Eletrónica (2022/2023)

Ficha Prática N.º 1

Introdução ao *Python* e à biblioteca *numpy*

O *Python* é uma linguagem de programação de alto nível que possui um modelo de desenvolvimento comunitário e aberto. Para além da sua biblioteca padrão, possui várias bibliotecas extra desenvolvidas por terceiros. Algumas das bibliotecas extra mais relevantes no contexto desta unidade curricular são o *numpy* e o *matplotlib*.

O ***numpy*** (***Num****erical* ***Py****thon* ou *Python* numérico) é uma biblioteca extra da linguagem *Python* que suporta *arrays* e matrizes multidimensionais. A referida biblioteca disponibiliza uma vasta coleção de funções matemáticas para trabalhar com as referidas estruturas, sendo, portanto, muito útil para resolver sistemas de equações.

A ***matplotlib*** é uma biblioteca extra da linguagem *Python* que suporta a criação de gráficos e a visualização de dados em geral. Possui uma usabilidade semelhante ao programa de computação numérica *MATLAB*. A referida biblioteca disponibiliza uma vasta coleção de funções matemáticas que permitem a construção de gráficos de forma simples, sendo, portanto, muito útil para visualizar e prever o comportamento de alguns sistemas eletrónicos que serão analisados nesta unidade curricular.

Nesta aula pretende-se que os alunos aprendam a instalar e a dominar as principais funções providenciadas por este software e que permitirão aos alunos compreender o principio de funcionamento dos sistemas eletrónicos em análise.

1. Instalação do *Phyton*

A versão do *Python* que será utilizada é a 3.7.5., sendo que o ficheiro de instalação que se encontra no moodle se destina a máquinas de 64 *bits* com o sistema operativo *Windows*. Se a máquina apresentar características diferentes das indicadas deverá o aluno descarregar o ficheiro de instalação adequado às características da sua máquina e para o efeito deve aceder a sítio da internet: https://www.python.org/

O processo de instalação do *Python* é bastante simples. Assim, deve:

- Selecionar a caixa de seleção "Adicionar Python 3.7 ao PATH".
- Executar a instalação como administrador.
- Selecionar a instalação do *Python* com as configurações padrão (já inclui o *IDLE*, *pip* e a documentação).

O *Python* possui vários ambientes integrados (*IDE*) que oferecem suporte à programação, tais como o *PyCharm* ou o *Microsoft Visual Studio*. No entanto, a maioria destes *IDE* apresentam muitas ferramentas que não serão necessárias no âmbito desta unidade curricular, motivo pelo qual será utilizado um *IDE* bastante mais simples: o *IDLE*

(*Integrated Development and Learning Environment*). O *IDLE* é instalado de forma automática, não sendo, portanto, necessário realizar mais nenhum passo [1].

Para verificar se a instalação foi realizada corretamente deve clicar no menu Iniciar (*Windows*) e ir a todos os aplicativos. O referido menu fornece uma lista alfabética de todos os aplicativos, nomeadamente o *Python* 3.7.5. Deve clicar na opção *IDLE* (Fig. 1).

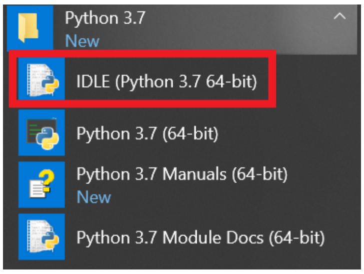

Fig. 1 – *IDLE* - *Integrated Development and Learning Environment*

Em seguida deve executar o código apresentado na Fig. 2.

