Programação em Python

CEFET-MG

CEFET-MG

- **Biblioteca Numpy**
  - O pacote fundamental para computação científica com Python.
  - Código otimizado (Escrito em C)
    - Funções matemáticas e estatísticas
      - Somatório.
      - Média e desvio padrão.
      - Logaritmos.
    - Manipulação de matrizes.
      - Leitura e escrita em arquivos txt e csv.
      - Multiplicação e divisão.
      - Transposta.

- **Biblioteca Numpy**
  - Matrizes (array)
    - Multi-dimensões
    - Todos os elementos do array devem ser do mesmo tipo de dados.
    - Uma vez criado, o tamanho total do array não pode mudar.
    - O formato deve ser "retangular“.

- **Biblioteca Numpy**
  - Matrizes (array)
    - Exemplo:

```
a = np.array([[1, 2, 3],
              [4, 5, 6]])

b = np.array([2., 2., 3.1, 4.])
```

```
a[1, 1] = ?
a[0, 0] =?
a[0, 2] = ?
b[0]    = ?
a[2, 2] = ?
b[: 3]  = ?
b[:-1]  =?
a[2:]   = ?
```

- **Biblioteca Numpy**
  - Atributos das matrizes:
    - ndim: número de dimensões
    - shape: número de linhas e colunas
    - size: número de elementos
    - dtype: tipo de dado.

- **Biblioteca Numpy**
  - Atributos das matrizes:
    - Exemplo:
      - a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
      - a.ndim == 2
      - a.shape == 3, 4
      - a.size == 12
      - a.dtype == int64

- **Biblioteca Numpy**
  - Criando matrizes:
    - a = np.zeros(2, dtype=np.int64): matriz de zeros, dois elementos inteiros
      - **Dica: tipo booleano np.bool_**
    - a = np.ones(10, np.float64): matriz de 1's, com 10 elementos reais
    - a = np.empty(): matriz vazia
    - a = np.arange(inicio, fim, passo): intervalo
    - a = np.linspace(inicio, fim, num=X): intervalo de X numeros linearmente espassado.

CEFET-MG

- **Biblioteca Numpy**
  - Ordenação de matrizes:
    - np.sort(*array*) : retorna uma cópia ordenada de mesmo *shape.*
    - Ex.:
      - a = np.array([4, 3, 2, 1])
      - b = np.array([9, 8, 7, 6, 5])
      - np.sort(a)
      - np.sort(b)

- **Biblioteca Numpy**
  - Concatenação de matrizes:
    - c = np.concatenate((a, b)): retorna a concatenação de a e b
      - a e b: mesma dimensão, numero diferente de elementos.
    - Ex.:
      - a = np.array([4, 3, 2, 1])
      - b = np.array([9, 8, 7, 6, 5])
      - c = np.concatenate((a, b))

- **Biblioteca Numpy**
  - Alterando o shape da matrizes:
    - c = a.reshape(3, 2) : retorna uma nova matriz (3, 2).
      - O número de elementos deve ser o mesmo.
    - Ex.:
      - a = np.arange(6)
      - b = a.reshape(3, 2)
      - c = a.reshape(3, 3)

- **Biblioteca Numpy**
  - Visualização condicional de elementos da matrizes:
    - Pode-se visualizar elementos que atendam condições especiais.
    - Ex.:
      - a = np.array([[1 , 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
      - print(a[a < 5])
      - five_up = (a >= 5)
      - print(a[five_up])
      - divisible_by_2 = a[a%2==0]
      - print(divisible_by_2)
      - c = a[(a > 2) & (a < 11)]
      - print(c)

CEFET-MG

- **Como criar uma matriz a partir de outra existente**
  - Pegando um trecho específico
  - Ex.:
    - a = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
    - b = a[3:8]
    - c = a[:-1]
    - d = a[1:-1]
    - e = a[-4:-1]
    - f = a[4:-1]

- **Como criar uma matriz a partir de outra existente**
  - Pegando um trecho específico
  - Ex.:
    - **E se alteramos o valor de b[0], o que acontece com a matriz a?**
    - a = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
    - b = a[3:8]
    - b[0] = 11
    - print(a)
    - **Agora teste: b = a.copy()**

- **Como criar uma matriz a partir de outra existente**
  - Pegando um trecho específico
    - Diagonal principal
      - x = np.diag(matriz)
      - Preenchendo a diagonal principal
        - np.fill_diagonal(matriz, 0)
    - Elementos únicos (retorna uma array com elementos)
      - y = np.unique(matriz)

- **Como criar uma matriz a partir de outra existente**
  - Repartir utilizando a coluna
  - np.hsplit(*array*, # de *colunas ou colunas específicas*)
  - Ex.:
    - x = np.arange(1, 25).reshape(2, 12)
    - y = np.hsplit(x, 3) # reparte x em tres matrizes de dimensões iguais
    - z = np.hsplit(x, (3, 4)) # Quebra x em três, até a 3 coluna, até a coluna 4, restante

- **Exercícios**

- **Criação e Ordenação de Matrizes**
  - Crie uma matriz 5x5 de números inteiros aleatórios no intervalo de 0 a 100.
  - Ordene os elementos de cada linha dessa matriz em ordem crescente.
  - Ordene a matriz inteira com base nos valores da primeira coluna (utilize numpy.argsort).

