Lab 3
Convolution

EE 235: Continuous-Time Linear Systems
Department of Electrical Engineering
University of Washington

1 Introduction

In this lab you will review the concept of convolution, and learn how to convolve signals in MATLAB.

Because you are working on a computer, you are necessarily working with finite-length, discrete-time versions of continuous-time signals. It is important to note that convolution in continuous-time systems cannot be exactly replicated in a discrete-time system. However, you can still explore the basic effects of convolution and gain some insight by using the MATLAB function `conv`.

You will learn more about discrete-time convolution and discrete-time methods in MATLAB when you take EE 341. One visual difference is that discrete-time signals are often plotted with the `stem` command. You may want to look at some of your plots – especially step functions– with `stem` for debugging purposes. However, to preserve the course's focus on continuous-time signals, always use the `plot` command when you submit your work or demo it to the lab instructor.

One artifact of using `plot` over `stem` is that discontinuities, such as the transition in a step function, do not appear instantaneous: you may see a slope or ramp in the plot instead of the vertical jump that you expect. Bear this in mind when explaining your output diagrams. You will also need to represent impulses in such a way that the height in the discrete-time representation is the same as the area in the continuous-time version.

Finally, there is one other difference regarding how you use discrete-time (DT) signals. When you plot or play a continuous-time (CT) signal, as you did in Lab 2, you specify the sampling frequency F_s. This parameter of the CT signal is used to represent the signal inside the (digital) computer. However, DT signals have as a parameter the *time increment*

[1] Last revision: Wed Apr 28 21:59:01 EDT 2010
[2] http://www.creativecommons.org/licenses/by/2.0

T_s between consecutive digital samples. Thus to plot a DT signal, you need to will define a time vector:

```
t = [0:Ts:end];
```

Recall that you found the duration of a CT signal by dividing the number of samples by the sampling rate. For a DT signal, the duration (that is, the correct value for `end`) is actually:

```
end = (length(sound) -1) * Ts;
```

To play a DT sound, you still need to provide Fs, which you calculate from $Fs = 1/T_s$.

Exercise 1:
• Explain why the above formula for `end` is correct.

2 Useful MATLAB Commands

In addition to what you already know about MATLAB, you may need to use the following commands. Use `help` to learn their correct syntax.

- `whos` – Lists all variables and their dimensions.
- `clear` – Clears (deletes) all existing variables.
- `zeros` – Creates a vector (or matrix) of all zeros, as in Lab 2.
- `ones` – Like `zeros`, but with 1 instead of 0.
- `conv` – Convolves two signals.
- `soundsc` – Normalizes an audio signal to have range within [-1,1] and plays it. You must specify the sampling rate F_s. Similar to what you implemented in Lab 2.
- `pause` – pauses the execution of a MATLAB script until any key is pressed.
- `find` – returns the indices of particular elements of a vector that satisfy some condition. In particular, `find(X == 0)` returns the index of all elements of `X` that are zero. You can use any elementwise comparison inside `find`.

3 Convolution

MATLAB has a function called `conv(x,h)` that you can use to convolve two signals `x(n)` and `h(n)`. It assumes that the time increment is the same for both signals. The input signals are finite-length, so the result of the convolution should have a length equal to the sum of the lengths of the inputs– which turns out to be `length(x) + length(h) - 1`.

As you have learned in class, a linear time-invariant (LTI) system is completely described by its impulse response. When working in MATLAB, this impulse response must also be discrete. Work through the following example:

Exercise 2:
• Consider a system with impulse response `h`:

```
h = [1 zeros(1,20) 0.5 zeros(1,10)];
```

• Suppose the input `x` to the system is

```
x = [0 1:10 5.*ones(1,5) zeros(1,40)];
```

• Convolve `x` and `h` as follows:

```
y = conv(x,h);
```

• Use `subplot` to show the impulse response, input, and output of the convolution. In order to make the plots have identical timing– and thus line up neatly– you will need to either add zeros to the end of `x` and `h` to make them the same length as `y`, or define a specific time vector for each signal.

Every non-zero coefficient of the impulse response `h` creates an echo of the original input. When you convolve the input `x` and the impulse response `h`, you are adding up all the scaled and time-shifted echoes.

Exercise 3:
• In the system you just made, change the second coefficient of the impulse response to be negative.
• How does this change the system output? Redo the plots.

3.1 Convolution and Echo

Much like you did in Lab 2, download the jazz trumpet lick `fall` from the course website[3], load it into MATLAB using `load('fall')`, and plot it against time. Make a single MATLAB script that executes all of the following exercise; use the `pause` to prevent the script from playing all the sounds at once.

Exercise 4:
• Load `fall` and set the sampling frequency `Fs` to be 8000Hz.
• Create the following impulse response:

```
h = [1 zeros(1,10000) 0.25 zeros(1,1000)];
```

[3] http://cnx.org/content/m14109/latest/fall.mat

• Convolve the trumpet lick with `h`:

```
y = conv(fall, h);
plot(y)
soundsc(y,Fs)
```

• Make a second impulse response `h2` that is the same as `h` except that the echo has a negative coefficient. This corresponds to a change in phase.
• What's the difference in how the new output `y2` sounds using `h2` versus `h`?
• Build a third system that delays the echo for a half second, by inserting `Fs/2` zeros before the second impulse:

```
h3 = [1 zeros(1,round(Fs/2)) 0.25 zeros(1,10000)];
```

• Pass `fall` through the new system to get a third output `y3`, then plot and play the result. How do the input and output signals compare?
• There are several built-in MATLAB sounds, such as `chirp, gong, handel, laughter, splat` and `train`. These sounds are built-in `*.mat` files that define the sound to appear in a vector named `y`, and also define the sample rate for that sound to be in `Fs`. To use these sounds, you will need to do the following:

```
>> load('soundName');
soundVariableName = y;
soundVariableFs = Fs;
```

Pick two of those sounds, and do the following for *each* of them:
`load` the sound, generate a new system response that echos and/or delays the input, and pass the sound through the system.
Play both the input and output, and plot the input, impulse response, and the output.