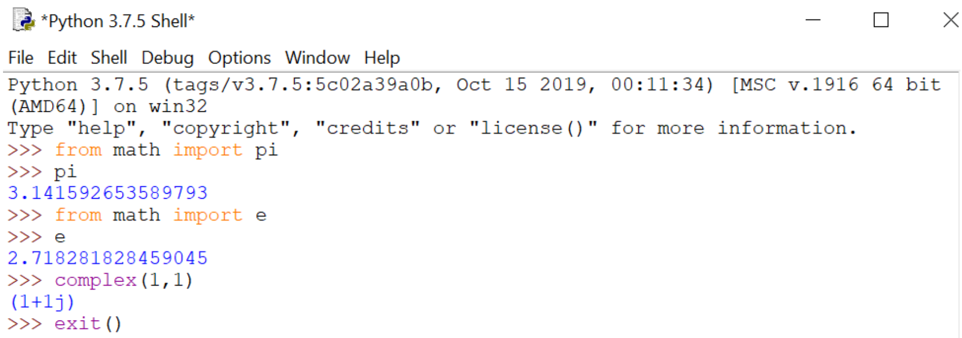

Fig. 2 – Definindo algumas constantes matemáticas.

O *Python* encontra-se corretamente instalado se após executar o código da figura anterior não ocorrer qualquer tipo de erro.

2. Instalação das Bibliotecas *numpy* e *matplolib*

Seguidamente apresentam-se as instruções que deve utilizar para instalar as Bibliotecas extra *matplotlib* e *numpy* do *Python*.

A instalação das referidas Bibliotecas não requer o seu download separado, sendo a sua instalação realizada através *prompt* de comando do Windows, o qual será utilizado para executar o *pip* (**p**rograma de **i**nstalação do ***P**ython*). Assim, num primeiro momento deve aceder à linha de comando do *Windows* (*Command prompt*) na qualidade de administrador. Seguidamente deve executar os seguintes comandos [2]:

- \>> python -mpip install -U pip
- \>> python -mpip install -U numpy
- \>> python -mpip install -U matplotlib
- \>> python -mpip install -U nose

Como foi referido anteriormente, o *pip* (*Python Installation Program*) é o programa de instalação do *Python*. Este programa permite encontrar pacotes na *web* e manter as versões consistentes entre si. A instalação de ambas as Bibliotecas é automática e faz com que a versão correta do *numpy* seja instalada. Os referidos comandos acedem à Internet para recuperar arquivos de um centro de distribuição *Python* online.

Para verificar se as referidas Bibliotecas foram instaladas deve correr o seguinte comando: >> pip list.

3. Biblioteca *numpy*

A Biblioteca *numpy* é uma biblioteca composta por objetos e rotinas que permitem realizar diversas operações matemáticas e lógicas com vetores e matrizes. Nesta secção apresentam-se as funções mais relevantes, no contexto desta unidade curricular, que permitem realizar diversas operações com vetores e matrizes.

Antes de executar os comandos associados à Biblioteca *numpy*, na linha de comando do *IDLE*, é fundamental importar a referida Biblioteca. Deste modo, deve utilizar o seguinte comando:

- >> import numpy.as np[1]

3.1. Operações sobre vetores

Para criar uma matriz deve utilizar o comando *array*. Por exemplo, suponha que pretende criar um vetor v = [1 2 3 4 5 6]. Deve utilizar o comando:

- >> v1 = np.array([1, 2, 3, 4, 5, 6])

Um vetor é uma matriz unidimensional. Neste caso o vetor é composto por números inteiros (*int*). Para identificar o tipo de dados que compõem a matriz deve utilizar o comando *type*.

- >> v1.dtype[2]

No entanto o vetor pode ser composto por números reais (*float*), para o efeito deve preencher o vetor com números reais:

- >> v2 = np.array([1.1, 2.2, 3.3, 4.4])

Ao executar o comando *v2.dtype* o resultado será igual a '*float64*'.

Ao criar o *array* automaticamente o tipo dos elementos que o compõem fica definido. Desta forma, se atribuir um numero real ao vetor v1, este será interpretado como sendo inteiro, sendo, portanto, considerada apenas a sua componente inteira. Por exemplo, considere que pretende substituir o segundo elemento do vetor v1 ('2') por ('5.6'). O segundo elemento[3] de v1 será igual a 5. Seguem os comandos que deverá executar.

- >> v1[1] = 5.6
- >> v1

Para realizar operações matemáticas sobre vetores pode utilizar os comandos:

- + → soma.
- - → subtração.
- * → multiplicação.
- / → divisão.
- ** → potência.

