SciPy - Constantes Físicas

SciPy é uma poderosa biblioteca de módulos do Python para computação científica, onde estão implementados diversos algoritmos usados em problemas de física e engenharia. Para instalar:
`conda install scipy`
O pacote `scipy.constants` contém várias constantes físicas tabeladas.

Example

```
In[x]:  import scipy.constants as pc
In[x]:  pc.physical_constants['electron mass']
Out[x]: (9.10938356e-31, 'kg', 1.1e-38)
In[x]:  pc.value('electron mass')
Out[x]: 9.10938356e-31
In[x]:  pc.physical_constants['Planck constant']
Out[x]: (6.62607004e-34, 'J s', 8.1e-42)
```

SciPy - Funções Especiais

O pacote `scipy.special` contém a implementação numérica de várias funções que aparecem constantemente em ciência.
Veja uma lista completa em
`docs.scipy.org/doc/scipy/reference/special.html`

Função Gama

A função $\Gamma(x)$, para x real, é definida como:

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$$

Exemplo:

```
In[x]: from scipy.special import gamma
In[x]:  x = [0.5,1,2.5,4]
        gamma(x)
Out[x]: array([1.77245385, 1., 1.32934039, 6.])
```

SciPy - Funções Especiais

Várias funções especiais disponíveis

- Funções de Airy
- Funções de Integrais Elípticas
- Funções de Bessel
- Zeros das Funções de Bessel
- Funções de Bessel esféricas
- Polinômios de Hermite

SciPy - Funções Especiais - Exemplo

Exemplo A intensidade da luz de um padrão de difração circular é dada por

$$I(r) = \left(\frac{J_1(kr)}{kr} \right)^2,$$

onde r é o raio de um círculo, medido a partir do centro, $k = 2\pi/\lambda$, e $J_1(x)$ é a função de Bessel de primeira espécie de ordem 1.

Vamos escrever um programa para visualizar o padrão de difração para um fonte de comprimento de onda de $\lambda = 0.5$ μm, observada num quadrado de lado 2 μm, utilizando a função especial `j1(x)`. Para melhor visualização, vamos usar `imshow(I,cmap='gray',vmax=0.01)`.

SciPy - Raízes de Funções

O pacote `scipy.optimize` implementa vários métodos para calcular raízes de funções. Os argumentos passados devem ser uma função contínua, $f(x)$, e um intervalo $[a, b]$ dentro do qual a raiz será encontrada, tal que sgn$[f(a)]$ = - sgn$[f(b)]$. Alguns dos métodos disponíveis:

- Método de Brent (`scipy.optimize.brentq`)
- Método da bisseção (`scipy.optimize.bisect`)
- Método de Newton (`scipy.optimize.newton`)

No caso do método Newton–Raphson, deve-se passar um ponto inicial, `x0` (próximo a raiz), e opcionalmente, a primeira derivada da função, `fprime`. Note que nesse método, temos menos controle sobre a raiz encontrada se a função tem várias raízes.
Métodos numéricos devem ser utilizados com cuidado. Verifique se a raiz x encontrada produz $f(x) \approx 0$.

Raízes - Método de Brent

Vamos encontrar uma das raízes da função abaixo pelo método de Brent

$$f(x) = \frac{1}{5} + x\cos\left(\frac{3}{x}\right).$$

```
In[x]: from scipy.optimize import brentq
In[x]: f = lambda x: 0.2 + x*np.cos(3/x)
In[x]: print(np.sign(f(-0.7)))
       print(np.sign(f(-0.5)))
In[x]: brentq(f,-0.7,-0.5)
Out[x:]: -0.5933306271014237
```

SciPy - Raízes de Funções - Exemplo

Exemplo Na teoria de campo médio do ferromagnetismo, a magnetização M de um material ferromagnético depende da temperatura T como

$$M = \mu \tanh \frac{JM}{k_B T},$$

onde μ é o momento magnético, J é a constante de acoplamento e k_B é a constante de Boltzmann. Fazendo $m = M/\mu$ e $C = \mu J/k_b$, temos

$$m = \tanh \frac{Cm}{T}.$$

Vamos resolver essa equação para diferentes valores de T considerando $C = 1$.

