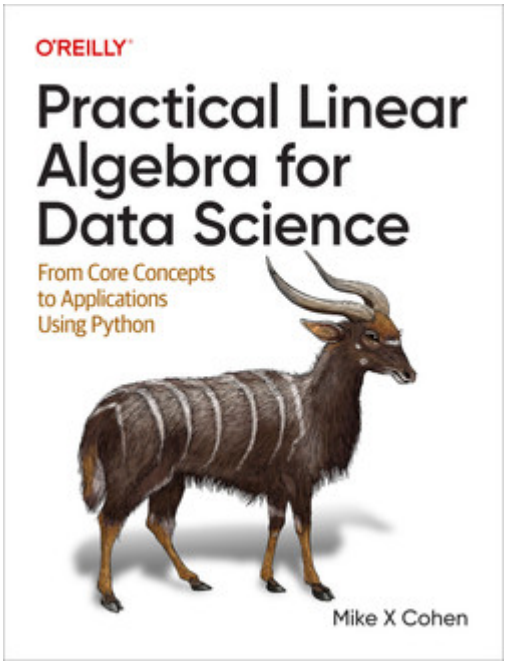
Matrizes - Parte 1



Uma matriz é um vetor levado ao próximo nível.

Matrizes são objetos matemáticos altamente versáteis. Elas podem guardar conjuntos de equações, transformações geométricas, as posições de partículas ao longo do tempo, registros financeiros e inúmeras outras coisas.

Em ciência de dados, algumas vezes as matrizes são chamadas de tabelas de dados, cujas linhas representam **observações** e as colunas as **variáveis**.

Bibliotecas

In [5]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import toeplitz
```

Criando e Visualizando Matrizes no NumPy

Dependendo do contexto, matrizes podem ser explicadas como um conjunto de vetores em colunas empilhadas uma ao lado da outra, ou linhas em camadas sobrepostas uma acima da outra, ou uma coleção ordenada de elementos individuais da matriz.

In [4]:
```
# Criando uma matriz 3x3 com valores inteiros
matriz = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(matriz)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

Visualizando, Indexando e Cortando Matrizes

$$\begin{bmatrix} 1 & 2 \\ \pi & 4 \\ 6 & 7 \end{bmatrix}, \begin{bmatrix} -6 & \frac{1}{3} \\ e^{4,3} & -1,4 \\ \frac{6}{5} & 0 \end{bmatrix}$$

Matrizes pequenas podem ser visualizadas inteiramente. Porém, na prática, muitas vezes elas serão muito maiores.

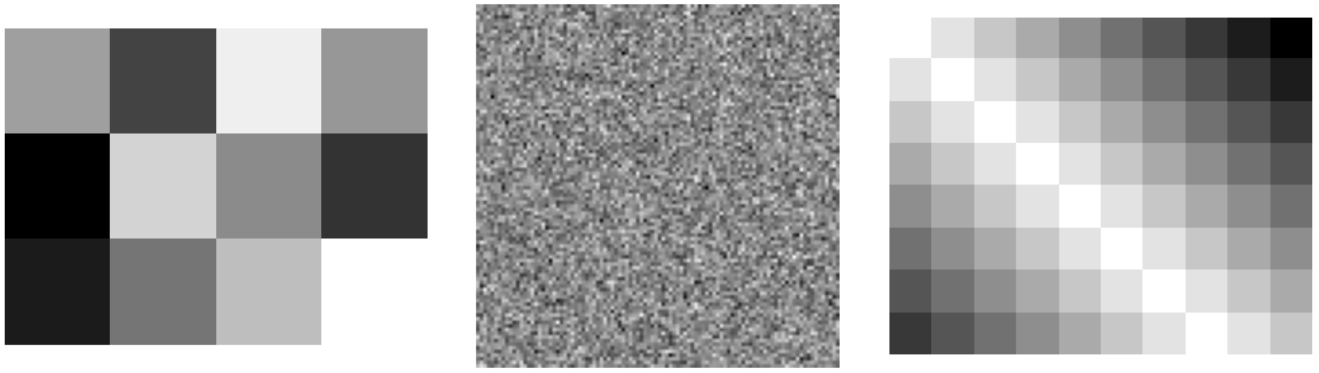
Portanto, elas podem ser representadas como imagens, onde os valores numéricos da matriz são substituídos por cores na imagem.

In [7]:
```
# Cria algumas matrizes
A = np.random.randn(3,4)
B = np.random.randn(100,100)
C = -toeplitz(np.arange(8),np.arange(10))

# E mostra elas como imagens
fig,axs = plt.subplots(1,3,figsize=(10,3))

axs[0].imshow(A,cmap='gray')
axs[1].imshow(B,cmap='gray')
axs[2].imshow(C,cmap='gray')

for i in range(3): axs[i].axis('off')
plt.tight_layout()
plt.show()
```



Matrizes são representadas por letras maiusculas em negrito, como **A** e **M**. O tamanho da matriz é indicado usando uma convenção de (linhas, colunas).

$$\boldsymbol{M} = \begin{bmatrix} 1 & 3 & 5 & 7 \\ 0 & 2 & 4 & 6 \\ 1 & 4 & 7 & 8 \end{bmatrix}$$

$$\text{A matriz M é uma matriz de tamanho } (3, 4) \text{ porque contém 3 linhas e 4 colunas.}$$

Você pode se referir a certo elemento da matriz pela posição do seu índice, por exemplo, o elemento na linha 2 e coluna 3 é representado por **M**(2, 3). Assim, **M**(2, 3) = 4.

