DEPARTAMENTO DE ELETRÓNICA, TELECOMUNICAÇÕES
E INFORMÁTICA

LICENCIATURA EM ENGENHARIA DE COMPUTADORES E INFORMÁTICA

ANO 2025/2026

MÉTODOS PROBABILÍSTICOS PARA ENGENHARIA DE COMPUTADORES E INFORMÁTICA

PRACTICAL GUIDE NO. 1

Task 1- Brief introduction to Matlab

During this semester the scientific programming language Matlab will be used in the lab classes of MPECI. This task aims to introduce the Matlab language by proposing some very simple problems and by providing some external sources.

- **Matlab instalation:**
  https://www.mathworks.com/academia/tah-portal/universidade-de-aveiro-40766421.html
  Use as suas credenciais de Utilizador Universal
- Help:
  https://www.mathworks.com/support/contact_us.html?s_tid=tah_po_helpbutton_ua.pt
  https://www.ua.pt/pt/stic/matlab
- Learn MATLAB basics in about two hours:
  Online couse: MATLAB Onramp (https://matlabacademy.mathworks.com/)

Brief introduction to Matlab based on document "Matlab num instante" (https://sweet.ua.pt/jnvieira/MyDocs/MatlabNumInstante.pdf) created by Professor José Vieira of the University of Aveiro.

Creation of matrices:

The creation of, for example, the following matrix with two rows and three columns

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

can be done with the command

```
>> A= [1 2 3;4 5 6]
```

or, alternatively

```
>> A= [1,2,3;4,5,6]
```

The column and row vectors are special cases of matrices and are created in a similar way. For example, the rows vector v = [1 2 3] is created with

```
>> v= [ 1 2 3]
```

and the corresponding column vector with

```
>> v= [ 1; 2; 3]
```

or transposing the row vector

```
>> v= [1 2 3]'
```

Indexes of matrices:

The element of row *i* and column *j* of matrix *A* is specified as *A(i, j)*. For example, the element of row 1 and column 3 of matrix *A* is specified as *A (1, 3)*. In Matlab, the element *A*(1, 3) of the previous matrix *A* is obtained with

```
>> A(1,3)
```

and the result is

```
ans= 3
```

To change the value of element $A(1, 3)$ to 7, execute

>> *A(1,3)*= 7

The indexes of matrices are lists of positive integer values that can be stored in previously created vectors.

For example, we can extract the second row of matrix *A* either with

>> *v*= *A*(2,[1 2 3])

or creating first a vector with the indexes of the columns

>> *k*= [1 2 3]

>> *v*= A(2, *k*)

Operator ":"

Matlab allows the creation of sequences of numbers in a compact way using the operator ":". The vector

a = [1, 2, 3, ..., 10] can be created with

>> *a*= 1:10

The general notation of the operator ":" is

Initial_number : step : Final_number

and allows the creation of sequences of integer numbers (as in the previous case) or of real numbers.

Some examples are:

>> *e*= 0:pi/20:2*pi

>> *f*= 10:-1:-10

The operator ":" can be used in the creation of vectors of indexes to obtain compact commands to deal with matrices. For example, the odd columns of matrix

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

can be extracted to a matrix *B* with

>> *B*=*A*(1:3,1:2:3)

In this case, all rows (of the odd columns) are extracted. For simplified notation, when all rows are to be extracted, we can use

>> *B*=*A*(:,1:2:3)

On the other hand, if the aim is to extract all columns of the first row, this can be done with

>> *A*(1,:)

Some problems:

(a) Create a row vector with a sequence of even numbers starting with number 4 and ending with the number 100.

(b) Create a row vector with a decreasing sequence of integers starting at 5 and ending at -5.

(c) Create a row vector with a sequence of equally spaced real numbers with 100 elements belonging to the range [0 ... 1].

(d) Create a matrix *B* with random values using command >> *B*= rand(20,30) (20 rows and 30 columns). Extract to matrix *C* the sub-matrix of *B* composed by its rows from 10 to 15 and its columns from 9 to 12.

(e) Create a sequence, $\underline{x}$, starting with the value -π, ending with the value π and with a step of π/15.

(f) Run the command >> plot(x, sin(4*pi*x). What do you obtain?

Task 2

The aim is to characterize the runtime performance of some MATLAB functions of interest. The runtime parameters of interest are the average runtime, the maximum runtime and the 95% percentile runtime.

1.1 Consider the `randperm(n)` MATLAB function that generates a vector with a random permutation of all integer values between `1` and `n`. To check the full syntax of the function, run on the MATLAB command window:

```
>> help randperm
```

1.2 Open a new script file (Ctrl+N). Write on the script file:

```
lista= randperm(20)
```

Run the script file multiple times to visualize the output result of the function.

1.3 To determine the runtime value of one run of the function with `n` = 10000, write on the script file:

```
nLista= 1e4;
t= tic;
lista= randperm(nLista);
tempo= toc(t);
fprintf('Running time= %.2f msec.\n',tempo*1000)
```

Explain each row of the script code. Then, run the script file multiple times. Is the runtime value always the same?

1.4 To estimate the runtime performance parameters, one must first save in a vector the runtime values of multiple runs of the function (in the code below, defined in variable `nIter`) and then, based on the saved values, estimate the performance parameters. To this aim, write on the script file:

```
nIter= 1e4;
nLista= 1e4;
tempos= zeros(1,nIter);
for i= 1:nIter
    t= tic;
    lista= randperm(nLista);
    tempos(i)= toc(t);
end
avTime= mean(tempos);
```

```
maxTime= max(tempos);
temposOrdenados= sort(tempos);
perTime= temposOrdenados(round(0.95*nIter));
fprintf('Average time   = %.2f msec\n',avTime*1000);
fprintf('95%% perc. time = %.2f msec\n',perTime*1000);
fprintf('Maximum time   = %.2f msec\n',maxTime*1000);
```

Explain each row of the script code. Run the script file 4 times and compare the different values of the 3 performance parameters. Check that the number of function runs is enough to correctly estimate the average and the 95% percentile runtimes. Register these values.

1.5 Repeat experiment **1.4** but now with `n` = 100000. Check that again the number of runs is enough to correctly estimate the average and the 95% percentile runtimes. Register these values, compare them with the ones obtained in **1.4** and take conclusions.

1.6 Consider now the `sort(lista)` MATLAB function that takes as input a vector `lista` and outputs a vector with the values of `lista` ordered in ascending order. To check the full syntax of the function, run on the MATLAB command window:

```
>> help sort
```

1.7 Open a new script file (Ctrl+N). Write on the script file:

```
lista= randperm(20)
listaOrdered= sort(lista)
```

Run the script file multiple times to visualize the output result of the 2 functions.

1.8 Write a code in the script file (adapting the one provided in **1.4**) to estimate the average runtime, the maximum runtime and the 95% percentile runtime of the `sort` function.

1.9 Run the script file 4 times considering `n` = 10000 and `n` = 100000 and compare the different performance parameter values. Again, check that the number of runs is enough to correctly estimate the average and the 95% percentile running times for the two values of `n`. Register these values.

1.10 Compare the runtime performance values of the `randperm` function (obtained in **1.4** and **1.5**) and of the `sort` function (obtained in **1.9**). Which of the two functions runs more efficiently?