Introdução à Computação em Física

PROGRAMAÇÃO SIMBÓLICA (PARTE 2) E GRÁFICOS EM PYTHON

PROF. WALBER

Aula anterior: Programação Simbólica

- Possibilidade de computação com objetos matemáticos e expressões matemáticas (de maneira analítica ao invés de puramente numérica);
- Pode ser utilizada para simplificação de problemas mais complexos que necessitam de métodos numéricos;
- Nesse contexto, faremos uso do módulo SymPy (Symbolic Python).

Mais informações em: www.sympy.org

SymPy

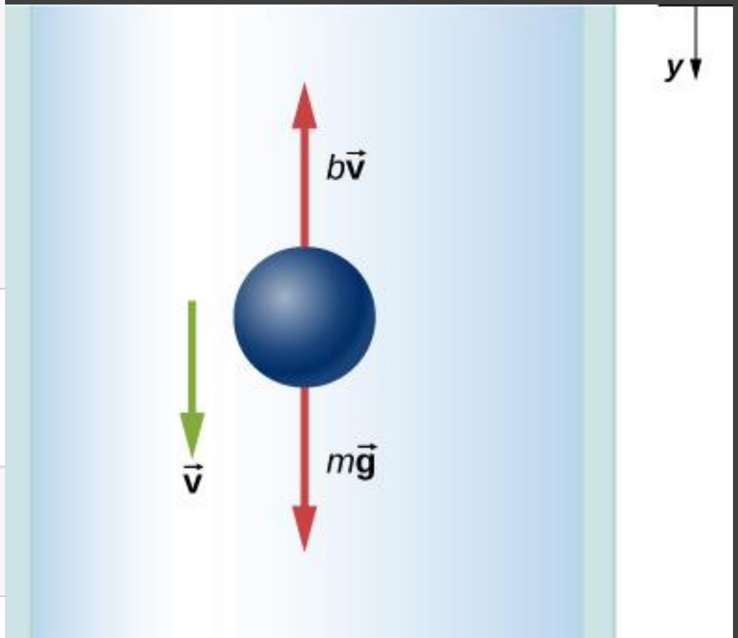
Queda livre com resistência do ar (com condições iniciais)

```python
import sympy as sp
sp.init_printing()
t, g, k = sp.symbols("t, g, k")

y = sp.Function("y")

eqy = sp.Eq(y(t).diff(t,2),g - k*y(t).diff(t,1))
eqy
```

$$\frac{d^2}{dt^2}y(t) = g - k\frac{d}{dt}y(t)$$

```python
ysol = sp.dsolve(eqy, ics={y(0):20, y(t).diff(t,1).subs(t,0): 0})
ysol
```

$$y(t) = \frac{gt}{k} + \frac{ge^{-kt}}{k^2} + \frac{-g + 20k^2}{k^2}$$



Obtendo resultados numéricos ou simplificações:

Pode ser feita com a opção .subs()

```
import sympy as sp
sp.init_printing()
x = sp.Symbol("x")

### Equação
A = 4*x**4 + 3*x**3 + 2*x**2 + 1
A
```

$$4x^4 + 3x^3 + 2x^2 + 1$$

```
### Substituindo x por 2.2
A.subs(x,2.2)
```

$$136.3264$$

Pode ser variáveis em uma equação:

```
B = 4*x + y
```

```
C = B.subs(x,y)
C
```

$$5y$$

Mais detalhes em:
https://docs.sympy.org/latest/tutorials/intro-tutorial/basic_operations.html

Obtendo resultados numéricos ou simplificações:

Pode ser feita com a opção .subs()

$$y(t) = \frac{gt}{k} + \frac{ge^{-kt}}{k^2} + \frac{-g + 20k^2}{k^2}$$

```
ysol.subs([(t,0),(g,9.8),(k,0.01)])
```

$$y(0) = 20.0$$

Trocamos os símbolos da equação y(t) por números.

Mais detalhes em:
https://docs.sympy.org/latest/tutorials/intro-tutorial/basic_operations.html

Obtendo resultados numéricos com uma dada precisão

Pode ser feita com a opção .evalf()

```
sp.pi.evalf(20)
```

3.1415926535897932385

```
D = sp.sqrt(20.4)
```

```
D.evalf(4)
```

4.517

Usada para calcular uma expressão numérica como números reais com uma dada precisão.

Mais detalhes em:
https://docs.sympy.org/latest/tutorials/intro-tutorial/basic_operations.html

Obtendo resultados numéricos

Pode ser feita com a opção .evalf()

$$y(t) = \frac{gt}{k} + \frac{ge^{-kt}}{k^2} + \frac{-g + 20k^2}{k^2}$$

```
ysol.evalf(7,subs={t:1.20, g:9.8, k:0.01})
```

$$y(1.2) = 27.02786$$

Usada para calcular uma expressão numérica como números reais considerando uma dada precisão

Mais detalhes em:
https://docs.sympy.org/latest/tutorials/intro-tutorial/basic_operations.html

Lembranças da função lambda

Função anônima (coringa) que não precisa ser declarada(definida) antes de sua utilização.

Exemplo:

```
def poli2(x):
    f = x**2 + 2.0
    return f

print("O valor de f(2.0) e igual a", poli2(2.0))
```

```
O valor de f(2.0) e igual a 6.0
```

Maneira padrão de uma dada função declarada

Função lambda:

```
f2 = lambda x: x**2 + 2.0
```

```
print("O valor de f(2.0) e igual a", f2(2.0))
```

```
O valor de f(2.0) e igual a 6.0
```

Sintaxe:

Funcao = lambda variavel1 variavel 2 variavel 3 ... : expressão

Lembranças da função lambda

Função anônima (coringa) que não precisa ser declarada(definida) antes de sua utilização.

