ACM Communications in Computer Algebra, Issue 178, Vol. 45, No. 4, December 2011

Open Source Computer Algebra Systems

SymPy

David Joyner[*], Ondřej Čertík[†], Aaron Meurer[‡], Brian E. Granger[§]

This survey[1] will look at SymPy, a free and open source computer algebra system started in 2005 by the second author (O.Č.). It is written entirely in Python, available from `http://sympy.org`. SymPy is licensed under the "modified BSD" license, as is its beautiful logo designed by Fredrik Johansson.

SymPy is also used in other mathematics and scientific software systems. For example, SfePy (`http://sfepy.org/`) includes SymPy. SfePy is a software for solving systems of coupled partial differential equations (PDEs) by the finite element method in 2D and 3D. NiPy (`http://nipy.org`), a neuroimaging package for Python, uses SymPy for its statistical formulae. The software Sage (`http://www.sagemath.org/`) also includes SymPy.

SymPy's latest release (as of October 2011) is 0.7.1.

1 SymPy's history and language

1.1 History

SymPy was started by O.Č. in 2005 while a physics undergraduate at the Charles University in Prague. He wrote initial code during the summer, then he improved it during the summer 2006, but otherwise the project stalled until February 2007, when Fabian Pedregosa joined the project and helped fixed many things, contributed documentation and made it alive again. The development hasn't stopped since then.

[*]Address: Mathematics Department US Naval Academy, Annapolis, MD 21402, USA, wdj@usna.edu
[†]University of Nevada, Reno
[‡]New Mexico Tech
[§]California Polytechnic State University, San Luis Obispo, CA

Many students improved SymPy incredibly as part of the Google Summer of Code (GSoC):

- GSoC2007: Mateusz Paprocki, Brian Jorgensen, Jason Gedge, Robert Schwarz and Chris Wu,
- GSoC2008: Fredrik Johansson,
- GSoC2009: Priit Laes, Freddie Witherden, A.M., Fabian Pedregosa, Dale Peterson,
- GSoC2010: Addison Cugini, Christian Muise, A.M., Matthew Curry, Øyvind Jensen,
- GSoC2011: Tom Bachmann, Gilbert Gede, Tomo Lazovich, Saptarshi Mandal, Sherjil Ozair, Vladimir Perić, Matthew Rocklin, Sean Vig, Jeremias Yehdegho.

SymPy is a team project and it was developed by a lot of people, not just GSoC students. For example in the early stages, Pearu Peterson joined the development during the summer 2007 and he has made SymPy much more competitive by rewriting the core from scratch, that has made it from 10x to 100x faster. The git repository at [1] contains all patches since July 19, 2007 and as of this writing (October 2011) it contains patches from around 132 different people (according to the `AUTHORS` file).

On Jan 4, 2011 O.Č. passed the project leadership to A.M..

1.2 License

SymPy uses the modified BSD license. O.Č. originally used GPL for SymPy, but early on in 2007 switched to BSD after discussion with other developers. This choice has worked very well for SymPy, as it doesn't enforce almost any restrictions on how people can use SymPy (for example SymPy is included in a commercial distribution or a paid Android application). SymPy developers believe in allowing people to use SymPy in any way possible as well as in creating a solid and active community around SymPy.

1.3 Active developers

SymPy is a project with an active community of over 100 developers contributing to it. However, among the most active (on the mailinglist, patches, reviews, management) as of this writing (October 2011) are A.M., O.Č., B.G., Matuesz Paprocki, Ronan Lamy, and Chris Smith. A good (but not perfect) indicator is the number of patches contributed by people in the last year, for example the top 10 are:

git-log

```
$ git shortlog -ns --since="1 year ago" | head -n 10
   590  Chris Smith
   579  Mateusz Paprocki
   413  Aaron Meurer
   172  Ronan Lamy
   149  Saptarshi Mandal
    91  Gilbert Gede
    90  Vladimir Perić
    89  Brian Granger
    79  Matthew Rocklin
    74  Ondřej Čertík
```

1.4 Language

SymPy is written in pure Python, which gives it three main advantages over other computer algebra systems (besides the fact that it is free). First, it is very portable, since it is written in pure Python

with no dependencies, it can be run literally anywhere where there is a Python environment . There are even versions of SymPy for smartphones!

In addition, this means that SymPy is very easy to install. From a teacher's perspective, this is important. Indeed, if the user only wishes to use it as a calculator and not as a library, he need not install it at all: he can simply run the `isympy` script from within the downloaded `sympy` directory.

