What does Sage do?

http://www.sagemath.org

Craig Citro and William Stein
UCLA Math / UW Math

January 21, 2009

Outline

1. What does Sage do?

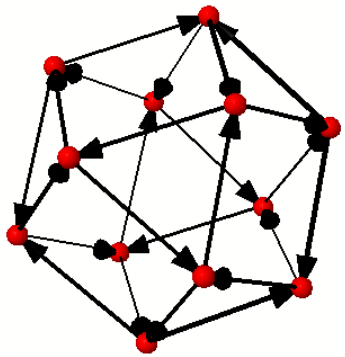
2. What Sage Does: The Details
   - Some of Sage's mathematical strengths
   - Cython
   - Parallel Computing in Sage
   - Interact
   - The Notebook
   - Open Source in Sage
   - Sage and Correctness Testing

Sage solves interesting mathematical problems

```
sage: factor(2009)
7^2 * 41

sage: time n = number_of_partitions(10^8)
CPU times: user 2.39 s, sys: 0.00 s, total: 2.39 s
sage: len(n.digits())
11132


sage: integrate(sin(x)*exp(x),x)
e^x*(sin(x) - cos(x))/2


sage: time w = pi.str(10^6)
CPU times: user 1.22 s, sys: 0.00 s, total: 1.22 s
```

Sage solves interesting mathematical problems **quickly**

```
sage: G = AlternatingGroup(4); H = G.cayley_graph(); H.show3d()
```

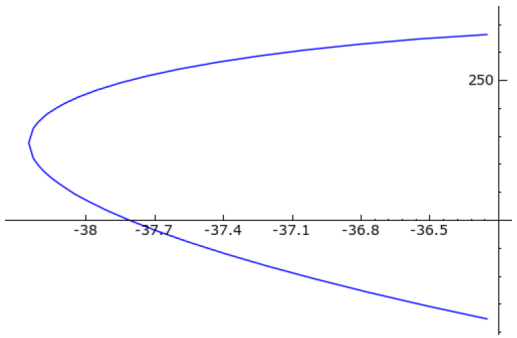


```
sage: A = H.automorphism_group(); A
Permutation Group with gens [(1,2,12)(3,5,4)(6,7,8)(9,11,10),
(1,4)(2,6)(3,8)(5,9)(7,10)]
sage: A.order()
24
sage: for n in [3..7]:
          An = AlternatingGroup(n); G = An.cayley_graph()
          print n, An.order(), G.automorphism_group().order()
3 3 3
4 12 24                  # <--- theorem or bug?
5 60 60
6 360 360
7 2520 2520
```

Sage solves interesting mathematical problems **quickly**

```
sage: E = EllipticCurve([12,3,4,5,6]); show(E)
```

$$y^2 + 12xy + 4y = x^3 + 3x^2 + 5x + 6$$




```
sage: factor(E.conductor())
2^4 * 5 * 11 * 13 * 277

sage: time E.rank()
2
CPU time: 0.02 s,  Wall time: 0.49 s

sage: time E.gens()
[(-6 : 66 : 1), (-2 : 20 : 1)]
CPU time: 0.00 s,  Wall time: 0.64 s
```

Sage makes mathematical experimentation easy ...

(2.3) **Conjecture.** *If $E(K)$ has rank 1, then the integer $c \cdot m \cdot u_K \cdot |Ш_K|^{\frac{1}{2}}$ is divisible by t.*

