TECHMIMO

Autor: Rafael Pereira da Silva

Seguem alguns recados para ajudá-los e para contribuir com o curso:

- Fiquem à vontade para me contatar pelo Linkdin, costumo responder por lá também: https://www.linkedin.com/in/rafael-pereira-da-silva-23890799/ (https://www.linkedin.com/in/rafael-pereira-da-silva-23890799/)
- Fiquem a vontade para compartilharem o certificado do curso no Linkedin. Eu costumo curtir e comentar para dar mais credibilidade
- Vocês podem usar esses notebooks para resolver os exercícios e desafios
- Não se esqueçam de avaliar o curso e dar feedback, eu costumo criar conteúdos baseado nas demandas de vocês
- Se tiverem gostando do curso, recomendem aos amigos, pois isso também ajuda a impulsionar e a crescer a comunidade
- Bons estudos e grande abraços!

Seção 11 - Scipy - Python para Ciências

https://docs.scipy.org/doc/scipy/reference/tutorial/integrate.html (https://docs.scipy.org/doc/scipy/reference/tutorial/integrate.html)

11.1 - Cálculo (integrate)

11.2.1 - Integral

from scipy.integrate import quad, dblquad

Funções	Descrição
integrate.quad(func, a, b)	Calcula a integral de uma função
integrate.dbquad(func, a, b, gfun, hfun)	Calcula a integral dupla de uma função

https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.quad.html#scipy.integrate.quad (https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.quad.html#scipy.integrate.quad)

https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.dblquad.html#scipy.integrate.dblquad (https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.dblquad.html#scipy.integrate.dblquad)

In [3]:

```
import numpy as np
from scipy.integrate import quad, dblquad
```

In [6]:

```
# x**2

f_x = lambda x: x**2
```

In [8]:

```
quad(f_x, 0,10)
```

Out[8]:

```
(333.33333333333326, 3.700743415417188e-12)
```

In [12]:

```
#check
I_x = lambda x: x**3/3
I_x(10)
```

Out[12]:

```
333.3333333333333
```

In [44]:

```
# x**2 + y

f_xy = lambda x,y: x*y
```

In [45]:

```
# dblquad(lambda x,y:eval(str(K_2_integ[i,j])),158.5,201.5,lambda x:0, lambda x:a)
```

In [48]:

```
dblquad(f_xy, 0,5, lambda y:0, lambda y:10)
```

Out[48]:

```
(624.9999999999999, 6.938893903907227e-12)
```

In [49]:

```
I_xy = lambda x,y: (x**2/2)*( y**2/2)
I_xy(5,10)
```

Out[49]:

```
625.0
```

11.2.2 Exercício 1

Dadas as funções, calcule sua integral com x variando de 0 a 10:

$$A(x) = e^x$$

$$B(x) = x^3$$

$$C(x) = 1/x$$

In [83]:

```
import numpy as np
from scipy.integrate import quad
```

In [87]:

```
A_x = lambda x: np.exp(x)
quad(A_x,0,10)
```

Out[87]:

```
(22025.465794806725, 6.239389118119916e-10)
```

In [89]:

```
np.exp(10)
```

Out[89]:

```
22026.465794806718
```

In [88]:

```
B_x = lambda x: x**3
quad(B_x, 0,10)
```

Out[88]:

```
(2500.0000000000005, 2.775557561562892e-11)
```

In [90]:

```
10**4/4
```

Out[90]:

```
2500.0
```

In [98]:

```
C_x = lambda x: 1/x
quad(C_x, 1,10)
```

Out[98]:

```
(2.302585092994052, 1.748598974092929e-08)
```

In [101]:

```
np.log(1)
```

Out[101]:

```
0.0
```

In [102]:

```
np.log(10)
```

Out[102]:

```
2.302585092994046
```

In [ ]:

```

```

11.2.3 Exercício 2

Calcule a integral dupla:

$$A(x,y) = \iint_A dx.dy$$

Considere: $0 \leq x \leq 3$ e $0 \leq y \leq 4$

In [103]:

```
from scipy.integrate import dblquad
```

In [104]:

```
A_xy = lambda x,y: 1
```

In [105]:

```
dblquad(A_xy, 0,3,lambda y:0, lambda y:4)
```

Out[105]:

```
(12.0, 1.3322676295501878e-13)
```

In [ ]:

```

```

In [ ]:

```

```

11.2.4 - EDO

from scipy.integrate import odeint

Funções	Descrição
integrate.odeint(func, y0, t)	Calcula a dada equação diferencial

https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html#scipy.integrate.odeint (https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html#scipy.integrate.odeint)

In [79]:

```
import sympy as sp

x = sp.symbols('x')
y_x = sp.Function('y_x')

dy_x = sp.Derivative(y_x(x),x)

eq = dy_x + 10*x

sp.dsolve(eq)
```

Out[79]:

$$y_x(x) = C_1 - 5x^2$$

In [56]:

```
from scipy.integrate import odeint
```

In [80]:

```
def eq_dif(u, x):
    y,dy = u
    dydx = [dy, -10*x]
    return dydx
```

In [81]:

```
x = list(range(11))

odeint(eq_dif,[0,0],x)
```

Out[81]:

```
array([[    0.        ,     0.        ],
       [   -1.66666669,    -5.00000002],
       [  -13.33333337,   -20.00000002],
       [  -45.00000006,   -45.00000002],
       [ -106.66666675,   -80.00000002],
       [ -208.33333344,  -125.00000002],
       [ -360.00000012,  -180.00000002],
       [ -571.66666681,  -245.00000002],
       [ -853.3333335 ,  -320.00000002],
       [-1215.00000019,  -405.00000002],
       [-1666.66666687,  -500.00000002]])
```

In [ ]:

```

```

11.2.4 - Desafio 1 (técnicas para resolver EDO de segunda ordem)

Resolver a EDO a seguir utilizando o Scipy.

$$m\ddot{x}(t) + c\dot{x}(t) + kx(t) = 0; \quad com\ x(0) = x_0\ e\ \dot{x}(0) = v_0$$

Para fazer a comparação, iremos adotar os valores:

$$10\ddot{x}(t) + 20\dot{x}(t) + 50x(t) = 0; \quad com\ x(0) = 10\ e\ \dot{x}(0) = 0$$

Solução exata usando o Sympy

In [106]:

```
import sympy as sp

t = sp.symbols('t')

x_t = sp.Function('x_t')
dx = sp.Derivative(x_t(t),t)
ddx = sp.Derivative(x_t(t),t,t)

edo = 10*ddx + 20*dx + 50*x_t(t)

(sp.dsolve(edo))
```

Out[106]:

$$\mathrm{x_t}(t) = (C_1 \sin(2t) + C_2 \cos(2t))\, e^{-t}$$

In [107]:

```
# Resolvendo o problema de valor inicial

#Montando x e dx
C1,C2 = sp.symbols(['C1','C2'])
x_t_solved = (C1*sp.sin(2*t) + C2*sp.cos(2*t))*sp.exp(-t)
dx_t_solved = sp.diff(x_t_solved,t)

#Aplicando as condições iniciais
# x_0
eq_1 = x_t_solved.subs(t,0) - 10
C2_solved = sp.solve(eq_1,C2)[0]
C2_solved


#v0
eq_2 = dx_t_solved.subs(t,0)
C1_solved = sp.solve(eq_2,C1)[0]
C1_solved



x_t = x_t_solved.subs(C1,C1_solved).subs(C2,C2_solved)
dx_t = sp.diff(x_t,t)
ddx_t = sp.diff(x_t,t,t)

print(x_t)
print(dx_t)
print(ddx_t)
```