Second is that it is built on top of a very strong and easy to use language, Python. This is contrasted over other computer algebra system, which that have invented their own languages. Even without considering the advantages of one language over another, the fact is that anyone who knows Python can write a program that uses SymPy.

Third, since it's just a Python module, it can very easily be used as a library. This is contrasted against many other computer systems, which may work very well as calculators, but are not so easy to use as part of a larger script, or to extend with custom classes.

SymPy runs inside a normal Python interpreter, such as the one that comes with your system's Python or IPython. It is started by executing the Python script `isympy` found within SymPy's bin subdirectory. This starts a normal Python shell (IPython shell if you have the IPython package installed), that (pre)executes the following commands:

Python

```
>>> from __future__ import division
>>> from sympy import *
>>> x, y, z, t = symbols('x y z t')
>>> k, m, n = symbols('k m n', integer=True)
>>> f, g, h = symbols('f g h', cls=Function)
```

So starting `isympy` is equivalent to starting Python (or IPython) and executing the above commands by hand. For example, the following simple Python commands were run from the command line after starting `isympy`.

SymPy

```
>>> L = [i^2 for i in range(10) if i%2==0]
>>> L; len(L)
[2, 0, 6, 4, 10]
5
```

The following example was taken from Mateusz Paprocki's master's degree thesis [P].

Example 1 Suppose we are in possession of American coins: pennies, nickels, dimes and quarters. We would like to compose a certain quantity out of those coins, say 117, such that the number of coins used is minimal. Let's forget about the minimality criterion for a moment. In this scenario it is not a big problem to compose the requested value. We can simply take 117 pennies and we are done, as long as we have so many of them. Alternatively we can take 10 dimes, 3 nickels and 2 pennies, or 2 quarters, 3 dimes, 5 nickels and 12 pennies, etc. There are quite a few combinations that can be generated to get the desired value. But which of those combinations leads to the minimal number of necessary coins? To answer this question we will take advantage of Gröbner bases computed with respect to a total degree ordering of monomials.

Our problem is a minimisation problem, so we take advantage of total degree ordering. This is a correct choice because total degree ordering takes monomials with smaller sums of exponents first and we can observe that the smaller the sum of exponents in a solution to our coins problem will be, the less coins will be needed. So, the chosen ordering of monomials encodes the cost function of our problem.

How to get the minimal number of required coins? Suppose we take any admissible solution to the studied problem. This can be the trivial solution in which we take 117 pennies or any other such that the total value of coins is equal to 117. We encode the chosen solution as a binomial with numbers of particular coins as exponents of p, n, d and q, and we reduce this binomial with respect to the Gröbner basis G utilizing graded lexicographic ordering of monomials.

SymPy

```
>>> var('p, n, d, q')
(p, n, d, q)
>>> F = [p**5 - n, p**10 - d, p**25 - q]
>>> G = groebner(F, order='grlex')
>>> reduced(p**117, G, order='grlex')[1]
     2  4
d n p  q
```

The answer, which we were able to compute with SymPy, is 1 dime, 1 nickel, 2 pennies, and 4 quarters which altogether give the requested value of 117. This is also the minimal solution to our problem.

This example showed SymPy's ability to compute with Gröbner basis, but the problem can be solved without such advanced machinery using greedy approach that involves only basic arithmetics:

SymPy

```
>>> k = Symbol('k', integer=True)
>>> coin_values = zip(var("p, n, d, q"), [1, 5, 10, 25])
>>> total, coin_counts = 117, []
>>> for coin, value in reversed(coin_values):
...     count = floor(solve(k*value <= total, k).rhs)
...     coin_counts.append((coin, count))
...     total -= count*value
...
>>> Mul(*[ coin**count for coin, count in coin_counts ])
     2  4
d n p  q
```

2 Documentation and capabilities

2.1 Basics

A great deal of information about SymPy is available on its website.

- Internet:
  Website :
  `http://sympy.org/`
- Documentation:
  On-line reference manual:
  `http://docs.sympy.org/`
  `http://docs.sympy.org/0.7.1/tutorial.html`
  `https://github.com/sympy/sympy/wiki`
- *Talks, blogs, papers, and lecture notes*:
  `http://planet.sympy.org/`
  `https://github.com/sympy/sympy/wiki/SymPy-Papers`

- Interfaces:
  Command line
  There are very useful command-line history, tab-completion, and command-line help features available.
  Online version:
  `http://live.sympy.org/`
  The Sage notebook interface (available in any Sage installation, available from `http://www.sagemath.org/`, or on-line at: `http://www.sagenb.org/` (select `sympy` in the drop-down menu).
- Availability:
  Source code and binaries: `http://sympy.org/download.html`

  SymPy is easy to install on all operating systems that support Python (e.g., MS Windows, Mac OSs, All flavors of Linux, and so on).

