Programação II

Gráficos

Parte 1

Hugo Pacheco

DCC/FCUP

21/22

“Uma imagem vale mil palavras”

- Enormes quantidades de informação em programação e, em particular, na análise computacional de dados

- Os humanos foram feitos para processar dados visuais, não textuais

- Gráficos, e outras formas de visualização, são essenciais para nos ajudar a compreender os dados

    - Ajudam a identificar padrões, tendências e correlações

    - São um veículo de eleição para partilhar informação

    - Permitem fornecer informação de variadas formas e com diferentes níveis de detalhe

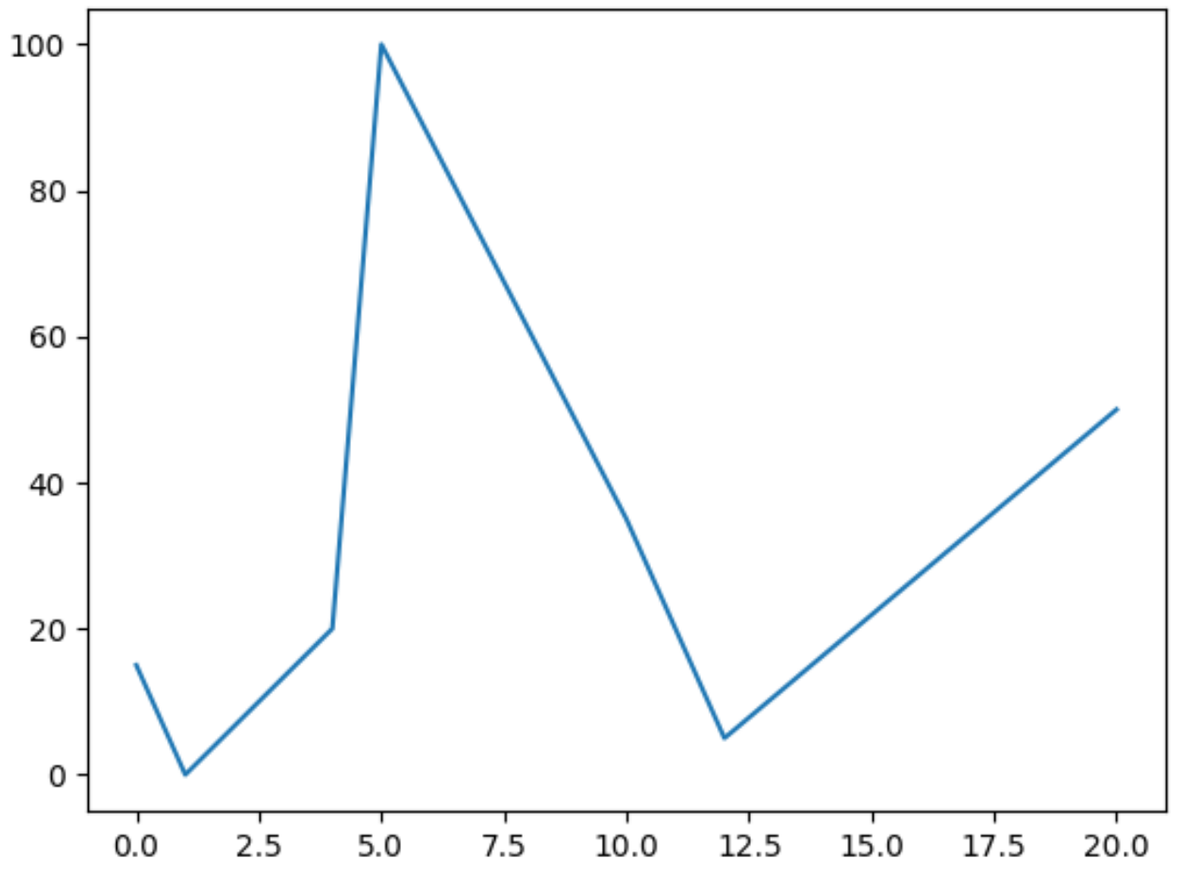
Gráficos

- Nas próximas aulas vamos aprender a desenhar com a biblioteca *matplotlib*, que é muito utilizada para gerar gráficos científicos para grandes conjuntos de dados
  - Gráficos 2D e 3D (vamos olhar apenas para 2D)
  - Diferentes formatos de output (PNG, PDF, GUI, etc)
  - Suporte para gráficos interativos
  - Suporte para mapas geográficos

Matplotlib

- Gráficos 2D são definidos por 2 sequências:
  - Sequência de valores no eixo dos X
  - Sequência de pontos no eixo dos Y para cada X

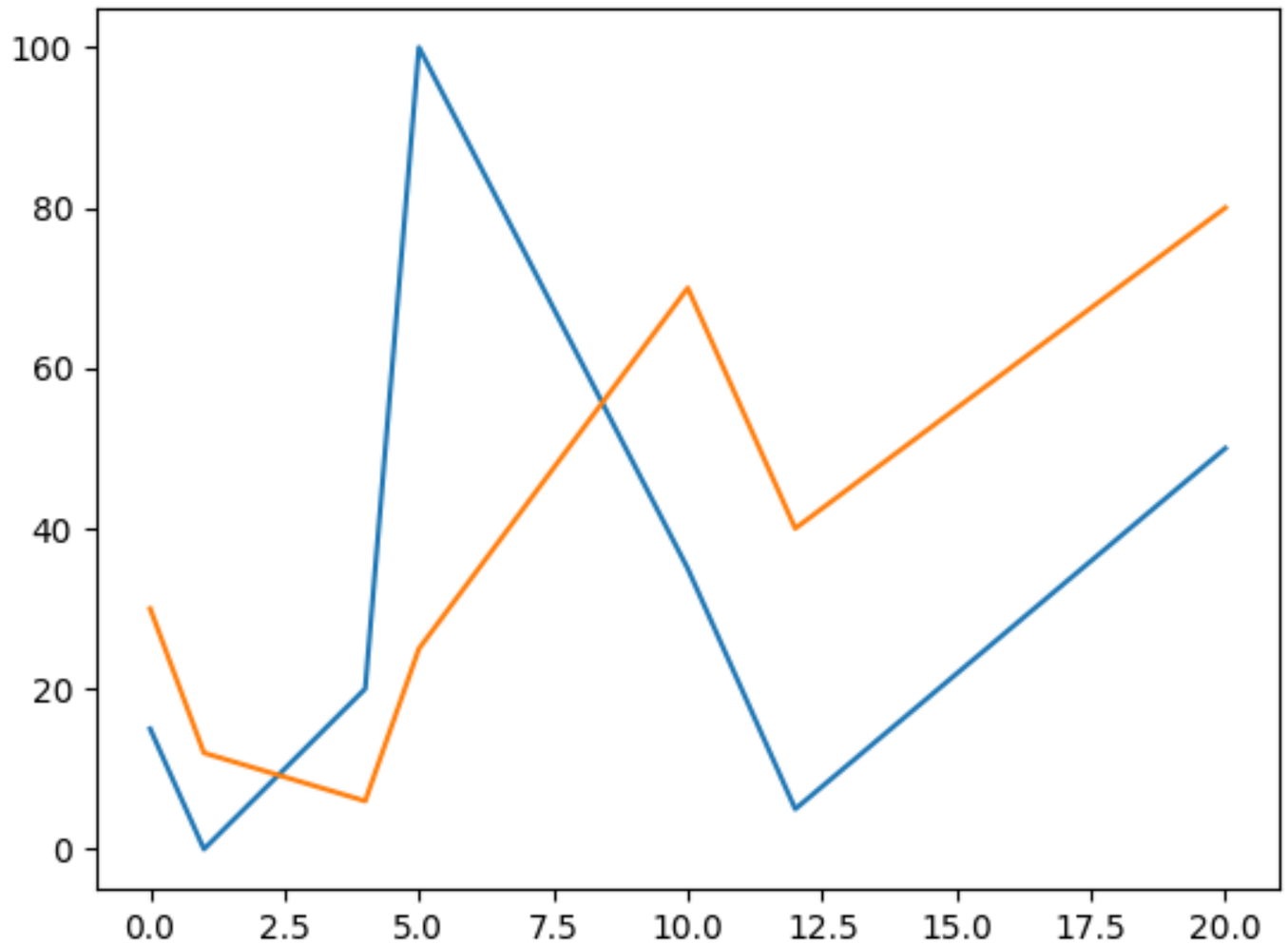
```
import matplotlib.pyplot as plt
x = [0,1,4,5,10,12,20]
y = [15,0,20,100,35,5,50]
plt.plot(x,y)
plt.show()
```



Matplotlib

- Podemos desenhar mais do que uma curva nos Y, para os mesmos X

```
import matplotlib.pyplot as
plt
x = [0,1,4,5,10,12,20]
y1 = [15,0,20,100,35,5,50]
y2 = [30,12,6,25,70,40,80]
plt.plot(x,y1)
plt.plot(x,y2)
plt.show()
```



Matplotlib (guardar)

- Podemos guardar um gráfico para um ficheiro
- Podemos escolher extensão (PNG, JPG, PDF, etc)

```
x = [0,1,4,5,10,12,20]
y = [15,0,20,100,35,5,50]
plt.plot(x,y)
# abre uma GUI
#plt.show()
# guarda para ficheiro
plt.savefig('grafico.png')
```

Matplotlib = magia?

