Universidade do Minho
Licenciatura em Física

Tarefa 6

Professores:	Cristiana Neto, Pedro Oliveira, Vitor Alves
Disciplina:	Linguagens para Computação Numérica
Tema:	Matrizes e Numpy
Data:	Março de 2022

1 Introdução

1.1 Matrizes

Matrizes são estruturas bidimensionais (tabelas) com m linhas e n colunas muito importantes na matemática, utilizadas por exemplo para a resolução de sistemas de equações.

Em Python, uma matriz pode ser representada como uma lista de listas, onde um elemento da lista contém uma linha da matriz, que por sua vez corresponde a uma lista com os elementos da coluna da matriz.

Por exemplo, a matriz $\begin{bmatrix} 2 & 3 & 4 \\ 0 & 7 & 5 \end{bmatrix}$ é representada por: `matriz = [[2,3,4] , [0, 7,5]]`. Neste caso:

- matriz[0][0] = 2 matriz[1][0] = 0
- matriz[0][1] = 3 matriz[1][1] = 7
- matriz[0][2] = 4 matriz[1][2] = 5

As matrizes podem ser criadas e manipuladas usando ciclos `for`. Por exemplo, o programa seguinte cria uma matriz `m x n` preenchida com zeros.

```
m = int(input('Digite a dimensão m (linhas) da matriz: '))
n = int(input('Digite a dimensão n (colunas) da matriz: '))

matriz = []
for i in range(m):
  linha = []
  for j in range(n):
    linha.append(0)
  matriz.append(linha)

print(matriz)
```

```
Digite a dimensão m (linhas) da matriz: 4
Digite a dimensão n (colunas) da matriz: 2
[[0, 0], [0, 0], [0, 0], [0, 0]]
```

1.2 Numpy

Tal como já vimos em aulas anteriores, o Python inclui um conjunto de bibliotecas que podem ser importados. Um deles é o NumPy, que facilita operações com matrizes e torna o seu processamento mais eficaz. Resumidamente, o NumPy é uma biblioteca a que permitem trabalhar com computação numérica. O seu principal objeto é o vetor n-dimensional, ou `array`.

Um vetor n-dimensional também é conhecido pelo nome tensor. A principal característica do `array` é que ele deve ser homogénio, ou seja, diferentemente do objeto lista, todos os seus elementos devem ser do mesmo tipo. Podemos criar um `array` através de uma lista:

```
import numpy as np
v = np.array([1,2,3,4])
print(v)
```

```
[1 2 3 4]
```

Os arrays possuem um atributo chamado `shape`. Esse atributo indica a forma do array, por exemplo:

```
print(v.shape)
```

```
(4,)
```

Neste caso, o array `v` possui 1 dimensão (ou eixo) com 4 elementos. Um array unidimensional corresponde a um vetor. Podemos também criar um array bidimensional (uma matriz) usando o atributo shape ou através da função `reshape`:

```
v = np.array([1,2,3,4])
v.shape = (2,2)
print(v)
```

```
[[1 2]
 [3 4]]
```

```
v = np.array([1,2,3,4]).reshape(2,2)
print(v)
```

```
[[1 2]
 [3 4]]
```

O número de eixos (ou dimensões) de um array é dado pelo atributo `ndim`, enquanto o número total de elementos é dado por `size`:

```
v = np.array(range(50)).reshape(2,5,5)

print('Shape = ', v.shape)
print('Número de dimensões = ', v.ndim)
print('Número de elementos = ', v.size)
```

```
Shape =  (2, 5, 5)
Número de dimensões =  3
Número de elementos =  50
```

As funções `zeros`, `ones` e `diag` são muito convenientes para a criação de arrays. Mais especificamente, a função `zeros` cria um array que contém apenas zeros e o seu argumento de entrada é um tuplo. A função `ones`, de forma similar, cria um array que contém apenas uns e também recebe um tuplo como argumento. Por fim, a função `diag` cria uma matriz diagonal e recebe como argumento uma lista, que representam os valores a colocar na diagonal principal da matriz.

```
V = np.zeros((3,3))
print('V = \n', V)
U = np.ones((3,3))
print('U = \n', U)
D = np.diag([10, 10, 10])
print('D = \n', D)
```

```
V =
 [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
U =
 [[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
D =
 [[10  0  0]
 [ 0 10  0]
 [ 0  0 10]]
```

