Symbolic manipulation in pure Python. Is it feasible?

Mateusz Paprocki <mattpap@gmail.com>

Wrocław University of Technology
SymPy Development Team

May 12, 2010

Presentation plan

- A few words about the author
- Short introduction to SymPy
  - the main goals of the project
  - listing of SymPy's capabilities
- The main topic
  - polynomials in SymPy
  - compare with other systems
  - examples, more examples ...

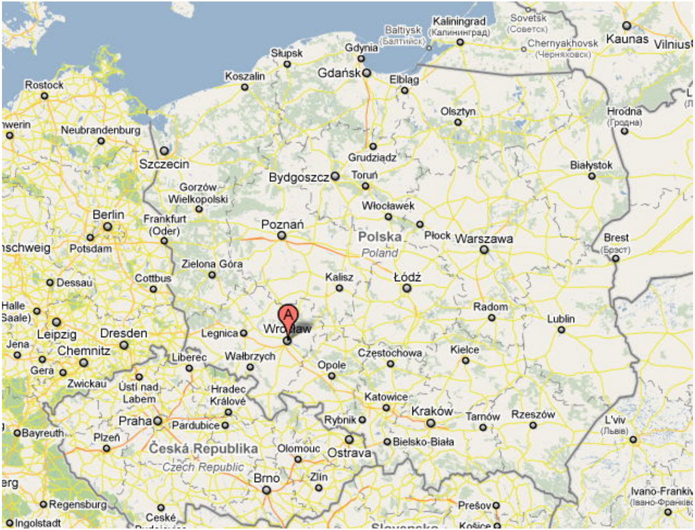
Presentation plan

- A few words about the author
- Short introduction to SymPy
  - the main goals of the project
  - listing of SymPy's capabilities
- The main topic
  - polynomials in SymPy
  - compare with other systems
  - examples, more examples ...

Presentation plan

- A few words about the author
- Short introduction to SymPy
  - the main goals of the project
  - listing of SymPy's capabilities
- The main topic
  - polynomials in SymPy
  - compare with other systems
  - examples, more examples ...

A few words about the author



Wrocław University of Technology (1)

Wrocław University of Technology (2)

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)
```

My role in the project

A short historical background

- cooperation started in March 2007
  - a few simple bugfixes and improvements
- next came Google Summer of Code 2007
  - algorithms for solving recurrence relations
  - algorithms for definite and indefinite summations
- and this is how it works:
  - algorithms for symbolic integration
  - algebraic structures, polynomials
  - expression simplification, . . .
- else:
  - GSoC 2009, 2010 mentor (PSU, PSF)
  - tutorial at EuroSciPy '09
  - master's thesis

My role in the project

A short historical background

- cooperation started in March 2007
  - a few simple bugfixes and improvements
- next came Google Summer of Code 2007
  - algorithms for solving recurrence relations
  - algorithms for definite and indefinite summations
- and this is how it works:
  - algorithms for symbolic integration
  - algebraic structures, polynomials
  - expression simplification, . . .
- else:
  - GSoC 2009, 2010 mentor (PSU, PSF)
  - tutorial at EuroSciPy '09
  - master's thesis

My role in the project

A short historical background

- cooperation started in March 2007
  - a few simple bugfixes and improvements
- next came Google Summer of Code 2007
  - algorithms for solving recurrence relations
  - algorithms for definite and indefinite summations
- and this is how it works:
  - algorithms for symbolic integration
  - algebraic structures, polynomials
  - expression simplification, . . .
- else:
  - GSoC 2009, 2010 mentor (PSU, PSF)
  - tutorial at EuroSciPy '09
  - master's thesis

My role in the project

A short historical background

- cooperation started in March 2007
  - a few simple bugfixes and improvements
- next came Google Summer of Code 2007
  - algorithms for solving recurrence relations
  - algorithms for definite and indefinite summations
- and this is how it works:
  - algorithms for symbolic integration
  - algebraic structures, polynomials
  - expression simplification, ...
- else:
  - GSoC 2009, 2010 mentor (PSU, PSF)
  - tutorial at EuroSciPy '09
  - master's thesis

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

List of SymPy's modules (1)

concrete	symbolic products and summations
core	Basic, Add, Mul, Pow, Function, ...
functions	elementary and special functions
galgebra	geometric algebra
geometry	geometric entities
integrals	symbolic integrator
interactive	for setting up pretty–printing
logic	new assumptions engine, boolean functions
matrices	Matrix class, orthogonalization etc.
mpmath	fast arbitrary precision numerical math

List of SymPy's modules (2)

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

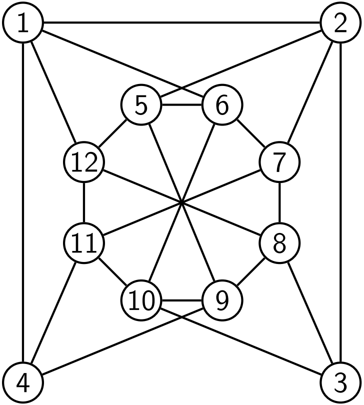
How to get involved?

- 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://git.sympy.org/sympy.git
```

