3

July 23, 2021

1 Primeiros Passos com o Sympy

1.1 Instalação

Você possivelmente deve estar se perguntando como instalar o `Sympy`. Se você já utilizou algum outro módulo em Python, possivelmente imaginou em instalá-lo utilizando o `pip` (software que pedimos que garantisse sua sua instalação no capítulo 0).

Contudo, se você utiliza está utilizando notebooks com o Anaconda (nossa recomendação para esse módulo), o `Sympy` já está instalado, basta carregá-lo.

Caso esteja desenvolvendo em outro ambiente, uma forma de instalar é com o pip, por exemplo:

```
pip install sympy
```

1.2 Carregando o Módulo

Para utilizar os comandos do `Sympy` de forma nativa em nossos scripts, precisamos importá-lo globalmente. Para isso utilizamos as palavras-chave `import` e `from`. Isso não foi abordado no capítulo anterior devido a sua complexidade, mas essa é uma forma de importar módulos em Python.

Portanto, basta criar e executar a seguinte *chunk*:

[1]:
```
from sympy import *
init_printing(use_unicode=True, use_latex='mathjax') # Para imprimir LaTeX
```

O `*` significa que estamos importando o módulo por completo.

1.3 Trabalhando com expressões matemáticas

Como o `Sympy` tem como objetivo o cálculo simbólico, tudo é baseado a partir dos simbólos. Ou seja, as nossas querídas variáveis (como x, y, e z) sendo interpretadas com suas propriedades matemáticas.

Portanto, para utilizá-las, precisamos criar seus símbolos. Por enquanto, vamos utilizar somente o x. Então, para o `x` do Python significar a variável x fazemos:

[2]:
```
x = symbols('x')
x
```

[2]:

x

Note que nossa saída matemática será processada por um compilador LaTeX para facilitar a leitura.

No caso, você pode utilizar x como um número, e as expressões apareceráo normalmente (sem igualdade).

[3]:
```
x**2 - 4*x + 3
```

[3]:

$x^2 - 4x + 3$

Caso você queira a solução de uma expressão que seja igual a 0 (ou suas raízes, em outras palavras), em respeito a uma variável, você pode usar a função solve(). Ela recebe dois parâmetros obrigatórios, sua expressão e a variável que você quer a solução.

[4]:
```
solve(x**2 - 4*x + 3, x)
```

[4]:

$[1, \ 3]$

[5]:
```
solve(sqrt(x) - (x/2),x)
```

[5]:

$[0, \ 4]$

Inclusive, caso queira uma resposta como costumamos escrever no papel, ou seja, em forma de conjunto e suas condições, podemos utilizar o solveset(). A maior diferença é que ele pode receber o conjunto numérico onde você quer trabalhar através do parâmetro domain. Na maioria das vezes podemos utilizar domain=S.Reals ou domain=S.Complexes.

[6]:
```
solveset(x**2 - 4*x  +20,x, domain=S.Reals)
```

[6]:

$\emptyset$

[7]:
```
solveset(x**2 - 4*x  +20,x, domain=S.Complexes)
```

[7]:

$\{2 - 4i, 2 + 4i\}$

[8]:
```
solveset(tan(x), x, domain=S.Reals)
```

[8]:

$\{2n\pi \mid n \in \mathbb{Z}\} \cup \{2n\pi + \pi \mid n \in \mathbb{Z}\}$

Vamos criar uma variável para armazenar essa primeira expressão para mostrar outros exemplos

[9]:
```
expr = x**2 - 4*x + 3
```

Podemos achar valores utilizando o método subs(). Novamente, devemos especficiar a variável.

[10]:
```
expr.subs(x,2) # Se y = expr, esse é o valor de y quando x = 2.
```

[10]:

-1

[11]:
```
expr.subs(x,1)
```

[11]: 0

Se tivermos uma expressão numérica não-inteira e quisermos achar a solução em um ponto flutuante (float), podemos usar o método evalf().

[12]:
```
my_sqrt = sqrt(8)
my_sqrt
```

[12]: $2\sqrt{2}$

[13]:
```
my_sqrt.evalf()
```

[13]: 2.82842712474619

Como viu acima, possivelmente há uma função do SymPy que represente uma operação ou função matemática. Por exemplo, temos sqrt(), log(), exp(), sin() e etc. Quando sentir necessidade de utilizar uma dessa, tente antes de consultar a documentação. Caso não consiga ``adivinhar'', faça uma consulta que, com toda certeza, haverá uma função que te atenderá.