No Python a indexação começa em 0, então o elemento na linha 2 e coluna 3 é representado por **M**[1, 2]. Assim, **M**[1, 2] = 4.

Extrair um subconjunto de linhas ou colunas de uma matriz é chamado de "fatiar" a matriz. Você pode fatiar uma matriz usando dois pontos (:) para indicar o intervalo de linhas ou colunas que você deseja extrair.

In [16]:
```
# Criando uma matriz 4x4
matriz = np.array([[10,  20,  30,  40 ],
                   [50,  60,  70,  80 ],
                   [90,  100, 110, 120],
                   [130, 140, 150, 160]])

# Fatiando a matriz: extraindo as linhas 1, 2 e as primeiras 3 colunas
fatiada = matriz[1:3, :3]

print('Matriz original:')
print(matriz)
print('')
print('Fatia da matriz:')
print(fatiada)
```

```
Matriz original:
[[ 10  20  30  40]
 [ 50  60  70  80]
 [ 90 100 110 120]
 [130 140 150 160]]

Fatia da matriz:
[[ 50  60  70]
 [ 90 100 110]]
```

Matrizes Especiais

As matrizes podem ser descritas usando uma quantidade relativamente pequena de características que criam "familias" ou categorias de matrizes. Essas categorias são importantes de conhecer porque elas aparecem em certas operações ou possuem certas propriedades específicas.

Matriz de números aleatórios

Essa é uma matriz que contém números aleatórios de alguma distribuição, normalmente a Gaussiana (conhecida tamém como Normal). Elas são úteis para explorar a algebra linear.

In [17]:
```
Mrows = 4
Ncols = 6

matriz = np.random.randn(Mrows, Ncols)
print('Matriz:')
print(matriz)
```

```
Matriz:
[[ 3.231923   -0.92424329  0.60807658  0.17821194 -0.30733498  1.21698037]
 [ 0.30868718  0.0108837   0.18840939 -0.75088801  0.2373341  -0.49358183]
 [ 0.04571863 -0.31281957  0.10859314  0.90486056 -1.41917678 -0.79975965]
 [-0.52468128 -0.96588907  0.16121462 -0.68795278  0.30199666 -0.42567723]]
```

Quadrada X Não-Quadrada (Retangular)

Uma matriz quadrada possui o mesmo número de linhas e colunas ($\mathbb{R}^{N\times N}$). Já uma matriz não-quadrada, ou retângular, tem um número diferente de linhas e colunas. Por exemplo, uma matriz 4×6 possui 4 linhas e 6 colunas.

Você pode criar matrizes quadradas ou não-quadradas ajustando os valores de `Mrows` e

`Ncols` no código anterior.

In [19]:
```
# Criando uma matriz quadrada 3x3
matriz_quadrada = np.random.randint(1, 10, size=(3, 3))
print("Matriz Quadrada:")
print(matriz_quadrada)

# Criando uma matriz retangular 3x6
matriz_retangular = np.random.randint(1, 10, size=(3, 6))
print("\nMatriz Retangular:")
print(matriz_retangular)
```

```
Matriz Quadrada:
[[6 6 5]
 [1 6 4]
 [7 4 5]]

Matriz Retangular:
[[4 5 2 4 8 6]
 [3 7 5 9 8 3]
 [8 5 2 6 8 6]]
```

Matrizes retangulares são chamadas de altas (tall) se tiverem mais linhas que colunas, ou largas (wide) se for ao contrário.

In [20]:
```
# Matriz retangular alta (6 linhas e 3 colunas)
matriz_alta = np.random.randint(1, 10, size=(6, 3))
print("Matriz Retangular Alta:")
print(matriz_alta)

# Matriz retangular larga (3 linhas e 6 colunas)
matriz_larga = np.random.randint(1, 10, size=(3, 6))
print("\nMatriz Retangular Larga:")
print(matriz_larga)
```

```
Matriz Retangular Alta:
[[2 4 2]
 [9 1 3]
 [2 2 4]
 [9 3 2]
 [1 2 5]
 [7 1 1]]

Matriz Retangular Larga:
[[9 8 7 8 2 1]
 [3 5 4 5 1 3]
 [6 1 5 9 7 7]]
```

Diagonal

A diagonal de uma matriz consiste nos elementos que começam no canto superior esquerdo e descem até o canto inferior direito. Uma matriz diagonal é aquela que possui zeros em todos os elementos fora da sua diagonal principal. Os elementos na diagonal principal podem conter zeros, mas são os únicos que podem ter valores diferentes de zero.

A função `np.diag()` do NumPy possui dois comportamentos distintos, dependendo da entrada fornecida:

1. Se a entrada for uma matriz, a função retornará um vetor contendo os elementos da

diagonal principal.

