Programação I / Introdução à Programação

Módulo Sympy

João Pedro Pedroso

2024/2025

Última aula:

- módulos úteis da biblioteca standard

Hoje:

- módulo sympy

Sympy

- Instalação: na shell:

  `$ pip3 install sympy`

- Para utilizar o sympy:

```
$ python
>>> from sympy import *
>>> x = Symbol('x')
>>> limit(sin(x)/x, x, 0)
1
>>> integrate(1/x, x)
log(x)
```

Computação simbólica

- Trata de objetos matemáticos como símbolos
  - em vez de números

```
>>> import math
>>> math.sqrt(8)
2.82842712475
```

  - podem ser representados de forma exata

```
>>> import sympy
>>> sympy.sqrt(8)
2*sqrt(2)
```

  - variáveis *não avaliadas (numericamente)* são tratadas com símbolos

Diferença entre cálculo simbólico e cálculo numérico

- Exemplo:

```
1  >>> import math
2  >>> math.pi
3  3.141592653589793
4  >>> math.sin(math.pi)
5  1.2246467991473532e-16
6  >>> import sympy
7  >>> sympy.pi
8  pi
9  >>> sympy.sin(sympy.pi)
10 0
```

Expressões com símbolos

- Expressão $x + 2y$

```
>>> from sympy import symbols
>>> x, y = symbols('x y')
>>> expr = x + 2*y
>>> expr
x + 2*y
>>> expr + 1
x + 2*y + 1
>>> expr - x
2*y
>>> x*expr
x*(x + 2*y)
```

Computação simbólica

Exemplos:

- simplificação de expressões
- calcular derivadas, integrais, limites
- resolver equações
- trabalhar com matrizes
- *pretty printing*
  - equações
  - gráficos
- geração de código
- ...

```
>>> from sympy import *
>>> x, t, z, nu = symbols('x t z nu')
>>> init_printing(use_unicode=True)    # pretty print with unicode characters
>>> diff(sin(x)*exp(x), x)    # take the derivative of sin(x) exp(x)
 x           x
e  sin(x) + e  cos(x)
>>> integrate(exp(x)*sin(x) + exp(x)*cos(x), x)
 x
e  sin(x)
```

Operações elementares

- Símbolos → tal como variáveis em Python têm de estar definidos:

```
>>> from sympy import *
>>> x + 1
Traceback (most recent call last):
...
NameError: name 'x' is not defined
>>> x = symbols('x')
>>> x + 1
x + 1
```

- Podemos definir vários símbolos de forma conveniente:

```
>>> x, y, z = symbols('x y z')
```

- Note a diferença entre *símbolo* e a *variável* que se refere a ele

```
>>> a, b = symbols('b a')
>>> a
b
>>> b
a
```

Igualdade e comparação

- Sintaxe semelhante à do Python na atribuição e comparação

```
>>> x + 1 == 4
False
```

- Igualdade "simbólica"

```
>>> Eq(x + 1, 4)
Eq(x + 1, 4)
```

- Comparação não considera simplificação de expressões:

```
>>> (x + 1)**2 == x**2 + 2*x + 1
False
```

- Para verificarmos igualdade de expressões:

```
>>> a = (x + 1)**2
>>> b = x**2 + 2*x + 1
>>> simplify(a - b)
0
```

- Outra forma:

```
>>> a = cos(x)**2 - sin(x)**2
>>> b = cos(2*x)
>>> a.equals(b)
True
```

Substituição

- Alteração do nome de variáveis; avaliação:

```
>>> from sympy import *
>>> x, y, z = symbols("x y z")
>>> expr = cos(x) + 1
>>> expr.subs(x, y)
cos(y) + 1
>>> expr.subs(x, 0)    # evaluate expression a x=0
2
```

- Construção de expressões mais complexas

```
>>> expr = x**y
>>> expr
x**y
>>> expr = expr.subs(y, x**y)
>>> expr
x**(x**y)
>>> expr = expr.subs(y, x**x)
>>> expr
x**(x**(x**x))
```

