Lab session 1:
Introduction to MATLAB® and sinusoids

Read this document **and prepare the required exercises before** the lab session in order to get out the most of the session.

1. Introduction to MATLAB®

This first part of this lab session aims to practice some relevant concepts of MATLAB's programing environment[1]. These concepts will show the basics tools to the student in order to correctly follow the laboratory sessions of the Signals and Systems Theory subject.

1.1. MATLAB's help

The MATLAB environment has a help documentation that can be useful to recall function syntaxes and operational modes. It can be prompted by different ways:

1. Choose *Help* > *Product Help* in the superior menu bar.
2. Click on help button [?] in the tools bar.
3. Push *F1* button.
4. Execute one of the following commands in the command line: `helpdesk`, `helpbrowser`, `doc`.

Once the help's browser has been prompted, you can access the different sections or just search for a particular function in the searching box. If you desire to prompt the help document of a particular function, you can:

- Execute `doc 'function'` in the command line (where `'function'` stands for the function name you are looking for). This way will open the help browser with the particular help document of the function you are asking for.
- Or, execute `help 'function'` (where `'function'` stands for the function name you are looking for). This way will show you the help document of the desired function in the command line box.

[1] Although this lab session is focused on the MATLAB's programming environment, Octave is a similar, open source software (see: http://www.gnu.org/software/octave/) that can be used for the same purpose.

If you do not recall the name of a function, the command `lookfor` can be helpful. Also, you must remember that when you want to end the execution of a code, you shall use the key combination *Ctrl-c.*

1.2. Interactive environment

In this first part, we will perform basic operations over vectors using the command line window:

a) Create the vectors:

```
>> A = [1 3 7 4 4 9 0]
>> a = [9 7 3 0 5 1 6]
```

and then generate the following transformations. Try to figure out what are the differences and the meaning of the operators "`;`", "`,`" and "`'`":

```
>> b = [a A]
>> c = [a; A]
>> d = [A , a]'
>> e = [a' A']
```

b) Generate a vector "`z`" with 50 zeros and a vector "`u`" with 25 ones. (try not to write down these vectors element by element, use the help).

c) MATLAB has a very interesting operator for creating vectors, which is ":". The basic syntax is "`start:step:end`". This will create a vector from number "`start`" to number "`end`" in "`step`" increments (or decrements). Try to use it to create the following vectors:

```
>> f = [1 2 3 … 30]
>> g = [22 20 18 … 2]
>> h = [0 0.1 0.2 … 1]
```

- **For PRO's:** Do these without the use of operator ":" (use the help).

Now, we will move forward to matrix and vector operations. First generate the vectors:

```
>> v1 = [1 2 3 5 6 7 8 9 0]
>> v2 = [11 12 13 14 15 17 18 20 22]
```

d) Select the first element of each vector using `v1(1)` or `v2(1)`.

e) Think about what will happen if we use the following commands (Try to figure out **before** running them): `v1(4)` and `v2(10)`.

f) Generate a vector `v3` as the 4 first elements of vector `v2`.

g) Generate a vector `v4` as the five last elements of vector `v1`.

h) Generate a vector `v6`, the same as `V2` but substituting the 4th element by 1.4.

i) Generate a vector `v7` as the vector `v1` without its fifth element.

j) Generate a vector `v8` as the odd elements **(not the values!)** of `v2`.

NOTE: All this operations must be done in 1 or 2 lines of code (preferably one) and without the usage of iterative programming (i.e. `for`, `while`, etc).

HINT: If the command window becomes too messy, you can use the commands "`clc`" that clears the entire command window.

k) Execute the following sentences, understand them and, in case of error, think about what is happening:

```
5 - v1          v1 * v2          v1 == v2          (v1 > 2) & (v2 < 15)
v1 * 5          v1.^2            v1 > 6            (v1 < 6) | (v2 <= 15)
v2 + v1         v1.^v2           v1 > v2                 any(v1)
v1 .* v2        v1^v2       v1 - (v2 > 15)          all(v2 > 12)
```