Raízes de Polinômio

Podemos encontrar todas as raízes de um polinômio (reais e complexas), usando a função do NumPy `np.roots()`. Os argumentos são os coeficientes do polinômio.

Raízes Raízes de Polinômio

Exemplo:

$$f(x) = x^4 + x + 1$$

```
In[x]: coef = [1,0,0,1,1]
       np.roots(coef)
Out[x]:
       array([ 0.72713608+0.93409929j,
               0.72713608-0.93409929j,
              -0.72713608+0.43001429j,
              -0.72713608-0.43001429j])
```

SciPy - Ajuste de Curvas

O método `scipy.optimize.curve_fit` permite passarmos de forma transparente os erros da variável *y* e obter as incertezas nos parâmetros ajustados. O método é chamado da seguinte forma: `curve_fit(f,xdata,ydata,p0, sigma, absolute_sigma)`.

- `f, xdata, ydata` são, respectivamente, a função a ser ajustada aos dados (xdata, ydata);
- `p0` é um valor inicial para os parâmetros;
- `sigma` é um array com as incertezas de `ydata`, de mesmo tamanho de `ydata`;
- `absolute_sigma` é uma variável booleana. Se **True**, os valores absolutos de `sigma` são usados. Essa deve ser a opção usada para obter os valores absolutos nas incertezas dos parâmetros. Se escolhermos a opção **False**, os valores de `sigma` são tratados como valores relativos.

SciPy - Ajuste de Curvas

O método `curve_fit` retorna o array `popt`, com o valor dos parâmetros ajustados, e o array 2D `pcov`, a matriz de covariância dos parâmetros. A incerteza nos parâmetros é dada pela raiz quadrada da diagonal de `pcov`: `np.sqrt(np.diag(pcov))`.
Para ilustrar o uso deste método, vamos ajustar uma função linear a um conjunto de pontos de um experimento para determinar a aceleração da gravidade local:

$$T^2 = \frac{4\pi^2}{g} L$$

onde $a = 4\pi^2/g$ é o parâmetro a ser ajustado.

SciPy - Integração Numérica

O pacote `scipy.integrate` contém funções para o cálculo numérico de integrais definidas próprias (limites finitos) e impróprias (limites infinitos). A rotina está implementada em `scipy.integrate.quad`, que é baseada na biblioteca QUADPACK (FORTRAN 77). Os argumentos básicos são o integrando (`func`), e os limites de intergração `a` e `b`. O resultado será um flutuante com o valor da integral e outro com uma estimativa do erro absoluto.

Example

$$I = \int_1^4 x^{-2} dx$$

```
In[x]: from scipy.integrate import quad
In[x]: f = lambda x: 1/x**(2)
In[x]: quad(f,a=1,b=4)
Out[x]:
(0.7500000000000002, 1.9132345482589 95e-09)
```

SciPy - Integração Numérica

O argumento `epsabs` permite estabelecer a tolerância absoluta (um limite superior). Vamos integrar a função $f(x) = e^{-|x|}\text{sen}^2 x^2$ entre -1 e 2:

Example

```
In[x]:
f2 = lambda x: np.exp(-np.abs(x))*np.sin(x**2)**2
In[x]: quad(f2,-1,2,epsabs=0.1)
Out[x]:
(0.29551455828969975, 0.0015295718279094 23)
In[x]: quad(f2,-1,2,epsabs=1.49e-8)
Out[x]:
(0.29551455505239044, 4.4497633167454 7e-10)
```

SciPy - Integração Numérica

Se uma função depende de outros parâmetros além da variável independente, estes devem ser passados como **tuplas** para o argumento `args`.