- **Exercícios**

```
import numpy as np

matriz = np.random.randint(0, 101, size=(5, 5))

print(matriz)

matriz_linhas_ordenadas = np.sort(matriz, axis=1)

print(matriz_linhas_ordenadas)

indices_ordenados = np.argsort(matriz[:, 0])  # Índices ordenados pela primeira coluna

matriz_ordenada_por_coluna = matriz[indices_ordenados]

print(matriz_ordenada_por_coluna)
```

- **Exercícios**

- **Cópia e Modificação de Matrizes**
  - Crie uma matriz 4x4 de números reais aleatórios no intervalo [0, 1].
  - Faça uma cópia da matriz.
  - Na matriz copiada, substitua todos os valores maiores que 0,5 pelo valor 1 e os demais por 0 (binarização da matriz).
  - Exiba a matriz original e a matriz modificada.

- **Exercícios**

```
matriz = np.random.rand(4, 4)

print("Matriz original:")

print(matriz)

matriz_copiada = matriz.copy()

matriz_copiada[matriz_copiada > 0.5] = 1

matriz_copiada[matriz_copiada <= 0.5] = 0

print(matriz_copiada)
```

- **Exercícios**
- **Atributos de Matrizes**
  - Crie uma matriz 3x4 contendo números inteiros de 1 a 12 e utilize os atributos do NumPy para:
    - Determinar o formato (shape) da matriz.
    - Obter o número total de elementos.
    - Identificar o tipo de dados da matriz.

CEFET-MG

- **Exercícios**

```
import numpy as np

# Criando a matriz
matriz = np.arange(1, 13).reshape(3, 4)

# Obtendo os atributos
formato = matriz.shape
num_elementos = matriz.size
tipo_dados = matriz.dtype

print("Matriz:\n", matriz)
print("Formato (shape):", formato)
print("Número de elementos:", num_elementos)
print("Tipo de dados:", tipo_dados)
```

- **Exercícios**
  - **Cópias de Matrizes**
    - Crie uma matriz 2×3 contendo números aleatórios entre 0 e 1. Em seguida:
      - Crie uma cópia da matriz.
      - Modifique os valores da cópia multiplicando-os por 10.
      - Verifique se a matriz original foi alterada.

- **Exercícios**

```
import numpy as np

# Criando a matriz original
matriz_original = np.random.random((2, 3))

# Criando uma cópia da matriz
matriz_copia = matriz_original.copy()

# Modificando a cópia
matriz_copia *= 10

print("Matriz Original:\n", matriz_original)
print("Cópia Modificada:\n", matriz_copia)
```

CEFET-MG

- **Exercícios**
- **Ordenação**
  - Crie uma matriz 4×4 com números inteiros aleatórios entre 1 e 50. Em seguida:
    - Ordene os elementos de cada linha.
    - Ordene os elementos de cada coluna.

- **Exercícios**

```
import numpy as np

# Criando a matriz
matriz = np.random.randint(1, 51, size=(4, 4))

# Ordenando por linha
matriz_linhas_ordenadas = np.sort(matriz, axis=1)

# Ordenando por coluna
matriz_colunas_ordenadas = np.sort(matriz, axis=0)

print("Matriz Original:\n", matriz)
print("Matriz com Linhas Ordenadas:\n", matriz_linhas_ordenadas)
print("Matriz com Colunas Ordenadas:\n", matriz_colunas_ordenadas)
```

- **Exercícios**
- **Manipulação de Matrizes**
  - Crie uma matriz 5×5 com números inteiros aleatórios entre 10 e 100. Em seguida, faça o seguinte:
    - Extraia todos os elementos da diagonal principal da matriz.
    - Substitua os elementos da diagonal principal por 0.
    - Crie uma nova matriz contendo apenas os elementos menores do que 50 da matriz original (sem repetições) e os ordene em ordem crescente.

CEFET-MG

- **Exercícios**

```
import numpy as np

matriz = np.random.randint(10, 101, size=(5, 5))

# Extraindo os elementos da diagonal principal
diagonal_principal = np.diag(matriz)

# Substituindo os elementos da diagonal principal por 0
matriz_sem_diagonal = matriz.copy()
np.fill_diagonal(matriz_sem_diagonal, 0)

elementos_menores_50 = np.unique(matriz[matriz < 50])
elementos_menores_50_ordenados =
np.sort(elementos_menores_50)
```

CEFET-MG

- **Operações com matrizes**
  - Matriz com mesmo shape
  - Elemento a elemento.
  - data = np.array([1, 2])
  - ones = np.ones(2, dtype=int)