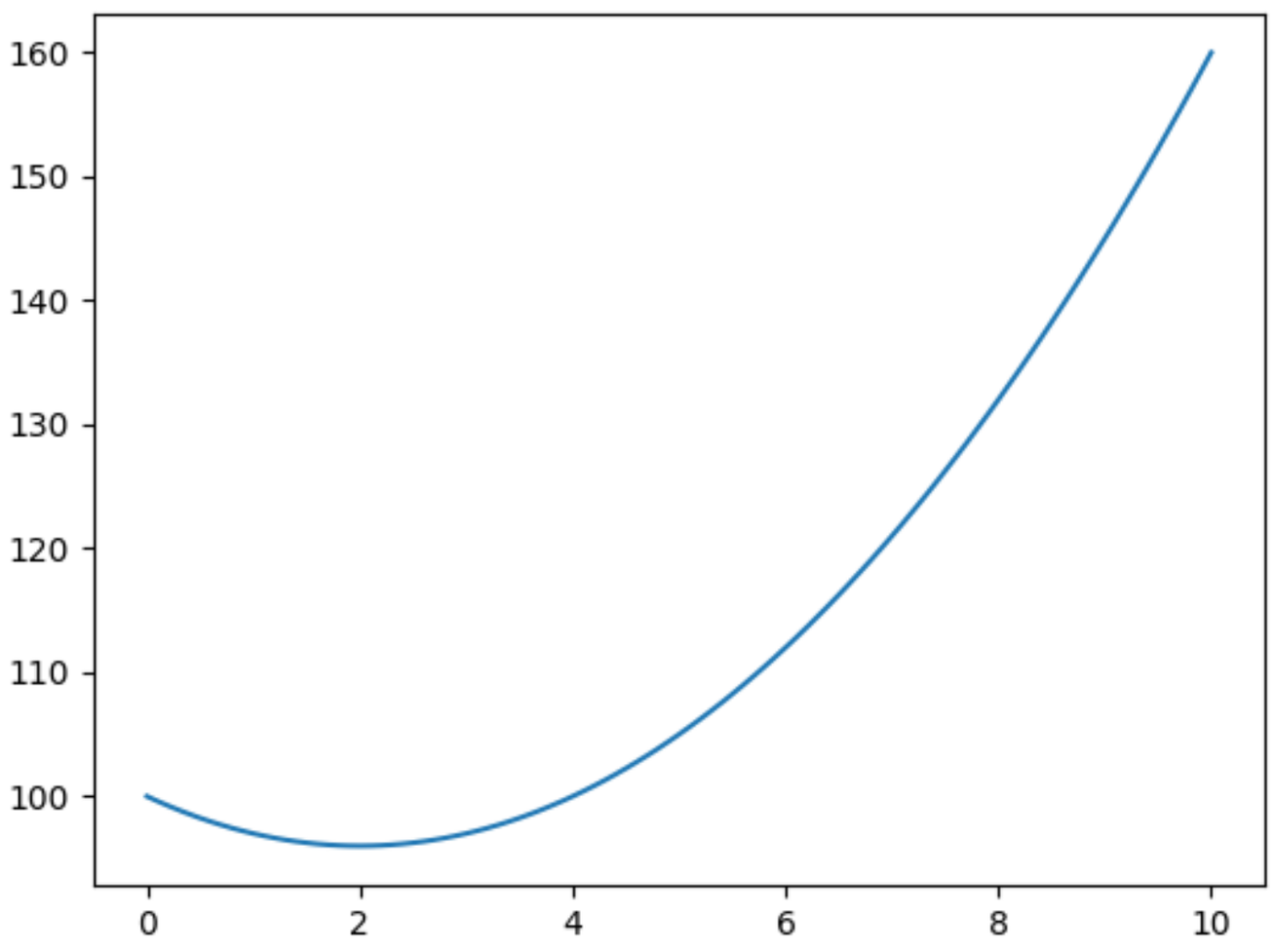
- Como é que a função *show()* sabe o que desenhar?
- **Cuidado**: o módulo tem estado global interno….
- Regra: apenas um *show()* por execução do programa
  - Caso contrário comportamento imprevisível

```
import matplotlib.pyplot as plt
x = [0,1,4,5,10,12,20]
y1 = [15,0,20,100,35,5,50]
y2 = [30,12,6,25,70,40,80]
plt.plot(x,y1)
plt.show()
plt.plot(x,y2)
plt.show()
```

Matplotlib + NumPy

- Podemos utilizar vetores *numpy* como sequências

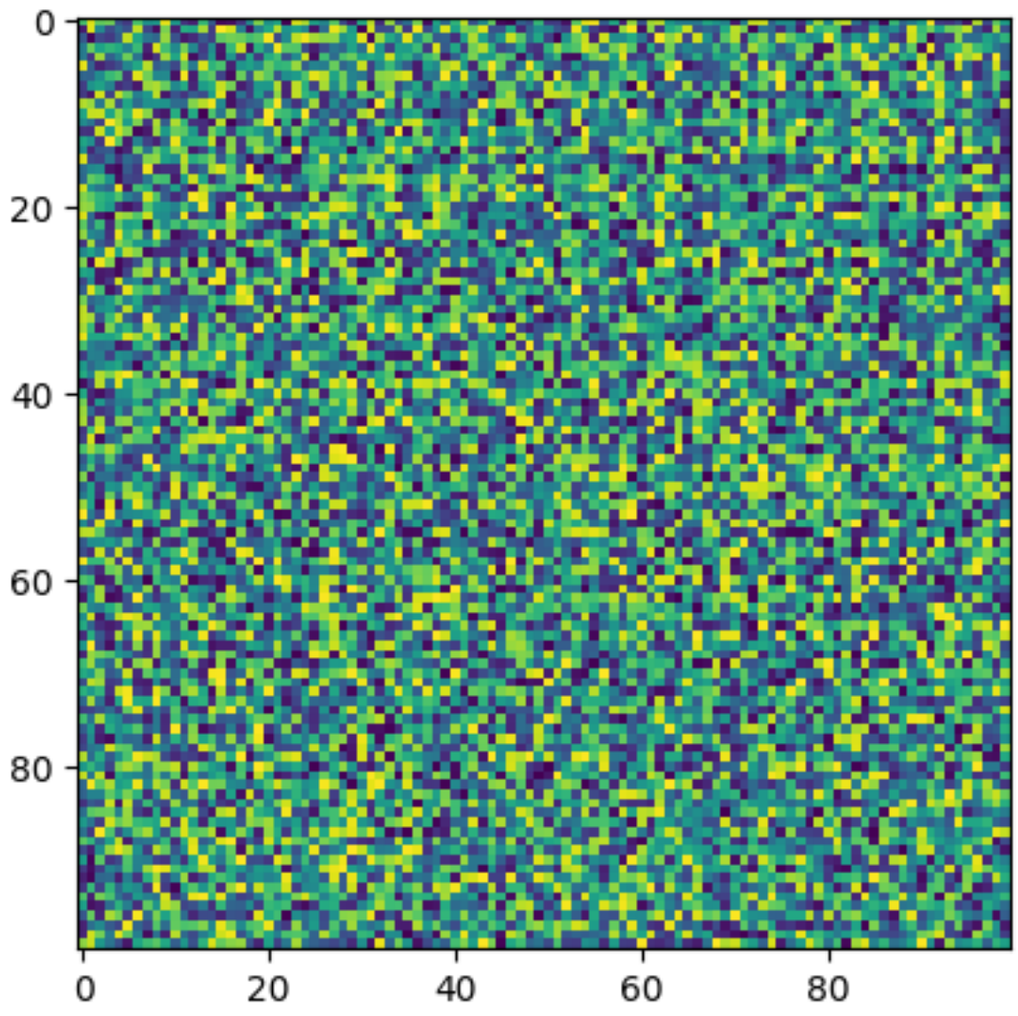
```
import numpy as np
import matplotlib.pyplot
as plt
# 101 values, equally
spaced in [0,10]
x = np.linspace(0,10,101)
# y as a function of x
y = 100 - 4*x + x**2
plt.plot(x,y)
plt.show()
```



Matplotlib + NumPy

- Podemos visualizar matrizes *numpy* como imagens
  - Mapa de cores escolhido automaticamente

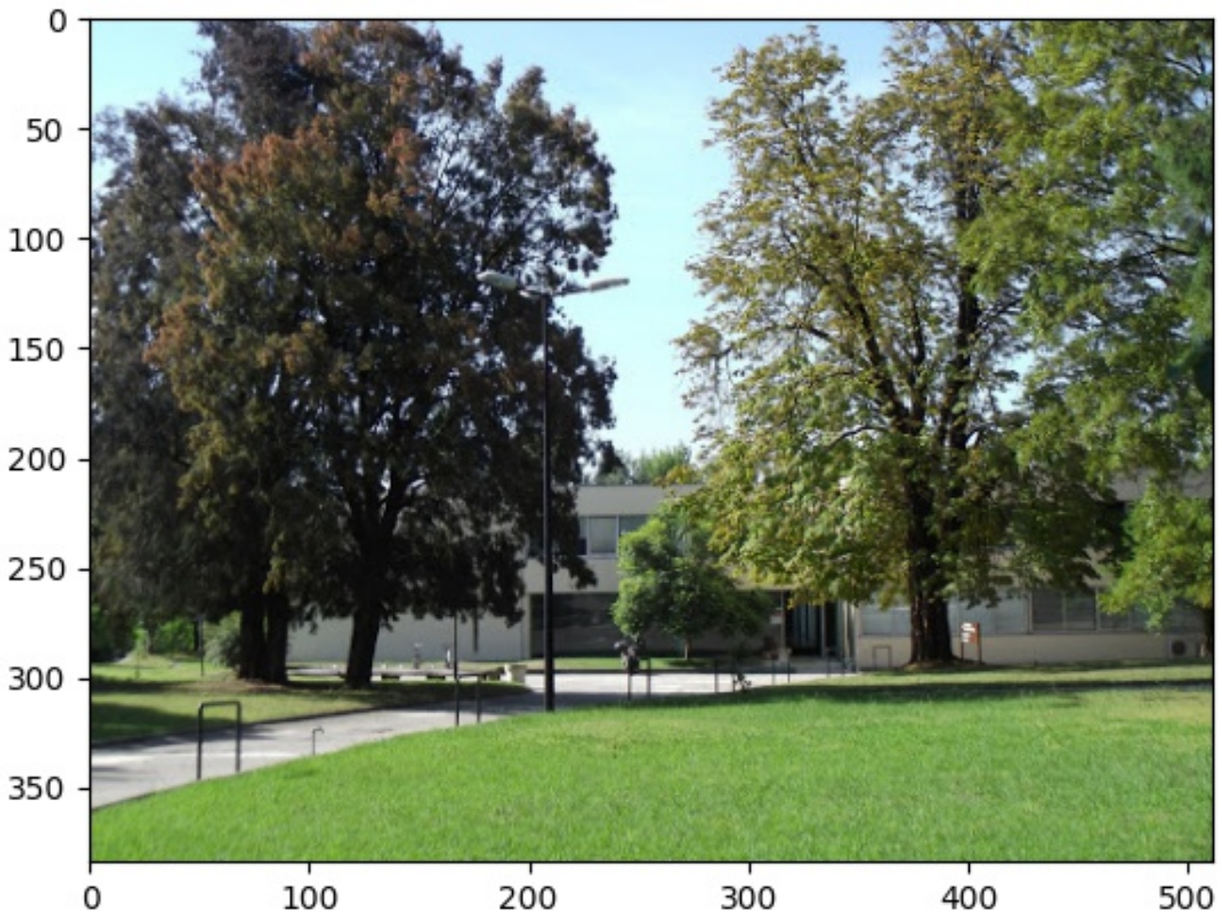
```
import numpy as np
import matplotlib.pyplot as plt
x = np.random.rand(100,100)
plt.imshow(x)
plt.show()
```



