Aula14_Lab2

May 14, 2020

1 Lab 2: Computação Simbólica em Python

Neste notebook veremos como utilizar recursos de computação algébrica, com expressões analíticas, dentro do ambiente Jupyter notebook. Para isso, usaremos a biblioteca SymPy. A intenção aqui é fornecer uma introdução básica ao tema, concentrando-se em como usar esses recursos no contexto do nosso curso. Para mais informações, você pode consultar a excelente documentação do **SymPy** que está disponível online. Um bom ponto de partida é o Tutorial do SymPy.

Em particular, **não deixe de ler** Gotchas!, para evitar problemas comuns de sintaxe!!

1.1 Computação Simbólica

A computação simbólica envolve a computação de objetos matemáticos simbolicamente. Isso significa que os objetos matemáticos são representados exatamente, não aproximadamente, e expressões matemáticas com variáveis não avaliadas são deixadas em forma simbólica, ou algébrica.

Para entender a diferença veja os exemplos abaixo

[1]:
```
import math
print('A raiz de 9 é:', math.sqrt(9))
print('A raiz de 8 é:', math.sqrt(8))
```

```
A raiz de 9 é: 3.0
A raiz de 8 é: 2.8284271247461903
```

[2]:
```
import sympy
print('A raiz de 9 é:', sympy.sqrt(9))
print('A raiz de 8 é:', sympy.sqrt(8))
```

```
A raiz de 9 é: 3
A raiz de 8 é: 2*sqrt(2)
```

Na aula passada, alguém perguntou porque o seno de π não era exatamente zero num dos resultados. Expliquei que isso era devido a aproximação finita que sempre existe ao representar um número no computador. As ferramentas de computação algébrica (CAS - *Computer Algebra System*), como o SymPy, usam recursos especiais para representar um número sem aproximações na memória do computador. Isso vem com um custo computacional, mas é bastante útil em várias situações, como veremos. Veja abaixo como SymPy avalia o sen(π)

[3]:
```
sympy.sin(sympy.pi)  # resultado do seno de pi, com o SymPy
```

[3]:
$$0$$

[4]:
```
math.sin(math.pi)    # resultado do seno de pi, com a biblioteca padrão do Python
```

[4]:
```
1.2246467991473532e-16
```

Além de eliminar os erros de arredondamento, a computação algébrica permite manipular algebricamente expressões matemática e até mesmo aplicar ferramentas de Cálculo, como derivadas e integrais. Veremos exemplos mais adiante.

1.1.1 Como usar? - variáveis simbólicas

De maneira bem resumida, além de carregar a biblioteca SymPy e aprender sua sintaxe, o primeiro passo principal é definir uma variável como simbólica, como é mostrado nos exemplos abaixo:

[5]:
```
import sympy
x, y, z, t = sympy.symbols('x y z t')
```

Agora os identificadores (variáveis) "x, y, z, t" são variáveis simbólicas e serão tratadas como objetos sympy, e não mais como identificadores usuais do Python. Podemos, então definir funções com essas variáveis e usá-las em expressões algébricas. O exemplo abaixo, calcula a derivada de e^{-2x^2}:

[6]:
```
sympy.diff(sympy.exp(-2*x**2))
```

[6]:
$$-4xe^{-2x^2}$$

Para simplificar a sintaxe, e evitar ficar digitando sympy. o tempo todo, podemos carregar as funções diretamente (mas cuidado com isso!!), ou usar uma atalho (alias), com já mostramos na última aula. Aqui, para simplificar, eu vou carregar todas as funções que usaremos neste tutorial, na linha abaixo.