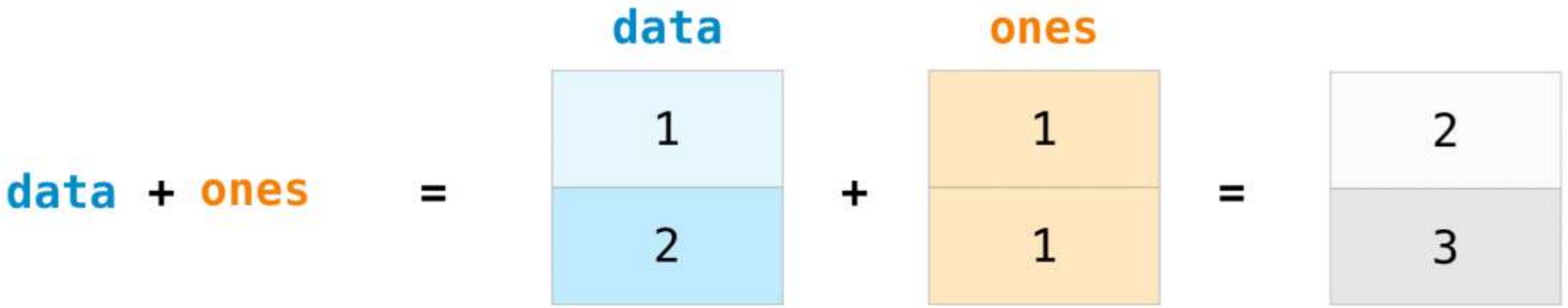
```
data = np.array([1,2])
```

data
1
2

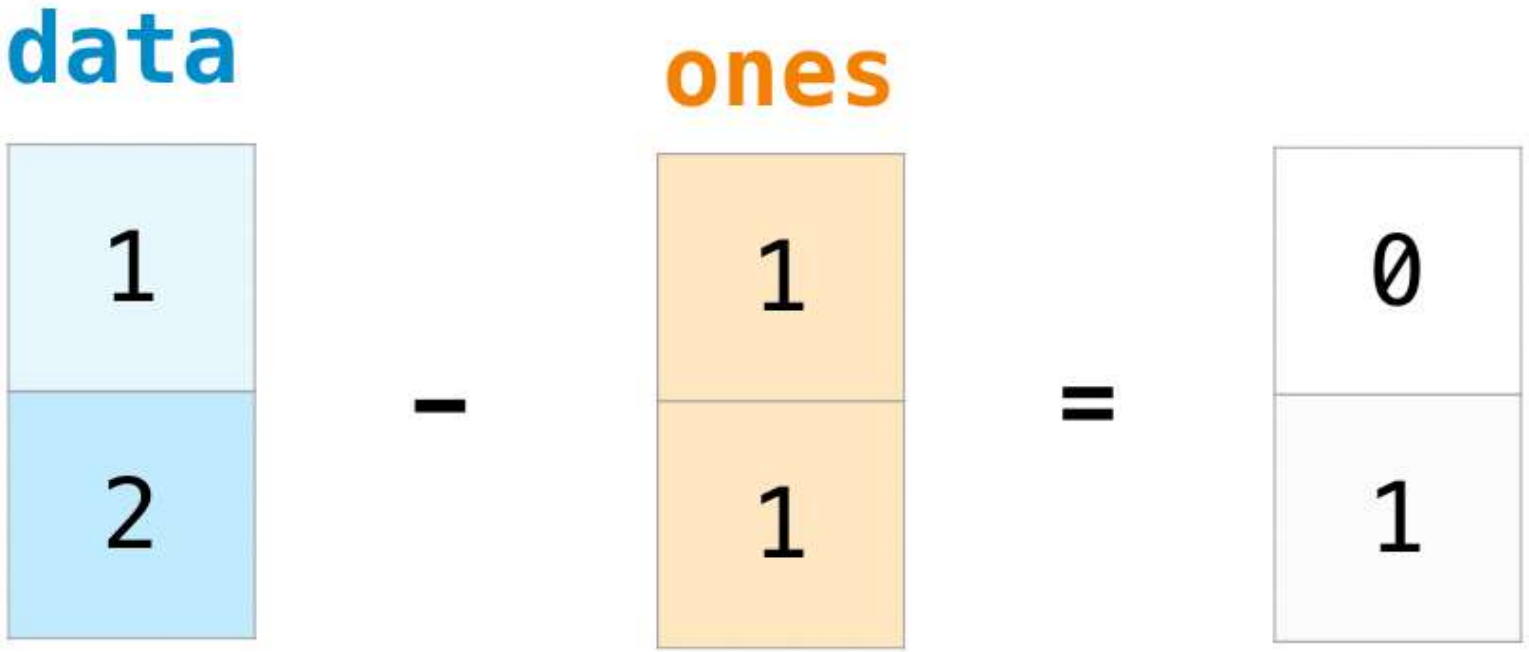
```
ones = np.ones(2)
```

ones
1
1

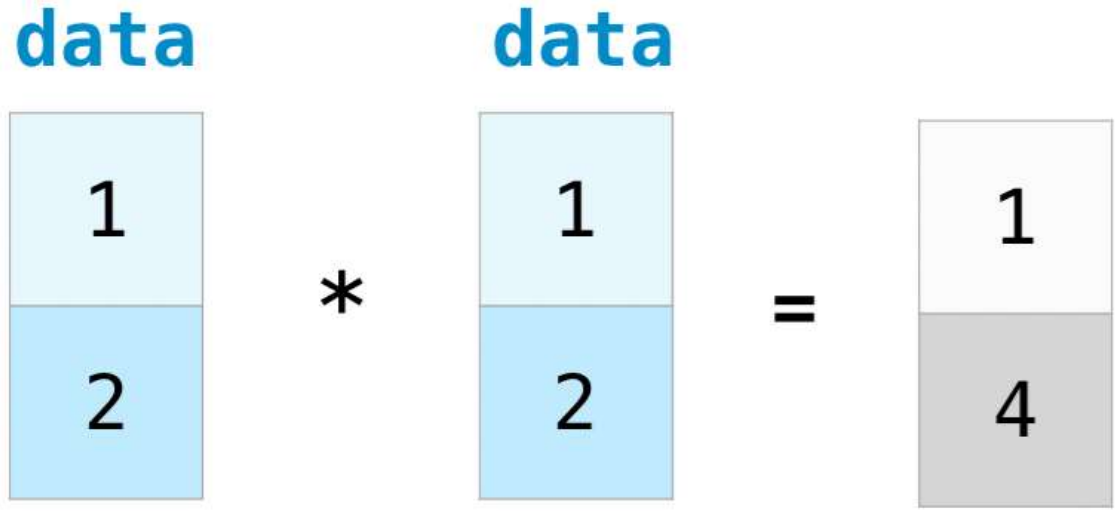
- **Operações com matrizes**
  - **Soma:**
    - data + ones