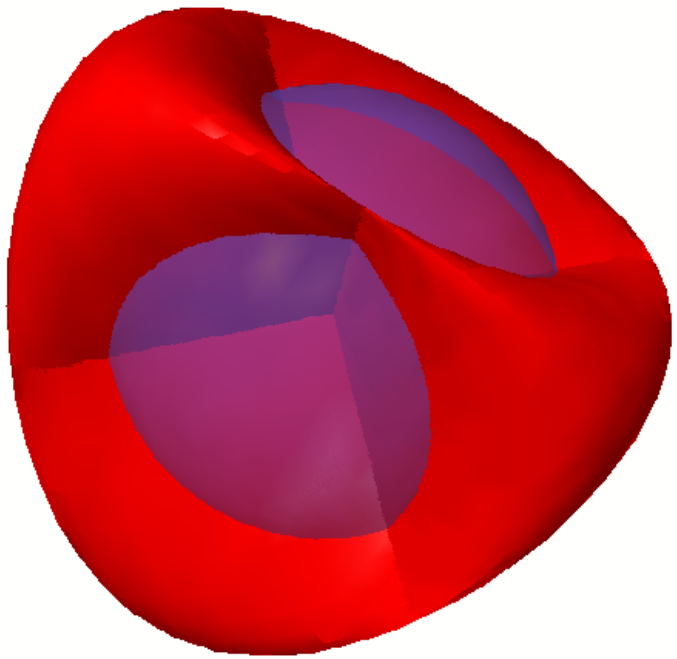
```
def data(E, D):
    F = E.quadratic_twist(D).minimal_model()
    rE = E.rank(); rF = F.rank()
    T = E.torsion_order()*F.torsion_order()
    cqprod = E.tamagawa_product()
    L_E = E.lseries().dokchitser().derivative(1, rE)
    L_F = F.lseries().dokchitser().derivative(1, rF)
    om2 = 2*abs(E.period_lattice().basis_matrix().det())
    sha = round( abs((L_E*L_F / (factorial(rE) * factorial(rF)) )
          * T^2 * sqrt(abs(D),prec=53) /
         (om2 * E.regulator() * F.regulator() * cqprod^2)) )
    sha = odd_part(sha)
    return odd_part(T), odd_part(cqprod), sqrt(sha)

sage: E = EllipticCurve('1862a1'); E.rank()
2
sage: for D in E.heegner_discriminants_list(10):
          print D, data(E,D)
-31 (3, 3, 1)
...
```

. . . using modern tools

```
sage: N = 2^217-1; M = 2^218-1; ls = [N,M]
sage: time factor(N)
127 * 5209 * 62497 * 2147483647 * 6268703933840364033151 * ...
CPU time: 1.00 s,  Wall time: 1.17 s

sage: time [ factor(x) for x in ls ]
[127 * 5209 * 62497 * ..., 3 * 104124649 * 745988807 * ...]
CPU time: 2.23 s,  Wall time: 2.44 s

sage: @parallel(2)
...     def f_para(n):
...         return factor(n)
sage: time v = list(f_para(ls))
CPU time: 0.03 s,  Wall time: 1.50 s
```

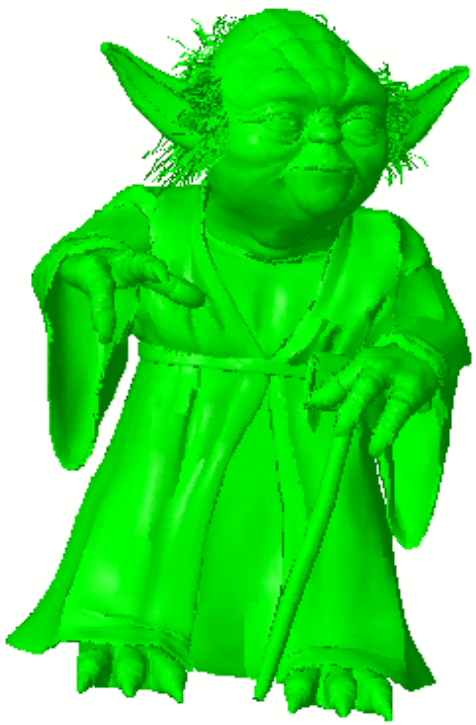
Sage draws pretty pictures ...

```
#  Steiner surface/Roman's surface with embedded sphere
sage: u, v = var('u,v')
sage: fx = sin(2*u) * cos(v) * cos(v); fy = sin(u) * sin(2 * v)
sage: fz = cos(u) * sin(2*v)
sage: (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"))
```