  There is also a version for the iPhone which is available separately from the iTunes online store (look for "Python Math" or go to `http://sabonrai.com/wp/pythonmath/`).

  For the smartphone OS "Maemo", there is a Sympy app described at
  `http://www.robertocolistete.net/Integral/`.
- *Support*:
  There is an email list for SymPy support and development:
  `http://groups.google.com/group/sympy?pli=1`
  `sympy@googlegroups.com`
- *License*:
  Modified BSD.

2.2 Capabilities

Currently, SymPy core has extensive computational capabilities including:

- basic arithmetics $*, /, +, -, **$;
- basic simplification (like $a * b * b + 2 * b * a * b \mapsto 3 * a * b^2$;
- expansion (like $(a + b)^2 \mapsto a^2 + 2 * a * b + b^2$);
- functions ($\exp, \ln, \ldots$);
- complex numbers (like exp(I*x).expand(complex=True) $\mapsto \cos(x) + I * \sin(x)$);
- differentiation;
- taylor (laurent) series;
- substitution (like $x \mapsto \ln(x)$, or $\sin \mapsto \cos$);
- arbitrary precision integers, rationals, and floats;
- noncommutative symbols;
- pattern matching;
- assumptions (like x is positive $\mapsto \sqrt{x^2} = x$).

Then there are many SymPy modules, such as (for example) for these tasks:

- more functions ($\sin, \cos, \tan$, atan, asin, acos, factorial, zeta, legendre, ...);
- limits (like $\text{limit}(x * \log(x), x, 0) \to 0$);
- integration using extended Risch-Norman heuristic;
- polynomials (division, gcd, square free decomposition, groebner bases, factorization, ...);
- solvers (algebraic and transcendental equations, difference equations, differential equations, and systems of equations);
- simplification;
- logic;
- symbolic matrices (determinants, LU decomposition, ...);
- number theory;
- Pauli and Dirac algebra;
- quantum physics;
- geometry;
- plotting (2D and 3D);
- 2D unicode pretty printing.
- tensors;
- stastistics;
- combinatorics.

2.2.1 Quantum physics

Thanks mostly to work done by B.G., and students he has directed, SymPy has extensive capabilities in quantum physics. These capabilities are focused on the various symbolic manipulations that arise in quantum mechanics, rather than on more traditional numerical computations.

To enable these symbolic manipulations to be performed using SymPy, we have implemented the Dirac notation in its most abstract and general form. This includes all of the needed mathematical entities: Hilbert spaces, bras and kets, operators, inner/outer/tensor products, commutators, anticommutators and so on.

The object-oriented nature of SymPy/Python has been particularly valuable in this work. The various mathematical entities listed above are implemented as base classes and then specific physical systems are implemented through subclassing. Thus, for example, when a spin operator is defined, it immediately inherits all of the capabilities of general hermitian operators. This allows new physical systems to be implemented quickly and easily.

The following quantum mechanical system have been implemented in this manner: i) quantum angular momentum including spin coupling, ii) second quantization for fermions and bosons, iii) gates and qubits used in quantum computing and quantum information, iii) position and momentum eigenstates and a variety of systems that utilize them and iv) density operators and matrices for representing mixed states.

The classes we have created for quantum computing are now being used in a research context. Being able to handle gates and qubits symbolically is extremely important, as the alternative requires constructing and multiplying extremely large matrices. This is enabling B.G. and co-workers to develop new optimization approaches for quantum circuits and algorithms.

As an example of the quantum computing capabilities in SymPy, we show how to apply a Hadamard gate to a qubit symbolically:

SymPy
```
>>> from sympy.physics.quantum import qapply
>>> from sympy.physics.quantum.gate import H
>>> from sympy.physics.quantum.qubit import Qubit
>>> c = H(0)*Qubit('0')
>>> qapply(c)
sqrt(2)*|0>/2 + sqrt(2)*|1>/2
```

In this example, we illustrate the rewriting of two angular momentum states in different bases and show the integrated latex rendering of SymPy:

SymPy
```
>>> from sympy import S, latex
>>> from sympy.physics.quantum.spin import JxKet
>>> latex(JxKet(S(1)/2,S(1)/2).rewrite('Jz'))
\frac{1}{2} \sqrt{2} {\left|\frac{1}{2},- \frac{1}{2}\right\rangle } + \frac{1
}{2} \sqrt{2} {\left|\frac{1}{2},\frac{1}{2}\right\rangle }
>>> latex(JxKet(1,1).rewrite('Jz'))
\frac{1}{2} {\left|1,-1\right\rangle } + \frac{1}{2} \sqrt{2} {\left|1,0\right
\rangle } + \frac{1}{2} {\left|1,1\right\rangle }
```