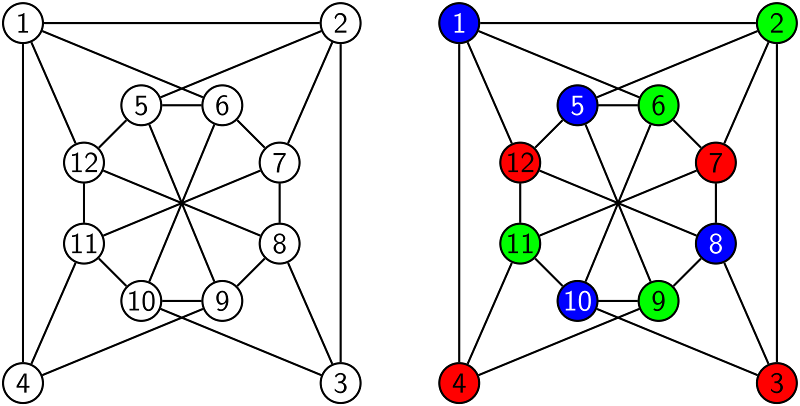
The first example

Vertex k–coloring of graphs



The first example

Vertex k–coloring of graphs



The first example

Graph coloring with Gröbner bases (1)

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

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

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

The first example

Graph coloring with Gröbner bases (1)

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

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

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

The first example

Graph coloring with Gröbner bases (1)

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

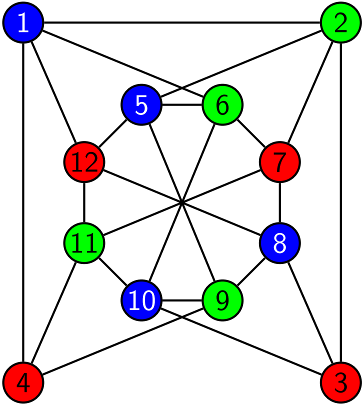
$$I_{\mathcal{G}} = \{x_i^{k-1} + x_i^{k-2}x_j + \ldots + x_i x_j^{k-2} + x_j^{k-1} : (i, j) \in E\}$$

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

The first example

Graph coloring with Gröbner bases (2)

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



The first example

Graph coloring with Gröbner bases 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]: Z = [ (X[i-1], X[j-1]) for i, j in E ]

In [5]: U = [ x**3 - 1 for x in X ]
In [6]: V = [ x**2 + x*y + y**2 for x, y in Z ]

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

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

A comparison with other systems

Graph coloring with Gröbner bases in Maxima

```
(i1) load(grobner);

(i2) 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]];

(i3) X: makelist(concat("x", i), i, 1, 12);

(i4) U: makelist(X[i]^3 - 1, i, 1, 12);
(i5) V: [];

(i6) for e in E do
          V: endcons(X[e[1]]^2 + X[e[1]]*X[e[2]] + X[e[2]]^2, V);

(i7) G: poly_reduced_grobner(append(U, V), X);

(i8) is(notequal(G, [1]));
(o8) true
```

A comparison with other systems

Graph coloring with Gröbner bases in Axiom

```
(1) -> 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]];

(2) -> X := [ concat("x", i::String)::Symbol for i in 1..12 ];
(3) -> Z := [ [X.(e.1), X.(e.2)] for e in E ];

(4) -> U := [ x**3 - 1 for x in X ];
(5) -> V := [ z.1**2 + z.1*z.2 + z.2**2 for z in Z];

(6) -> G := groebner([ w::DMP(X, INT) for w in concat(U, V) ]);

(7) -> (G ~= [1]) @ Boolean
   (7) true
```

A comparison with other systems

Graph coloring with Gröbner bases in Mathematica

```
In[1]:= Unprotect[E];
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 := Table[Symbol["x" <> ToString[i]], {i,1,n}]
In[4]:= h[{i_, j_}] := X[[i]]^2 + X[[i]] X[[j]] + X[[j]]^2

In[5]:= U := Map[(#^3-1)&, X]
In[6]:= V := Map[h, E]

In[7]:= G := GroebnerBasis[Join[U, V], X]

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

And what about the speed of computations?

In the case of our example . . .

	SymPy	Maxima	Axiom	Mathematica
Time [s]	15.4	17.6	3.6	0.34

Thank you for your attention!

SymPy