1_basics

September 18, 2018

1 Sympy basics

This series of notebooks is aimed at showing some of the capabilities of Python for symbolic computation, through the module SymPy, as an alternative to propietary software like Mathematica. The first four notebooks deal with symbolic computation, while the remaining two introduce the numeric modules NumPy and SciPy.

1.0.1 Python and sympy objects

First we need to understand the logic under Python modules. Usually when we want to use a mathematical function in Python to perform non-trivial operations we must first import it from its module,

```
In [1]: from math import sqrt
```

Now we can use the function `sqrt` from the module `math` in our code

```
In [2]: sqrt(3)
```

```
Out[2]: 1.7320508075688772
```

Alternatively, we may import the whole module and call all the functions contained in it with the following syntax

```
In [3]: import math

        math.sqrt(3)
```

```
Out[3]: 1.7320508075688772
```

Different modules may contain functions with the same name, but the functions themselves are *different*. For instance, in the case of the function `sqrt`, it is defined in the module `sympy` as well

```
In [4]: import sympy

        sympy.sqrt(3)
```

```
Out[4]: sqrt(3)
```

Now, sqrt(3) is interpreted as the symbolic object $\sqrt{3}$ not as a floating-point number 1.73205... We can check how Python works with this objects internally

```
In [5]: sq = math.sqrt(3)
        type(sq)

Out[5]: float

In [6]: sq = sympy.sqrt(3)
        type(sq)

Out[6]: sympy.core.power.Pow
```

It is important to bear this difference in mind while working in Python to avoid conflicts between different functions with the same name. We will come back to this point with `numpy`.

1.0.2 Defining symbols and functions

In all our sessions with `sympy` we will call first (WARNING!!: this is widely regarded as a bad practice in Python, in our case we will only do it with the sympy module)

```
In [7]: from sympy import *
```

That is, we will import every function in `sympy`, just to avoid the long-hand notation `sympy.function()`. We will also call the function `init_printing()` in `sympy`, that allows for a fancier output

```
In [8]: init_printing()
```

The square root should be printed now in a much prettier way

```
In [9]: sqrt(3)

Out[9]:
```

$$\sqrt{3}$$

Ok, after setting up our enviroment we will start to do some symbolic computation

```
In [10]: sqrt(5)*sqrt(8)

Out[10]:
```

$$2\sqrt{10}$$

Admittedly not very interesting. To go beyond this elementary example we need to learn how to define variables and functions. Every non-numeric character must be defined before-hand in `sympy` (otherwise Python thinks it is a standard variable).

Symbols

```
In [11]: x, y, z, t, nu, a = symbols('x y z t nu alpha')
```

In the left-hand we have the name that we will use in our code, while on the right-hand side we have defined the name that will be associated to the variable in the output. For instance, our variable 'a' will be displayed as α

```
In [12]: a**2
```

```
Out[12]:
```

$$\alpha^2$$

In this example we can see that the greek letters are displayed correctly, but we must be careful with λ. In fact, `lambda` has a special meaning in Python so to represent λ in `sympy` we must use `lamda` instead

```
In [13]: l = symbols('lamda')
         l
```

```
Out[13]:
```

$$\lambda$$

Functions We can construct explicit functions with a direct assignment

```
In [14]: expr = cos(x)**2 + 1
```

```
In [15]: expr + x**2
```

```
Out[15]:
```

$$x^2 + \cos^2(x) + 1$$

or with the usual definition in Python

```
In [16]: def my_func(x, a=1, b=0, c=0):
             """
             Usual Python function
             """
             return a*x**2 + b*x + c
```

```
In [17]: my_func(x, b=nu)
```

```
Out[17]:
```

$$\nu x + x^2$$

It will be useful too to declare some symbols as implicit functions (to define differential equations for instance)

```
In [18]: f, g, h = symbols('f g h', cls=Function)
```

1.0.3 Equalities

There are at least three types of 'equalities',

- Assignment (=): associates a value to a variable

```
In [19]: expr = 3*x**2
         expr
```

```
Out[19]:
```

$$3x^2$$

```
In [20]: C = 3.34*23
         C
```

```
Out[20]:
```

$$76.82$$

- Equality (==): boolean operator, tests the equality of two variables. In `sympy`, it tests structural identity (i.e. two expressions mathematically identical but written in a different way -> evaluates to False)

```
In [21]: C2 = 76.82

         C == C2
```

```
Out[21]: True
```

```
In [22]: expr = cos(x)**2 + sin(x)**2

         expr == 1
```

```
Out[22]: False
```

- Symbolic equality (Eq): belongs to sympy. We will to use this form to define equations.

```
In [23]: Eq(2*x**2 + 3, -5)
```

```
Out[23]:
```

$$2x^2 + 3 = -5$$

1.0.4 Getting help

- Jupyter notebook supports autocompletion, if you don't remember a command or the name of a method try pressing Tab.

- If you forget the use of some function, the order of its parameters or something like that, the best course of action is to look it up in the built-in documentation with the commands:

```
In [24]: init_printing?
```

there you can consult its use and even some examples. If you want to dig in a bit more, you can see the source code with

```
In [25]: init_printing??
```

it may be not very illuminating though. Another function that works like '?' is

```
In [26]: #help(init_printing)
```

this last expression works on a standard Python shell as well, the other two belong to Jupyter. Sometimes it is useful to find out the methods in some object, dir() does the job

```
In [27]: #dir(sympy.Symbol)
```

- It is always advisable to check the official documentation in the dedicated webpage for each module.
- Before giving up all hope, check Google. It will probably redirect you to StackOverflow, a most reliable source (dwelling of expert developers and sages alike).

1.0.5 Documentation

All the material presented in this set of notebooks is but a succint version of the official tutorials and documentation: - Sympy User Guide. - NumPy User Guide. - SciPy User Guide.

Useful complementary material - Matplotlib Introduction: Jupyter notebook showing the basics of Matplotlib. Check references therein. - Python Introduction: Jupyter notebook introducing Python for scientists. Check references therein.

Finally don't forget that the Jupyter has a Help tab and a tutorial, if you have issues with the notebook format. There is **plenty** of material out there on Python, if you want to perform a specific task don't hesitate to Google it, you may find the answer and learn something else along the way.