Métodos Computacionais para a Engenharia Eletrotécnica

Ficha Laboratorial 3 – Funções e *strings*

1. Objetivo

A presente ficha de trabalho laboratorial tem como objetivo a introdução dos seguintes conceitos:

1. Funções.
2. Definição e manipulação de *strings*.

2. Funções

Para além do quase interminável número de funções que o MATLAB disponibiliza na sua configuração base e nas suas várias *toolboxes*, o utilizador pode criar as suas próprias funções. De forma genérica, uma função consiste num ficheiro escrito com a seguinte sintaxe:

```
function [var_out1, var_out2, ...] = nome_da_funcao(var_in1, var_in2, ...)
% As primeiras linhas de comentário são usadas como texto de ajuda para
% a função sempre que for usado o comando help na janela de comando
...
%% código %%
...
end
```

Deverão ser observadas as seguintes considerações:

1. Deve criar um ficheiro por função
2. Recomenda-se vivamente que o nome do ficheiro tenha o mesmo nome da função. Se assim não o for, o MATLAB irá utilizar o nome do ficheiro em vez do nome da função, o que pode gerar alguma confusão.
3. A primeira instrução da função tem que ser impreterivelmente a definição da função através da palavra reservada **`function`**. Pode, no entanto, escrever linhas de comentário antes da palavra reservada **`function`**. Neste caso, é este o texto que é usado quando invocado o **`help`** da função.

Veja-se o seguinte exemplo da função **`media.m`** que devolve a média dos valores contidos num vetor:

```
function y = media(x)
if ~isvector(x)                         % verifica se x é um vetor
    error('A entrada deve ser um vetor') % passa uma mensagem de erro na janela de comando
end
y = sum(x)/length(x);
end
```

A utilização desta função na janela de comando pode, por exemplo, devolver os seguintes resultados:

```
>> media(10)
ans =
    10
>> media(1:10)
ans =
    5.5000
>> media([1 2; 3 4])
Error using media
A entrada deve ser um vetor
```

Como se pode observar, esta função verifica, através da função **`isvector()`**[1], se a variável de entrada é um vetor. Se não for um vetor, é devolvida uma mensagem de erro através da função **`error()`**, a qual força o final da função.

3. Strings

3.1 Definição de *strings*

Uma *string* corresponde a um vetor de carateres, delimitados por plicas, ou seja, é um vetor do tipo ***char***. As seguintes instruções exemplificam a criação de algumas *strings,* atribuídas a algumas variáveis:

```
>> str_vazia = '';           % gera uma string vazia (apenas duas plicas!!)
>> str1 = 'Hello world';     % não podia faltar, não é?
>> str2 = 'MCEE 2022/2023';  % também não podia faltar, não é?
>> str3 = '0303456';         % string com carateres numéricos
```

Outra maneira de atribuir uma *string* a uma variável é, como se mostrou na ficha anterior, com recurso à função **`input()`**:

```
>> nome = input('Introduza o seu nome: ', 's'); % não esquecer 's' no 2º argumento
```

Estas variáveis podem ser manipuladas como qualquer outra variável em MATLAB:

```
>> length(str_vazia)
ans =
     0

>> length(str1)
ans =
    11

>> sub_str2 = str2(1:4)
sub_str2 =
    'MCEE'
```

Podemos criar uma matriz composta por *strings*. No entanto, é preciso não esquecer que uma *string* não é mais do que um vetor de carateres, pelo que todas as linhas da matriz a criar devem conter o mesmo número de colunas, i.e., cada linha deve conter o mesmo número de carateres:

[1] **`isvector`** faz parte de um conjunto de funções designadas de *detect state functions*. Na linha de comandos, faça **`doc is*`** para obter mais informações. Chama-se ainda a atenção particular às funções **`ischar`**, **`isempty`**, **`isscalar`** , **`ismatrix`** , e **`isnumeric`**, para nomear algumas mais comummente utilizadas.

```
>> M = ['MCEE'; '2023'; '2024']
M =
  3×4 char array
    'MCEE'
    '2023'
    '2024'
>> size (M)
ans =
     3     4

>> N = ['short'; 'Supercalifragilisticexpialidocious']
Error using vertcat
Dimensions of arrays being concatenated are not consistent.
```