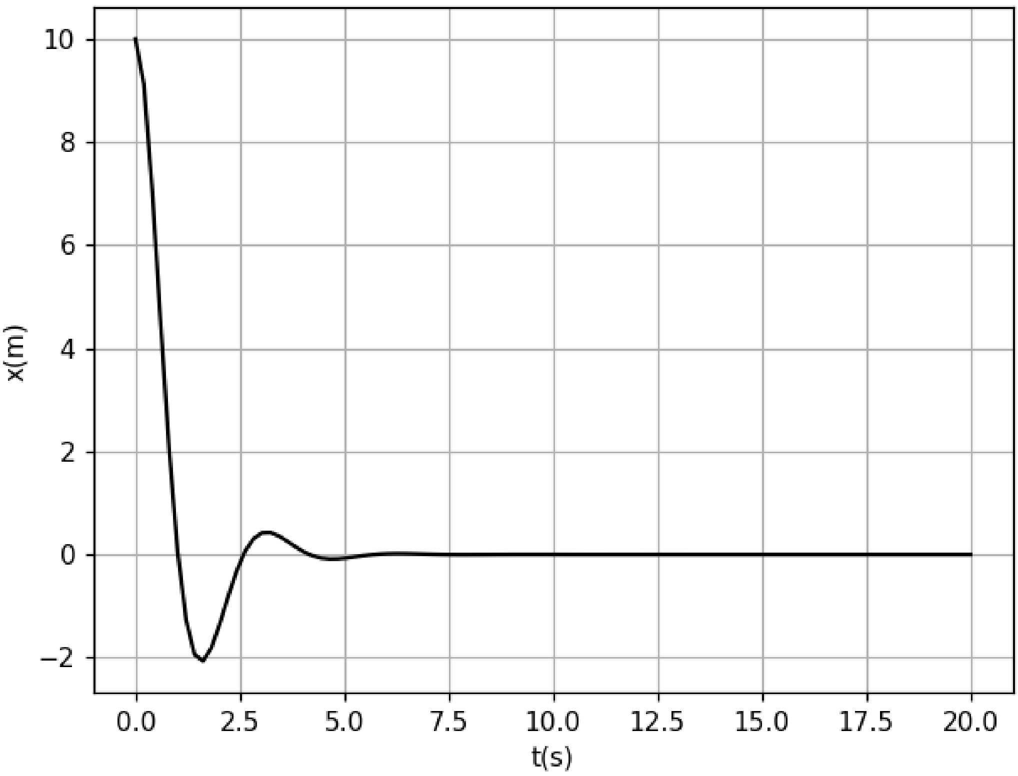
```
(5*sin(2*t) + 10*cos(2*t))*exp(-t)
(-20*sin(2*t) + 10*cos(2*t))*exp(-t) - (5*sin(2*t) + 10*cos(2*t))*exp(-t)
5*(5*sin(2*t) - 10*cos(2*t))*exp(-t)
```

In [108]:

```
import numpy as np

def func_x(t):
    return((5*np.sin(2*t) + 10*np.cos(2*t))*np.exp(-t))

def func_dx(t):
    return((-20*np.sin(2*t) + 10*np.cos(2*t))*np.exp(-t) - (5*np.sin(2*t) + 10*np.cos(2*t))


def func_ddx(t):
    return(5*(5*np.sin(2*t) - 10*np.cos(2*t))*np.exp(-t))


t_arr_1 = np.linspace(0,20,100)
x_arr_1 = func_x(t_arr_1)
dx_arr_1 = func_dx(t_arr_1)
ddx_arr_1 = func_ddx(t_arr_1)
```

In [123]:

```
import matplotlib.pyplot as plt
%matplotlib notebook
plt.plot(t_arr_1,x_arr_1,'k-')
plt.xlabel('t(s)')
plt.ylabel('x(m)')
plt.grid()
```