Pode ser usada para uma lista através das funções map e list:

```
lista1 =  [1, 2, 3, 4, 5, 6]

lista2 = list(map(lambda x: x**2, lista1))
lista2
```

```
[1, 4, 9, 16, 25, 36]
```

Mais informações em: https://realpython.com/python-map-function/

Voltando para o Sympy

https://docs.sympy.org/latest/tutorials/intro-tutorial/basic_operations.html

No Sympy temos uma função que converte uma expressão em uma função lambda (Lambdificação)

```
import sympy as sp
import numpy as np
sp.init_printing()
x = sp.Symbol("x")

#### A = cos(2*x)
A = sp.cos(2*x)

### lista de valores x
xl = np.linspace(0,1,5)

### conversao de uma expressao A em uma outra que
### pode ser calculada numericamente usando a numpy
f = sp.lambdify(x,A,"numpy")

#### valores de f na lista xl
print("xl = ", xl)
print("f(xl) = ", f(xl))
```

```
xl =  [0.   0.25 0.5  0.75 1.  ]
f(xl) =  [ 1.          0.87758256  0.54030231  0.0707372  -0.41614684]
```

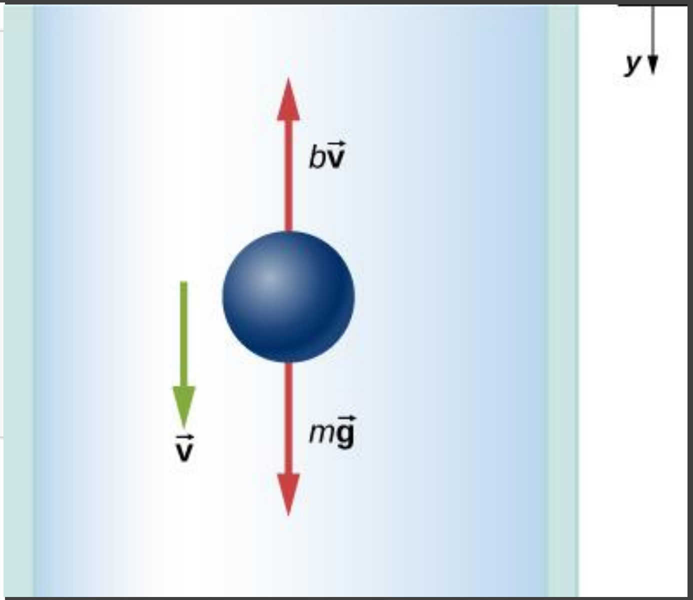
Voltando para a queda livre

```python
import sympy as sp
import numpy as np
sp.init_printing()
t, g, k = sp.symbols("t, g, k")
y = sp.Function("y")
eqy = sp.Eq(y(t).diff(t,2),g - k*y(t).diff(t,1))
eqy
```

$$\frac{d^2}{dt^2}y(t) = g - k\frac{d}{dt}y(t)$$

```python
ysol = sp.dsolve(eqy, ics={y(0):20, y(t).diff(t,1).subs(t,0): 0})
ysol
```

$$y(t) = \frac{gt}{k} + \frac{ge^{-kt}}{k^2} + \frac{-g + 20k^2}{k^2}$$



Voltando para a queda livre

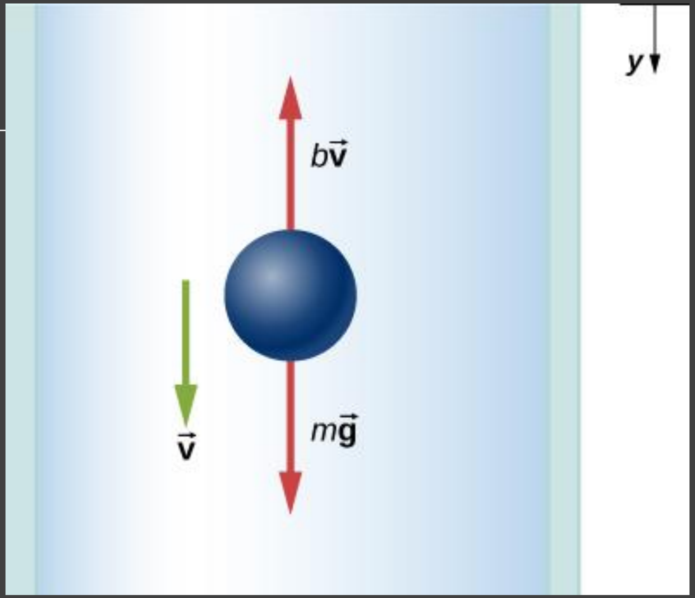
$$y(t) = \frac{gt}{k} + \frac{ge^{-kt}}{k^2} + \frac{-g + 20k^2}{k^2}$$

```
k1 = 0.01
tl = np.linspace(0,3,10)
```

```
y_t = sp.lambdify(t, ysol.rhs.subs({g:9.8, k:k1}), 'numpy')
```

```
y_t(tl)
```

```
array([20.        , 20.54384001, 22.17294633, 24.88370742, 28.67252377,
       33.53580785, 39.46998406, 46.4714887 , 54.53676994, 63.66228775])
```



Seria interessante fazer uma análise gráfica para diferentes valores de k.

Gráficos em python

Gráficos no Python:

Para plot de gráficos, iremos utilizar em alguns casos a biblioteca Numpy (Numerical python):

Numpy é uma biblioteca para computação científica em Python, que fornece objetos como arrays, funções para algebra linear, leitura e escrita de dados, etc.

```
import numpy as np
```

www.numpy.org

Gráficos no Python:

Iremos também fazer uso da biblioteca Matplotlib:

Biblioteca para geração de gráficos 2D e 3D, em uma grande variedade de formatos

```
import matplotlib.pyplot as plt
```