Matplotlib + NumPy

- Podemos ler/escrever imagens para/de matrizes *numpy*
- Podemos transformar a imagem manipulando a matriz
  - E.g., aumentar a imagem 2x

```
dcc=plt.imread("dcc.jpg")
print(dcc.shape) #(384, 512, 3)
x,y,z = dcc.shape

dcc_big = np.empty((x*2, y*2, z),
dtype=dcc.dtype)
for i,row in enumerate(dcc):
    for j,pixel in
enumerate(row):
        dcc_big[i*2-1:i*2+1,j
*2-1:j*2+1] = pixel

plt.imshow(dcc_big)
plt.show()
```



Matplotlib + NumPy

- Podemos ler/escrever imagens para/de matrizes *numpy*
- Podemos transformar a imagem manipulando a matriz
  - E.g., acrescentando “blur”

```
dcc=plt.imread("dcc.jpg")
x,y,z = dcc.shape
dcc_blur =
np.empty((x,y,z),dtype=dcc.dtype)
blur = 2
for i in range(x):
    for j in range(y):
        pixes = dcc[max(0,i-
blur):i+blur+1,max(0,j-
blur):j+blur+1]
        dcc_blur[i, j] =
pixes.mean(axis=(0,1))
plt.imshow(dcc_blur)
plt.show()
```

Matplotlib + NumPy

- Podemos ler/escrever imagens para/de matrizes *numpy*
- Podemos transformar a imagem manipulando a matriz
  - E.g., convertendo em “escala de cinza”

```
dcc=plt.imread("dcc.jpg")
dcc_gray = dcc.mean(axis=2)
plt.imshow(dcc_gray,cmap='gray')
plt.axis('off')
plt.savefig('gray.png')
```

PIL + NumPy

- **Nota:** O *matplotlib* não é a biblioteca ideal para escrever imagens (resultado é um gráfico com barras, margens, etc)

- Num cenário mais real podemos usar uma biblioteca de processamento de imagens como o *PIL*

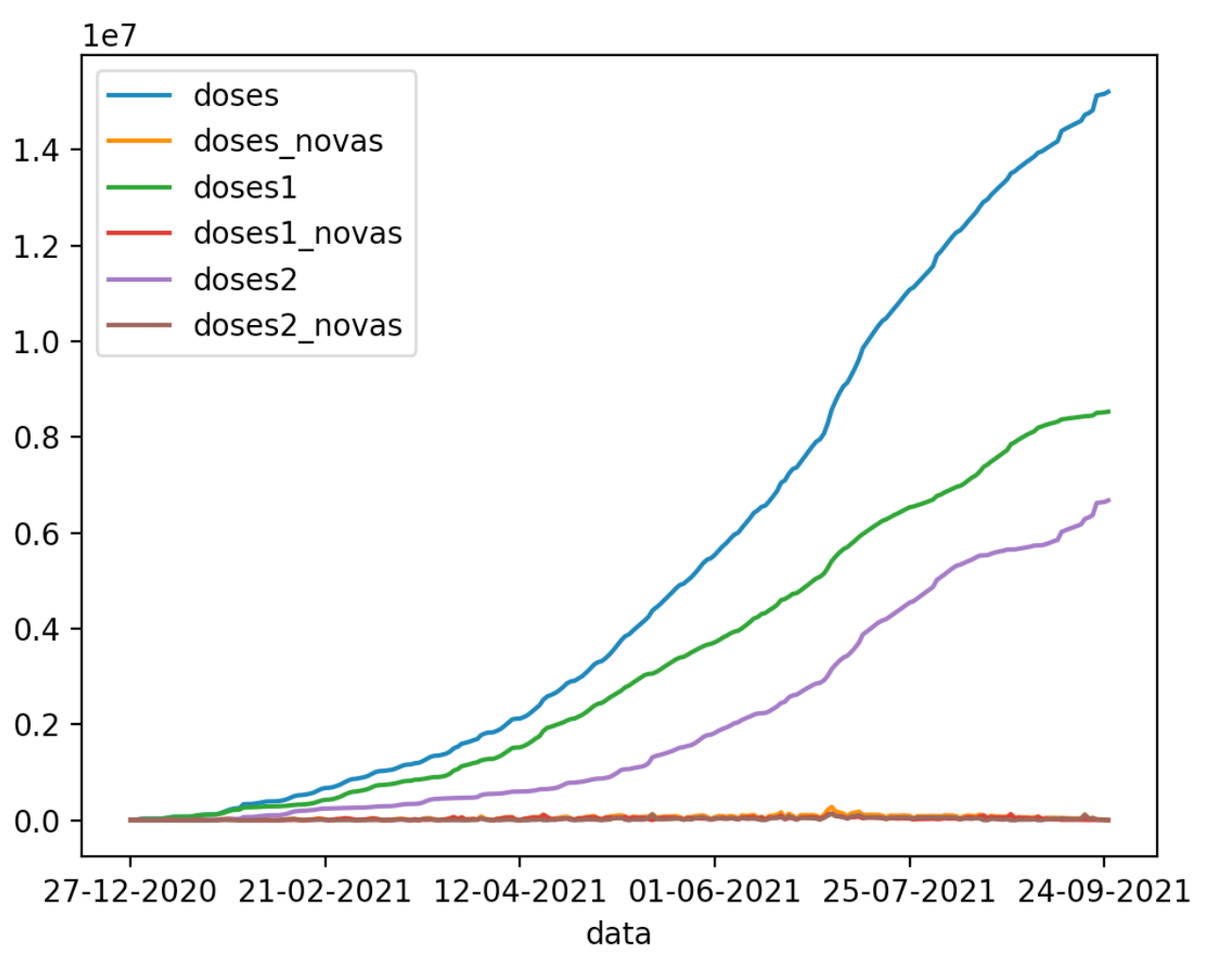
- **Mesma ideia:** manipular arrays *numpy*

```
import numpy as np
from numpy import asarray
from PIL import Image

dcc=asarray(Image.open("../../dados/dcc.jpg"))
x,y,z = dcc.shape
dcc_big = np.empty((x*2, y*2, z), dtype=dcc.dtype)
for i,row in enumerate(dcc):
     for j,pixel in enumerate(row):
          dcc_big[i*2-1:i*2+1,j*2-1:j*2+1] = pixel
Image.fromarray(dcc_big).save('dcc2x.jpg')
```

Matplotlib + Pandas

- Podemos visualizar facilmente um *DataFrame*
  - Índices definem o eixo dos X
  - Cada coluna é uma curva no eixo dos Y
- E.g., dados de vacinação contra a COVID-19 publicados pela DGS e disponíveis aqui