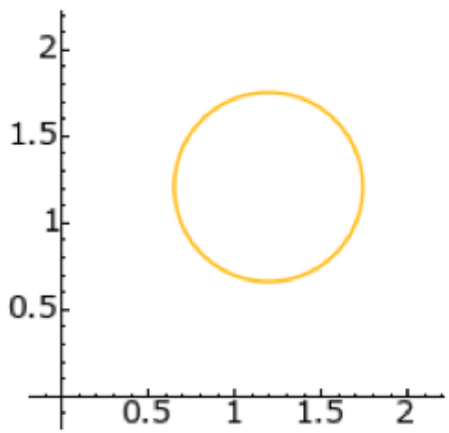
. . . and makes them move!

```
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)
```



Even by themselves!

```
sage: a = animate([circle((i,i), 1-1/(i+1), hue=i/10)
                   for i in srange(0,2,0.2)],
                  xmin=0,ymin=0,xmax=2,ymax=2,figsize=[2,2])

sage: show(a)
```



Sage provides interfaces to other mathematical software

```
sage: gp(2) + singular(5)
7

sage: var('x')
sage: f = sin(x^2) + pi/2
sage: show(f.integrate())
```

$$\frac{\sqrt{\pi}\left(\left(\sqrt{2}i+\sqrt{2}\right)\mathrm{erf}\left(\frac{\left(\sqrt{2}i+\sqrt{2}\right)x}{2}\right)+\left(\sqrt{2}i-\sqrt{2}\right)\mathrm{erf}\left(\frac{\left(\sqrt{2}i-\sqrt{2}\right)x}{2}\right)\right)}{8}+\frac{\pi x}{2}$$

```
sage: g = mathematica(f); g
Pi/2 + Sin[x^2]
sage: g.Integrate(x)
(Pi*x)/2 + Sqrt[Pi/2]*FresnelS[Sqrt[2/Pi]*x]
```

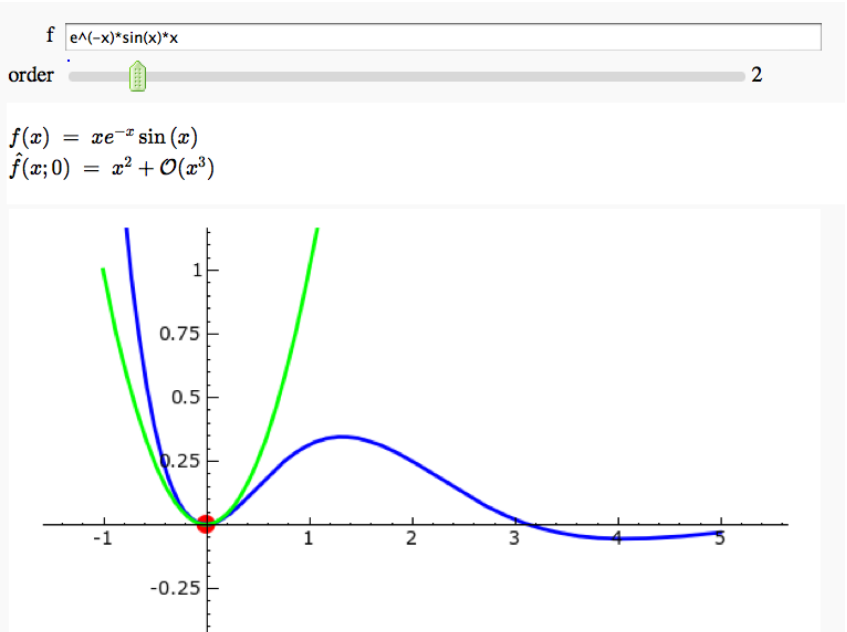
$$\frac{\pi x}{2}+\sqrt{\frac{\pi}{2}}S\left(\sqrt{\frac{2}{\pi}}x\right)$$