2. Se a entrada for um vetor, a função retornará uma matriz com os elementos do vetor posicionados na diagonal principal, enquanto os demais elementos serão preenchidos com zeros.

In [21]:
```
#Criar uma matriz diagonal a partir de um vetor
vetor = np.array([1, 2, 3, 4])
matriz_diagonal = np.diag(vetor)
print("Matriz Diagonal criada a partir do vetor:")
print(matriz_diagonal)

#Extrair a diagonal principal de uma matriz
diagonal_extraida = np.diag(matriz_quadrada)
print("\nDiagonal principal extraída da matriz_quadrada:")
print(diagonal_extraida)
```

```
Matriz Diagonal criada a partir do vetor:
[[1 0 0 0]
 [0 2 0 0]
 [0 0 3 0]
 [0 0 0 4]]

Diagonal principal extraída da matriz_quadrada:
[6 6 5]
```

Triângular

Uma matriz triangular possui apenas zeros acima ou abaixo da diagonal principal. A matriz é chamada de triangular superior se os elementos diferentes de zero estão acima da diagonal, ou triangular inferior se estiverem abaixo.

O NumPy tem funções dedicadas para extrair o triângulo superior (`np.triu()`) ou inferior (`np.tril()`) de uma matriz.

In [22]:
```
# Criando uma matriz triangular superior a partir de uma matriz existente
triangular_superior = np.triu(matriz_quadrada)
print("Matriz Triangular Superior:")
print(triangular_superior)

# Criando uma matriz triangular inferior a partir de uma matriz existente
triangular_inferior = np.tril(matriz_quadrada)
print("\nMatriz Triangular Inferior:")
print(triangular_inferior)
```

```
Matriz Triangular Superior:
[[6 6 5]
 [0 6 4]
 [0 0 5]]

Matriz Triangular Inferior:
[[6 0 0]
 [1 6 0]
 [7 4 5]]
```

Identidade

Equivalente ao número 1, no sentido de que qualquer matriz ou vetor multiplicado pela matriz idêntidade resultar na mesma matriz ou vetor. A matriz identidade é uma matriz diagonal quadrada com todos os elementos diagonais tendo o valor de 1. Ela é indicada

pela letra $\bold{I}$ e pode ter um número subscrito informando seu tamanho ($\bold{I}_5$ é uma matriz idêntidade 5×5).

Você pode criar uma matriz identidade no Python usando o `np.eye()`.

In [23]:
```
# Criando uma matriz identidade 4x4
matriz_identidade = np.eye(4)
print("Matriz Identidade:")
print(matriz_identidade)
```

```
Matriz Identidade:
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

In [29]:
```
# Multiplicando a matriz identidade pela matriz_quadrada
matriz_identidade = np.eye(matriz_quadrada.shape[0])
resultado = np.dot(matriz_identidade, matriz_quadrada)

print('Matriz Quadrada:')
print(matriz_quadrada)
print('\nMatriz Identidade:')
print(matriz_identidade)
print('\nResultado da multiplicação da matriz identidade pela matriz_quadrada:')
print(resultado)
```

```
Matriz Quadrada:
[[6 6 5]
 [1 6 4]
 [7 4 5]]

Matriz Identidade:
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]

Resultado da multiplicação da matriz identidade pela matriz_quadrada:
[[6. 6. 5.]
 [1. 6. 4.]
 [7. 4. 5.]]
```

Zeros

A matriz zero é comparavel ao vetor zero: uma matriz com apenas zeros. Ela é indicada pelo zero em negrito: $\bold{0}$.

Pode ser criada pela função `np.zeros()`.

In [30]:
```
# Criando uma matriz zero 3x3
matriz_zero = np.zeros((3, 3))
print("Matriz Zero:")
print(matriz_zero)
```

```
Matriz Zero:
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

In [33]:
```
## Criando imagens das matrizes

# Quadrada
```

```
M1 = np.random.permutation(16).reshape(4,4)

# Triangular superior
M2 = np.triu(np.random.randint(10,20,(3,3)))

# Triangular inferior
M3 = np.tril(np.random.randint(8,16,(3,5)))

# Diagonal
M4 = np.diag( np.random.randint(0,6,size=8) )

# Identidade
M5 = np.eye(4,dtype=int)

# Zeros
M6 = np.zeros((4,5),dtype=int)

matrices  = [ M1,M2,M3,M4,M5,M6 ]
matLabels = [ 'Quadrada','Triângular Superior','Triângular Inferior','Diagonal','Id


_,axs = plt.subplots(2,3,figsize=(12,6))
axs = axs.flatten()

for mi,M in enumerate(matrices):
  axs[mi].imshow(M,cmap='gray',origin='upper',
                 vmin=np.min(M),vmax=np.max(M))
  axs[mi].set(xticks=[],yticks=[])
  axs[mi].set_title(matLabels[mi])

  # Labels de texto
  for (j,i),num in np.ndenumerate(M):
    axs[mi].text(i,j,num,color=[.8,.8,.8],ha='center',va='center',fontweight='bold'

plt.tight_layout()
plt.show()
```