MATLAB Project of Chapter 2

1. Write a MATLAB program to generate a discrete-time exponential signal. Use this function to plot the exponential $x[n]=(0.9)^n$ over the range n=0, 1, 2, …, 20.

2. Given a differential equation:

$$y[n] - 1.8\cos\left(\frac{\pi}{16}\right)y[n-1] + 0.81y[n-2] = x[n] + \frac{1}{2}x[n-1]$$

generate and plot the impulse response $h[n]$ of the difference equation

(a) using recursion $y[n] = 1.8\cos\left(\frac{\pi}{16}\right)y[n-1] - 0.81y[n-2] + x[n] + \frac{1}{2}x[n-1]$

(b) using the **filter** function.

Plot $h[n]$ in the range of $-10 \le n \le 100$.

Please upload your program and the results to the ftp site within two weeks after the date of project assignment. The ftp site can be found at the course website.

Example

Given a differential equation: `y[n] - 0.9y[n-1] + 0.81y[n-2] = x[n] - x[n-2]`

a. Find the impulse response for `h[n], n=0,1,2` using recursion.

b. Find the impulse response using MATLAB command filter.

Answer:

This is actually quite simple, because the differential equation contains the body of the recursive function almost entirely: y[n] = **0.9y[n-1]** - **0.81y[n-2]** + x[n] - x[n-2]

The parts in bold are actually the recursive calls! What you need to do is to build a function (let's call it `func`) that receives `x` and `n`, and calculates `y[n]`:

```
function y = func(x, n)
   if (n < 0)
      %# Handling for edge case n<0
      return 0
   else if (n == 0)
      %# Handling for edge case n=0
      return x(0)
   else
      %# The recursive loop
      return 0.9 * func(x, n-1) - 0.81 * func(x, n-2) + x(n) - x(n-2)
   end
```

Note that it's *pseudo-code*, so you still have to check the edge cases and deal with the indexation (indices in MATLAB start with 1 and not 0!).

Using filters

The response of a digital filter is actually the y[n] that you're looking for. As you probably know from lesson, the coefficients of that filter would be the coefficients specified in the differential equation. MATLAB has a built-in function `filter` that emulates just that, so if you write:

```
B = [1, 0, 1];        %# Coefficients for x

A = [1, 0.9, -0.81];  %# Coefficients for y

y = filter(B, A, x);
```

You'd get an output vector which holds all the values of y[n].