Tutorial

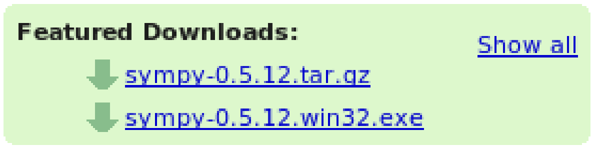
Introduction

SymPy is a Python library for symbolic mathematics. It aims to become a full-featured computer algebra system (CAS) while keeping the code as simple as possible in order to be comprehensible and easily extensible. SymPy is written entirely in Python and does not require any external libraries.

This tutorial gives an overview and introduction to SymPy. Read this to have an idea what SymPy can do for you (and how) and if you want to know more, read the *SymPy User's Guide*, *SymPy Modules Reference*. or the sources directly.

First Steps with SymPy ¶

The easiest way to download it is to go to http://code.google.com/p/sympy/ and download the latest tarball from the Featured Downloads:


Featured Downloads:
Show all
sympy-0.5.12.tar.gz
sympy-0.5.12.win32.exe

Unpack it:

```
$ tar xzf sympy-0.5.12.tar.gz
```

and try it from a Python intepreter:

```
$ cd sympy-0.5.12
$ python
Python 2.4.4 (#2, Jan  3 2008, 13:36:28)
[GCC 4.2.3 20071123 (prerelease) (Debian 4.2.2-4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from sympy import Symbol, cos
>>> x = Symbol("x")
>>> (1/cos(x)).series(x, 0, 10)
1 + (1/2)*x**2 + (5/24)*x**4 + (61/720)*x**6 + (277/8064)*x**8 + O(x**10)
```

You can use SymPy as shown above and this is indeed the recommended way if you use it in your program. You can also install it using `./setup.py install` as any other Python module, or just install a package in your favourite Linux distribution, e.g.:

Installing SymPy in Debian

```
$ sudo apt-get install python-sympy
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  python-sympy
0 upgraded, 1 newly installed, 0 to remove and 18 not upgraded.
Need to get 991kB of archives.
After this operation, 5976kB of additional disk space will be used.
Get:1 http://ftp.cz.debian.org unstable/main python-sympy 0.5.12-1 [991kB]
Fetched 991kB in 2s (361kB/s)
Selecting previously deselected package python-sympy.
(Reading database ... 232619 files and directories currently installed.)
Unpacking python-sympy (from .../python-sympy_0.5.12-1_all.deb) ...
Setting up python-sympy (0.5.12-1) ...
```

For other means how to install SymPy, consult the Downloads tab on the SymPy's webpage.

isympy Console

For experimenting with new features, or when figuring out how to do things, you can use our special wrapper around IPython called `isympy` (located in `bin/isympy` if you are running from the source directory) which is just a standard python shell that has already imported the relevant sympy modules and defined the symbols x, y, z and some other things:

```
$ cd sympy-0.5.12
$ bin/isympy
Python 2.4.4 console for SymPy 0.5.12-hg. These commands were executed:
>>> from __future__ import division
>>> from sympy import *
>>> x, y, z = symbols('xyz')
>>> k, m, n = symbols('kmn', integer=True)
>>> f = Function("f")

Documentation can be found at http://sympy.org/


In [1]: (1/cos(x)).series(x, 0, 10)
Out[1]:
     2       4        6          8
    x     5*x     61*x     277*x
1 + ── + ──── + ───── + ────── + O(x**10)
    2      24      720      8064
```

Note: Commands entered by you are bold. Thus what we did in 3 lines in a regular Python interpeter can be done in 1 line in isympy.

Using SymPy as a calculator

Sympy has three built-in numeric types: Real, Rational and Integer.

The Rational class represents a rational number as a pair of two Integers: the numerator and the denominator, so Rational(1,2) represents 1/2, Rational(5,2) 5/2 and so on.

```
>>> from sympy import *
>>> a = Rational(1,2)

>>> a
1/2

>>> a*2
1

>>> Rational(2)**50/Rational(10)**50
1/88817841970012523233890533447265625
```

proceed with caution while working with python int's since they truncate integer division, and that's why:

```
>>> 1/2
0

>>> 1.0/2
0.5
```

You can however do:

```
>>> from __future__ import division

>>> 1/2 #doctest: +SKIP
0.5
```

True division is going to be standard in python3k and `isympy` does that too.

