math 480 - 20050521 - symbolics

Math 480: 2005-05-21

Symbolic Calculus in Sage

In Sage, **Symbolic Calculus** is mainly about solving the sorts of problems that come up in typical calculus courses, including

1. Symbolic differentiation
2. Symbolic integration
3. Computing limits
4. Computing taylor series
5. Solving equations
6. Numerical approximation to integrals

You can do all this in Sage right now. **Quick tour of the relevant section of the reference manual...**

The main software systems out there for symbolic calculus are:

1. Mathematica (commercial)
2. Maple (commercial)
3. MuPAD (commercial)
4. Maxima (free)

There's also Axiom and Xcas, but neither have both the number of users and maturity of the above systems yet. Note that Magma has essentially no symbolic calculus capabilities. Matlab has the symbolic toolbox which is really Maple embeded into Matlab.

History: Macsyma

The first major open source program for symbolic calculus was Macsyma, which is now dead. It was started in July, 1968 by Carl Engelman, William A. Martin and Joel Moses. Macsyma was written entirely in **LISP**.

> Macsyma was developed at M.I.T. from 1968 to 1982 with funds from the U.S. Defense Advanced Research Projects Agency [DARPA] and some from the Department of Energy [DOE]. The government lost interest in Macsyma around 1977 when numerical analysts persuaded the government than numerical libraries on supercomputers were better suited to performing the engineering computations needed by the defense establishment.
>
> In the late 1970s the professor in charge of Macsyma at M.I.T. wanted to lead a company to commercialized Macsyma while he remained a professor at M.I.T. M.I.T. has never permitted this sort of arrange- ment, and would not approve the plan.

... [Macsyma was then crushed by Maple and Mathematica.] ..."

See this post for more.

History: Maxima

A descendant of Macsyma called Maxima is now alive and well. I think Maxima is easily the most popular open source symbolic calculus system.

"Maxima is a descendant of Macsyma, the legendary computer algebra system developed in the late 1960s at the Massachusetts Institute of Technology. It is the only system based on that effort still publicly available and with an active user community, thanks to its open source nature. Macsyma was revolutionary in its day, and many later systems, such as Maple and Mathematica, were inspired by it.

The Maxima branch of Macsyma was maintained by William Schelter from 1982 until he passed away in 2001. In 1998 he obtained permission to release the source code under the GNU General Public License (GPL). It was his efforts and skill which have made the survival of Maxima possible, and we are very grateful to him for volunteering his time and expert knowledge to keep the original DOE Macsyma code alive and well. Since his passing a group of users and developers has formed to bring Maxima to a wider audience.

We are constantly updating Maxima, to fix bugs and improve the code and the documentation. We welcome suggestions and contributions from the community of Maxima users. Most discussion is conducted on the Maxima mailing list."
http://maxima.sourceforge.net/

What Sage Does and How: A Quick Tour

There are three options for symbolic calculus in Sage:

1. Maxima -- Sage includes Maxima, so anything you can do in Maxima you can do in Sage
2. Sympy -- a standalone pure Python symbolic calculus module included in Sage
3. Sage's native calculus module -- builds on Maxima

Maxima in Sage

```
%maxima

integrate(sin(x)*cos(x) + log(3*x),x)
```

```
    (3*x*log(3*x)-3*x)/3-cos(x)^2/2
```

```
%maxima

diff((3*x*log(3*x)-3*x)/3-cos(x)^2/2, x)
```

```
    log(3*x)+cos(x)*sin(x)
```

```
timeit("maxima.eval('diff((3*x*log(3*x)-3*x)/3-cos(x)^2/2, x)')")
```

```
    125 loops, best of 3: 6.65 ms per loop
```

```
maxima.eval('integrate(sin(x)*cos(x) + log(3*x),x)')
```

```
    '(3*x*log(3*x)-3*x)/3-cos(x)^2/2'
```

```
timeit("maxima.eval('integrate(sin(x)*cos(x) + log(3*x),x)')")
```

```
    125 loops, best of 3: 5.37 ms per loop
```

```
maxima.eval('f : sin(x)*cos(x) + log(3*x); g :
(3*x*log(3*x)-3*x)/3-cos(x)^2/2;')
```

```
    'log(3*x)+cos(x)*sin(x)<sage-display>(%o2192)(3*x*log(3*x)-3*x)/3\
    -cos(x)^2/2'
```

```
timeit('maxima.eval("expand(f*g)")')
```

```
    125 loops, best of 3: 7.4 ms per loop
```

```
maxima.eval('limit((3*x*log(3*x)-3*x)/3-cos(x)^2/2, x, 1)')
```

```
    Traceback (click to the left for traceback)
    ...
    Is  x+1  positive or negative?
```