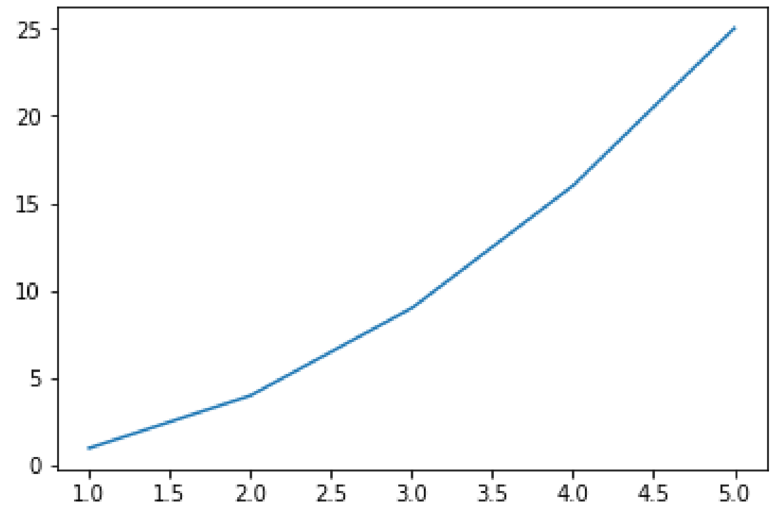
www.matplotlib.org

Galeria: https://matplotlib.org/gallery/index.html

Relembrando:

Gráfico simples, onde iremos plotar os números de uma sequência ao quadrado:

```python
import matplotlib.pyplot as plt

## lista de numeros
x = [1,2,3,4,5]
## quadrado dos numeros
y = [1,4,9,16,25]

plt.plot(x,y)

## Mostrar grafico

plt.show()
```



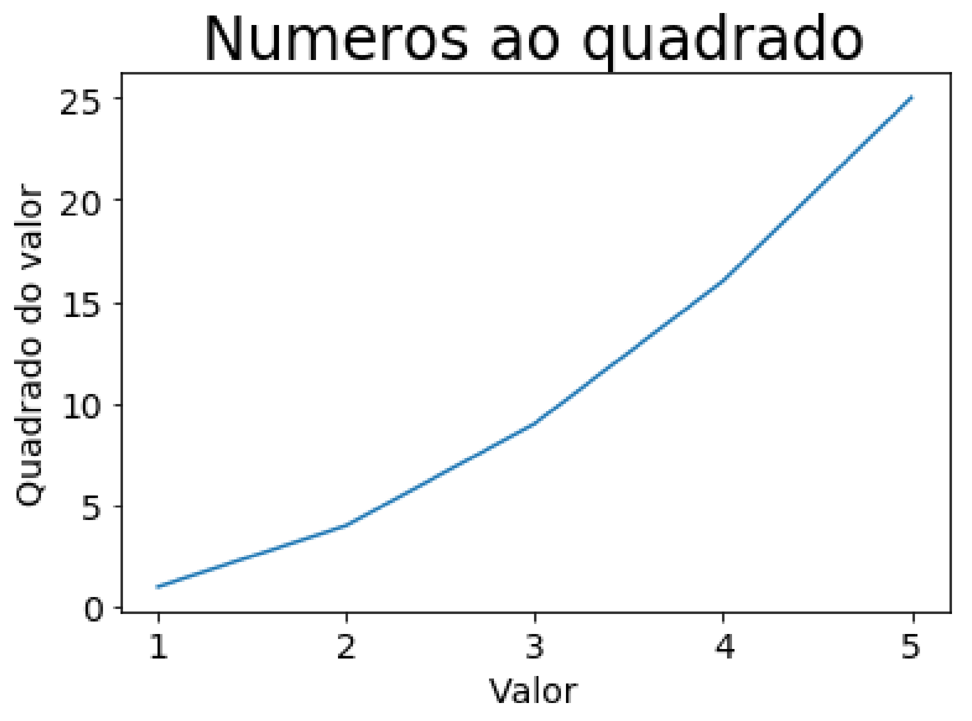
Dados definidos pelas listas x e y;

Comando plt.plot(x,y) plota y em função de x. Note que as listas x e y devem possuir a mesma dimensão.

Comando plt.show() mostra o gráfico

Incrementando o gráfico:

```python
import matplotlib.pyplot as plt

## lista de numeros
x = [1,2,3,4,5]
## quadrado dos numeros
y = [1,4,9,16,25]

## Definir o titulo e nomeia os eixos do grafico
plt.title("Numeros ao quadrado", fontsize=24)
## eixo x
plt.xlabel("Valor", fontsize=14)
## eixo y
plt.ylabel("Quadrado do valor", fontsize=14)



plt.plot(x,y)

## controla o tamanho dos marcadores
plt.tick_params(axis='both', labelsize=14)

## salvando arquivo figura
plt.savefig('plotsimples.png', bbox_inches='tight')

## Mostrar grafico

plt.show()
```



Extensões suportadas:

eps, pdf, pgf, png, ps, raw, rgba, svg, svgz

Exemplo 2

Plotando pontos com scatter(), que auxiliam enfatizar determinadas caracteristicas:

```
import matplotlib.pyplot as plt

## lista de numeros
x = [1,2,3,4,5]
## quadrado dos numeros
y = [1,4,9,16,25]

## Definir o titulo e nomeia os eixos do grafico
plt.title("Numeros ao quadrado", fontsize=24)
## eixo x
plt.xlabel("Valor", fontsize=14)
## eixo y
plt.ylabel("Quadrado do valor", fontsize=14)

## color indica a cor manualmente
## s indica o tamanho dos pontos

plt.scatter(x,y, color='orange', s=16)

## controla o tamanho dos marcadores
plt.tick_params(axis='both', labelsize=14)

## salvando arquivo figura
plt.savefig('plotsimples.png', bbox_inches='tight')

## Mostrar grafico

plt.show()
```

Exemplo 3

Utilização do colormap para enfatizar os pontos com maior valor:

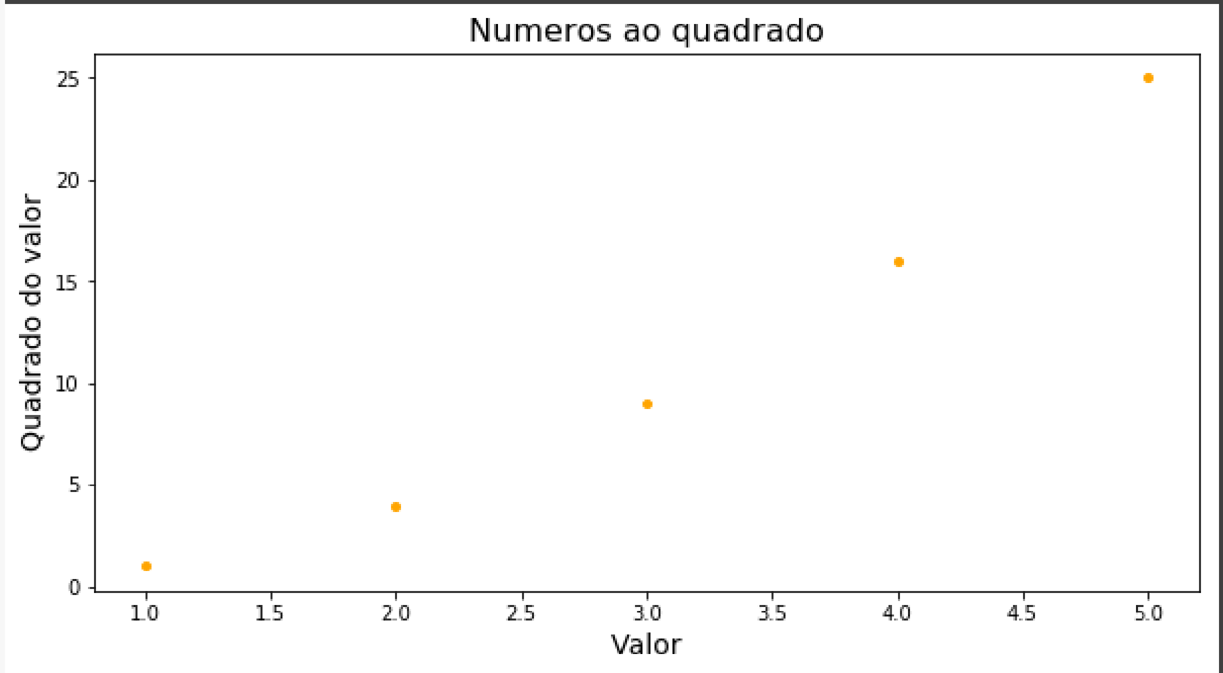
```
import matplotlib.pyplot as plt

## definicao lista x
x = list(range(1,1001))
## obtencao dos quadrados
y = [i**2 for i in x]

## s define o tamanho do ponto e edge color tira o contorno do ponto
## c = indica os valores que serao enfatizados com colormap
## cmap define o mapa de cores usados
plt.scatter(x,y, c=y, cmap=plt.cm.hot, edgecolor='none', s=8)

## Definir o titulo e nomeia os eixos do grafico
plt.title("Numeros ao quadrado", fontsize=16)
## eixo x
plt.xlabel("Valor", fontsize=14)
## eixo y
plt.ylabel("Quadrado do valor", fontsize=14)
## barra de cores
plt.colorbar()

plt.show()
```

Esquemas de cores disponíveis em: https://matplotlib.org/3.1.0/tutorials/colors/colormaps.html

Estruturação de gráficos

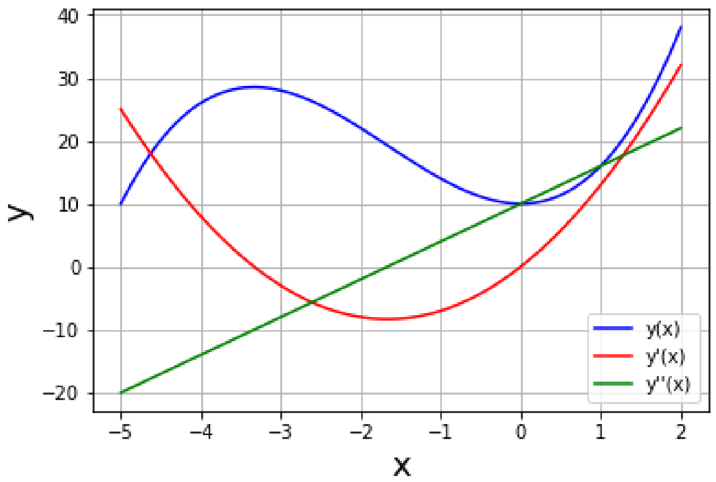
O Matplotlib estrutura o gráfico em Figures com opções de tamanho:

```
import matplotlib.pyplot as plt


### definindo propriedades do Figure

fig = plt.figure(figsize=(10,5))

x = [1,2,3,4,5]
y = [1,4,9,16,25]

plt.title("Numeros ao quadrado", fontsize=16)
plt.xlabel("Valor", fontsize=14)
plt.ylabel("Quadrado do valor", fontsize=14)
plt.scatter(x,y, color='orange', s=16)

plt.show()
```



Exemplo Plot1

```
In [6]: import numpy as np
        import matplotlib.pyplot as plt

        ## geração da malha de pontos x
        x = np.linspace(-5, 2, 100)

        y1 = x**3 + 5*x**2 + 10   ### funçao y(x)
        y1_1d = 3*x**2 + 10*x     ### primeira derivada
        y1_2d = 6*x + 10          ### segunda derivada

        fig, ax = plt.subplots()
        ax.plot(x,y1, color='blue', label='y(x)')
        ax.plot(x,y1_1d, color='red', label="y'(x)")
        ax.plot(x,y1_2d, color='green',label="y''(x)")
        ax.set_xlabel("x", fontsize=18)
        ax.set_ylabel("y", fontsize=18)
        ax.legend()

        plt.grid()
        plt.show()
```



Estruturação de gráficos (subplot)

O Matplotlib também utiliza o conceito de subplot para geração de vários gráficos em uma única janela;

Numpy utilizado para geração dos dados para o gráfico (x);