1.3. Scripts and functions

a) Create a new script called "`script1`" from MATLAB's menu *File* > *New* > *Script*, or by executing `edit script1`. Copy the following lines and save it in the default directory:

```
x = 1:5;
y = 3:7;
z = x + y;
```

b) Now, we will create a function that performs the same task; save it as *function1.m* (execute `edit function1`, copy the next lines and save it):

```
function z = function1(x,y)
z = x+y;
```

c) Then, in the command window, execute the following lines **one-by-one**, and in case of error, think about what is happening:

```
>> script1
>> x
>> y
>> z
>> z = script1
>> clear all % Deletes all variables
>> x = 2:6;
>> y = 5:9;
>> function1
>> z = function1(x,y)
>> g
>> a = 1:10;
>> b = 2:11;
>> function1(a,b)
```

d) Create a new function called "`cosgen`" and copy the following code lines:

```
function xx = cosgen(f,fs,dur)
%COSGEN Function to generate a cosine wave
% usage:
% xx = cosgen(f,fs,dur)
% f = desired frequency of the cosine
% fs = sampling frequency (in Hertz)
% dur = duration of the waveform (in Seconds)
%
xx = zeros(1,fs*dur); % Initialization of variable xx
for i=1:fs*dur
     xx(i) = cos(2*pi*f*i/fs);
end
stem(0:1/fs:dur-(1/fs),xx) % Plot the signal
```

This function generates a cosine. Execute it with f=20, fs = 1000 and dur = 0.05; then observe the results.

e) Modify the function "`cosgen`" using indexation in just 1/2 lines of code that substitute the first line and the "`for`" loop.

HINT: Apply what you learnt in exercises 1.2.d–j).

f) Generate the same cosine as in d) using anonymous functions in MATLAB.

HINT: Define the cosine function using an input variable. For instance, the square root is sqr = @(x) x.^2

g) Execute the following lines of code and explain its behavioral:

```
A = randn(6,3);

A = A .* (A>0);
```

Then, understand what task is performed by the following function:

```
function Z = replacez(A)
%REPLACEZ Does something ☺
% usage:
% Z = replacez(A)
% A = input matrix
%
[M,N] = size(A);
for i=1:M
      for j=1:N
            if A(i,j) < 0
                  Z(i,j) = 77;
            else
                  Z(i,j) = A(i,j);
            end
      end
end
```

h) **Re-write** this function without using the "`for`" loops. Use the idea shown above and also consult the section on vector logicals in the help documentation.

i) Vectorization and indexing are very efficient programming techniques that can substitute in many situations the use of "`for`" and "`while`" loops.

Use the commands "`tic`" and "`toc`" before and after (respectively) a certain line of code (for instance, before/after calling the original "`cosgen`" and your modified version **from exercise e)**) in order to see the amount of time it takes to be computed.

2. Sinusoids

In this second part of the laboratory session, we will explore some basic concepts of sinusoids and operations with them. We will generate two 3000 Hertz sinusoids with different amplitudes and phases:

$$x_1(t) = A_1 \cos(2\pi(3000)t + \emptyset_1)$$

$$x_2(t) = A_2 \cos(2\pi(3000)t + \emptyset_2)$$

HINT: For some tasks of this section, you can re-use the codes implemented in section 1.

a) Let the value of the amplitudes be as follows: let $A_1 = A_2 = 10$. For the phases, let $\emptyset_1 = 45°$, and $\emptyset_2 = 90°$. Phases should be in radians.

   Plot both signals over **a range of t that will exhibit approximately 3 cycles**. Make sure the plot includes $t = 0$, and make sure that you have at least 40 samples per period of the sinusoid.

b) Verify that the phase of the two signals $x_1(t)$ and $x_2(t)$ is correct at $t = 0$. Also verify that each one has the correct maximum amplitude.

c) Use `subplot(3,1,1)` and `subplot(3,1,2)` to make a three-panel subplot that puts both of these plots on the same window. **See `help subplot`**.

d) Create a third sinusoid as:

$$x_3(t) = x_1(t) + x_2(t).$$

   Make a plot of $x_3(t)$ over the same range of time as used in the previous two plots. Include this as the third panel of the plot by using the command `subplot(3,1,3)`.

e) Check the magnitude and phase of $x_3(t)$ and **compare them with the theoretical values.**