3.2 Manipulação de *strings*

A **concatenação** de *strings* consiste em juntar duas ou mais strings numa única *string.* Tal pode ser conseguido definindo um novo vetor que inclua as *strings* pretendidas, ou recorrendo à função **`strcat()`**:

```
>> str1 = 'Início  '; % dois espaços no fim
>> str2 = '   Fim';   % três espaços no início
>> v1 = [str1, str2]
v1 =
    'Início     Fim'
>> length(v1)
ans =
    14

>> v2 = strcat(str1, str2)
v2 =
    'Início   Fim'

>> length(v2)
ans =
    12
```

Existe uma diferença entre estas duas alternativas de concatenar *strings*. Na primeira, todos os espaços que se encontrem quer no início, quer no final de cada *strings* são incluídos na *string* concatenada. Na segunda, todos os espaços no final de cada string são removidos. Isto fica mais evidente no exemplo seguinte:

```
>> s1 = 'MCEE    ';
>> s2 = ' 2023   ';
>> s3 = '/2024   ';
>> strcat(s1, s2, s3)
ans =
    'MCEE 2023/2024'
```

A função **`sprintf`** permite escrever dados formatados na forma de um vetor de carateres. Dito de outra forma, funciona da mesma forma que a função **`fprintf`**, previamente explorada, mas, em vez de apresentar uma mensagem no ecrã, gera uma *string*. Todas as opções de formatação consideradas na função **`fprintf`** são válidas em **`sprintf`**:

```
>> t = 0.1;
>> q = 15.32;
>> msg1 = sprintf('Carga do condensador, q(t = %.1f) = %.2f', t, q)
msg1 =
    'Carga do condensador, q(t = 0.1) = 15.32'
```

Existem também funções que permitem converter os carateres de uma *string* em carateres maiúsculos, **`upper()`**, e em carateres minúsculos, **`lower()`**:

```
>> str = 'OlHa uMa StRiNg!';
>> lower(str)
ans =
    'olha uma string!'

>> upper(str)
ans =
    'OLHA UMA STRING!'
```

3.3 Comparação de *strings*

Ao comparar *strings* pode cair-se no erro de usar o operador relacional "**==**". Tal como referido na ficha anterior, os operadores relacionais são aplicados elemento a elemento, isto é, neste caso, a comparação é feita carater a carater, sendo por isso o resultado um vetor de valores lógicos:

```
>> str1 = 'gato';
>> str2 = 'rato';
>> str3 = 'GATO';
>> str1 == str2
ans =
  1×4 logical array
   0   1   1   1

>> str1 == str3
ans =
  1×4 logical array
   0   0   0   0
```

Se se pretender obter um resultado lógico *verdade* (as *strings* são idênticas) ou falso (as *strings* são diferentes), deve usar a função **`strcmp()`** ou **`strcmpi()`**. A primeira compara as *strings* considerando a diferença entre as letras maiúsculas e minúsculas, a segunda função ignora esta diferença:

```
>> strcmp(str1, str2)
ans =
  logical
   0

>> strcmp(str1, str3)
ans =
  logical
   0

>> strcmpi(str1, str3) % ignora se as letras são maiúsculas/minúsculas
ans =
  logical
   1
```

