SUMMER SCHOOL 2021

Demonstration of Sympy - Symbolic Mathematics with Python

Andrew Gill

Summary of commands used in this lecture

Command	Meaning
symbols	Define symbolic variables for sympy
subs	Substitute a value or expression into another sympy expression
evalf	Evaluate, yielding an approximate floating-point value
expand	Multiply out a sympy expression
factor	Factorize a sympy expression
simplify	Attempt to intelligently simplify an expression by any means possible
diff	Differentiate an expression symbolically
integrate	Integrate an expression symbolically
limit	calculate the limit as some variable tends to a value
Derivative	Returns a symbolic derivative object
Integral	Returns a symbolic integral object
Limit	Returns an symbolic limit object
init_printing	Formats sympy output to look more like maths
latex	Generates LaTeX code for arbitrary expressons
solve	solves the roots of arbitrary expression in one variable
prime	Returns the nth prime number

Introduction

The purpose of sympy is to give Python some of the same functionality as computer algebra systems such as Maple, Mathematica or Scientific Notebook (which are commercially licensed) or Maxima, (which is open-source).

Sympy doesn't claim to provide all the functionality of such specialised software, but it is, nonetheless, powerful and useful. Add to this the fact that it can be combined with all the other functionality and modules of python (numpy, scipy, matplotlib and so on) and you have a very versatile tool indeed. And its open-source and free (which is always good).

Symbolic manipulation

Ordinarly in programming terms, a variable is a name we attach to a space in memory that stores a bit of data - perhaps an integer, a floating-point number or a string. In Sympy, however, there is a type of variable available to use that behaves much more like a variable in algebra - it is a symbol which we can manipulate in its own right, which need not have a definite numerical value associated with it.

To create such a variable we use the symbols function:

```
>>> import sympy
>>> x,y=sympy.symbols("x y")
>>> x**2+2*x*y+y**2
x**2 + 2*x*y + y**2
>>> expre = x**2+2*x*y+y**2
>>> expre+x
x**2 + 2*x*y + x + y**2
>>> expre+x*y
x**2 + 3*x*y + y**2
>>>
```

Note that we are able to store an algebraic expression in a variable (which I've named "expre" here; this name has no special significance to sympy).

Also note that the parameter fed to the symbols function is a string of the names that you'd like to see sympy display for each symbolic variable. Usually, if you are using the letters of the alphabet, you'll want to keep them the same as the variable name, though if you really wanted to, you could do something like the following (probably out of sheer perversity):

```
>>> p,q=sympy.symbols("r t")
>>> p+q
r + t
```

Also note what happens when you write the square root of 2:

```
>>> sympy.sqrt(2)
sqrt(2)
```

It doesn't return a floating-point approximation with 14 decimal places after the point, but rather keeps it in "symbolic" form.

Substitutions and Evaluating Expressions

If you want to substitute a value into an algebraic expression, you can use the subs function. Of course, you can also substitute one algebraic expression into another:

```
>>> a=sympy.symbols("a")
>>> x,y=sympy.symbols("x y")
>>> func1=x**2-y**2
>>> func1.subs([[x,3],[y,4]])
25
>>> func2=sympy.sqrt(1-a**2)
>>> func1.subs(x,func2)
-a**2 - y**2 + 1
>>>
```

And if you do want to evaluate the square root of 2 approximately as a floating-point number, you can use evalf:

```
>>> sympy.sqrt(2)
sqrt(2)
>>> sympy.sqrt(2).evalf()
1.41421356237310
```

Factorization, Expansion and Simplifying expressions

So far so good. Now suppose we have an algebraic expression that is written in a factored form that we'd like to have expanded. The expand function is useful here:

```
>>> formula1=2*x*((x**2+3)**2+(3*y+5))/(x**2+3)
>>> sympy.expand(formula1)
2*x**5/(x**2 + 3) + 12*x**3/(x**2 + 3) + 6*x*y/(x**2 + 3) + 28*x/
(x**2 + 3)
```

Alternatively, if we have an expression that we'd like to have factorized for us because Grade 10 maths class was a *loooong* time ago:

```
>>> formula2=x**3-y**3
>>> sympy.factor(formula2)
(x - y)*(x**2 + x*y + y**2)
```

There is another, even more powerful function included in sympy called simplify. It will attempt to use whatever factorizations, trig identities, etc. it can to simplify your expression as much as possible:

```
>>> sympy.simplify((x+3)**2+sympy.sin(x)**2+sympy.cos(x)**2)
x**2 + 6*x + 10
```

Integration, Differentiation and limits

Sympy can do a lot more than factorize or expand expressions. It can also handle symbolic differentiation and many symbolic integrations (not all, but this is an area of ongoing development, so each version does a few more).