Sage provides includes lots of mathematical software

```
sage: r_console()

R version 2.6.1 (2007-11-26)
Copyright (C) 2007 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

...

> 3
[1] 3
> x <- c(1,2,3,4)
> x
[1] 1 2 3 4
> 1/x
[1] 1.0000000 0.5000000 0.3333333 0.2500000
> q()

sage:
```

Sage makes visualizing mathematics easy

```
var('x')
x0 = 0
@interact
def _(f = input_box(sin(x)*e^(-x)), order = (1..12)):
    p = plot(f, -1, 5, thickness = 2)
    dot = point((x0,f(x0)), pointsize = 80, rgbcolor = (1,0,0))
    ft = f.taylor(x, x0, order)
    pt = plot(ft, -1, 5, color = 'green', thickness = 2)
    html('$f(x)\;=\;%s$'%latex(f))
    html('$\hat{f}(x;%s)\;=\;%s+\mathcal{O}(x^{%s})$'%(
          x0, latex(ft), order+1))
    show(dot + p + pt, ymin = -.5, ymax = 1)
```



Curious? Easy access to source code. Change anything.

```
sage: n = 5.1
sage: n.floor??
    def floor(self):
        """
        Returns the floor of this number

        EXAMPLES:
            sage: R = RealField()
            sage: (2.99).floor()
            2
            sage: (2.00).floor()
            2
            sage: floor(RR(-5/2))
            -3
            sage: floor(RR(+infinity))
            Traceback (most recent call last):
            ...
            ValueError: Calling floor() on infinity or NaN
        """
        cdef RealNumber x
        if not mpfr_number_p(self.value):
            raise ValueError, 'Calling floor() on infinity or NaN'
        x = self._new()
        mpfr_floor(x.value, self.value)
        return x.integer_part()
```

Very Curious? Look at MPFR's source...

Look at `spkg/standard/mpfr-2.3.2/src/rint.c`

```
mpfr_floor (mpfr_ptr r, mpfr_srcptr u)
{
  return mpfr_rint(r, u, GMP_RNDD);
}
...
mpfr_rint (mpfr_ptr r, mpfr_srcptr u, mpfr_rnd_t rnd_mode)
{
... about 270 lines nested 8 levels deep ...
          /* More limbs in the integer part of u than in r.
             Just round u with the precision of r. */
          MPFR_ASSERTD (rp != up && un > rn);
          MPN_COPY (rp, up + (un - rn), rn); /* r != u */
          if (rnd_away < 0)
            {
              /* This is a rounding to nearest mode (GMP_RNDN or G
                 Decide the rounding direction here. */
              if (rnd_mode == GMP_RNDN &&
                  (rp[0] & (MPFR_LIMB_ONE << sh)) == 0)
                { /* halfway cases rounded towards zero */
                  mp_limb_t a, b;
                  /* a: rounding bit and some of the following bit
                  /* b: boundary for a (weight of the rounding bit
...
      if (rnd_away && mpn_add_1(rp, rp, rn, MPFR_LIMB_ONE << sh))
```

Very Very Curious? Look at MPIR's source...

Look at spkg/standard/gmp-mpir-svn1555.p0/src/mpn/x86_64/add_n.as

```
;  Copyright 2004 Free Software Foundation, Inc.
;  Copyright 2008 William Hart
;  This file is part of the MPIR Library.
...
; INPUT PARAMETERS
; rp     rdi
; up     rsi
; vp     rdx
; n      rcx

%include '../yasm_mac.inc'

    BITS    64

GLOBAL_FUNC mpn_add_n
        lea     rsi, [rsi+rcx*8]
        lea     rdi, [rdi+rcx*8]
        lea     rdx, [rdx+rcx*8]
        neg     rcx
        xor     eax, eax                        ; clear cy

        align 4                                 ; minimal alignmen
...
```

Outline

1 What does Sage do?

2 What Sage Does: The Details
- Some of Sage's mathematical strengths
- Cython
- Parallel Computing in Sage
- Interact
- The Notebook
- Open Source in Sage
- Sage and Correctness Testing

What does Sage do?