Para partir uma *string* em sub-*strings*, podemos recorrer à função **`strtok`**. Esta função procura o primeiro carater *delimitador* da *string*, i.e., o carater que define o ponto onde a *string* será dividida, e nesse ponto divide a *string* em duas: a primeira *string* consiste na parte inicial da *string* original até ao carater delimitador, com este excluído; a segunda *string* consiste no resto da *string* original, incluindo o carater delimitador. Por omissão,

o carater delimitador é considerado o carater espaço. No entanto, o utilizador pode definir outro carater que lhe seja mais conveniente. Se o carater delimitador estiver no início da *string*, este é ignorado:

```
>> str = '  Universidade de Coimbra';
>> [str1, resto] = strtok(str)
str1 =
    'Universidade'
resto =
    ' de Coimbra'

>> [str2, resto] = strtok(resto)
str2 =
    'de'
resto =
    ' Coimbra'

>> [str3, resto] = strtok(resto)
str3 =
    'Coimbra'
resto =
  0×0 empty char array
```

Fica ainda, no final desta secção, uma pequena tabela com uma descrição sumária de algumas funções mais usualmente utilizadas na manipulação de *strings,* para além daquelas aqui apresentadas.

Uma última nota para o facto de, desde a versão R2016b, ter sido introduzido o **tipo de dados string**. As *strings* aqui discutidas são na verdade vetores do tipo **char**, i.e., cada elemento do vetor é apenas um carater. No tipo string, cada elemento da matriz é, de facto, uma *string*. Como o **tipo de dados string** não será aprofundado no âmbito desta unidade curricular, este assunto não será explorado nesta ficha.

Função	Descrição
`blanks(`*n***`)`**	Cria um vetor com *n* carateres de espaço.
`count(`*str, sub_str***`)`**	Conta o número de vezes que o padrão definido em *sub_str* ocorre em *str*.
`deblank(`*str***`)`**	Remove os carateres em branco que se encontrem no final da *string str*.
`erase(`*str, sub_str***`)`**	Apaga em *str* o padrão definido em *sub_str*.
`int2str(`*n***`)`**	Converte o número inteiro *n* num vetor de carateres. Se *n* não for inteiro, é primeiro arredondado.
`isletter(`*str***`)`**	Determina que carateres correspondem a letras na *string str*.
`ispace(`*str***`)`**	Determina que carateres correspondem a espaços na *string str*.
`num2str(`*n***`)`**	Converte o número *n* num vetor de carateres.
`str2num(`*str***`)`**	Converte a *string str* num valor numérico. Se *str* contiver carateres não numéricos, é devolvida uma matriz vazia.
`strncmp(`*s1, s2, n***`)`** **`strncmpi(`***s1, s2, n***`)`**	Compara os primeiros *n* carateres das *strings* s1 e s2. **`strncmpi()`** ignora a diferença entre as maiúsculas e minúsculas.
`strfind(`*str, sub_str***`)`**	Devolve o índice na *string str* onde começa o padrão definido em *sub_str*.
`strrep(`*str, old, new***`)`**	Procura a *sub-string old* na *string str* e substitui pela *sub-string new.*
`strtrim(`*str***`)`**	Remove os carateres em branco que se encontrem no início e no final da *string str*.

4. Atividades

4.1 Atividade 1 – Introdução

Nesta primeira atividade pretende-se implementar a função apresentada no enunciado da atividade e verificar o seu funcionamento através da janela de comando.

a) Crie uma função contendo as seguintes instruções (não esqueça que **o ficheiro deve ter o mesmo nome da função** e ignore a numeração das linhas):

```
1     function [V, P] = voltpower(R, I)
2     % A função voltpower devolve o valor da tensão aos terminais
3     % e a potência dissipada numa resistência R, percorrida por
4     % uma corrente I
5     % Utilização:
6     %     [V, P] = voltpower(R, I)
7     %      V - tensão aos terminais de R (V)
8     %      P - Potência dissipada em R (W)
9     %      R - Resistência elétrica, R >= 0 Ohms (escalar)
10    %      I - Corrente elétrica (A) (vector)
11    V = R * I;
12    P = R * I.^2;
```

b) Execute a função na janela de comando usando os seguintes comandos e verifique se está a funcionar corretamente:

