

# Programação II

+

# Estruturas de Dados para Bioinformática

Hugo Pacheco

DCC/FCUP  
23/24

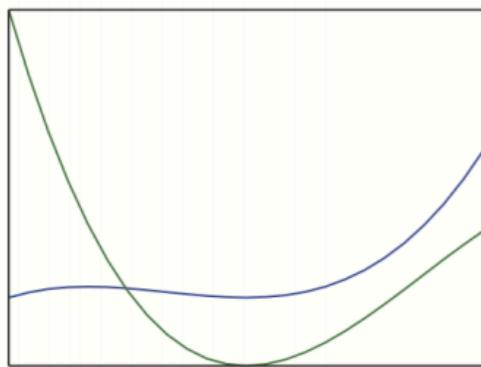
# Gráficos estáticos (matplotlib)

# Gráficos

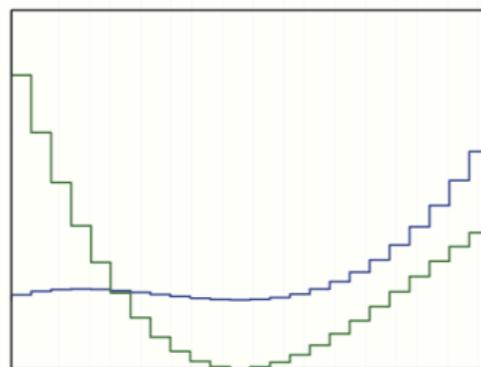
- Última aula
  - Introdução a gráficos no *matplotlib*
  - Tipos de gráficos *plot* e *imshow*
  - Customização de gráficos no *matplotlib*
- Esta aula:
  - Outros tipos de gráficos
  - Exemplos práticos

# Tipos de gráficos\*

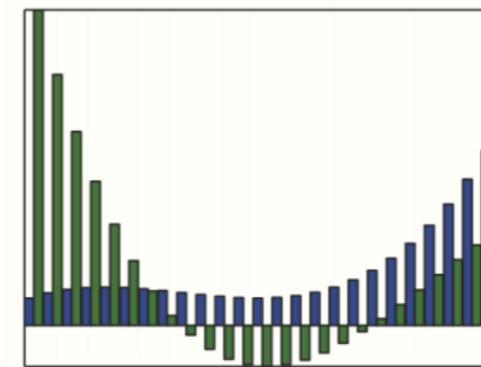
Axes.plot



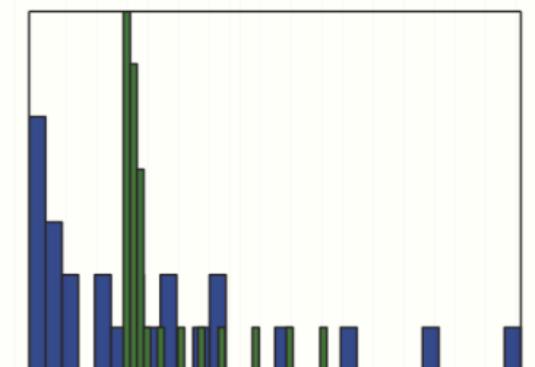
Axes.step



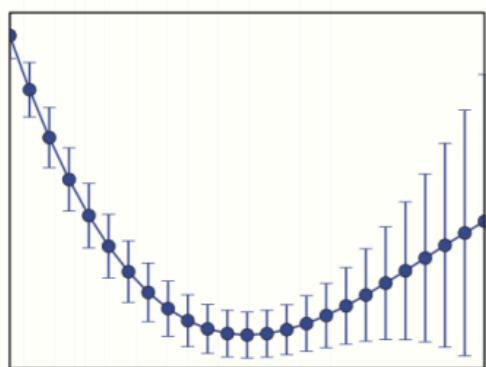
Axes.bar



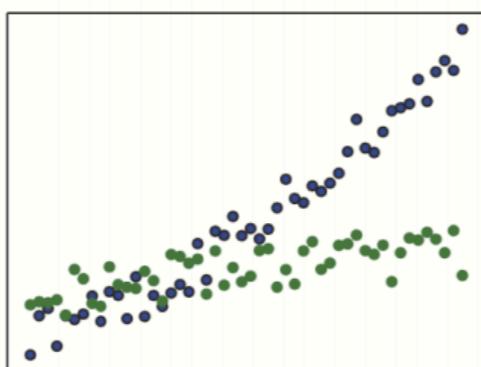
Axes.hist



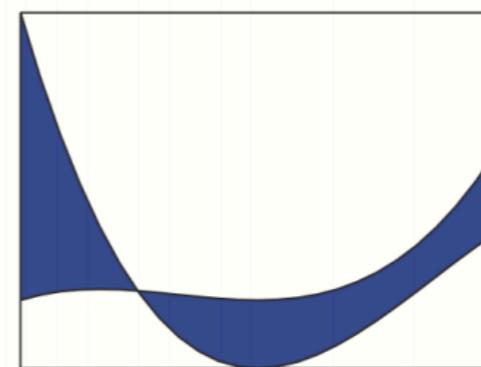
Axes.errorbar



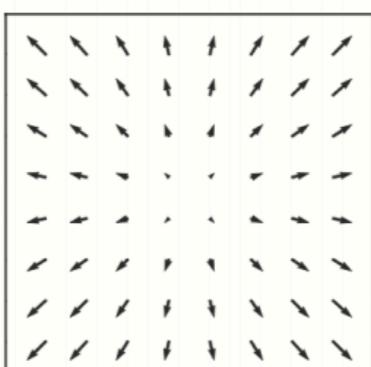
Axes.scatter



Axes.fill\_between



Axes.quiver

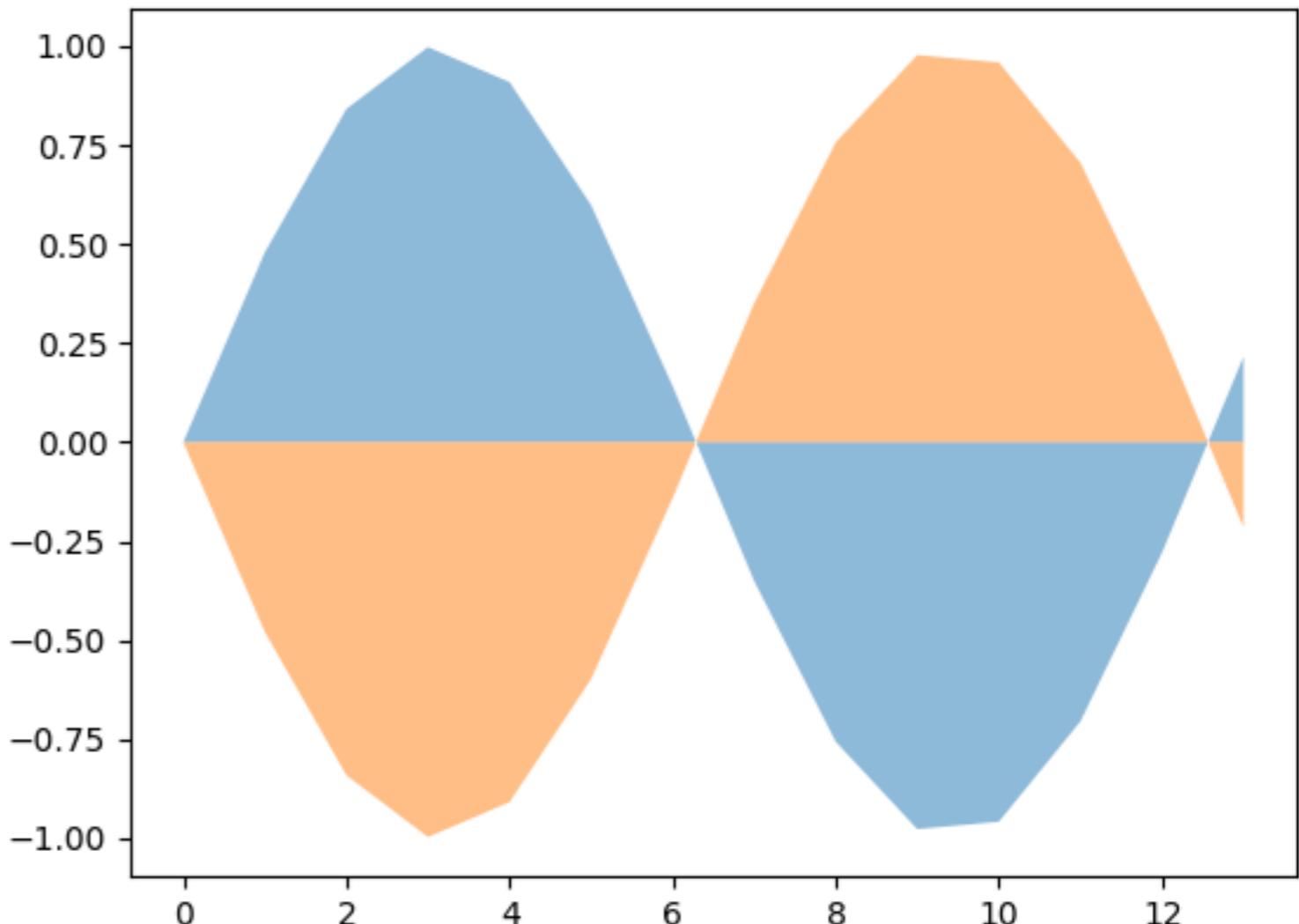


# Matplotlib (fill)

```
x = np.arange(14)
y1 = np.sin(x / 2)
y2 = np.cos(math.pi/2 + x / 2)

plt.fill_between(x,y1,alpha=0.5) # blue
plt.fill_between(x,y2,alpha=0.5) # orange
plt.show()
```

