ECE 3793
Matlab Project 2

Spring 2017 Dr. Havlicek

DUE: 04/7/2017, 11:59 PM

What to Turn In:

Make one file that contains your solution for this assignment. It can be an MS WORD file or a PDF file. Make sure to include your name in your turn-in file and add a title at the top of the first page that says "ECE 3793" and "Matlab Project 2." Number the problems and paste in your Matlab code and the resulting command window output. Paste in the figures and graphs and make sure to include answers for the discussion questions.

Submit your solution electronically on Canvas by uploading your turn-in file to

`ECE-3793-001 > Assignments > Matlab 02`

If you are using the Virtual Labs, make sure to save all your files before you log out!

The Assignment:

1. For two **finite length** discrete-time signals $x[n]$ and $h[n]$, you can compute the convolution $y[n] = x[n] * h[n]$ using the Matlab `conv` command.

   The resulting Matlab array that holds the values of $y[n]$ will always have a length equal to the length of $x[n]$ plus the length of $h[n]$ minus one.

   Note: a finite length signal that starts at $n = n_1$ and ends at $n = n_2$ has a length of $n_2 - n_1 + 1$. For example, if the signal starts at $n = 1$ and ends at $n = 4$, then it has a length of $4 - 1 + 1 = 4$.

   Here is an example convolution problem: suppose that

   $$x[n] = \delta[n+1] - 2\delta[n] + 3\delta[n-1] - 4\delta[n-2] + 2\delta[n-4] + \delta[n-5]$$

   and

   $$h[n] = 3\delta[n] + 2\delta[n-1] + \delta[n-2] - 2\delta[n-3] + \delta[n-4] - 4\delta[n-6] + 3\delta[n-8].$$

   Let's use Matlab to compute and plot the convolution $y[n] = x[n] * h[n]$.

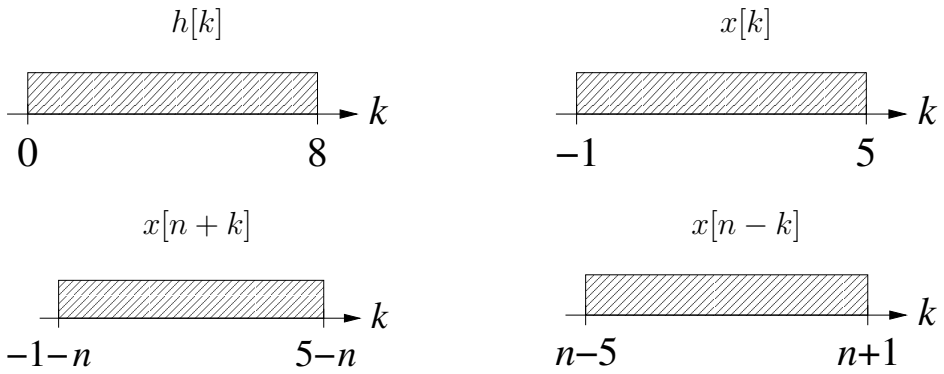
   Note that the signal $x[n]$ starts at $n = -1$ and ends at $n = 5$. So the length of $x[n]$ is $5 - (-1) + 1 = 7$. Similarly, $h[n]$ starts at $n = 0$ and ends at $n = 8$. So the length of $h[n]$ is $8 - 0 + 1 = 9$. Therefore, `conv` will return the convolution result in an array of length $7 + 9 - 1 = 15$ (remember, the result returned by `conv` always has a length equal to the length of $x[n]$ plus the length of $h[n]$ minus one).

Now, the index of a Matlab array always starts at 1. So the Matlab variable $x(1)$ will be equal to $x[-1]$, the value of the input signal at $n = -1$. This means that we will have to keep track of the time variable n for each signal ourselves... Matlab will not do this for us.

