Notas de aula de Python

Matplotlib - Apresentação 1

Prof. Louis Augusto
louis.augusto@ifsc.edu.br

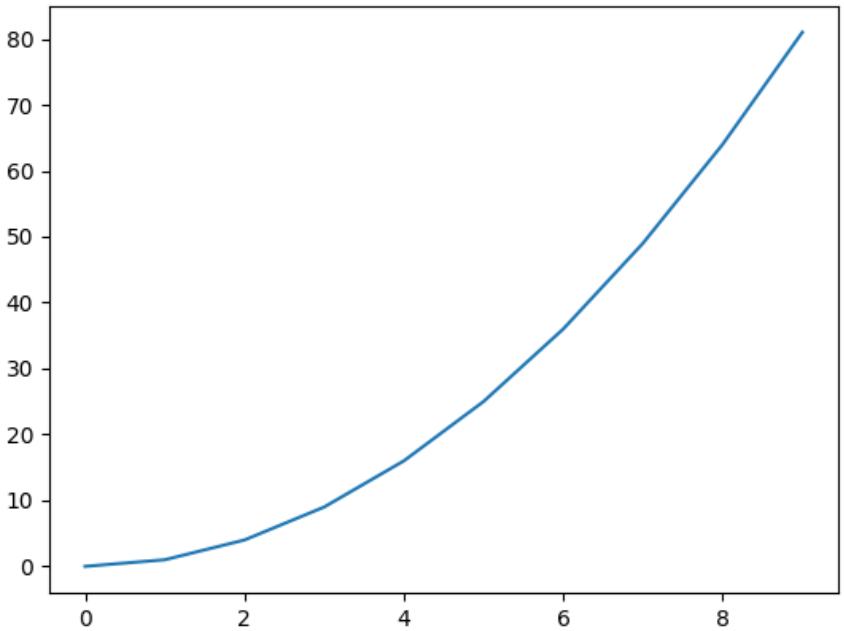
Instituto Federal de Santa Catarina
Campus São José

Índice

Sumário

Um gráfico MUITO simples

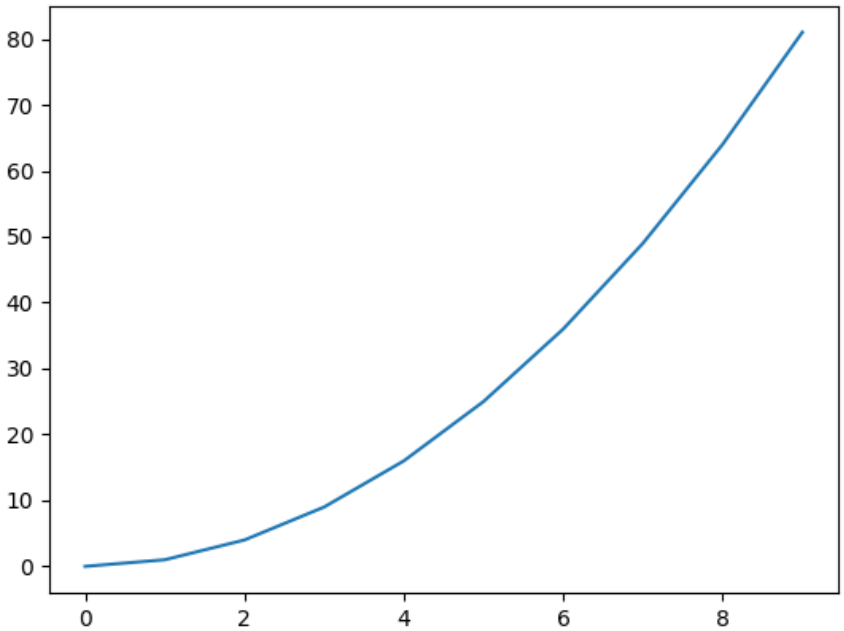
```
import matplotlib.pyplot as plt
X = list(range(10)) #Constrói um vetor de 10 posições
Y = [x**2 for x in X]
    #Pega os valores de x^2 para cada x e coloca em Y
plt.plot(X, Y)
plt.show()
```



O que importa para fazer este gráfico é montar ambos os vetores, x e y, colocar os dados no matplotlib com plot, e depois mostrar com show.
O vetor Y possui os valores de x^2 para cada valor de x.

Um gráfico MUITO simples

```
import matplotlib.pyplot as plt
X = list(range(10)) #Constrói um vetor de 10 posições
Y = [x**2 for x in X]
    #Pega os valores de x^2 para cada x e coloca em Y
plt.plot(X, Y)
plt.show()
```

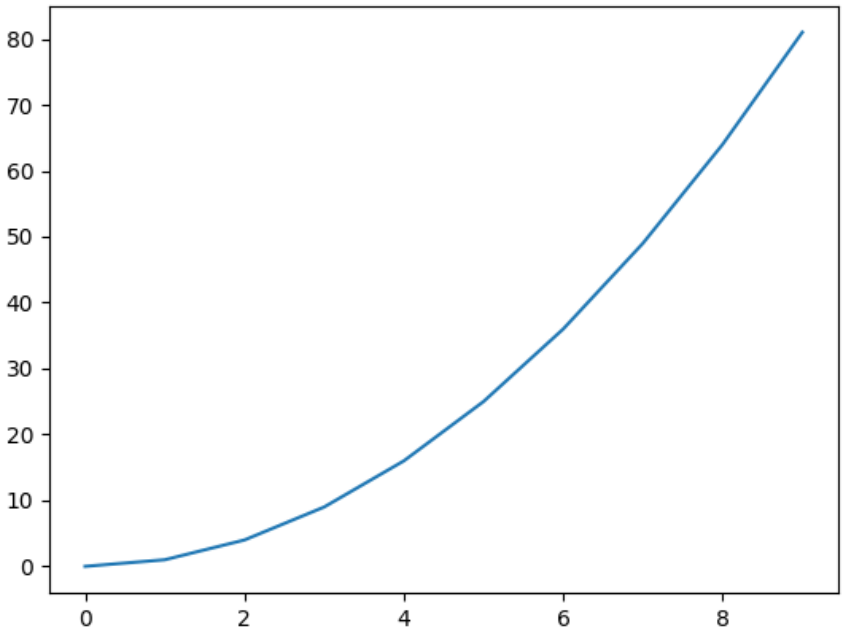


O que importa para fazer este gráfico é montar ambos os vetores, x e y, colocar os dados no matplotlib com plot, e depois mostrar com show.

O vetor Y possui os valores de x^2 para cada valor de x.

Um gráfico MUITO simples

```
import matplotlib.pyplot as plt
X = list(range(10)) #Constrói um vetor de 10 posições
Y = [x**2 for x in X]
    #Pega os valores de x^2 para cada x e coloca em Y
plt.plot(X, Y)
plt.show()
```



O que importa para fazer este gráfico é montar ambos os vetores, x e y, colocar os dados no matplotlib com plot, e depois mostrar com show.
O vetor Y possui os valores de x^2 para cada valor de x.

Sumário

Ajuste de escalas dos gráficos

O matplotlib, caso não seja comandado em contrário, estipula a escala que está sendo feito o gráfico.

Observe os gráficos abaixo à esquerda (com ajuste automático) e à direita, com escala de mesma razão.


```
import matplotlib.pyplot as plt
import numpy as np
X = np.linspace(0,2,1000)
Y = [x**2 for x in X]
ax = plt.axes()
ax.set_aspect(0.4)
#ax.set_aspect(1)
plt.plot(X, Y)
plt.title("Eixos em razão 0.4")
#plt.title("Eixos em razão 1:1")
plt.show()
```

Caso não sejam inseridas as linhas `ax = plt.axes()` e `ax.set_aspect(0.4)` teremos uma escolha automática de razão de escala entre os gráficos. Neste exemplo, temos que a razão da escala da altura e do comprimento é 0,4. Fazendo `ax.set_aspect(2)` a escala de altura seria duas vezes maior que a escala de comprimento.

Ajuste de escalas dos gráficos

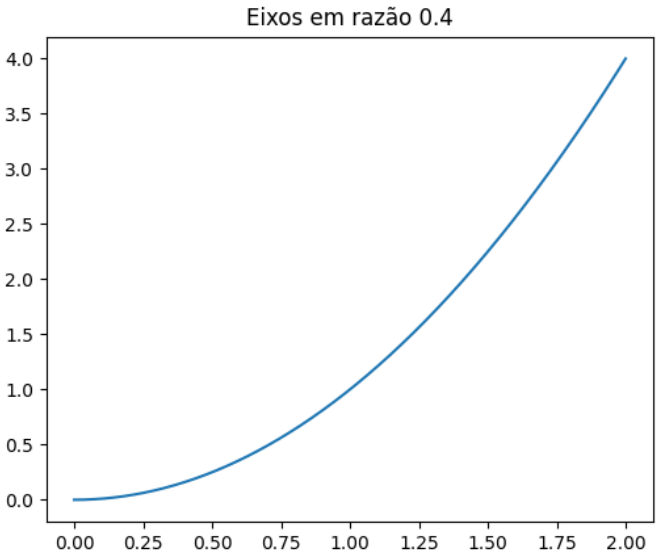
O matplotlib, caso não seja comandado em contrário, estipula a escala que está sendo feito o gráfico.
Observe os gráficos abaixo à esquerda (com ajuste automático) e à direita, com escala de mesma razão.