We also have some special constants, like e and pi, that are treated as symbols (1+pi won't evaluate to something numeric, rather it will remain as 1+pi), and have arbitrary precission:

```
>>> pi**2
pi**2

>>> pi.evalf()
3.141592653589793238462643383

>>> (pi+exp(1)).evalf()
5.859874482048203234066343309
```

as you see, evalf evaluates the expression to a floating-point number

There is also a class representing mathematical infinity, called `oo`:

```
>>> oo > 99999
True
>>> oo + 1
oo
```

Symbols

In contrast to other Computer Algebra Systems, in SymPy you have to declare symbolic variables explicitly:

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

Then you can play with them:

```
>>> x+y+x-y
2*x

>>> (x+y)**2
(x+y)**2

>>> ((x+y)**2).expand()
2*x*y+x**2+y**2
```

And substitute them for other symbols or numbers using `subs(old, new)`:

```
>>> ((x+y)**2).subs(x, 1)
(1+y)**2

>>> ((x+y)**2).subs(x, y)
4*y**2
```

Algebra

For partial fraction decomposition, use `apart(expr, x)`:

```
In [1]: 1/( (x+2)*(x+1) )
Out[1]:
        1
───────────────
(2 + x)*(1 + x)

In [2]: apart(1/( (x+2)*(x+1) ), x)
Out[2]:
  1       1
───── - ─────
1 + x   2 + x

In [3]: (x+1)/(x-1)
```

```
Out[3]:
-(1 + x)
────────
 1 - x

In [4]: apart((x+1)/(x-1), x)
Out[4]:
      2
1 - ─────
    1 - x
```

To combine things back together, use `together(expr, x)`:

```
In [7]: together(1/x + 1/y + 1/z)
Out[7]:
x*y + x*z + y*z
───────────────
     x*y*z

In [8]: together(apart((x+1)/(x-1), x), x)
Out[8]:
-1 - x
──────
1 - x

In [9]: together(apart(1/( (x+2)*(x+1) ), x), x)
Out[9]:
       1
───────────────
(2 + x)*(1 + x)
```

Calculus

Limits

Limits are easy to use in sympy, they follow the syntax limit(function, variable, point), so to compute the limit of f(x) as x -> 0, you would issue limit(f, x, 0):

```
>>> from sympy import *
>>> x=Symbol("x")
>>> limit(sin(x)/x, x, 0)
1
```

you can also calculate the limit at infinity:

```
>>> limit(x, x, oo)
oo

>>> limit(1/x, x, oo)
0
```

```
>>> limit(x**x, x, 0)
1
```

for some non-trivial examples on limits, you can read the test file test_demidovich.py

Differentiation

You can differentiate any SymPy expression using `diff(func, var)`. Examples:

```
>>> from sympy import *
>>> x = Symbol('x')
>>> diff(sin(x), x)
cos(x)
>>> diff(sin(2*x), x)
2*cos(2*x)

>>> diff(tan(x), x)
cos(x)**(-2)
```

You can check, that it is correct by:

```
>>> limit((tan(x+y)-tan(x))/y, y, 0)
cos(x)**(-2)
```

Higher derivatives can be calculated using the `diff(func, var, n)` method:

```
>>> diff(sin(2*x), x, 1)
2*cos(2*x)

>>> diff(sin(2*x), x, 2)
-4*sin(2*x)

>>> diff(sin(2*x), x, 3)
-8*cos(2*x)
```

Series expansion

Use `.series(var, point, order)`:

```
>>> from sympy import *
>>> x = Symbol('x')
>>> cos(x).series(x, 0, 10)
1 - 1/2*x**2 + (1/24)*x**4 - 1/720*x**6 + (1/40320)*x**8 + O(x**10)
>>> (1/cos(x)).series(x, 0, 10)
1 + (1/2)*x**2 + (5/24)*x**4 + (61/720)*x**6 + (277/8064)*x**8 + O(x**10)
```