Example

$$I = \int_{-\pi/2}^{\pi/2} \text{sen}^n x \cos^m x \, dx$$

```
In[x]: def f3(x,n,m):
           return np.sin(x)**n*np.cos(x)**m
In[x]: quad(f3,-np.pi/2,np.pi/2,args=(2,1))
Out[x]:
(0.6666666666666666, 1.6257269518146785e-13)
```

Note que os parâmetros `n` e `m` devem aparecer como argumentos do integrando **depois** da variável de integração `x`.

SciPy - Integração Numérica

Para integrar funções com singularidades, devemos passar um lista de pontos onde ocorrem as divergências usando o argumento `points`.

Example

$$I = \int_{-1}^{1} \frac{dx}{\sqrt{|x|}}$$

```
In[x]: f5 = lambda x: 1/np.sqrt(np.abs(x))
In[x]: quad(f5,-1,1)
Out[x]:
RuntimeWarning: divide by zero encountered in
double_scalars
(inf, inf)
In[x]: quad(f5,-1,1,points=[0,])
Out[x]:
(3.9999999999999813, 5.6843418860808e-14)
```

SciPy - Integração Numérica

Para integrais com limites infinitos, usamos `np.inf`

Example

$$I = \int_0^{\infty} e^{-x^2} dx$$

```
In[x]: f4 = lambda x: np.exp(-x**2)
In[x]: quad(f4,0,np.inf)
Out[x]:
(0.8862269254527579, 7.101318390472462e-09)
```

Exemplo A função de onda do n-ésimo nível de energia do oscilador harmônico quântico em uma dimensão é:

$$\psi_n = \frac{1}{\sqrt{2^n n! \sqrt{\pi}}} e^{-x^2/2} H_n(x),$$

onde $H_n(x)$ é o n-ésimo polinômio de Hermite. Vamos calcular a incerteza na posição de uma partícula no nível n, dada por

$$\langle x^2 \rangle = \int_{-\infty}^{\infty} x^2 \left|\psi_n(x)\right|^2 dx.$$

Integrais duplas, triplas e múltiplas ($n > 3$) podem ser calculadas, respectivamente, com os métodos `dblquad`, `tplquad` e `nquad`. O método `dblquad` calcula integrais do tipo:

$$I = \int_a^b \int_{g(x)}^{h(x)} f(y,x)dydx.$$

O integrando deve ser definido como uma função de pelo menos duas variáveis, `func(y,x...)`, tomando, **necessariamente**, `y` como primeiro argumento e `x` como segundo. Os limites de integração devem ser passados como flutuantes, `a` e `b`, para a integral na variável `x`, e como **funções** de x para a variável `y`.

Example

$$I = \int_1^4 \int_0^2 x^2 y \, dydx$$

```
In[x]: f6 = lambda y,x: x**2*y
       g = lambda x: 0
       h = lambda x: 2
In[x]: dblquad(f6,1,4,g,h)
Out[x]:
(42.000000000000001, 4.662936703425658e-13)
```

SciPy - Integração Numérica - Integrais Múltiplas

O método `tplquad` toma como argumentos uma função `func(z,y,x)` e mais seis argumentos para os limites: `a` e `b` (limites de x), `gfun(x)` `hfun(x)` (limites de y), e `qfun(x,y)` e `rfun(x,y)` (limites de z).

Example

$$V = \int_0^{2\pi} \int_0^{\pi} \int_0^{1} r^2 \text{sen}\theta \, dr d\theta d\phi$$

```
In[x]: def f8(phi,theta,r):
           return r**2*np.sin(theta)
In[x]:
gf = lambda r: 0 ; hf = lambda r: np.pi
qf = lambda r,theta: 0; rf = lambda r,theta: 2*np.pi
In[x]: tplquad(f8,0,1,gf,hf,qf,rf)
Out[x]:
(4.1887902047863905, 1.3890950796949 93e-13)
```

SciPy - Integração Numérica - Integral de Arrays

Para integrar arrays (*samples*), usamos o método `simpson`:
`simpson(y, x=None, dx=1.0, axis=- 1)`.
O argumento `dx` é o espaçamento entre os pontos, usado apenas quando o array `x` não é dado.