Os referidos comandos aplicam-se posição a posição. Considere o seguinte exemplo, pretende-se somar os vetores: v5 = v3 + v4 = [1 2 3 4] + [4 5 6 7] = [5 7 9 11]

- >> v3 = np.array([1,2,3,4])

---

[1] Esta instrução permite criar um objeto do tipo *numpy*.
[2] No computador os números são representados em binário, sendo utilizado para o efeito um conjunto específico de *bits* para a sua representação. Por exemplo: '*int32*' significa que cada um dos elementos que compõe o vetor necessita de *32 bits*.
[3] A primeira posição de um vetor corresponde à posição *0*.

- >> v4 = np.array([4,5,6,7])
- >> v5 = v3 + v4

O mesmo raciocínio pode ser aplicado às restantes operações matemáticas. Aconselha-se o aluno a testar as restantes operações.

As operações entre matrizes e números escalares impõem que a referida operação seja aplicada a todos os elementos que compõem a matriz. Por exemplo, suponha que pretende somar 10 a todos os elementos do vetor v5, deve executar a seguinte operação:

- >> v5 = v5+10
  Ao executar o comando obtém: v5 = [15 17 19 21]. Aconselha-se o aluno a testar as restantes operações.

Os vetores podem ser definidos automaticamente se o espaçamento entre os diferentes elementos for sempre o mesmo. Para o efeito deve utilizar o comando: np.arange(início, fim+*esp*, *esp*), sendo *esp* o espaçamento.

Suponha que pretende criar um vetor que começa um 0 e termina em 10, cujos os diferentes elementos possuem um espaçamento de 0.1. Deve utilizar o comando:

```
>>> v6=np.arange(0, 10, 0.1)
>>> v6
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. , 1.1, 1.2,
       1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2. , 2.1, 2.2, 2.3, 2.4, 2.5,
       2.6, 2.7, 2.8, 2.9, 3. , 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8,
       3.9, 4. , 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 5. , 5.1,
       5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 6. , 6.1, 6.2, 6.3, 6.4,
       6.5, 6.6, 6.7, 6.8, 6.9, 7. , 7.1, 7.2, 7.3, 7.4, 7.5, 7.6, 7.7,
       7.8, 7.9, 8. , 8.1, 8.2, 8.3, 8.4, 8.5, 8.6, 8.7, 8.8, 8.9, 9. ,
       9.1, 9.2, 9.3, 9.4, 9.5, 9.6, 9.7, 9.8, 9.9])
```

Fig. 3 – Criar vetor de forma automática (início=0, fim=10, espaçamento=0.1).

3.2. Dividindo vetores (*slicing*)

Existem situações em que é necessário fatiar um vetor, ou seja, recolher só certas partes do vetor. Nestes casos deve ser utilizado o seguinte comando.

- vetor[inicio:fim+*esp*:*esp*], sendo *esp*=espaçamento[4].

Suponha que pretende criar um vetor de números inteiros cujo primeiro elemento é o -100 e o último elemento é o 100.

- >> vetor=np.arange(-100,101,1)

Pretende-se representar apenas os elementos positivos, para o efeito, deve utilizar o comando:

- >> vetor[100 :]
  Todos os valores de 0 a 100.

Pretende-se representar apenas os elementos negativos, para o efeito deve utilizar o comando[5]:

- >> vetor[: -100]
  Todos os valores de -100 a 0.

[4] Quando se observa o vetor do seu início, a primeira posição corresponde ao valor 0. À medida que nos deslocamos para o fim do vetor deve-se adicionar o valor um ao índice.

[5] Quando se observa o vetor do seu fim, à última posição do vetor que corresponde ao valor -1. À medida que nos deslocamos para o início do vetor deve-se subtrair o valor um ao índice.

Pretende-se representar apenas os elementos pares positivos, para o efeito deve utilizar o comando:

- >> vetor[100 : : 2]

Pretende-se representar apenas os elementos ímpares negativos, para o efeito deve utilizar o comando:

- >> vetor[1 :-101 : 2]

3.3. Funções matemáticas

O *numpy* fornece um conjunto de funções matemáticas[6], tais como:

- sin → seno.
- cos → cosseno.
- tan → tangente.
- arcsin → ângulo do seno.
- arccos → ângulo do cosseno.
- arctan → ângulo da tangente.
- exp → exponencial.
- log → logaritmo natural.
- log10 → logaritmo de base 10.