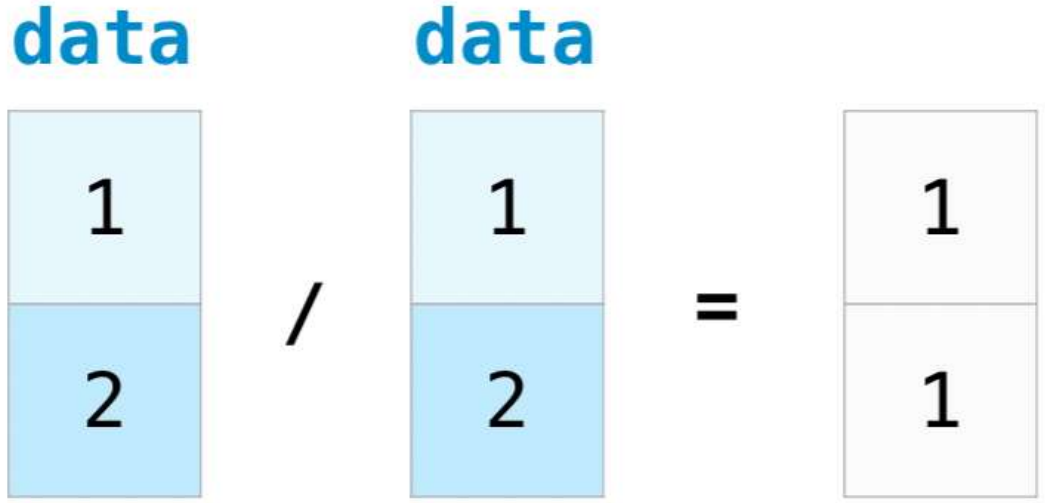
- **Operações com matrizes**
  - **Subtração:**
    - data - ones



- **Operações com matrizes**
  - **Multiplicação:**
    - data * data



- **Operações com matrizes**
  - **Divisão:**
    - data / data



- **Operações com matrizes**
  - Multiplicação entre matrizes (linhas = colunas)
    - A @ B
    - A.dot(B)

A

1	1
0	1

B

2	0
3	4

1x2 + 1x3	1x0 + 1x4	=	5	4
0x2 + 1x3	0x0 + 1x4		3	4

- **Operações com matrizes**
  - **Somatório dos elementos**
    - a = np.array([1, 2, 3, 4])
    - x = a.sum()
    - Somatório de um eixo específico:
      - b = np.array([[1, 1], [2, 2]])
      - x = b.sum(axis=0)

- **Operações com matrizes**
  - **Outras funções:**
    - a = np.array([1, 2, 3, 4])
    - x = a.max(): máximo
    - x = a.mim(): mínimo
    - x = a.mean(): média
    - x = a.std(): desvio padrão

- **Exercícios**
  - Cálculo da Média e Desvio Padrão (Estatística de Dados): No monitoramento de sensores industriais, é comum calcular estatísticas para identificar anomalias em medições. Um sensor registra as temperaturas de uma máquina ao longo do dia. As medições estão armazenadas em um array NumPy:
    - temperaturas = np.array([22.4, 23.1, 22.8, 23.0, 22.7, 22.9, 23.2, 22.7, 23.1, 23.6])
    - Calcule a média das temperaturas.
    - Calcule o desvio padrão das temperaturas.
    - Plot um gráfico boxplot e verifique se existe algum outlier.

Programação em Python

CEFET-MG

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

# Dados de temperatura
temperaturas = np.array([22.4, 23.1, 22.8, 23.0, 22.7, 22.9, 23.2, 22.7, 23.1, 23.6])

# Cálculo da média
media = np.mean(temperaturas)
print(f"Média das temperaturas: {media:.2f} °C")

# Cálculo do desvio padrão
desvio_padrao = np.std(temperaturas)
print(f"Desvio padrão das temperaturas: {desvio_padrao:.2f} °C")

# Plot do boxplot
plt.figure(figsize=(6,4))
plt.boxplot(temperaturas)
plt.title("Boxplot das Temperaturas do Sensor")
plt.ylabel("Temperatura (°C)")
plt.grid(True)
plt.show()
```

- **Exercícios**
  - A soma de matrizes é usada em processamento de imagens, por exemplo, ao combinar filtros aplicados a uma imagem. Considere duas matrizes que representam partes processadas de uma imagem:
    - A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
    - B = np.array([[9, 8, 7], [6, 5, 4], [3, 2, 1]])
  - Escreva um programa que utilize NumPy para somar as matrizes somente

CEFET-MG

```
import numpy as np

# Matrizes da imagem
A = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])

B = np.array([[9, 8, 7],
              [6, 5, 4],
              [3, 2, 1]])

# Soma das matrizes
C = A + B

print("\nSoma das Matrizes (A + B):")
print(C)
```

CEFET-MG

- **Matriz transposta**
  - a = np.array([1, 2, 3, 4],

    [4, 5, 6, 7])
  - x = a.transpose()
  - x = a.T
  - x = a.reshape(4, 2) **# a: 2 x 4**

CEFET-MG

- **Matriz Inversa**

np.linalg.inv(matriz)

Ex.:

```
a = np.array([[1, 2, 3, 4], [4, 5, 6, 7]])
try:
    a_inversa = np.linalg.inv(a)
except np.linalg.LinAlgError: print("A matriz não é inverssível (determinante é zero).")
```

CEFET-MG

- **Determinante da Matriz**

np.linalg.det(matriz)

Ex.:

```
a = np.array([[1, 2, 3, 4], [4, 5, 6, 7]])

x = np.linalg.det(a)
```

CEFET-MG

- **Solução de sistemas lineares**
  - **Seja o sistema Ax = B**
    - A matriz quadrada.
    - B é o vetor (ou matriz) de termos constantes
    - x é o vetor (ou matriz) de incógnitas que será calculado.
    - x = np.linalg.solve(A, B)

- **Interpolação**
  - Dado um conjunto de dados.
  - Ajuste um polinômio de grau n: $P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x^1 + a_0$.
    - np.polyfit(x, y, n)
    - x: Array dos valores das variáveis independentes.
    - y: Array dos valores das variáveis dependentes.
    - deg: Grau do polinômio que você deseja ajustar.
    - w (opcional): Pesos a serem aplicados aos dados, para dar mais relevância a certos pontos.

- **Interpolação**
  - Dado um conjunto de dados.
  - Ajuste um polinômio de grau n: $P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x^1 + a_0$.
  - Previsões futuras
    - coef = np.polyfit(x, y, n)
    - previsão = np.polyval (coef, prev)
      - prev: valor para o qual deve-se fazer a previsão.

- **Lendo e escrevendo em arquivos**
  - Arquivos Numpy (Mais eficiente)
    - Arquivo npy: Armazena um único array em um arquivo binário.
    - Arquivo npz: múltiplos arrays em um único arquivo compactado.
      - np.save('filename', matriz) **# Salva em um arquivo npy**
      - np.savez('filename', matriz1, matriz2) **# Salva em um arquivo npz**
      - np.load('filename') **# Lê matriz do arquivo npz ou npy**

CEFET-MG

- **Lendo e escrevendo em arquivos**
  - Ex.:

```
a = np.array([[1, 1, 1], [2, 2, 2], [3, 3, 3], [4, 4, 4]])

np.save("matriz.npy",a)

b = np.load("matriz.npy")

print(b)
```

CEFET-MG

- **Lendo e escrevendo em arquivos**
  - Ex.:

```
a = np.array([[1, 1, 1], [2, 2, 2], [3, 3, 3], [4, 4, 4]])
b = np.array([[0, 0, 0], [0, 0, 0], [5, 5, 5], [6, 6, 6]])
np.savez("matriz.npz",a, b)
dados = np.load("matriz.npz")
c = dados['arr_0'] #'arr_0‘: nome padrão (quando não explicitado)
d = dados['arr_1'] #'arr_1‘: nome padrão (quando não explicitado)
print(c)
print(d)
```

- **Lendo e escrevendo em arquivos**
  - Ex.:

```
a = np.array([[1, 1, 1], [2, 2, 2], [3, 3, 3], [4, 4, 4]])
b = np.array([[0, 0, 0], [0, 0, 0], [5, 5, 5], [6, 6, 6]])
np.savez("matriz.npz", m1=a, m2=b) #m1 e m2: nomes explícitos
dados = np.load("matriz.npz")
c = dados['m1']
d = dados['m2']
print(c)
print(d)
```

- **Lendo e escrevendo em arquivos**
  - Arquivos texto (.txt ou .csv)
    - np.savetxt('filename', matriz) **#Uma matriz por arquivo**
    - np.loadtxt('filename') **#Todo o conteúdo: matriz única**

Para .csv: melhor usar o Pandas

CEFET-MG

- **Lendo e escrevendo em arquivos**
  - Ex.:

```
a = np.array([[1, 1, 1], [2, 2, 2], [3, 3, 3], [4, 4, 4]])
b = np.array([[0, 0, 0], [0, 0, 0], [5, 5, 5], [6, 6, 6]])
np.savetxt("matriza.txt",a)
np.savetxt("matrizb.txt",b)
c = np.loadtxt("matriza.txt")
d = np.loadtxt("matrizb.txt")
print(c)
print(d)
```

CEFET-MG

- **Lendo e escrevendo em arquivos**
  - Ex.:

```
a = np.array([[1, 1, 1], [2, 2, 2], [3, 3, 3], [4, 4, 4]])
b = np.array([[0, 0, 0], [0, 0, 0], [5, 5, 5], [6, 6, 6]])
np.savetxt("matriza.csv",a)
np.savetxt("matrizb.csv",b)
c = np.loadtxt("matriza.csv")
d = np.loadtxt("matrizb.csv")
print(c)
print(d)
```

- **Outras funções do Numpy**
  - **Funções Trigonométricas**
    - np.sin(x) – Calcula o seno.
    - np.cos(x) – Calcula o cosseno.
    - np.tan(x) – Calcula a tangente.
    - np.arcsin(x) – Calcula o arco-seno (seno inverso).
    - np.arccos(x) – Calcula o arco-cosseno (cosseno inverso).
    - np.arctan(x) – Calcula o arco-tangente (tangente inversa).
    - np.hypot(x, y) – Calcula a hipotenusa de x e y.
    - np.deg2rad(x) – Converte graus para radianos.
    - np.rad2deg(x) – Converte radianos para graus.

- **Outras funções do Numpy**
  - **Funções Exponenciais e Logarítmicas**
    - np.exp(x) – Calcula e^x para cada elemento.
    - np.log(x) – Calcula o logaritmo natural de x.
    - np.log10(x) – Calcula o logaritmo base 10.
    - np.log2(x) – Calcula o logaritmo base 2.
    - np.power(base, expoente) – Calcula base^expoente.

- **Outras funções do Numpy**
  - **Funções de Arredondamento**
    - np.round(x, decimals=n) – Arredonda para n casas decimais.
    - np.floor(x) – Arredonda para baixo.
    - np.ceil(x) – Arredonda para cima.
    - np.trunc(x) – Trunca os valores (remove a parte decimal).
    - np.rint(x) – Retorna o inteiro mais próximo.

- **Outras funções do Numpy**
  - **Funções de Distribuições Aleatórias (usando numpy.random)**
    - np.random.uniform(low, high, size) – Gera valores aleatórios uniformemente em um intervalo.
    - np.random.normal(mean, std, size) – Gera valores de uma distribuição normal com média e desvio padrão especificados.

- **Exercícios**

- Movimentos aleatórios, como o passeio aleatório, são usados para modelar partículas em física ou movimentos financeiros.
- Simule o movimento aleatório de uma partícula em uma linha reta por 100 passos, onde cada passo é +1 ou −1, com igual probabilidade. Plote o deslocamento cumulativo.
- **Dica use: np.random.choice()** para descrever o caminho.

  np.cumsum() para o deslocamento cumulativo

CEFET-MG

- **Exercícios**

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

np.random.seed(42)  # Para resultados reproduzíveis

passos = np.random.choice([-1, 1], size=100)
deslocamento = np.cumsum(passos)

plt.plot(deslocamento)
plt.title("Movimento Aleatório")
plt.xlabel("Passos")
plt.ylabel("Deslocamento")
plt.grid()
plt.show()
```

- **Exercícios**
- Converter imagens para tons de cinza é um passo comum em análise de imagens. Dada uma matriz representando uma imagem colorida em formato 3 x 3, onde cada célula contém valores RGB:

```
imagem = np.array([
    [[100, 150, 200], [50, 100, 150], [200, 250, 100]],
    [[150, 200, 50], [100, 50, 200], [250, 150, 50]],
    [[200, 100, 150], [50, 200, 250], [150, 50, 100]]
])
```

- Converta a imagem para tons de cinza usando a fórmula:
  - Cinza= 0.2989R + 0.5870G + 0.1140B

Programação em Python

CEFET-MG

- **Exercícios**

```
import numpy as np

imagem = np.array([
    [[100, 150, 200], [50, 100, 150], [200, 250, 100]],
    [[150, 200, 50], [100, 50, 200], [250, 150, 50]],
    [[200, 100, 150], [50, 200, 250], [150, 50, 100]]
])

cinza = 0.2989 * imagem[:, :, 0] + 0.5870 * imagem[:, :, 1] + 0.1140 *
imagem[:, :, 2]

print("Imagem em tons de cinza:")
print(cinza)
```

- **Exercícios**

- Um engenheiro da computação está analisando um circuito elétrico que contém três malhas interligadas. Cada malha tem resistores e uma fonte de tensão, e as correntes I_1, I_2, e I_3 precisam ser determinadas usando as Leis de Kirchhoff.
- Esquema do Circuito:
  - Malha 1: $5I_1 + 2I_2 = 10$ (tensão da fonte é 10V).
  - Malha 2: $2I_1 + 4I_2 - I_3 = 12$ (tensão da fonte é 12V).
  - Malha 3: $-I_2 + 3I_3 = 5$ (tensão da fonte é 5V).
- Determine o valor das correntes I_1, I_2, e I_3.

Programação em Python

CEFET-MG

```
import numpy as np

A = np.array([[5, 2, 0],
              [2, 4, -1],
              [0, -1, 3]], dtype=float)

b = np.array([10, 12, 5], dtype=float)

I = np.linalg.solve(A, b)

print("I1 = {:.6f} A".format(I[0]))
print("I2 = {:.6f} A".format(I[1]))
print("I3 = {:.6f} A".format(I[2]))

# Verificação (resíduo)
residuo = A.dot(I) - b
print("\nResíduo (deve ser ~0):", residuo)
```

- **Exercícios**
  - A geração de sinais aleatórios é usada em simulações de ruídos em comunicações ou modelos financeiros.
  - Gere um vetor de 100 valores aleatórios seguindo uma distribuição normal com média 0 e desvio padrão 1.
  - Calcule a média e o desvio padrão dos valores gerados.