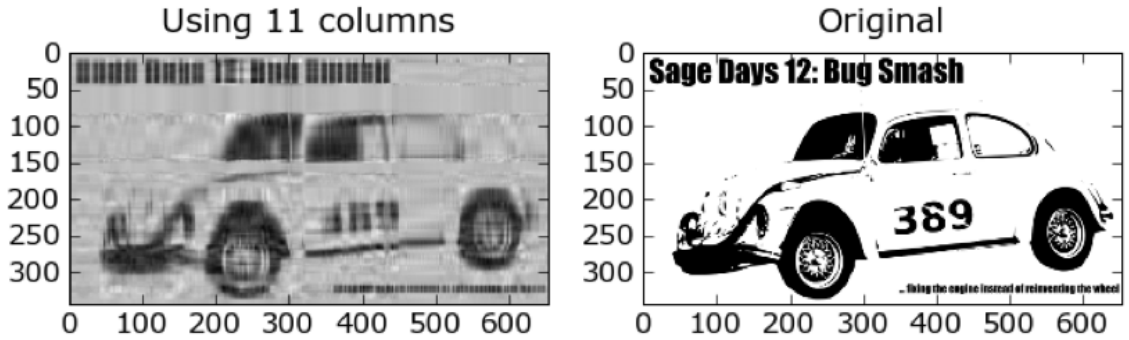
Sage:

- solves interesting problems quickly.
- makes experimentation easy.
- interfaces with everything under the sun.
- makes creating and interacting with graphics simple.
- makes sharing resources and collaborating natural.
- shows you every line of source code.
- local or on the Web – no difference.
- is **completely and totally FREE.**

Some of Sage's strengths

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

Numerical computation using scipy/numpy/matplotlib (Matlab replacement)




```
import pylab
A = pylab.imread(DATA + 'bug.png')
Agray = pylab.mean(A, 2)
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('x.png')

@interact
def energy_in_svd(i=slider(0,50,1)):
    svd_image(Agray,i)
```

Determinants of Integer Matrices

```
sage: a = random_matrix(ZZ, 200, x=-2^128,y=2^128)
sage: time d = a.det()
CPU time: 4.25 s,  Wall time: 4.49 s
```

Table 9.2: Time in seconds to compute the **determinant** of a random $n \times n$ matrix whose entries are uniformly distributed in the interval $[-2^b, 2^b]$, where b =bits.

	n	8 bits	32 bits	128 bits	256 bits	512 bits
Sage	50	0.0	0.1	0.3	0.8	3.7
	250	1.3	2.0	9.1	31.5	138.2
	500	7.2	12.6	54.9	190.2	646.6
	1000	55.6	105.0	397.3	1057.4	3435.1
	1500	230.6	407.0	1242.7	3255.8	8133.3
	2000	544.1	997.1	2828.0	6138.2	15533.5
	3000	1991.8	3132.5	7473.8	14385.3	39834.7
Magma	50	0.1	0.1	0.3	0.6	1.8
	250	0.4	12.8	62.9	192.3	659.2
	500	3.8	108.5	455.7	1175.5	4362.9
	1000	40.1	677.7	3871.2	9406.8	25430.3
	1500	122.4	3085.8	12327.8	28987.9	78741.3
	2000	219.7	6097.1	26438.8	63898.0	185228.1
	3000	1175.2	17868.7	82417.0	207316.0	
NTL	50	0.0	0.0	0.1	0.1	0.3
	250	2.2	9.5	40.3	43.0	93.2
	500	46.6	119.3	359.3	1104.5	2100.8
	1000	659.8	1943.2	7158.3	14538.8	28711.5
GAP	50	0.1	0.5	2.9	5.5	20.5
	250	107.7	548.4	4533.3		
Pari	50	0.0	0.1	0.4	1.0	3.2
	250	667.6	1288.3	3856.3		

Sage – What's inside?