Suponha que pretende representar a seguinte função para o intervalo de tempo [0, 10] segundos.

- funcao_seno=5*seno(2*pi *t)

```
>>> from math import pi
>>> import numpy as np
>>> t=np.arange(0,10,0.1)
>>> funcao_seno=5*np.sin(2*pi*t)
>>> funcao_seno
array([ 0.00000000e+00,  2.93892626e+00,  4.75528258e+00,  4.75528258e+00,
        2.93892626e+00,  6.12323400e-16, -2.93892626e+00, -4.75528258e+00,
       -4.75528258e+00, -2.93892626e+00, -1.22464680e-15,  2.93892626e+00,
        4.75528258e+00,  4.75528258e+00,  2.93892626e+00,  1.83697020e-15,
       -2.93892626e+00, -4.75528258e+00, -4.75528258e+00, -2.93892626e+00,
       -2.44929360e-15,  2.93892626e+00,  4.75528258e+00,  4.75528258e+00,
        2.93892626e+00,  3.06161700e-15, -2.93892626e+00, -4.75528258e+00,
       -4.75528258e+00, -2.93892626e+00, -3.67394040e-15,  2.93892626e+00,
        4.75528258e+00,  4.75528258e+00,  2.93892626e+00,  4.28626380e-15,
       -2.93892626e+00, -4.75528258e+00, -4.75528258e+00, -2.93892626e+00,
       -4.89858720e-15,  2.93892626e+00,  4.75528258e+00,  4.75528258e+00,
        2.93892626e+00,  5.51091060e-15, -2.93892626e+00, -4.75528258e+00,
       -4.75528258e+00, -2.93892626e+00, -6.12323400e-15,  2.93892626e+00,
        4.75528258e+00,  4.75528258e+00,  2.93892626e+00,  2.44991258e-14,
       -2.93892626e+00, -4.75528258e+00, -4.75528258e+00, -2.93892626e+00,
       -7.34788079e-15,  2.93892626e+00,  4.75528258e+00,  4.75528258e+00,
        2.93892626e+00, -9.80336420e-15, -2.93892626e+00, -4.75528258e+00,
       -4.75528258e+00, -2.93892626e+00, -8.57252759e-15,  2.93892626e+00,
        4.75528258e+00,  4.75528258e+00,  2.93892626e+00,  2.69484194e-14,
       -2.93892626e+00, -4.75528258e+00, -4.75528258e+00, -2.93892626e+00,
       -9.79717439e-15,  2.93892626e+00,  4.75528258e+00,  4.75528258e+00,
        2.93892626e+00, -7.35407060e-15, -2.93892626e+00, -4.75528258e+00,
       -4.75528258e+00, -2.93892626e+00, -1.10218212e-14,  2.93892626e+00,
        4.75528258e+00,  4.75528258e+00,  2.93892626e+00,  2.93977130e-14,
       -2.93892626e+00, -4.75528258e+00, -4.75528258e+00, -2.93892626e+00])
>>> |
```

Fig. 4 – Gerar a função seno em *numpy*.

[6] https://numpy.org/doc/stable/reference/routines.math.html

3.4. Matrizes

Para criar uma matriz bidimensional deve utilizar o comando *array*. Por exemplo, suponha que pretende criar a matriz.

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Deve utilizar o comando:

- >> A = np.array([[1, 2], [3, 4]])

A dimensão da matriz pode ser determinada pelo *shape*.

- >> A.shape

Para atribuir um valor a posições especificas da matriz A pode utilizar o comando: A[X][Y], representando X a linha e Y a coluna[7]. Suponha que pretende modificar o valor da primeira linha segunda coluna para 13. Para o efeito deve utilizar o comando: A[0][1]=13.