- Preencher a área entre o eixo dos X e cada curva

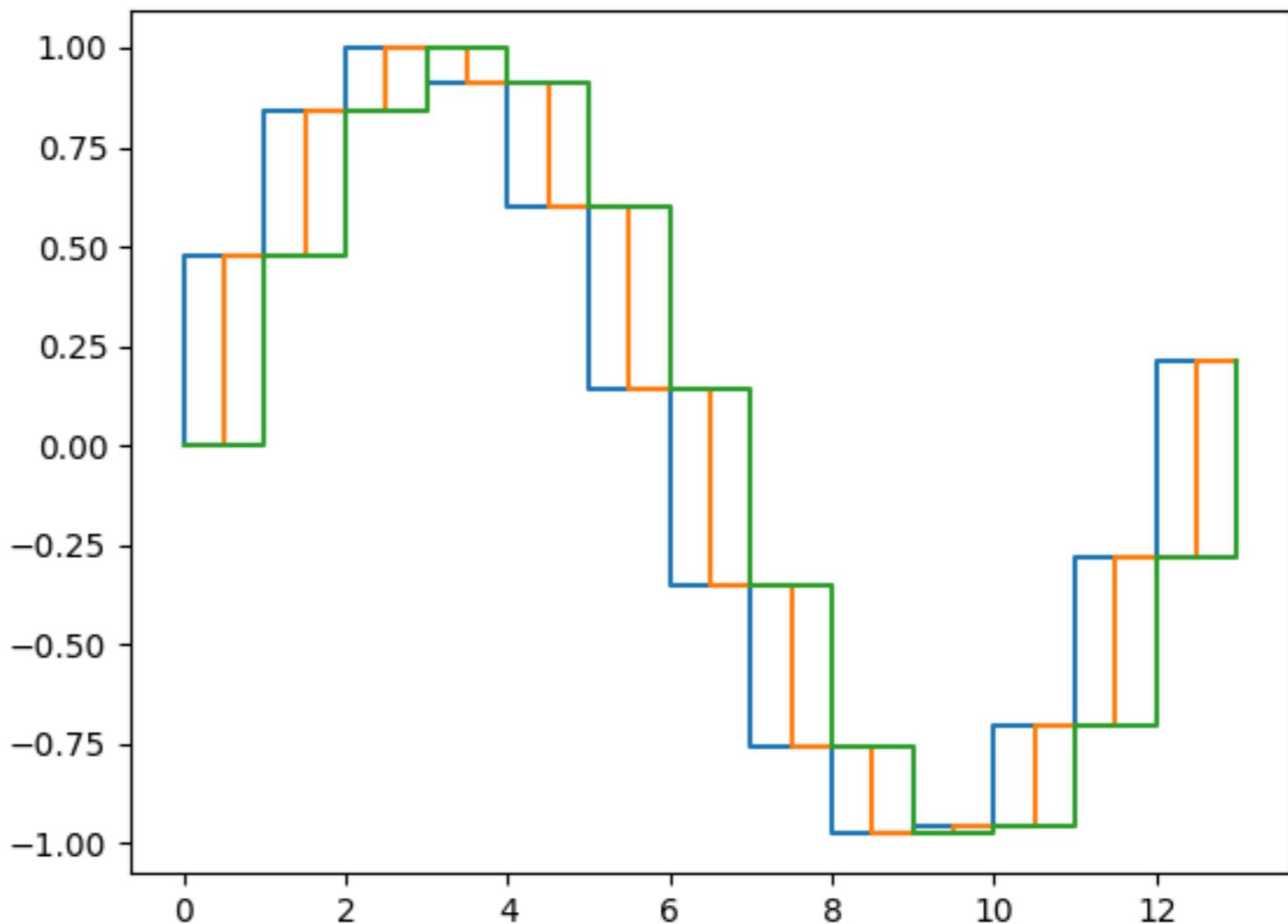


# Matplotlib (step)

- Desenhar linhas retas em vez de curvas
- Podemos controlar a posição de cada ponto em relação ao nível

```
x = np.arange(14)
y = np.sin(x / 2)

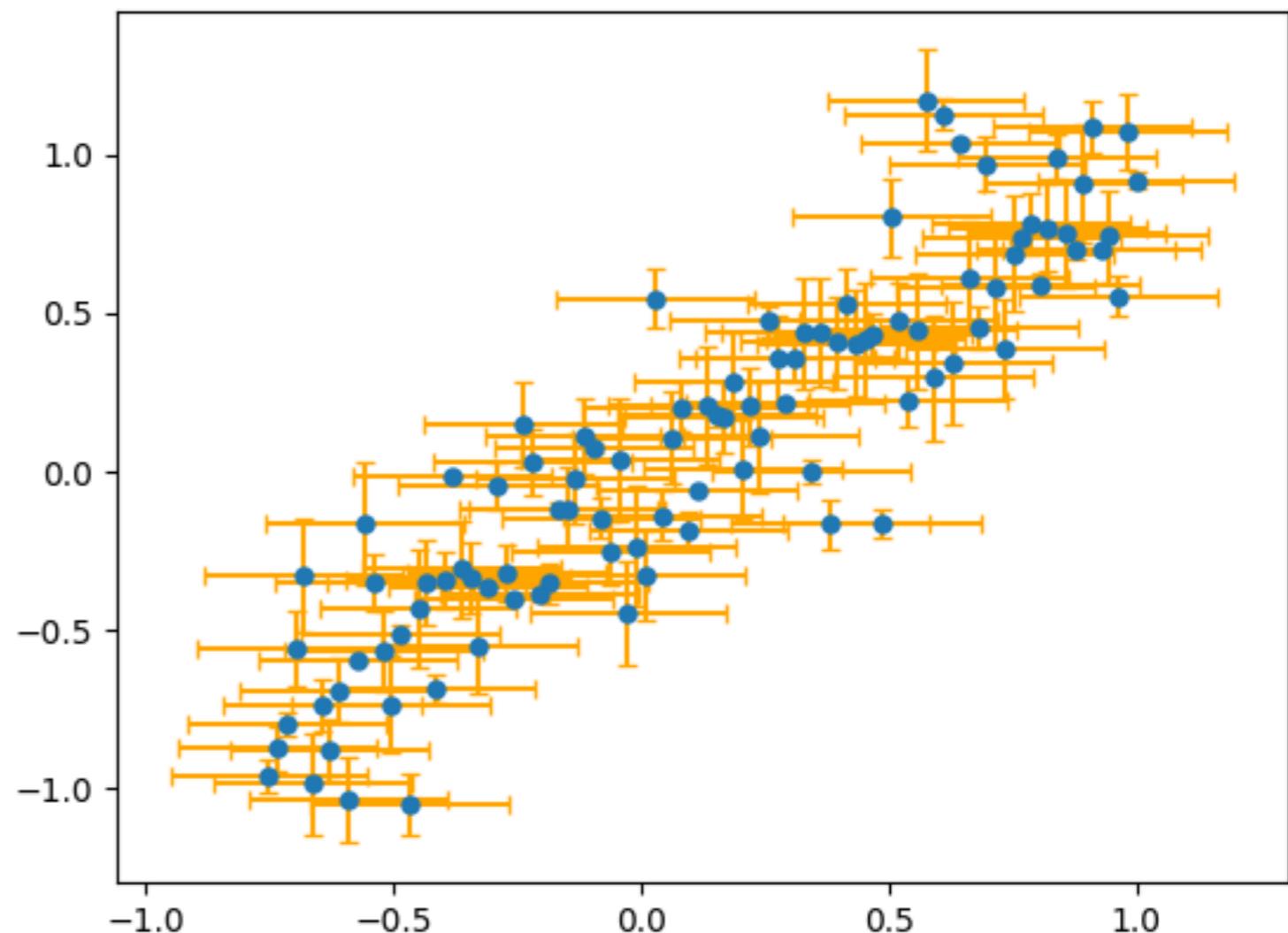
plt.step(x, y) # blue
plt.step(x, y,
         where='mid') # orange
plt.step(x, y,
         where='post') # green
plt.show()
```



# Matplotlib (errorbar)

- Atribuir uma margem de erro a cada dimensão
- Podemos controlar desenho dos intervalos

```
xx = np.linspace(-0.75, 1., 100)
yy = xx +
0.25*np.random.randn(len(xx))
sigma_y = np.random.uniform(0, 0.2,
len(xx))
plt.errorbar(xx, yy, yerr=sigma_y,
xerr=0.2, fmt='o', markersize=5,
ecolor="orange", capsize=3)
plt.show()
```



# Matplotlib (bar)

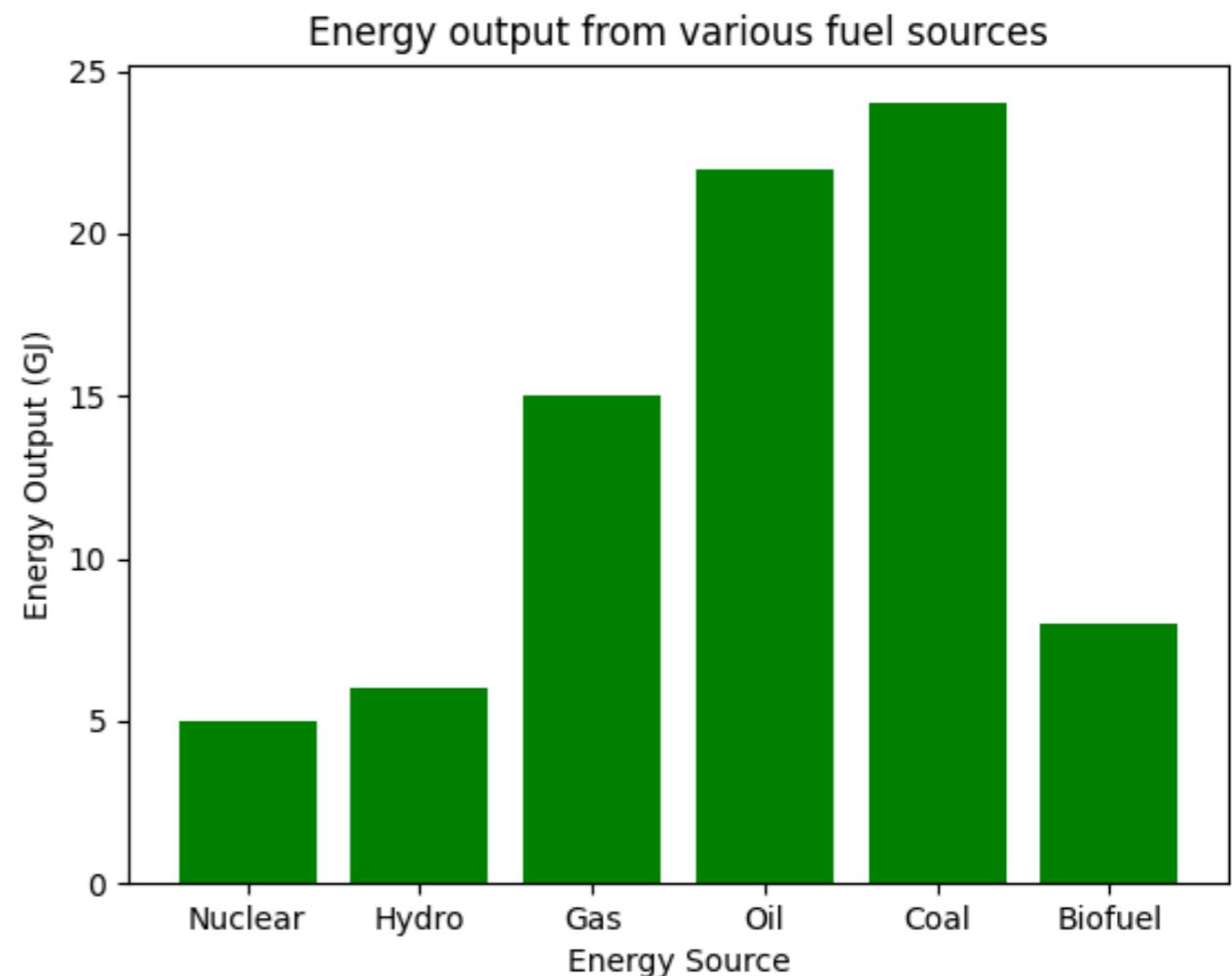
```
x = ['Nuclear', 'Hydro', 'Gas', 'Oil', 'Coal', 'Biofuel']
energy = [5, 6, 15, 22, 24, 8]
x_pos = [i for i, _ in enumerate(x)]

plt.bar(x_pos, energy, color='green')
plt.xlabel("Energy Source")
plt.ylabel("Energy Output (GJ)")
plt.title("Energy output from various fuel sources")
plt.xticks(x_pos, x)
plt.show()
```

- Desenhar cada ponto como uma barra vertical



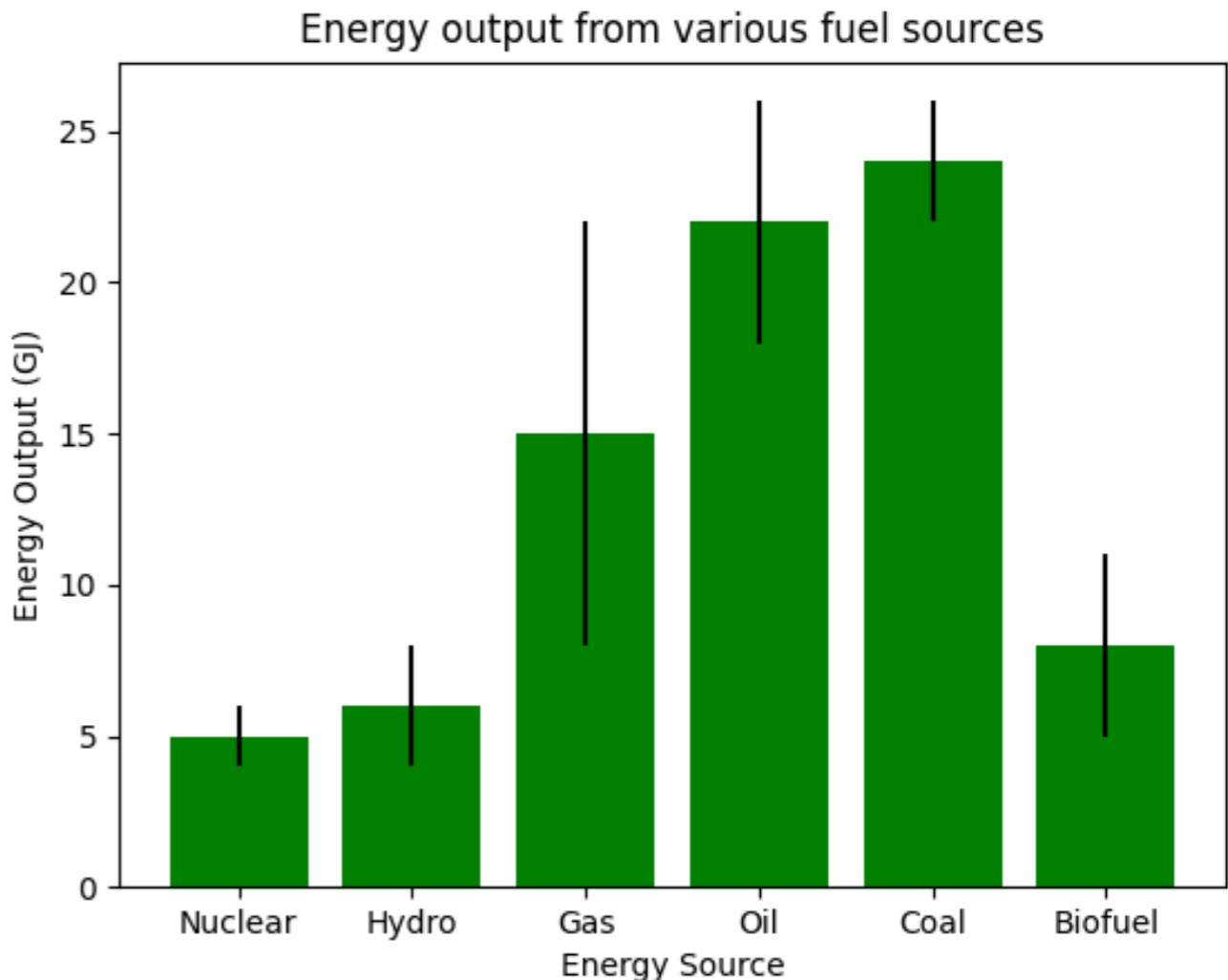
Valores dos eixos têm que ser numéricos



# Matplotlib (bar)

```
x = ['Nuclear', 'Hydro', 'Gas', 'Oil', 'Coal', 'Biofuel']
energy = [5, 6, 15, 22, 24, 8]
variance = [1, 2, 7, 4, 2, 3]
x_pos = [i for i, _ in enumerate(x)]  
  
plt.bar(x_pos, energy, color='green', yerr=variance)
plt.xlabel("Energy Source")
plt.ylabel("Energy Output (GJ)")
plt.title("Energy output from various fuel sources")
plt.xticks(x_pos, x)
plt.show()
```