Existem algumas funções que ``fazem Álgebra'' por si só. Veja alguns exemplos:

[14]:
```
# Simplifica
simplify((x**2 + x)/x)
```

[14]: $x + 1$

[15]:
```
# Fatora
factor(1-1/x)
```

[15]: $\frac{x-1}{x}$

[16]:
```
# Expande
expand((x**2 + 3*x)**3)
```

[16]: $x^6 + 9x^5 + 27x^4 + 27x^3$

[17]:
```
# Agrupa potências de uma variável (que vai como segundo parâmetro)
collect(x**2 + 4*x - 2*x**2 + x -20 + x**3 + 2, x)
```

[17]: $x^3 - x^2 + 5x - 18$

[18]:
```
# Separa fração em frações parciais
apart((x**2 + 8*x-18)/(x**3 + 3*x**2))
```

[18]: $-\frac{11}{3(x+3)} + \frac{14}{3x} - \frac{6}{x^2}$

Além desses principais, ainda há trigsimp() e expand_trig() que simplificam e expandem funções trigonométricas (a partir das identidades de adição de arco). E outras que fazem o mesmo para potências, logaritmos e outros tipos de funções. Nesse caso, acho que vale a pena dar uma olhada na documentação. Elas todas são bem parecidas.

Finalizando esse tópico inicial, temos como substituir uma função em termos de outra. Por exemplo:

[19]:
```
sin(x).rewrite(cos)
```

[19]: $\cos\left(x - \frac{\pi}{2}\right)$

[20]:
```
(x**3).rewrite(exp)
```

[20]: $e^{3\log(x)}$

1.3.1 Equações

Ok, nós vimos como utilizar expressões. Mas, como tratamos equações? Como vimos no capítulo anterior = significa atribuição e == é uma operação booleana (ou seja, recebemos True ou False).

Para equações criamos uma classe Eq(). Não se preocupe com a nomenclatura, é só uma forma de criar um objeto do tipo Eq. Para criar esse objeto, passamos dois argumentos: cada lado da equação, respectivamente. Veja:

[21]:
```
eq = Eq(x**2, 2)
eq
```

[21]: $x^2 = 2$

Podemos utilizar o mesmo método para encontrar suas raízes.

[22]:
```
solve(eq,x)
```

[22]: $\left[-\sqrt{2}, \ \sqrt{2}\right]$

1.3.2 Igualdade

Como verificar igualdade entre duas expressões? O == só servirá para expressões identicas (não somente em valor, mas também nos termos expressos). Para isso, utilizamos o método equals(). Veja:

[23]:
```
expr_1 = sin(x)**2
```

[24]:
```
expr_2 = .5*(1 -cos(2*x))
```

Como veremos abaixo, pelo == as expressões seriam diferentes.

[25]:
```
expr_1 == expr_2
```

[25]: False

Vejamos pelo método equals():

[26]:
```
expr_1.equals(expr_2)
```

[26]: True

Podemos visualizar a igualdade da seguinte forma:

[27]:
```
Eq(expr_1,expr_2)
```

[27]:

$$\sin^2(x) = 0.5 - 0.5\cos(2x)$$

1.3.3 Sistemas de Equações

Podemos, também utilizando solve() encontrar as soluções de um sistema de equações. Basta passar uma lista com as equações como parâmetro.

[28]:
```
y, z = symbols('y z') # Criando o y simbólico
Eqs = []
Eqs.append(Eq(3*x - 3*y, 20))
Eqs.append(Eq(-7*x + 9*y, -10))
solve(Eqs, x, y)
```

[28]:

$$\left\{x : 25, \ y : \frac{55}{3}\right\}$$

Caso esteja tentando resolver um sistema linear (como o acima), é possível utilizar o linsolve().

