

SymPy: Symbolic Mathematics in Pure Python

Mateusz Paprocki <mattpap@gmail.com>

SymPy Development Team
University of Nevada, Reno

March 2, 2011

Presentation plan

- Introduction to SymPy
 - What is SymPy and why we need it?
 - Pure Python pros and cons
 - List of features
 - How to contribute
- Examples
 - Graph k -coloring with Gröbner bases

What is SymPy?

- A pure 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)
```

What is SymPy?

- A pure 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)
```

Symbolic capabilities (1)

Lets consider the following function (Gruntz, 1996):

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

We would like to compute the following limit:

$$\lim_{x \rightarrow 0^+} f(x) = ?$$

Lets try numerical approach:

k	1	2	3	4	5
$O(f(10^{-10^k}))$	10^{-9}	10^{-48}	10^{-284}	10^{-1641}	10^{-7836}

Symbolic capabilities (1)

Lets consider the following function (Gruntz, 1996):

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

We would like to compute the following limit:

$$\lim_{x \rightarrow 0^+} f(x) = ?$$

Lets try numerical approach:

k	1	2	3	4	5
$O(f(10^{-10^k}))$	10^{-9}	10^{-48}	10^{-284}	10^{-1641}	10^{-7836}

Symbolic capabilities (1)

Lets consider the following function (Gruntz, 1996):

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

We would like to compute the following limit:

$$\lim_{x \rightarrow 0^+} f(x) = ?$$

Lets try numerical approach:

k	1	2	3	4	5
$O(f(10^{-10^k}))$	10^{-9}	10^{-48}	10^{-284}	10^{-1641}	10^{-7836}

Symbolic capabilities (2)

This suggests that:

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

We can use SymPy to **prove this guess wrong**:

```
In [1]: from sympy import var, log, limit
In [2]: var('x')
Out[2]: x
In [3]: f = x**(1 - log(log(log(log(1/x))))))
In [4]: limit(f, x, 0)
Out[4]: oo
```

Symbolic capabilities (2)

This suggests that:

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

We can use SymPy to **prove** this guess **wrong**:

```
In [1]: from sympy import var, log, limit
In [2]: var('x')
Out [2]: x
In [3]: f = x**(1 - log(log(log(log(1/x))))))
In [4]: limit(f, x, 0)
Out [4]: oo
```

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 to 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

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 to 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 does pure Python mean?

- simply download and start computing

```
$ git clone git://github.com/sympy/sympy.git
```

```
$ cd sympy
```

```
$ bin/isympy
```

- no dependencies by default (besides Python)
 - works under Python 2.4, 2.5, 2.6, 2.7
 - support for Python 3.x under development
 - extra dependencies allowed for additional features
 - `gmpy`, `Cython` — speed improvement
 - `Pyglet`, `Matplotlib` — 2D & 3D plotting
 - `IPython` — interactive sessions
- preview every algorithm implemented in SymPy
 - `source(obj)`, `obj??`

What does pure Python mean?

- simply download and start computing

```
$ git clone git://github.com/sympy/sympy.git
$ cd sympy
$ bin/isympy
```
- no dependencies by default (besides Python)
 - works under Python 2.4, 2.5, 2.6, 2.7
 - support for Python 3.x under development
 - extra dependencies allowed for additional features
 - [gmpy](#), [Cython](#) — speed improvement
 - [Pyglet](#), [Matplotlib](#) — 2D & 3D plotting
 - [IPython](#) — interactive sessions
- preview every algorithm implemented in SymPy
 - `source(obj)`, `obj??`

What does pure Python mean?

- simply download and start computing

```
$ git clone git://github.com/sympy/sympy.git
$ cd sympy
$ bin/isympy
```
- no dependencies by default (besides Python)
 - works under Python 2.4, 2.5, 2.6, 2.7
 - support for Python 3.x under development
 - extra dependencies allowed for additional features
 - [gmpy](#), [Cython](#) — speed improvement
 - [Pyglet](#), [Matplotlib](#) — 2D & 3D plotting
 - [IPython](#) — interactive sessions
- preview every algorithm implemented in SymPy
 - `source(obj)`, `obj??`

Issues with pure Python approach (1)

- you have to define symbols before using them

```
In [1]: t
(...)
NameError: name 't' is not defined

In [2]: var('t')
Out[2]: t

In [3]: symbols('a0:5')
Out[3]: (a0, a1, a2, a3, a4)
```

- $1/3$ is not what you may expect

```
In [3]: 1/3
Out[3]: 0.3333333333333333

In [4]: Rational(1, 3)
Out[4]: 1/3

In [5]: S("1/3")
Out[5]: 1/3
```

Issues with pure Python approach (1)

- you have to define symbols before using them

```
In [1]: t
(... )
NameError: name 't' is not defined

In [2]: var('t')
Out[2]: t

In [3]: symbols('a0:5')
Out[3]: (a0, a1, a2, a3, a4)
```

