SymPy — A System for Exploring Symbolic Mathematics

Mateusz Paprocki <mattpap@gmail.com>

Wrocław University of Technology

May 19, 2009

What is SymPy?

- A Python library for symbolic mathematics

```
>>> from sympy import *
>>> x = Symbol('x')

>>> limit(sin(pi*x)/x, x, 0)
pi

>>> integrate(x + sinh(x), x)
(1/2)*x**2 + cosh(x)

>>> diff(_, x)
x + sinh(x)
```

Why symbolics matters?

Consider a real–valued function:

$$f(x) = \frac{1}{x^{\log\left(\log\left(\log\left(\log\left(\frac{1}{x}\right)\right)\right)\right)-1}}$$

on interval $(0, \infty)$ and take its limit in right neighbourhood of 0:

$$\lim_{x \to 0^+} f(x) = \ldots$$

Lets try numerical evaluation

Why symbolics matters?

Evaluate $f(x_k)$ where $x_k = 10^{-10^k}$:

k	$f(x_k)$
1	$2.17 \cdot 10^{-9}$
2	$4.87 \cdot 10^{-48}$
3	$1.56 \cdot 10^{-284}$
4	$3.42 \cdot 10^{-1641}$
5	$1.06 \cdot 10^{-7836}$

We clearly see that $f(x) \longrightarrow 0$ when $x \longrightarrow 0^+$. Right?

Lets try symbolic computation

Why symbolics matters?

Lets use a symbolic manipulation system to solve our problem:

```
In [1]: f = 1/x**(log(log(log(log(1/x)))) - 1)

In [2]: limit(f, x, 0)
Out[2]: oo
```

So our predictions based on numerical evaluation failed:

$$\lim_{x \to 0^+} f(x) = \infty$$

Why reinvent the wheel for the 37th time?

There are numerous symbolic manipulation systems:

- Proprietary software:
  - Mathematica, Maple, Magma, ...
- Open Source software:
  - AXIOM, GiNaC, Maxima, PARI, Sage, Singular, Yacas, ...

Problems:

- all invent their own language
  - need learn yet another language
  - separation into core and library
  - hard to extend core functionality
  - except: GiNaC and Sage
- all need quite some time to compile
  - slow development cycle

What we want to achieve?

- pure Python library
  - no new environment, language, ...
  - works out of the box on any platform
  - non Python modules could be optional
- simple design
  - small code base
  - easy to extend
- rich functionality
  - support most important fields of mathematics
  - implement modern algorithms, e.g. Gruntz algorithm
- use Cython for time critical code
  - optional, accompanying interpreted version
- liberal licence: BSD
  - freedom on ways of using SymPy

Why we chose Python?

- widely used language
  - Google, NASA, ...
- very clean language
  - simple syntactics and semantics
  - usually one way to do things
  - easy to read and maintain
- huge number of libraries
  - numerical computation: NumPy, SciPy
  - physics, simulation, bioinformatics
  - visualisation, 3D graphics, plotting
  - databases, networking, ...
- easy to bind with other code
  - C/C++ via native API or Cython
  - Fortran using f2py bindings

But wait, there is Sage . . .

Sage aims to:

- create a viable free open source alternative to Maple, Mathematica, Matlab and Magma
- glue together useful mathematics software packages and provide transparent interface to them
- Cons:
  - difficult to use as a library
  - Sage is a software distribution
  - very large in size and with long build times
  - Sage prefers to use a preparser (it can be turned off)
- Pros:
  - it includes most recent version of SymPy
  - nice interface to lots of packages, easy to install

Sage vs SymPy

- Sage example:

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

- SymPy example:

```
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)
```

Capabilities

What SymPy can do?

- core functionality
  - differentiation, truncated series
  - pattern matching, substitutions
  - non–commutative algebras
  - assumptions engine, logic
- symbolic ...
  - integration, summation
  - limits
- polynomial algebra
  - Gröbner bases computation
  - multivariate factorization
- matrix algebra
- equations solvers
  - algebraic, transcendental
  - recurrence, differential
- systems solvers
  - linear, polynomial
- pretty–printing
  - Unicode, ASCII
  - LaTeX, MathML
- 2D & 3D plotting
- ...

Unicode pretty–printing

```
Python 2.6.2 console for SymPy 0.6.5-git (cache: off)

In [1]: var('mu')
Out[1]: μ

In [2]: M = Matrix(4, 4, lambda i,j: i*j + mu)

In [3]: M
Out[3]:
⎡μ    μ      μ      μ  ⎤
⎢                      ⎥
⎢μ  1 + μ  2 + μ  3 + μ⎥
⎢                      ⎥
⎢μ  2 + μ  4 + μ  6 + μ⎥
⎢                      ⎥
⎣μ  3 + μ  6 + μ  9 + μ⎦

In [4]: M.eigenvals()
Out[4]:
⎧                 ___________________________                         ___________________________    ⎫
⎪                ╱                         2                         ╱                         2     ⎪
⎨               ╲╱  196 + 32⋅μ + 16⋅μ                               ╲╱  196 + 32⋅μ + 16⋅μ            ⎬
⎪0: 2, 7 + 2⋅μ + ───────────────────────────: 1, 7 + 2⋅μ - ───────────────────────────: 1⎪
⎩                            2                                           2                           ⎭
```

Example

Computing minimal polynomial of an algebraic number (1)

```
In [1]: from sympy import *

In [2]: y = sqrt(2) + sqrt(3) + sqrt(6)

In [3]: var('a,b,c')
Out[3]: (a, b, c)

In [4]: f = [a**2 - 2, b**2 - 3, c**2 - 6, x - a-b-c]

In [5]: g = groebner(f, a,b,c,x, order='lex')

In [6]: g[-1]
Out[6]: 529 - 1292*x**2 + 438*x**4 - 44*x**6 + x**8
```

Example

Computing minimal polynomial of an algebraic number (2)

```
In [7]: u, v = factor(_).args

In [8]: u
Out[8]: 23 - 48*x + 22*x**2 - x**4

In [9]: simplify(u.subs(x, y))
Out[9]: -96*2**(1/2) - 96*3**(1/2) - 96*6**(1/2)

In [10]: v
Out[10]: 23 + 48*x + 22*x**2 - x**4

In [11]: simplify(v.subs(x, y))
Out[11]: 0
```

Contact

How to get involved?

- Visit our main web site:
  - `www.sympy.org`
- and additional web sites:
  - `wiki.sympy.org`
  - `docs.sympy.org`
  - `live.sympy.org`
- Contact us on our mailing list:
  - `sympy@googlegroups.com`
- or/and IRC channel:
  - `#sympy` on FreeNode
- Clone source repository:

```
git clone git://git.sympy.org/sympy.git
```

Conclusion

- What is done
  - basic core functionality and class structure
  - algorithms for most fields of mathematics
  - efficient workflow established
    - GIT + patch review
- What is not done
  - full test coverage
  - optimizations in Cython
  - robust ODE and PDE solvers
    - under development withing GSoC
  - robust assumptions engine
    - under development withing GSoC
  - robust symbolic integrator, ...
- What won't be done
  - GUI, notebook etc.

Thank you for your attention!