[7]:
```
from sympy import sin, cos, exp, diff, integrate, series, solve, pi, Matrix,␣
↪simplify,oo
diff(exp(-2*x**2))
```

[7]:
$$-4xe^{-2x^2}$$

Podemos usar solve para resolver equações, como, por exemplo $x^2 - 8 = 0$

[8]:
```
solve(x**2 - 8,x)
```

[8]:
```
[-2*sqrt(2), 2*sqrt(2)]
```

para tornar a apresentação das resposta mais agradável, podemos usar o comando

[9]:
```
sympy.init_printing(use_latex=True)
```

[10]:
```
solve(x**2 - 8,x)
```

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

Ou simplificar expressões algébricas

[11]:
```
sympy.pprint((x**3 + x**2 - x - 1)/(x**2 + 2*x + 1))
simplify((x**3 + x**2 - x - 1)/(x**2 + 2*x + 1))
```

[11]: $x - 1$

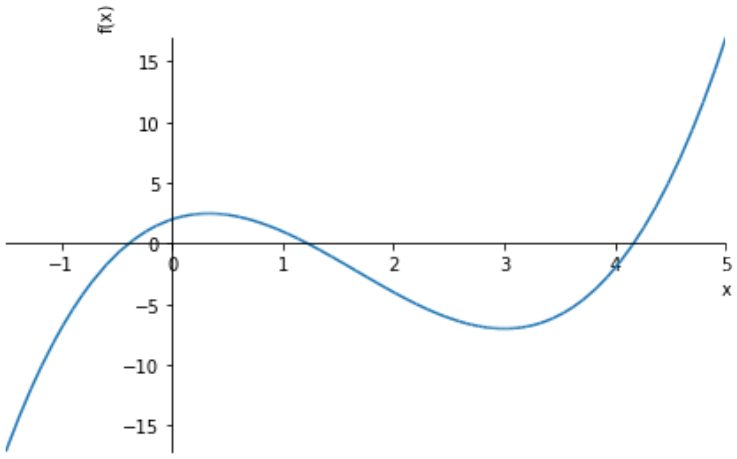
1.2 Funções e gráficos

A forma mais simples de definir uma função no SymPy é simplesmente definir uma nova variável atribuindo uma expressão com variáveis simbólicas

[12]:
```
expr = x**3 -5*x**2 + 3*x + 2
expr
```

[12]: $x^3 - 5x^2 + 3x + 2$

[13]:
```
sympy.plot(expr,(x,-1.5,5));  # é fácil plotar o gráfico de uma função
```



Para mais informações veja a documentação online.

1.3 Matrizes

É possível definir matrizes com o comando Matrix()

[14]:
```
M = Matrix([[1,2],[1,0]])  # define M com a nossa velha conhecida matriz...
M
```

[14]: $\begin{bmatrix} 1 & 2 \\ 1 & 0 \end{bmatrix}$

[15]:
```
M.det()   # calcula o determinante da matriz M
```

[15]: -2

[16]:
```
M.inv()   # matriz inversa
```

[16]: $\begin{bmatrix} 0 & 1 \\ \frac{1}{2} & -\frac{1}{2} \end{bmatrix}$

[17]:
```
M.eigenvals()   # autovalores
```

[17]: $\{-1 : 1,\ 2 : 1\}$

[18]:
```
M.eigenvects()  # autovetores
```

[18]: $\left[\left(-1,\ 1,\ \left[\begin{bmatrix} -1 \\ 1 \end{bmatrix}\right]\right),\ \left(2,\ 1,\ \left[\begin{bmatrix} 2 \\ 1 \end{bmatrix}\right]\right)\right]$

[19]:
```
sympy.eye(4)  # matriz identidade de ordem 4
```

[19]: $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Podemos também definir uma matriz de variáveis e funções simbólicas!

[20]:
```
G = Matrix([[x**2,y+x],[2*x**3*y, 5*y**3]])
G
```

[20]: $\begin{bmatrix} x^2 & x+y \\ 2x^3y & 5y^3 \end{bmatrix}$

Para mais informações e recursos, veja a documentação online. Vale a pena ver a parte de diagonalização.

1.4 Cálculo

Vejamos alguns exemplos de como usar sympy em Cálculo...

Derivadas

[21]:
```
diff(sin(x)*exp(x), x)
```

[21]: $e^x \sin(x) + e^x \cos(x)$

[22]:
```
diff(exp(2*x),x,3)  # calcula a terceira derivada de exp(2*x)
```

[22]: $8e^{2x}$

Podemos aplicar derivadas em objetos mais complexos, como matrizes, por exemplo.