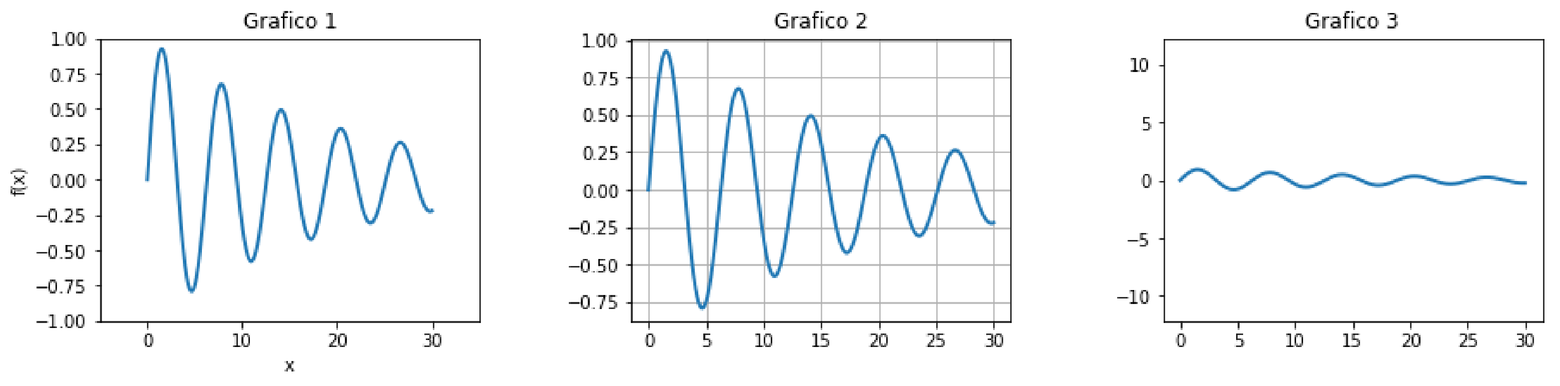
np.linspace(-5,2,100) gera um array de pontos x (100 no total) que inicia em –5.0 e vai até 2.0;

Legendas com ax.legend() e grade com plt.grid();

Estruturação de gráficos (subplot)

Três gráficos em uma janela:

```
import matplotlib.pyplot as plt
import numpy as np

### definindo propriedades do Figure (1x3)

fig, axes = plt.subplots(1,3, figsize=(15,3))

x = np.linspace(0,30,500)

### funcao y(x) = sin(x)*e^{-x/20}

y = np.sin(x)*np.exp(-x/20)

## primeiro grafico

axes[0].plot(x,y,lw=2)         ### lw espessura da linha
axes[0].set_xlim(-5,35)        ### limites eixo x
axes[0].set_ylim(-1,1)         ### limites eixo y
axes[0].set_title("Grafico 1")
axes[0].set_xlabel("x")
axes[0].set_ylabel("f(x)")

## segundo grafico

axes[1].plot(x,y,lw=2)
axes[1].axis('tight')        ### ajusta sem espaco vago
axes[1].set_title("Grafico 2")
axes[1].grid()
## terceiro grafico

axes[2].plot(x,y,lw=2)
axes[2].axis('equal')    ### ranges mais igualados
axes[2].set_title("Grafico 3")

## espacamento entre graficos

plt.subplots_adjust(wspace=0.4)

plt.show()
```

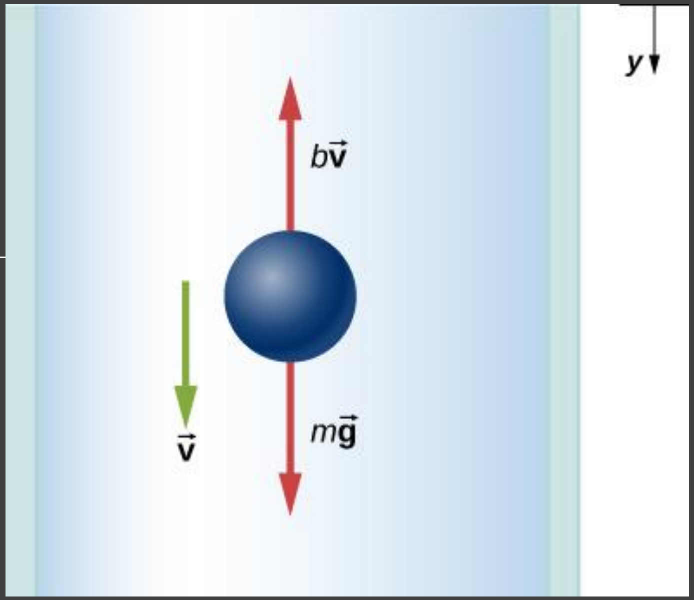


plt.subplots(nlinhas,ncolunas, ...)
axes[N]. : N número do gráfico, contando a partir de 0;
plt.subplots_adjust() controla o espaçamento entre os gráficos

Voltando ao nosso exemplo: Partícula em queda livre com resistência do ar

```python
import sympy as sp
import numpy as np
import matplotlib.pyplot as plt
sp.init_printing()
t, g, k = sp.symbols("t, g, k")
y = sp.Function("y")
eqy = sp.Eq(y(t).diff(t,2),g - k*y(t).diff(t,1))
eqy
```

$$\frac{d^2}{dt^2}y(t) = g - k\frac{d}{dt}y(t)$$



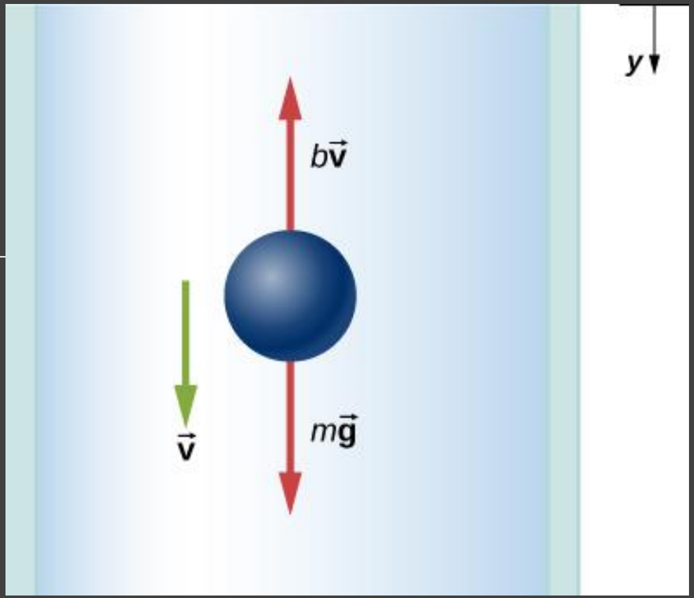

```python
ysol = sp.dsolve(eqy, ics={y(0):20, y(t).diff(t,1).subs(t,0): 0})
ysol
```

