Lab 04 Supplement: Filters In Action

EE 20 Spring 2014
University of California, Berkeley

1 Introduction

In this supplement to Lab 4, we will apply the concepts of filters, frequency response, and impulse response to denoising a corrupted audio signal.

2 A More General Comb Filter

In Lab 4 you saw the frequency response of a "comb filter",

$$H(\omega) = \frac{1}{2}(1 - e^{-i\omega N}).$$

In general, we say an LTI system H is a comb filter if it can be represented by an LCCDE of the form

$$y[n] = x[n] + \alpha x[n - N].$$

By plugging in $x[n] = e^{i\omega n}$, we see that

$$\begin{aligned} H(\omega)e^{i\omega n} &= e^{i\omega n} + \alpha e^{i\omega(n-N)} \\ \Rightarrow H(\omega) &= 1 + \alpha e^{-i\omega N}. \end{aligned}$$

Thus, in Lab 04, we had a comb filter with $\alpha = -1$ which was scaled by $\frac{1}{2}$. Note that the magnitude of the frequency response varies with the choice of N and α. For our purposes, let's restrain α to have $|\alpha| \leq 1$. Choose a few values of N (say, $\{1, 2, 3, 4\}$) and plot $|H(\omega)|$ for positive and negative α (say, $\{-0.5, 0.5\}$). What can you say about the locations and magnitude of the local maxima ("peaks") and minima ("notches")? How are they dependent on N and α (try varying α from 0 to ± 1 and observe how the peaks/notches change) [1]?

3 The Impulse Response & Convolution

Recall from lecture that the response $y[n]$ of an LTI system H to an input signal $x[n]$ is the *convolution* of the input signal with the impulse response $h[n]$. In other words,

[1] Hint: `http://en.wikipedia.org/wiki/Comb_filter#Frequency_response`

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

As an exercise, verify that convolution is indeed a commutative operation (hint: try a change of variables). Above is the mathematical definition of the convolution operation, but we can easily derive it from what we know about LTI systems. First of all, recall that the impulse response is, by definition, the response of the system to the unit impulse (the Kronecker delta function):

$$\delta[n] \rightarrow \boxed{H} \rightarrow h[n]$$

Since the system is time invariant, shifting the input by any integer k yields the same shift in the output:

$$\delta[n-k] \rightarrow \boxed{H} \rightarrow h[n-k]$$

Since the systems is linear, scaling the input by any constant $x[k]$ also scales the output by $x[k]$:

$$x[k]\delta[n-k] \rightarrow \boxed{H} \rightarrow x[k]h[n-k]$$

Linearity also gives us *superposition*, which means if we send a sum of signals as input to the system, the resulting output is the same as sending the signals through the system individually and summing up their respective outputs. In other words:

$$\sum_{k=-\infty}^{\infty} x[k]\delta[n-k] \rightarrow \boxed{H} \rightarrow \sum_{k=-\infty}^{\infty} x[k]h[n-k]$$

Now, note that the sum on the left is simply some arbitrary input signal $x[n]$ which has been written as a linear combination of Kronecker deltas. With this and the definition of convolution in mind, we have:

$$x[n] \rightarrow \boxed{H} \rightarrow y[n] = (x \star h)[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$$

We can now see why $x[n] = e^{i\omega n} \Rightarrow y[n] = H(\omega)e^{i\omega n}$. Plugging in, we have

$$\begin{aligned}
x[n] = e^{i\omega n} \rightarrow \boxed{H} \rightarrow y[n] &= \sum_{k=-\infty}^{\infty} x[k]h[n-k] \\
&= \sum_{k=-\infty}^{\infty} h[k]x[n-k] \\
&= \sum_{k=-\infty}^{\infty} h[k]e^{i\omega(n-k)} \\
&= \sum_{k=-\infty}^{\infty} h[k]e^{i\omega n}e^{-i\omega k} \\
&= e^{i\omega n} \sum_{k=-\infty}^{\infty} h[k]e^{-i\omega k} \\
&= e^{i\omega n} H(\omega),
\end{aligned}$$

where we have defined $H(\omega) = \sum_{k=-\infty}^{\infty} h[k]e^{-i\omega k}$. This expression will show up later in our study of the Fourier Transform.

In previous labs, we used MATLAB's `filter` function to determine a system's response to some input. Now we have another method: convolution! Instead of finding the LCCDE coefficients `A` and `B` and computing `y = filter(B,A,x)`, we can find the impulse response `h` and compute `y = conv(x,h)`.

Before we move on, let's return to the comb filter. What is its impulse response (it will also be a function of α and N)?

4 Noise

Let's consider a setting similar to Problem 4 on the midterm. Consider a signal $x[n]$ which represents a song you'd like to send to your friend. Suppose you have some wireless communication system which can send and receive songs between laptops, and you use this system to transmit $x[n]$ to your friend, who receives $y[n]$ (ideally, $y[n] = x[n]$). Unfortunately, when your friend receives the song, they find it has been corrupted by noise! In this section, we will use LTI filters to combat *additive noise*, that is, noise which can be modeled as an unwanted signal that is added to the desired signal in transmission. In other words, given some noise signal $w[n]$, we have

$$y[n] = x[n] + w[n].$$

In this section, we will be using the materials found at `https://github.com/nosaesa/ee20`. You will be provided with several corrupted audio files (the arrays `y1.mat` and `y2.mat`), and we will explore several denoising strategies. You'll need MATLAB and a pair of headphones! Open the script `denoising.m` and fill in your code as necessary.