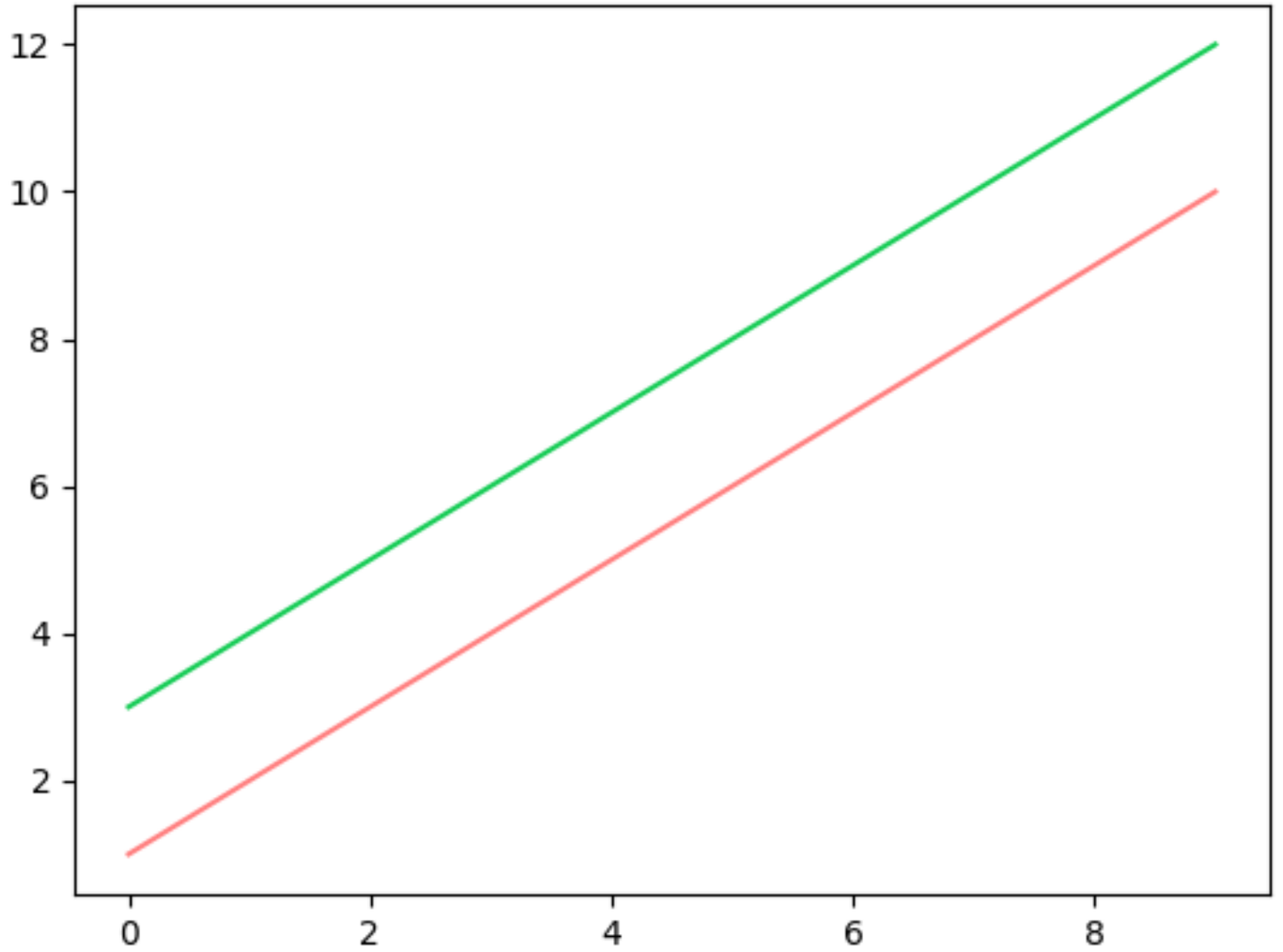
```
vacinas =
pd.read_csv('vacinas.csv',in
dex_col='data')
vacinas = vacinas[[col for
col in vacinas.columns if
col.startswith('doses')]]
vacinas = vacinas.dropna()

vacinas.plot()
plt.show()
```



Matplotlib (cores)

- Podemos controlar as cores de cada curva
- Lista de nomes de cores

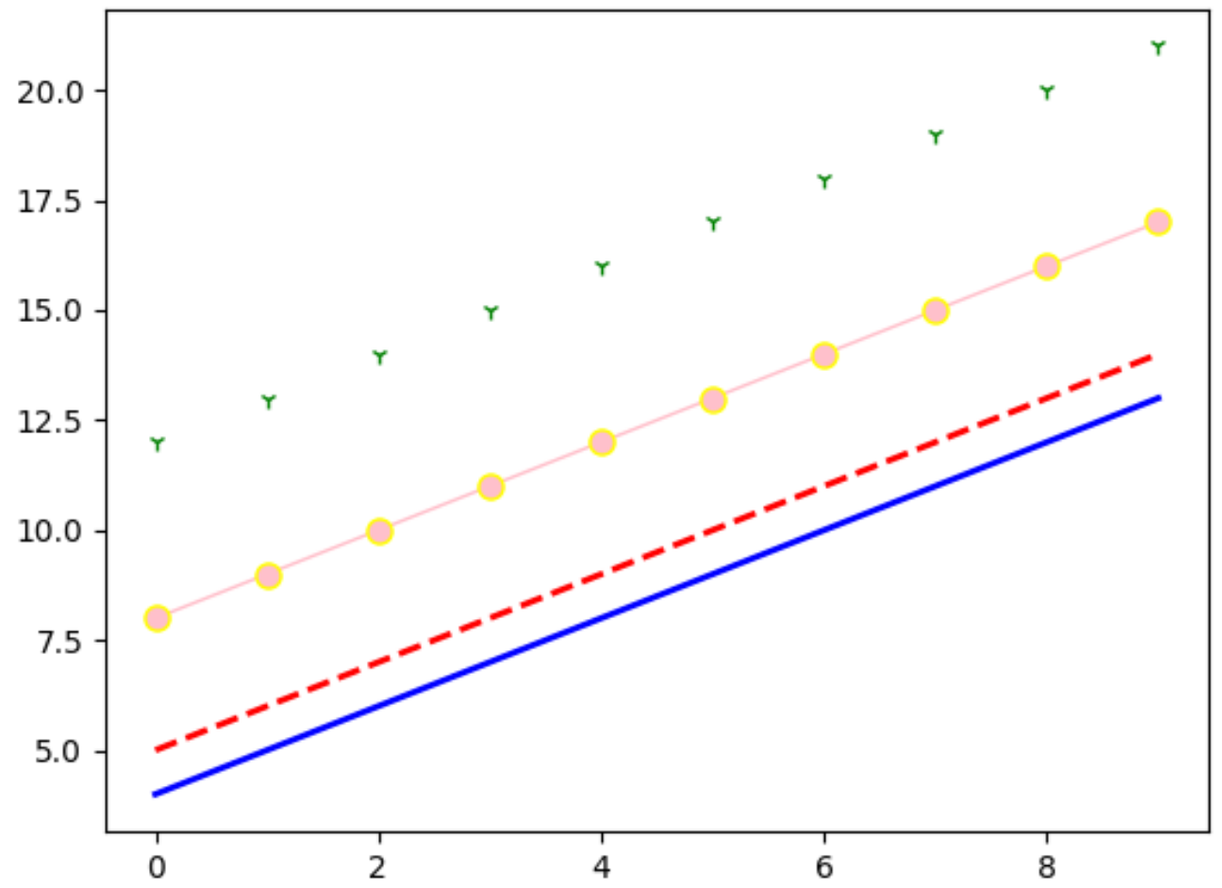
```
x = np.arange(10)
# half-transparant red
plt.plot(x, x+1, color="red",
alpha=0.5)
# RGB hex code for green
plt.plot(x, x+3, color="#15cc55")
plt.show()
```



Matplotlib (estilo)

- Podemos controlar a grossura, estilo e marcadores de cada linha
  - Estilos de linha: ‘-‘, ‘–’, ‘-.’, ‘:’, ‘steps’,’none’
  - Marcadores: ‘+’, ‘o’, ‘*’, ’s', ',', '.', '1', '2', '3', '4', ...

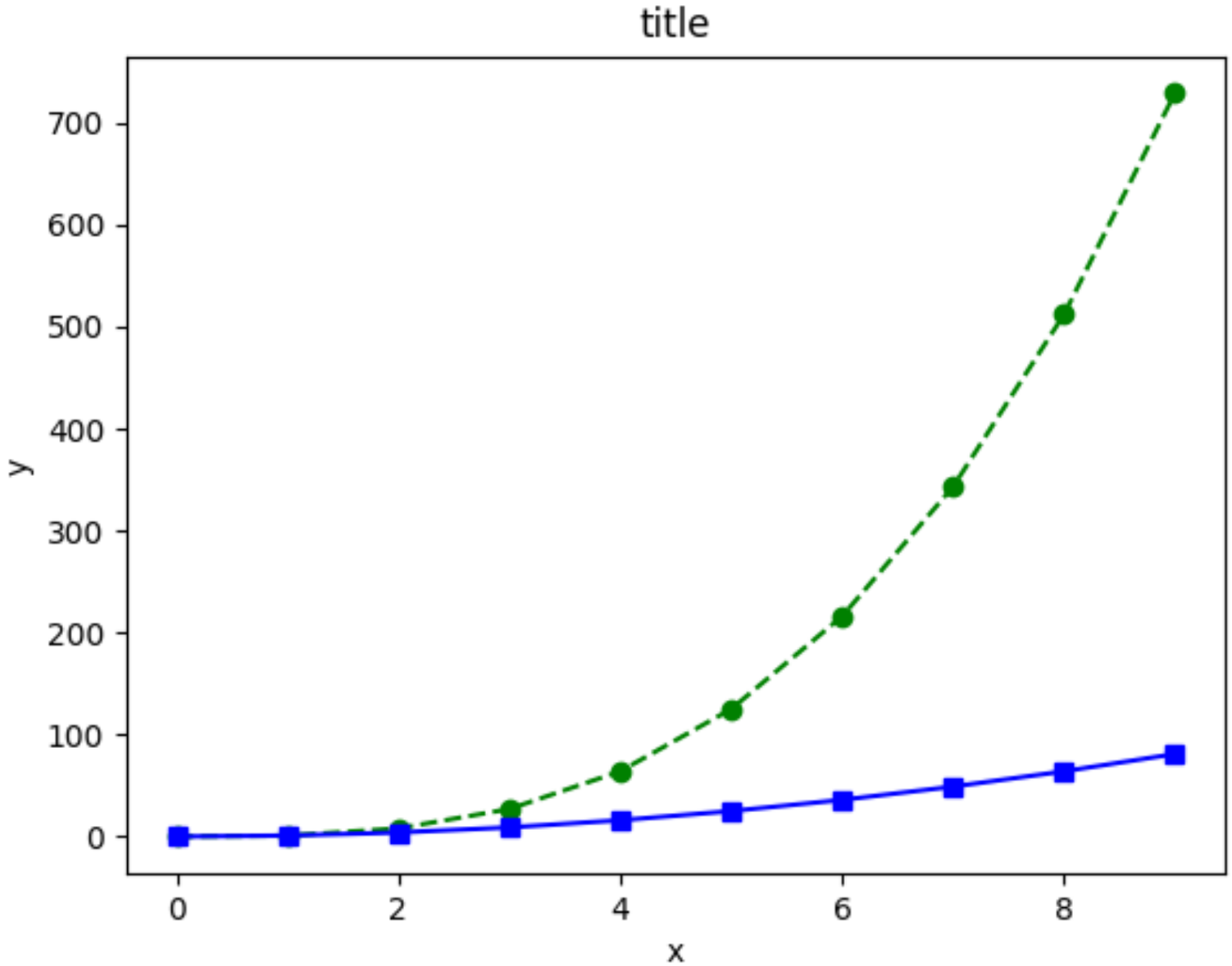
```
x = np.arange(10)
plt.plot(x, x+4, c="blue",
linewidth=2.00)
plt.plot(x, x+5, c="red", lw=2,
linestyle='--')
plt.plot(x, x+8, c="pink", lw=1,
ls='-', marker='o',
markersize=8,markeredgecolor="yellow"
)
plt.plot(x, x+12, c="green",
ls='none', marker='1')
plt.show()
```



