

Departamento de Engenharia de Sistemas e Computação

Matrizes

Tabelas de dados experimentais e pequenas simulações

João Araujo

Algoritmos Computacionais I

Aula 19

Roteiro

- Dados em tabela: linhas e colunas (Cap. 5.10)
- Listas de listas (5.11)
- Percorrendo e construindo matrizes
- O erro clássico: `[[0]*3]*4`
- Aplicações: tabelas experimentais e uma pequena simulação

Matrizes e tabelas de dados

Dados experimentais costumam vir em **tabelas**:

Ensaio	V (V)	I (A)	T (°C)
1	5,0	0,25	21,3
2	10,0	0,52	22,1
3	15,0	0,76	23,4
4	20,0	1,01	25,0

- Cada **linha** é um ensaio
- Cada **coluna** é uma grandeza
- Em matemática: matriz 4×3
- Em Python: uma **lista de listas**

Lista de listas

```
ensaios = [
    [5.0, 0.25, 21.3],
    [10.0, 0.52, 22.1],
    [15.0, 0.76, 23.4],
    [20.0, 1.01, 25.0],
]
print(ensaios[1])          # [10.0, 0.52, 22.1]   (uma linha)
print(ensaios[1][2])       # 22.1   (linha 1, coluna 2)
print(len(ensaios))        # 4 linhas
print(len(ensaios[0]))     # 3 colunas
```

- Dois índices: `matriz[linha][coluna]`
- Cada linha é uma lista independente (e poderia ter tamanho diferente!)

Percorrendo uma matriz

```
for i in range(len(ensaios)):
    for j in range(len(ensaios[i])):
        print(ensaios[i][j], end='\t')
    print()
```

Ou diretamente:

```
for linha in ensaios:
    for valor in linha:
        print(valor, end='\t')
    print()

# resistência (V/I) de cada ensaio
resistencias = [linha[0] / linha[1] for linha in ensaios]
# média de cada coluna
medias = [sum(l[j] for l in ensaios) / len(ensaios) for j in range(3)]
```

Construindo uma matriz

```python
# matriz 3 x 4 de zeros, usando laços aninhados
matriz = []
for i in range(3):
    linha = []
    for j in range(4):
        linha.append(0)
    matriz.append(linha)

# o mesmo com compreensão aninhada
matriz = [[0 for j in range(4)] for i in range(3)]
print(matriz)     # [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

- Exercício 5.27: criar uma matriz bidimensional usando dois comandos `for`
- Exercício 5.28: matriz tridimensional com compreensão (lista de listas de listas)

O erro clássico

```
matriz = [[0] * 3] * 4
print(matriz)          # [[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
matriz[1][0] = 1
print(matriz)
```

```
[[1, 0, 0], [1, 0, 0], [1, 0, 0], [1, 0, 0]]
```

O que aconteceu?

`[x] * 4` repete a *mesma referência* quatro vezes: as 4 linhas são o **mesmo objeto** (aliasing da Aula 18). Use a compreensão: `[[0]*3 for i in range(4)]`.

Fonte: `code/Cap5/prog.5.b_testematriz.py`

Operações com matrizes

```python
def soma(a, b):
    """Soma duas matrizes de mesmas dimensões."""
    return [[a[i][j] + b[i][j] for j in range(len(a[0]))]
            for i in range(len(a))]

def transposta(a):
    return [[a[i][j] for i in range(len(a))]
            for j in range(len(a[0]))]

A = [[1, 2, 3], [4, 5, 6]]
print(transposta(A))    # [[1, 4], [2, 5], [3, 6]]
```

Para álgebra linear séria (produto de matrizes, inversa, autovalores), usaremos **NumPy** na próxima aula.

Pequena simulação: um tabuleiro

- Matrizes representam **estados**: tabuleiros, mapas, grades de células
- Exemplo: o "jogo da velha" como matriz 3×3
- Valores: `' '`, `'X'`, `'O'`

Funções do programa

```
novo_tabuleiro()
imprime(tab)
joga(tab, i, j, sim)
venceu(tab, sim)
```

Jogo da velha: estrutura

```
def novo_tabuleiro():
    return [[' '] * 3 for _ in range(3)]

def imprime(tab):
    for linha in tab:
        print(' | '.join(linha))
        print('-' * 9)

def venceu(tab, s):
    linhas = any(all(c == s for c in l) for l in tab)
    colunas = any(all(tab[i][j] == s for i in range(3)) for j in range(3))
    diag = all(tab[i][i] == s for i in range(3)) or \
           all(tab[i][2 - i] == s for i in range(3))
    return linhas or colunas or diag
```

Cada função faz uma coisa; o jogo completo é um desafio para o laboratório.

Simulação: espalhando calor em uma barra

```
# 1 linha por instante de tempo, 1 coluna por posição
n, passos = 10, 5
T = [[0.0] * n for _ in range(passos)]
T[0][0] = 100.0                          # extremidade quente
for t in range(1, passos):
    for x in range(n):
        esq = T[t-1][x-1] if x > 0 else T[t-1][x]
        dir_ = T[t-1][x+1] if x < n-1 else T[t-1][x]
        T[t][x] = (esq + T[t-1][x] + dir_) / 3
for linha in T:
    print([round(v, 1) for v in linha])
```

Cada instante é calculado a partir do anterior: a matriz guarda o *histórico* da simulação. (Esse tipo de problema é muito mais simples em NumPy.)

Atividade

1. Armazene a tabela de ensaios (tensão, corrente, temperatura) em uma matriz. Calcule a resistência de cada ensaio e a resistência média.
2. Escreva `produto(a, b)`: produto matricial de duas matrizes quadradas $n \times n$.
3. Escreva uma função que verifica se uma matriz quadrada é **simétrica**.
4. Complete o jogo da velha (entrada das jogadas e detecção de empate).

Desafio opcional (livro, Cap. 5): o **Jogo da Vida** de Conway é uma simulação em matriz; veja `code/Cap5/prog.5.c_lifegame.py`.

Para praticar: exercícios do livro

Site do livro – Capítulo 5

- Listas de listas: **5.27** (matriz com `for`), **5.28** (matriz tridimensional)
- Revise também **5.5** e **5.9** (listas e fatias)

Enunciados e soluções comentadas em
`https://www.araujo.eng.uerj.br/livro/exercicios/cap5/`

Exercícios adicionais (sem solução): `https://www.araujo.eng.uerj.br/livro/extras/`

Resumo da aula

- Matriz = lista de listas; acesso `m[i][j]`
- Use compreensão aninhada para criar matrizes: `[[0]*c for _ in range(l)]`
- Nunca use `[[0]*c]*l`: as linhas ficam compartilhadas
- Próxima aula: dicionários e tuplas