Example

```
In[x]: x = np.arange(0,10)
       y = np.arange(0,10)
In[x]: simpson(y,x)
Out[x]: 40.5
```

SciPy - Equações Diferenciais

Equações diferenciais ordinárias (EDOs) podem ser resolvidas numericamente com `scipy.integrate.odeint` ou `scipy.integrate.solve_ivp`. Esses métodos resolvem equações da forma:

$$\frac{d\boldsymbol{y}}{dt} = \boldsymbol{F}(\boldsymbol{y}, t)$$

onde $\boldsymbol{y}$ é um vetor de componentes $y_i(t)$, e $\boldsymbol{F}$ um vetor de componentes $F(y_i, t)$.

Para resolver EDOs de ordem $n > 1$, devemos transformá-las em um sistema de EDOs de primeira ordem (exemplos nos próximos slides).

O método `scipy.integrate.solve_ivp` toma pelo menos três argumentos: uma função que retorna dy/dt, os pontos iniciais e finais da variável t, e um conjunto de condições iniciais y_0.

Exemplo 1:

$$\frac{dy}{dt} = -ky$$

1. Primeiro, definimos dy/dt (note a ordem das variáveis!)
```
def dydt(t,y):
    return -k*y
```
2. Os tempos iniciais e finais devem ser passados como tuplas para o argumento `t_span`: `t_span = (t0,tf)`.
3. Os valores iniciais `y0` devem ser passados como sequência (lista, array), **mesmo que só tenha um valor**.
4. A solução será um objeto `soln` com os arrays `soln.y`, `soln.t` e `soln.success` (booleano).

EDOs Acopladas

$$
\begin{aligned}
\frac{dy_1}{dt} &= f_1(y_1, y_2, ..., y_n; t), \\
\frac{dy_2}{dt} &= f_2(y_1, y_2, ..., y_n; t), \\
&... \\
\frac{dy_n}{dt} &= f_n(y_1, y_2, ..., y_n; t).
\end{aligned}
$$

Nesse caso, a função a ser passada para o método `solve_ivp()` dever retornar uma sequência com as funções $f_i(y_1, y_2, ...y_n; t)$.

EDOs Acopladas - Implementação

```
# Y = [y1, y2, y3, ...]
#(sequencia de variáveis independentes)
def deriv(t, Y):
    y1,y2,y3... = Y
    dy1dt = f1(Y, t)
    dy2dt = f2(Y, t)
    #....
    return dy1dt, dy2dt, ...,dyndt
solve_ivp(deriv, (t0, tf), y0 )
```

Note que agora, `y0` será um sequência de n elementos.

Exemplo 2: Evolução de uma Epidemia (Modelo SIR)

$$\begin{aligned} \frac{dS}{dt} &= -\beta SI, \\ \frac{dI}{dt} &= \beta SI - \gamma I, \\ \frac{dR}{dt} &= \gamma I, \end{aligned}$$

onde as constantes β e γ são, respectivamente, a taxa de transmissão e a taxa de recuperação. Queremos resolver esse sistema para $S(t)$, $I(t)$ e $R(t)$. Para uma população de tamanho fixo N, temos, em qualquer instante, $N = S(t) + I(t) + R(t)$.

Veja mais detalhes em
`www.professores.uff.br/andrenepomuceno/artigos`.