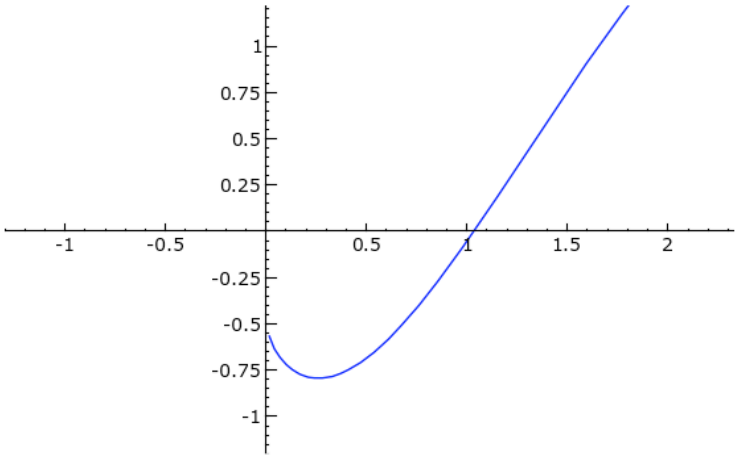
```
maxima.eval('limit((3*x*log(3*x)-3*x)/3-cos(x)^2/2, x, 1, plus)')
```

```
    '(2*log(3)-cos(1)^2-2)/2'
```

```
maxima.eval('limit((3*x*log(3*x)-3*x)/3-cos(x)^2/2, x, 1, minus)')
```

```
    Traceback (click to the left for traceback)
    ...
    Is  x+1  positive or negative?
```

```
var('x')
plot((3*x*log(3*x)-3*x)/3-cos(x)^2/2, 0,2)
```




```
maxima.eval('a : 5')
maxima.eval('a^2')
```

```
    '25'
```

```
maxima.eval('sum (1/3^n, n, 1, inf), simpsum');
```

```
    '1/2'
```

```
maxima.eval('sum (1/3^n + 1/n^4, n, 1, inf), simpsum');
```

```
    '%pi^4/90+1/2'
```

Sympy in Sage

Sympy "is a Python library for symbolic mathematics. It aims to become a full-featured computer algebra system (CAS) while keeping the code as simple as possible in order to be comprehensible and easily extensible. SymPy is written entirely in Python and does not require any external libraries."

Part of the point of sympy is that if you like using Python quite a lot for symbolic computation, but are in a situation where installing Sage is prohibitive, you can certainly install sympy no problem.

Sympy is very new code all in Python, where as much of Sage's calculus is built on the very old Maxima which is in lisp. It has been observed often by Ondrej Certik, the project director of Sympy, that Sage/Maxima has way fewer bugs in symbolic calculus than Sympy.

```
import sympy
```

```
x = sympy.Symbol('x')
```

```
f = sympy.sin(x)*sympy.cos(x) + sympy.log(3*x)
```

```
type(f)
```
```
    <class 'sympy.core.add.Add'>
```
```
f
```
```
    cos(x)*sin(x) + log(3*x)
```
```
f.integral(x)    # bug report filed...
```
```
    Traceback (click to the left for traceback)
    ...
    NameError: global name 'Equality' is not defined
```
```
f.integral()
```
```
    Integral(cos(x)*sin(x) + log(3*x), x)
```
```
timeit('f.integral(x)')
```
```
    Traceback (click to the left for traceback)
    ...
    NameError: global name 'Equality' is not defined
```
```
g = (3*x*sympy.log(3*x)-3*x)/3-sympy.cos(x)^2/2
```

```
type(g)
```
```
    <class 'sympy.core.add.Add'>
```
```
g.diff()
```
```
    -x - 1/2*cos(x)**2 + x*log(3*x)
```
```
timeit('g.diff()')
```

```
    625 loops, best of 3: 3.4 Âµs per loop
```

```
g.limit(x,1)
```

```
    (-1) - 1/2*cos(1)**2 + log(3)
```

```
float(g.limit(x,1))
```

```
    -0.047351002195103999
```

```
h = f*g; h
```

```
    (cos(x)*sin(x) + log(3*x))*(-x - 1/2*cos(x)**2 + x*log(3*x))
```

```
type(h)
```

```
    <class 'sympy.core.mul.Mul'>
```

```
h.expand()
```

```
    x*log(3*x)**2 - x*log(3*x) - 1/2*cos(x)**2*log(3*x) -
    1/2*cos(x)**3*sin(x) - x*cos(x)*sin(x) + x*cos(x)*log(3*x)*sin(x)
```

```
timeit('(f*g).expand()')
```

```
    625 loops, best of 3: 520 Âµs per loop
```

```