- Podemos acrescentar uma margem de erro



# Matplotlib (bar)

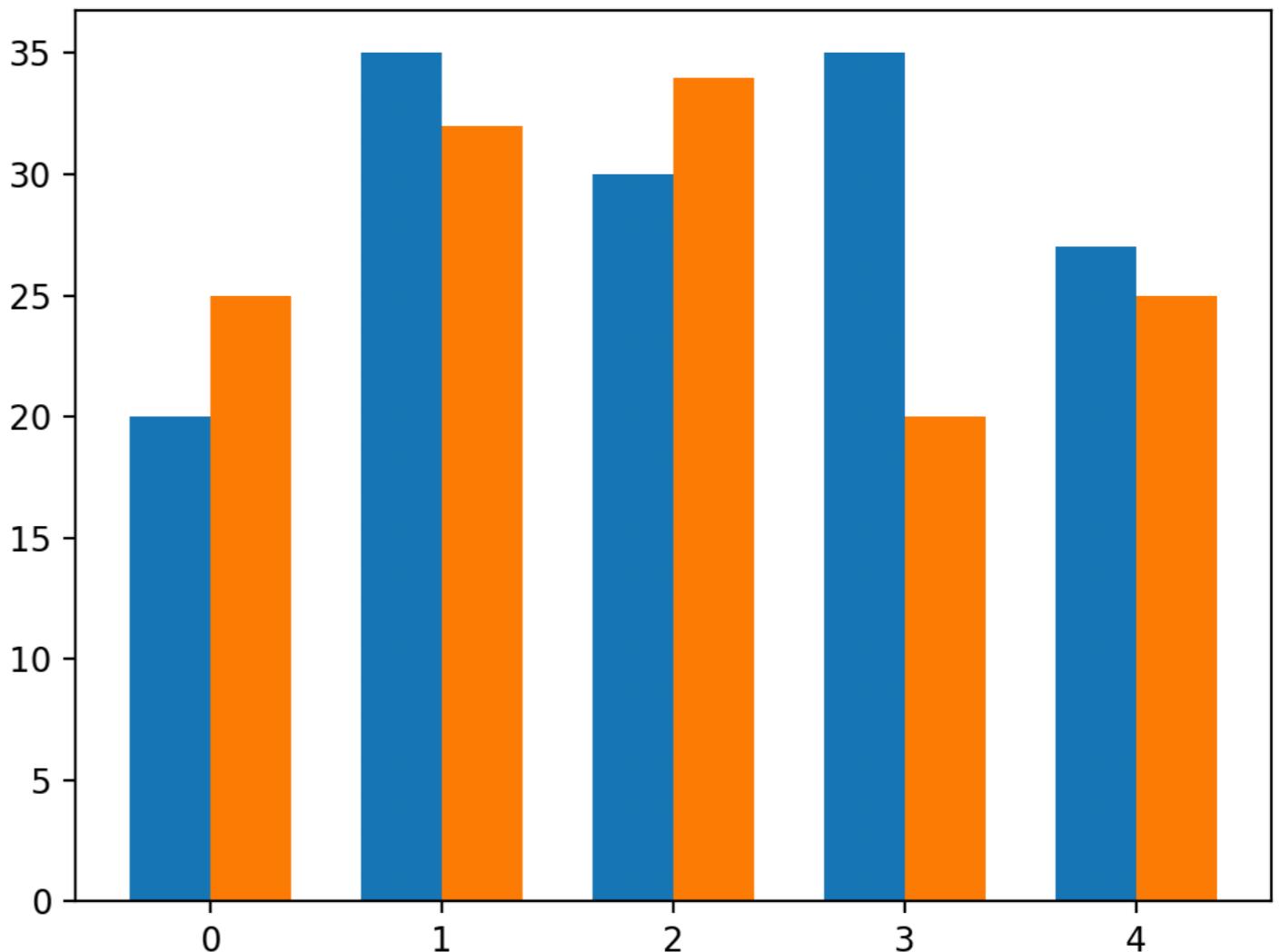
```
xx = np.arange(5)
men_means = (20, 35, 30, 35, 27)
women_means = (25, 32, 34, 20, 25)

width=0.35
plt.bar(xx - width/2, men_means, width=width)
plt.bar(xx + width/2, women_means, width=width)
plt.show()
```

- Podemos desenhar múltiplas barras



Controlando a posição nos X



# Matplotlib (bar)

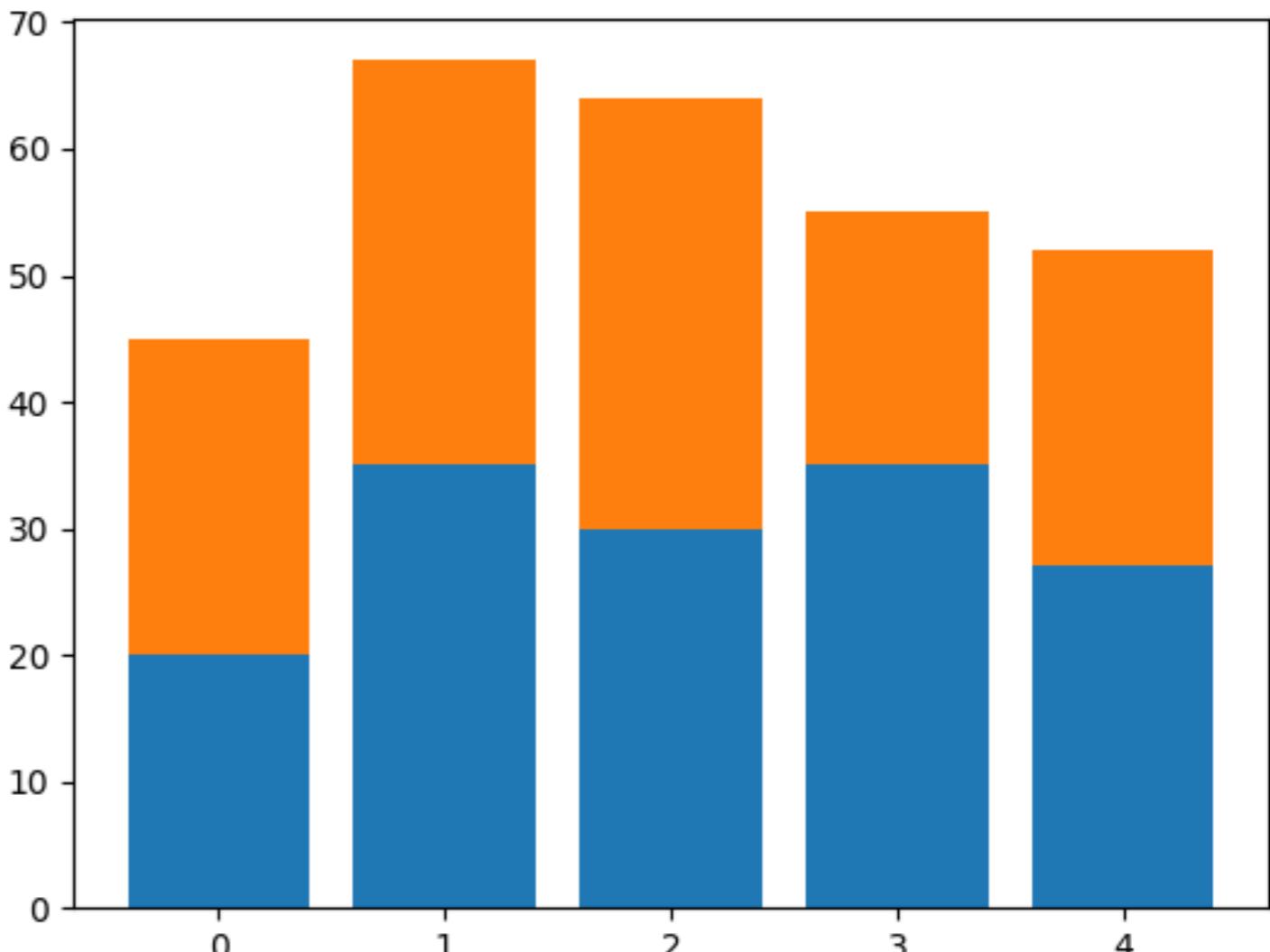
```
xx = np.arange(5)
men_means = (20, 35, 30, 35, 27)
women_means = (25, 32, 34, 20, 25)

plt.bar(xx,men_means)
plt.bar(xx,women_means,bottom=men_means)
plt.show()
```

- Podemos desenhar múltiplas barras



Controlando a posição nos Y



# Matplotlib (bar)

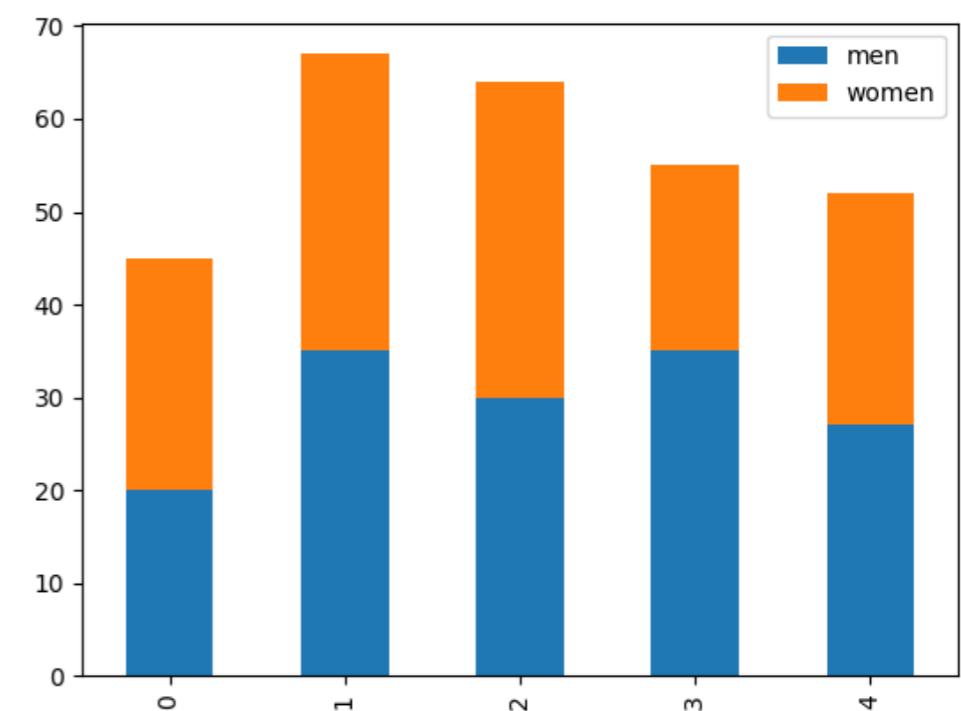
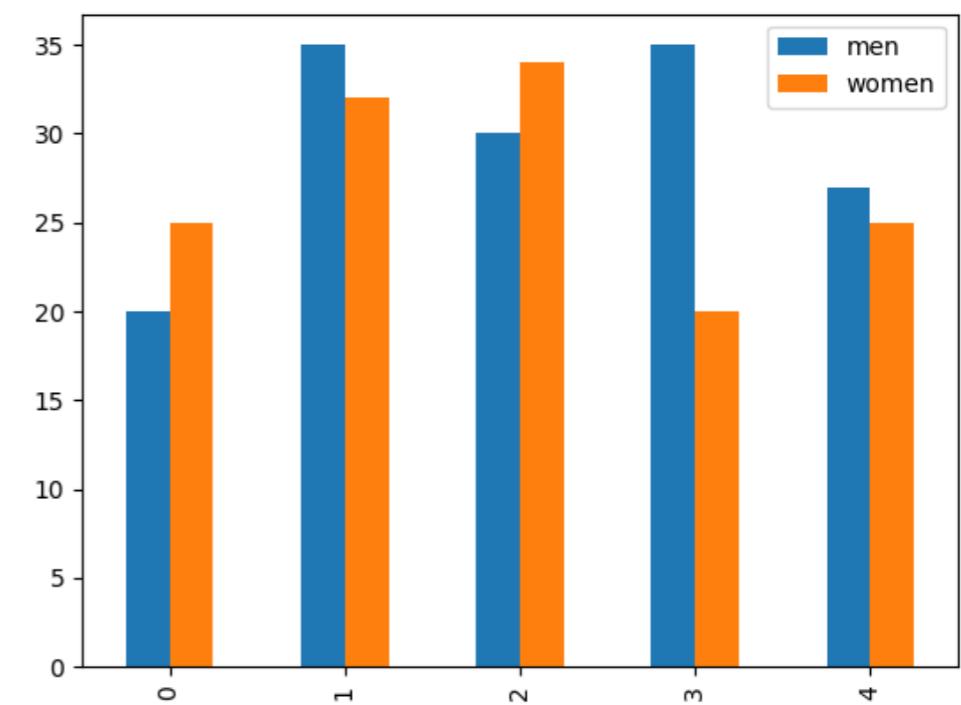