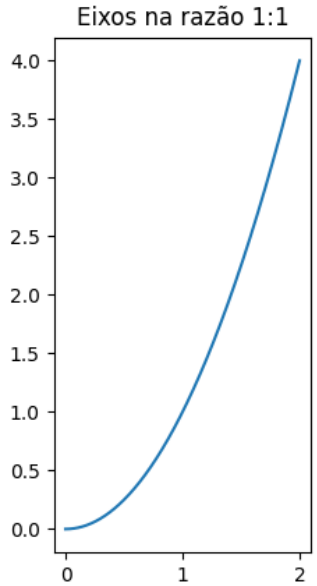

```
import matplotlib.pyplot as plt
import numpy as np
X = np.linspace(0,2,1000)
Y = [x**2 for x in X]
ax = plt.axes()
ax.set_aspect(0.4)
#ax.set_aspect(1)
plt.plot(X, Y)
plt.title("Eixos em razão 0.4")
#plt.title("Eixos em razão 1:1")
plt.show()
```

Caso não sejam inseridas as linhas `ax = plt.axes()` e `ax.set_aspect(0.4)` teremos uma escolha automática de razão de escala entre os gráficos. Neste exemplo, temos que a razão da escala da altura e do comprimento é 0,4. Fazendo `ax.set_aspect(2)` a escala de altura seria duas vezes maior que a escala de comprimento.

Ajuste de escalas dos gráficos

O matplotlib, caso não seja comandado em contrário, estipula a escala que está sendo feito o gráfico.
Observe os gráficos abaixo à esquerda (com ajuste automático) e à direita, com escala de mesma razão.



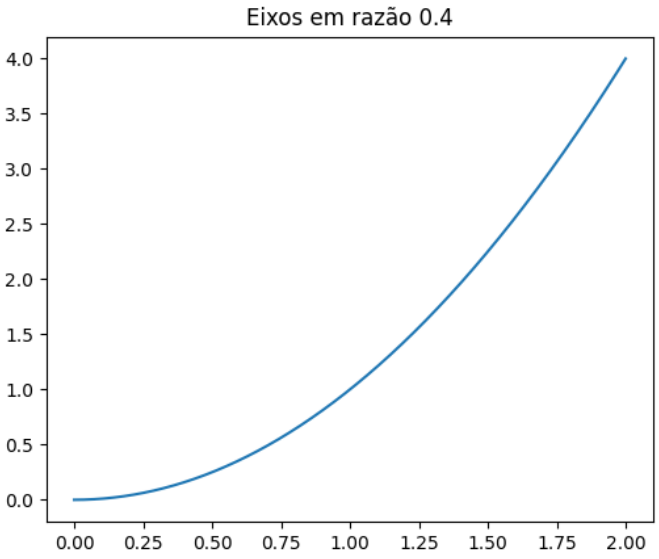


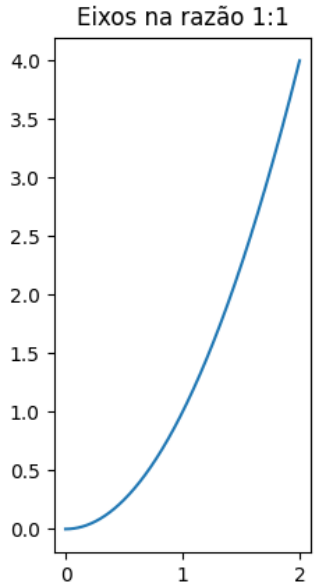

```
import matplotlib.pyplot as plt
import numpy as np
X = np.linspace(0,2,1000)
Y = [x**2 for x in X]
ax = plt.axes()
ax.set_aspect(0.4)
#ax.set_aspect(1)
plt.plot(X, Y)
plt.title("Eixos em razão 0.4")
#plt.title("Eixos em razão 1:1")
plt.show()
```

Caso não sejam inseridas as linhas `ax = plt.axes()` e `ax.set_aspect(0.4)` teremos uma escolha automática de razão de escala entre os gráficos. Neste exemplo, temos que a razão da escala da altura e do comprimento é 0,4. Fazendo `ax.set_aspect(2)` a escala de altura seria duas vezes maior que a escala de comprimento.

Ajuste de escalas dos gráficos

O matplotlib, caso não seja comandado em contrário, estipula a escala que está sendo feito o gráfico.
Observe os gráficos abaixo à esquerda (com ajuste automático) e à direita, com escala de mesma razão.





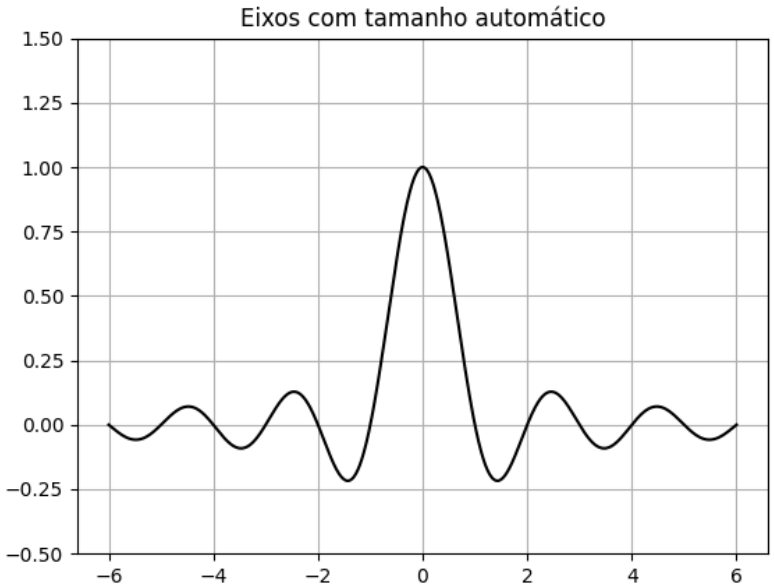

```
import matplotlib.pyplot as plt
import numpy as np
X = np.linspace(0,2,1000)
Y = [x**2 for x in X]
ax = plt.axes()
ax.set_aspect(0.4)
#ax.set_aspect(1)
plt.plot(X, Y)
plt.title("Eixos em razão 0.4")
#plt.title("Eixos em razão 1:1")
plt.show()
```

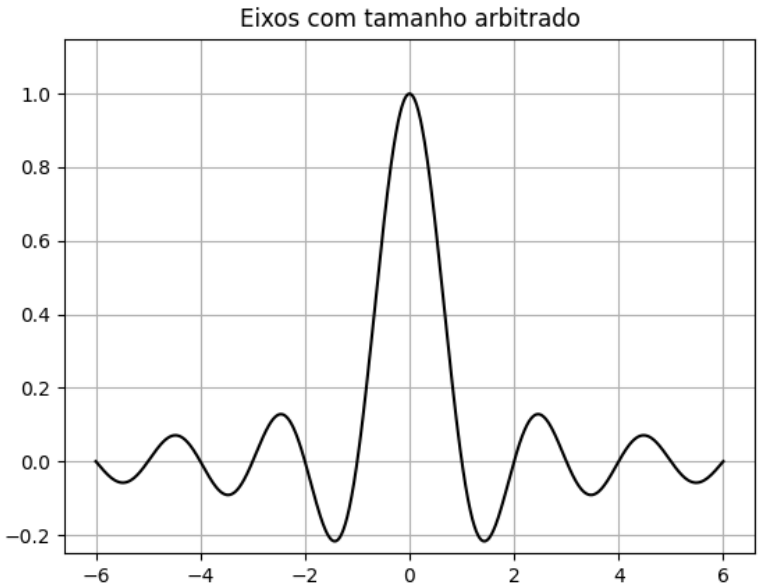
Caso não sejam inseridas as linhas `ax = plt.axes()` e `ax.set_aspect(0.4)` teremos uma escolha automática de razão de escala entre os gráficos. Neste exemplo, temos que a razão da escala da altura e do comprimento é 0,4. Fazendo `ax.set_aspect(2)` a escala de altura seria duas vezes maior que a escala de comprimento.