As funções `arange` e `linspace` permitem a criação de sequências como arrays. A função `arange` retorna valores uniformemente espaçados dentro de um determinado intervalo. Ou seja, junto com um valor inícial e um valor final parada, é também decidido o valor de incremento. Por sua vez, a função `linspace` retorna números espaçados de modo uniforme num intervalo. Ou seja, dado um valor inicial e um valor final, bem como como a quantidade de valores, esta função irá distribuí-los uniformemente num array.

```
v = np.arange(0, 5, 0.5)
u = np.linspace(0, 5, 10)
print("v =", v)
print("u =", u)
```

```
v = [0.  0.5 1.  1.5 2.  2.5 3.  3.5 4.  4.5]
u = [0.         0.55555556 1.11111111 1.66666667 2.22222222 2.77777778
 3.33333333 3.88888889 4.44444444 5.        ]
```

As operações sobre os arrays (*, - , + , /, **) são realizadas elemento a elemento:

```
v = np.array([10,20,30])
u = np.array([2,2,2])
w = u+v
print(w)
```

```
[12 22 32]
```

A média, menor valor, maior valor, entre outros, possuem funções específicas:

```
x = np.arange(10)
media = x.mean()
menor_valor = np.min(x)
arg_max = np.argmax(x)
print("Média =", media)
print("Menor valor =", menor_valor)
print("Arg max =", arg_max)
```

```
Média = 4.5
Menor valor = 0
Arg max = 9
```

Em NumPy, matrizes são arrays com duas dimensões:

```
A = np.array([[10, 20], [30, 40]])
print(A)
```

```
[[10 20]
 [30 40]]
```

Ainda relativamente às matrizes, existem algumas funções específicas. A função `eye` cria uma matriz identidade. Por sua vez, a função `diag` cria uma matriz diagonal.

```
I = np.eye(5)
print(I)
```

```
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]
```

```
D = np.diag(np.arange(5))
print(D)
```

```
[[0 0 0 0 0]
 [0 1 0 0 0]
 [0 0 2 0 0]
 [0 0 0 3 0]
 [0 0 0 0 4]]
```

Relativamente às operações sobre matrizes, a multilplicação de vetores e matrizes é realizada por meio da função `dot`:

```
A = np.ones((2,2))
B = 10*np.ones((2,2))
C= np.dot(A,B)
print(C)
```

```
[[20. 20.]
 [20. 20.]]
```

A transposição de matrizes pode ser feita com a função `transpose` ou simplesmente com a letra "T":

```
A = np.arange(4).reshape(2,2)
print("Transposta de A =\n", A.transpose())
print("Transposta de A =\n", A.T)
```

```
Transposta de A =
 [[0 2]
 [1 3]]
Transposta de A =
 [[0 2]
 [1 3]]
```

2 Exercícios

Com esta ficha de trabalho pretende-se apresentar alguns problemas exemplificativos da utilização de estruturas de controlo repetitivas e de algumas técnicas específicas, relacionadas com a sua utilização, para a resolução de problemas.

1. Faça duas funções: uma que faça a leitura dos elementos de uma matriz e outra que faça a impressão formatada dessa mesma matriz. Nota: a primeira função deve receber inicialmente a dimensão m e n da matriz e só depois deve pedir os valores a inserir na matriz ao utilizador.

2. Faça um algoritmo que solicite ao utilizador números e os armazene em uma matriz 6×6. Em seguida, crie um vetor que armazene os elementos da diagonal principal da matriz.

3. Tendo uma matriz 10×10 preenchida com valores aleatórios entre 10 e 50 (use a biblioteca `random`), mostre a média dos elementos da diagonal secundária.

4. Dada uma matriz, indicar se se trata de uma matriz triangular superior (matriz quadrada com todos os valores debaixo da diagonal a zero).

5. Dada uma matriz, indicar quantos valores existem superiores à média.

6. Dada uma matriz, indicar se é simétrica (simetria de valores relativamente à diagonal)

7. Dada uma matriz e um valor, pretende-se uma matriz que seja o produto dos dois.

8. Dada uma matriz pretende-se a matriz transposta (as linhas passam a ser as colunas).

9. Dadas duas matrizes pretende-se a sua soma (têm de ter as mesmas dimensões).

10. Uma fábrica possui 5 unidades produtivas. Cada unidade produtiva tem na sua estrutura um conjunto de 10 máquinas. Dado o rendimento de cada uma das máquinas de cada unidade produtiva, indicar qual a unidade produtiva com maior rendimento e a máquina com menor rendimento (obs.: cada máquina é identificada por um número de 1 a 10 e pela unidade a que pertence, de 1 a 5).

11. Resolva os exercícios 2, 3, 4, 5, 6, 7, 8 e 9 usando o Numpy.