Let's think of the convolution this way:

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

The figures below show the starting and stopping values of k for the signals $h[k]$, $x[k]$, $x[n+k]$, and $x[n-k]$:



From these figures, you can see that the first time $y[n]$ can be nonzero is when $n+1 = 0$, or $n = -1$. The last time that $y[n]$ can be nonzero is when $n - 5 = 8$, or $n = 13$. So $y[n]$ starts at $n = -1$, ends at $n = 13$, and has a length of $13 - (-1) + 1 = 15$.

What we have figured out so far is that:

- $h[n]$ starts at $n = 0$, ends at $n = 8$, and has length 9;
- $x[n]$ starts at $n = -1$, ends at $n = 5$, and has length 7;
- $y[n]$ starts at $n = -1$, ends at $n = 13$, and has length 15.

Remember: when you use the Matlab `conv` function, you have to keep track of these things yourself.

Now consider the following Matlab code which computes the convolution and plots it:

```
%-------------------------------------------------------
% P1a
%
% compute and plot a discrete convolution
%
h = [3 2 1 -2 1 0 -4 0 3];  % Impulse response
```

```
x = [1 -2 3 -4 0 2 1];      % Input signal
y = conv(x,h);              % y[n] = x[n] * h[n]
subplot(3,1,1);             % 3x1 array of graphs
stem([-1:5],x);             % plot x[n]
title('x[n]');
xlabel('n');
ylabel('x[n]');
subplot(3,1,2);             % make 2nd graph active
stem([0:8],h);              % plot h[n]
title('h[n]');
xlabel('n');
ylabel('h[n]');
subplot(3,1,3);             % make 3rd graph active
stem([-1:13],y);            % plot y[n]
title('y[n]');
xlabel('n');
ylabel('y[n]');
```

(a) Type in this code and run it. You can type it in line-by-line at the command prompt or you can create an *m*-file.

(b) Modify this code to compute and plot the convolution of $h[n]$ with the new input signal

$$x[n] = 3\delta[n-3] + \delta[n-4] - 4\delta[n-6] + \delta[n-7] - 5\delta[n-8].$$

Make sure to briefly explain your calculations for the starting and stopping times of $y[n]$.

(c) Use Matlab to compute and plot the convolution $y[n] = x[n] * h[n]$ where

$$x[n] = \left(\frac{1}{4}\right)^n (u[n] - u[n-6])$$

and

$$h[n] = \left(\frac{1}{2}\right)^n (u[n+3] - u[n-3]).$$

Explain your calculations for the starting and stopping times of $y[n]$.

2. Now we're going to use Matlab to help us work problem 4.26(a)(i) from the course text (note that this problem was assigned as part of Homework 6). We are given an LTI system with impulse response $h(t) = e^{-4t}u(t)$ and input $x(t) = te^{-2t}u(t)$. We are asked to find the system output $y(t)$.

From Table 4.2, we can write down immediately that

$$X(\omega) = \frac{1}{(2 + j\omega)^2}$$

and

$$H(\omega) = \frac{1}{4 + j\omega}.$$

So the Fourier transform of $y(t)$ is given by

$$\begin{aligned} Y(\omega) = X(\omega)H(\omega) &= \frac{1}{(4 + j\omega)(2 + j\omega)^2} \\ &= \frac{1}{(j\omega)^3 + 8(j\omega)^2 + 20j\omega + 16}. \end{aligned} \tag{1}$$

To invert $Y(\omega)$ and find $y(t)$, we need to compute the partial fraction expansion

$$Y(\omega) = \frac{A}{4 + j\omega} + \frac{B}{2 + j\omega} + \frac{C}{(2 + j\omega)^2}. \tag{2}$$

This is where Matlab can help. By considering the numerator and denominator of $Y(\omega)$ in (1) to be polynomials in $j\omega$, we can use the Matlab `residue` function to compute the partial fraction expansion as follows:

```
numer = [1];
denom = [1 8 20 16];
[r p k] = residue(numer,denom)

r =

    0.2500
   -0.2500
    0.5000


p =

   -4.0000
   -2.0000
   -2.0000


k =

     []
```

In the Matlab output, vector `r` gives the residues in the partial fraction expansion (these are the numbers A, B, and C in (2)). Vector `p` gives the poles.