Sumário

Limitação dos eixos dos gráficos

Podemos limitar o tamanho dos eixos dos gráficos.





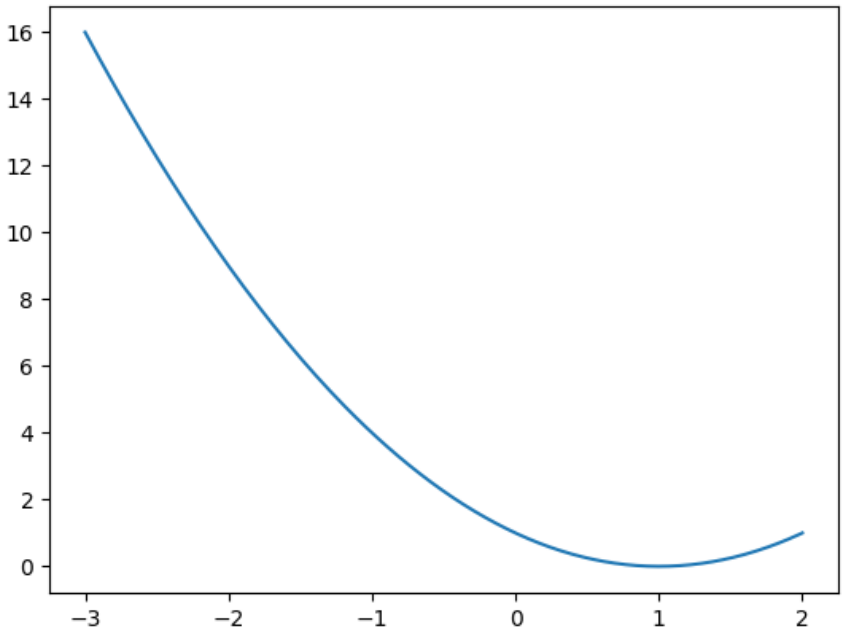

```
import numpy as np
import matplotlib.pyplot as plt
X = np.linspace(-6, 6, 1024)
plt.ylim(-.3, 1.2)
plt.plot(X, np.sinc(X), c = 'k')
plt.ylim(-.25, 1.15)
plt.title("Eixos com tamanho arbitrado")
plt.grid()
plt.show()
```

A utilização da linha `plt.ylim(-.25, 1.15)` limita o eixo *y* do gráfico. Poderíamos, mesmo tendo definido x entre −6 e 6, limitado o eixo *x* entre −5 e 5 ou quaisquer outros valores adicionando a linha `plt.xlim(-5,5)`, por exemplo.

Sumário

Usando numpy

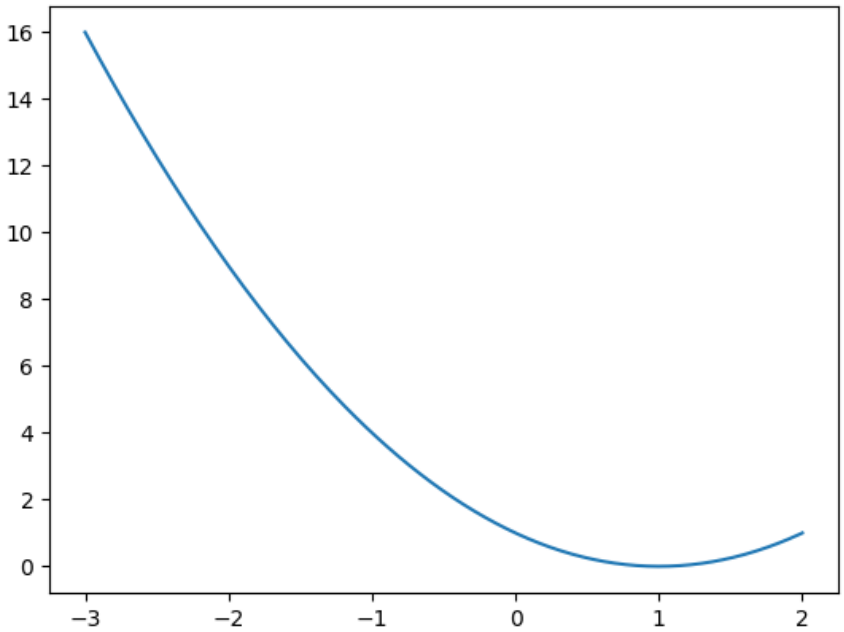
```
import numpy as np
import matplotlib.pyplot as plt
X = np.linspace(-3, 2, 200)
Y = X ** 2 - 2 * X + 1.
plt.plot(X, Y)
plt.show()
```



Definindo o vetor pelo numpy pode-se fazer operações de forma direta, sem precisar agir em cada posição do vetor *Y* diretamente. A função linspace do numpy criou um vetor de 200 pontos, variando de -3 até 2, inclusive.

Usando numpy

```
import numpy as np
import matplotlib.pyplot as plt
X = np.linspace(-3, 2, 200)
Y = X ** 2 - 2 * X + 1.
plt.plot(X, Y)
plt.show()
```

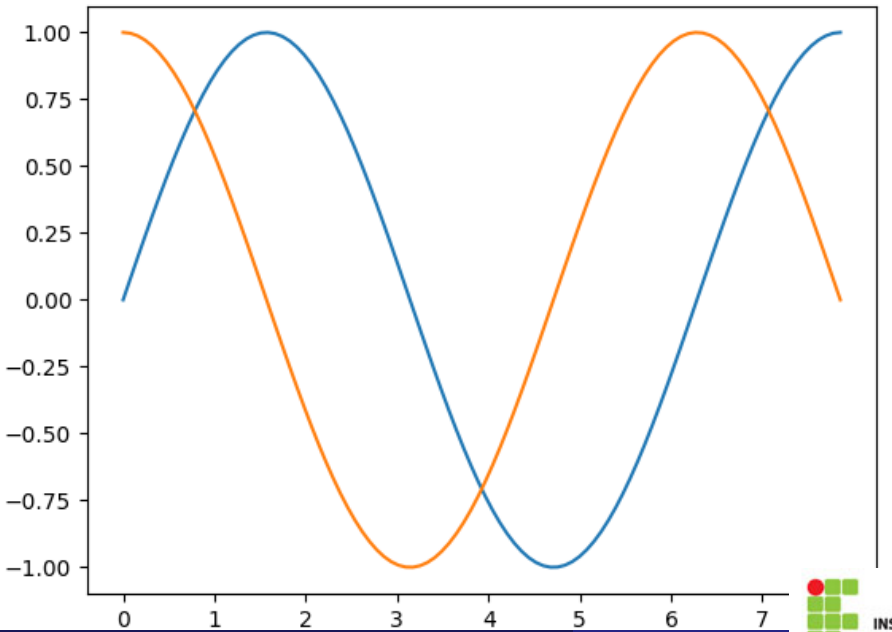


Definindo o vetor pelo numpy pode-se fazer operações de forma direta, sem precisar agir em cada posição do vetor *Y* diretamente. A função linspace do numpy criou um vetor de 200 pontos, variando de -3 até 2, inclusive.

Sumário

Gráficos simultâneos

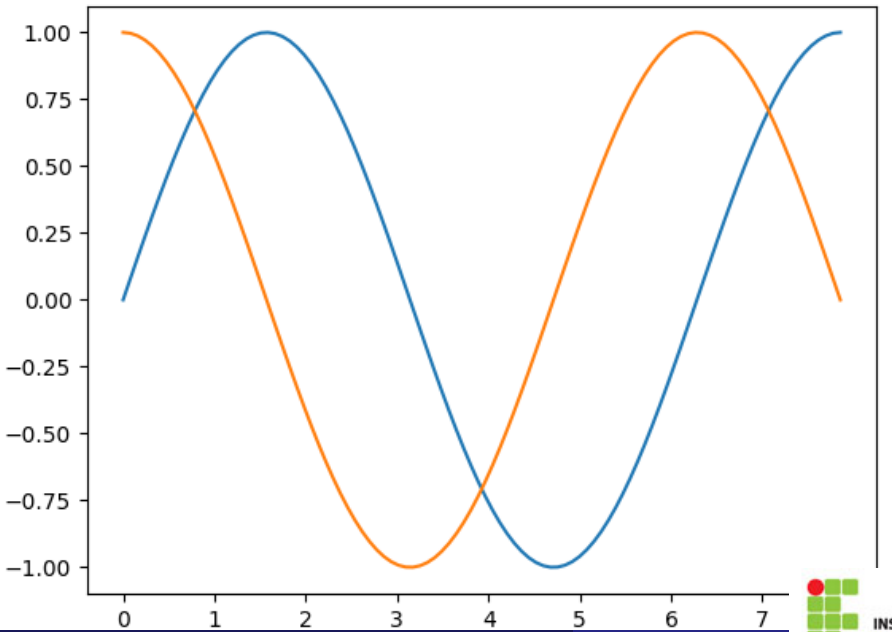
```
import numpy as np
import matplotlib.pyplot as plt
X = np.linspace(0, 2.5 * np.pi, 100)
Ya = np.sin(X)
Yb = np.cos(X)
plt.plot(X, Ya)
plt.plot(X, Yb)
plt.show()
```



Aqui se mostra melhor como funciona o matplotlib. No caso enviamos dois gráficos para o matplotlib, e pedimos para mostrar. Poderíamos ter mandado mais gráficos, tantos quanto forem necessários.