Matplotlib (nomes)

- Podemos atribuir um título ao gráfico, e dar nomes aos eixos dos X e dos Y

```
x = np.arange(10)
plt.plot(x, x**3,
'g--',marker='o')
plt.plot(x, x**2,
'b-',marker='s')
plt.title('title')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

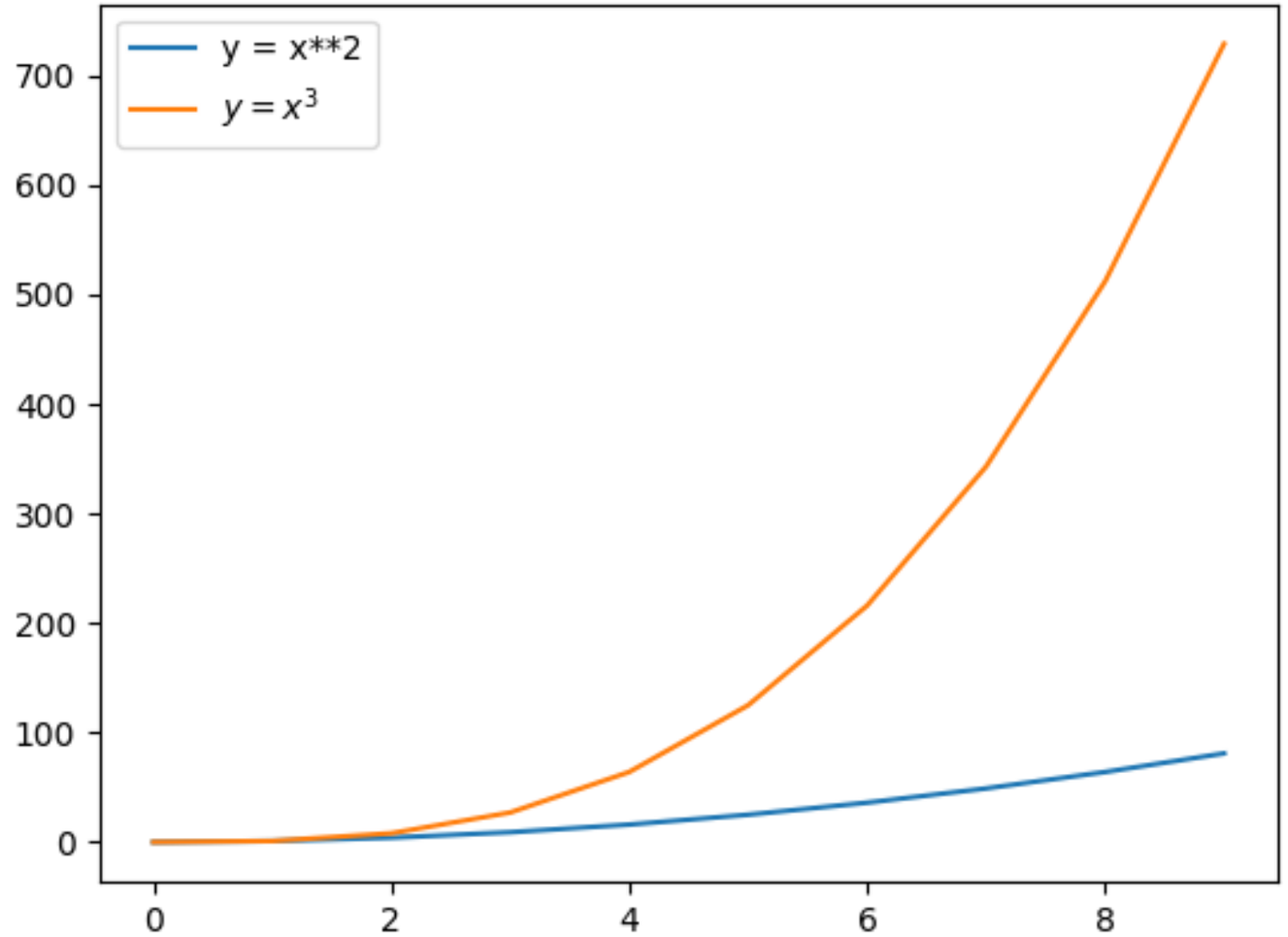


Matplotlib (legendas)

- Podemos definir manualmente uma legenda, definindo uma label por linha e escolhendo o seu posicionamento

- Podemos utilizar notação LaTeX

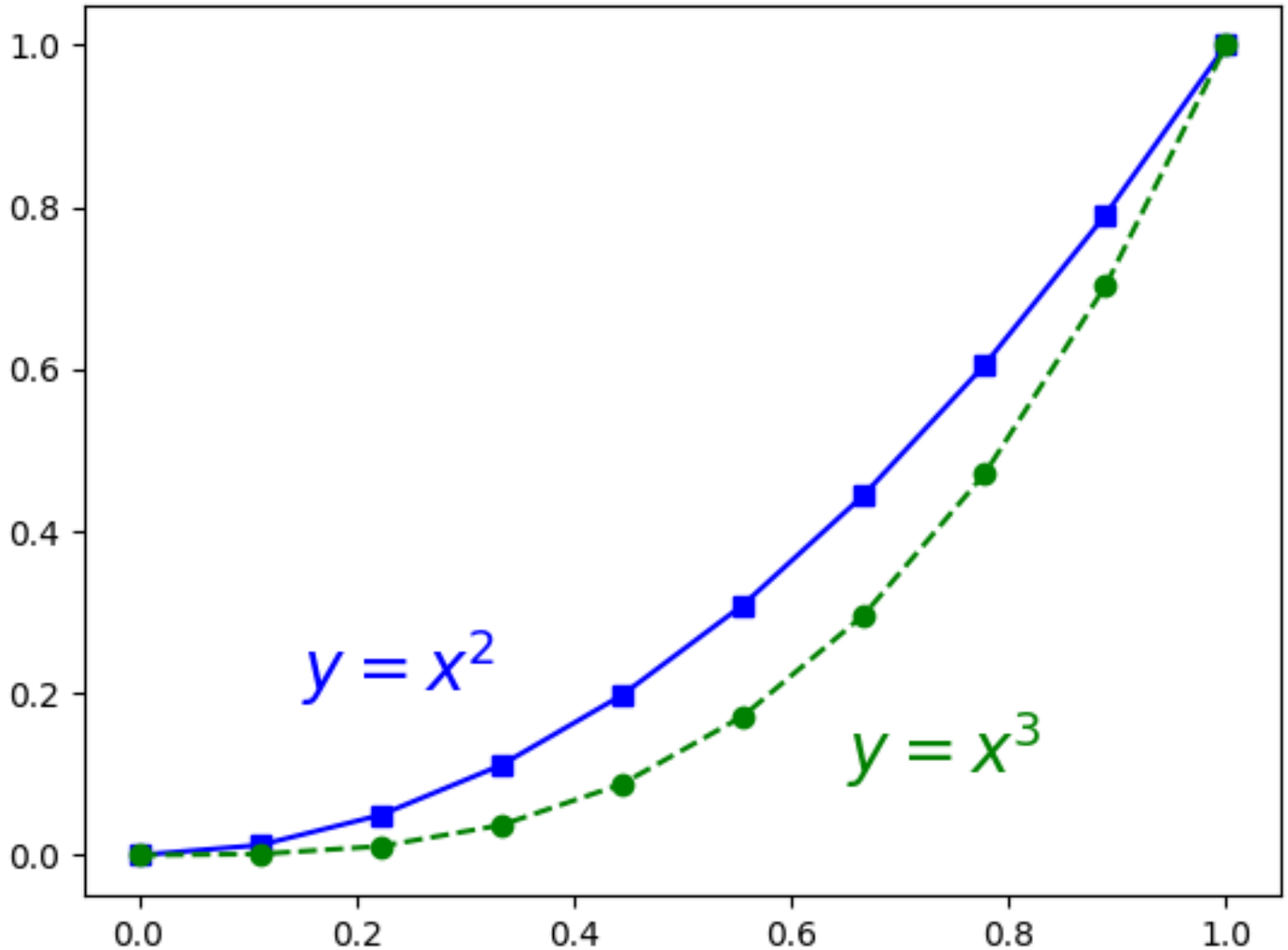
```
x = np.arange(10)
plt.plot(x, x**2,
label="y = x**2")
plt.plot(x, x**3,
label="$y = x^3$")
plt.legend(loc='upper
left')
plt.show()
```



Matplotlib (anotações)

