Convolution

Version 1.6: Oct 12, 2008 11:03 am GMT-5

UW EE235 TA UW EE235 TA

Abstract

In this lab, we will explore convolution and how it can be used with signals such as audio.

1 Introduction

In this lab, we will explore convolution and how it can be used with signals such as audio.

Since we are working on a computer, we are working with finite-length, discrete-time versions of signals. It is important to note that convolution in continuous-time systems cannot be exactly replicated in a discrete-time system, but using MATLAB's `conv` function for convolution, we can explore the basic effects and gain insight into what is going on. (You can learn more about discrete-time convolution in the UW EE 341 class.)

When you are explicitly working with discrete-time signals, you would plot them with `stem`. However, since we want to think of these as continuous time, we'll still use the `plot` command. An artifact that you may notice is that discontinuities (as in a step function) are not instantaneous – they have a small slope in the plot. In addition, you need to represent impulses with the height in discrete time equal to the area in continuous time.

When you want to play or plot the discrete-time signal, you need to specify the time increment Ts between samples. As you found in the previous lab, when playing a sound you specify Fs=1/Ts. (Fs is set for you when you load a sound.) When plotting, you need to define a time vector, e.g. t=[0:Ts:end] where end=(length-1)*Ts.

2 Some Useful MATLAB Commands

- `whos`, list all variables and their sizes.
- `clear`, clears all variables.
- `zeros`, creates a vector (or matrix) of zeros.
- `ones`, creates a vector (or matrix) of ones.
- `conv`, convolves two signals.
- `soundsc`, plays an audio signal, normalizing if the values are greater than +/-1. Requires the sampling rate.

[*]http://creativecommons.org/licenses/by/2.0/

3 Convolution

MATLAB has a function called `conv(x,h)` that you can use to convolve two discrete-time functions `x(n)` and `h(n)`. It assumes that the time steps are the same in both cases. The input signals must be finite length, and the result of the convolution has a length that is the sum of the lengths of the two signals you are convolving (actually L1+L2-1).

1. Recall that a linear time-invariant system is completely described by its impulse function. In MATLAB, the impulse response must be discrete. For example, consider the system with impulse response

   ```
   h = [1 zeros(1,20) .5 zeros(1,10)];
   ```

   Plot the impulse response using the `plot` command.
2. Consider an input to the system,

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

   Plot the input with the `plot` command.
3. Use the command `conv` to convolve `x` and `h` like this,

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

   Use subplot to show the impulse response, input, and output of the convolution. Note that you need to add zeros to the end of x and h (to make them the same length as y) or define a time vector for each signal in order to make the timing comparable in the different subplots.
4. Every non-zero coefficient of the impulse response `h`, acts as an echo. When you convolve the input `x` and impulse response `h`, you add up all the time-shifted and scaled echoes. Try making the second coefficient negative. How does this change the final result?

Exercise 1 Convolution and Echo

1. Create a new script for this problem. Download the trumpet jazz lick "fall" here[1] , and then load it into MATLAB using `load('fall')` and plot it. Use `whos` to see that the variables `fall` and `Fs` are created for you. (The sampling rate (Fs) for this signal should be 8000 Hz.)
2. Use the following commands to convolve the following impulse response `h`, with the trumpet sound.

```
    Fs = 8000     % for this example
h = [1 zeros(1,10000) .25 zeros(1,1000)];
y = conv(fall, h);
plot(y)
soundsc(y, Fs)
```

3. What if the second echo (in `h`) is a negative coefficient? When you play it, it should not sound different since your ear is not sensitive to that sort of modification (simple phase change).
4. Now let's build a system that delays the echo for a quarter second by inserting `Fs/4` zeros before the second impulse:

```
h = [1 zeros(1, round(Fs/4)) 0.25 zeros(1,1000)];
```

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

Pass the `fall` input signal through the system to get the output `y`:

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

How do the input and output signals compare in the above step? (Look and listen). Experiment with different numbers of zeros, and try repeating this with some of the built-in MATLAB sounds.