Desenhar múltiplas barras é mais simples utilizando um *DataFrame*

```
men_means = (20, 35, 30, 35, 27)
women_means = (25, 32, 34, 20, 25)
df =
pd.DataFrame({'men':men_means, 'women':women_means})
```

```
# lado a lado
df.plot(kind='bar')
plt.show()
```

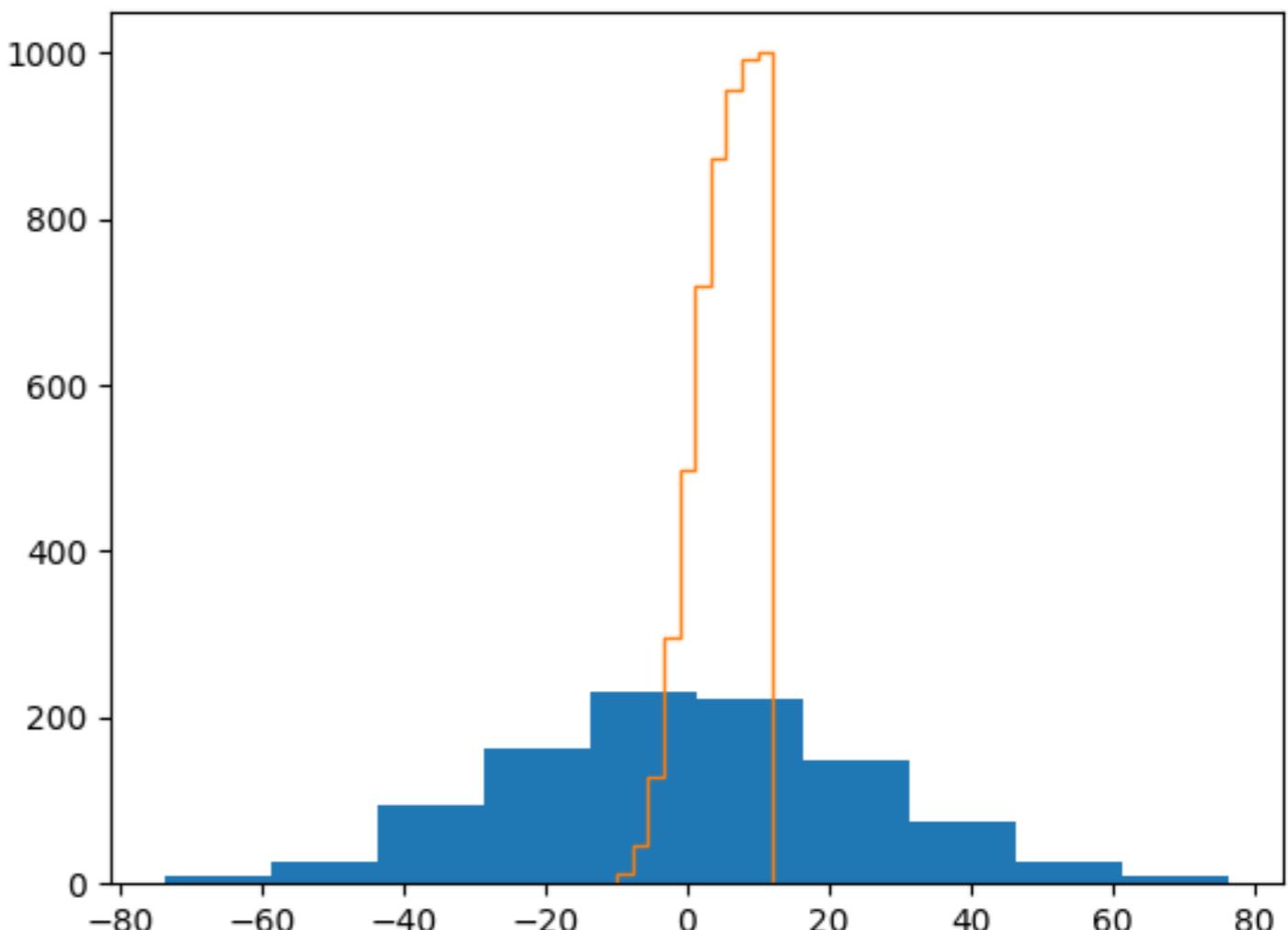
```
# em cima uma da outra
df.plot(kind='bar', stacked=True)
plt.show()
```



# Matplotlib (hist)

- Desenhar um histograma a partir de uma sequência
  - X = valores na sequência
  - Y = número de ocorrências de cada valor

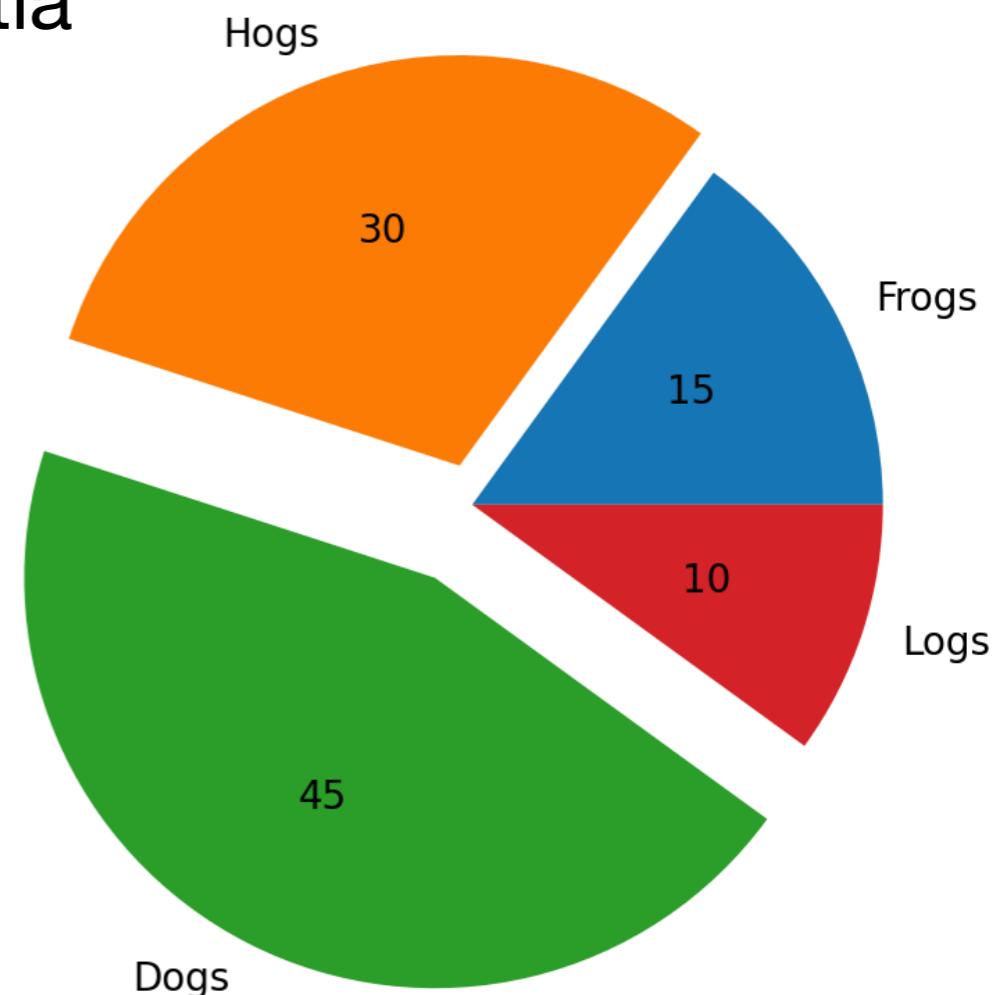
```
data1 = np.random.normal(1.,  
25., 1000)  
data2 = np.random.normal(1.,  
4., 1000)  
plt.hist(data1)  
plt.hist(data2, histtype="step"  
, cumulative=True)  
plt.show()
```



# Matplotlib (pie)

- Desenhar um gráfico em forma de “tarte”
- Podemos controlar expansão de cada fatia
- Podemos desenhar o valor de cada fatia  
(*autopct* = função que recebe o valor numérico e retorna uma string)

```
labels = ['Frogs', 'Hogs', 'Dogs', 'Logs']
sizes = [15, 30, 45, 10]
explode = (0, 0.1, 0.2, 0)
plt.pie(sizes, explode=explode,
        labels=labels, autopct=lambda
x:str(round(x)))
plt.show()
```

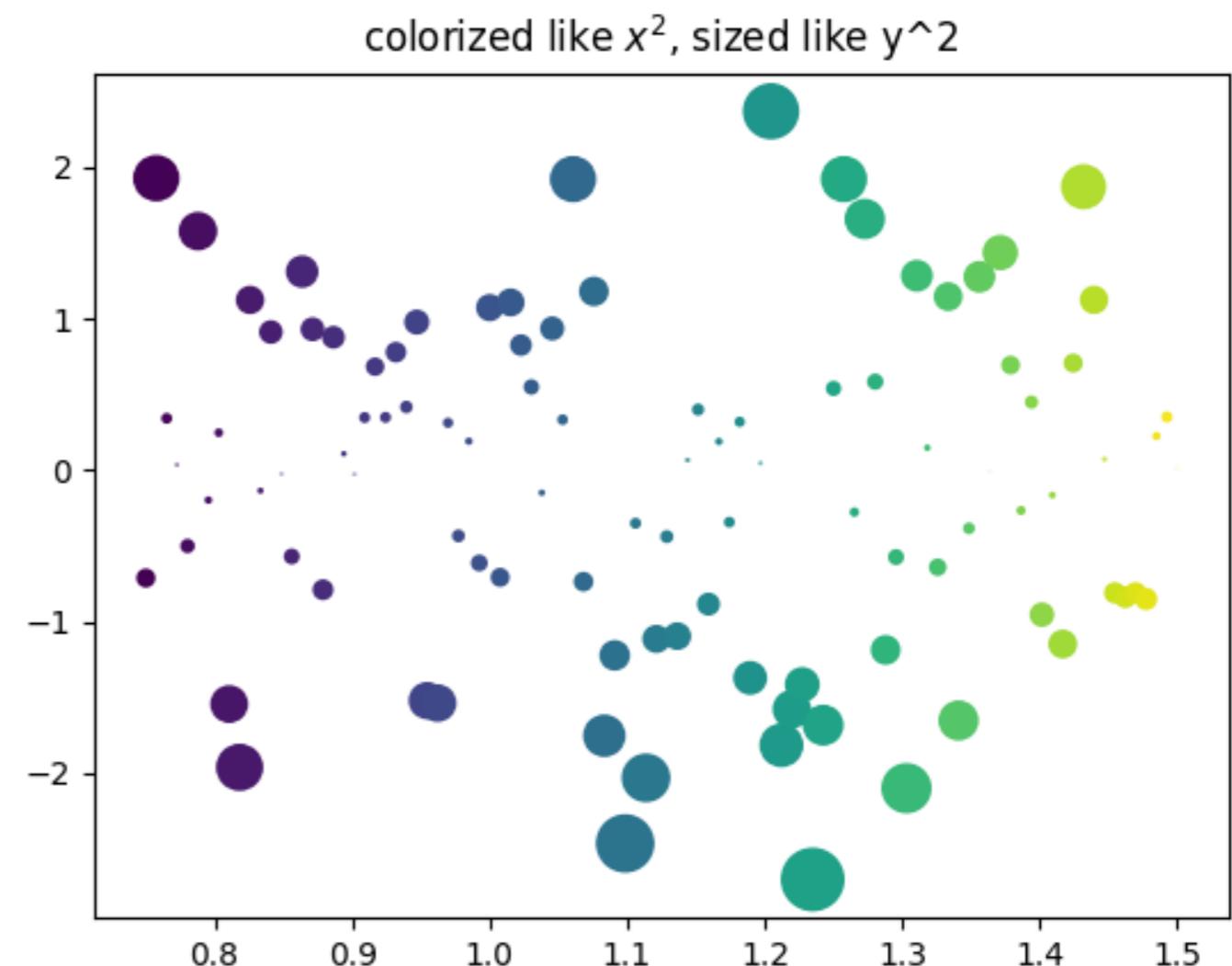


# Matplotlib (scatter)

- Desenhar cada ponto independentemente
- Podemos controlar cor e tamanho de cada ponto

```
xx = np.linspace(0.75, 1.5, 100)
yy = np.random.randn(len(xx))
zz = xx**2
ww = yy**2

plt.scatter(xx, yy, c=zz, s=ww*50)
plt.title("colorized like $x^2$,
sized like $y^2$")
plt.show()
```