```
x = np.linspace(0., 1., 10)
plt.plot(x, x**2, 'b-',marker='s')
plt.plot(x, x**3, 'g--',marker='o')
plt.text(0.15, 0.2, r"$y=x^2$", fontsize=20, color="blue")
plt.text(0.65, 0.1, r"$y=x^3$", fontsize=20, color="green");
plt.show()
```

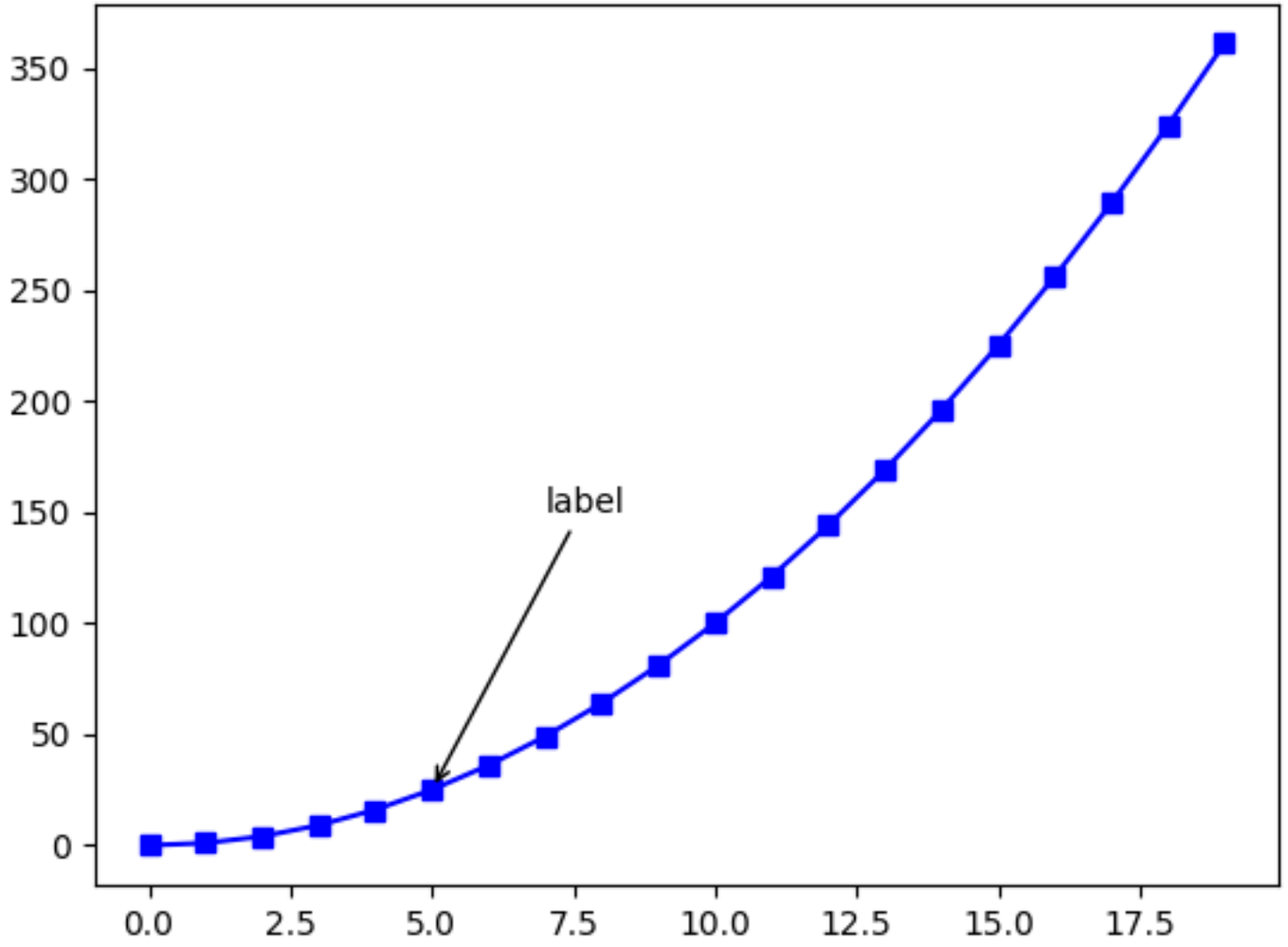
- Podemos definir anotações de texto numa posição (x,y) do gráfico



Matplotlib (anotações)

```
x = np.arange(20)
plt.plot(x, x**2, 'b-',marker='s')
plt.annotate("label",xy=(5,25), xycoords='data',xytext=(7,150),
textcoords='data',arrowprops=dict(arrowstyle="->",connectionstyle="arc3"))
plt.show()
```

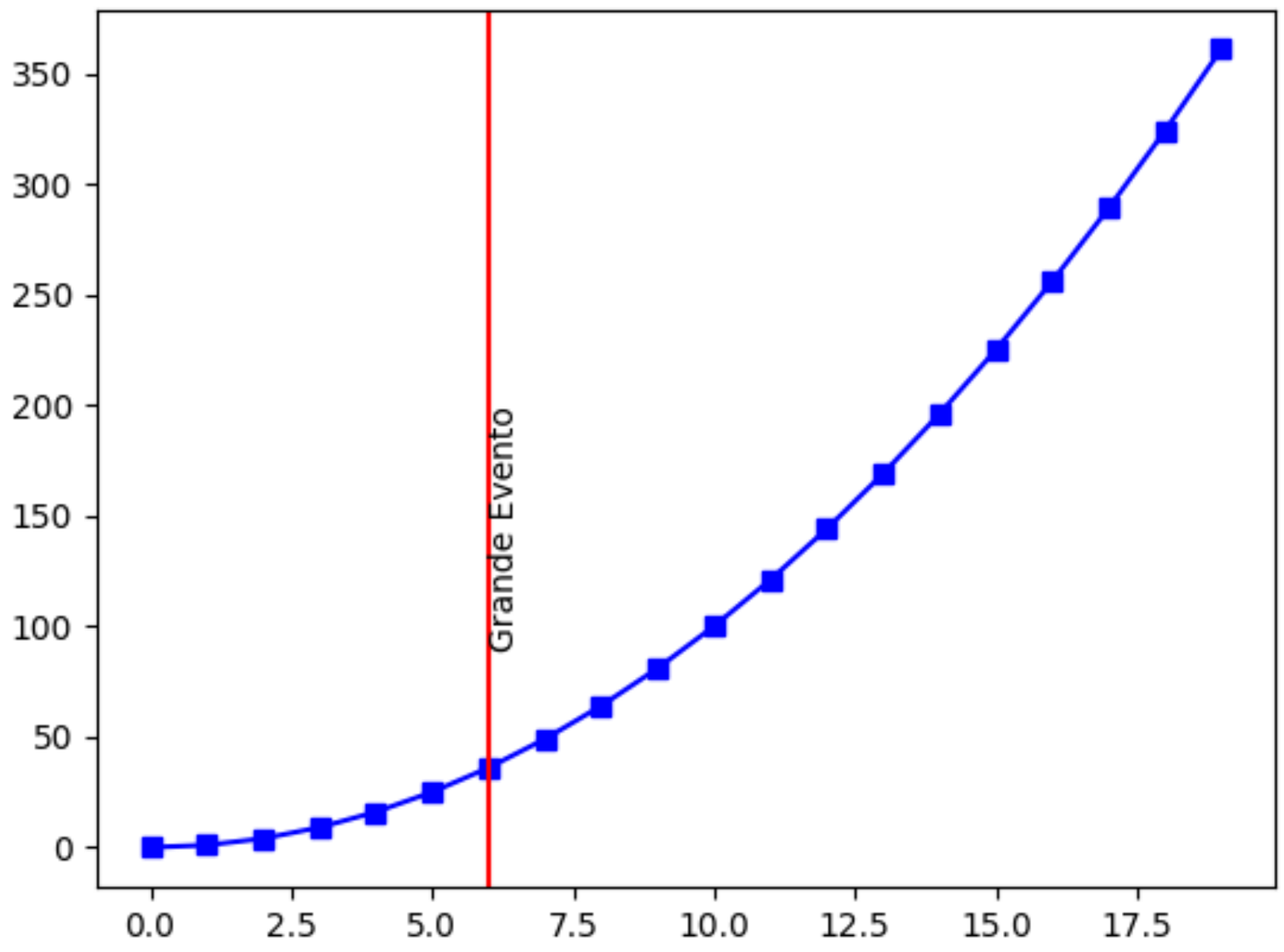
- Podemos definir setas, definindo a posição (x,y) de origem e a posição (x,y) de destino



Matplotlib (anotações)

```
x = np.arange(20)
plt.plot(x, x**2, 'b-',marker='s')
plt.axvline(6, color='red')
plt.text(6,200,'Grande Evento',rotation=90,va='top')
plt.show()
```

- Podemos definir linhas verticais, definindo a posição nos X

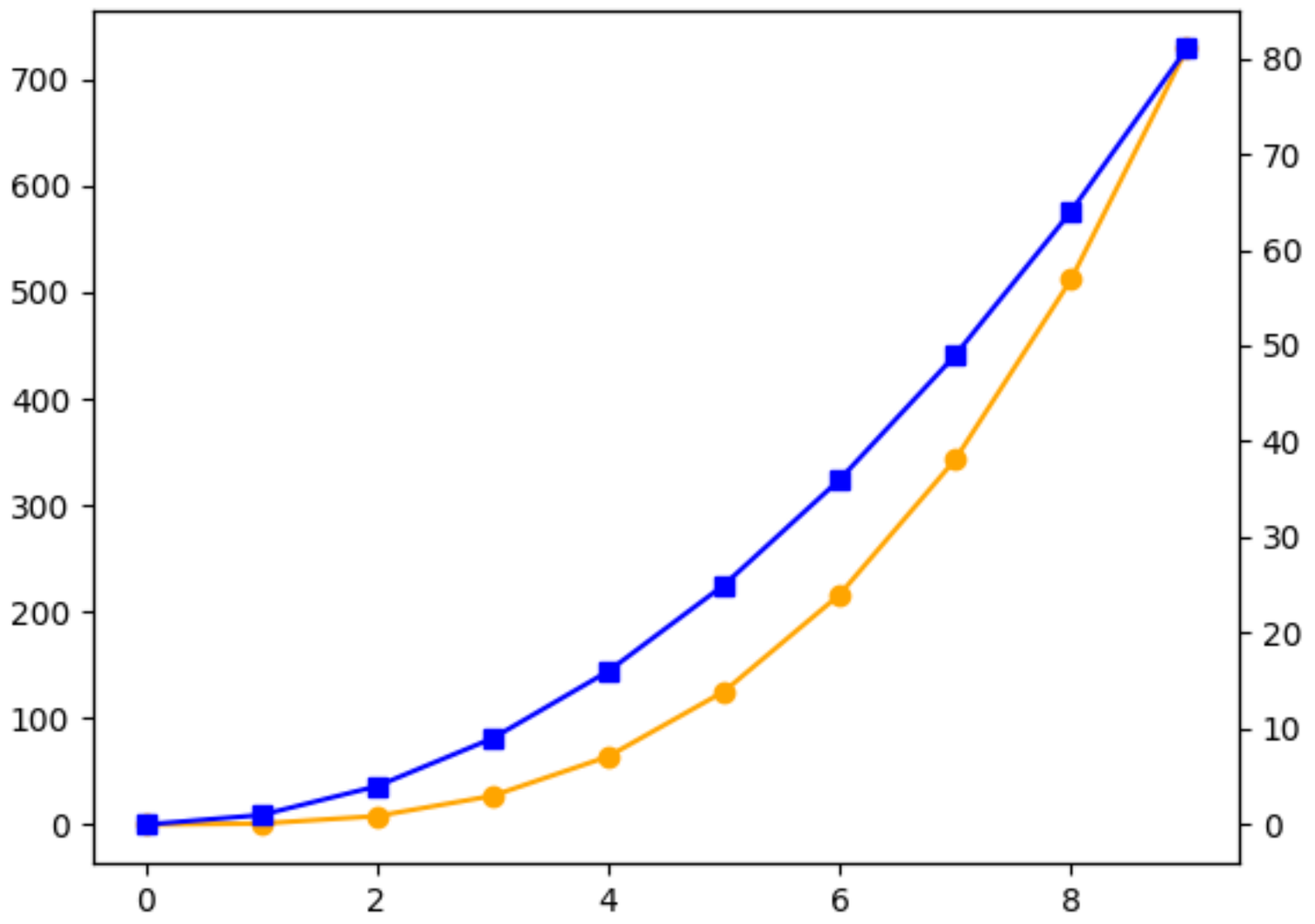


Matplotlib (eixos)