NOTE: Some built-in sounds in MATLAB are `chirp`, `gong`, `handel`, `laughter`, `splat`, and `train`. Load them with the `load` command and the sound data will be loaded into the variable `y` and the sampling rate in `Fs`.

5. Show the TA your script file. You should be able to run it and have it generate any plots and sounds.

   NOTE: You can use the `pause` command to pause MATLAB until a key is pressed to prevent it from playing all your sounds at once.

Exercise 2 Convolution and Smoothing

1. Build a box impulse response:

```
h2=[ones(1,50)/50 zeros(1,20)];
```

   Create a new signal `y2` by convolving "fall" with `h2`
2. How does the output sound different from the input signal?
3. Visually, a difference is that the input signal `fall` looks like it's centered around value 0, and the system output `y2` looks like it's more positive. Let's look more closely. Find the average value of the signal `fall` (use `sum(fall)/length(fall)`), and you should see that in fact the `fall` signal isn't really centered around 0.
4. Next, to see what this system does to the input signal, zoom in on part of the signal:

```
    subplot(2,1,1), plot(6400:6500, fall(6400:6500))
subplot(2,1,2), plot(6400:6500, y2(6400:6500))
```

   The convolved signal should look a little smoother to you. This is because this impulse response applies a low-pass filter to the signal. We'll learn more about filters a bit later, but basically the idea is that the original signal is made up of sounds at many different frequencies, and the lower frequencies pass through the system, but the higher frequencies are attenuated. This affects how it sounds as well as how it looks.

Exercise 3 Box Function

1. Create a new function called `unitstep.m` in MATLAB. The function should take two parameters, a time vector that specifies the finite range of the signal and a time shift value.

   NOTE: Calling `unitstep([time],ts)` should be equivalent to `u(t + ts)`
2. Use the `unitstep` function to create a box-shaped time signal. Write a new function called `boxt.m` that creates a box with specified start and end times `t1` and `t2`. In other words, your function should take three inputs: scalars `t1` and `t2`, and a time vector `t`, and should output a vector of the same size as `t`, which contains the values of `u(t-t1)-u(t-t2)` evaluated at each point in `t`.

3. Create a script file called `boxtscript.m` that uses the function to create a box that starts at time `t = -1` and ends at time `t = 1`, where the signal lasts from time `t = -3` to `t = 3`. Generate three different versions of this box using three different time granularities, where the finest granularity has very sharp edges similar to the ideal box and the coarsest granularity has a step size of 0.5.

   NOTE: The different versions should all have the same time span; the difference in the plots should only be at the edges of the box because of artifacts in continuous plotting of a discrete-time signal.

4. Plot all three versions in one figure using subplot and save it as `boxtscript.tif`.
5. Time: If `u` is a vector of length `n` with time span `tu = t1:del:t2`, and `v` is a vector of length `m` with time span `tv = t3:del:t4`, and both have the same time step `del`, then the result of `conv(u,v)` will be a vector of length `n + m - 1` with a time span `tc = (t1+t3):del:(t2+t4)`.
6. Using the box function that you wrote in step 2 with a sufficiently fine grained step size (for example, `del = 0.01`), create box signals from (0,4) and (-1,1), with time span of (-5,10). Find and plot the result of the convolution of the two boxes and save it as `convplot.tif`. Use the above discussion of Time to create the appropriate time vector in your plot. Verify that the timing of signal rising and falling matches what you expect in theory.
7. Amplitude: In the resulting plot from the previous step, you should notice that the amplitude is much higher than the max of 2 that you would expect from analytically computing the convolution. This is because it is thinking that the length of the box is `n` rather `n del`, which impacts the area computation in convolution. To get the correct height, you need to scale by `del`. Scale and plot the resulting function, and verify that the height is now 2. Save the figure as `scaled.tif`
8. Triangle: Design the impulse response for a system `h` and a system input `x` such that you get a perfectly symmetric triangle of length 100 as the system output `y`. Use subplot to plot `x`, `h`, and `y`, and save the plot as `tri.tif`.
9. Be able to demonstrate your code, show your plots, and play sounds.