SymPy – A Pure Python Symbolic Manipulation Package

SymPy Development Team, presenting Ondřej Čertík

University of Nevada, Reno

January 7, 2009

SymPy

- A Python library for symbolic mathematics
- http://code.google.com/p/sympy/

```
>>> from sympy import Symbol, limit, sin, oo
>>> x=Symbol("x")
>>> limit(sin(x)/x, x, 0)
1
>>> integrate(x+sinh(x), x)
>>> (1/2)*x**2 + cosh(x)
```

SymPy

What SymPy can do

- basics (expansion, complex numbers, differentiation, taylor (laurent) series, substitution, arbitrary precision integers, rationals and floats, pattern matching)
- noncommutative symbols
- limits and some integrals
- polynomials (division, gcd, square free decomposition, groebner bases, factorization)
- symbolic matrices (determinants, LU decomposition...)
- solvers (some algebraic and differential equations)
- 2D geometry module
- plotting (2D and 3D)

Motivation

Why?

Motivation

Why?

- BSD licensed (like SciPy and NumPy) $\longrightarrow$ use it the way you want
- small, pure python $\longrightarrow$ easily include it your own projects
- It's in Debian, Ubuntu, Gentoo, Arch, Sage, ...

Motivation

Why Python?

Motivation

Why Python?

- widely used language (Google, NASA, YouTube, ...)
- easy for you to define your own data types and methods on it.
- very clean language that results in easy to read code.
- easy to learn (good online documentation, several good books, some free).
- a huge number of libraries: statistics, networking, databases, bioinformatic, physics, video games, 3d graphics, numerical computation (numpy and scipy)
- easy to interface C/C++/Fortran code (Cython and f2py).

SymPy

Other symbolic manipulation software: GiNaC, Giac, Qalculate, Yacas, Eigenmath, Axiom, PARI, Maxima, Sage, Singular, Mathomatic, Octave, ...
Problems:

- all use their own language (except GiNaC, Giac and Sage)
- GiNaC and Giac still too complicated (C++), difficult to extend

What we want

- Python library and that's it (no environment, no new language, nothing)
- Rich funcionality
- Pure Python (non Python modules could be optional) – works on Linux, Windows, Mac out of the box

SymPy console

Acutally, I didn't tell the full truth, we have one nice thing – isympy:

```
$ bin/isympy
Python 2.4.4 console for SymPy 0.5.6-hg. These commands wer
>>> from __future__ import division
>>> from sympy import *
>>> x, y, z = symbols('xyz')
>>> k, m, n = symbols('kmn', integer=True)


In [1]: integrate(ln(x), x)
Out[1]: -x + x*log(x)
```

Unicode prettyprinting

```
In [4]: a = Symbol("alpha")

In [5]: a
Out[5]: α

In [6]: b = Symbol("beta")

In [7]: Integral((a+b)**2, a)
Out[7]:
⌠
⎮         2
⎮ (α + β)  dα
⌡

In [8]: Integral((a+b)**2, a).doit()
Out[8]:
 3
α        2       2
── + α*β  + β*α
3
```

Recent changes in isympy:

- pretty printing by default
- use unicode printing if available

Sage

- Create a viable open source alternative to Maple, Mathematica, Matlab and Magma
- http://www.sagemath.org/
- aims to glue together every useful open source mathematics software package and provide a transparent interface to all of them

```
sage: limit(sin(x)/x, x=0)
1
sage: integrate(x+sinh(x), x)
cosh(x) + x^2/2
```

```
In [1]: limit(sin(x)/x, x, 0)
Out[1]: 1

In [2]: integrate(x+sinh(x), x)
Out[2]: (1/2)*x**2 + cosh(x)
```

History I

In 2005, I wanted to use symbolic mathematics in Python

- pyginac used boost-python, very slow compilation (30s per file),
- I wrote swiginac together with Ola Skavhaug in SWIG, it works, but too difficult to extend the GiNaC core behind it
- Is it really that difficult to have a system, that can calculate all I need and still be easy to extend?

Let's reinvent the wheel for the 35th time.

History II

- end of summer 2005: I implemented my first code, mostly translating ideas from GiNaC to Python.
- spring 2006: I discovered the Gruntz algorithm for limits
- end of summer 2006: I implemented limits in SymPy
- February 2007: Fabian Seoane joined and this was the boost to SymPy's development
- Google Summer of Code, SymPy is under the umbrella of Python Software Foundation, the Space Telescope Science Institute and Portland State University

Our Team

We try hard to work as a team:

Team

Ondřej Čertík, Fabian Seoane, Jurjen N.E. Bos, Mateusz Paprocki, Marc-Etienne M.Leveille, Brian Jorgensen, Jason Gedge, Robert Schwarz, Pearu Peterson, Fredrik Johansson, Chris Wu, Kirill Smelkov, Ulrich Hecht, Goutham Lakshminarayan, David Lawrence, Jaroslaw Tworek, David Marek, Bernhard R. Link, Andrej Tokarčík, Or Dvory, Saroj Adhikari, Pauli Virtanen, Robert Kern, James Aspnes, Nimish Telang, Abderrahim Kitouni, Pan Peng, Friedrich Hagedorn, Elrond der Elbenfuerst, Rizgar Mella, Felix Kaiser, Roberto Nobrega, David Roberts, Sebastian Krämer, Vinzent Steinberg, Riccardo Gori, Case Van Horsen, Štěpán Roučka, Ali Raza Syed, Stefano Maggiolo, Robert Cimrman, Bastian Weber, Sebastian Krause, Sebastian Kreft, Dan, Alan Bromborsky, Boris Timokhin, Robert, Andy R. Terrel, Hubert Tsang, Konrad Meyer, Henrik Johansson, Priit Laes, Freddie Witherden, Brian E. Granger

Development

How SymPy development is done:

- all patches have to be reviewed by at least one other developer
- all tests need to pass, all new functionality should be tested
- frequent releases (we try at least once a month)

the Schwarzschild solution in the General Relativity

spherically symmetric metric ($diag(-e^{\nu(r)}, e^{\lambda(r)}, r^2, r^2 \sin^2\theta)$) $\rightarrow$ Christoffel symbols $\rightarrow$ Riemann tensor $\rightarrow$ Einstein equations $\rightarrow$ solver

```
ondra@pc232:~/sympy/examples$ time python relativity.py
...
[SKIP]
...
----------------------------------------
metric:
-C1 - C2/r 0 0 0
0 1/(C1 + C2/r) 0 0
0 0 r**2 0
0 0 0 r**2*sin(\theta)**2

real 0m1.092s
user 0m1.024s
sys 0m0.068s
```

Symbolic limits

- Gruntz algorithm
- the algorithm is so simple that everyone should know how it works :)

Comparability classes

$$L \equiv \lim_{x \to \infty} \frac{\log |f(x)|}{\log |g(x)|}$$

We define $<$, $>$, $\sim$:

- $f > g$ when $L = \pm\infty$
  - f is greater than any power of g
  - f is more rapidly varying than g
  - f goes to ∞ or 0 faster than g
- $f < g$ when $L = 0$
  - f is lower than any power of g
  - ...
- $f \sim g$ when $L \neq 0, \pm\infty$
  - both f and g are bounded from above and below by suitable integral powers of the other

Examples:

$$2 < x < e^x < e^{x^2} < e^{e^x}$$

$$2 \sim 3 \sim -5$$

$$x \sim x^2 \sim x^3 \sim \frac{1}{x} \sim x^m \sim -x$$

$$e^x \sim e^{-x} \sim e^{2x} \sim e^{x+e^{-x}}$$

$$f(x) \sim \frac{1}{f(x)}$$

The Gruntz algorithm I

$$f(x) = e^{x+2e^{-x}} - e^x + \frac{1}{x}$$

$$\lim_{x\to\infty} f(x) = ?$$

Strategy:

- mrv set: the set of most rapidly varying subexpressions
  - $\{e^x, e^{-x}, e^{x+2e^{-x}}\}$
  - the same comparability class
- take an item ω converging to 0 at infinity
  - $\omega = e^{-x}$
  - if not present in the mrv set, use the relation $f(x) \sim \frac{1}{f(x)}$
- rewrite the mrv set using ω
  - $\{\frac{1}{\omega}, \omega, \frac{1}{\omega}e^{2\omega}\}$
- substitute back in $f(x)$ and expand in ω:
  - $f(x) = \frac{1}{x} - \frac{1}{\omega} + \frac{1}{\omega}e^{2\omega} = 2 + \frac{1}{x} + 2\omega + O(\omega^2)$

The Gruntz algorithm II

Crucial observation: ω is from the mrv set, so

$$f(x) = e^{x+2e^{-x}} - e^x + \frac{1}{x} = 2 + \frac{1}{x} + 2\omega + O(\omega^2) \to 2 + \frac{1}{x}$$

- We iterate until we get just a number, the final limit
- Gruntz proved this always works and converges in his Ph.D. thesis

Generally:

$$f(x) = \underbrace{\cdots}_{\infty} + \underbrace{\frac{C_{-2}(x)}{\omega^2}}_{\infty} + \underbrace{\frac{C_{-1}(x)}{\omega}}_{\infty} + C_0(x) + \underbrace{C_1(x)\omega}_{0} + \underbrace{O(\omega^2)}_{0}$$

- we look at the lowest power of ω
- the limit is one of: 0, $\lim_{x\to\infty} C_0(x)$, ∞

The question of speed

- Being pure Python has many advantages
- speed is good enough for many purposes
- use Cython (or C++, C) as an optional module to speed the core up