SymPy — a library for symbolic mathematics in pure Python

Mateusz Paprocki <mattpap@gmail.com>

Wrocław University of Technology
SymPy Development Team

July 24, 2009

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)
```

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 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 the interpreted version
- liberal licence: BSD
  - freedom on ways of using SymPy

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 the interpreted version
- liberal licence: BSD
  - freedom on ways of using SymPy

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 the interpreted version
- liberal licence: BSD
  - freedom on ways of using SymPy

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 the interpreted version
- liberal licence: BSD
  - freedom on ways of using SymPy

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 the 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

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

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

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

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

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
- . . .

ASCII pretty–printing

```
Python 2.6.2 console for SymPy 0.6.5.rc2-git (cache: off)

In [1]: var('mu')
Out[1]: mu

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

In [3]: M
Out[3]:
[mu    mu       mu       mu   ]
[                             ]
[mu  1 + mu  2 + mu  3 + mu]
[                             ]
[mu  2 + mu  4 + mu  6 + mu]
[                             ]
[mu  3 + mu  6 + mu  9 + mu]

In [4]: M.eigenvals()
Out[4]:

                         ______________________                          ______________________
                        /                     2                         /                     2
                      \/  196 + 32*mu + 16*mu                         \/  196 + 32*mu + 16*mu
{0: 2, 7 + 2*mu + -------------------------: 1, 7 + 2*mu - -------------------------: 1}
                                  2                                        2
```

Unicode pretty–printing

```
Python 2.6.2 console for SymPy 0.6.5.rc2-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                      ⎭
```

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

Internals

So, how does SymPy work?

```
In [1]: 7
Out[1]: 7

In [2]: type(_)
Out[2]: <type 'int'>

In [3]: sympify(7)
Out[3]: 7

In [4]: type(_)
Out[4]: <class 'sympy.core.numbers.Integer'>

In [5]: sympify('factor(x**5+1, x)')
Out[5]: (1 + x)*(1 - x + x**2 - x**3 + x**4)
```

Internals

Object oriented model

- Basic
  - Add
  - Mul
  - Pow
  - Symbol
  - Integer
  - Rational
  - Function
    - sin
    - cos
    - ...

Each class has `__new__` method:

- automatic simplification of arguments
- no intermediate classes construction

Example:

- `Add(Add(x,y), x)` becomes `Add(Mul(2,x), y)`

Internals

Automatic expression evaluation

```
In [1]: Add(x, 7, x, y, -2)
Out[1]: 5 + y + 2*x

In [2]: x + 7 + x + y - 2
Out[2]: 5 + y + 2*x

In [3]: Mul(x, 7, x, y, 2)
Out[3]: 14*y*x**2

In [4]: x*7*x*y*2
Out[4]: 14*y*x**2

In [5]: sin(2*pi)
Out[5]: 0
```

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

In [7]: F = factors(_, x)[1]
```

Example

Computing minimal polynomial of an algebraic number (2)

```
In [8]: len(F)
Out[8]: 2

In [9]: (u, _), (v, _) = F

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

In [11]: simplify(u.subs(x, y))
Out[11]: -96*2**(1/2) - 96*3**(1/2) - 96*6**(1/2)

In [12]: v
Out[12]: 23 + 48*x + 22*x**2 - x**4

In [13]: simplify(v.subs(x, y))
Out[13]: 0
```

But wait, isn't this slow?

Lets compare SymPy with SAGE (1)

Sum of terms in the following form:

$$x^k (ky)^{2k} z^{y^k}, k \in [1, 200]$$

```
In [1]: f = lambda k: x**k*(k*y)**(2*k)*z**y**k
In [2]: %timeit a = Add(*map(f, xrange(1, 200)))
10 loops, best of 3: 146 ms per loop
```

```
sage: f = lambda k: x**k*(k*y)**(2*k)*z**y**k
sage: %timeit a = sum(map(f, xrange(1, 200)))
10 loops, best of 3: 30.9 ms per loop
```

But wait, isn't this slow?

Lets compare SymPy with SAGE (2)

Sum of terms in the following form:

$$sin(kx)^k(k \cdot cos(ky))^{2k}, k \in [1, 200]$$

```
In [3]: g = lambda k: sin(k*x)**k*(k*cos(k*y))**(2*k)
In [4]: %timeit a = Add(*map(g, xrange(1, 200)))
10 loops, best of 3: 527 ms per loop
```

```
sage: g = lambda k: sin(k*x)**k*(k*cos(k*y))**(2*k)
sage: %timeit a = sum(map(g, xrange(1, 200)))
10 loops, best of 3: 38.6 ms per loop
```

But wait, isn't this slow?