Gráficos simultâneos

```
import numpy as np
import matplotlib.pyplot as plt
X = np.linspace(0, 2.5 * np.pi, 100)
Ya = np.sin(X)
Yb = np.cos(X)
plt.plot(X, Ya)
plt.plot(X, Yb)
plt.show()
```



Aqui se mostra melhor como funciona o matplotlib. No caso enviamos dois gráficos para o matplotlib, e pedimos para mostrar. Poderíamos ter mandado mais gráficos, tantos quanto forem necessários.

Sumário

Gráficos criados em funções

```
import numpy as np
import matplotlib.pyplot as plt
plt.clf()
def plot_extra(X, Y):
    Xs = X[1:] - X[:-1]
    Ys = Y[1:] - Y[:-1]
    plt.plot(X[1:], Ys / Xs)
X = np.linspace(-3, 3, 150)
Y = np.exp(-X ** 2)
plt.plot(X, Y)
plot_extra(X, Y)
plt.show()
```

Observe que foi criada uma função que insere no matplotlib uma informação de gráfico, mas sem mostrar o gráfico.
Não precisa usar return para incluir o gráfico. O matplotlib fica ativo o tempo todo. Por isso, caso se queira fazer vários gráficos num só script recomenda-se limpar o buffer usando `plt.clf()`.
Experimente colocar a linha `plt.clf()` depois de `plt.plot(X,Y)` para entender.

Utilizaram-se os cortes de vetores do python. No caso $X[1:]$ é um vetor que inicia na posição 1 de X e termina onde X termina, logo tem uma posição a menos que X. O vetor $X[:-1]$ também tem uma posição a menos que X, mas começa na posição 0 e termina na posição $len(X) - 1$.

Gráficos criados em funções

```
import numpy as np
import matplotlib.pyplot as plt
plt.clf()
def plot_extra(X, Y):
    Xs = X[1:] - X[:-1]
    Ys = Y[1:] - Y[:-1]
    plt.plot(X[1:], Ys / Xs)
X = np.linspace(-3, 3, 150)
Y = np.exp(-X ** 2)
plt.plot(X, Y)
plot_extra(X, Y)
plt.show()
```

Observe que foi criada uma função que insere no matplotlib uma informação de gráfico, mas sem mostrar o gráfico.

Não precisa usar return para incluir o gráfico. O matplotlib fica ativo o tempo todo. Por isso, caso se queira fazer vários gráficos num só script recomenda-se limpar o buffer usando `plt.clf()`.

Experimente colocar a linha `plt.clf()` depois de `plt.plot(X,Y)` para entender.

Utilizaram-se os cortes de vetores do python. No caso $X[1:]$ é um vetor que inicia na posição 1 de X e termina onde X termina, logo tem uma posição a menos que X. O vetor $X[:-1]$ também tem uma posição a menos que X, mas começa na posição 0 e termina na posição $len(X) - 1$.

Gráficos criados em funções

```
import numpy as np
import matplotlib.pyplot as plt
plt.clf()
def plot_extra(X, Y):
    Xs = X[1:] - X[:-1]
    Ys = Y[1:] - Y[:-1]
    plt.plot(X[1:], Ys / Xs)
X = np.linspace(-3, 3, 150)
Y = np.exp(-X ** 2)
plt.plot(X, Y)
plot_extra(X, Y)
plt.show()
```

Observe que foi criada uma função que insere no matplotlib uma informação de gráfico, mas sem mostrar o gráfico.
Não precisa usar return para incluir o gráfico. O matplotlib fica ativo o tempo todo. Por isso, caso se queira fazer vários gráficos num só script recomenda-se limpar o buffer usando `plt.clf()`.
Experimente colocar a linha `plt.clf()` depois de `plt.plot(X,Y)` para entender.

Utilizaram-se os cortes de vetores do python. No caso $X[1:]$ é um vetor que inicia na posição 1 de X e termina onde X termina, logo tem uma posição a menos que X. O vetor $X[:-1]$ também tem uma posição a menos que X, mas começa na posição 0 e termina na posição $len(X) - 1$.

Gráficos criados em funções

```
import numpy as np
import matplotlib.pyplot as plt
plt.clf()
def plot_extra(X, Y):
    Xs = X[1:] - X[:-1]
    Ys = Y[1:] - Y[:-1]
    plt.plot(X[1:], Ys / Xs)
X = np.linspace(-3, 3, 150)
Y = np.exp(-X ** 2)
plt.plot(X, Y)
plot_extra(X, Y)
plt.show()
```

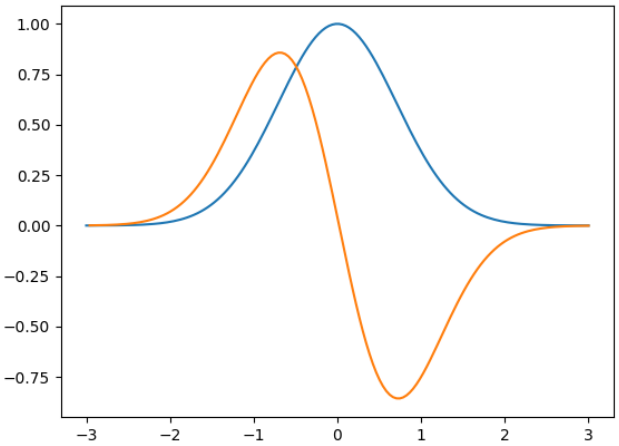
Observe que foi criada uma função que insere no matplotlib uma informação de gráfico, mas sem mostrar o gráfico.
Não precisa usar return para incluir o gráfico. O matplotlib fica ativo o tempo todo. Por isso, caso se queira fazer vários gráficos num só script recomenda-se limpar o buffer usando `plt.clf()`.
Experimente colocar a linha `plt.clf()` depois de `plt.plot(X,Y)` para entender.

Utilizaram-se os cortes de vetores do python. No caso $X[1:]$ é um vetor que inicia na posição 1 de X e termina onde X termina, logo tem uma posição a menos que X. O vetor $X[:-1]$ também tem uma posição a menos que X, mas começa na posição 0 e termina na posição $len(X) - 1$.

Gráficos criados em funções

```
import numpy as np
import matplotlib.pyplot as plt
plt.clf()
def plot_extra(X, Y):
    Xs = X[1:] - X[:-1]
    Ys = Y[1:] - Y[:-1]
    plt.plot(X[1:], Ys / Xs)
X = np.linspace(-3, 3, 150)
Y = np.exp(-X ** 2)
plt.plot(X, Y)
plot_extra(X, Y)
plt.show()
```

Observe que foi criada uma função que insere no matplotlib uma informação de gráfico, mas sem mostrar o gráfico.
Não precisa usar return para incluir o gráfico. O matplotlib fica ativo o tempo todo. Por isso, caso se queira fazer vários gráficos num só script recomenda-se limpar o buffer usando `plt.clf()`.
Experimente colocar a linha `plt.clf()` depois de `plt.plot(X,Y)` para entender.

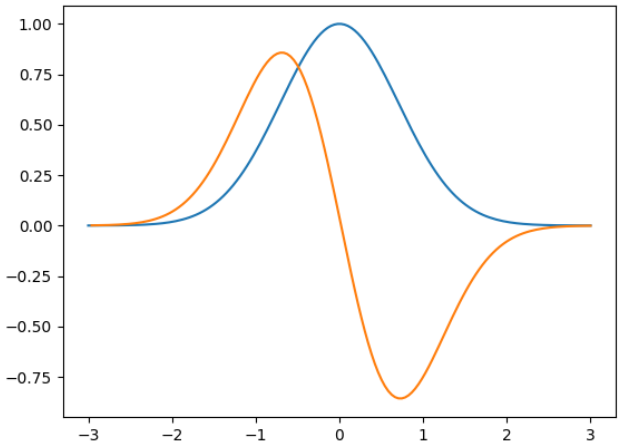


Utilizaram-se os cortes de vetores do python. No caso $X[1:]$ é um vetor que inicia na posição 1 de X e termina onde X termina, logo tem uma posição a menos que X. O vetor $X[:-1]$ também tem uma posição a menos que X, mas começa na posição 0 e termina na posição $len(X) - 1$.

