Systems and Circuits

Lab 2: Systems

1. Objectives

In this second laboratory session, we will learn how to

- Implement discrete-time systems as functions in Matlab
- Implement a discrete-time convolution in Matlab
- Simulate an scenario of acoustic echo using LTI systems.
- Study the linearity and time-invariance properties of a system employing Matlab.

2. Evaluation

- Students must complete the exercises proposed during the lab session.

3. Project 1: Implement discrete-time systems as functions in Matlab

As we studied in the last lab session, using Matlab we will represent signals using pairs of vectors, one that stores the non-zero values of the signal (the values of $x[n]$) and one that stores the corresponding time-instants at which $x[n]$ takes such values (the values of n). Systems are physical devices or simply functions implemented via software that transform an input signal into an output signal. In Matlab, we will represent a system by a function that takes as input argument the input signal and returns as output argument the output signal. For instance, we are going to create a Matlab function that represents the system defined by the expression

$$y[n] = 0.2x[n]^2 - 0.3x[n-1]^2 + \sin(n)x[n] - 0.4x[n-2]$$

To this end, create a Matlab file called `system.m` and save it in your working folder. The file should contain the following lines to implement the above system:

```
function [ny, y] = system(nx, x)
%
% y[n] = 0.2 x[n]^2 -0.3 x[n-1]^2 + sin(n) x[n] - 0.4 x[n-2]
%
% Inputs:
% nx: range of time instants at which the input signal is non-zero.
% We assume the input signal is zero out of this range.
% x: values of the input signal at the time instants in nx
%
% Outputs:
% ny: range of time instants at which the output signal is non-zero
% y: values of the output signal
%
% Creating the temporal axis
% If x[n] is defined between nx(1) and nx(end), then the last output sample with
% a non-zero value is nx(end)+2, because of the 0.4*x[n-2] term of the equation
% In the same way, the first sample to use from x[n] is nx(1)-2, which is zero
%
% We define the following vectors considering all the above mentioned:
n_init = nx(1) - 2; % First time point at which we need x[n]
n_end = nx(end) + 2 % Las time where y[n] is non-zero
ny = nx(1):1:n_end; % y[n] will be non-zero between nx(1) and n_end
```

```
%
% We define the output signal vector (initialized to 0)
y = zeros(1, length(ny));
%
% The output is obtained by adding 4 signals
% x[n]^2 Term
% First we extend the vector x to have the same length as y
% To this end we simply include zeros for n>nx(end)
%
x1 = [x, zeros(1,n_end - nx(end))];
y1 = 0.2*x1.^2;
%
% -.3x[n-1]^2 term
x2 = [0 x 0]; % The first 0 comes from x[n-1]=0 for n=nx(1) -1
y2 = -.3*x2.^2
%
% sin(n) x[n]
y3 = sin(ny).*x1;
%
% - 0.4 x[n-2] term
x4 = [zeros(1, nx(1) - n_init), x];
y4 = -0.4*x4;
%
% We add the form components
y = y1 + y2 + y3 + y4;
```

Try the performance of this system, which essentially introduces non-linear distortion to the input signal, using the audio signals provided in the lab session 1. Using the above system and what you learnt in the first lab session, program a function `system2.m` that implements the system described by the equation:

$$y[n] = 0.5x[4-n] + 0.7x[n-5] - 0.4\cos(2\pi x[2-n])$$

Keep in mind that:

- The output is the sum of three components.
- Each of these components performs an operation over the independent variable. Keep this in mind to determine the time instants at which the output signal is defined.

4. Project 2: Implementing the discrete-time convolution between finite-length signals

LTI systems are perfectly defined by the impulse response h[n]. Given this signal, the output signal for any arbitrary input signal is given by the convolution operation:

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k].$$

Despite Matlab already provides a function to implement the convolution operation `conv`, in this exercise we are going to program a version that is able to work with signals defined with two vectors (one for the time values and one for the values of the signal). To this end, program in a file called `discreteconvolution.m` a function that implements the convolution between to signals that are finite in time. The structure of the function would be as follows:

```
function [ny, y] = discreteconvolution(nx, x, nh, h)
%
% Inputs:
% nx: range of time instants at which the input signal is non-zero.
% x: values of the input signal at the time instants in nx
% nh: range of time instants at which the impulse response is non-zero.
% h: values of the impulse response
%
```

```
% Outputs
% ny: range of time instants at which the output signal is non-zero
% y: values of the output signal
```

The Matlab function `conv` computes the convolution between two vectors of length N, assuming that the first sample is at $n = 1$ and the last one is at $n = N$ Given the pair of input signals $x[n]$ and $h[n]$, which start at n_x and n_h , we can find a pair of signals $x1[n]$ and $h1[n]$ starting at $n = 1$ such that $x[n] = x1[n - n_x + 1]$ and $h[n] = h1[n - n_h + 1]$. For instance, the signal

$$x[n] = \begin{cases} 1 & ,5 \leq n \leq 12 \\ 0 & \text{, otherwise} \end{cases}$$

can be written as follows

$$x[n] = x1[n] * \delta[n - 4]$$

where

$$x1[n] = \begin{cases} 1 & ,1 \leq n \leq 8 \\ 0 & \text{, otherwise} \end{cases}$$

Hence, the output of the LTI system $y[n] = x[n]h[n]$ can be alternatively computed as follows:

$$\begin{aligned} y[n] = x[n] * h[n] = (x1[n] * \delta[n + 1 - n_x]) * (h1[n] * \delta[n + 1 - n_h]) = \\ = (x1[n] * h1[n]) * (\delta[n + 1 - n_x] * \delta[n + 1 - n_h]) = \\ = y1[n] * \delta[n + 2 - (n_x + n_h)] \end{aligned}$$

Given the above result, the function `discreteconvolution.m` should contain the following steps

1. Transform both sequences into their equivalents, starting at $n = 1$
2. Compute the convolution of the equivalent signals
3. Shift the output signal to get the desired signal

Alternatively, we encourage you to program your own convolution function to be used in step 2. and compare the result using the `conv` function provided by Matlab.

5. Project 3: simulating an audio signal with echo

Given what you have learnt so far, we can simulate an audio signal with an acoustic echo. Assume someone is giving an speech in the countryside, close to a mountain. Someone is listening the speech the signal that indeed is receiving is the sum between the original audio signal and a degraded version of this signal, that has been delayed and modified after being reflected in the mountain. The echo can be seen as an interconnection of systems in parallel. One of the branches consists on an LTI system whose output is equal to the input signal, i.e., an LTI with inputs response $\delta[n]$. The other branch is the one causing the echo. This one can be modeled as a chain of three LTI systems:

- The first system models the delay between the person giving the speech and the mountain. In discrete time, we will model this effect by simply introducing a delay in the audio signal. If the audios that we provided you have been sampled with f_s samples per second and the sound speed is 340 m/s, for every meter travelled by the audio signal we have a delay of $f_s/340$ samples.
- The second system models the degradation introduced into the delayed signal after it hits the face of the mountain. The system output signal is indeed the sum of all the reflected signals. For this system, we will use the impulse response stored in the file `mountain.mat`.
- Finally, we have a third system that models the attenuation suffered by the signal after being reflected in the mountain and in its way back to the listener. The impulse response of this system is simply of the form $A\delta[n]$, where$A < 1$ is the coefficient that models the attenuation.

The project consists on simulating the interconnection of the described systems and use the audio signals provided in the first lab session. Listen to the effect of the echo and try to investigate for what values of attenuation and delay the audio signal is severely degraded.