# Matplotlib + Shapely

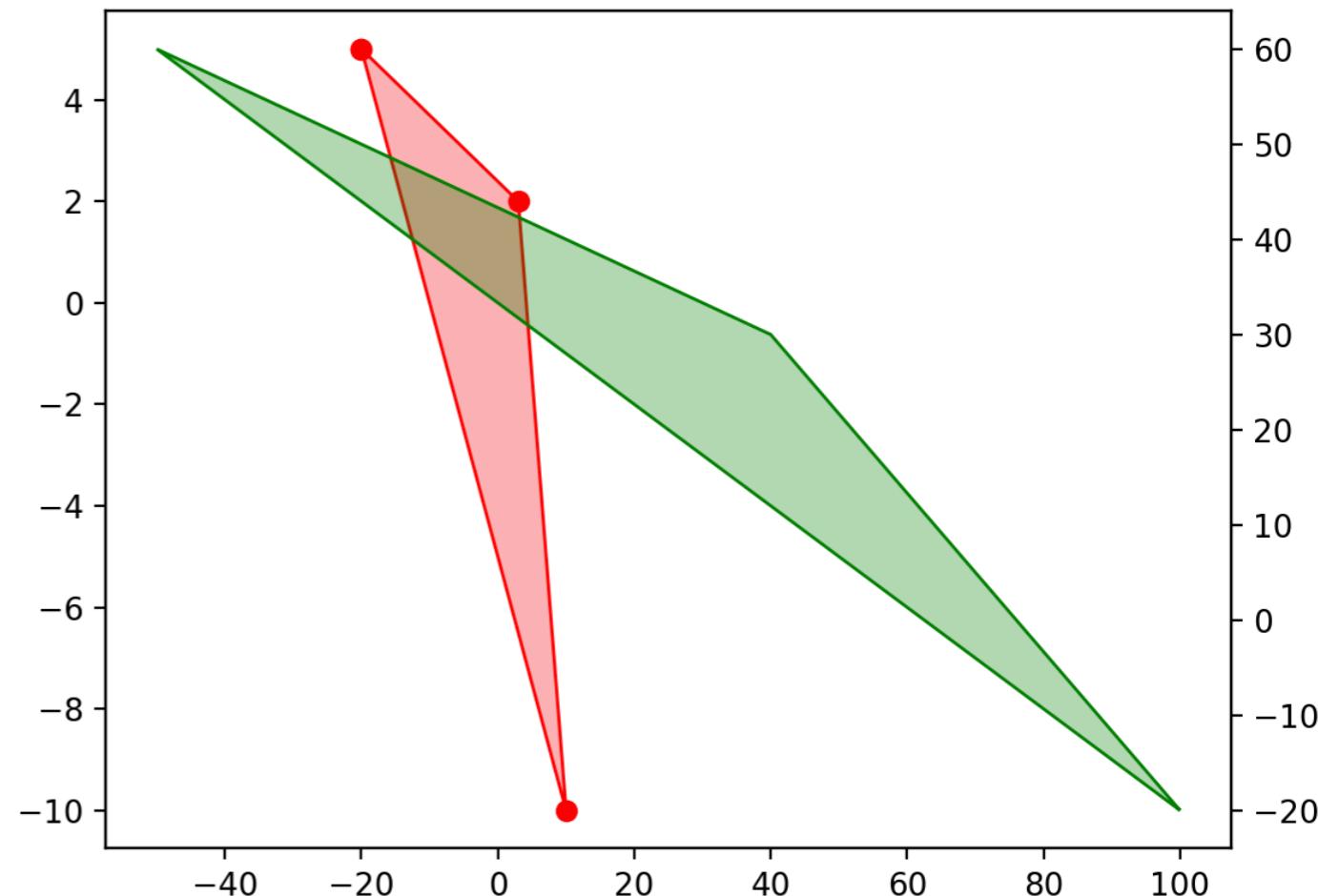
- Podemos desenhar formas geométricas com a biblioteca *shapely*
- Juntamente com outros gráficos e funcionalidades

```
import matplotlib.pyplot as plt
from shapely import *
from shapely.plotting import *

_, ax1 = plt.subplots()
ax2 = ax1.twinx()

p1 = Polygon([(-20, 5), (10, -10), (3, 2)])
p2 = Polygon([(100, -20), (40, 30), (-50, 60)])

plot_polygon(p1, ax=ax1, color="red")
plot_polygon(p2, ax=ax2, color="green",
, add_points=False)
plt.show()
```



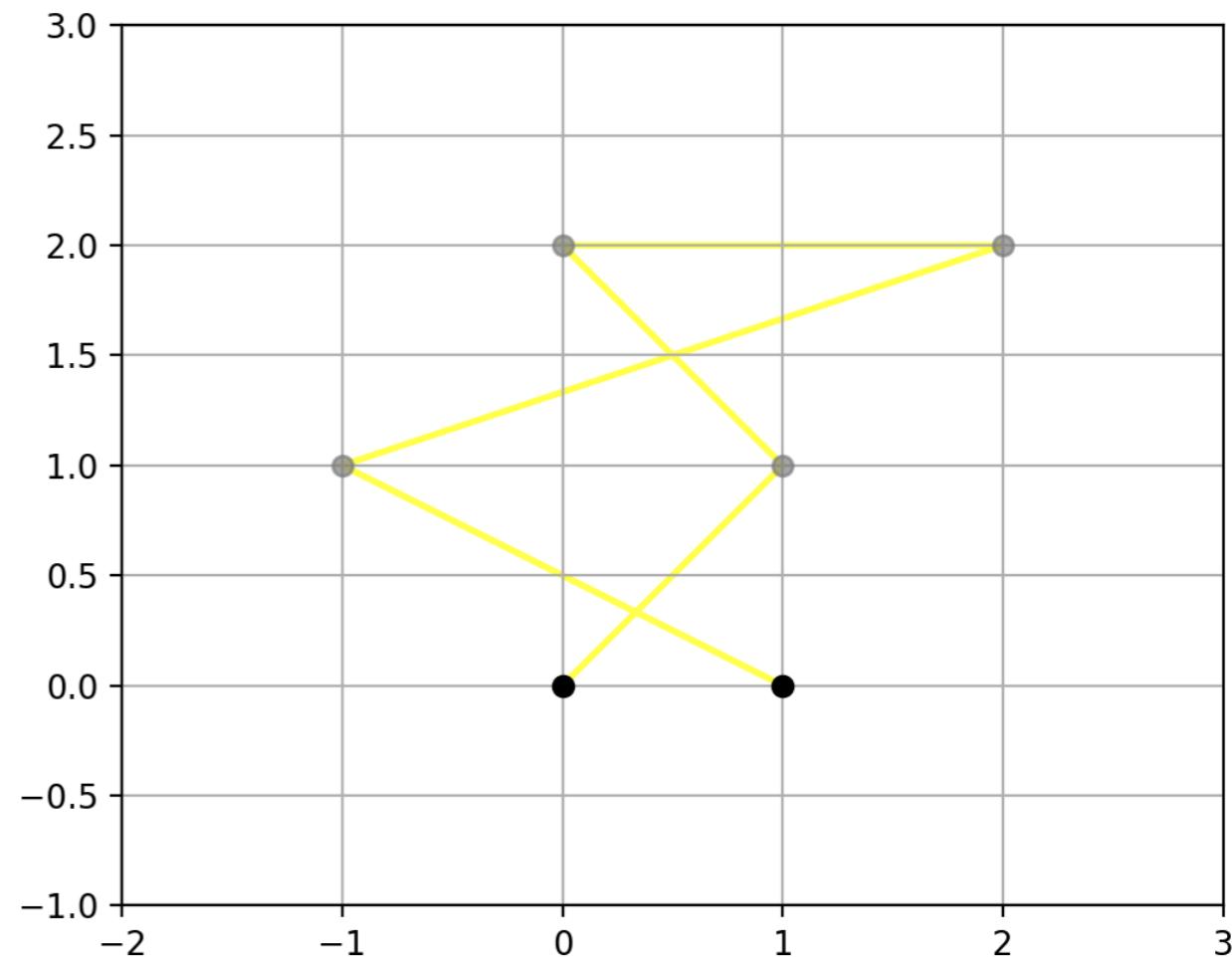
# Matplotlib + Shapely

- Podemos desenhar formas geométricas com a biblioteca *shapely*
- Juntamente com outros gráficos e funcionalidades

```
import matplotlib.pyplot as plt
from shapely import *
from shapely.plotting import *

line2 = LineString([(0, 0), (1, 1), (0, 2),
(2, 2), (-1, 1), (1, 0)])

plot_line(line2, add_points=False,
color="yellow", alpha=0.7)
plot_points(line2, color="gray", alpha=0.7)
plot_points(line2.boundary, color="black")
plt.grid(visible=True)
plt.xlim(-2, 3)
plt.ylim(-1, 3)
plt.show()
```



# Exemplo 1

- Desenhar um gráfico de áreas que apresente a percentagem de testes COVID-19 de cada tipo ao longo do tempo

```
amostras = pd.read_csv('amostras.csv')  
amostras.dropna(inplace=True)
```

```
amostras['pcr'] = amostras['amostras_pcr'] / amostras['amostras']  
amostras['antigenio'] = amostras['amostras_antigenio'] / amostras['amostras']  
amostras[['pcr', 'antigenio']].plot(kind='area', stacked=True)
```

```
plt.tick_params(axis='x', bottom=False, labelbottom=False)  
plt.ylim(0.4, 1)  
plt.yticks([0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1],  
          ["40%", "50%", "60%", "70%", "80%", "90%", "100%"])  
plt.show()
```



# Exemplo 2

- Dados climatéricos anuais de longa duração fornecidos pelo IPMA [aqui](#).
- Gráfico da evolução das temperaturas mínima e máxima por ano

```
dfs = pd.read_excel("PT100-tx-tn-prec.xlsx", sheet_name=None)
years = ['year', 'Annual']

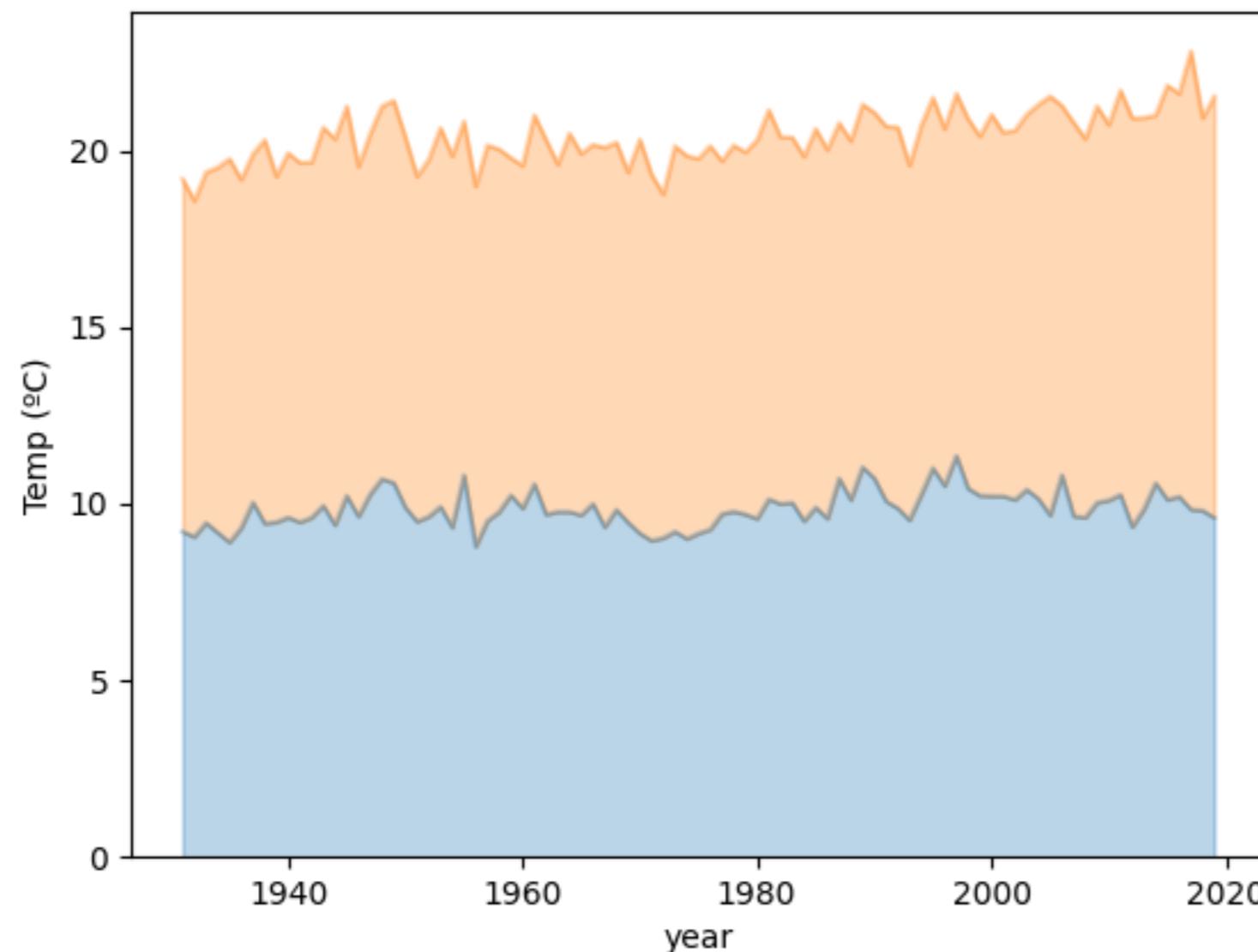
tmins = dfs['tmin']
tmins = tmins[years].copy()
tmins.dropna(inplace=True)

tmaxs = dfs['tmax']
tmaxs = tmaxs[years].copy()
tmaxs.dropna(inplace=True)

tempo = pd.merge(tmins, tmaxs, on='year', suffixes=('_tmin', '_tmax'))
tempo['year'] = tempo['year'].astype('uint16')
tempo.set_index('year', inplace=True)
tempo['Annual_tmax'] = tempo['Annual_tmax'] - tempo['Annual_tmin']
tempo.plot(kind='area', alpha=0.3, stacked=True, legend=False, ylabel=
'Temp (°C)')
```