Gráficos criados em funções

```
import numpy as np
import matplotlib.pyplot as plt
plt.clf()
def plot_extra(X, Y):
    Xs = X[1:] - X[:-1]
    Ys = Y[1:] - Y[:-1]
    plt.plot(X[1:], Ys / Xs)
X = np.linspace(-3, 3, 150)
Y = np.exp(-X ** 2)
plt.plot(X, Y)
plot_extra(X, Y)
plt.show()
```

Observe que foi criada uma função que insere no matplotlib uma informação de gráfico, mas sem mostrar o gráfico.
Não precisa usar return para incluir o gráfico. O matplotlib fica ativo o tempo todo. Por isso, caso se queira fazer vários gráficos num só script recomenda-se limpar o buffer usando `plt.clf()`.
Experimente colocar a linha `plt.clf()` depois de `plt.plot(X,Y)` para entender.



Utilizaram-se os cortes de vetores do python. No caso $X[1:]$ é um vetor que inicia na posição 1 de X e termina onde X termina, logo tem uma posição a menos que X. O vetor $X[:-1]$ também tem uma posição a menos que X, mas começa na posição 0 e termina na posição $len(X) - 1$.

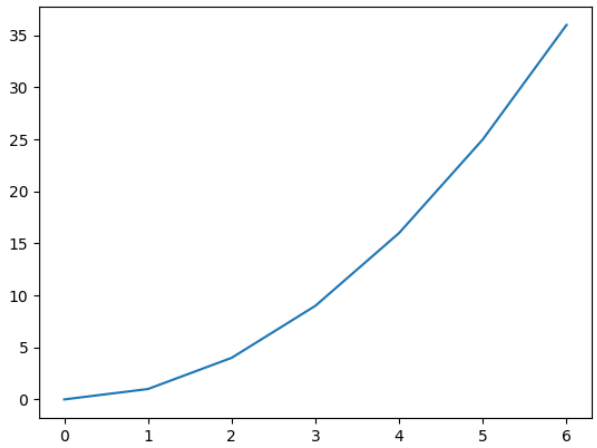
Sumário

Lendo dados em arquivos

```
import matplotlib.pyplot as plt
X, Y = [], []
for line in open('data.txt', 'r'):
    values = [float(s) for s in line.split()]
    if(len(values)):
        X.append(values[0])
        Y.append(values[1])
plt.plot(X, Y)
plt.show()
```

Arquivo data.txt

```
0 0
1 1
2 4
4 16
5 25
6 36
```



Procedimento do código.

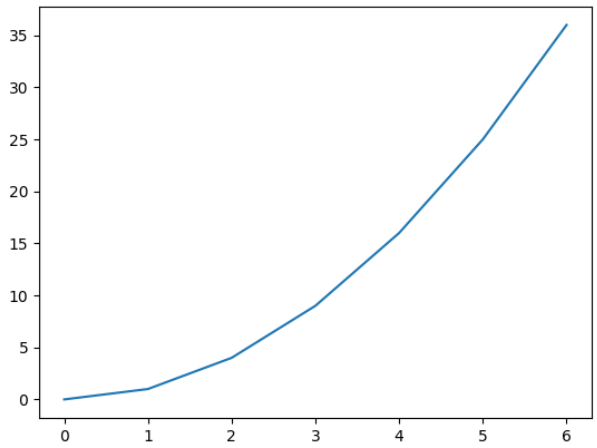
- Abrir dois vetores simultamenamente.
- No loop já definir a variavel line para cada linha do arquivo.
- values é um vetor que terá tantas colunas quantas houver em cada linha do arquivo.
- Para evitar que uma linha vazia quebre o código insere-se o if.
- Insere-se o valor da primeira coluna em X e da segunda em Y.

Lendo dados em arquivos

```
import matplotlib.pyplot as plt
X, Y = [], []
for line in open('data.txt', 'r'):
    values = [float(s) for s in line.split()]
    if(len(values)):
        X.append(values[0])
        Y.append(values[1])
plt.plot(X, Y)
plt.show()
```

Arquivo data.txt

```
0 0
1 1
2 4
4 16
5 25
6 36
```



Procedimento do código.

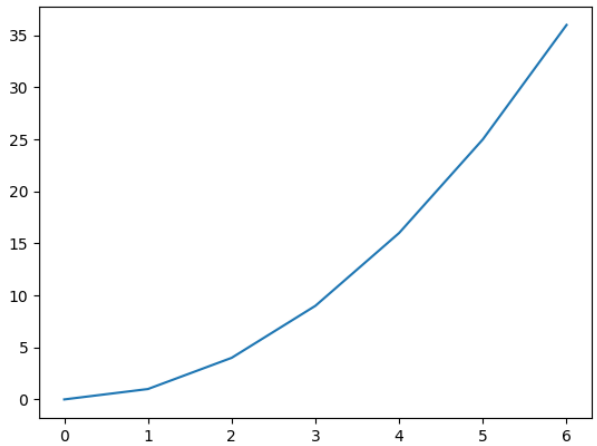
- Abrir dois vetores simultamenamente.
- No loop já definir a variavel line para cada linha do arquivo.
- values é um vetor que terá tantas colunas quantas houver em cada linha do arquivo.
- Para evitar que uma linha vazia quebre o código insere-se o if.
- Insere-se o valor da primeira coluna em X e da segunda em Y.

Lendo dados em arquivos

```
import matplotlib.pyplot as plt
X, Y = [], []
for line in open('data.txt', 'r'):
    values = [float(s) for s in line.split()]
    if(len(values)):
        X.append(values[0])
        Y.append(values[1])
plt.plot(X, Y)
plt.show()
```

Arquivo data.txt

```
0 0
1 1
2 4
4 16
5 25
6 36
```



Procedimento do código.

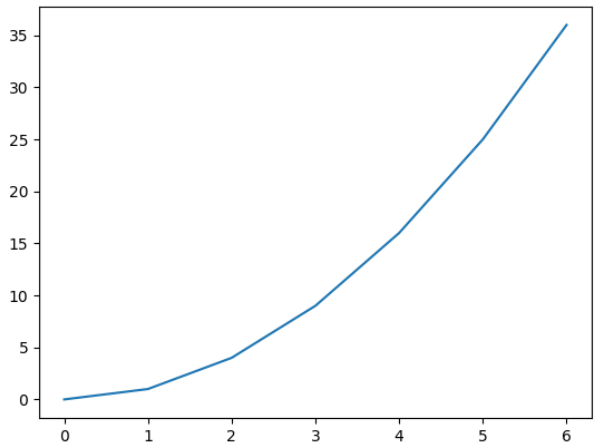
- Abrir dois vetores simultamenamente.
- No loop já definir a variavel line para cada linha do arquivo.
- values é um vetor que terá tantas colunas quantas houver em cada linha do arquivo.
- Para evitar que uma linha vazia quebre o código insere-se o if.
- Insere-se o valor da primeira coluna em X e da segunda em Y.

Lendo dados em arquivos

```
import matplotlib.pyplot as plt
X, Y = [], []
for line in open('data.txt', 'r'):
    values = [float(s) for s in line.split()]
    if(len(values)):
        X.append(values[0])
        Y.append(values[1])
plt.plot(X, Y)
plt.show()
```

Arquivo data.txt

```
0 0
1 1
2 4
4 16
5 25
6 36
```



Procedimento do código.

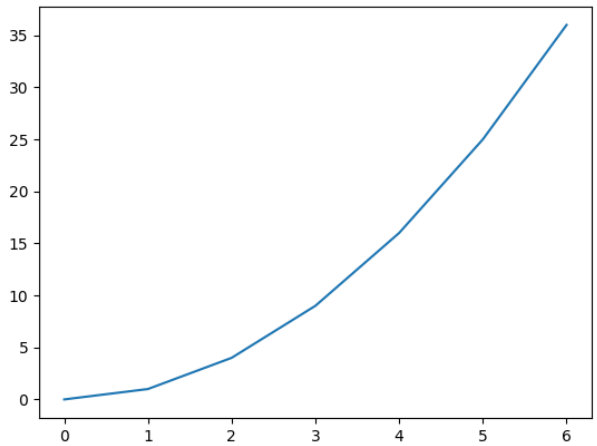
- Abrir dois vetores simultamenamente.
- No loop já definir a variavel line para cada linha do arquivo.
- values é um vetor que terá tantas colunas quantas houver em cada linha do arquivo.
- Para evitar que uma linha vazia quebre o código insere-se o if.
- Insere-se o valor da primeira coluna em X e da segunda em Y.