```

```

```

Sage's Native Calculus

```
x = var('x')
```

```
f = sin(x)*cos(x) + log(3*x); show(f)
```

$$\log\left(3x\right) + \cos\left(x\right)\sin\left(x\right)$$

```
integrate(f,x)
```

```
    (3*x*log(3*x) - 3*x)/3 - cos(x)^2/2
```

```
timeit('integrate(sin(x)*cos(x) + log(3*x),x)')
```

```
    5 loops, best of 3: 42 ms per loop
```

```
g = (3*x*log(3*x) - 3*x)/3 - cos(x)^2/2; show(g)
```

$$\frac{3x\log\left(3x\right) - 3x}{3} - \frac{\cos\left(x\right)^{2}}{2}$$

```
g.diff()
```

```
    log(3*x) + cos(x)*sin(x)
```

```
timeit('g.diff()')
```

```
    25 loops, best of 3: 31.2 ms per loop
```

```
h = f*g; show(h)
```

$$(\log(3x) + \cos(x)\sin(x))\left(\frac{3x\log(3x) - 3x}{3} - \frac{\cos(x)^2}{2}\right)$$

```
show(h.expand())
```

$$x(\log(3x))^2 + x\cos(x)\sin(x)\log(3x) - \frac{\cos(x)^2\log(3x)}{2} - x\log(3x) - \frac{\cos(x)^3\sin(x)}{2} - \ldots$$

```
timeit('(f*g).expand()')
```

```
    5 loops, best of 3: 54.1 ms per loop
```

```
# Notice that this takes 100 times longer than sympy!
54.1/.520
```

```
    104.038461538462
```

```

```

```

```

Sage: Symbolic Calculus's Future

Problems

As can be seen above, Sage's symbolic calculus though extremely full featured is, by being based on Maxima, way too slow for many important operations. Also, Symbolic calculus is presently **not** the core of Sage, partly because people in both number theory and scientific tend to think they don't need it so much (they're perhaps wrong).

Another major problem with the current version of calculus is that it's difficult to fix under the hood, and deep bugs are impossibly hard to deal with, unless you're a lisp programmer that understand's maxima nearly undocumented code well.

```
integrate(sin(x)*cos(10*x)*log(x))
```

```
    Traceback (click to the left for traceback)
    ...
    Too many contexts.
```

```
# NOW the MAXIMA backend is completely BROKEN!
integrate(x)
```

```
    Traceback (click to the left for traceback)
    ...
    Too many contexts.
```

```
%html
```

```
See <a
href="http://trac.sagemath.org/sage_trac/ticket/3013">http://trac.sager
```

See http://trac.sagemath.org/sage_trac/ticket/3013.

The Future

1. Funded by Google, Gary Furnish is completely rewriting Sage's symbolic calculus from scratch to be faster, powerful, and robust.
2. This *can* take a while, since it will work very much like the current symbolic calculus, which is already just fine for 90% of people/uses.
3. Gary has made substantial progress.
4. We are not completely building this new symbolic calculus code on top of Sympy because:
    1. Sympy is almost completely pure python, hence too slow in the long run.
    2. Sympy is too focused on 1-d calculus.
    3. Sympy doesn't "understand" Sage's much more mathematical framework (elements, rings, categories, coercions, etc.)
    4. In the long run, probably Sympy is to Sage's Calculus as Numpy is to Sage matrices.