Exemplo 3: EDO de segunda ordem

Para resolver uma EDO de ordem $n > 1$, primeiro devemos reduzi-la a um sistema de EDOs de primeira ordem:

$$\frac{d^2x}{dt^2} = -\omega^2 x$$

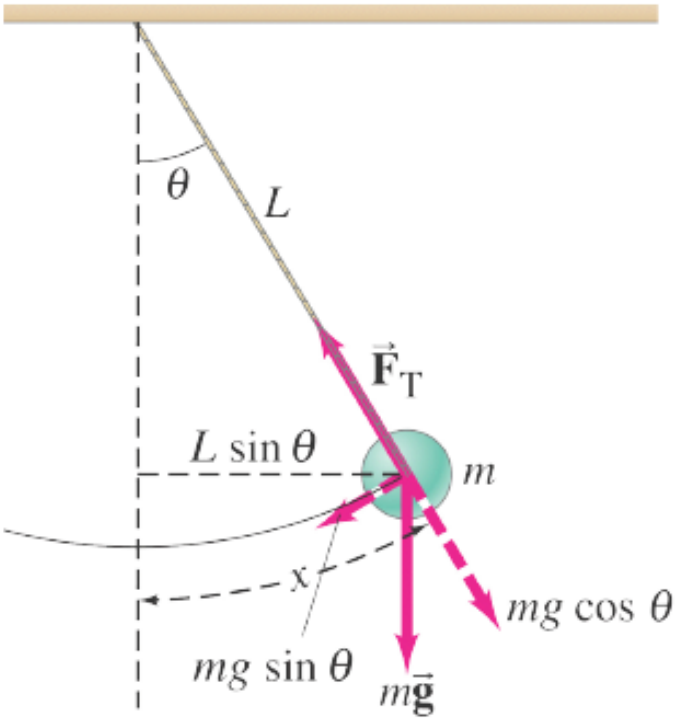
$$\frac{dx}{dt} = v,$$

$$\frac{dv}{dt} = -\omega^2 x,$$

Métodos disponíveis (argumento `method`)

- **RK45**(default): Explicit Runge-Kutta method of order 5(4)
- **RK23** Explicit Runge-Kutta method of order 3(2)
- **DOP853** Explicit Runge-Kutta method of order 8
- **Radau** Implicit Runge-Kutta method of the Radau IIA family of order 5
- **BDF** Implicit multi-step variable-order (1 to 5) method based on a backward differentiation formula for the derivative approximation.

Exemplo 4: Pêndulo Não-Linear



Equação do Movimento:

$$\frac{d^2\theta}{dt^2} + \omega^2 \mathrm{sen}\theta = 0$$

onde $\omega = \sqrt{g/L}$.

Energia total

$$E = \frac{1}{2} m L^2 \dot{\theta}^2 + mg\, l(1 - cos\theta)$$

Fast Fourier Transforms (FFTs)

`fft`(x[, n, axis, norm, overwrite_x, ...])	Compute the 1-D discrete Fourier Transform.
`ifft`(x[, n, axis, norm, overwrite_x, ...])	Compute the 1-D inverse discrete Fourier Transform.
`fft2`(x[, s, axes, norm, overwrite_x, ...])	Compute the 2-D discrete Fourier Transform
`ifft2`(x[, s, axes, norm, overwrite_x, ...])	Compute the 2-D inverse discrete Fourier Transform.
`fftn`(x[, s, axes, norm, overwrite_x, ...])	Compute the N-D discrete Fourier Transform.
`ifftn`(x[, s, axes, norm, overwrite_x, ...])	Compute the N-D inverse discrete Fourier Transform.
`rfft`(x[, n, axis, norm, overwrite_x, ...])	Compute the 1-D discrete Fourier Transform for real input
`irfft`(x[, n, axis, norm, overwrite_x, ...])	Computes the inverse of `rfft`.
`rfft2`(x[, s, axes, norm, overwrite_x, ...])	Compute the 2-D FFT of a real array.
`irfft2`(x[, s, axes, norm, overwrite_x, ...])	Computes the inverse of `rfft2`