# Exemplo 2

- Dados climatéricos anuais de longa duração fornecidos pelo IPMA [aqui](#).
- Gráfico da evolução das temperaturas mínima e máxima por ano



# Exemplo 3

- Dados climatéricos anuais de longa duração fornecidos pelo IPMA [aqui](#).
- Gráfico de ano, temperatura mínima e temperatura máxima por estação
  - E.g., qual o Verão mais quente de que há registo?

```
dfs = pd.read_excel("PT100-tx-tn-prec.xlsx", sheet_name=None)

seasons = ['Spring', 'Summer', 'Autumn', 'Winter']
yseasons = ['year'] + seasons

# temperaturas mínimas para cada ano e estação
tmins = dfs['tmin']
tmins = tmins[yseasons].copy()
tmins.dropna(inplace=True)
tmins['year'] = tmins['year'].astype('uint16')

# temperaturas máximas para cada ano e estação
tmaxs = dfs['tmax']
tmaxs = tmaxs[yseasons].copy()
tmaxs.dropna(inplace=True)
tmaxs['year'] = tmaxs['year'].astype('uint16')
```

# Exemplo 3

- Calcular temperaturas mínimas e máximas por estação, e anos correspondentes

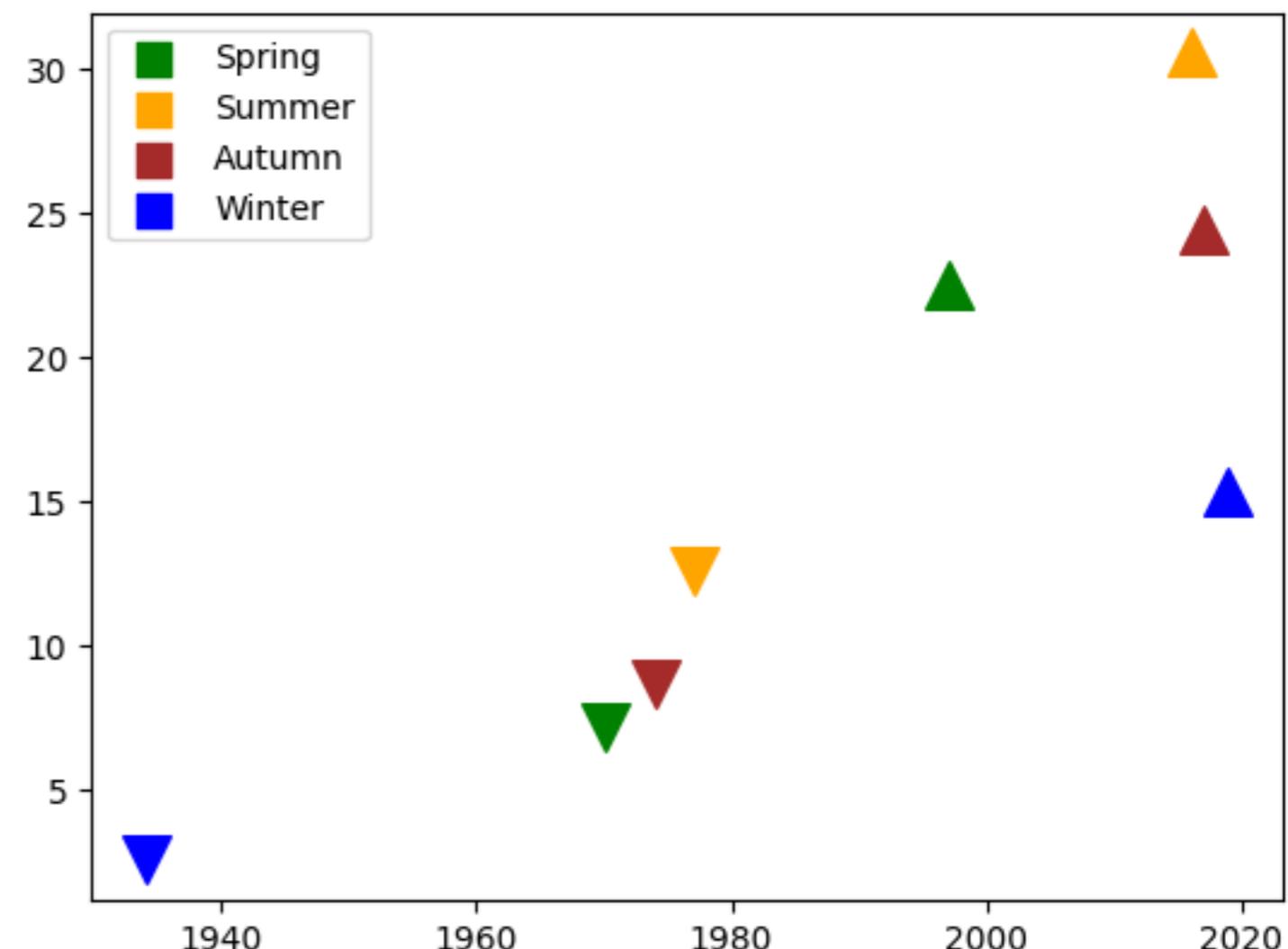
```
# colapsar as estações para índices
tmins = tmins.melt(id_vars=["year"],var_name="season",value_name="tmin")
tmins.set_index('year',inplace=True)
years = tmins.groupby('season').idxmin().rename(columns={'tmin':'year'})
temps = tmins.groupby('season').min()
tmins = years.join(temps)

# colapsar as estações para índices
tmaxs = tmaxs.melt(id_vars=["year"],var_name="season",value_name="tmax")
tmaxs.set_index('year',inplace=True)
years = tmaxs.groupby('season').idxmax().rename(columns={'tmax':'year'})
temps = tmaxs.groupby('season').max()
tmaxs = years.join(temps)
```

# Exemplo 3

```
for season in seasons:  
    plt.scatter(tmns['year'][season],tmns['tmin'][season],c=season_color[season],s=200,marker="v")  
    plt.scatter(tmxs['year'][season],tmxs['tmax'][season],  
c=season_color[season], s=200, marker="^")  
    plt.scatter([],[],c=season_color[season],s=100,marker='s',label=season)  
plt.legend()  
plt.show()
```

- Desenhar o gráfico



# Gráficos interativos (matplotlib)

# Gráficos

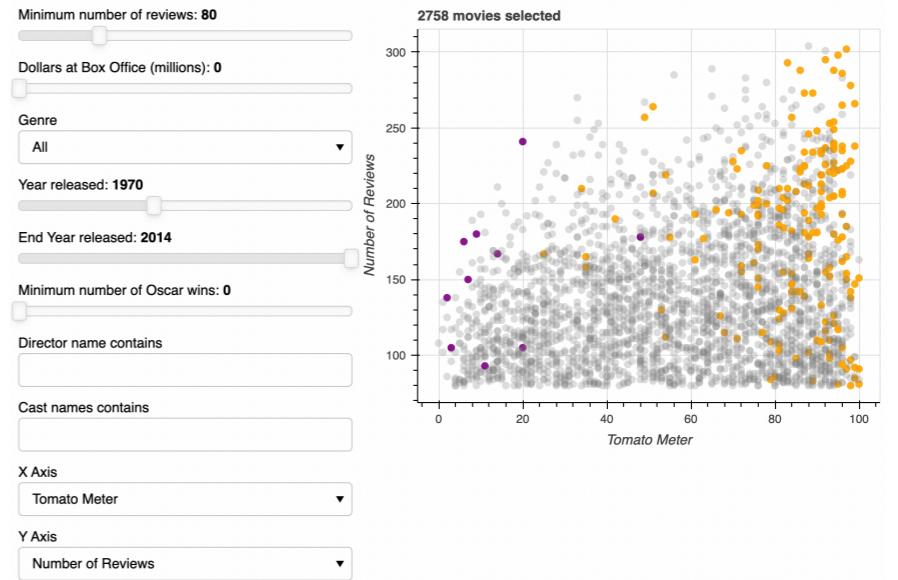
- Até agora:
  - Gráficos estáticos em *matplotlib*
    - Visualizar sempre a mesma informação
    - Exportar para imagem
- De seguida:
  - Gráficos interativos em *matplotlib*
    - Alterar a visualização consoante input do utilizador: botões, barras, ...
    - Utilizar a GUI para modo interativo
- Não vamos ver agora: outras bibliotecas para gráficos interativos (Bokeh, *plotly*, ...)

## Bokeh web example

### AN INTERACTIVE EXPLORER FOR MOVIE DATA

Interact with the widgets on the left to query a subset of movies to plot. Hover over the circles to see more information about each movie.

Inspired by the [Shiny Movie Explorer](#). (Information from OMDB)



# Gráficos interativos

- Em *matplotlib*, um gráfico interativo consiste em:
  - Gráficos estáticos desenhados normalmente
  - Widgets que respondem a eventos
- Eventos podem:
  - Mostrar/esconder gráficos já desenhados
  - Alterar componentes dos gráficos (cor, eixos, ...)

# Matplotlib (Axes)

- Um gráfico em *matplotlib* é um objeto da classe *Axes*
- Algumas funções criam explicitamente *Axes*, e.g.

```
# criar um gráfico vazio
_,ax = plt.subplots()
# desenha uma Series e retorna um novo gráfico
ax = series.plot()
# desenha um DataFrame e retorna um novo gráfico
ax = dataframe.plot()
```

- Com métodos *get\_atributo* e *set\_atributo*, podemos alterar um *atributo* de um gráfico (lista completa):
  - *visible*, *xlim*, *ylim*, *xticks*, *yticks*, *xlabel*, *ylabel*, ...

# Matplotlib (*Line2D*)

- Um desenho de uma curva num gráfico é um objeto da classe *Line2D*
- A função *plt.plot(...)* cria explicitamente uma lista de *Line2D*, e.g.

```
# criar um gráfico vazio
_,ax = plt.subplots()
# desenha um plot no gráfico ax
ls1 = ax.plot(...)
# desenha um plot num novo gráfico
ls2 = plt.plot(...)
```

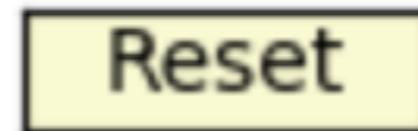
- Com métodos *get\_atributo* e *set\_atributo*, podemos alterar um *atributo* de uma *Line2D* (lista completa):
  - *visible, label, xdata, ydata, linestyle, linewidth, marker, ...*

# Widgets

- Alguns widgets *matplotlib* que vamos ver:
  - Botões
  - Botões de seleção
  - Botões de seleção exclusiva
  - Barras deslizantes

# Botões

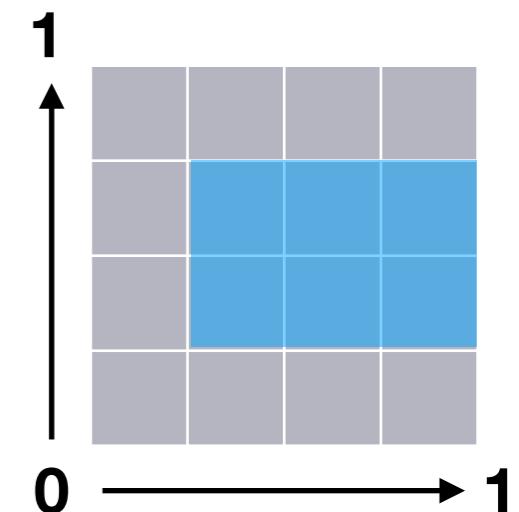
- Um botão é um objeto retangular com texto e um evento “fui clickado”



- Um retângulo é construído como uma lista com 4 elementos entre 0 e 1 relativos à janela:

[0.25,0.25,0.75,0.5]

- *[posicaoX, posicaoY, comprimentoX, comprimentoY]*
- 0 = esquerda/baixo e 1 = direita/cima
- Um evento é uma função que recebe informação sobre o evento (e.g. a posição do rato) e não retorna nada, mas altera alguma propriedade dos gráficos como efeito secundário