CEFET-MG

```
import numpy as np

# Geração do vetor com 100 valores de distribuição normal (média=0, desvio=1)
valores = np.random.normal(loc=0, scale=1, size=100)

# Cálculo da média e do desvio padrão
media = np.mean(valores)
desvio = np.std(valores)

print("Média dos valores gerados:", media)
print("Desvio padrão dos valores:", desvio)
```

- **Exercícios**
  - Antes de alimentar um modelo de aprendizado de máquina, é comum normalizar os dados para melhorar a performance.
  - Dado o vetor de dados:
    - dados = np.array([50, 60, 70, 80, 90])
  - Normalize os dados para que fiquem no intervalo [0, 1]

$$x_{norma} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

CEFET-MG

```
import numpy as np

dados = np.array([50, 60, 70, 80, 90])

# Normalização Min-Max
min_val = np.min(dados)
max_val = np.max(dados)

dados_normalizados = (dados - min_val) / (max_val - min_val)

print("Dados normalizados:", dados_normalizados)
```

- **Exercícios**

- Um aluno está trabalhando na análise estrutural de um sistema de treliças simples. O sistema é composto por três barras interconectadas, formando um triângulo, onde forças atuam nos nós. O objetivo é determinar as forças internas em cada barra usando métodos de álgebra linear.

- Um nó de treliça está sujeito a:
  - Uma força horizontal de 100 N à direita.
  - Uma força vertical de 200 N para baixo.

- **Exercícios**

- As forças nas barras F_1, F_2, e F_3 precisam ser determinadas para que o sistema esteja em equilíbrio (somatório das forças é zero).
- As equações de equilíbrio resultantes são:
  - $F_1\cos(30) + F_2 - 100 = 0$ (equilíbrio na direção x).
  - $F_1\sin(30) + F_3 - 200 = 0$ (equilíbrio na direção yy).
  - $F_3 - F_2 = 0$ (equilíbrio no nó).
- Resolva o sistema de equações

Programação em Python

CEFET-MG

```
import numpy as np

# Ângulos em radianos
cos30 = np.cos(np.deg2rad(30))
sin30 = np.sin(np.deg2rad(30))

# Matriz do sistema
A = np.array([
    [cos30, 1,     0],
    [sin30, 0,     1],
    [0,    -1,     1]
], dtype=float)

# Vetor de termos independentes
b = np.array([100, 200, 0], dtype=float)

# Solução do sistema
F = np.linalg.solve(A, b)

F1, F2, F3 = F

print("F1 =", F1)
print("F2 =", F2)
print("F3 =", F3)
```

- **Exercícios**
  - Um engenheiro da computação está desenvolvendo um sistema para analisar dados coletados por um sensor de temperatura que registra medições a cada segundo por 1 minuto. O objetivo é realizar operações estatísticas para identificar padrões e possíveis anomalias.
  - Gere 60 valores aleatórios entre 20°C e 30°C para simular as leituras do sensor.
  - Calcule a média, mediana e desvio padrão das temperaturas.
  - Calcule a temperatura máxima e mínima registradas.
  - Identifique quantas leituras estão fora de 2σ da média (dois desvios-padrão).
  - Plote um gráfico simples (usando matplotlib) para visualizar as leituras

Programação em Python

CEFET-MG

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

# Gerando dados simulados de temperatura
temperaturas = np.random.uniform(20, 30, 60)

# Estatísticas básicas
media = np.mean(temperaturas)
mediana = np.median(temperaturas)
desvio_padrao = np.std(temperaturas)
max_temp = np.max(temperaturas)
min_temp = np.min(temperaturas)

# Identificando leituras fora de 2σ
fora_2sigma = np.sum((temperaturas < media - 2 *
              desvio_padrao) | (temperaturas > media
              + 2 * desvio_padrao))

# Exibindo os resultados
print(f"Média: {media:.2f}°C")
print(f"Mediana: {mediana:.2f}°C")
print(f"Desvio padrão: {desvio_padrao:.2f}°C")
print(f"Temperatura máxima: {max_temp:.2f}°C")
print(f"Temperatura mínima: {min_temp:.2f}°C")
print(f"Leituras fora de 2σ: {fora_2sigma}")

# Gráfico opcional
plt.plot(temperaturas, marker='o', label=
              "Temperaturas")
plt.axhline(media, color='red', linestyle='--',
              label="Média")
plt.xlabel("Tempo (segundos)")
plt.ylabel("Temperatura (°C)")
plt.legend()
plt.show().
```

- **Exercícios**
  - A população de uma cidade cresce ao longo dos anos de acordo com os seguintes dados:
    - Ano=[2000, 2005, 2010, 2015, 2020, 2025]
    - População=[50000, 60000, 72000, 85000, 100000, 120000]
  - Ajuste um polinômio de grau 3 aos dados fornecidos e Use o modelo para prever a população em 2030.
  - Plote os dados originais como pontos.
  - Plote a curva ajustada no intervalo de 2000 a 2030.
  - Indique no gráfico a previsão para o ano de 2030.
  - Discuta se o modelo ajustado parece adequado para prever valores futuros.

Programação em Python

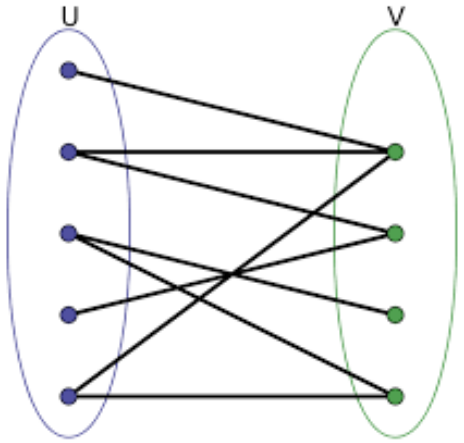
CEFET-MG