[23]:
```
diff(G,x)  # a matriz G foi definida acima
```

[23]:
$$\begin{bmatrix} 2x & 1 \\ 6x^2y & 0 \end{bmatrix}$$

Para calcular derivadas parciais, como $\frac{\partial^2}{\partial x \partial y} e^{(x^2+y^3)}$

[24]:
```
diff(exp(x**2+y**3),x,y)  # derivadas parciais com relação a x e y
```

[24]:
$$6xy^2 e^{x^2+y^3}$$

Integração - integrais indefinidas: integrate(expr, variável) - integrais definidas: integrate(expr, (var_int, lim_inf, lim_sup))

[25]:
```
integrate(cos(x),x)
```

[25]:
$$\sin(x)$$

[26]:
```
# integral de uma gaussiana no intervalo {0,oo}
integrate(exp(-x**2),(x,0,oo))  # O símbolo "oo" representa infinito
```

[26]:
$$\frac{\sqrt{\pi}}{2}$$

Há ainda outra forma de definir de a integral, que pode ser conveniente

[27]:
```
integral = sympy.Integral(sin(x**2),x)
integral
```

[27]:
$$\int \sin\left(x^2\right) dx$$

A variável *integral* representa a integração indicada acima. Para realizar o cálculo, pode-se usar .doit()

[28]:
```
integral.doit()
```

[28]:
$$\frac{3\sqrt{2}\sqrt{\pi} S\left(\frac{\sqrt{2}x}{\sqrt{\pi}}\right)\Gamma\left(\frac{3}{4}\right)}{8\Gamma\left(\frac{7}{4}\right)}$$

O método .doit() pode ser usado em outros contextos também.

1.5 Séries de potências

Podemos facilmente expandir em série de Taylor, usando o comando series()

[29]:
```
series(exp(x))
```

[29]:
$$1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120} + O\left(x^6\right)$$

Se definirmos uma expressão (ou função), podemos fazer a sua expansão usando o método .series(). Neste caso, a sintaxe geral tem a forma: f(x).series(x, x0, n), onde x0 indica o ponto de expansão e n indica a ordem da expansão.

[30]:
```
expr = exp(-2*x**2)
expr.series(x,0,9)
```

[30]:
$$1 - 2x^2 + 2x^4 - \frac{4x^6}{3} + \frac{2x^8}{3} + O\left(x^9\right)$$

1.6 Séries de Fourier

Também é possível expandir em Série de Fourier. Para isso, vamos carregar a função

[31]:
```
from sympy import fourier_series
s = fourier_series(x**2)
s
```

[31]:
$$-4\cos\left(x\right) + \cos\left(2x\right) + \frac{\pi^2}{3} + \ldots$$

[32]:
```
#sympy.init_printing(pretty_print=True,use_unicode=True)
s1 = fourier_series(x**2)
s2 = fourier_series(x**3)
sympy.pprint(s1)
sympy.pprint(s2)
```

É possível definir melhor o ponto de expansão, o intervalo e a ordem da expansão

[33]:
```
s = fourier_series(x**2, (x,-pi,pi))
s.truncate(5)
```

[33]:
$$-4\cos\left(x\right) + \cos\left(2x\right) - \frac{4\cos\left(3x\right)}{9} + \frac{\cos\left(4x\right)}{4} + \frac{\pi^2}{3}$$

1.7 Transformada de Fourier

[34]:
```
from sympy import fourier_transform, exp
from sympy.abc import x, k
fourier_transform(exp(-3*x**2), x, k)
```

[34]:
$$\frac{\sqrt{3}\sqrt{\pi}e^{-\frac{\pi^2 k^2}{3}}}{3}$$

Para mais detalhes e informações, consulte a documentação online.