Another simple example:

```
from sympy import Integral, Symbol, pprint
```

```
x = Symbol("x")
y = Symbol("y")

e = 1/(x + y)
s = e.series(x, 0, 5)

print(s)
pprint(s)
```

That should print the following after the execution:

```
1/y + x**2*y**(-3) + x**4*y**(-5) - x*y**(-2) - x**3*y**(-4) + O(x**5)
         2     4           3
1    x     x     x     x
─ + ── + ── - ── - ── + O(x**5)
y     3     5     2     4
     y     y     y     y
```

Integration

SymPy has support for indefinite and definite integration of transcendental elementary and special functions via *integrate()* facility, which uses powerful extended Risch-Norman algorithm and some heuristics and pattern matching:

```
>>> from sympy import *
>>> x, y = symbols('xy')
```

You can integrate elementary functions:

```
>>> integrate(6*x**5, x)
x**6
>>> integrate(sin(x), x)
-cos(x)
>>> integrate(log(x), x)
x*log(x) - x
>>> integrate(2*x + sinh(x), x)
x**2 + cosh(x)
```

Also special functions are handled easily:

```
>>> integrate(exp(-x**2)*erf(x), x)
(1/4)*pi**(1/2)*erf(x)**2
```

It is possible to compute definite integral:

```
>>> integrate(x**3, (x, -1, 1))
0
>>> integrate(sin(x), (x, 0, pi/2))
1
>>> integrate(cos(x), (x, -pi/2, pi/2))
```

```
2
```

Also improper integrals are supported as well:

```
>>> integrate(exp(-x), (x, 0, oo))
1
>>> integrate(log(x), (x, 0, 1))
-1
```

Complex numbers

```
>>> from sympy import Symbol, exp, I
>>> x = Symbol("x")
>>> exp(I*x).expand()
exp(I*x)
>>> exp(I*x).expand(complex=True)
1/exp(im(x))*cos(re(x)) + I/exp(im(x))*sin(re(x))
>>> x = Symbol("x", real=True)
>>> exp(I*x).expand(complex=True)
I*sin(x) + cos(x)
```

Functions

trigonometric:

```
In [1]: sin(x+y).expand(trig=True)
Out[1]: cos(x)*sin(y) + cos(y)*sin(x)

In [2]: cos(x+y).expand(trig=True)
Out[2]: cos(x)*cos(y) - sin(x)*sin(y)

In [3]: sin(I*x)
Out[3]: □*sinh(x)

In [4]: sinh(I*x)
Out[4]: □*sin(x)

In [5]: asinh(I)
Out[5]:
π*□
───
 2

In [6]: asinh(I*x)
Out[6]: □*asin(x)

In [15]: sin(x).series(x, 0, 10)
Out[15]:
     3      5      7        9
    x      x      x        x
x - ── + ─── - ──── + ────── + O(x**10)
    6    120   5040   362880
```

```
In [16]: sinh(x).series(x, 0, 10)
Out[16]:
     3     5     7       9
    x     x     x       x
x + ── + ─── + ──── + ────── + O(x**10)
    6    120   5040   362880

In [17]: asin(x).series(x, 0, 10)
Out[17]:
     3      5      7       9
    x    3*x    5*x    35*x
x + ── + ──── + ──── + ───── + O(x**10)
    6     40    112     1152

In [18]: asinh(x).series(x, 0, 10)
Out[18]:
     3      5      7       9
    x    3*x    5*x    35*x
x - ── + ──── - ──── + ───── + O(x**10)
    6     40    112     1152
```

spherical harmonics:

```
In [1]: from sympy.abc import theta, phi

In [2]: Ylm(1, 0, theta, phi)
Out[2]:
  □□□
╲╱ 3 *cos(θ)
────────────
      □□□
  2*╲╱ π

In [3]: Ylm(1, 1, theta, phi)
Out[3]:
   □□□           □*φ
-╲╱ 6 *│sin(θ)│*ℯ
────────────────────
          □□□
      4*╲╱ π

In [4]: Ylm(2, 1, theta, phi)
Out[4]:
   □□□□                  □*φ
-╲╱ 30 *│sin(θ)│*cos(θ)*ℯ
────────────────────────────
              □□□
          4*╲╱ π
```

