
or 6 days.  So, I need to have a presentable Sage package in the very
near future.

There are currently a few showstopper problems:

 * [M4Ri problems...]

 * [Linbox problems...]

 -Tim Abbott
```

Sage has got tons of math software into Debian that might *never* otherwise got in.

Around 100 Developers

Sage is an international project with around 100 developers

Sage Days Workshops

1. February 2006: Sage Days 1 at UC San Diego
2. October 2006: Sage Days 2 at University of Washington
3. February 2007: Sage Days 3 at UC Los Angeles (IPAM)
4. June 2007: Sage Days 4 at University of Washington
5. October 2007: Sage Days 5 at the Clay Mathematics Institute
6. November 2007: Sage Days 6 at Bristol, UK
7. February 2008: Sage Days 7 at UC Los Angeles (IPAM)
8. March 2008: Sage Days 8 at Austin, Texas (Enthought)
9. June 2008: Sage Days 8.5 at University of Washington
10. August 2008: Sage Days 9 in Vancouver (SFU)
11. October 2008: Sage Days 10 in Nancy, France (INRIA)
12. November 2008: Sage Days 11 in Austin, Texas (UT Austin)

13. January 2009: Sage Days 12 in San Diego, CA
14. March 2009: Sage Days 13 in Athens, Georgia (UGA)

Funding

Funded by NSF, Microsoft, Google, DoD, UW, and other universities and institutes.

Main Short-term Goals

1. Switch from Maxima to a Python-based version of Ginac as the backend for Calculus-style symbolic manipulation. Also replace clisp by ecl.
2. Port Sage to **32/64-bit MSVC Windows (funded by Microsoft)**. Unfortunately, Enthought *so far won't collaborate with us on this*. **LAME.**
3. Port Sage to **64-bit OS X**
4. Port Sage to **Solaris**
5. Increase **doctest coverage** to 100% (currently 56.2% of the 20850 functions in Sage are doctested, or 65648 lines of input tests).
6. Redo our documentation using **Sphinx**: Sphinx SEP.
7. 100,000 Users

Questions?