É possível criar matrizes com uma dimensão especifica, com apenas zeros, ou com apenas uns. Para o efeito deve utilizar o comando *zeros* e o comando *ones* respetivamente.

- >> D = np.ones((2,2))
  Cria um matriz D de uns com dimensão 2x2.
- >> E = np.zeros((2,2))
  Cria matriz E de zeros com dimensão 2x2.
- >> F = np.full((2,2),X)
  Cria matriz F de valores inteiros X com dimensão 2x2.

Por exemplo, suponha que pretende criar uma matriz 4x4 com todos os valores iguais a 122.

```
>>> G=np.full((4,4),112)
>>> G
array([[112, 112, 112, 112],
       [112, 112, 112, 112],
       [112, 112, 112, 112],
       [112, 112, 112, 112]])
>>> |
```

Fig. 5 – Matriz 4x4 composta cujos elementos possuem o mesmo valor.

É possível criar a matriz identidade (Fig. 6a) e uma matriz com números aleatórios (Fig. 6b). Para o efeito deve utilizar os comandos:

```
>>> G=np.identity(4)
>>> G
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

matriz identidade

(a)

```
>>> H=np.random.rand(4,4)
>>> H
array([[9.78042829e-01, 2.02689185e-01, 7.14992064e-01, 3.77136604e-01],
       [4.10479894e-01, 3.25954699e-01, 3.09045660e-01, 5.97823436e-01],
       [9.15267921e-04, 1.77317407e-02, 9.78198338e-01, 8.59431722e-02],
       [3.76832699e-01, 7.32577288e-01, 2.07457878e-01, 2.09972424e-01]])
```

matriz de valores aleatórios reais - tipo float

(b)

Fig. 6 – Matriz 4x4 (a) identidade e (b) com valores aleatórios reais.

[7] A primeira linha e a primeira coluna correspondem ao valor 0.

3.5. Dividindo Matrizes (*slicing*)

Existem situações em que é necessário fatiar uma matriz, ou seja, recolher só certas partes de uma matriz. Nestes casos deve ser utilizado o seguinte comando.

- matriz[vetor das linhas: vetor das colunas], vetor=inicio:fim+esp:esp, com esp=espaçamento

Suponha que possui a seguinte matriz:

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

Pretende-se representar apenas a primeira linha, para o efeito deve utilizar o comando:

- >> A[0:1, :]

Pretende-se representar apenas a terceira coluna, para o efeito deve utilizar o comando:

- >> A[: , 2:3]

Pretende-se representar a sub-matriz 2x2, que representa o canto inferior esquerdo, para o efeito deve utilizar o comando:

- >> A[2:4 , 2:4]

Pretende-se representar a sub-matriz 2x2, composta pelas linhas pares (0,2) e colunas (1,3) ímpares, para o efeito deve utilizar o comando:

- >> A[0:4:2 , 1:5:2]

Chama-se a atenção que as sub-matrizes indicadas correspondem a partes da memória onde se encontra a matriz A, o que significa que o comando:

- >> B = A[0:4:2 , 1:5:2]

Não cria uma nova matriz (B), mas simplesmente, gera um ponteiro que aponta para uma certa posição da memória onde se encontra a matriz A. Desta forma, o comando:

- >> B[0][0]=112

Irá modificar o elemento que se encontra na posição (0,1) da matriz A.

Para criar uma nova matriz deve utilizar o comando *copy*. Assim, após atribuir o ponteiro para a matriz A, deve executar o seguinte comando.

>> B = B.copy()

Suponha que pretende criar a seguinte matriz recorrendo aos comandos definidos anteriores.

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 4 & 0 & 4 \\ 1 & 0 & 4 & 4 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

Apresentam-se em seguida os comandos do tipo *slicing* que permitem gerar a matriz anterior de forma automática.