factorials and gamma function:

```
In [1]: x = Symbol("x")

In [2]: y = Symbol("y", integer=True)
In [3]: factorial(x)
Out[3]: Γ(1 + x)

In [4]: factorial(y)
Out[4]: y!

In [5]: factorial(x).series(x, 0, 3)
Out[5]:
                         2            2     2  2
                        x *EulerGamma      π *x
1 - x*EulerGamma + ─────────────── + ───── + O(x**3)
                              2            12
```

zeta function:

```
In [18]: zeta(4, x)
Out[18]: ζ(4, x)

In [19]: zeta(4, 1)
Out[19]:
 4
π
──
90

In [20]: zeta(4, 2)
Out[20]:
      4
     π
-1 + ──
     90

In [21]: zeta(4, 3)
Out[21]:
        4
  17   π
- ── + ──
  16   90
```

polynomials:

```
In [1]: chebyshevt(2, x)
Out[1]:
       2
-1 + 2*x

In [2]: chebyshevt(4, x)
Out[2]:
      2       4
1 - 8*x  + 8*x

In [3]: legendre(2, x)
Out[3]:
          2
       3*x
-1/2 + ───
        2

In [4]: legendre(8, x)
Out[4]:
          2          4          6          8
 35   315*x    3465*x    3003*x    6435*x
─── - ────── + ─────── - ─────── + ───────
128     32        64        32       128

In [5]: assoc_legendre(2, 1, x)
Out[5]:
        ________
       ╱      2
-3*x*╲╱  1 - x

In [6]: assoc_legendre(2, 2, x)
Out[6]:
       2
3 - 3*x

In [7]: hermite(3, x)
Out[7]:
           3
-12*x + 8*x
```

Differential Equations

In isympy:

```
In [4]: f(x).diff(x, x) + f(x)
Out[4]:
   2
  d
─────(f(x)) + f(x)
dx dx

In [5]: dsolve(f(x).diff(x, x) + f(x), f(x))
Out[5]: C₁*sin(x) + C₂*cos(x)
```

Algebraic equations

In isympy:

```
In [7]: solve(x**4 - 1, x)
Out[7]: [ⅈ, 1, -1, -ⅈ]

In [8]: solve([x + 5*y - 2, -3*x + 6*y - 15], [x, y])
Out[8]: {y: 1, x: -3}
```

Linear Algebra

Matrices

Matrices are created as instances from the Matrix class:

```
>>> from sympy import Matrix
>>> Matrix([[1,0], [0,1]])
[1 0]
[0 1]
```

you can also put Symbols in it:

```
>>> x = Symbol('x')
>>> y = Symbol('y')
>>> A = Matrix([[1,x], [y,1]])
>>> A #doctest: +NORMALIZE_WHITESPACE
[1 x]
[y 1]

>>> A**2 #doctest: +NORMALIZE_WHITESPACE
[1+x*y 2*x]
[2*y 1+x*y]

>>> 1
1
```

For more information an examples with Matrices, see the LinearAlgebraTutorial.

Pattern matching

Use the `.match()` method, along with the `Wild` class, to perform pattern matching on expressions. The method will return a dictionary with the required substitutions, as follows:

```
>>> from sympy import *
>>> x = Symbol('x')
>>> p = Wild('p')
>>> q = Wild('q')
>>> (5*x**2 + 3*x).match(p*x**2 + q*x)
{p_: 5, q_: 3}

>>> (x**2).match(p*x**q)
{p_: 1, q_: 2}
```

If the match is unsuccessful, it returns **`None`**:

```
>>> print (x+1).match(p**x)
None
```

One can also make use of the *WildFunction* class to perform more specific matches with functions and their arguments:

```
>>> f = WildFunction('f', nofargs=1)
>>> (5*cos(x)).match(p*f)
{p_: 5, f_: cos(x)}
>>> (cos(3*x)).match(f(p*x))
{p_: 3, f_: cos}
>>> g = WildFunction('g', nofargs=2)
>>> (5*cos(x)).match(p*g)
None
```

One can also use the exclude parameter of the `Wild` class to ensure that certain things do not show up in the result:

```
>>> x = Symbol('x')
>>> p = Wild('p', exclude=[1,x])
>>> print (x+1).match(x+p) # 1 is excluded
None
>>> print (x+1).match(p+1) # x is excluded
None
>>> print (x+1).match(x+2+p) # -1 is not excluded
{p_: -1}
```

Printing

There are many ways how expressions can be printed.