1. `>> voltpower(5, 2)`
2. `>> V = voltpower(5, 2)`
3. `>> [V, P] = voltpower(5, 2)`
4. `>> [V1, P1] = voltpower(5, 1:5)`
5. `>> help voltpower`

c) Como pode observar, o primeiro argumento (**R**) da função deve ser um escalar igual ou maior a zero. Edite a função **voltpower** de forma a que seja apresentado um erro caso se tente passar um valor de **R** que não seja um escalar, i.e., **R** não pode ter mais do que uma linha e uma coluna (*dica: não se esqueça das* detect state functions *disponíveis em MATLAB, referidas na pág. 2)*. Recorra a função **error()** para devolver a mensagem de erro na janela de comando e forçar o fim da função. A função deverá agora ter um comportamento semelhante ao seguinte (note que a linha do erro pode variar, dependendo do código que escrever):

```
>> [V, P] = voltpower([1 2 3],2)
Error using voltpower
O valor de R deve ser um escalar
```

d) Edite novamente a função, agora para devolver uma mensagem de erro caso o utilizador passe um valor negativo de **R**:

```
>> [V, P] = voltpower(-5,2)
Error using voltpower
O valor de R deve ser igual ou maior a zero
```

e) Finalmente, deverá editar a função de forma a verificar se o segundo argumento passado na função é um vetor, i.e., não tem mais do que uma linha ou uma coluna. Mais uma vez, tenha em consideração a utilização de uma das *detect state functions* para testar se **`I`** é ou não um vetor:

```
>> [V, P] = voltpower(5,[1 2; 3 4])
Error using voltpower
I deve ser um escalar ou um vector
```

4.2 Atividade 2

Escreva as seguintes funções:

a) **`x = inch2cm(num)`**: recebe um valor em polegadas (*num*) e devolve o resultado em centímetros (1 polegada = 2.54 cm).

```
>> inch2cm(3)
ans =
    7.6200
```

b) **`[v1, v2] = posneg(v)`**: recebe um vetor de valores (linha ou coluna) e devolve dois vetores: um contendo todos os elementos de **`v`** maiores ou iguais a zero; outro contendo todos os elementos negativos de **`v`**. A função deve ainda devolver uma mensagem de erro caso seja não seja passado um vetor como argumento de entrada (pode admitir escalares).

```
>> [x, y] = posneg([1 -2 3 -4 5 -6 7])
x =
     1     3     5     7
y =
    -2    -4    -6
```

c) **`v = preenchedir(str, n)`**: esta função preenche à direita a *string* **`str`** com carateres espaço até que o seu comprimento seja igual a **`n`**. Se a *string* **`str`** tiver mais do que **`n`** elementos, então ela é truncada, ficando com o comprimento de **`n`** carateres.

```
>> preenchedir('Wasserflasche', 20)
ans =
    'Wasserflasche       '

>> preenchedir('Wasserflasche', 6)
ans =
    'Wasser'

>> preenchedir('Wasserflasche', 0)
Error using preenchedir
n tem de ser maior ou igual a 1

>> preenchedir(['MCEE'; '2023'], 3)
Error using preenchedir
str tem de ser um vetor-linha de carateres
```

4.3 Atividade 3

Uma maneira simples de cifrar mensagens consiste na utilização da primeira letra de cada palavra de uma frase para ocultar a mensagem que se pretende codificar. Escreva o *script* **atividade3.m** que solicite ao utilizador a frase que se pretende descodificar, e que devolve ao utilizador a mensagem que estava codificada.