The diff function handles differentiation. It takes two parameters, the first being the expression to differentiate, while the second is the symbol/variable in terms of which to differentiate. If only a single variable is present in the expression, the second parameter may be omitted:

```
>>> sympy.diff(x**2-2*x+1 +sympy.sin(x))
2*x + cos(x) - 2
>>> sympy.diff(x**2-2*x+1 +sympy.sin(x),x)
2*x + cos(x) - 2
>>> sympy.diff(x**2*y-2*x+1 +sympy.sin(x),x)
2*x*y + cos(x) - 2
>>> sympy.diff(x**2*y-2*x+1 +sympy.sin(x*y),x)
2*x*y + y*cos(x*y) - 2
```

If, instead of symbolically differentiating your expression, you want to use it as a symbol, you can do this with the Derivative function:

```
>>> func1=x*y**2+x**2*y+3*x-4*y
>>> sympy.Derivative(func1,x)
Derivative(x**2*y + x*y**2 + 3*x - 4*y, x)
```

If you want to integrate an expression, you use the integrate function:

```
>>> func2=3*x**2+2*x+1
>>> sympy.integrate(func2,x)
x**3 + x**2 + x
```

As with Derivative, there is an Integral function which can be used when you want an integral to treat as a symbolic unit rather than evaluate it:

```
>>> func2=3*x**2+2*x+1
>>> sympy.Integral(func2,x)
Integral(3*x**2 + 2*x + 1, x)
```

To calculate limits, the limit function is used. The Limit function (note the capital) yields a symbolic representation that can be used for further manipulation.

```
>>> sympy.limit(1/x**2, x, sympy.oo)
0
>>> sympy.Limit(1/x**2, x, sympy.oo)
Limit(x**(-2), x, oo, dir='-')
```

Note that sympy.oo is sympy's representation of infinity.

Miscellaneous Cool Tricks in sympy

If you'd like to see your mathematical formulae more prettily formatted, you should call the init_printing function at the beginning of your script or interactive session.

```
>>> sympy.init_printing()
>>> sympy.Integral(func2,x)
⌠
⎮ ⎛   2          ⎞
⎮ ⎝3⋅x  + 2⋅x + 1⎠ dx
⌡
```

Those of you who are using latex to write your papers or thesis (I hope there are a few of you) can also use it to generate latex code for equations:

```
>>> expr1=3*x**2-2*x+1+sympy.sin(x)
>>> integ1=sympy.Integral(expr1,x)
>>> integ1
Integral(3*x**2 - 2*x + sin(x) + 1, x)
>>> sympy.latex(integ1)
'\\int \\left(3 x^{2} - 2 x + \\sin{\\left (x \\right )} + 1\\
right)\\, dx'
```

Sympy can find the roots of equations for you, using the solve function:

```
>>> sympy.solve(3*x**2-2*x-10)
[1/3 + sqrt(31)/3, -sqrt(31)/3 + 1/3]
>>> sympy.solve(3*x**2-2*x-10)[0].evalf()
2.18925478761001
>>> sympy.solve(3*x**2-2*x-10)[1].evalf()
-1.52258812094334
```

And, for those of you who didn't enjoy coding the prime number exercise, you can find the 327^{th} prime number with just one line using sympy:

```
>>> sympy.prime(327)
2179
```

Going Further

As with all of these notes, this is really only a brief introduction to the library. It can do partial differerential equations, limits and many other things besides. You really should look at the official documentation online, which includes a much more detailed tutorial:

https://docs.sympy.org/latest/tutorial/index.html