4.1 A Single Tone

Let $w[n] = \cos(\frac{2\pi}{3}n)$. The resulting received signal is stored in the MATLAB array `y1.mat`. Give it a listen – can you hear the song? Since the unwanted noise is simply one sinusoid, we ideally want to use a filter whose frequency response has magnitude 1 everywhere except at $\omega = \frac{2\pi}{3}$, where it has magnitude zero. This preserves the desired signal (the song), but removes the noise. However, such a filter is not easy to implement at this point in the course, so we will use a comb filter. Choose α and N such that the noise is "notched out" and the remainder of the signal is preserved as best as possible. It may be helpful to note that most of the frequencies of interest in the song are "low". Once you've designed your filter, construct the impulse response in MATLAB as an array `h` (it should be very "sparse" (few nonzero values), and again, it is a function of the same α and N present in the frequency response!). Keep in mind that arrays in MATLAB are indexed from 1, not 0! Now filter the corrupted by computing `out1 = conv(y1,h)` and listen. If you designed your filter correctly, you should be able to hear the song without that pesky noise!

4.2 Is It Really That Easy?

There are a few things that should bother you about the previous example. In the real world, how would we know exactly what frequency is present in the noise? And how are we supposed to know that the "frequencies of interest" in the song are "low"? These are questions that you will be able to answer very soon! In EE 20, you will learn about one of the most powerful tools in signal processing: the Fourier Transform. Hopefully these examples will motivate why we would want such a tool that allows us to look at the frequency content of signals.

The other glaring issue is that of the noise itself! Do you think noise in the real world presents itself as a single frequency? This is certainly the case with 60 Hz noise [2], but in general noise is *random*. In wireless communications, your signal is transmitted as an electromagnetic wave which propagates through free space. Noise is then the result of your signal bouncing off objects or passing through buildings or interfering with other signals. This noise is generally modeled as additive noise, where $w[n]$ is *completely random* [3]. Don't worry too much about this for now – all you need to know is that "real noise" doesn't just occur at a single frequency. It "hates" all frequencies equally, and thus you can think of this random noise as containing many, many frequencies. This definitely presents a problem! How do we filter out noise that occurs at every frequency, including the ones we care about? Our strategy of zeroing out the

[2]See `http://en.wikipedia.org/wiki/Mains _hum`

[3]http://en.wikipedia.org/wiki/Additive_white_Gaussian_noise

noise frequencies will definitely not work, since the noise is present in all of the frequencies in our song!

4.3 Random Noise

Now let $w[n]$ be random noise [4]. The resulting received signal is stored in the MATLAB array `y2.mat`. Give it a listen – how does it differ from the single-tone noise? Does it sound familiar? Since this noise hates all frequencies equally, we have a problem. The "notch" comb filter used previously will definitely not work (try it!). In fact, it is not possible to remove all the noise given our very limited bag of tricks at this point in the course! However, since we know that most frequencies in the song are low, we can try low-pass filtering the signal. We should try this because our signal has no important high frequencies, but the noise adds unwanted high frequency content. Thus, if we remove the high frequencies, we will get rid of some of the noise. To accomplish this, implement the filter given by System 4 in Lab 4 via MATLAB's `filter` function (since its impulse response is not pretty). How does it sound? Any better?

5 Closing Thoughts

Hopefully this supplement has convinced you that LTI filters can be fun and useful, but there's a lot more to signal processing than reading off coefficients from an LCCDE. For example, did you notice that MATLAB's `conv` function takes *forever* to run?!?! If you've taken CS 170, you may recall that convolution runs in $O(n^2)$ time! It's a terribly slow operation, so in the real world, filters are not implemented this way. As you will learn later, the Fourier Transform allows us to perform filtering operations MUCH faster (via the FFT algorithm, which runs in $O(n\log(n))$, for those who've taken CS 170)! Try it out for yourself – MATLAB's `fftfilt` function implements filtering `y = fftfilt(x,h)` via the Fourier Transform method that you'll learn later. Is it faster?

If you're interested in problems like this (communicating information reliably and efficiently in the presence of noise), consider taking courses like EE 121 and 123. EE 121 (Coding for Digital Communication & Beyond) serves as a brief introduction to the field of "Coding Theory" (the theory of error correcting codes, for those who've taken CS 70). In this course, you will learn about ways of "shielding" your signal from noise. Notice that in our very basic wireless communication system described earlier, the signal $x[n]$ was basically transmitted "naked", and was therefore very susceptible to noise. Coding Theory lets you encode signals by adding redundancy in such a way that they can be transmitted through noise with very little error (sometimes none at all!). EE 123 (Digital Signal Processing) deals with more advanced filter design and applications of

[4] If you're familiar with probability theory, "random noise" in this case means each sample of $w[n]$ is an i.i.d. Gaussian random variable, in other words, $\forall n$, $w[n] \sim \mathcal{N}(0,1)$. If you haven't seen probability before, don't worry about it.

signal processing techniques in communications, compression, and many other cool areas of study.