Lendo dados em arquivos

```
import matplotlib.pyplot as plt
X, Y = [], []
for line in open('data.txt', 'r'):
    values = [float(s) for s in line.split()]
    if(len(values)):
        X.append(values[0])
        Y.append(values[1])
plt.plot(X, Y)
plt.show()
```

Arquivo data.txt

```
0 0
1 1
2 4
4 16
5 25
6 36
```



Procedimento do código.

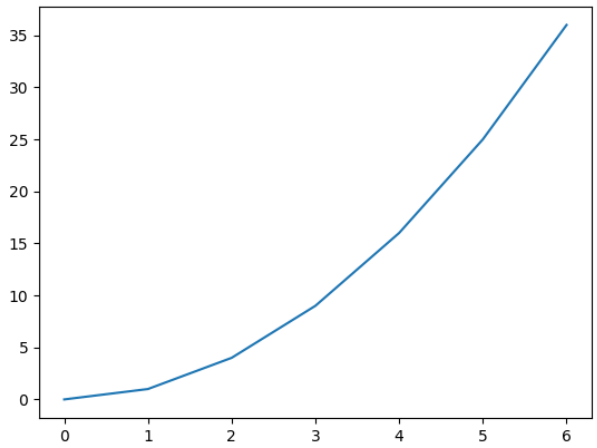
- Abrir dois vetores simultamenamente.
- No loop já definir a variavel line para cada linha do arquivo.
- values é um vetor que terá tantas colunas quantas houver em cada linha do arquivo.
- Para evitar que uma linha vazia quebre o código insere-se o if.
- Insere-se o valor da primeira coluna em X e da segunda em Y.

Lendo dados em arquivos

```
import matplotlib.pyplot as plt
X, Y = [], []
for line in open('data.txt', 'r'):
    values = [float(s) for s in line.split()]
    if(len(values)):
        X.append(values[0])
        Y.append(values[1])
plt.plot(X, Y)
plt.show()
```

Arquivo data.txt

```
0 0
1 1
2 4
4 16
5 25
6 36
```



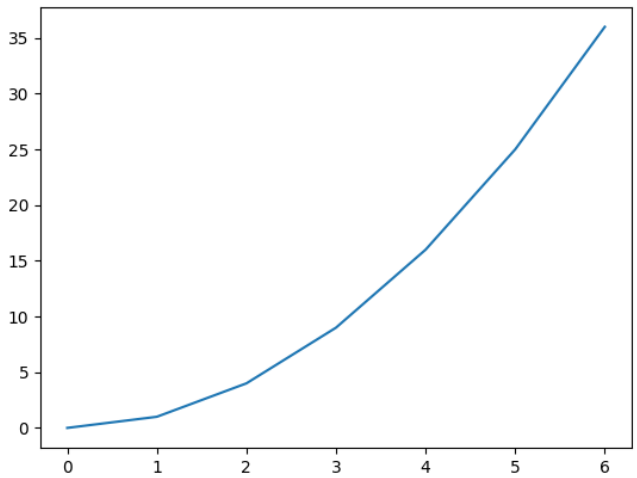
Procedimento do código.

- Abrir dois vetores simultamenamente.
- No loop já definir a variavel line para cada linha do arquivo.
- values é um vetor que terá tantas colunas quantas houver em cada linha do arquivo.
- Para evitar que uma linha vazia quebre o código insere-se o if.
- Insere-se o valor da primeira coluna em X e da segunda em Y.

Forma alternativa de leitura com numpy

```
import matplotlib.pyplot as plt
import numpy as np
data = np.loadtxt('data.txt')
plt.plot(data[:,0], data[:,1])
plt.show()
```

Neste código, usamos o comando do numpy loadtxt, que gera uma matriz com quantas colunas houver no arquivo.
O numpy já converte os valores lidos para numéricos. Usa plot para enviar ao matplotlib. Na versão anterior não se pode usar uma matriz, como mat = [] e depois inserir values, ao invés de X e Y, porque line.split() gera tuplas, que são imutáveis, e não permite slice do tipo[:,1].



Múltiplos gráficos com arquivos

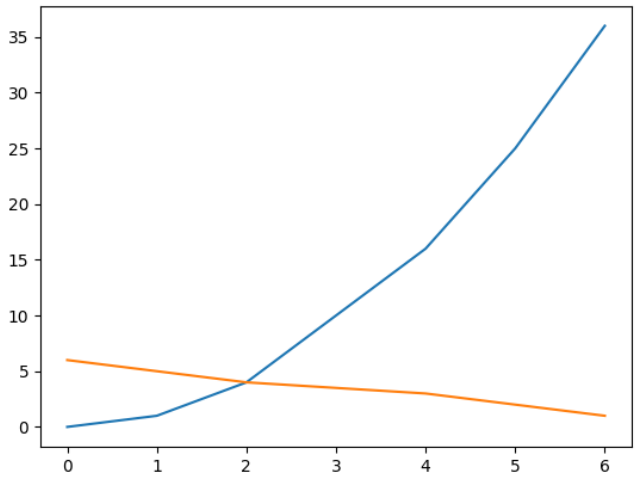
Queremos proceder à leitura do arquivo

```
0  0  6
1  1  5
2  4  4
4  16  3
5  25  2
6  36  1
```

Arquivo data2.txt

Considere que a primeira coluna sejam os valores da abscissa e as outras duas colunas as ordenadas dos gráficos simultâneos.

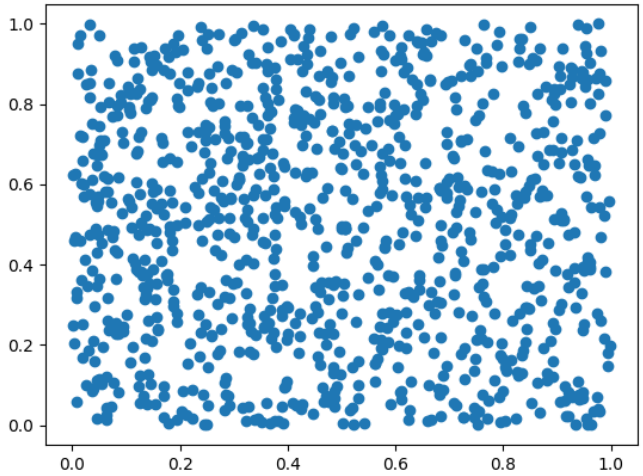
```
import numpy as np
import matplotlib.pyplot as plt
data = np.loadtxt('data2.txt')
flag = False
for column in data.T:
    if flag:
        plt.plot(data[:,0], column)
    else:
        flag = True
plt.show()
```



Sumário

Gráfico de pontos simples

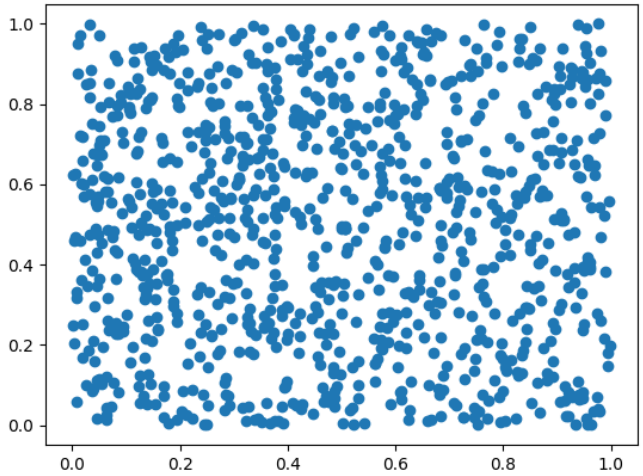
```
import numpy as np
import matplotlib.pyplot as plt
data = np.random.rand(1000, 2)
plt.scatter(data[:,0], data[:,1])
plt.show()
```



Neste exemplo usamos `random.rand(1000, 2)` para criar uma matriz $1000x2$ com números entre 0 e 1 aleatórios. Passou-se para a função scatter a primeira coluna e a segunda coluna e o resultado são os pontos em posição aleatória dentro de um quadrado de 1 por 1.

Gráfico de pontos simples

```
import numpy as np
import matplotlib.pyplot as plt
data = np.random.rand(1000, 2)
plt.scatter(data[:,0], data[:,1])
plt.show()
```

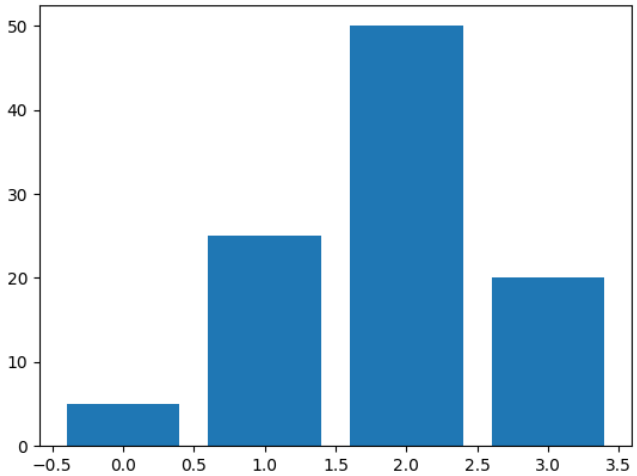


Neste exemplo usamos `random.rand(1000, 2)` para criar uma matriz $1000x2$ com números entre 0 e 1 aleatórios. Passou-se para a função scatter a primeira coluna e a segunda coluna e o resultado são os pontos em posição aleatória dentro de um quadrado de 1 por 1.