This renders to:

$$\frac{1}{2}\sqrt{2}\left|\frac{1}{2},-\frac{1}{2}\right\rangle + \frac{1}{2}\sqrt{2}\left|\frac{1}{2},\frac{1}{2}\right\rangle$$

$$\frac{1}{2}|1,-1\rangle + \frac{1}{2}\sqrt{2}|1,0\rangle + \frac{1}{2}|1,1\rangle$$

For further details, see the papers by Cugini [Cug] and Curry [Cur], or blog posts related to quantum mechanics on Planet SymPy (`http://planet.sympy.org/`).

2.2.2 Packages

To plot a function in SymPy you need to install a separate plotting package, such as matplotlib or pyglet. To install them into your local Python install, follow the instructions on their website (that is, `http://www.pyglet.org/` or `http://matplotlib.sourceforge.net/`).

2.2.3 Examples

This section merely presents a few examples to illustrate some of SymPy's functionality.

To plot a function in 2d, such as $y = \sin(x)$, use the following SymPy commands.

SymPy
```
>>> x = Symbol("x")
>>> Plot(sin(x), [x, -3, 3])
[0]: sin(x), 'mode=cartesian'
```

A separate window will pop-up with your (color) plot in it. See Figure 1.

To plot a surface in 3d, such as $z = xy^3 - yx^3$, use the following SymPy commands.

SymPy
```
>>> var('x y z')
>>> Plot(x*y**3-y*x**3)
```

A separate window will pop-up with your (color) plot in it. See Figure 1.

SymPy has good functionality with linear algebra. The method `rref` returns the row-reduced echelon form of a matrix, as well as a list of the pivot columns.

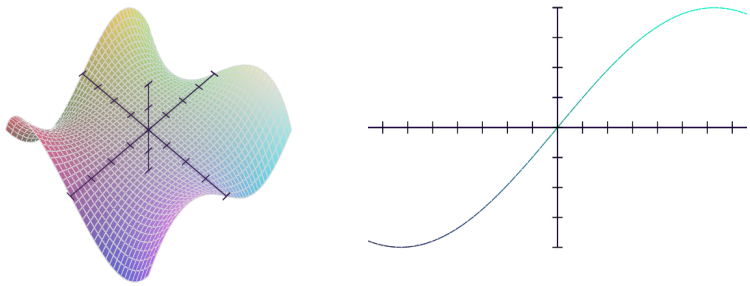

Figure 1: 2D and 3D plots using SymPy

SymPy

```
>>> M = Matrix(3,3,lambda i,j: i+j)
>>> print M
[0, 1, 2]
[1, 2, 3]
[2, 3, 4]
>>> print M.rref()
([1, 0, -1]
[0, 1,  2]
[0, 0,  0], [0, 1])
```

Indeed, there is a pivot in the first two columns but not the third, so SymPy returns only $[0, 1]$, as it should.

SymPy

```
>>> M = Matrix([[1,1,2],[1,2,1],[2,1,1]])
>>> print M
[1, 1, 2]
[1, 2, 1]
[2, 1, 1]
>>> print M.eigenvals()
{1: 1, -1: 1, 4: 1}
>>> print M.eigenvects()
[(1, 1, [[ 1]
[-2]
[ 1]]), (-1, 1, [[-1]
[ 0]
[ 1]]), (4, 1, [[1]
[1]
[1]])]
>>> print M.charpoly(x)
Poly(x**3 - 4*x**2 - x + 4, x, domain='ZZ')

>>> lam = M.eigenvals()
>>> lam.keys()
[1, -1, 4]
>>> lamb = lam.keys()
>>> expand((x-lamb[0])*(x-lamb[1])*(x-lamb[2]))
 3      2
x  - 4*x  - x + 4
```

In the last command, we merely verify that the characteristic polynomial is indeed a polynomial whose roots are the eigenvalues.

SymPy can solve higher order linear ordinary differential equations with constant coefficients with full generality. See Meuer [M] for details of the implementation. To solve the 3-rd order constant coefficient ordinary differential equation,

$$y''' - 3y'' + 3y' - y = 10e^x,$$

use the following SymPy commands.