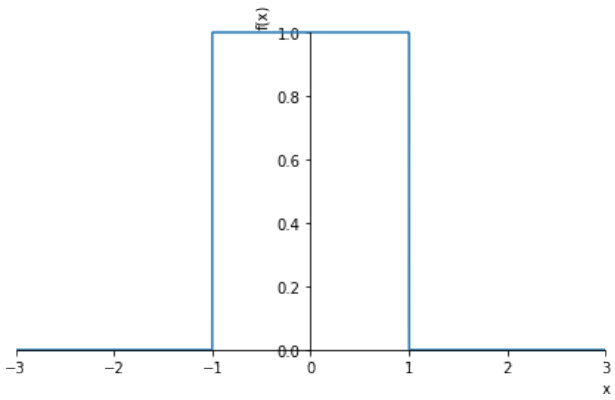
1.8 Exemplos e exercícios

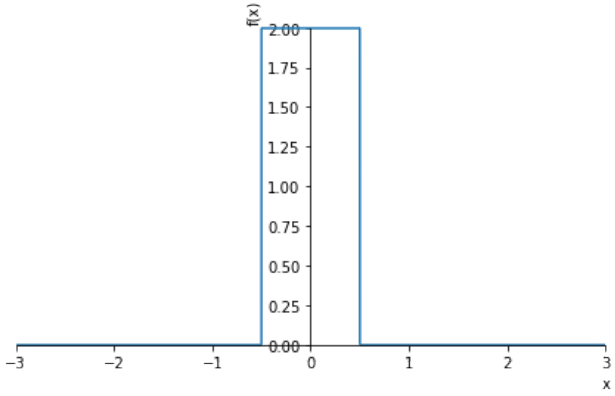
1.8.1 Exemplo 1

Vamos resolver aqui o Exemplo 2.6 do livro do Griffiths, para ilustrar alguns recursos interessantes.

Para mais aprender mais sobre Funções Especiais do sympy, veja a documentação online.

[35]:
```
from sympy import Heaviside, DiracDelta, plot, symbols, oo, Eq, sqrt, pi, exp,␣
↪sin
psi, A, a = symbols('psi, A, a')
h = A*(Heaviside(x+a)-Heaviside(x-a))
plot(h.subs(((a,1),(A,1))) ,(x,-3,3));
plot(h.subs(((a,0.5),(A,2))) ,(x,-3,3));
```

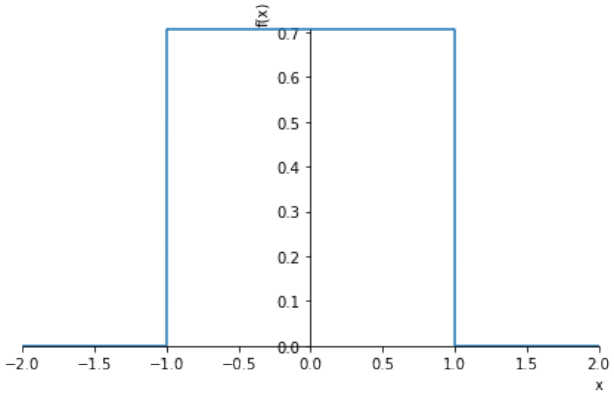




[36]:
```
psi = h/sqrt(2*a)
psi
#psi1 = psi.subs(((A,1),(a,1)))
#psi1
```

[36]:
$$\frac{\sqrt{2}A\left(-\theta\left(-a+x\right)+\theta\left(a+x\right)\right)}{2\sqrt{a}}$$

[37]:
```
psi1 = psi.subs(((A,1),(a,1)))
plot(psi1,(x,-2,2));
```



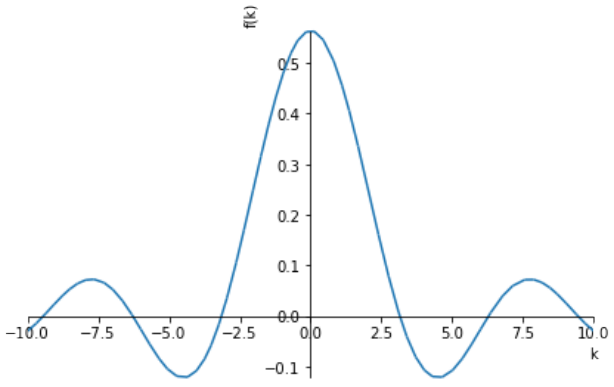
[38]:
```
phi = (1/sqrt(2*pi))*(1/sqrt(2*a))*integrate(exp(-1j*k*x),(x,-a,a))
phi
```

[38]:
$$\frac{\begin{cases} -\frac{1.0ie^{1.0iak}}{k} + \frac{1.0ie^{-1.0iak}}{k} & \text{for } k > -\infty \wedge k < \infty \wedge k \neq 0 \\ 2a & \text{otherwise} \end{cases}}{2\sqrt{\pi}\sqrt{a}}$$