```
>>> import numpy as np
>>> v=np.arange(1,5)
>>> v
array([1, 2, 3, 4])
>>> A=np.ones([4,4])
>>> A
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
>>> A=v*A
>>> A
array([[1., 2., 3., 4.],
       [1., 2., 3., 4.],
       [1., 2., 3., 4.],
       [1., 2., 3., 4.]])
>>> A[1:3,1:3]=np.identity(2)*4
>>> A
array([[1., 2., 3., 4.],
       [1., 4., 0., 4.],
       [1., 0., 4., 4.],
       [1., 2., 3., 4.]])
>>>
```

Fig. 7 – Gerar uma matriz de através de comandos do tipo *slicing*.

Este tipo de comandos pode ser muito útil quando as matrizes que se pretendem gerar são muito extensas.

3.6. Matrizes bidimensionais – operações elemento a elemento

Os operadores matemáticos indicados anteriormente (+, -, *, /, **, sin, cos, log, exp, etc…) são aplicados elemento a elemento.

Considere as seguintes matrizes:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad e \quad B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

A operação:
>> C = A + B
Produz a soma dos elementos individuais das matrizes, ou seja:

$$C = \begin{bmatrix} 1+4 & 2+2 \\ 3+3 & 4+1 \end{bmatrix} = \begin{bmatrix} 5 & 4 \\ 6 & 5 \end{bmatrix}$$

O mesmo raciocínio aplica-se às restantes operações matemáticas.

Podem ser utilizados escalares para realizar operações matemáticas com matrizes. A referida operação irá afetar todos os elementos da matriz de igual forma.

A operação:

>> C = C/2
A operação anterior produz o resultado:

$$C = \begin{bmatrix} \frac{5}{2} & \frac{4}{2} \\ \frac{6}{2} & \frac{5}{2} \end{bmatrix} = \begin{bmatrix} 2.5 & 2.0 \\ 3.0 & 2.5 \end{bmatrix}$$

O mesmo raciocínio aplica-se às restantes operações matemáticas.

3.7. Matrizes bidimensionais – álgebra linear

Os métodos que serão apresentados nesta secção permitem obter a resolução de sistema de equações lineares, sendo por isso de extrema importância na análise de circuitos elétricos.

- Para realizar o produto de matrizes deve ser utilizada a função *matmul*(A,B), sendo *A* e *B* matrizes.
  Relembramos que o produto de duas matrizes deve respeitar a regra: *o nº de colunas da primeira matriz deve ser igual ao nº de linhas da segunda matriz*.
  Suponha que pretende multiplicar a seguintes matrizes:

$$A = \begin{bmatrix} 1 & 4 \\ 3 & 2 \end{bmatrix} \quad B = \begin{bmatrix} 3 & 5 & 5 \\ 1 & 2 & 8 \end{bmatrix} \Rightarrow C = A \times B = \begin{bmatrix} 7 & 13 & 37 \\ 11 & 19 & 31 \end{bmatrix}$$

  Para o efeito deve utilizar os comandos:

```
>>> import numpy as np
>>> A=np.array([[1,4],[3,2]])
>>> A
array([[1, 4],
       [3, 2]])
>>> B=np.array([[3,5,5],[1,2,8]])
>>> B
array([[3, 5, 5],
       [1, 2, 8]])
>>> C=np.matmul(A,B)
>>> C
array([[ 7, 13, 37],
       [11, 19, 31]])
>>> |
```

Fig. 8 – Álgebra linear: multiplicação de duas matrizes

- Para calcular o determinante de uma matriz pode utilizar o seguinte comando *linalg.det*
  Suponha que pretende calcular o determinante da matriz A. Deve utilizar o comando: >> determinante_A = np.linalg.det(A)

- Para calcular a inversa de uma matriz pode utilizar o seguinte comando *linalg.inv*
  Suponha que pretende calcular a inversa da matriz A. Deve utilizar o comando: >> inversa_A = np.linalg.inv(A)

3.8. Regra de *Cramer*

A regra de *Cramer* é um método popular utilizado na resolução de sistemas de equações lineares.

$$\begin{cases} a_{11} \cdot x_1 + a_{12} \cdot x_2 + a_{13} \cdot x_3 = b_1 \\ a_{21} \cdot x_1 + a_{22} \cdot x_2 + a_{23} \cdot x_3 = b_2 \\ a_{31} \cdot x_1 + a_{32} \cdot x_2 + a_{33} \cdot x_3 = b_3 \end{cases} \Rightarrow A \times x = b \Rightarrow \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

A regra de *Cramer* exige o cálculo de determinantes. Segundo a regra de *Cramer*, é possível calcular a incógnita que se encontra na linha *i* da matriz *x*, através da equação [3],[4]:

$$x_i = \frac{\det(A_i)}{\det(A)}$$

em que A_i corresponde a uma matriz cujas colunas são iguais às colunas da matriz A, exceto a coluna i que foi substituída pelo vetor coluna b [3],[4].

Assim, no caso do sistema de equações anterior as soluções poderiam ser obtidas através das equações [3],[4]:

$$x_1 = \frac{\begin{vmatrix} b_1 & a_{12} & a_{13} \\ b_2 & a_{22} & a_{23} \\ b_3 & a_{32} & a_{33} \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}}, \quad x_2 = \frac{\begin{vmatrix} a_{11} & b_1 & a_{13} \\ a_{21} & b_2 & a_{23} \\ a_{31} & b_3 & a_{33} \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}} \quad x_3 = \frac{\begin{vmatrix} a_{11} & a_{12} & b_1 \\ a_{21} & a_{22} & b_2 \\ a_{31} & a_{32} & b_3 \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}}$$

Suponha que pretende calcular a solução do seguinte sistema de equações:

$$\begin{cases} 3 \,.\, x_1 + 5 \,.\, x_2 + 7 \,.\, x_3 = 3 \\ 4 \,.\, x_1 + 3 \,.\, x_2 + 1 \,.\, x_3 = 5 \\ 6 \,.\, x_1 + 7 \,.\, x_2 + 9 \,.\, x_3 = 1 \end{cases} \Rightarrow A \times x = b \Rightarrow \begin{bmatrix} 3 & 5 & 7 \\ 4 & 3 & 1 \\ 6 & 7 & 9 \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \\ 1 \end{bmatrix}$$

Para o efeito deve executar os seguintes comandos:

```
>>> import numpy as np
>>> A=np.array([[3,5,7],[4,3,1],[6,7,9]])
>>> A
array([[3, 5, 7],
       [4, 3, 1],
       [6, 7, 9]])
>>> b=np.array([3,5,1])
>>> b
array([3, 5, 1])
>>> A1=A.copy()
>>> A2=A.copy()
>>> A3=A.copy()
>>> A1[:,0]=b
>>> A2[:,1]=b
>>> A3[:,2]=b
```

1º passo

```
>>> A1
array([[3, 5, 7],
       [5, 3, 1],
       [1, 7, 9]])
>>> A2
array([[3, 3, 7],
       [4, 5, 1],
       [6, 1, 9]])
>>> A3
array([[3, 5, 3],
       [4, 3, 5],
       [6, 7, 1]])
```

2º passo

```
>>> x1=np.linalg.det(A1)/np.linalg.det(A)
>>> x2=np.linalg.det(A2)/np.linalg.det(A)
>>> x3=np.linalg.det(A3)/np.linalg.det(A)
>>> x1
-3.1999999999999998
>>> x2
6.9999999999999995
>>> x3
-3.1999999999999998
```

3º passo

Fig. 9 – Cálculo da solução de um sistema de equações lineares com recurso ao *numpy* e à regra de *Cramer*

Como alternativa à regra de *Cramer* poderia ser utilizada a equação:

- X = A^{-1} . b

A aplicação da equação anterior implica a utilização dos métodos *linalg.inv(A) e matmul(A,B)* como se pode constatar na figura seguinte:

```
>>> solucao=np.matmul(np.linalg.inv(A),b)
>>> solucao
array([-3.2,  7. , -3.2])
```

Fig. 9 – Cálculo da solução de um sistemas de equações lineares com recurso à equação: X = A^{-1} . b

O *numpy* fornece igualmente um método que permite calcular diretamente as soluções do um sistema de equações lineares.

- x = np.linalg.solve(A,b)