SymPy

```
>>> x = Symbol("x")
>>> y = Function('y')
>>> de = y(x).diff(x,3)-3*y(x).diff(x,2)+3*y(x).diff(x)-y(x)-6*exp(x)
>>> soln = dsolve(de, y(x))
>>> print soln
y(x) == (C1 + C2*x + C3*x**2 + x**3)*exp(x)
```

Surprisingly, it seems Maxima cannot do this at the present time.

SymPy handles easily a wide range of calculus computations, as the following commands illustrate.

SymPy

```
>>> print legendre(8, x).diff(x)
6435*x**7/16 - 9009*x**5/16 + 3465*x**3/16 - 315*x/16
>>> print integrate(legendre(8, x),x)
715*x**9/128 - 429*x**7/32 + 693*x**5/64 - 105*x**3/32 + 35*x/128
>>> j0 = lambda x: besselj(0,x)
>>> limit(j0(x),x,0)
besselj(0, 0)
>>> N(limit(j0(x),x,0))
1.00000000000000
```

SymPy handles Taylor series expansions with no problem.

SymPy

```
>>> x = Symbol('x')
>>> f = 1/cos(x)
>>> print f.series(x, 0, 10)
1 + x**2/2 + 5*x**4/24 + 61*x**6/720 + 277*x**8/8064 + O(x**10)
```

Did you know SymPy can solve a linear recurrence sequence? For example, consider

$$u_{n+2} = 2u_{n+1} + 8u_n,$$

where we know $u_0 = 2$ and $u_1 = 7$.

SymPy

```
>>> n = Symbol('n', integer=True)
>>> u = Function('u')
>>> f=u(n+2)-2*u(n+1)-8*u(n)
>>> print rsolve(f, u(n), {u(0):2, u(1):7})
(-2)**n/6 + 11*4**n/6
```

SymPy contains wide range of special functions and it can easily be used to evaluate complicated expressions. For example here is how to evaluate the formula for the third order perturbative correction to the magnetic moment of an electron in quantum electrodynamics (this example also shows how to sum an infinite series), you can compare the formula and the value with the abstract of the paper [QED] and see that it agrees:

SymPy

```
>>> from sympy import pi, zeta, S, log, summation, var, oo
>>> var("n")
n
>>> a4 = summation(1/(2**n * n**4), (n, 1, oo))
>>> A_3 = 83*pi**2*zeta(3)/72 - 215*zeta(5)/24 + 100*(a4 + log(2)**4/24 - \
...         pi**2*log(2)**2/24)/3 - \
...         239*pi**4/2160 + 139*zeta(3)/18 - 298 * pi**2 * log(2)/9 + \
...         17101 * pi**2 / 810 + S(28259)/5184
>>> A_3.n()
1.18124145658720
```

In short, if you need to use a free mathematics program in teaching or research, and one which is easily installed on various platforms, SymPy is a great place to start. If you get stuck, there is plenty of information on-line, or you can join the SymPy googlegroups and email your question to `sympy@googlegroups.com`.

Acknowledgements. We thank Mateusz Paprocki and Vladimir Perić for helpful comments and corrections. (Any remaining errors are of course our responsibility). Some of the material above is taken with only slight modification from Paprocki's paper [P], the tutorial [PM], or the documentation at the official SymPy website [S].

References

[Cug] Addison Cugini, *Quantum Mechanics, Quantum Computation, and the Density Operator in SymPy*, California Polytechnic State University, Senior Thesis, 2011.
`http://digitalcommons.calpoly.edu/physsp/38`

[Cur] Matthew Curry, *Symbolic Quantum Circuit Simplifcation in SymPy*, California Polytechnic State University, Senior Thesis, 2011.
`http://digitalcommons.calpoly.edu/physsp/39`

[M] Aaron Meurer, *Variation of Parameters and More*, blog post available at
`http://asmeurersympy.wordpress.com/2009/08/01/variation-of-parameters-and-more/`.

[P] Mateusz Paprocki, *Design and Implementation Issues of a Computer Algebra System in an Interpreted, Dynamically Typed Programming Language*, Master's Thesis, 2010, University of Technology of Wroclaw, Poland.
`https://github.com/mattpap/masters-thesis`
`http://mattpap.github.com/masters-thesis/html/index.html`

[PM] — and Aaron Meurer, *Guide to symbolic mathematics with SymPy*, SciPy conference 2011 presentation,
`http://mattpap.github.com/scipy-2011-tutorial/html/mathematics.html`

[QED] S.Laporta, E.Remiddi *The analytical value of the electron (g-2) at order α^3 in QED*, Phys.Lett. B379 (1996) 283-291
`http://arxiv.org/abs/hep-ph/9602417`

[S] SymPy website. `http://www.sympy.org/`