[39]:
```
phi.simplify()
```

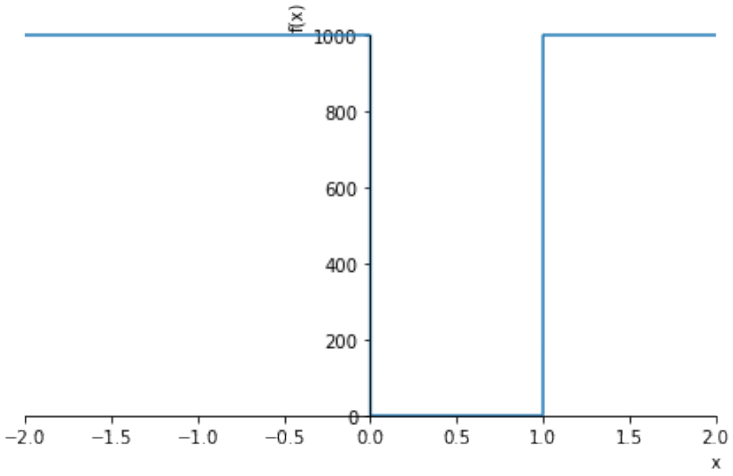
[39]:
$$\begin{cases} \frac{1.0\sin(1.0ak)}{\sqrt{\pi}\sqrt{a}k} & \text{for } k > -\infty \wedge k < \infty \wedge k \neq 0 \\ \frac{\sqrt{a}}{\sqrt{\pi}} & \text{otherwise} \end{cases}$$

[40]:
```
phi1 = phi.subs(a,1)
plot(phi1,(k,-10,10));
```



1.8.2 Exemplo 2

[41]:
```
from sympy import Heaviside, Function, symbols, plot, simplify, exp, sin, pi
from sympy import integrate, dsolve, Eq, Derivative as D
x, k, a, A, n = symbols('x, k, a, A, n', real=True)
psi = Function('psi')(x)
V = 1000*(1-(Heaviside(x)-Heaviside(x-1)))
plot(V,(x,-2,2));
```



[42]:
```
eq = Eq(D(psi,x,x) + k**2*psi,0)
eq
```

[42]:
$$k^2\psi(x) + \frac{d^2}{dx^2}\psi(x) = 0$$

[43]:
```
dsolve(eq,psi)
```

[43]:
$$\psi(x) = C_1 \sin(x\,|k|) + C_2 \cos(kx)$$

Ao analisar a solução acima, nos limites em que $x = 0$ e $x = a$, onde $\psi \to 0$, temos que

$$\psi(0) = C_1 \sin(0) + C_2 \cos(0) = C_2 = 0 A = C_1$$

e

$$\psi(a) = A \sin(ak) = 0 \sin(ak) = 0 \to \quad k = \frac{n\pi}{a}, \quad n = 1, 2, 3, \ldots$$

Como pode-se verificar

[44]:
```
psi = A*sin(n*pi*x/a)
norma = integrate(psi*psi, (x,0,a)).simplify()
norma
```

[44]:
$$\begin{cases} \frac{A^2 a\left(\pi n - \frac{\sin(2\pi n)}{2}\right)}{2\pi n} & \text{for } \frac{\pi n}{a} \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

[45]:
```
solve(Eq( norma.subs(n,1) ,1),A  )
```

[45]:
$$\left[-\sqrt{2}\sqrt{\frac{1}{a}},\ \sqrt{2}\sqrt{\frac{1}{a}}\right]$$

Portanto, a condição de normalização resulta em $A = \sqrt{\frac{2}{a}}$, e os estados estacionários são

$$\psi_n(x) = \sqrt{\frac{2}{a}}\,\text{sen}\left(\frac{n\pi}{a}x\right)$$

Para visualizar estes estados, podemos fazer $a = 1$ e plotar para $n = 1, 2, 3$

[46]:
```
psi.subs(((A,1),(a,1)))
```

[46]:
$$\sin(\pi n x)$$

[47]:
```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0,1,51)

for k in range(1,5):
    plt.plot( x, np.sin(k*np.pi*x) )
```