Sumário

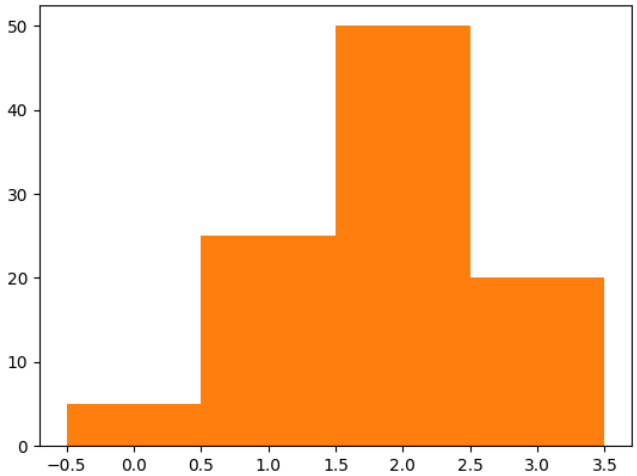
Gráfico de barras simples

```
import matplotlib.pyplot as plt
data = [5., 25., 50., 20.]
plt.bar(range(len(data)), data)
plt.show()
```



Deve-se ter em mente que range(len(data)) cria um vetor com comprimento len(data) começando em zero. Para construir barras sem espaço fazemos:

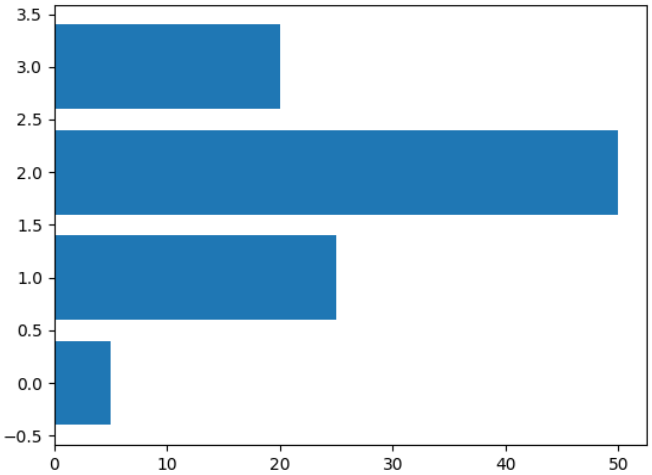
```
import matplotlib.pyplot as plt
data = [5., 25., 50., 20.]
plt.bar(range(len(data)), data, width = 1.)
plt.show()
```



Sumário

Gráfico de barras horizontais

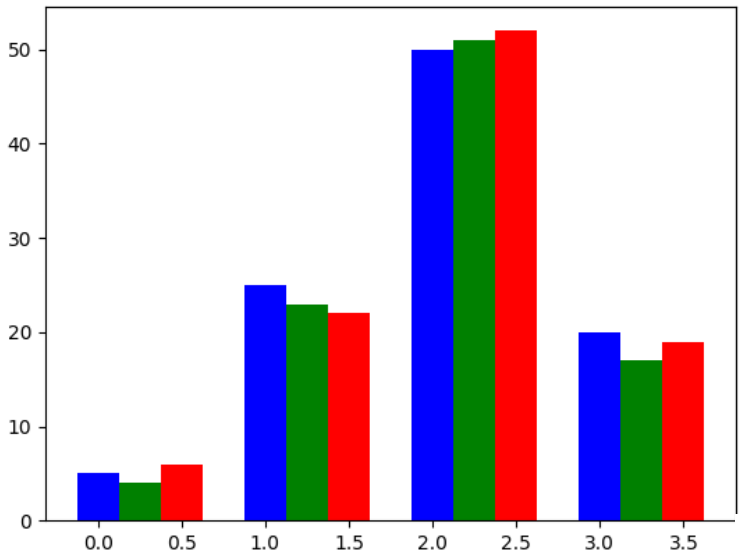
```
import matplotlib.pyplot as plt
data = [5., 25., 50., 20.]
plt.barh(range(len(data)), data)
plt.show()
```



Mesmos dados de antes, mas com histograma horizontal.

Gráfico múltiplo de barras

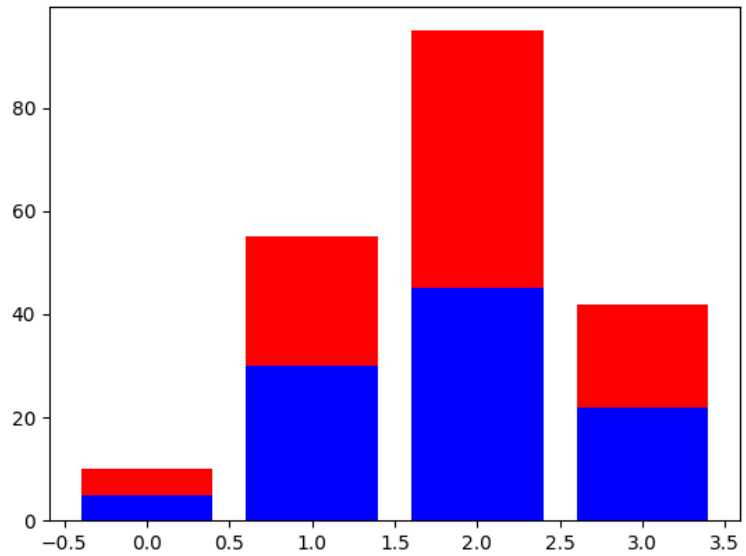
```
import numpy as np
import matplotlib.pyplot as plt
data = [[5., 25., 50., 20.],[4., 23., 51., 17.],
[6., 22., 52., 19.]]
X = np.arange(4)
plt.bar(X + 0.00, data[0], color = 'b', width = 0.25)
plt.bar(X + 0.25, data[1], color = 'g', width = 0.25)
plt.bar(X + 0.50, data[2], color = 'r', width = 0.25)
plt.show()
```



Para este caso devemos colocar a posição da barra, no primeiro argumento, o dado no segundo, a cor no terceiro, e no quarto a largura.

Gráfico de barras empilhadas

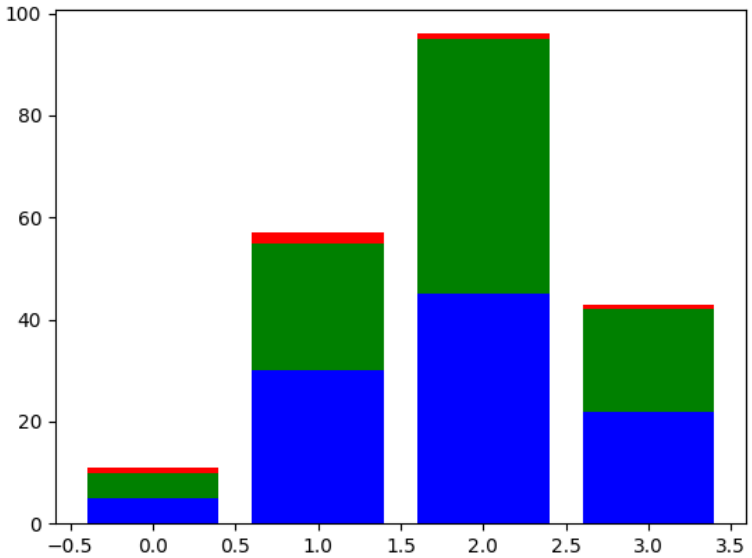
```
import matplotlib.pyplot as plt
A = [5., 30., 45., 22.]
B = [5., 25., 50., 20.]
X = range(4)
plt.bar(X, A, color = 'b')
plt.bar(X, B, color = 'r', bottom = A)
plt.show()
```



O parâmetro *bottom* é opcional e permite que se tenha um valor inicial para a barra ao invés de iniciar da altura zero.