Sage comes standard with over 70 packages, including:

Arithmetic	**GMP, MPFR, Givaro, MPFI**
Commutative Algebra	**PolyBoRi, SINGULAR** (libSINGULAR)
Linear Algebra	**LinBox, M4RI, IML, fpLLL**
Cryptosystems	**GnuTLS, PyCrypto**
Integer Factorization	**FlintQS, ECM**
Group Theory	**GAP**
Combinatorics	**Symmetrica, sage-combinat**
Graph Theory	**NetworkX**
Number Theory	**PARI, NTL, Flint, mwrank, eclib**
Numerical Computation	**GSL, Numpy, Scipy, ATLAS**
Calculus, Symbolic Comp.	**Maxima, Sympy, Pynac**
Statistics	**R**, Scipy.stats
User Interface	Sage Notebook, **jsmath, Moin wiki, IPython**
Graphics	**Matplotlib, Tachyon, libgd, JMol**
Networking	**Twisted**
Databases	**ZODB, SQLite, SQLAlchemy**, Python pickle
Programming Language	**Python, Cython** (compiled)

Python

Sage is written in Python, a widely used and well regarded programming language, and also uses Python as the user language. So unlike every other piece of mathematical software in the world, Sage uses a mainstream programming language as the user language, instead of creating our own new language.

Python

The choice of Python has a huge number of positive impacts, including:

- **lots of well-written introductions and reference material** for Python already out there, for programmers of every level of experience
- **an extremely fast development cycle** – companies such as Google and LucasFilm use Python for rapid prototyping and development
- **an insane number of standard and third-party libraries** available, for everything from web programming to image compression to scientific computing
- we don't have to be in the business of being language designers!

However, there is one disadvantage: since it's such a dynamic programming language, Python programs are often slower than their direct equivalents in C. The answer: Cython.

Cython

Cython (`http://www.cython.org`) lets you:

- declare attributes for your classes with C datatypes
- declare methods to take and return C datatypes
- interface with your existing C/C++ libraries

Cython

```
sage: def mysum(N):
...       s = int(0)
...       for k in range(1,N):
...           s += k
...       return s
sage: time mysum(10^6)
499999500000L
CPU time: 0.18 s,  Wall time: 0.26 s

sage: %cython
sage: def mysum_cython(N):
...       cdef int k
...       cdef long long s = 0
...       for k in range(N):
...           s += k
...       return s
sage: time mysum_cython(10^6)
499999500000L
CPU time: 0.00 s,  Wall time: 0.00 s
```

Cython

```
sage: timeit('mysum(10^6)')
5 loops, best of 3: 254 ms per loop

sage: timeit('mysum_cython(10^6)')
625 loops, best of 3: 1.24 ms per loop

sage: 254/1.24
204.838709677419

sage: mysum_cython(10^10)
Traceback (click to the left for traceback)
...
OverflowError: long int too large to convert to int
```

Cython

No one wants to declare types for all of their objects, and manually allocate and deallocate our C objects – this is one of the reasons we aren't using C in the first place!

We don't have to. The Cython development model:

- Write code in Python.
- Get it working **correctly**.
- Profile the code.
- Move the inner loops to Cython.

Cython: It Works

Jason Grout:

> I spent two or three days working on this. Here is the end result: 0.24
> seconds compared to 150 seconds. Such is the power of Cython :). That's
> a speedup of a factor of 150.64/0.24=627!

This particular function, because it is so fast now, has become a regular tool in our research and has led to discovering at least one counter-example to a conjecture that was open for several months.

Cython

Not that you needed any more reasons, but here are a few more amazing things that Cython has to offer:

- Built-in profiling/annotation tools for performance analysis
- Automatic conversion between most Python and C types (whenever it would make sense)
- Cython can also be used to interface with C++ libraries (only a small amount of black magic needed!)

Parallel and Distributed Computing

By now, everyone's heard that we need to move to multicore and distributed computing to be taking advantage of tomorrow's finest hardware. Sage has two primary ways of doing this:

- Python is an excellent scripting language, which means it's fairly trivial to write Python scripts to queue up jobs on your local supercomputing cluster. Unlike other math software, Sage has no site licenses – which makes this completely straightforward. We've done this on the supercomputers at UT Austin, with hundreds of Sage processes running concurrently for a week or more at a time.
- The `@parallel` decorator, built using the `pyprocessing` Python extension. This gives a quick and easy way to take advantage of multiple cores. The `@parallel` decorator is simple to use and quite robust – there's nothing to learn, it **just works**.