In cases where $Y(\omega)$ is an *improper* fraction, there will also be *direct transmission* terms in the partial fraction expansion that will be given in the output vector `k`. But since our $Y(\omega)$ is a proper fraction in this problem, we see that the output vector `k` is returned as the null vector because we don't have any direct transmission terms.

Thus, from the output of the Matlab `residue` function, we can write down the partial fraction expansion for $Y(\omega)$:

$$Y(\omega) = \frac{\frac{1}{4}}{j\omega + 4} - \frac{\frac{1}{4}}{j\omega + 2} + \frac{\frac{1}{2}}{(j\omega + 2)^2}$$

(note that this agrees with the solution to Homework 6). From Table 4.2, it is then easy to write down the final expression for $y(t)$:

$$y(t) = \left[\frac{1}{4}e^{-4t} - \frac{1}{4}e^{-2t} + \frac{1}{2}te^{-2t}\right] u(t).$$

(a) Use the Matlab `residue` function to work text problem 4.33 (all three parts). Note: there *is* a direct transmission term in part (c). For that part, you should get that

$$H(\omega) = 2 + \frac{-\sqrt{2} + j\sqrt{2}}{j\omega + \left(\frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2}\right)} + \frac{-\sqrt{2} - j\sqrt{2}}{j\omega + \left(\frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2}\right)}.$$

(b) Use the Matlab `residue` function to work text problem 4.34(b).

You can also use the Matlab Symbolic Math Toolbox to compute Fourier transforms. To do this, you need to declare your Matlab variables to be *symbolic variables* as shown in the examples below. In the Symbolic Math Toolbox, the unit step function $u(t)$ is called `heaviside(t)` and the Dirac delta $\delta(t)$ is called `dirac(t)`.

The Symbolic Math Toolbox also provides functions `simplify` and `pretty` that will often make your answers look better.

The following Matlab session log gives an example where we set `xt` equal to the signal $e^{-2t}u(t) + 3\delta(t)$, set `Xw` equal to the Fourier transform of this signal, and finally set `xt2` equal to the inverse Fourier transform of `Xw` (which is the same as the original `xt`, of course).

```
syms w t
xt = exp(-2*t)*heaviside(t) + 3*dirac(t)

xt =

3*dirac(t) + exp(-2*t)*heaviside(t)
```

```
Xw = fourier(xt,t,w)

Xw =

1/(w*i + 2) + 3

pretty(Xw)

     1
  ------- + 3
  2 + w i

xt2 = ifourier(Xw,w,t)

xt2 =

(6*pi*dirac(t) + 2*pi*exp(-2*t)*heaviside(t))/(2*pi)

xt2 = simplify(xt2)

xt2 =

3*dirac(t) + exp(-2*t)*heaviside(t)
```

Here is another example that shows how to work text problem 4.3(b) using symbolic math:

```
syms t w
xt = 1 + cos(6*pi*t + pi/8)

xt =

cos(pi/8 + 6*pi*t) + 1

Xw = fourier(xt,t,w)

Xw =

pi*(dirac(w - 6*pi)*((2^(1/2) + 2)^(1/2)/2 + ((2 - 2^(1/2))^(1/2)*i)/2)
  + dirac(6*pi + w)*((2^(1/2) + 2)^(1/2)/2 - ((2 - 2^(1/2))^(1/2)*i)/2))
  + 2*pi*dirac(w)
```

3. Now it's your turn: use the Matlab Symbolic Math Toolbox to work text problem 4.3(a).

4. To specify a shifted step function like $u(t-2)$ in Matlab, you simply type `heaviside(t-2)`. The same syntax applies for a shifted Dirac delta like $\delta(t-2)$. Use the Symbolic Math toolbox to work text problem 4.1(a).

5. Use Symbolic Math to work text problem 4.26(a)(iii). Note: if you have $X(\omega)$ in a symbolic variable `Xw` and $H(\omega)$ in a symbolic variable `Hw`, then you can compute $Y(\omega)$ using the Matlab syntax

```
Yw = Xw * Hw
```