Para descodificar a mensagem, deverá escrever a função **descodifica(*str*)**, a qual recebe como argumento a frase que contém a frase a descodificar e que devolve a mensagem que estava codificada. Assuma que cada palavra é separada apenas por um carater espaço.

```
Atividade 3
Descodificador de mensagens

Frase a descodificar: Hoje estavam longe, paciência!
Mensagem codificada: Help
```

Dica 1: Considere que o número de palavras numa frase é igual ao número de espaços mais um. Tenha em atenção que a função **count** *permite contar o número de ocorrências de um determinado padrão numa string.*

Dica 2: Recorra à função **strtok** *para passar à palavra seguinte da string.*

5. Fora da sala de aula

5.1 Exercício 1

Escreva a função **[r, t, f] = CoordEsf(x, y, z)** que recebe as coordenadas **x**, **y**, e **z**, e calcula as coordenadas esféricas correspondentes. Os argumentos **x**, **y**, e **z** podem ser vetores, devendo, nesse caso, garantir-se que têm o mesmo número de elementos. A função deverá devolver uma mensagem de erro caso sejam passados valores de **x**, **y**, e **z** que não sejam vetores. Ainda, deverá devolver uma mensagem de erro se os vetores não tiverem o mesmo número de elementos. Os valores de θ e ϕ são devolvidos em graus.

$$r = \sqrt{x^2 + y^2 + z^2}, \quad \theta = \arccos\left(\frac{z}{r}\right), \quad \phi = \arctan\left(\frac{y}{x}\right)$$

```
>> help CoordEsf
   CoordEsf converte valores em coordenadas cartesianas
   em coordenadas esféricas.
     [r, theta, phi] = CoordEsf(x, y, z)
     x, y, e z devem ser vetores com a mesma dimensão
     theta e phi são calculados em graus.

>> [a, b, c] = CoordEsf(0, 2, 3)
a =
    3.6056
b =
   33.6901
c =
    90

>> [a, b, c] = CoordEsf([10 20], [5 0], [-4 2])
a =
   11.8743   20.0998
b =
  109.6857   84.2894
c =
   26.5651         0
```

5.2 Exercício 2

Escreva a função **`B = troca(A, i, j)`** que devolve a matriz **`A`** com as colunas **`i`** e **`j`** trocadas. A matriz **`A`** deve ter pelo menos duas colunas e os argumentos **`i`** e **`j`** passados na função não podem ter valores superiores ao número de colunas de A. O não cumprimento de qualquer destes casos devolve uma mensagem de erro.

```
>> help troca
   troca devolve a matriz A com as colunas i e j trocadas
     M = troca(A, i, j)
     A é uma matriz com pelo menos 2 colunas
     i e j devem ser iguais ou inferiores ao número de colunas de A

>> M = [10  40  70  100;  20  50  80  110; 30  60  90  120]
M =
    10    40    70   100
    20    50    80   110
    30    60    90   120

>> N = troca(M, 2, 4)
N =
    10   100    70    40
    20   110    80    50
    30   120    90    60

>> troca(M, 2, 6)
Error using troca
i e j devem ser iguais ou inferiores ao número de colunas de A
```

5.3 Exercício 3

Escreva a função **`CheckerBoard`** que recebe um valor inteiro *n*, e que devolve uma matriz M_{nxn} que tem a seguinte forma (e.g., *n* = 5):

$$M = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Note que o elemento M(1, 1) é sempre igual a 1, qualquer que seja a dimensão *n* escolhida, devendo *n* ser um inteiro positivo.

*Dica: Este exercício pode ser feito de várias maneiras bastante diferentes. Um caminho pode consistir em criar um vetor para as linhas ímpares e outro para as linhas pares, que podem depois ser replicadas com a função **repmat**. Outra alternativa passa por verificar que os '1' (ou os '0') estão dispostos ao longo de diagonais. Com algum trabalho, consegue obter um ciclo que permita gerar as diagonais necessárias, usando a função **diag**.*

```
>> M = CheckerBoard(1)
M =
     1

>> M = CheckerBoard(2)
M =
     1     0
     0     1
```

```
>> M = CheckerBoard(3)
M =
     1     0     1
     0     1     0
     1     0     1
```