. . . using modern tools

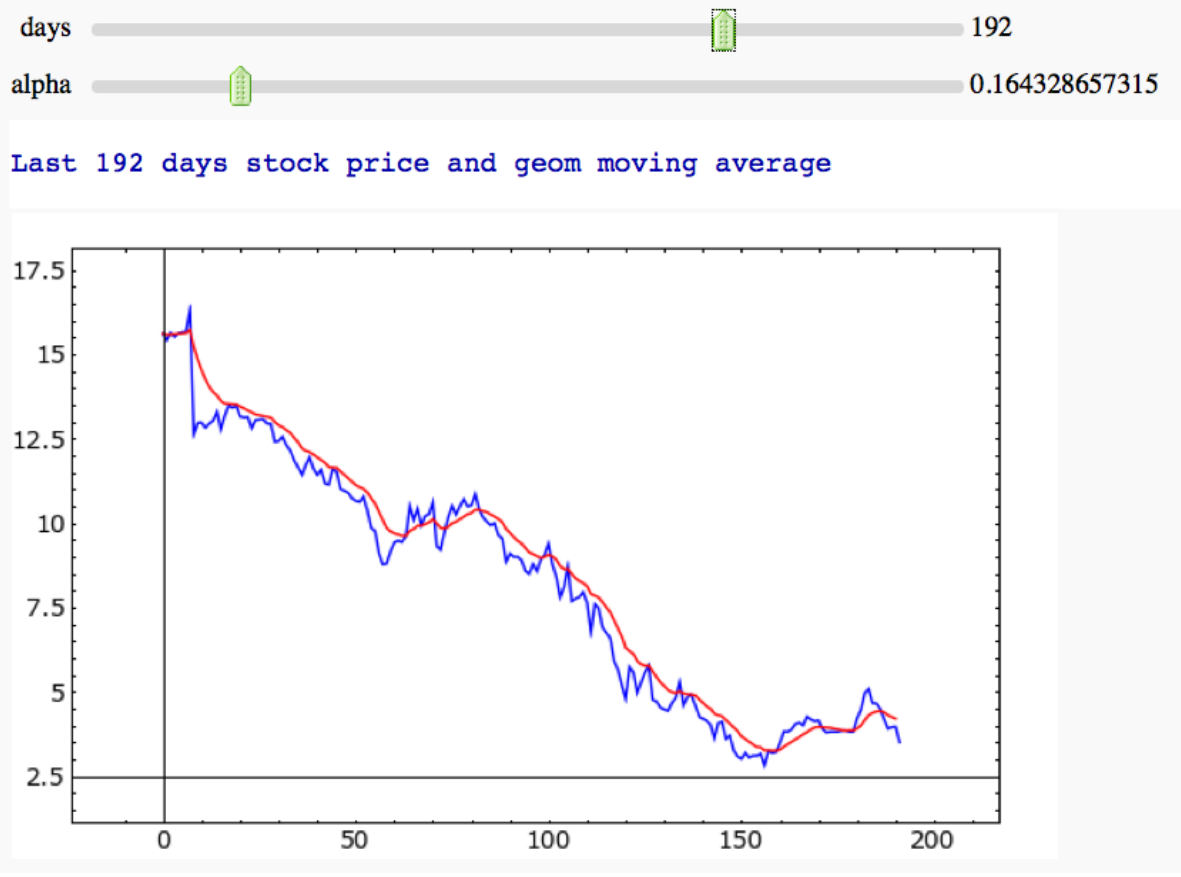
```
sage: ls = [2^n-1 for n in [190..210]]
sage: time v = [ factor(x) for x in ls ]
[127 * 5209 * 62497 * ..., 3 * 104124649 * 745988807 * ...]
Time: CPU 9.32 s, Wall: 9.32 s

sage: @parallel(6)
sage: def f_para(n):
....:     return factor(n)
sage: time v = list(f_para(ls))
Time: CPU 0.04 s, Wall: 2.95 s   # 2.95s = longest factorization
```

Interact

- Interact is a great example of a useful tool: it turns one moderately annoying task (running the same series of commands on varying inputs) into a ridiculously easy task. Therein lies its utility.
- Interact allows you to visually explore a data set, and get data as you explore – regardless of whether you want textual or visual information (or both!).