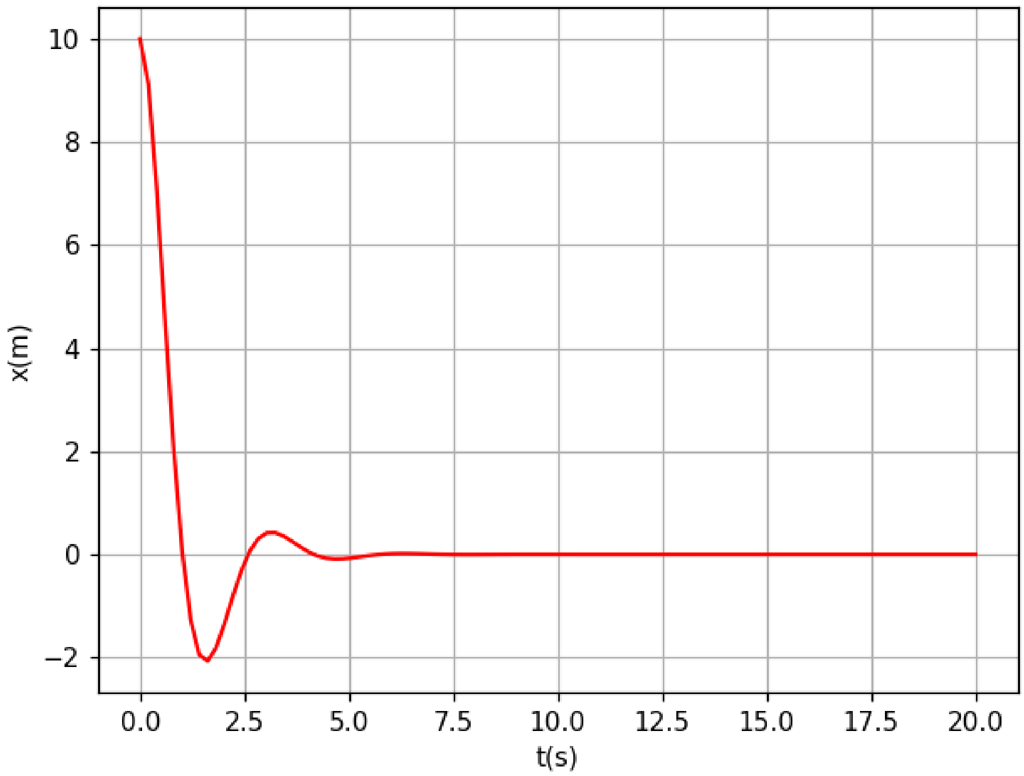
Solução aproximada usando o Odeint do Scipy

In [119]:

```
import numpy as np
from scipy.integrate import odeint

#ddx_dt = Fm/m - g*np.sin(theta) - mu*g**np.cos(theta) - 0.5*rho*Cd*dx**2
# https://sam-dolan.staff.shef.ac.uk/mas212/notebooks/ODE_Example.html

def dU_dt(U,t):
    # U[0] = x e U[1] = dx
    return [U[1], - 2*U[1] - 5*U[0] ]

U0 = [10,0]

ts = np.linspace(0,20,100)
Us = odeint(dU_dt,U0,ts)
```

In [125]:

```
%matplotlib notebook
plt.plot(ts,Us[:,0],'r-')
plt.xlabel('t(s)')
plt.ylabel('x(m)')
plt.grid()
```



Comparação entre as soluções

Os erros são calculados por meio da soma de todos os erros dos vetorers de posição, velocidade e aceleração. Logicamente o $\ddot{x}(t)$ teve o maior erro, mas ainda assim foi muito baixo. A solução numérica do scipy apresenta resultados muito próximos do da solução exata.

In [129]:

```
Error_x = x_arr_1 - Us[:,0]
Error_x.max()
```

Out[129]:

```
8.551405050738481e-08
```

In [130]:

```
Error_dx = dx_arr_1 - Us[:,1]
Error_dx.max()
```

Out[130]:

```
5.739379513869380e-07
```

In [131]:

```
Error_ddx = ddx_arr_1 - (- 2*Us[:,1] - 5*Us[:,0])
Error_ddx.max()
```

Out[131]:

```
9.364198785277722e-07
```

In [ ]:

```

```