# Botões

- E.g., um botão que mostra/esconde uma curva

- Evento

- Redesenha

```

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import Button

# calcula um seno
xx = np.arange(0.0, 2.0, 0.01)
yy = np.sin(2*np.pi*xx)

# desenha uma linha, ln é o objeto
ln, = plt.plot(xx, yy)

# um retângulo perto do canto superior direito
rect = plt.axes([0.7, 0.9, 0.25, 0.07])
# um botão
button = Button(rect, 'Mostrar / Esconder')

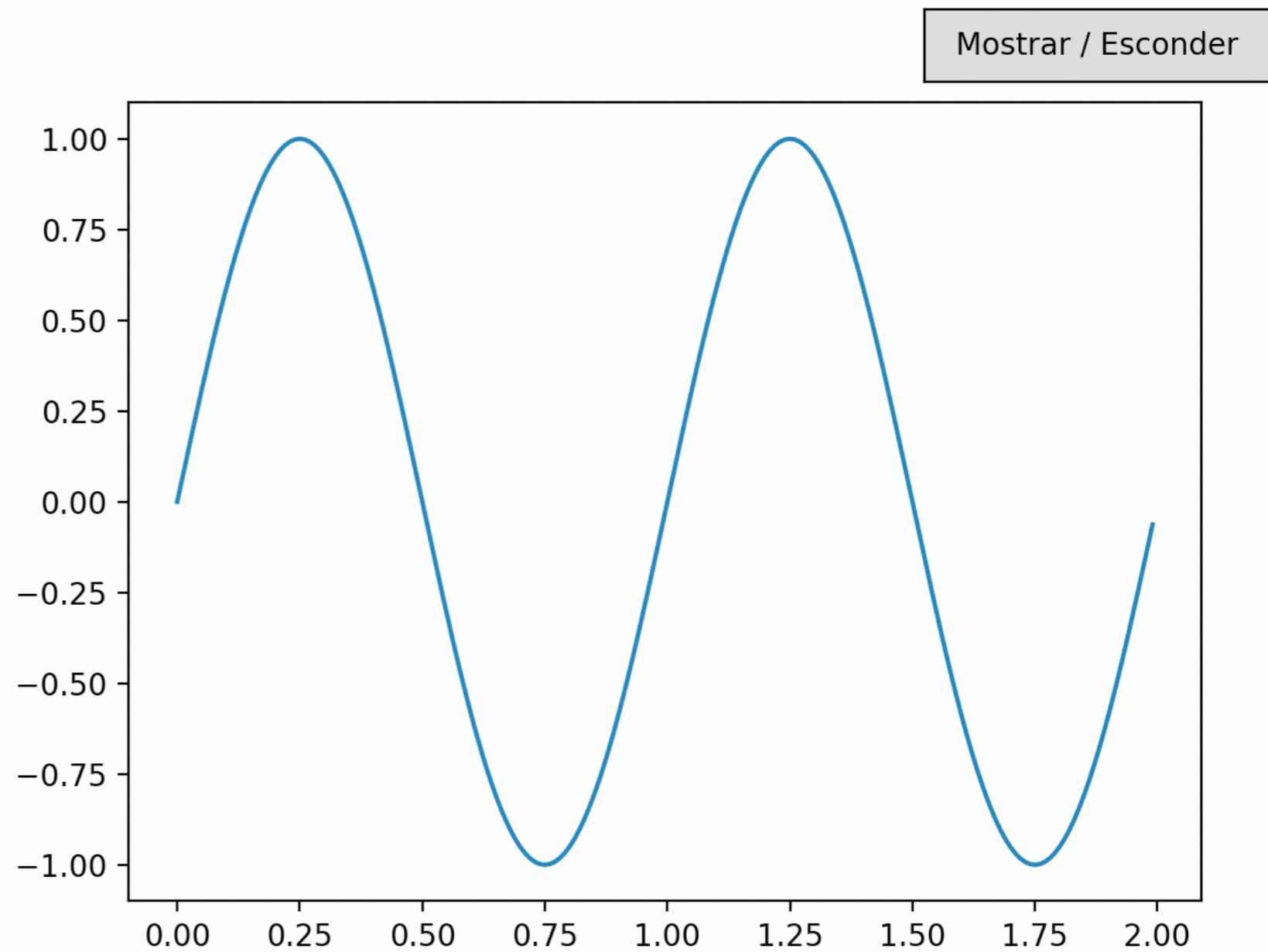
# a função que reage ao evento
def reage(info):
    # lê e altera a visibilidade do gráfico
    ln.set_visible(not ln.get_visible())
    plt.draw()

# liga o evento à função
button.on_clicked(reage)
plt.show()

```

# Botões

- E.g., um botão que mostra/esconde uma curva



# Botão de seleção

- Um botão de seleção (*check button*) é um objeto retangular com uma sequência de opções que podem ser ativadas/desativadas
  - Recebe um parâmetro opcional para definir o estado inicial de cada opção
- É como se cada opção fosse um botão independente
- O evento “fui clickado” diz qual a opção “clickada”



# Botão de seleção

- E.g., um *check button* que permite mostrar/esconder múltiplas curvas

```

from matplotlib.widgets import CheckButtons

xx = np.arange(0.0, 2.0, 0.01)
yy1 = np.sin(2*np.pi*xx)
yy2 = np.sin(4*np.pi*xx)

ln1, = plt.plot(xx,yy1)
ln2, = plt.plot(xx,yy2)

rect = plt.axes([0.7,0.9,0.1,0.08])
# um botão de seleção
button = CheckButtons(rect, ('2 Hz', '4 Hz'),
                      (True, True))

def reage_ln(ln):
    ln.set_visible(not ln.get_visible())
    plt.draw()

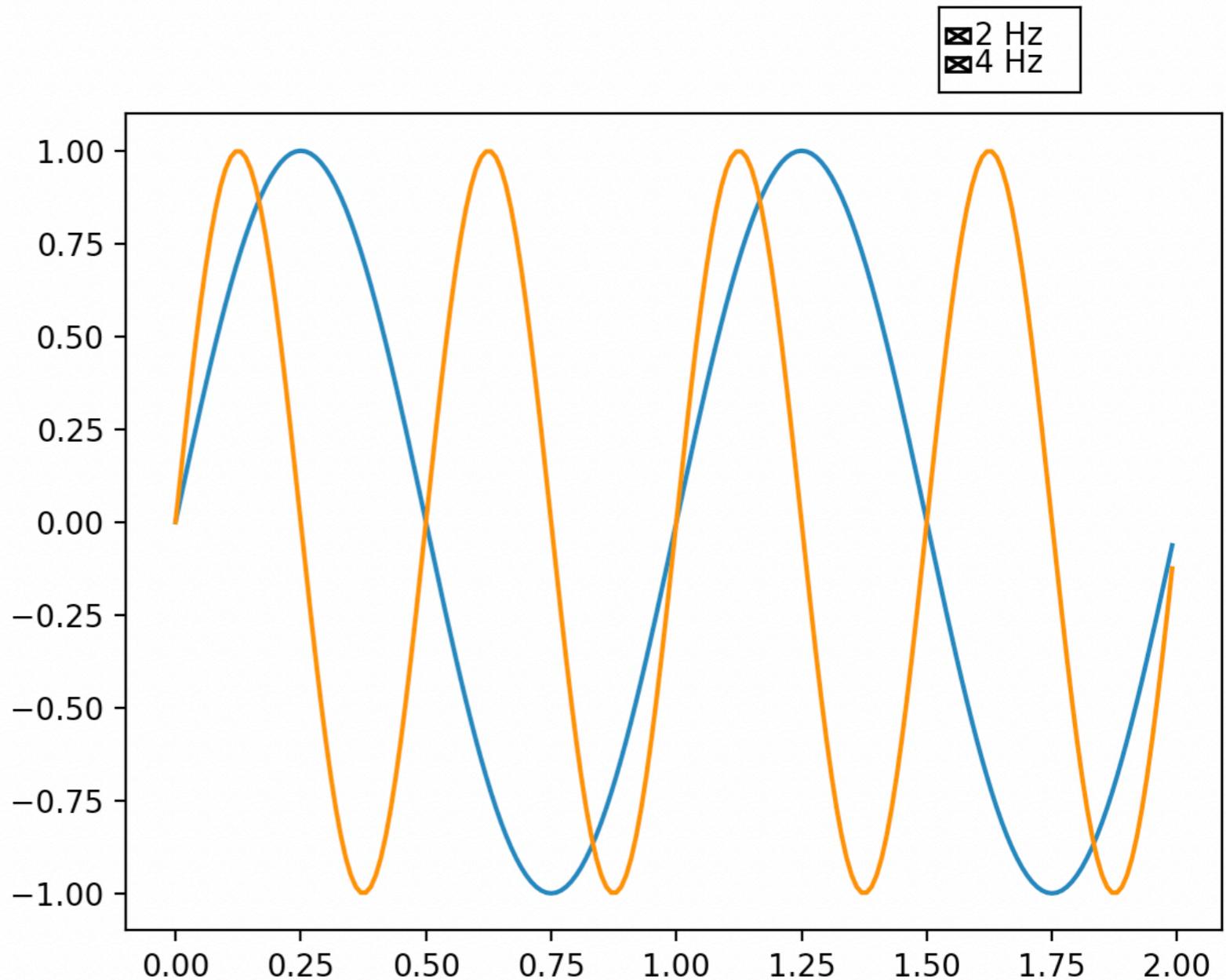
# a função que reage ao evento recebe a opção
def reage(label):
    if label=='2 Hz': reage_ax(ln1)
    elif label=='4 Hz': reage_ax(ln2)

button.on_clicked(reage)
plt.show()

```

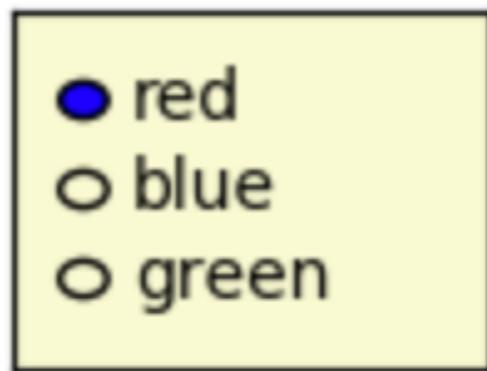
# Botão de seleção

- E.g., um *check button* que permite mostrar/esconder múltiplas curvas



# Botão de seleção exclusiva

- Um botão de seleção exclusiva (*radio button*) é um objeto retangular com uma sequência de opções em que uma e apenas uma está ativada



- O evento “fui clickado” diz qual a opção ativa de momento

# Botão de seleção exclusiva

- E.g., um *radio button* que permite alterar a frequência da curva

```
from matplotlib.widgets import RadioButtons

xx = np.arange(0.0, 2.0, 0.01)
yy = np.sin(2*np.pi*xx)

ln, = plt.plot(xx, yy)

rect = plt.axes([0.7, 0.9, 0.1, 0.08])
# um botão de seleção exclusiva
button = RadioButtons(rect, ('2 Hz', '4 Hz'))

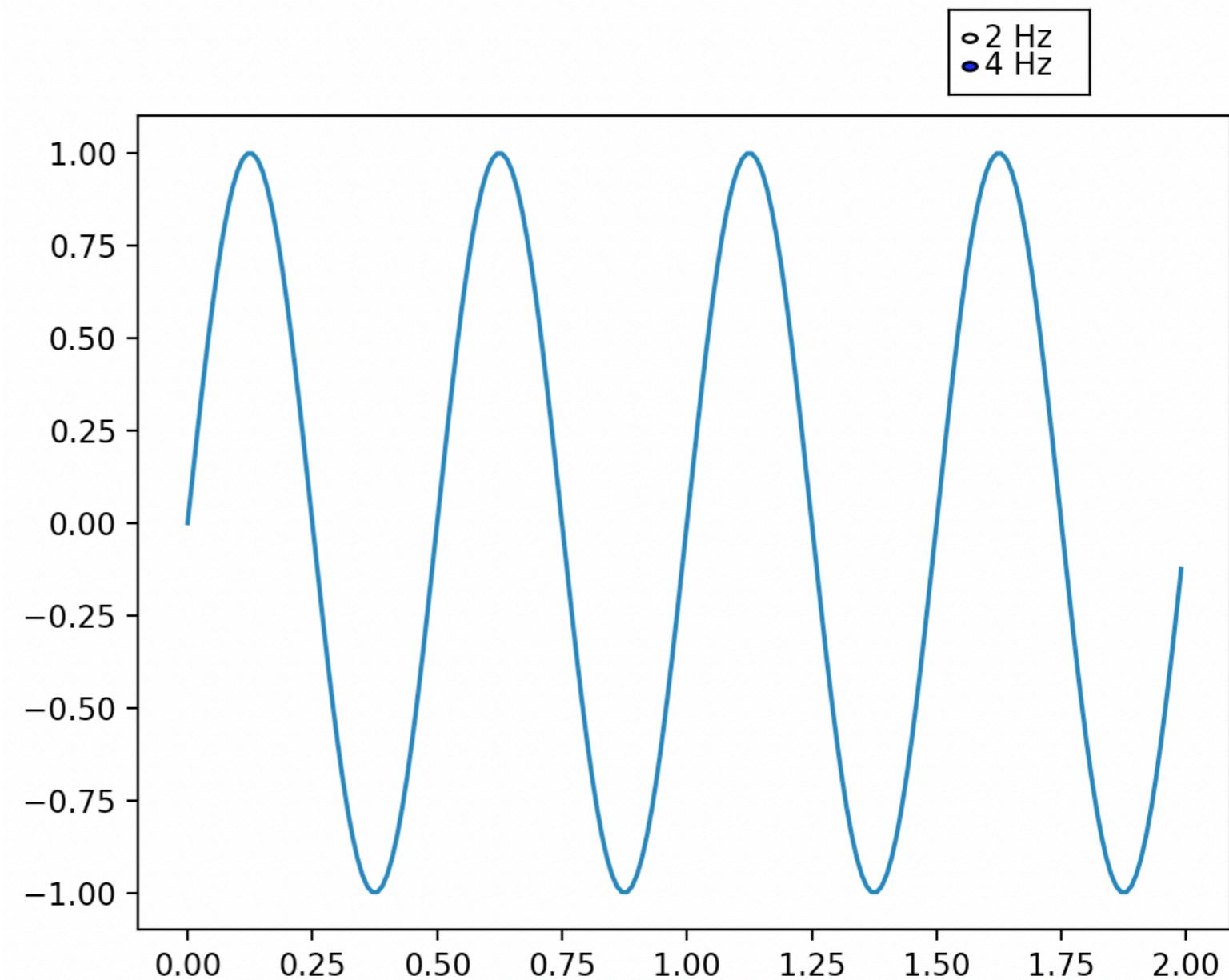
def reage_ln(n):
    # altera a curva desenhada
    ln.set_ydata(np.sin(n*np.pi*xx))
    plt.draw()

# a função que reage ao evento recebe a opção
def reage(label):
    if label=='2 Hz': reage_ln(2)
    elif label=='4 Hz': reage_ln(4)

button.on_clicked(reage)
plt.show()
```