Lets compare SymPy with SAGE (3)

Cyclotomic factorization of a polynomial:

```
In [5]: %timeit a = factor(x**462 + 1)
10 loops, best of 3: 215 ms per loop
```

```
sage: %timeit a = factor(x**462 + 1)
10 loops, best of 3: 637 ms per loop
```

Factorization of a multivariate polynomial:

```
In [6]: %timeit a = factor(x**20 - z**5*y**20)
10 loops, best of 3: 614 ms per loop
```

```
sage: %timeit a = factor(x**20 - z**5*y**20)
10 loops, best of 3: 44.4 ms per loop
```

But wait, isn't this slow?

Lets compare SymPy with SAGE (3)

Cyclotomic factorization of a polynomial:

```
In [5]: %timeit a = factor(x**462 + 1)
10 loops, best of 3: 215 ms per loop
```

```
sage: %timeit a = factor(x**462 + 1)
10 loops, best of 3: 637 ms per loop
```

Factorization of a multivariate polynomial:

```
In [6]: %timeit a = factor(x**20 - z**5*y**20)
10 loops, best of 3: 614 ms per loop
```

```
sage: %timeit a = factor(x**20 - z**5*y**20)
10 loops, best of 3: 44.4 ms per loop
```

What can be done to improve speed?

- use better algorithms, if available, e.g.
  - use modular techniques in polynomial problems
- rewrite core modules in Cython
  - better (?): use pure mode

```
import cython

@cython.locals(i=cython.int)
cpdef divisors(int n)
```

- improve CPython, e.g.
  - see unladen–swallow project

Our workflow

- we use GIT for source code management
  - there is one official repository with master branch
  - each developer has a development repository
  - patch review + branch tracking
- each public function must have
  - a test suite associated
  - a docstring written
  - examples prepared
- all tests must pass in the official repository
- we use buildbots to test different architectures
  - amd64-py2.4, amd64-py2.5, amd64-py2.6
  - i386-py2.4, i386-py2.5, i386-py2.6
  - amd64-py2.5-Qnew-sympy-tests
- we use ReST to write documentation

Our workflow

- we use GIT for source code management
  - there is one official repository with master branch
  - each developer has a development repository
  - patch review + branch tracking
- each public function must have
  - a test suite associated
  - a docstring written
  - examples prepared
- all tests must pass in the official repository
- we use buildbots to test different architectures
  - amd64-py2.4, amd64-py2.5, amd64-py2.6
  - i386-py2.4, i386-py2.5, i386-py2.6
  - amd64-py2.5-Qnew-sympy-tests
- we use ReST to write documentation

Our workflow

- we use GIT for source code management
  - there is one official repository with master branch
  - each developer has a development repository
  - patch review + branch tracking
- each public function must have
  - a test suite associated
  - a docstring written
  - examples prepared
- all tests must pass in the official repository
- we use buildbots to test different architectures
  - amd64-py2.4, amd64-py2.5, amd64-py2.6
  - i386-py2.4, i386-py2.5, i386-py2.6
  - amd64-py2.5-Qnew-sympy-tests
- we use ReST to write documentation

Our workflow

- we use GIT for source code management
  - there is one official repository with master branch
  - each developer has a development repository
  - patch review + branch tracking
- each public function must have
  - a test suite associated
  - a docstring written
  - examples prepared
- all tests must pass in the official repository
- we use buildbots to test different architectures
  - amd64-py2.4, amd64-py2.5, amd64-py2.6
  - i386-py2.4, i386-py2.5, i386-py2.6
  - amd64-py2.5-Qnew-sympy-tests
- we use ReST to write documentation

Our workflow

- we use GIT for source code management
  - there is one official repository with master branch
  - each developer has a development repository
  - patch review + branch tracking
- each public function must have
  - a test suite associated
  - a docstring written
  - examples prepared
- all tests must pass in the official repository
- we use buildbots to test different architectures
  - amd64-py2.4, amd64-py2.5, amd64-py2.6
  - i386-py2.4, i386-py2.5, i386-py2.6
  - amd64-py2.5-Qnew-sympy-tests
- we use ReST to write documentation

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
  - robust
    - integrator
    - assumptions engine
    - ODE and PDE solvers, ...
- What won't be done:
  - GUI, notebook etc.

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
  - robust
    - integrator
    - assumptions engine
    - ODE and PDE solvers, ...
- What won't be done:
  - GUI, notebook etc.

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
  - robust
    - integrator
    - assumptions engine
    - ODE and PDE solvers, ...
- What won't be done:
  - GUI, notebook etc.

Thank you for your attention!

SymPy