- $1/3$ is not what you may expect

```
In [3]: 1/3
Out[3]: 0.3333333333333333

In [4]: Rational(1, 3)
Out[4]: 1/3

In [5]: S("1/3")
Out[5]: 1/3
```

Issues with pure Python approach (2)

- `^` is not exponentiation operator

```
In [6]: 2^3
```

```
Out [6]: 1
```

```
In [7]: 2**3
```

```
Out [7]: 8
```

- large(er) computations may require tweaking Python

```
In [8]: f = Poly(range(100), x)
```

```
In [9]: horner(f)
```

```
Out [9]:
```

```
(...)
```

```
RuntimeError: maximum recursion depth exceeded
```

```
In [10]: %time _ = horner(f)
```

```
CPU times: user 0.01 s, sys: 0.00 s, total: 0.01 s
```

```
Wall time: 0.01 s
```

Issues with pure Python approach (2)

- `^` is not exponentiation operator

```
In [6]: 2^3
```

```
Out [6]: 1
```

```
In [7]: 2**3
```

```
Out [7]: 8
```

- large(er) computations may require tweaking Python

```
In [8]: f = Poly(range(100), x)
```

```
In [9]: horner(f)
```

```
Out [9]:
```

```
(...)
```

```
RuntimeError: maximum recursion depth exceeded
```

```
In [10]: %time _ = horner(f)
```

```
CPU times: user 0.01 s, sys: 0.00 s, total: 0.01 s
```

```
Wall time: 0.01 s
```

List of SymPy's modules (1)

- assumptions** assumptions engine
 - concrete** symbolic products and summations
 - core** basic class structure: Basic, Add, Mul, Pow, ...
- functions** elementary and special functions
- galgebra** geometric algebra
- geometry** geometric entities
- integrals** symbolic integrator
- interactive** interactive sessions (e.g. IPython)
 - logic** boolean algebra, theorem proving
- matrices** linear algebra, matrices
- mpmath** fast arbitrary precision numerical math

List of SymPy's modules (2)

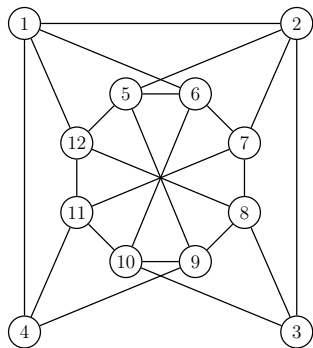
- `ntheory` number theoretical functions
- `parsing` Mathematica and Maxima parsers
- `physics` physical units, quantum stuff
- `plotting` 2D and 3D plots using Pyglet
 - `polys` polynomial algebra, factorization
- `printing` pretty-printing, code generation
 - `series` symbolic limits and truncated series
- `simplify` rewrite expressions in other forms
- `solvers` algebraic, recurrence, differential
- `statistics` standard probability distributions
- `utilities` test framework, compatibility stuff

How to get involved?

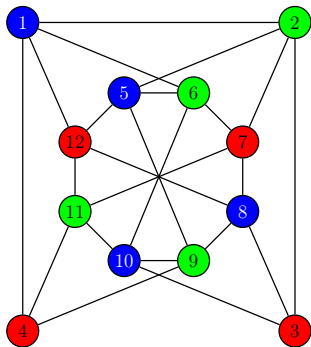
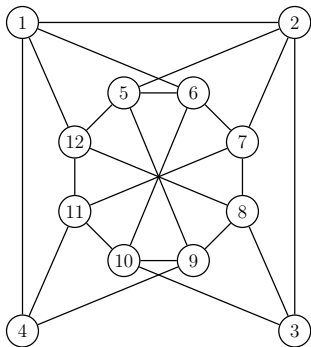
- Visit our main web site:
 - www.sympy.org
- and additional web sites:
 - docs.sympy.org
 - wiki.sympy.org
 - live.sympy.org
- Contact us on our mailing list:
 - sympy@googlegroups.com
- or/and IRC channel:
 - #sympy on FreeNode
- Clone source repository:

```
git clone git://github.com/sympy/sympy.git
```

Graph k -coloring with Gröbner bases (1)



Graph k -coloring with Gröbner bases (1)



Graph k -coloring with Gröbner bases (2)

Given a graph $\mathcal{G}(V, E)$ we write two sets of equations:

- I_k — allow one of k colors per vertex

$$I_k = \{x_i^k - 1 : i \in V\}$$

- I_G — adjacent vertices have different colors assigned

$$I_G = \{x_i^{k-1} + x_i^{k-2}x_j + \dots + x_ix_j^{k-2} + x_j^{k-1} : (i, j) \in E\}$$

Next we solve $I_k \cup I_G$ using the Gröbner bases method.