```
_,plt1 = plt.subplots()
plt2 = plt1.twinx()

x = np.arange(10)
plt1.plot(x,x**3,marker='o',c="orange")
plt2.plot(x,x**2,marker='s',c="blue")
plt.show()
```

- Podemos definir eixos dos Y duais com o método *twinx()*

- Note a utilização de *subplots*, cada linha é desenhada num plot diferente

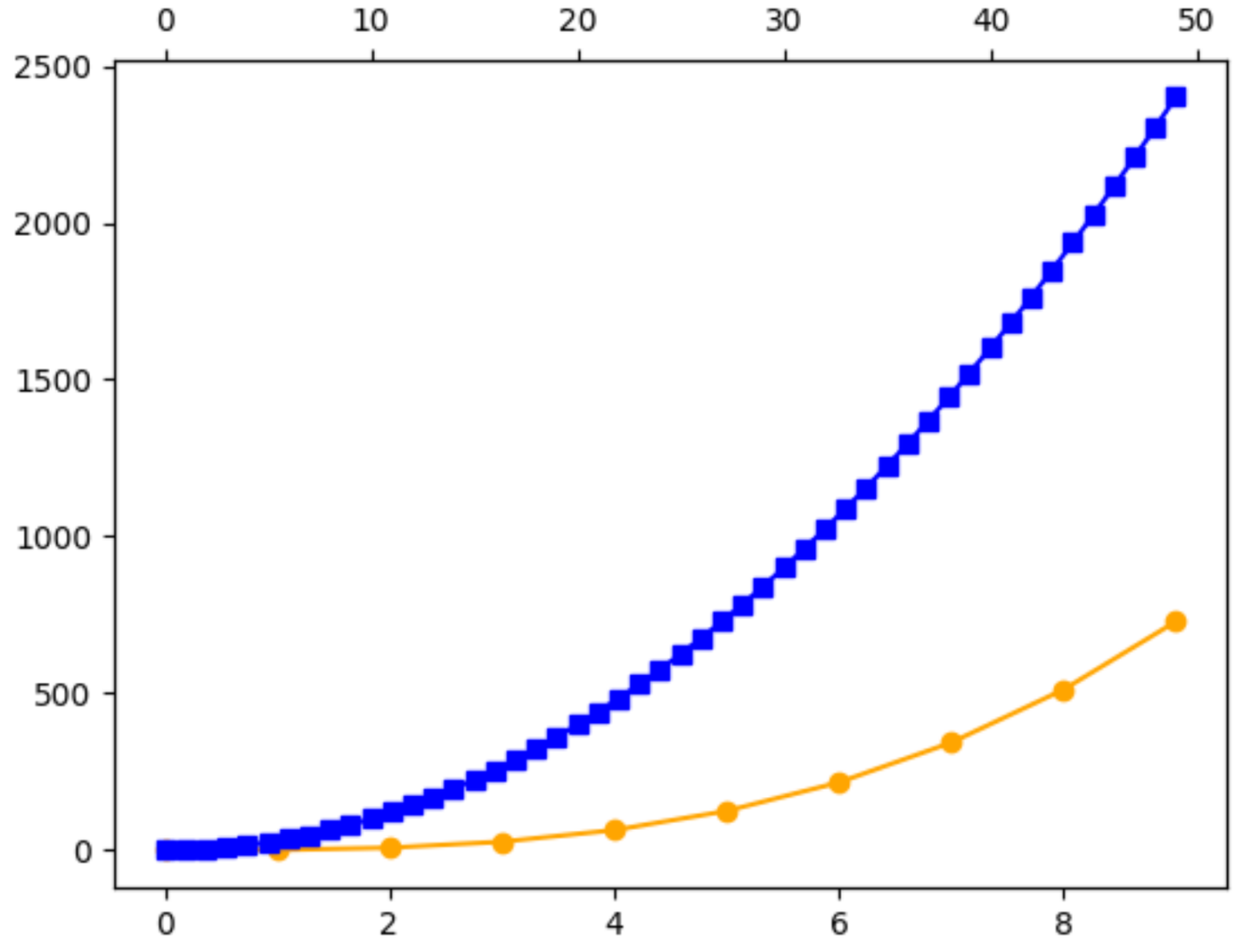


Matplotlib (eixos)

```
_,plt1 = plt.subplots()
plt2 = plt1.twiny()

x1 = np.arange(10)
x2 = np.arange(50)
plt1.plot(x1,x1**3,marker='o',c="orange")
plt2.plot(x2,x2**2,marker='s',c="blue")
plt.show()
```

- Podemos definir eixos dos X duais com o método *twiny()*

- Note a utilização de *subplots*, cada linha é desenhada num plot diferente

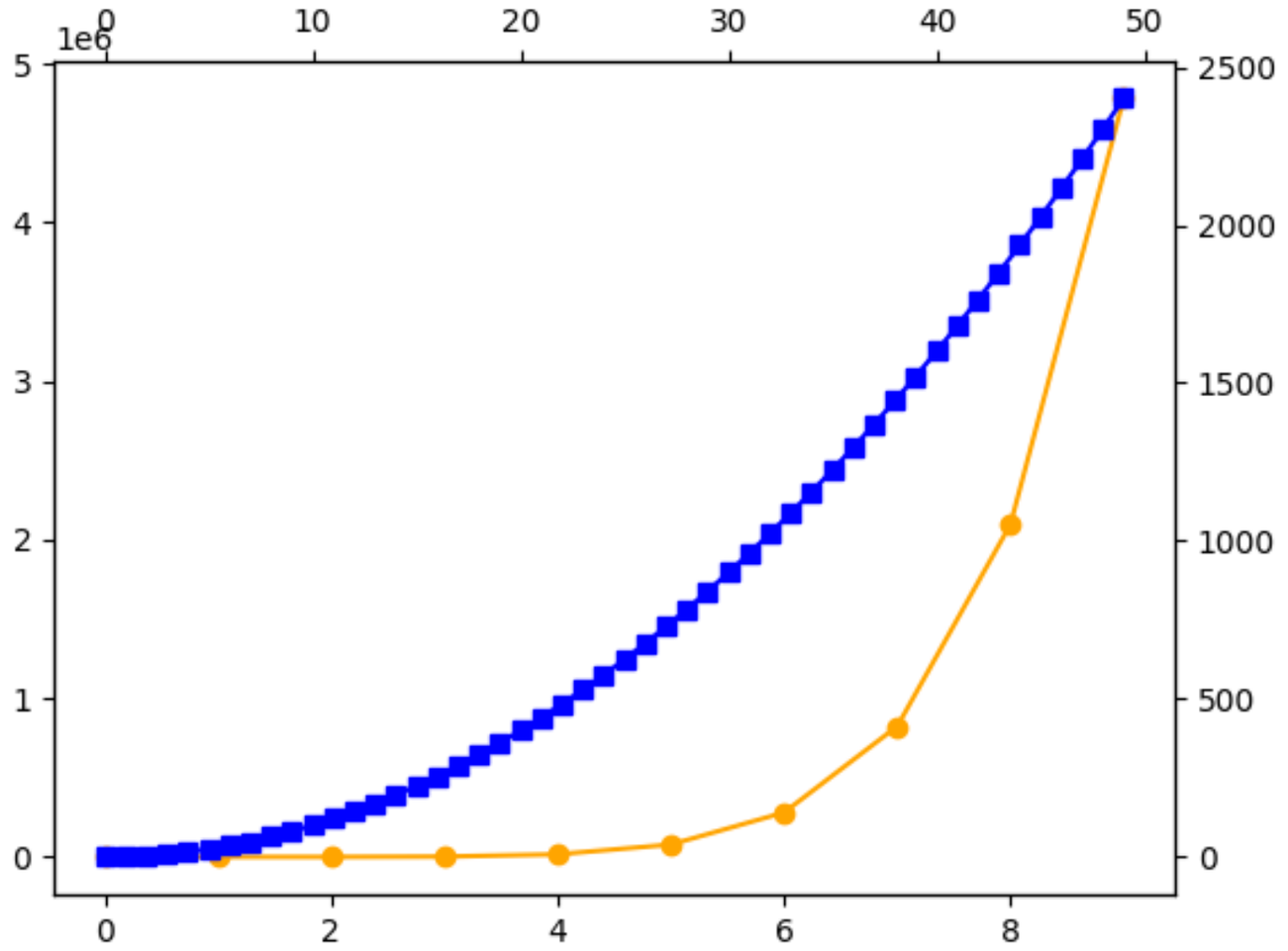


Matplotlib (eixos)