$$y(t) = \frac{gt}{k} + \frac{ge^{-kt}}{k^2} + \frac{-g + 20k^2}{k^2}$$

Voltando a nosso exemplo: Partícula em queda livre com resistência do ar

```
ysol = sp.dsolve(eqy, ics={y(0):20, y(t).diff(t,1).subs(t,0): 0})
ysol
```

$$y(t) = \frac{gt}{k} + \frac{ge^{-kt}}{k^2} + \frac{-g + 20k^2}{k^2}$$




```
k1 = 0.01
tl = np.linspace(0,3,10)
```

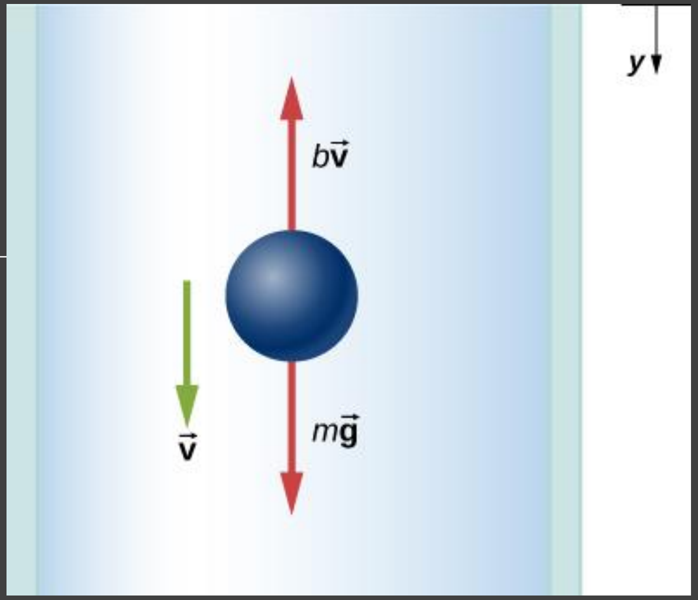
```
y_tk1 = sp.lambdify(t, ysol.rhs.subs({g:9.8, k:k1}), 'numpy')
```

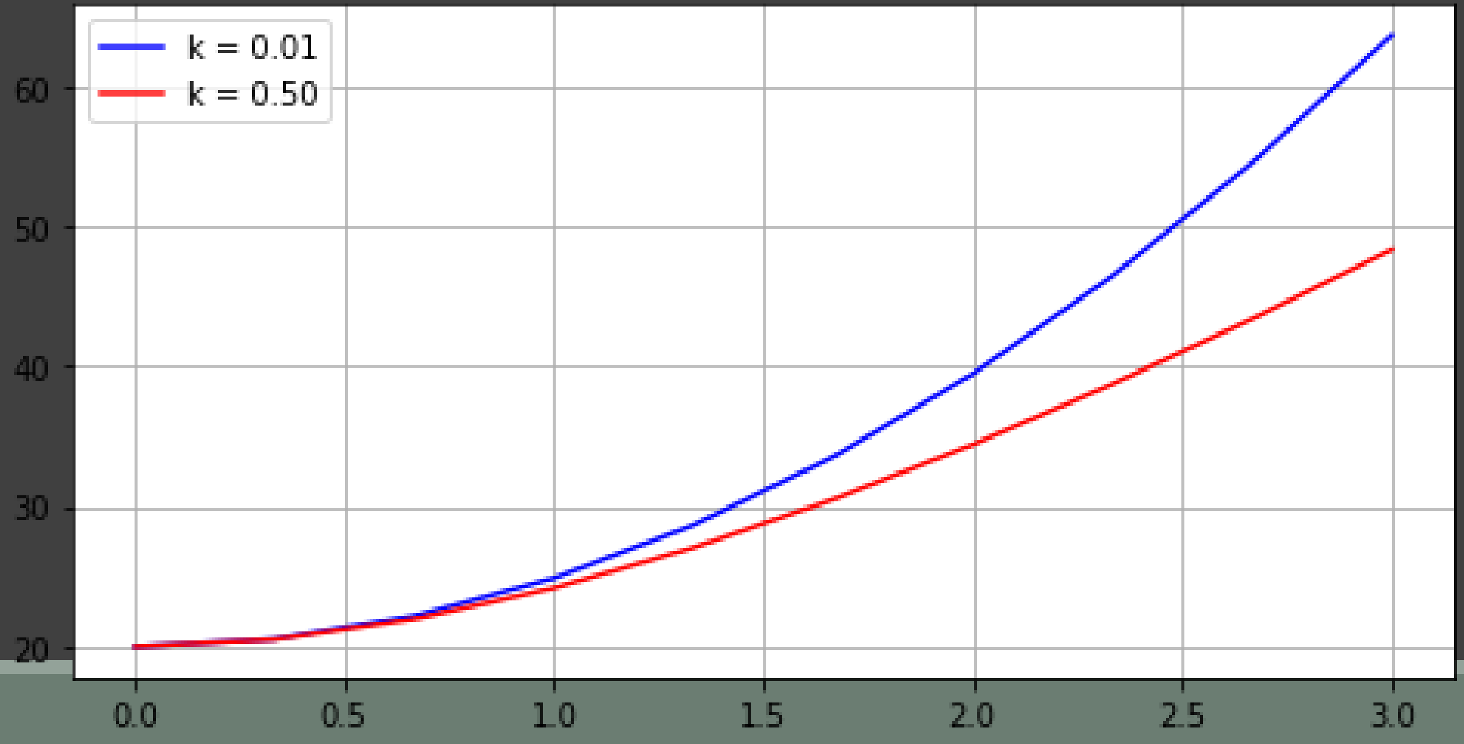
```
A = y_tk1(tl)
```

```
k2 = 0.50
y_tk2 = sp.lambdify(t, ysol.rhs.subs({g:9.8, k:k2}), 'numpy')
B = y_tk2(tl)
```