Graph k -coloring with Gröbner bases (2)

Given a graph $\mathcal{G}(V, E)$ we write two sets of equations:

- I_k — allow one of k colors per vertex

$$I_k = \{x_i^k - 1 : i \in V\}$$

- $I_{\mathcal{G}}$ — adjacent vertices have different colors assigned

$$I_{\mathcal{G}} = \{x_i^{k-1} + x_i^{k-2}x_j + \dots + x_ix_j^{k-2} + x_j^{k-1} : (i, j) \in E\}$$

Next we solve $I_k \cup I_{\mathcal{G}}$ using the Gröbner bases method.

Graph k -coloring with Gröbner bases (2)

Given a graph $\mathcal{G}(V, E)$ we write two sets of equations:

- I_k — allow one of k colors per vertex

$$I_k = \{x_i^k - 1 : i \in V\}$$

- $I_{\mathcal{G}}$ — adjacent vertices have different colors assigned

$$I_{\mathcal{G}} = \{x_i^{k-1} + x_i^{k-2}x_j + \dots + x_ix_j^{k-2} + x_j^{k-1} : (i, j) \in E\}$$

Next we solve $I_k \cup I_{\mathcal{G}}$ using the Gröbner bases method.

Graph k -coloring with Gröbner bases (2)

Given a graph $\mathcal{G}(V, E)$ we write two sets of equations:

- I_k — allow one of k colors per vertex

$$I_k = \{x_i^k - 1 : i \in V\}$$

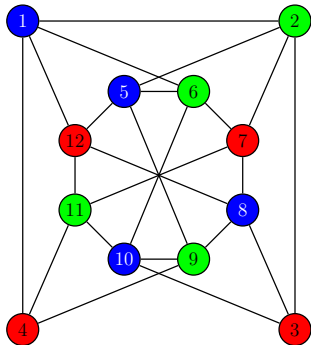
- $I_{\mathcal{G}}$ — adjacent vertices have different colors assigned

$$I_{\mathcal{G}} = \{x_i^{k-1} + x_i^{k-2}x_j + \dots + x_ix_j^{k-2} + x_j^{k-1} : (i, j) \in E\}$$

Next we solve $I_k \cup I_{\mathcal{G}}$ using the Gröbner bases method.

Graph k -coloring with Gröbner bases (3)

$$\left\{ \begin{array}{l} x_1 + x_{11} + x_{12}, \\ x_2 - x_{11}, \\ x_3 - x_{12}, \\ x_4 - x_{12}, \\ x_5 + x_{11} + x_{12}, \\ x_6 - x_{11}, \\ x_7 - x_{12}, \\ x_8 + x_{11} + x_{12}, \\ x_9 - x_{11}, \\ x_{10} + x_{11} + x_{12}, \\ x_{11}^2 + x_{11}x_{12} + x_{12}^2, \\ x_{12}^3 - 1 \end{array} \right\}$$



Graph k -coloring with Gröbner bases (4)

Here is how to solve 3-coloring problem in SymPy:

```
In [1]: V = range(1, 12+1)
In [2]: E = [(1,2),(2,3),(1,4),(1,6),(1,12),(2,5),(2,7),
(3,8),(3,10),(4,11),(4,9),(5,6),(6,7),(7,8),(8,9),(9,10),
(10,11),(11,12),(5,12),(5,9),(6,10),(7,11),(8,12)]

In [3]: X = [ Symbol('x' + str(i)) for i in V ]
In [4]: E = [ (X[i-1], X[j-1]) for i, j in E ]

In [5]: I3 = [ x**3 - 1 for x in X ]
In [6]: Ig = [ x**2 + x*y + y**2 for x, y in E ]

In [7]: G = groebner(I3 + Ig, X, order='lex')

In [8]: G != [1]
Out[8]: True
```

Graph k -coloring with Gröbner bases (4)

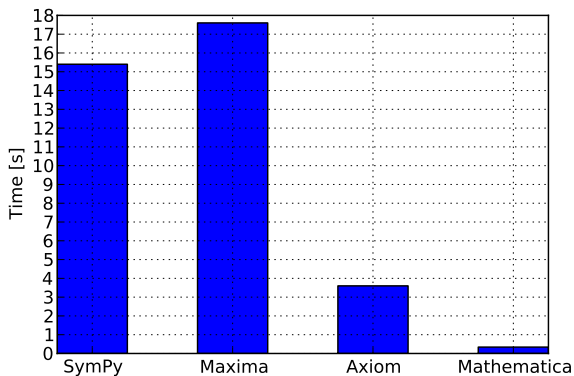
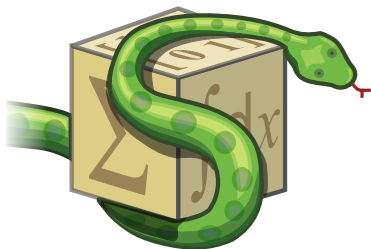


Figure: Average timing of Gröbner basis computation

Thank you for your attention!

Questions, remarks, discussion . . .



SymPy