```
_,plt1 = plt.subplots()
plt2 = plt1.twinx()
plt3 = plt2.twiny()

x1 = np.arange(10)
x2 = np.arange(50)
plt1.plot(x1,x1**7,marker='o',c="orange")
plt3.plot(x2,x2**2,marker='s',c="blue")
plt.show()
```

- Podemos definir eixos dos X e dos Y duais

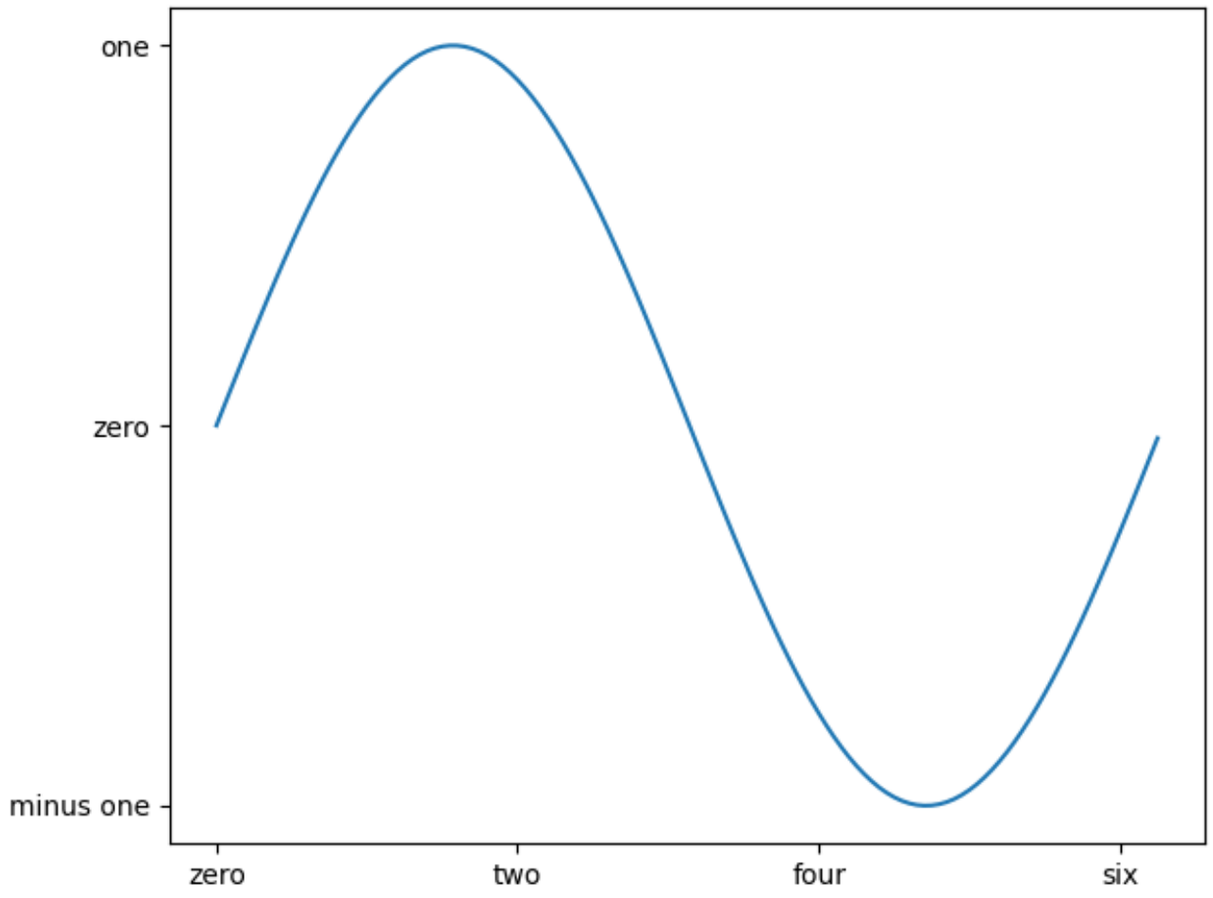
- Note a utilização de *subplots*, cada linha é desenhada num plot diferente



Matplotlib (eixos)

```
x = np.arange(0, math.pi*2, 0.05)
y = np.sin(x)

plt.plot(x, y)
plt.xticks([0,2,4,6],['zero','two','four','six'])
plt.yticks([-1,0,1],['minus one','zero','one'])
plt.tight_layout()
plt.show()
```

- Podemos controlar os marcadores em cada eixo
- Podemos controlar os nomes de cada marcador

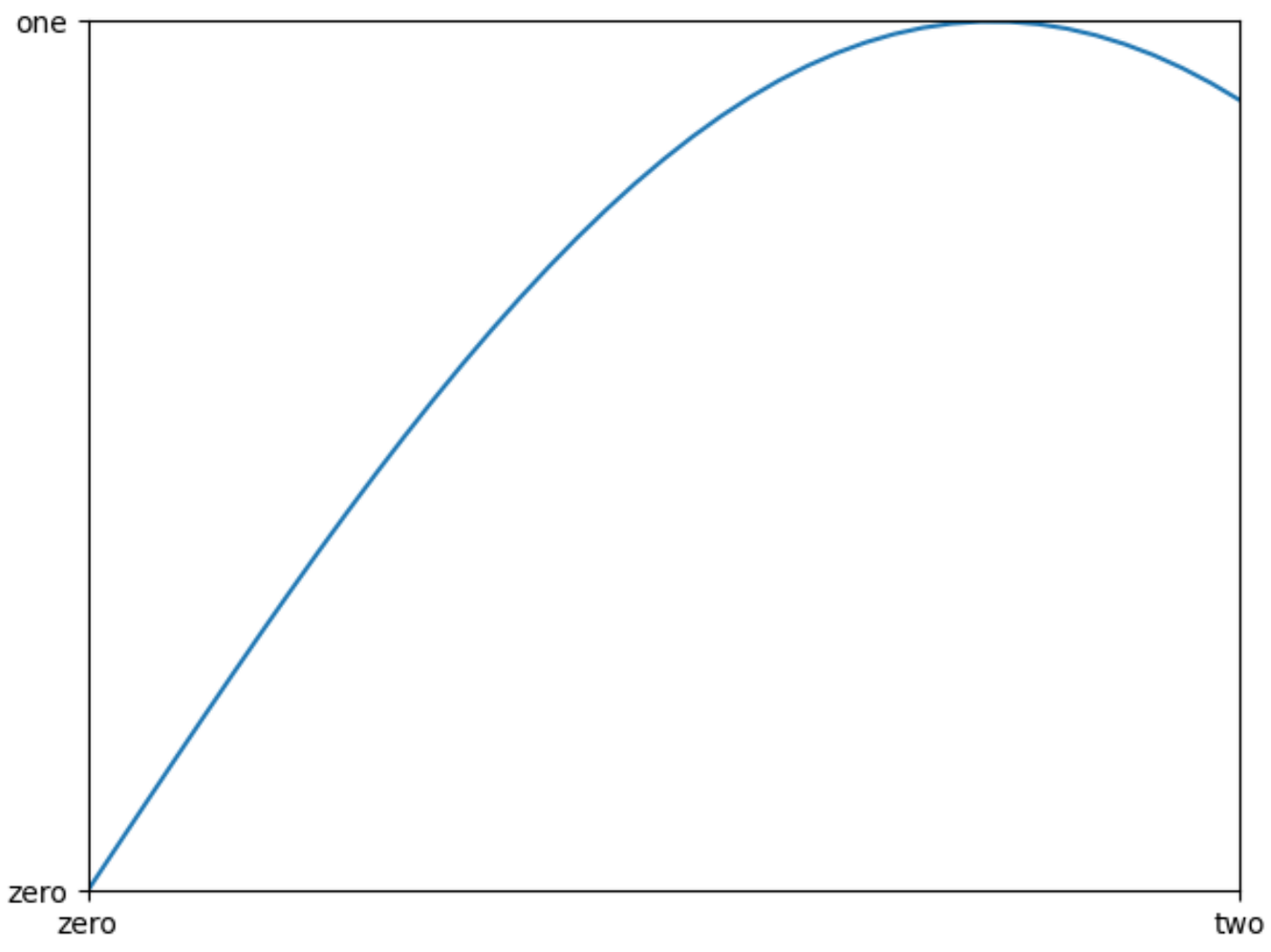


Matplotlib (eixos)

```
x = np.arange(0, math.pi*2, 0.05)
y = np.sin(x)

plt.plot(x, y)
plt.xticks([0,2,4,6],['zero','two','four','six'])
plt.yticks([-1,0,1],['minus one','zero','one'])
plt.xlim(0,2)
plt.ylim(0,1)
plt.tight_layout()
plt.show()
```

- Podemos controlar os limites do gráfico em cada eixo

- E.g., zoom in ou zoom out



Matplotlib (eixos)

```
x = np.arange(0, math.pi*2, 0.05)
y = np.sin(x)

plt.plot(x,y)
plt.xticks([0,2,4,6],['zero','two','four','six'])
plt.yticks([-1,0,1],['minus one','zero','one'])
plt.tick_params(axis='x',bottom=False,labelbottom=Fal
se,top=True,labeltop=True)
plt.tick_params(axis='y',right=True,labelright=True)
plt.tight_layout()
plt.show()
```

- Podemos controlar eixos manualmente