Lecture 3 - Systems (Before Lecture)

Prelude: a system can be defined as an entity that transforms a signal. Consequently, if we apply a signal to a system as *input*, the system responds to the signal by producing another signal called the *output*[a]. To put it another way, a system performs a transformation to an input signal, yielding an output signal.
Previously, we discovered the utility of employing local functions or subfunctions within a script in Matlab. In the first part of this activity we use subfunctions to program DT systems. For example, consider first the system given by

$$y[n] = \cos(x[n])$$

This system can be achieved by the following function

```
function y=sysEx1(x)
    y=cos(x)
end
```

To compute the output of the system, Matlab first creates an empty vector y that has the same dimensions of vector x. Then, the result of the operation cos is computed for each element in vector x, and the result is placed in the corresponding position within vector y.

In this case, the system is a *memoryless* or *static system* since its output at time n depends exclusively on the input at time n. This is different from the situation of *systems with memory* or *dynamic systems* in which the output at time n depends on the present and past values of the input (i.e., values of the input at time indices $n, n-1, n-2, \ldots$)[b]. As an example, consider

$$y[n] = x[n] - x[n-1]$$

To program the input-output relationship of this system, we may use a for loop to go through all values of n and compute the output in each case, note that the system is time-invariant, so there is no need for a discrete time vector to compute the output.

```
function y=sysEx2(x)
    y=zeros(1,length(x)); % create a vector full of zeros to store the output
    xnm1=0;               % the value of x[-1] is supposed to be zero
    for i=1:1:length(x)   % go through all the vector x[n]
        y(i)=x(i)-xnm1;   % compute y[n] for each n
        xnm1=x(i);       % the new past value is equal to x(i)
    end
end
```

a We use *system* as short hand for single-input single-output (SISO) system: a system that operates on only one input signal to produce a unique output.
b If the system is anticipatory (non-causal), the value of the output at time n depends as well on future values of the input at instants $n+1, n+2, \ldots$

1. **Consider the systems described by the following input-output relationships**

$$\begin{aligned} \text{S1} \quad y[n] &= \cos[0.2\pi n]\, x[n] \\ \text{S2} \quad y[n] &= x[n] + 3x[n-1] \end{aligned} \tag{1}$$

   a) **Develop a subfunction to generate the following test signals. Each subfunction has as input a time index vector n**

$$\begin{aligned} x_1[n] &= ne^{-0.2n}(u[n] - u[n-20]) \\ x_2[n] &= \cos[0.05\pi n](u[n] - u[n-20]) \end{aligned} \tag{2}$$

Hint: if you want to use the `dtstep` function developed for last lecture,. both documents must be in the same directory.

b) **Write a subfunction within a .m file for each system (Hint:** system S1 requires two input arguments, the vector of the input signal values `x` and the time index vector `n`**)**

c) **Compute the outputs of S1 and S2 to the inputs** $x_1[n]$ **and** $x_2[n]$

Intermezzo: for the analysis of dynamic systems, it is useful to employ graphical interfaces to represent and simulate mathematical models representing a physical system. Matlab counts with a graphical interface called Simulink. In Simulink, models are represented graphically as *block diagrams.* A wide array of blocks are available to the user in provided libraries for representing various phenomena and models in a range of formats. Take a look at the videos accompanying this lecture on the Google Classroom to see some examples.

We will use Simulink to evaluate linearity and time-invariance of continuous-time systems.

2. **Read carefully the "During Lecture" activity description**