[29]:
```
linsolve(Eqs, x, y)
```

[29]:

$$\left\{\left(25, \ \frac{55}{3}\right)\right\}$$

[30]:
```
Eqs = []
Eqs.append(Eq(3*x - 3*y + 2*z, 20))
Eqs.append(Eq(-7*x + 9*y -4*z, -10))
Eqs.append(Eq(-7*x + 9*y + 5*z, 40))
linsolve(Eqs, x, y,z)
```

[30]:

$$\left\{\left(\frac{175}{9}, \ \frac{445}{27}, \ \frac{50}{9}\right)\right\}$$

[31]:
```
Eqs = []
Eqs.append(Eq(x**2 + y**2, 18)) #  Elipse de centro (0,0) e R^2 = 18
Eqs.append(Eq(x, y)) # Reta identidade
nonlinsolve(Eqs, x, y) # Sistema não-linear (solve também serve). No nosso␣
↪caso, 2 pontos de intersecção.
```

[31]:

$$\{(-3, \ -3), (3, \ 3)\}$$

1.4 Matrizes

É bem trivial trabalhar com matrizes no Sympy. De modo geral, basta criar um objeto a partir da classe Matrix. E passamos uma lista de listas, sendo cada uma das listas uma linha. Veja:

[32]:
```python
Matrix([[1,2,3],[2,3,1]])
```

[32]:
$$\begin{bmatrix}1 & 2 & 3\\2 & 3 & 1\end{bmatrix}$$

E podemos manipulá-las normalmente, com as operações comuns. Além disso, há algumas outras operações especiais.

[33]:
```python
A = Matrix([[1,2,3],[2,3,1]])
B = Matrix([[3,2], [2,2], [1,4]])
A, B
```

[33]:
$$\left(\begin{bmatrix}1 & 2 & 3\\2 & 3 & 1\end{bmatrix}, \begin{bmatrix}3 & 2\\2 & 2\\1 & 4\end{bmatrix}\right)$$

[34]:
```python
A * B # Multiplicação
```

[34]:
$$\begin{bmatrix}10 & 18\\13 & 14\end{bmatrix}$$

[35]:
```python
B.T # Transpor
```

[35]:
$$\begin{bmatrix}3 & 2 & 1\\2 & 2 & 4\end{bmatrix}$$

[36]:
```python
A + B.T # Soma
```

[36]:
$$\begin{bmatrix}4 & 4 & 4\\4 & 5 & 5\end{bmatrix}$$

[37]:
```python
A.row(1) # Começa em 0
```

[37]:
$$\begin{bmatrix}2 & 3 & 1\end{bmatrix}$$

[38]:
```python
B.col(0) # Também começa em 0
```

[38]:
$$\begin{bmatrix}3\\2\\1\end{bmatrix}$$

[39]:
```python
B = B.col_insert(2,Matrix([2,3,2])) # Insere Coluna
B
```

[39]:
$$\begin{bmatrix}3 & 2 & 2\\2 & 2 & 3\\1 & 4 & 2\end{bmatrix}$$

[40]:
```python
B**-1
```

[40]:
$$\begin{bmatrix}\frac{4}{7} & -\frac{2}{7} & -\frac{1}{7}\\\frac{1}{14} & -\frac{2}{7} & \frac{5}{14}\\-\frac{3}{7} & \frac{5}{7} & -\frac{1}{7}\end{bmatrix}$$

1.5 Exercícios

Utilizando o que aprendeu nesse capítulo, tente resolver os seguintes exercícios:

1. Encontre as raízes de cada uma das expressões abaixo. Depois encontre um par (x, y) para cada:

$$x^3 - 8x^2 + 4x + 3$$

$$\sin(x) + 2\cos(x)$$

$$\log\left|\frac{x^2 - x}{2}\right|$$

$$e^{-x^3+5x^2-x} - 1$$

2. Encontre as soluções das equações abaixo:

$$x^4 - 4x^3 + x^2 - 30 = -x^2 + x - 40$$

$$2^{x^2-x} = 3^x$$

$$\log\left|x^3 - 2x^2 + x\right| = \log\left|x^2 + 6x\right|$$

3. Verifique se as igualdades são verdadeiras:

$$\Gamma\left(\frac{3}{2}\right) = \frac{\sqrt{\pi}}{2}$$

$$\sin(2x^2) - \cos(x^2 + x) = \sin(x)\sin\left(x^2\right) + 2\sin\left(x^2\right)\cos\left(x^2\right) - \cos(x)\cos\left(x^2\right)$$

$$(x^2 - 3x)(2x^4 + x^3 - x)(-4x^2) = 8x^8 + 20x^7 + 12x^6 + 4x^5 - 12x^4$$

4. Encontre as soluções do sistema:

$$\begin{cases} 4x - 3y + 2z = 60 \\ (x - 10)^2 + y^2 + z^2 = 72 \\ 2x + 9y + z = 20 \end{cases}$$