5.4 Exercício 4

A partir da função escrita na atividade 3, escreva a função **`descodifica2`**, a qual permite receber no seu argumento uma *string* que contenha mais do que um espaço a separar as palavras. Esta função deverá, antes de descodificar a mensagem contida na *string,* eliminar os carateres espaço em excesso e, depois, obter a mensagem codificada.

```
>> [msg, clean] = descodifica2('Hoje     estavam  longe,     paciência!')
msg =
    'Help'
clean =
    'Hoje estavam longe, paciência!'
```

5.5 Exercício 5

O jogo de tabuleiro "O Homem que Mordeu o Cão" propicia a criação de algumas frases desconcertantes obtidas pela seleção aleatória das componentes que constituem a estrutura das mesmas. Pretende-se escrever um *script* que componha, de forma aleatória, uma frase com a seguinte estrutura:

"[Onde], [Quem] [O Quê] [A Quem] por causa de [Causa]"

Neste script, deverá definir as matrizes de carateres **`Onde`**, **`Quem`**, **`OQue`**, **`AQuem`** e **`Causa`**. Estas matrizes deverão conter as seguintes linhas de texto:

Onde = {'Na FCTUC', 'Em Coimbra', 'Na Rua Larga', 'Na Portagem'}

Quem = {'homem', 'mulher', 'cão', 'ladrão'}

Que = {'falou com', 'gritou com', 'surpreendeu a', 'mordeu'}

AQuem = {'um polícia', 'um padeiro', 'uma alface', 'um urso polar'}

Causa = {'duma noite mal dormida', 'dum exame difícil', 'do preço da gasolina', 'dos resultados do Benfica'}

Note que cada uma das linhas propostas para estas matrizes têm um número de carateres diferentes, pelo que recomenda a utilização da função **`char`** para gerar cada uma das matrizes. Pode, naturalmente, acrescentar mais elementos às várias matrizes.

O *script* deverá então selecionar aleatoriamente uma linha de cada matriz, e compor uma frase segundo a estrutura atrás indicada. Depois de cada frase, o *script* deverá solicitar ao utilizador se quer repetir, podendo o utilizador indicar escrever 's' ou simplesmente premir <enter> para continuar. Qualquer outra resposta termina o *script*.

```
O Homem que mordeu o cão
Prima uma tecla para continuar...

Em Coimbra, mulher gritou com uma alface por causa duma noite mal dormida
Outra vez [S/n]: S
Na FCTUC, homem gritou com um urso polar por causa dos resultados do Benfica
Outra vez [S/n]:
```

```
Na FCTUC, ladrão falou com um polícia por causa duma noite mal dormida
Outra vez [S/n]: n
```

Dica 1: Pode usar o comando **pause** *para suspender o programa até o utilizador premir uma tecla.*

Dica 2: Considere a utilização da função **isempty** *para testar quando o utilizador decidir apenas premir <enter> para continuar.*

Dica 3: Se optar pela instrução ***strcat*** *para compor a frase, considere usar o valor ASCII 32 para definir carater espaço (****strcat(****str1, **32**, str2****)****).*

5.6 Exercício 6

Escreva o *script* **ppt.m** que permite reproduzir o popular jogo "Pedra, papel ou tesoura". O utilizador seleciona a opção com recurso às letras 'p' (Pedra), 'l' (papeL) e 't' (Tesoura). O jogo é repetido continuamente até o utilizador premir apenas a tecla <enter> sem introduzir qualquer opção. Em cada partida, deverá ser indicada qual foi a seleção do jogador, qual a seleção do computador (gerada de forma aleatória), o resultado da partida, e a pontuação geral dos jogadores.

```
[P]edra, Pape[L] ou [T]esora? (<enter> para terminar): p
Jogador: Pedra - Computador: Pedra
Empate!

Resultado:
   Jogador: 0
   Computador: 0

[P]edra, Pape[L] ou [T]esora? (<enter> para terminar): L
Jogador: Papel - Computador: Pedra
Ganhou!

Resultado:
   Jogador: 1
   Computador: 0

[P]edra, Pape[L] ou [T]esora? (<enter> para terminar):

>>
```