Voltando a nosso exemplo: Partícula em queda livre com resistência do ar

```python
fig, ax = plt.subplots(figsize=(8, 4))
ax.plot(tl, A,color='blue', label='k = 0.01')
ax.plot(tl, B,color='red', label='k = 0.50')
plt.legend()
plt.grid()
plt.show()
```





Outros tipos de gráficos

Gráficos de funções de duas variáveis (colormap)

Representação de dados com cores (heat meaps) de uma função z(x,y), o valor de z é representado por uma cor:

```
import matplotlib.pyplot as plt
import numpy as np
import matplotlib as mpl

### definindo malha de pontos x e y

x = y = np.linspace(-10,10,50)

X, Y = np.meshgrid(x,y)

np.shape(x)
```

```
(50,)
```

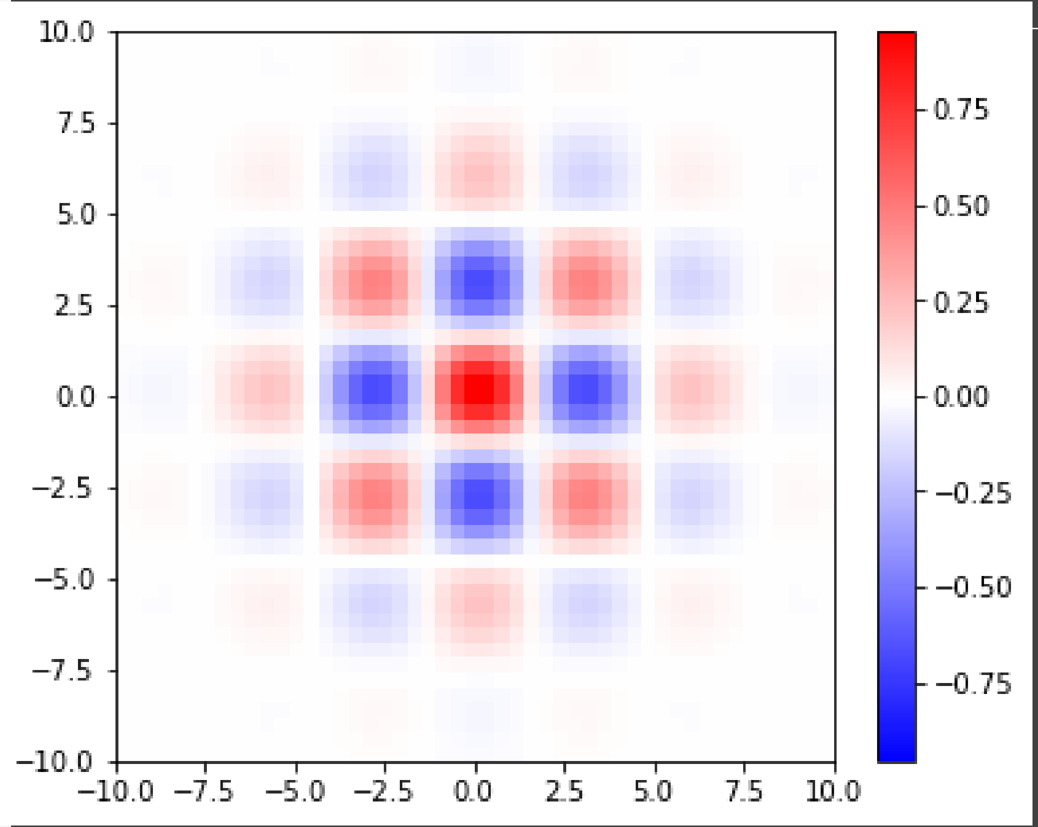
```
np.shape(X)
```

```
(50, 50)
```

```
### funcao z(x,y)
Z = np.cos(X)*np.cos(Y)*np.exp(-(X/5)**2 - (Y/5)**2)

np.shape(Z)
```

```
(50, 50)
```

```
fig,ax = plt.subplots(figsize=(6,5))

### Normalizacao dos dados para intervalo -1 a 1

norm = mpl.colors.Normalize(-abs(Z).max(), abs(Z).max())

### comando para plotar z(x,y)
p = ax.pcolor(X,Y,Z, norm=norm, cmap=mpl.cm.bwr)

fig.colorbar(p)
```