Standard

This is what `str(expression)` returns and it looks like this:

```
>>> from sympy import Integral
>>> from sympy.abc import x
>>> print x**2
x**2
>>> print 1/x
1/x
>>> print Integral(x**2, x)
Integral(x**2, x)
>>>
```

Pretty printing

This is a nice ascii-art printing produced by a `pprint` function:

```
>>> from sympy import Integral, pprint
>>> from sympy.abc import x
>>> pprint(x**2)
 2
```

```
x
>>> pprint(1/x)
1
─
x
>>> pprint(Integral(x**2, x))
⌠
⎮  2
⎮ x  dx
⌡
>>>
```

See also the wiki Pretty Printing for more examples of a nice unicode printing.

Tip: To make the pretty printing default in the python interpreter, use:

```
$ python
Python 2.5.2 (r252:60911, Jun 25 2008, 17:58:32)
[GCC 4.3.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from sympy import *
>>> import sys
>>> sys.displayhook = pprint
>>> var("x")
x
>>> x**3/3
 3
x
──
3
>>> Integral(x**2, x)
⌠
⎮  2
⎮ x  dx
⌡
```

Python printing

```
>>> from sympy.printing.python import print_python
>>> from sympy import Integral
>>> from sympy.abc import x
>>> print_python(x**2)
x = Symbol('x')
e = x**2
>>> print_python(1/x)
x = Symbol('x')
e = 1/x
>>> print_python(Integral(x**2, x))
x = Symbol('x')
e = Integral(x**2, x)
>>>
```

LaTeX printing

```
>>> from sympy import Integral, latex
>>> from sympy.abc import x
>>> latex(x**2)
'${x}^{2}$'
>>> latex(1/x)
'${x}^{-1}$'
>>> latex(Integral(x**2, x))
'$\\int {x}^{2}\\,dx$'
>>>
```

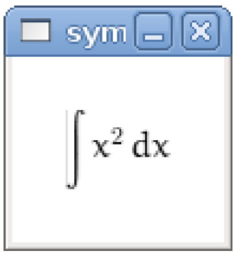
MathML

```
>>> from sympy.printing.mathml import print_mathml
>>> from sympy import Integral, latex
>>> from sympy.abc import x
>>> print_mathml(x**2)
<apply>
    <power/>
    <ci>
        x
    </ci>
    <cn>
        2
    </cn>
</apply>

>>> print_mathml(1/x)
<apply>
    <power/>
    <ci>
        x
    </ci>
    <cn>
        -1
    </cn>
</apply>

>>>
```

Pyglet

```
>>> from sympy import Integral, pngview
>>> from sympy.abc import x
>>> pngview(Integral(x**2, x))
```

And a pyglet window with the LaTeX rendered expression will popup:



$\int x^2\, dx$

Notes

`isympy` calls `pprint` automatically, so that's why you see pretty printing by default.

Note that there is also a printing module available, `sympy.printing`. Other printing methods available trough this module are:

- `pretty(expr)`, `pretty_print(expr)`, `pprint(expr)`: Return or print, respectively, a pretty representation of `expr`. This is the same as the second level of representation described above.
- `latex(expr)`, `print_latex(expr)`: Return or print, respectively, a LaTeX representation of `expr`
- `mathml(expr)`, `print_mathml(expr)`: Return or print, respectively, a MathML representation of `expr`.
- `print_gtk(expr)`: Print `expr` to Gtkmathview, a GTK widget that displays MathML code. The Gtkmathview program is required.

Further documentation

Now it's time to learn more about SymPy. Go through the *SymPy User's Guide* and *SymPy Modules Reference*.

Be sure to also browse our public wiki.sympy.org, that contains a lot of useful examples, tutorials, cookbooks that we and our users contributed and we encourage you to edit it.