```
anos = np.array([2000, 2005, 2010, 2015, 2020, 2025])
populacao = np.array([50000, 60000, 72000, 85000,
100000, 120000])
# Ajuste de polinômio (grau 3)
coef = np.polyfit(anos, populacao, 3)
# Extensão até 2030
anos_ext = np.arange(2000, 2031)
pop_pred = np.polyval(coef, anos_ext)
# Previsão para 2030
pop_2030 = np.polyval(coef, 2030)
print(f"População estimada para 2030: {pop_2030:.2f}")
```

```
plt.scatter(anos, populacao, color='blue',
label='Dados Reais')
plt.plot(anos_ext, pop_pred, color='red',
label='Modelo (Grau 3)')
plt.axvline(2030, color='green', linestyle='--',
label='Ano 2030')
plt.scatter(2030, pop_2030, color='purple',
label=f'Previsão: {pop_2030:.0f}')
plt.title("Modelagem de Crescimento
Populacional")
plt.xlabel("Ano")
plt.ylabel("População")
plt.legend()
plt.grid(True)
plt.show()
```

- **Exercícios**

Grafo bipartido, é um grafo em que os vértices podem ser divididos em dois conjuntos disjuntos, de forma que cada aresta conecte um vértice de um conjunto a um vértice do outro, como na imagem.



Verifique se o grafo representado pela matriz de adjacência abaixo é bipartido:

- grafo = np.array([[0, 1, 0, 1], [1, 0, 1, 0], [0, 1, 0, 1], [1, 0, 1, 0]])

Programação em Python

CEFET-MG

```
import numpy as np

grafo = np.array([
    [0, 1, 0, 1],
    [1, 0, 1, 0],
    [0, 1, 0, 1],
    [1, 0, 1, 0]
], dtype=int)

def eh_bipartido(adj):
    n = adj.shape[0]
    color = [-1] * n   # -1 = não colorido

    for start in range(n):
        if color[start] != -1:
            continue

        # Usando lista como fila
        fila = [start]
        color[start] = 0

        while fila:
            u = fila.pop(0)  # remove o primeiro elemento

            for v in range(n):
                if adj[u, v] == 0:
                    continue

                if color[v] == -1:
                    color[v] = 1 - color[u]
                    fila.append(v)
                elif color[v] == color[u]:
                    return False, None

    part0 = [i for i in range(n) if color[i] == 0]
    part1 = [i for i in range(n) if color[i] == 1]
    return True, (part0, part1)

bip, partes = eh_bipartido(grafo)
print("É bipartido?", bip)
if bip:
    print("Partições:", partes)
```

Programação em Python

CEFET-MG

- Na construção de uma rede neural, uma função de ativação comum é a **sigmoid**. Um engenheiro precisa calcular a saída da sigmoid para diferentes valores de entrada (pesos e bias).
  - Crie um array com 50 valores igualmente espaçados entre -10 e 10.
  - Implemente a função sigmoid: $sigmoid = 1/(1 + e^{-x})$
  - Calcule o valor da sigmoid para cada entrada.
  - Determine os valores de entrada para os quais a saída está acima de 0.9.
  - Plote o gráfico da função sigmoid.

Programação em Python

CEFET-MG

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

# 1. Criar array com 50 valores de -10 a 10
x = np.linspace(-10, 10, 50)

# 2. Função sigmoid
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# 3. Calcular valores da sigmoid
y = sigmoid(x)

# 4. Determinar valores de entrada cuja saída > 0.9
indices_acima_09 = np.where(y > 0.9)[0]
valores_acima_09 = x[indices_acima_09]

print("Valores de entrada para os quais a sigmoid > 0.9:")
print(valores_acima_09)

# 5. Plotar gráfico da sigmoid
plt.plot(x, y)
plt.title("Função Sigmoid")
plt.xlabel("x")
plt.ylabel("sigmoid(x)")
plt.grid(True)
plt.show()
```

- **DESAFIO: enviar para SIGAA**
  - O algoritmo de Dijkstra é usado para encontrar o caminho mais curto em grafos ponderados, como em mapas de rotas. Dado o grafo representado pela matriz de adjacência ponderada:

    grafo = np.array([[0, 10, 0, 30, 100], [10, 0, 50, 0, 0], [0, 50, 0, 20, 10], [30, 0, 20, 0, 60], [100, 0, 10, 60, 0]])

  - Implemente o algoritmo de Dijkstra para encontrar o menor caminho do nó 0 para todos os outros nós.

CEFET-MG

```
import numpy as np


# Grafo dado
grafo = np.array([
    [0, 10, 0, 30, 100],
    [10, 0, 50, 0, 0],
    [0, 50, 0, 20, 10],
    [30, 0, 20, 0, 60],
    [100, 0, 10, 60, 0]
])
```

Programação em Python

CEFET-MG

```
def dijkstra(grafo, origem):
    n = grafo.shape[0]
    visitado = np.zeros(n, dtype=bool)

    # Inicializa distâncias com infinito
    distancia = np.full(n, np.inf)
    distancia[origem] = 0

    for _ in range(n):
        # Seleciona o nó não visitado com menor distância
        u = np.argmin(np.where(visitado, np.inf, distancia))
        visitado[u] = True

        # Atualiza as distâncias dos vizinhos
        for v in range(n):
            peso = grafo[u, v]
            if peso > 0 and not visitado[v]:
                nova_distancia = distancia[u] + peso
                if nova_distancia < distancia[v]:
                    distancia[v] = nova_distancia

    return distancia
```

CEFET-MG

```
# Executando Dijkstra a partir do nó 0

dist = dijkstra(grafo, 0)

print("Menores distâncias a partir do nó 0:")

print(dist)
```