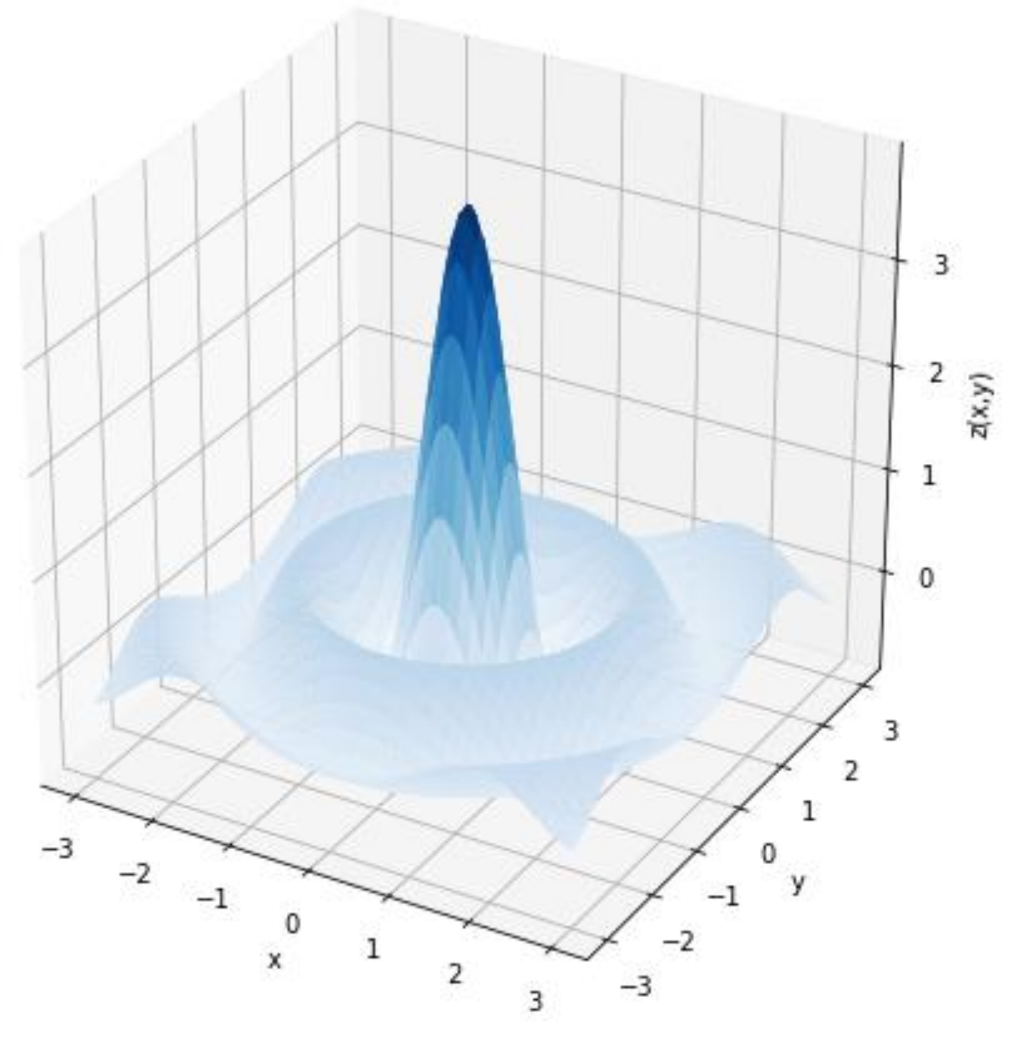


Fazendo uso da instrução pcolor

Gráficos de funções de duas variáveis (3D)

Representação de dados z(x,y), agora em 3D (plot_surface):

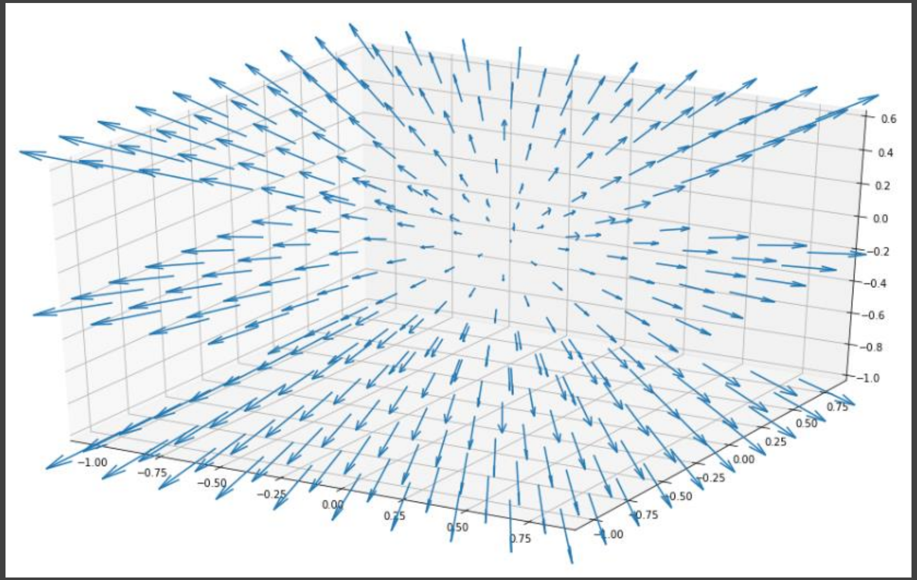
```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import axes3d

x = y = np.linspace(-3,3, 100)
X,Y = np.meshgrid(x,y)

### funcao z(x,y) = sin(4*sqrt(x^2 + y^2))/sqrt(x^2+y^2)

Z = np.sin(4*np.sqrt(X**2 + Y**2))/np.sqrt(X**2 + Y**2)

### plot
fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(111, projection='3d')

ax.plot_surface(X, Y, Z, cmap=mpl.cm.Blues)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z(x,y)')

plt.show()
```



Campo vetorial (3D):

Representação de um campo vetorial em 3D (quiver):

```python
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import axes3d

fig = plt.figure(figsize=(18,10))
ax = fig.gca(projection='3d')

### Malha de pontos x,y,z

x, y, z = np.meshgrid(np.arange(-1, 1, 0.2),
                      np.arange(-1, 1, 0.2),
                      np.arange(-1, 1, 0.8))

### componentes do Campo vetorial: \vec{A} = x \hat{i} + y \hat{j} + z \hat{k}

u=x
v=y
w=z
### plotar campo vetorial
ax.quiver(x, y, z, u, v, w, length=0.2)

plt.show()
```



Atividades práticas:

1. Escrever um código em python para calcular a velocidade terminal (equação) do objeto em queda livre na presença de um termo de resistência do ar. Fazer os plots da velocidade na ausência e na presença da resistência do ar.

2. Fazer os plots das atividades práticas da aula passada (sympy): (a) lançamento do projeto com resistência do ar, (b) oscilador harmônico e (c) oscilador harmônico amortecido.

3. Escreva um programa para plotar o campo vetorial :

$$\vec{g} = \frac{\vec{r}}{r^3}$$