Gráfico de múltiplas barras empilhadas

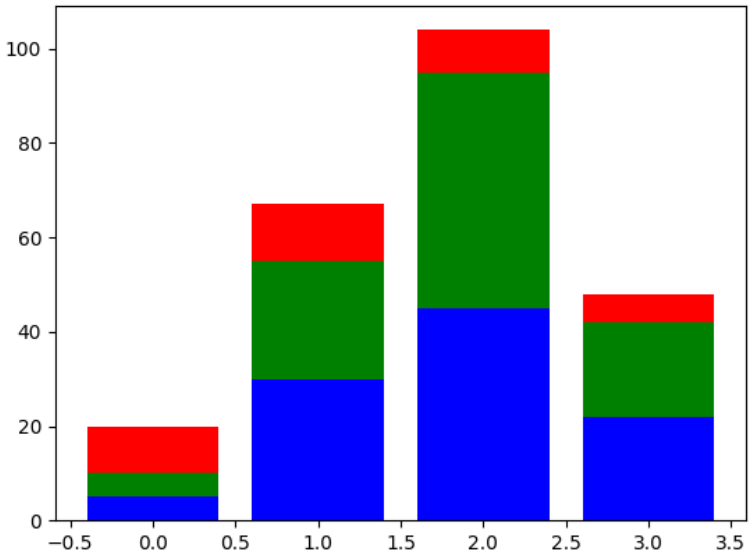
```
import numpy as np
import matplotlib.pyplot as plt
A = np.array([5., 30., 45., 22.])
B = np.array([5., 25., 50., 20.])
C = np.array([1., 2., 1., 1.])
X = np.arange(4)
plt.bar(X, A, color = 'b')
plt.bar(X, B, color = 'g', bottom = A)
plt.bar(X, C, color = 'r', bottom = A + B)
plt.show()
```



Para computar uma terceira barra empilhada precisamos usar $A + B$. O problema deste código é que funciona somente para três barras empilhadas.

Gráfico de múltiplas barras empilhadas

```
import numpy as np
import matplotlib.pyplot as plt
data = np.array([[5., 30., 45., 22.],
    [5., 25., 50., 20.],
    [10., 12., 9., 6.]])
color_list = ['b', 'g', 'r']
X = np.arange(data.shape[1])
for i in range(data.shape[0]):
    plt.bar(X, data[i],bottom = np.sum(data[:i], axis = 0),
        color = color_list[i % len(color_list)])
plt.show()
```



Para este gráfico usamos um array do numpy onde cada linha representa uma sequência da barra. Poderíamos assim colocar uma pilha maior.

Sumário

Histogramas

Histogramas são gráficos de barra em que as barras siginificam frequências.
Podemos simplesmente fazer todas as contas e criar um gráfico de barras, ou deixar que o matplotlib faça este trabalho.
No exemplo foi criado um vetor aleatório com 10000 pontos com valores da normal padronizada, ou seja, com $\sigma = 1$ e $\mu = 0$, e foi indicado o número de classes que se deseja, no caso 200. O matplotlib ordena e dispõe os dados nas classes correspondentes.
Mais dados (como frequência relativa, acumulada, pesos etc.) em:
`https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hist.html`


```
import numpy as np
import matplotlib.pyplot as plt
X = np.random.randn(10000)
plt.hist(X, bins = 200)
plt.show()
```

Histogramas

Histogramas são gráficos de barra em que as barras siginificam frequências. Podemos simplesmente fazer todas as contas e criar um gráfico de barras, ou deixar que o matplotlib faça este trabalho.

No exemplo foi criado um vetor aleatório com 10000 pontos com valores da normal padronizada, ou seja, com $\sigma = 1$ e $\mu = 0$, e foi indicado o número de classes que se deseja, no caso 200. O matplotlib ordena e dispõe os dados nas classes correspondentes.

Mais dados (como frequência relativa, acumulada, pesos etc.) em:

`https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hist.html`


```
import numpy as np
import matplotlib.pyplot as plt
X = np.random.randn(10000)
plt.hist(X, bins = 200)
plt.show()
```

Histogramas

Histogramas são gráficos de barra em que as barras siginificam frequências. Podemos simplesmente fazer todas as contas e criar um gráfico de barras, ou deixar que o matplotlib faça este trabalho.
No exemplo foi criado um vetor aleatório com 10000 pontos com valores da normal padronizada, ou seja, com $\sigma = 1$ e $\mu = 0$, e foi indicado o número de classes que se deseja, no caso 200. O matplotlib ordena e dispõe os dados nas classes correspondentes.
Mais dados (como frequência relativa, acumulada, pesos etc.) em:

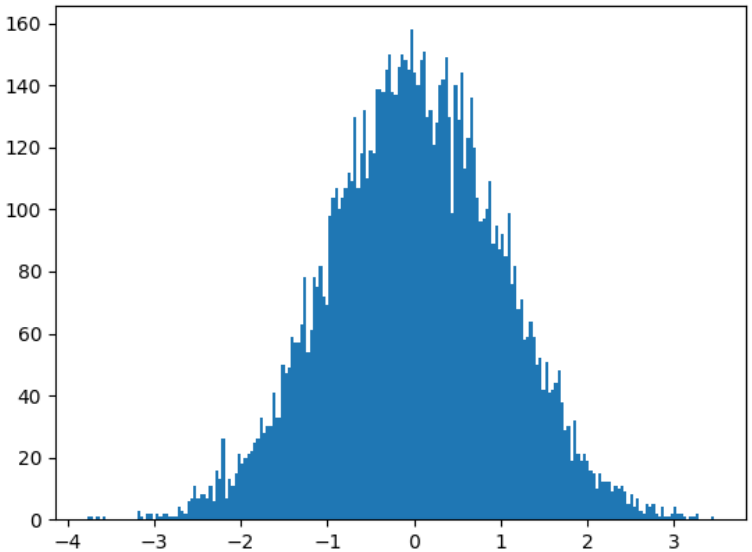
`https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hist.html`


```
import numpy as np
import matplotlib.pyplot as plt
X = np.random.randn(10000)
plt.hist(X, bins = 200)
plt.show()
```

Histogramas

Histogramas são gráficos de barra em que as barras siginificam frequências. Podemos simplesmente fazer todas as contas e criar um gráfico de barras, ou deixar que o matplotlib faça este trabalho.
No exemplo foi criado um vetor aleatório com 10000 pontos com valores da normal padronizada, ou seja, com $\sigma = 1$ e $\mu = 0$, e foi indicado o número de classes que se deseja, no caso 200. O matplotlib ordena e dispõe os dados nas classes correspondentes.
Mais dados (como frequência relativa, acumulada, pesos etc.) em:

`https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hist.html`

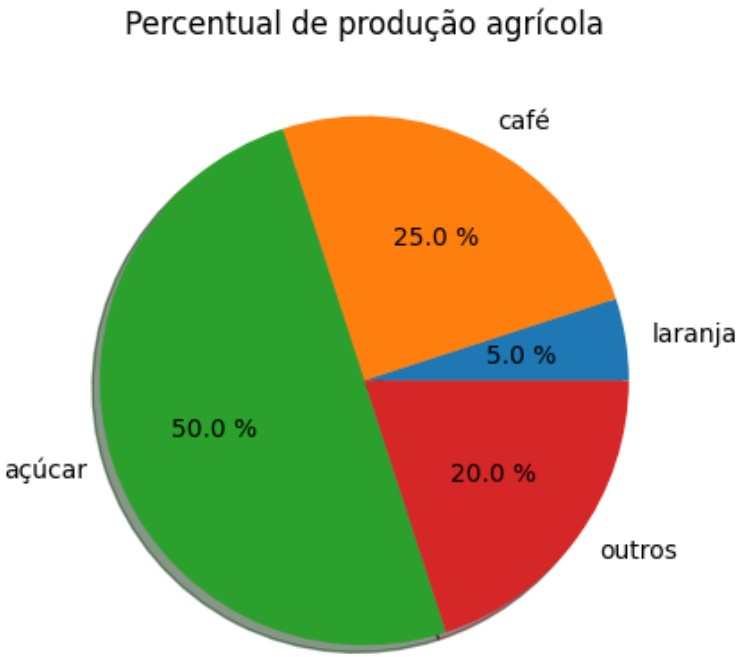



```
import numpy as np
import matplotlib.pyplot as plt
X = np.random.randn(10000)
plt.hist(X, bins = 200)
plt.show()
```

Sumário

Gráfico de pizza simples

Geralmente gráficos de pizza servem para comparar quantidades, utiliza a função pyplot.pie().




```
import matplotlib.pyplot as plt
produtos = ['laranja', 'café', 'açúcar', 'outros']
quantidade = [5, 25, 50, 20]
plt.pie(quantidade, labels= produtos, radius = 1,
    autopct = "%.1f %%", frame = False,
    shadow = True, normalize = True,
    rotatelabels = False)
plt.title("Percentual de produção agrícola")
plt.show()
```