```
>>> solucao=np.linalg.solve(A,b)
>>> print(solucao)
[-3.2  7.  -3.2]
>>> |
```

Fig. 10 – Cálculo da solução de um sistemas de equações lineares com recurso à função *linalg.solve*

4. Exercícios

Utilize o *IDLE* conjuntamente com os comandos do *numpy* adequados para realizar os seguintes exercícios:

1. Crie um vetor composto pelos seguintes elementos e identifique o tipo de elementos que compõem o vetor:

$$v1 = \begin{bmatrix} 1 & 6 & 9 & 10 & 4 & 1 & 5 \end{bmatrix}$$

2. Crie um vetor composto pelos seguintes elementos e identifique o tipo de elementos que compõem o vetor:

$$v2 = \begin{bmatrix} 1.6 & 6.1 & 9.9 & 10.1 & 4.5 & 1.7 & 0.7 \end{bmatrix}$$

3. Atribua ao segundo elemento de v1 o valor -1 e o quinto elemento de v2 o valor 0.123.
4. Multiplique todos os elementos do vetor v1 por 4 e divida todos os elementos do vetor v2 por 5.
5. Gere um vetor v3 que resulta da divisão, ponto a ponto, do vetor v2 com v1 e identifique o tipo de dados que compõem o novo vetor (v3).
6. Gere um vetor (v4) composto por 100 elementos com valor inicial igual a 0 e final igual a 99.
7. Copie os primeiros 30 elementos do vetor v4 para um outro vetor v5.
8. Copie os últimos 30 elementos do vetor v4 para um outro vetor v6.
9. Obtenha um vetor v7 que represente a evolução temporal da seguinte função:

$$\text{funcao} = 4 \times \sin(2 \times \text{pi} \times 0.01 \times v4)$$

10. Crie uma matriz 4x4 de uns.
11. Crie a matriz A.

$$A = \begin{bmatrix} 1.4 & 4.5 & 6.7 \\ 3.4 & 6.9 & 1.2 \\ 5.6 & 2.1 & 6.2 \end{bmatrix}$$

12. Crie um vetor (v8) composto pela primeira linha da matriz A
13. Crie um vetor (v9) composto pela terceira coluna da matriz A

14. Crie a seguinte matriz com recurso a comandos do tipo *slicing*

$$B = \begin{bmatrix} -5 & 0 & 0 & 0 & 0 & 5 \\ -5 & 4 & 0 & 0 & 0 & 5 \\ -5 & 0 & 4 & 0 & 0 & 5 \\ -5 & 0 & 0 & 4 & 0 & 5 \\ -5 & 0 & 0 & 0 & 4 & 5 \\ -5 & 0 & 0 & 0 & 0 & 5 \end{bmatrix}$$

15. Considere o seguinte sistema de equações:

$$\begin{cases} 4.x_1 + 2.x_2 + 3.x_3 + 6.x_4 + 2.x_5 = 5 \\ 3.x_1 + 4.x_2 + 2.x_3 + 5.x_4 + 8.x_5 = 1 \\ 7.x_1 + 6.x_2 + 5.x_3 + x_4 + 7.x_5 = 6 \\ 2.x_1 + x_2 + 4.x_3 + 5.x_4 + 3.x_5 = 1 \\ 5.x_1 + 7.x_2 + 9.x_3 + 8.x_4 + 5.x_5 = 9 \end{cases}$$

a. Obtenha a solução com recurso à regra de *Cramer*.

b. Obtenha a solução através da manipulação algébrica de matrizes.

c. Obtenha a solução com recurso à função *linalg.solve*.

Referências Bibliográficas

[1] Numpy community (2020), NumPy User Guide - Release 1.19.0, https://numpy.org/devdocs/release/1.19.0-notes.html, acedido em Agosto 2020.

[2] Amaral, Acácio (2021), Eletrónica Aplicada, Edições Silabo, Lisboa, Portugal.

[3] Amaral, Acácio (2017), Electrónica Analógica: Princípios, Análise e Projectos, Edições Silabo, Lisboa, Portugal.

[4] Amaral, Acácio (2015), Análise de Circuitos e Dispositivos Eletrónicos, Publindústria, Porto (2ª edição).