- Nota: objetos Sympy são imutáveis

Conversão de *strings* para expressões

- `sympify`

```
>>> str_expr = "x**2 + 3*x - 1/2"
>>> expr = sympify(str_expr)
>>> expr
x**2 + 3*x - 1/2
>>> expr.subs(x, 2)
19/2
```

- Avaliação numérica: `evalf`

```
>>> expr = sqrt(8)
>>> expr.evalf()
2.82842712474619
```

- Sympy trabalha com precisão arbitrária:

```
>>> pi.evalf(100)
3.14159265358979323846264338327950288419716939937510582097494459230781640628620
```

lambdify

- Podemos converter expressões Sympy para expressões Python, que podem ser avaliada numericamente:

```
>>> expr = sin(x)
>>> f = lambdify(x, expr, "math")    # using math.sin
>>> f(0.1)
0.0998334166468
>>> import numpy
>>> a = numpy.arange(10)
>>> f = lambdify(x, expr, "numpy")    # using numpy.sin
>>> f(a)
[ 0.          0.84147098  0.90929743  0.14112001 -0.7568025  -0.95892427
 -0.2794155   0.6569866   0.98935825  0.41211849]
```

Impressão

- Expressões podem ser convertidas para diversos formatos de impressão
- Em ciência, um dos mais utilizados é Latex:

```
>>> print(latex(Integral(sqrt(1/x), x)))
\int \sqrt{\frac{1}{x}}\, dx
```

$$\int \sqrt{\frac{1}{x}}\, dx$$

Simplificação

- Forma conveniente:

```
>>> simplify(sin(x)**2 + cos(x)**2)
1
>>> simplify((x**3 + x**2 - x - 1)/(x**2 + 2*x + 1))
x - 1
>>> simplify(factorial(x)/factorial(x-2))
x (x - 1)
>>>
```

- Há formas mais específicas, eg:

```
>>> expand((x + 1)**2)
x**2 + 2*x + 1
>>> expand((cos(x) + sin(x))**2)
   2                          2
sin (x) + 2 sin(x) cos(x) + cos (x)
>>> print(latex(expand((cos(x) + sin(x))**2)))
\sin^{2}{\left(x \right)} + 2 \sin{\left(x \right)} \cos{\left(x \right)} + \c
```

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

```
>>> factor(x**3 - x**2 + x - 1)
(x - 1)*(x**2 + 1)
```

Tipos de simplificação

- polinomial/racional
  `expand` / `factor` / `collect` / `cancel` / `...`
- trigonométrica
  `trigsimp` / `expand_trig` / `...`
- potências
  `powsimp` / `expand_power_exp` / `expand_power_base` / `...`
- exponenciais e logaritmos
  `expand_log` / `logcombine`
- funções especiais

Cálculo

- Derivadas:

```
>>> from sympy import *
>>> x, y, z = symbols('x y z')
>>> diff(x**4, x)
4*x**3
>>> diff(x**4, x, 3)
24 x
```

- Integrais:

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

- Limites:

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

- Expansão de séries
- Diferenças finitas
- ...

Resolução de equações

- Conjunto de soluções

```
>>> solveset(Eq(x**2, 1), x)
{-1, 1}
>>> solveset(Eq(x**2 - 1, 0), x)
{-1, 1}
>>> solveset(x**2 - 1, x)
{-1, 1}
```

- Resolução algébrica
- Equações diferenciais
- ...

Matrizes

- Manipulação simbólica
- Complemento a NumPy

```
>>> M = Matrix([[1, 3], [-2, 3]])
>>> N = Matrix([[0, 3], [0, 7]])
>>> print(latex(M + N))
\left[\begin{matrix}1 & 6\\-2 & 10\end{matrix}\right]
```

$$\begin{bmatrix} 1 & 6 \\ -2 & 10 \end{bmatrix}$$

Aula de hoje:

- *SymPy Tutorial*
  `https://docs.sympy.org/latest/tutorial/index.html`

Próxima aula

- Não há próxima aula!
- Exame: em que dia? a que horas? onde?