Interact at work

```
S = finance.Stock('java').close()
@interact
def _(days=(80,(20..len(S))), alpha=(0.2,(0,1))):
    print "%s days of Sun stock & geometric moving average"%days
    Z = S[-days:]
    G = Z.exponential_moving_average(alpha)[1:]
    (Z.plot() + G.plot(rgbcolor='red')).show(frame=True)
```



Inheritance hierarchy of 1

```
def class_hierarchy(cls, v):
    v.append(str(cls))
    for supercls in cls.__bases__:
        class_hierarchy(supercls, v)
@interact
def _(object=1):
    print '<html><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'>
```

Sage on the Web

Sage has the command line interface you'd expect, but Sage also comes with a web interface that works with your favorite browser.

Guido van Rossum, creator of Python

In recent times, I have not had either the need or the desire to do any GUI development at all. Probably not at all in the last three years. I've done plenty of web development. In general there is a lot of focus change where people don't develop as many desktop apps anymore. ... There is less and less need for GUI apps.

On the face of it, the Notebook is a smooth and clean way to build a graphical interface on several platforms, without having to devote a huge amount of our resources on designing and maintaining a UI ourselves. However, the Notebook gives you so much more ...

The Sage Notebook

Here are some of the many uses of the Sage Notebook:

- By starting a Notebook server on your work machine, you can now use any computer with a web browser (read: any computer made after 1994) to connect to it and make use of that hardware.
- The Notebook makes an excellent tool for teaching.
- Sage worksheets are excellent for giving presentations.
- By sharing your worksheets with other users on a common Notebook server, you can easily collaborate with anyone who has access to that machine.

Inspecting Every Detail

We've taken several important lessons from the world of mathematics and applied them to Sage.

- As you've seen, you can inspect every line of code used in a computation in Sage, just as you can look at every line in a proof of a theorem, and a proof of every theorem it uses, etc.
- Just as with mathematical papers, every line of code that makes it into Sage nowadays goes through a formal refereeing process.

But do **you** trust it?

Before every single release, we:

- build Sage on dozens of different combinations of CPU and operating system (the "build farm"),
- run the doctest suite on **every one of these machines**, currently around 77000 doctests (not counting the testing we do on the various components of Sage),
- ask volunteers on `sage-devel` to do the same, and
- report all issues, and wait to release until these are fixed.

We also keep track of all known bugs on our bug tracker, `http://trac.sagemath.org`.

Unit Tests and You

So it's really reassuring to find out that we take so much care to maintain the correctness of the Sage codebase, and that we regularly test before releasing. It makes trusting Sage much easier. However . . . it should also make you want to contribute your code to Sage right now.

Why should you want to drop what you're doing and contribute code to Sage? Because it means that **we** will do the job of testing and maintaining your code for the rest of the life of the Sage project. If you can get your code cleaned up and in shape enough to be part of the Sage codebase, it becomes part of the Sage doctest suite. This is particularly nice for examples in books or papers.

Sage Design Principles

There are a number of important principles that guide Sage development. Here are a few of the key ones:

- **Don't reinvent the wheel. Build the car instead.** This means that as much as possible, Sage uses existing open source libraries and packages instead of spending time repeating existing efforts.
- **If it isn't tested, it's broken.** We want Sage to be powerful and robust system that people can use and trust for their research. Having an extensive test suite is absolutely vital for this purpose.
- **Talk is cheap. Show me the code.**
- **Development should be a public process.**

Any questions?

Thanks for listening!