3.2 Convolution and Smoothing

Exercise 5:
• Build an impulse response in the shape of a rectangular pulse:

```
hpulse = [ones(1,50)/50 zeros(1,20)];
```

• Create a new signal `ypulse` by convolving `hpulse` with `fall`.
• Plot the input, system response, and output.
How does the output sound different from the input?
• Visually, the input signal `fall` looks like it's centered around value 0, and the output `ypulse` looks like it is more positive. Find the average value of the output signal, using:

```
avgYPulse = sum(ypulse)./length(ypulse);
```

Now find the average value of the input. How do they compare? Can you explain?

Let's look at the input and output signals more closely in order to better see what the system is doing to the input:

Exercise 6:
• Zoom in on a piece of the signals:

```
subplot(2,1,1), plot(6400:6500, fall(6400:6500))
subplot(2,1,2), plot(6400:6500, ypulse(6400:6500))
```

• How do the plots differ? Can you explain?

The impulse response `hpulse` applies a low-pass filter to the signal. You will work more with filters later, but the basic idea here is that the original signal is made up of many different frequencies. The system `hpulse` is letting the lower frequencies pass through, but the higher frequencies are being attenuated. This affects both how the signal looks when it is plotted, and how it sounds.

3.3 Convolution and the Box Function

Rectangular pulses (or 'box functions') are very useful when building system responses. You defined a box function in the previous section, but it would be good to have a function that could build arbitrary box functions without needing to count how many ones and zeros to use each time.

Remember to structure your function files correctly, as in Lab 2: Include the function definition, the `help` comment specifying the syntax, comment your code, and with the footer with your name and information.

Exercise 7:
• Create a new function called `unitstep` to produce the unit step function $u(t)$:
$u(t) = 0$ for $t < 0$, and $u(t) = 1$ for $t \geq 0$.
The function should take two parameters:
a time vector `Time` that specifies the finite range of t for the whole signal,
and a time-shift value `tshift` that indicates where the unit step function changes value.
`Time` can be any range, such as `0:25`, `-5:0.1:5`, or `linspace(-2,8,1000)`.
The output should have the same dimensions as `Time`.

`tshift` can be any scalar value (positive or negative).
`unitstep(Time, 0)` should produce the unit step function as defined above.
`unitstep(Time, 4)` should be equivalent to the delayed unit step $u(t-4)$,
`unitstep(Time, -4)` should produce $u(t+4)$.
• Demonstrate that your `unitstep` function works by creating unit step functions over different time vectors, and plotting the results. Show at least one positive, negative, and zero shift.

Exercise 8:
• Write a function called `boxt` that creates a rectangular pulse. It should take three parameters: a start time `t1`, an end time `t2`, and a time vector `Time`.
`t1` and `t2` are scalar values.
`Time` is defined identically as for `unitstep`.
The output should have the same dimensions as `Time`.
`boxt(t1, t2, Time)` is a box function that is 0 for all $t < t1$ and $t \geq t2$, and 1 for $t1 \leq t < t2$.
To simplify the calculation, use your `unitstep` function to make your `boxt` function. How can you describe a box function in terms of two unit steps?
• Demonstrate that your `unitstep` function works by creating box functions over different time vectors, and plotting the results.

Convolution affects the duration of the output signal. Suppose we have two signals: s_1 is non-zero between t_1 and t_2, and s_2 is non-zero between t_3 and t_4. Then their convolution $s_1 * s_2$ is non-zero between $(t_1 + t_3)$ and $(t_2 + t_4)$. However, convolution also affects the duration of the time vector over which the signals are defined!
If signals `u` and `v` are defined over time vectors with identical increments, such as:

```
TimeU = t1:ts:t2;
TimeV = t3:ts:t4;
```

Then `conv(u,v)` will be defined over `TimeConv = (t1+t3):ts:(t2+t4)`.

Exercise 9:
• Use your `boxt` function to generate two box signals over the time span [-5,10]: One box should be nonzero over [0,4], and the other one should be nonzero over [-1,1].
Pick an appropriately small time increment.
• What should their convolution look like, and over what range is it non-zero?
• Verify your answer by convolving the two box functions.

- Plot the two inputs and the output using appropriate time vectors.
- Are the plots accurate?

Exercise 10:

- The built-in convolution function assumes that the step sizes are integers, so it thinks that the true length (in time) of a box signal is the number of vector elements used to construct it. However, the correct duration is really just `length` $\times$ `ts`. This discrepancy affects the area computation in convolution. Because the length of the output is fixed, this means that the height of the output is off.
- Repeat the previous exercise, but scale the output correctly.
- Demonstrate that the output is now correct.

EOF