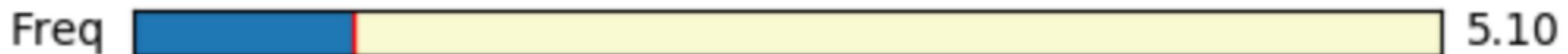
# Botão de seleção exclusiva

- E.g., um *radio button* que permite alterar a frequência da curva



# Barra deslizante

- Uma barra deslizante (*slider*) é um objeto retangular com um nome e um intervalo de valores possíveis
  - Recebe parâmetros opcionais como valor inicial e “salto” mínimo



- O evento “fui alterada” diz qual o valor atual (dentro do intervalo)

# Barra deslizante

- E.g., um *slider* que permite alterar a frequência da curva

```
from matplotlib.widgets import Slider

xx = np.arange(0.0, 2.0, 0.01)
yy = np.sin(2*np.pi*xx)

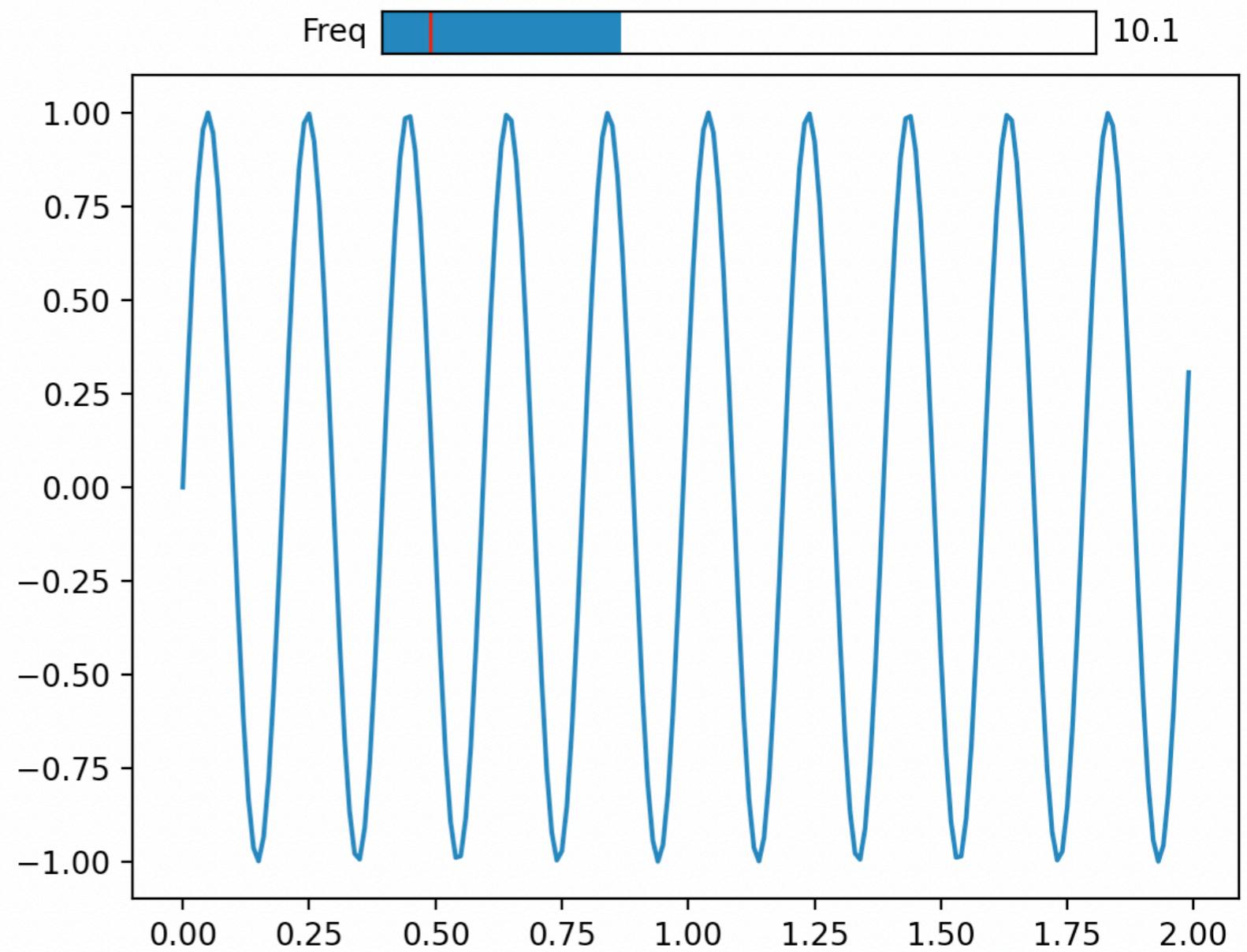
line, = plt.plot(xx,yy)
rect = plt.axes([0.3,0.9,0.5,0.04])
# um slider
slider = Slider(rect, 'Freq', 0.1, 30.0,
valinit=2, valstep=0.5)

# a função que reage ao evento recebe o
# novo valor
def reage(freq):
    # altera a curva desenhada
    line.set_ydata(np.sin(freq*np.pi*xx))
    plt.draw()

slider.on_changed(reage)
plt.show()
```

# Barra deslizante

- E.g., um *slider* que permite alterar a frequência da curva



# Exemplo 1

- Dados climatéricos anuais de longa duração fornecidos pelo IPMA [aqui](#).
- Gráfico interativo das temperaturas mínima e máxima por mês, ano a ano

```
dfs = pd.read_excel("PT100-tx-tn-prec.xlsx", sheet_name=None)
months =
['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct',
'Nov', 'Dec']
ymonths = ['year'] + months

tmins = dfs['tmin']
tmins.rename(columns={'Dez': 'Dec'}, inplace=True)
tmins = tmins[ymonths].copy()
tmins.dropna(inplace=True)
tmins['year'] = tmins['year'].astype('uint16')
tmins.set_index('year', inplace=True)

tmaxs = dfs['tmax']
tmaxs = tmaxs[ymonths].copy()
tmaxs.dropna(inplace=True)
tmaxs['year'] = tmaxs['year'].astype('uint16')
tmaxs.set_index('year', inplace=True)
```

# Exemplo 1

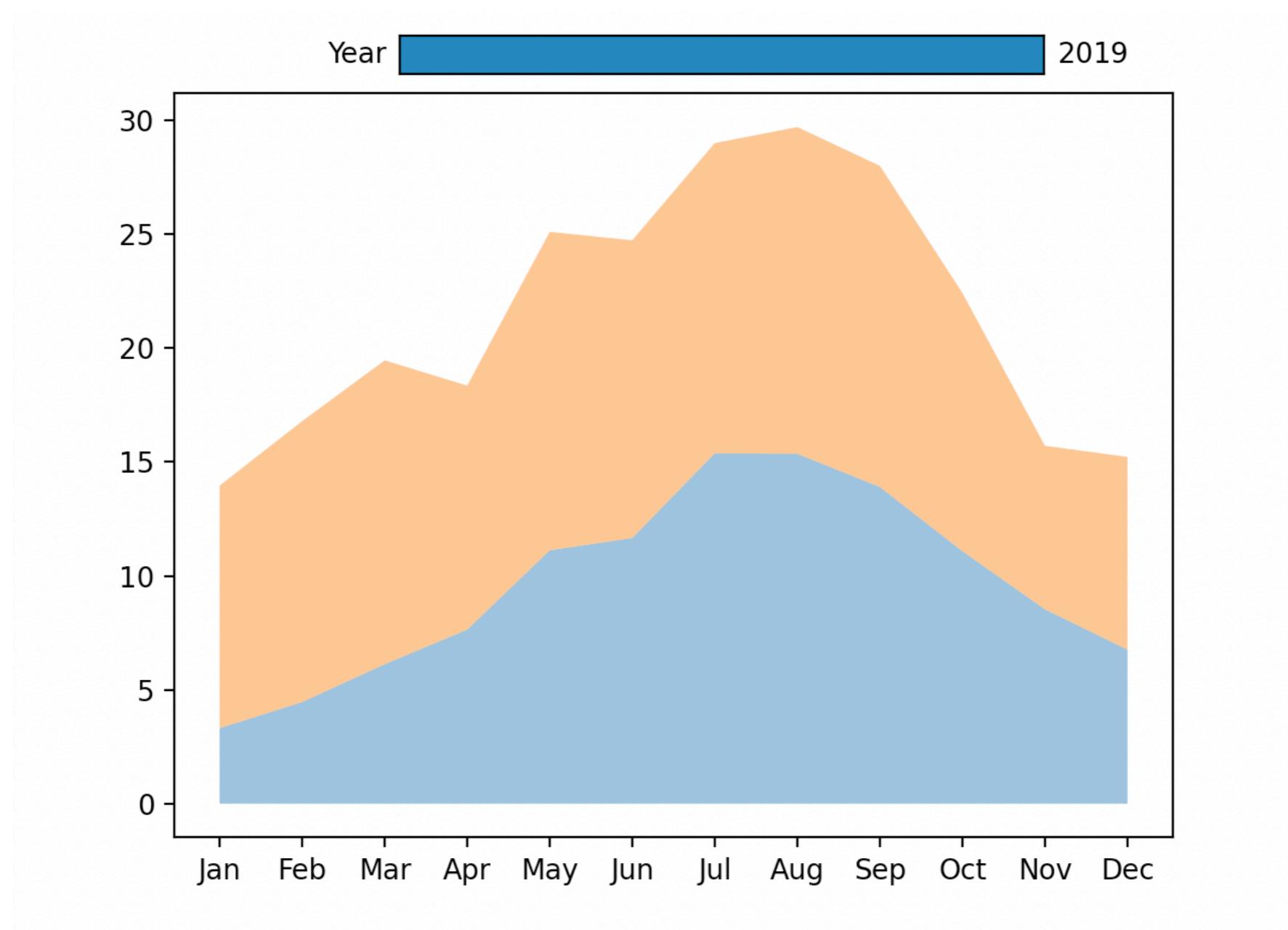
- Desenhar o gráfico com um *slider* anual

```
minyear = tmins.index[0]
maxyear = tmins.index[-1]
# cria um gráfico
_,ax = plt.subplots()
# desenha um ano
def draw_year(year):
    ax.fill_between(months,tmins.loc[year],alpha=0.5)
    ax.fill_between(months,tmins.loc[year],tmaxs.loc[year],alpha=0.5)
# desenha o ano mais recente por defeito
draw_year(maxyear)

# cria um novo slider
rect = plt.axes([0.3,0.9,0.5,0.04])
slider = Slider(rect,'Year',minyear,maxyear, valinit=maxyear, valstep=1)
# redesenha o gráfico para o ano atual
def reage(year):
    ax.clear()
    draw_year(year)
slider.on_changed(reage)
plt.show()
```

# Exemplo 1

- Desenhar o gráfico com um *slider* anual



# Exemplo 1

- Acrescentar visualização de precipitação

```
#carrega dados precipitação
precs = dfs['prec']
precs = precs[ymonths].copy()
precs.dropna(inplace=True)
precs['year'] = precs['year'].astype('uint16')
precs.set_index('year', inplace=True)

# cria um num novo eixo dos Y
ax2 = ax.twinx()
# desenha precipitação
lprec, = ax2.plot(months, precs.loc[maxyear], color='green')
# redesenha precipitação
def redraw_prec(year):
    lprec.set_ydata(precs.loc[year])
    ax2.set_ylim(0, precs.loc[year].max() + 10)
# atualiza também precipitação
def reage(year):
    ...
    redraw_prec(year)
```

# Exemplo 1

- Acrescentar *check button*

```
# cria dois eixos nos Y
ax.set_yticks([])
# renomear ax para ax1 no resto do código
ax1 = ax.twinx()
ax2 = ax1.twinx()

# um botão de seleção
rect = plt.axes([0.15, 0.77, 0.1, 0.08])
button = CheckButtons(rect, ('Temp', 'Prec'), (True, True))
def altera(label):
    if label=='Temp': ax1.set_visible(not ax1.get_visible())
    elif label=='Prec': ax2.set_visible(not ax2.get_visible())
    plt.draw()
button.on_clicked(alterar)
```

# Exemplo 1

- Gráfico de temperatura e precipitação anual

